MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CCD MATRIX PROCESSOR

Honeywell Systems and Research Center

P. C. T. Roberts
J. D. Joseph
J. A. Heschette
R. Fitch
R. R. Hanzel
D. B. Yanke

APPROVED: *Paul W. Pellegrini*

PAUL W. PELLEGRINI
Project Engineer

APPROVED: *Harold Roth*

HAROLD ROTH
Director, Solid State Sciences Division

FOR THE COMMANDER: *John P. Huss*

JOHN P. HUSS
Acting Chief, Plans Office

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-82-294	2. GOVT ACCESSION NO. AD-A128 643	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) CCD MATRIX PROCESSOR		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 25 Aug 80 - 1 Mar 82
		6. PERFORMING ORG. REPORT NUMBER 82SRC59
7. AUTHOR(s) P. C. T. Roberts R. Fitch J. D. Joseph B. R. Hanzal J. A. Hoschette D. B. Yanke		8. CONTRACT OR GRANT NUMBER(s) F19628-80-C-0154
9. PERFORMING ORGANIZATION NAME AND ADDRESS Honeywell Systems and Research Center 2600 Ridgway Parkway Minneapolis MN 55413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2305J135
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ESED) Hanscom AFB MA 01731		12. REPORT DATE April 1983
		13. NUMBER OF PAGES 234
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Paul W. Pellegrini (ESED)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Parallel image processor
CCD matrix array
Image processing algorithms
Threshold, edge enhancement

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
A new CCD Matrix Array has been fabricated including a Microprocessor
Host-CPU and Controller Exerciser Electronics Unit. Each unit cell of
the array performs analog arithmetic and logic operations on input data
matrix elements in the Single Instruction Multiple Data architecture.
Ultra-high signal and image processing throughput is demonstrated.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

CONTENTS

CONTENTS (continued)

CONTENTS (concluded)

Accession For

NTIS GRA&I	X
DTIC TAB	
Unannounced	☐
Justification	

By
Distribution/
Availability Codes
Avail and/or
Dist Special

A

v

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (continued)

LIST OF ILLUSTRATIONS (continued)

LIST OF ILLUSTRATIONS (concluded)

LIST OF TABLES

SECTION I

INTRODUCTION

OVERVIEW OF THE CONTRACT

The objectives of the work described in this final report on Contract
F19628-80-C-0154 are straightforward:

o   Analyze by simulation and measurement the performance capabilities
    of an ultra-high throughput CCD matrix processor employing parallel
    analog processor cells

o   Design a second-generation CCD matrix processor cell capable of
    executing a complete set of image processing algorithms

o   Construct a flexible, programmable Exerciser unit capable of
    operating the CCD matrix processor chip in all modes of
    functionality

o   Deliver the 15 best-effort CCD matrix processor chips and the
    Exerciser unit, complete with basic software

All of the objectives have been met. The contract was extended as per
modification P00003 requested by RADC to provide extra time and funding for
the greatly improved capabilities desired for the Exerciser unit. The unit
now contains expansion capabilities to meet future matrix processor
requirements without significant hardware modifications. The Exerciser is a
microprocessor-controlled unit capable of exercising not only parallel image
processing (PIP) algorithms, but also a broad range of general signal
processing functions which are expressible as single instruction, multiple
data (SIMD) sequences of primitive operators.

1

Early characterization tasks revealed that the original CCD cell and array designs were deficient in some key respects. After consultation with RADC (Contract Monitor, Mr. Paul Pellegrini), it was decided that suitable redesign of the mask set was required and would greatly improve the performance of the final CCDs to be delivered to RADC.

The changes involved provision of charge sink diodes at the originally unterminated boundaries of the matrix and also a redesign of the structure of the floating gate amplifier (FGA) and gain-control circuitry (Multiplier). Funding for the new mask set was provided by internal Honeywell sources.

No architectural changes were made to the cell design at that time since a minimum-risk improvement was desired: functional CCDs from the original mask set had been obtained before the start of the contract. However, low yield of chips has continued to hamper characterization of some aspects of the cell performance. A remaining design error discovered after the first wafer processing run was immediately corrected. However, the second and third wafer runs (12 wafers each) proved to be substandard due to high charge transfer inefficiency and also to random short-circuit metal tracks. Additional wafer process runs have been performed on Honeywell funding. The fourth run has yielded excellent transfer inefficiency data and verified that the design modifications are effective in completely removing the termination blemishes. We have now obtained some functional devices, but because of randomly open circuit control gates in the cell and threshold nonuniformity, there are no perfect chips available. There are no known design errors and failure analysis in progress to identify the causes of the fabrication problems. Other types of CCDs fabricated on the same facility have yielded satisfactorily good area arrays, and there is no obvious explanation for the random defects. Wafer runs and probing will continue on Honeywell funding until a satisfactory result is obtained, since

success of this device is of primary importance to second-generation chip viability. Partially functional chips have been diced and packaged ready for delivery.

As a major task in the contract, a detailed redesign of the cell architecture and array peripherals was performed. The basis for this redesign was the determined need to provide a minimum of six storage sites in the cell interior as well as to provide a coarse scaling function (x1/5) in addition to the FGA. Fully parallel I/O was found to be a primary requirement for even small array numbers; further, a fully programmable random access readout was highly significant for overall system throughput.

A major breakthrough was achieved by simply restructuring the storage locations into a "racetrack" or ring three-phase CCD. This configuration provides six storage locations while using the inherently high packing density of a conventional CCD shift register and requiring only five distinct clock phases in place of 12 with the old architecture. With this dramatic packing density improvement came the necessary real estate to provide not only the desired charge splitter (coarse-scaler) but also a *random access* analog output FET readable from the periphery of the array. All the earlier limitations on throughput have now been overcome, and an elegantly simple solution to the fully parallel sensor-to-host CPU has been attained in concept. It is even feasible to read data into the array from a line-parallel serial input concurrently with fully parallel pixel data directly into the cell. This opens the way to real-time offset compensation without the need to store coefficients in the cell; an auxiliary storage device (either analog MNOS or digital ROM) can be read once every frame. Depending on the severity of offset it may be sufficient to use a Shutterless Compensation Algorithm (Reference 1). No detailed studies were attempted to determine feasibility of this approach because of limitations on the scope of the contract.

Software complexity was dealt with in two ways. First, a custom-designed
8086 microprocessor-based Exerciser unit, benefiting from coordinated
development efforts on other associated programs, was built for delivery to
RADC. This unit is in practice a complete subsystem which not only stores
the Controller clock waveform primitives in RAM--2K x 48 in six chips loaded
from EPROM--but can also serve as the host to interface to any other system
through the RS232C port. A built-in expansion provides for MATROX RGB-256
video frame memory, allowing 16 grey levels NTSC (or PAL) drive to a
high-resolution (256 x 256 x 4) monitor driven from a MATROX frame
digitizer. The complete unit therefore allows algorithm development in real
time using any standard video source (camera, VTR, etc. on RS170 format) and
also predetermined test patterns from EPROM/RAM. To modify clock waveforms
requires hook-up to a Microprocessor Development System (e.g., iMDS 235),
but to use the Honeywell-supplied primitives a simple keyboard input is all
that is required. A Texas Instruments (TI) "Silent 700" is supplied with
the unit.

The second aspect of software management concerns the CCD architecture. A
"standard" was adopted for the chip types #2214 and #3022 (first generation)
requiring simple concatenation of commands, each of which has a Default or
Initial condition as the first and last clock states of the sequence. The
second-generation matrix processor will have a substantially greater degree
of concurrency as a designed feature, and so a new "standard" is needed.
The Exerciser unit will satisfy both generations, since concurrency is
invisible to the user insofar as a new set of primitives is all that is
required to be defined. The hierarchy for upper levels remains unrestricted.

Follow-on development is discussed in Section III of this report.

4

SYNOPSIS OF THE REPORT

The report is organized in order of the seven tasks which were executed. In many instances, these tasks were executed simultaneously. The data presented in each of the task details is not chronological; rather, an attempt is made to order the information to make logical reading possible.

Much has, by necessity, been omitted from the implications of such a powerful device. It is hoped, however, that the reader will be left with the impression of a highly multi-disciplinary development, not merely a single chip design.

Section III gives some background of Honeywell's demonstrated capabilities in indium bump direct sensor interface technology. An indication is given of the growth potential for a family of lesser or greater capabilities for smart sensor/seeker applications in the near term, up to five years from now.

As an aid to brevity, the mnemonic "PIP" will be used in place of "CCD matrix processor"; however, this should not be construed as an implied limitation of the device to parallel image processing.

LIST OF ITEMS DELIVERED TO RADC

Reporting

o   5 quarterly R&D status reports

Hardware

o   15 packaged CCD chips: 3022-3-8 #'s 27, 28,
                              33, 36, 44, 45, 46, 50, 52,
                              55, 58, 70, 72, 82, 91

o   1 custom-assembled Exerciser unit containing basic primitives and
    Intel SBC 86/12 operating software in EPROM

o   1 Texas Instruments Silent 700 Terminal

Documentation

o   1 complete set of engineering drawings specifying hardware units

o   User manual for Exerciser unit

o   Complete bit-pattern specification of CCD primitives covering basic
    operation of the matrix processor and at least one edge extractor
    (Roberts Cross) algorithm

o   Computer program listings for FGA transfer function computation
    with constant capacitance load

o   Final report

6

SECTION II

DETAIL OF TASKS PERFORMED

TASK 1--CHIP EXERCISER DESIGN AND FABRICATION

In this section the overall description of the custom designed circuits for operating the #3022 PIP is presented.

The Exerciser unit comprises one custom-fabricated metal cabinet mounted on top of a standard Intel 660 chassis. The custom-fabricated CCD clock driver and signal conditioning circuits are mounted on a separate PC board mounted loosely on top of the entire unit. This configuration was chosen as the most useful for access to all the individual clock voltage control potentiometers and was also the least costly assembly. Wire-wrap connections are used extensively throughout the unit to allow repairs and any future alterations to be easily implemented.

A photograph of the completed unit together with associated cameras, VTR, and monitor is shown in Figure 1 with the Intel MDS 235 in the background.

Circuit diagrams showing parts and interconnections are detailed in the engineering drawings delivered to RADC.

I/O Board Addressing

In the exerciser, all boards are located on I/O ports of the SBC 86/12A board and are addressed by the eight lower bits of the multibus address lines; the upper bits are arbitrary. Excluding any I/O addressed functions on the 86/12A board, there are four boards on the multibus: the RGB-256
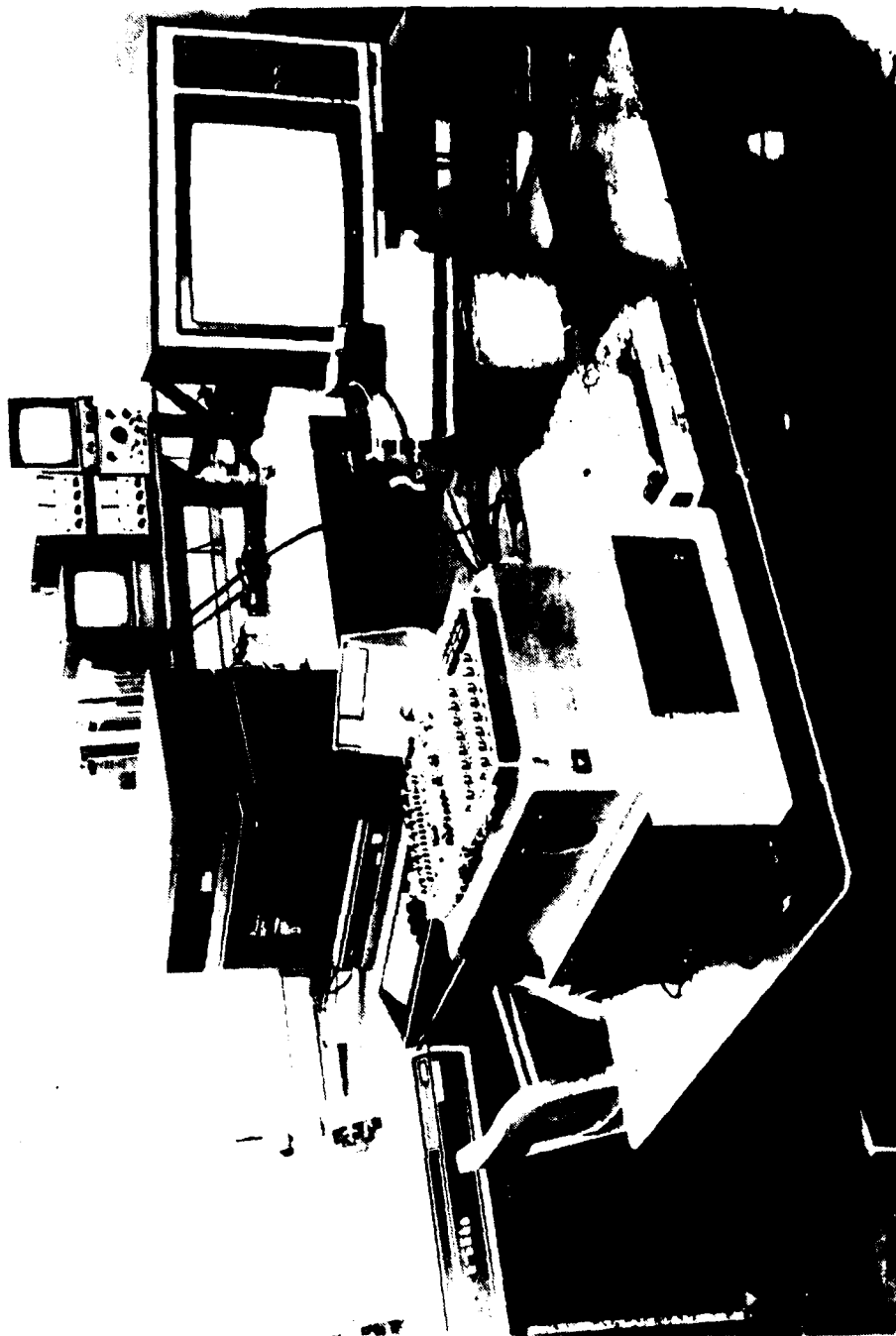
Figure 1. Completed Unit Together with Associated Cameras, VTR, and Monitor

8

frame buffer, the FG01/8 digitizer, the timing generator board, and the I/O buffer board. The boards are addressed by the 5-bit A_{7-3}. The lower three bits A_{2-0} are used for functions on the board. Up to 32 I/O boards can be addressed. The addresses of these four boards are listed in Table 1.

Timing Generator Board

The PIP clock generator is built around an AMI2910 microprogrammable sequencer capable of 10 MHz operation and having a versatile programming mix of branching, looping, and subroutine instructions. This design will handle almost all future timing needs.

A block diagram of the timing generator board is shown in Figure 2. The key elements are the 2K x 48, 90 nsec random access memory and a 2910 sequencer. The timing generator is configured as a classical double-pipelined microprogrammable sequencer. The memory holds microinstructions of two types: clock generation and sequencer branch control. Under the clock generation mode of operation, the memory can simultaneously generate states for up to 32 PIP clocks. Under the sequencer branch control mode of operation, the memory clocks are used for looping, branching, and subroutine calls.

Timing Generator Multibus Commands

Following is a brief summary of the multibus commands that can be issued to the timing generator board.

The format is:

I/O Address (Hex)	R/W	Command
18	W	Load Random Address LSB (Byte)

TABLE 1. MULTIBUS I/O PORT ADDRESSES

RGB-256/4 Frame Buffer Board

08	W	Data intensity
	R	Data
09	W	Data
	R	Data
0A	W	Scroll--line to be displayed top screen
	R	Flags
0B	W	Scroll
	R	Flags
0C	W	X location
	R	Erase
0D	W	X location ⎫ Not used
	R	Erase ⎭
0E	W	Y location
	R	--
0F	W	Y location
	R	--

FG-01/8 Digitizer

10	W	Data word should be set to 00

Timing Generator Board

18	R	NOP
	W	Load random memory address LSB

Timing Generator Board (concluded)

19	R	NOP
	W	Load random memory address MSB
1A	R	Read memory data byte
	W	Write memory data byte
1B	R	NOP
	W	Write command register
1C	R	Read memory address (LSB)
	W	Write macro address (MSB)
1D	R	Read memory address (MSB)
	W	Write macro address (MSB)
1E	R	NOP
	W	NOP
1F	R	NOP
	W	NOP

I/O Buffer Board

20	R	NOP
	W	Load memory address (LSB)
21	R	NOP
	W	Load memory address (MSB)
22	R	Read memory data byte
	W	Write memory data byte
23	R	NOP
	W	Write command register
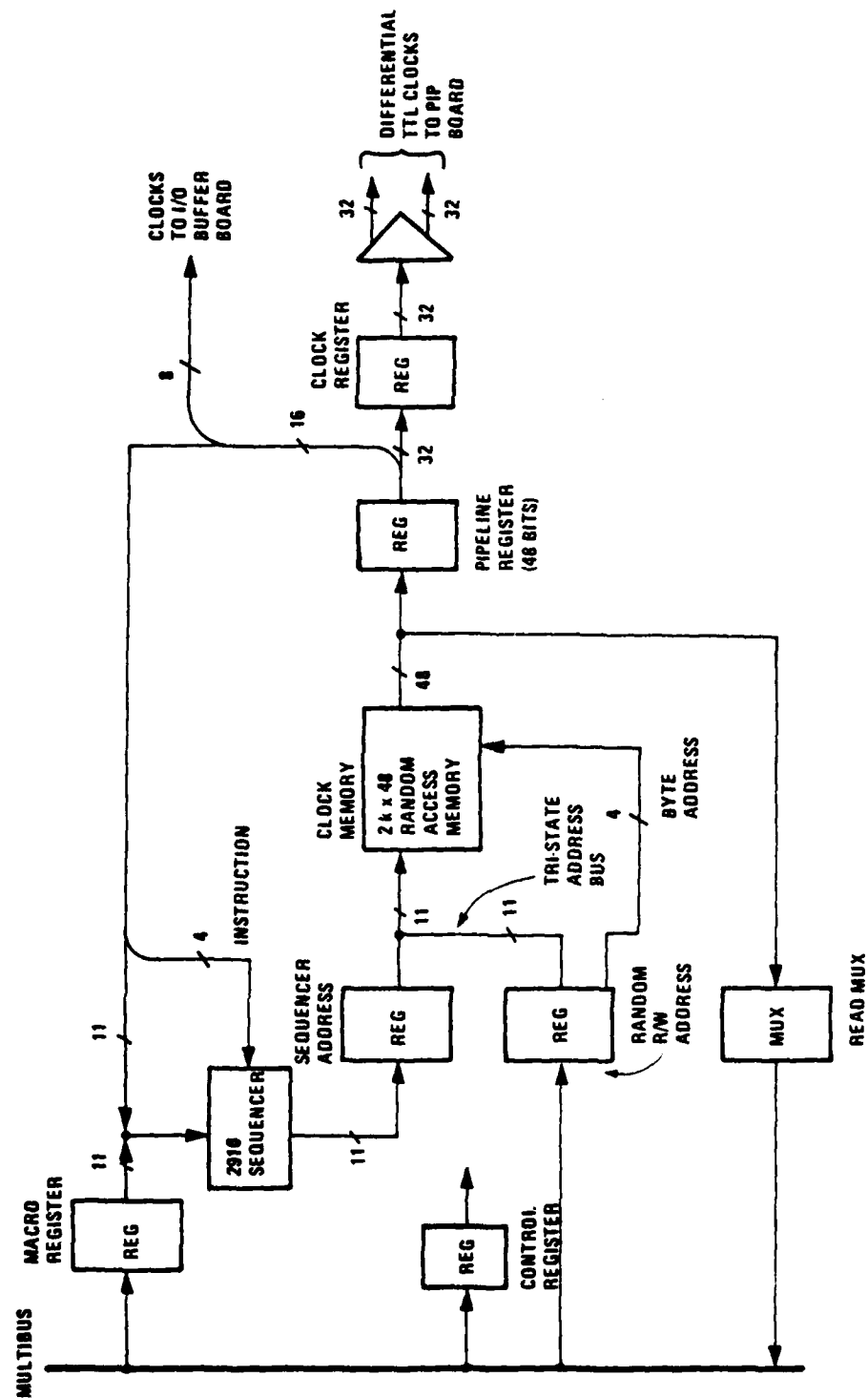
Figure 2. Block Diagram of Timing Generator Board

11

The command below loads the least significant byte of the memory address via the multibus data bus. The format is:

BIT	7 6 5 4	3 2 1 0
	Lower 4 bits of 11 Bit Word Address	Memory Chip Select

The address is stored in a register after loading, and is not modified by any other command or operation.

The command below loads the most significant byte of the random read/write memory address via the multibus:

I/O Address	R/W	Command
19	W	Load Random Address MSB (Byte)

The format of the data byte is:

Multibus Bit
(Bits 8—15 Unused)

7	6 5 4 3 2 1 0
0	7 Most Significant Bits of the 11-Bit Word Address

The address is stored in a register that is only modified by this command:

I/O Address	R/W	Command
1A	W	Write to Timing Generator Memory (LSB)

This command writes the least significant byte of data on the multibus data lines to the timing generator memory location defined by 18 and 19. The data byte format is:

```
                                           7  6  5  4  3  2  1  0
              Multibus                 ┌──────────────────────────┐
                Bit                    │   Byte of Data Written to │
         (Bits 8—15 Unused)           │  Timing Generator Memory  │
                                       └──────────────────────────┘
```
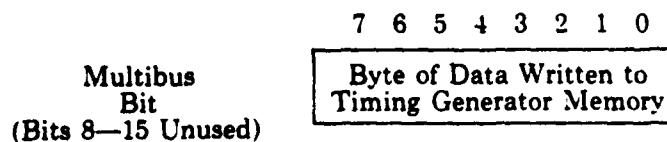
The memory byte address is always that specified by the random address of 18
and 19. The word address of the memory depends on the state of DIAG in the
command register. If DIAG = 1, the random address is used. If DIAG = 0, the
sequencer address is used.

The command below reads a data byte from the timing generator memory into the
least significant byte of the multibus data bus (the most significant byte
will have arbitrary contents):

I/O Address	R/W	Command
1A	R	Read Timing Generator Memory/Data Byte

The data byte format is:

```
                                           7  6  5  4  3  2  1  0
              Multibus                 ┌──────────────────────────┐
                Bit                    │   Byte of Data from Timing│
         (Bits 8—15 Unused)           │     Generator Memory      │
                                       └──────────────────────────┘
```
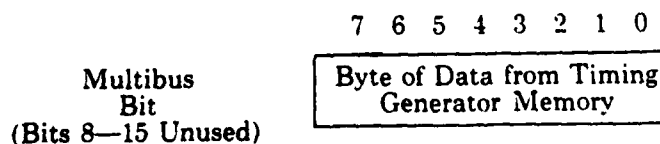
The memory address is the same as for the write instruction of this command,
except when DIAG = 0, the last instruction executed by the sequencer can be
read from memory.

The command below writes the least significant byte of the multibus data bus
into the command register on the timing generator board:

13

I/O Address	R/W	Command
1B	W	Write Command Register Byte

The format of the command byte is:

	7 6 5	4	3	2	1	0
Multibus Bit (Bits 8—15 Unused)	Clock Speed 2-ϕ	D I A G	E N D V R	S S	R U N	R E S E T

where

RESET = 0   resets all board functions (set to 0 on power-on)

= 1   disables reset function

RUN = 1   sets the sequencer to run mode

0   halts the sequencer (set to 0 on power-on)

SS = 1   single-steps the sequencer one instruction (RUN = 0)

= 0   no single step (set to 0 on power-on)
SS   must be toggled to step multiple instructions

ENDVR = (not used at present)

DIAG = 1   selects the random address as the timing generator memory address (RUN must equal zero)

= 0   selects address from sequencer as timing generator address (set to 0 on power on)

CLKRT = 000 clock sequencer runs at 8 MHz

= 001 4.16 MHz

= 010 1.78 MHz

= 011 890 kHz

= 100 445 kHz

- 101 223 kHz

- 110 111 kHz

- 111 55 kHz

The reset true condition will cause a jump-to-zero instruction to be put on the 2910 instruction lines.

The command below loads the least significant byte of the multibus data bus into the least significant byte of the 11-bit macro address register:

I/O Address	R/W	Command
1C	W	Load MACRO Address LSB (Byte)

The MACRO address is a word-address only and has no byte address. The format of the data byte is:

```
           7  6  5  4  3  2  1  0
Multibus                              Bit
Bit       ┌──────────────────────────┐
Bits 8-15 │ Least Significant Byte of the 11-bit │
Unused)   │       MACRO  Address       │
          └──────────────────────────┘
```
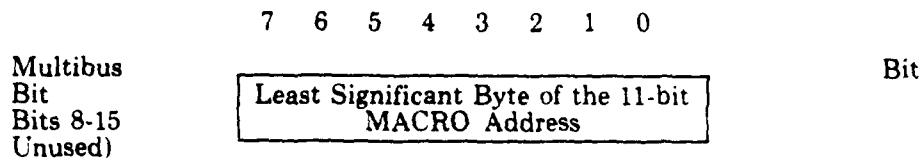
This address can be continuously written to the board. Wait states (until the sequencer uses the address) are automatically generated and are transparent to the user. The macro address LSB can only be changed by this command. If the timing generator is in a single-step mode, outputting this command will generate a single-step cycle.

The command below reads onto the multibus data bus the least significant word current address on the timing generator for memory:

15

I/O Address	R/W	Command
1C	R	Read Least Significant Byte of the Timing Generator Memory Address

The source of the address is determined by the state of DIAG in the command register. If DIAG = 1, then the address is the word-address portion of the random address. If DIAG = 0, then the address is from the sequencer (not to be confused with the macro address). The format of the data to the multibus is given below:

```
              7   6   5   4   3   2   1   0              Bit
Multibus      ┌──────────────────────────────┐
Bit           │    Least Significant Byte of │
(Bits 8-15    │    Current Timing Generation  │
Arbitrary     │       Memory Address          │
              └──────────────────────────────┘
```
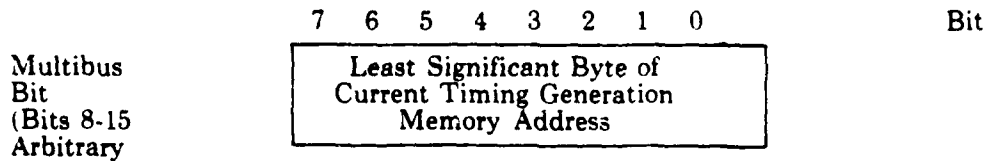
The command below loads the least significant byte of the multibus data bus into the most significant bits of the 11-bit macro address:

I/O Address	R/W	Command
1D	W	Load MACRO Address MSB

The format of the data byte is:

```
              7   6   5   4   3   2   1   0              Bit
Multibus      ┌────────────────┬───┬──────────────┐
Bit           │    UNUSED      │ 0 │ 3 Most Significant │
(Bits 8-15    │                │   │ MACRO Address │
Unused)       │                │   │    Bits       │
              └────────────────┴───┴──────────────┘
                                  Reserved
                                  for
                                  Expansion
```

16
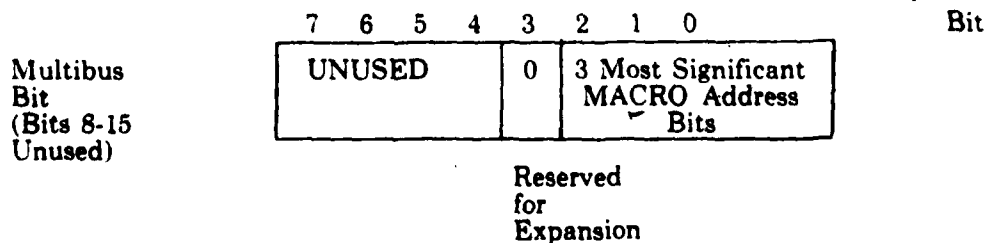
This address can be continuously written to the board. Wait states are automatically generated. The macro address MSB can only be changed by this command.

The command below reads onto the multibus data bus the most significant bits of the current timing generator memory word address:

I/O Address	R/W	Command
1D	R	Read the Most Significant Byte of the Timing Generator Memory Address

The source of the address is determined by DIAG, as explained in the LSB address read command (100). The data format is:

```
                      7   6   5   4   3   2   1   0              Bit
                    ┌───────────────┬───┬───────────┐
Multibus            │   UNUSED      │ 0 │  3 Most   │
Data                │   (Zero)      │   │ Significant│
(Bits 8-15          │               │   │ Address Bits│
Arbitrary           └───────────────┴───┴───────────┘
                                          Reserved
```

Timing Generator 2K x 48 Bit Memory

There are two 48-bit formats for the microinstruction, as shown in Figures 3 and 4. Figure 3 is the clock generation format and Figure 4 is the branch address format. In clock generation, 32 bits of the instruction indicate clock states (C_1 to C_{32}). The specific PIP clock assignments in this instruction are given in Table 2. The 2910 instruction (I_{3-0}) must not be a specified address branch type (but it can be a stack pull or subroutine return). The instruction is usually a continue. The condition code (CC_{1-0}) is unused. The enable bit (EN), when 1, loads the clock states into the clock register (to go to external devices) on the next sequencer instruction. If

17
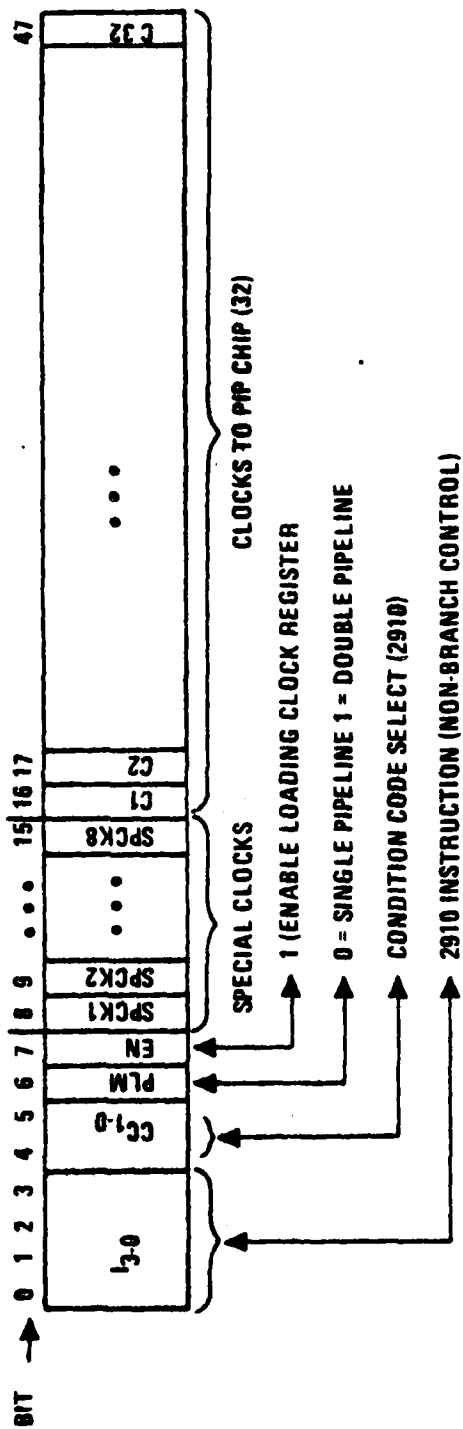
Figure 3. PIP Clock Generation Format
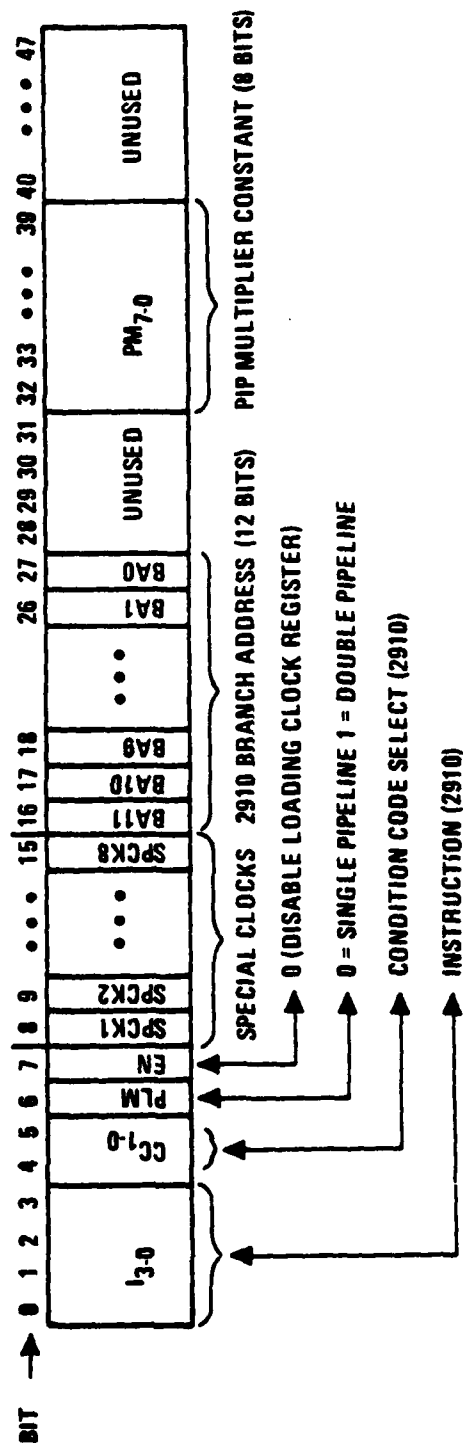
Figure 4. Branch Address Format

TABLE 2. CLOCK ASSIGNMENTS

3022 PIP Clock	PIP Mnemonic (for programming)	Default Binary State	Timing Generator Microinstruction Bit Position
0_1	PH1	H(high)	16
0_2	PH2	L(low)	17
0_3	PH3	L	18
0_4	PH4	L	19
0_A	PHA	L	20
0_B	PHB	L	21
0_C	PHC	L	22
0_D	PHD	L	23
0_X	PHX	H	24
0_{XS}	PHXS	H	25
ST1	ST1	H	26
ST2	ST2	H	27
ST3	ST3	H	28
V_S	VS	H	29
0_{RS}	PHRS	L	30
FGR	FGR	H	31
RSNI	RSNI	L	32
RSI	RSI	L	33
0_T	PHT	L	34
0_{XM}	PHXM	L	35
M_D	MD	H	36

EN is 0, the clock register will not be loaded (EN is almost always 1 for this instruction format). The pipeline mode bit (PLM), when 0, runs the 2910 as a single-pipeline machine. If PLM is 1, the 2910 runs as a double-pipeline machine. The only constraint to switching modes is that, when going from single- to double-pipeline, the first double-pipeline instruction executes twice.

The instruction format in Figure 4 is used for all branching or looping instructions. A 12-bit address (BA_{11-0}) is available, but only 11 bits are used at present (i.e., $BA_{11} = 0$). The instruction (I_{3-0}) can be any of the 16 used by the 2910. The only instruction that is not implemented in a standard way is the conditional jump vector (CJV), which will be used for macro address branching. The condition code on a CJV should always be zero. Also, a jump-to-zero instruction should always follow the CJV. The CJV should always be used at address zero in the timing generator in the single-pipeline mode.

The branch instruction has a PIP multiplier field (PM_{7-0}) in bits 32 to 39. This serves as a parameter when a CJSB 2910 instruction is used. The target subroutine can output the field to the PIP multiplier port using one of the special clocks.

In both microinstruction formats, eight special clocks $(SPCK_{1-8})$ are used for various operations, such as one I/O buffer sequential read/write operation for PIP I/O.

Double-Pipeline Branching

Double-pipeline mode is normally only used when instructions are to be executed consecutively. Branching is usually done in single-pipeline mode, but can be done in double-pipeline mode also. If a branch to address X is executed at location N, the instruction in location N + 1 is also executed due to the double pipeline. The instruction at N + 1 must not be a branch.

Timing Generator Memory Map

The organizing of the timing generator memory is to have groups of clock sequences broken into functions. The functions will be linked via subroutine calls in macro sequences. The macro sequences will be linked by address in the macro address register put out at a high speed by the SBC 86/12A (see Figures 2 and 3). A CJV instruction in location 0 (branch to zero executed on reset from the command register) polls the macro register until it is loaded by the 86/12A. When it is loaded, a branch to that address occurs. A sequence of subroutine calls (JSB) are executed to form a macro sequence. The macro sequence terminates with a jump-to-zero command. The subroutines are clock-state sequences, which generate the clock waveform patterns for a specific function. These sequences end with a return from subroutine. The clocks are output at high speed in the double-pipeline mode, while all branching is done in the single-pipeline mode.

Condition Codes (CC$_{1-0}$)

There are eight possible condition codes, but only two are currently implemented. They are described in Table 3.

I/O Buffer Board

The I/O buffer is shown in Figure 5. The board stores input/output data to/from the PIP and does digital/analog and analog/digital conversions. Buffer 1 is loaded via the multibus with data going to the PIP (up to 4K words of 8 bits each). Buffer 2 is loaded with data from the PIP and interrogated by the multibus. Both buffer memories can be read by/written to the multibus for diagnostic purposes.

21

TABLE 3. CONDITION CODES

CC_{1-0}	Description
00	True: the macro address register has been loaded by the multibus, but its contents have never been branched to (i.e., has not been "unloaded") False: macro address register is either "empty" or has been branched to ("unloaded")
01	True: always true (unconditional branch) False: never
10	(Not assigned)
11	(Not assigned)

As far as the multibus is concerned, the two *buffer memories appear as a single address 8K x 8 memory*, with the lower 4K x 8 as buffer 1 and the upper 4K x 8 as buffer 2.

I/O Buffer Board Multibus Commands

The following is a brief summary of the multibus commands that can be issued to the I/O buffer board:

I/O Address	R/W	Command
20 (HEX)	W	Load Memory Address (LSB)

This command loads the least significant byte of the 13-bit memory address via the multibus data bus.

22

Figure 5. I/O Buffer Board

23

The format of the data byte is:

```
           7   6   5   4   3   2   1   0
          ┌─────────────────────────────┐
Multibus  │                             │
Bit       │    8 LSB of Memory Address  │
(Bits 8-15 unused)                      │
          └─────────────────────────────┘
```
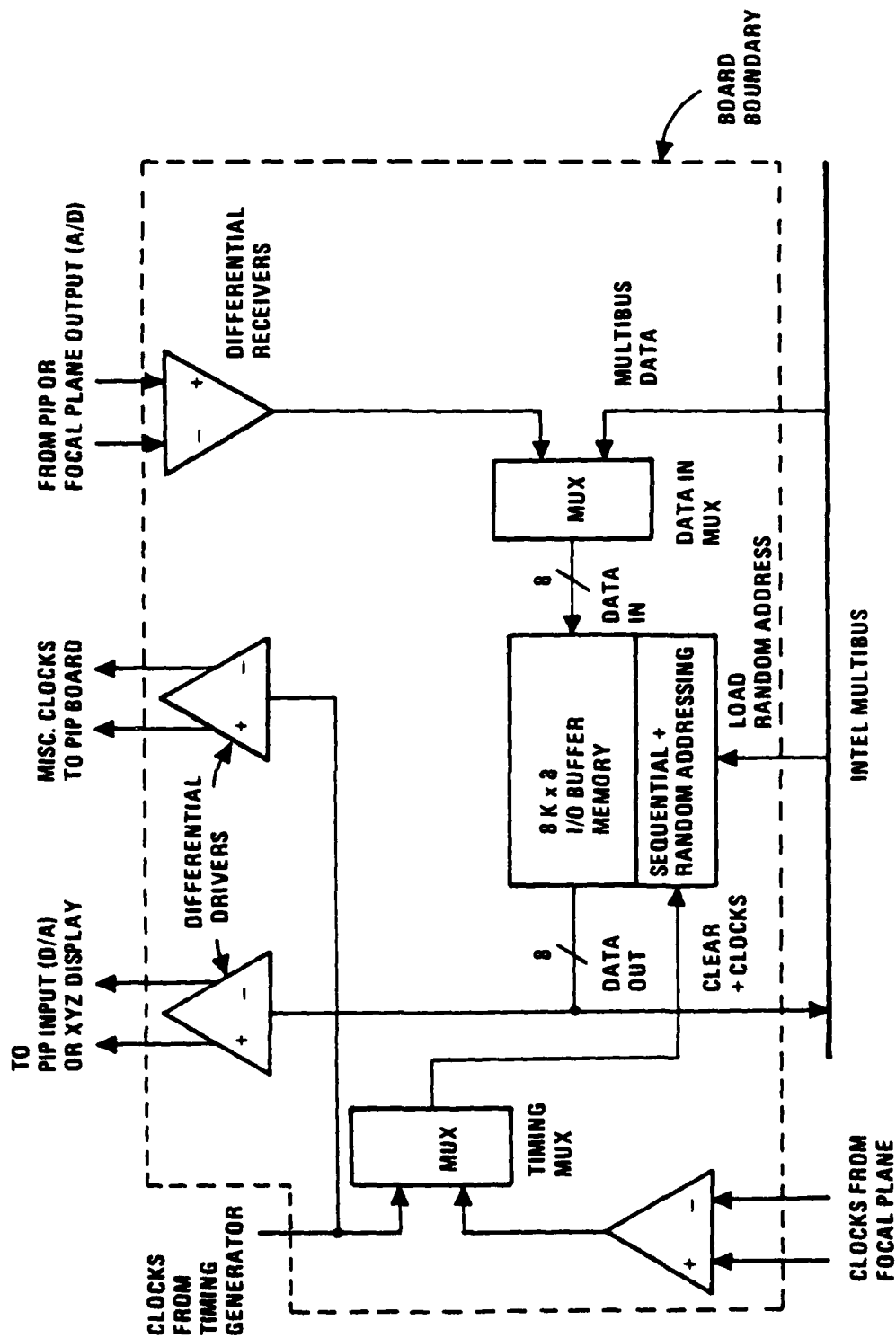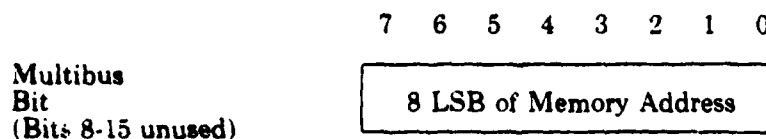
This address is loaded in a register that can be modified if external control is enabled in the command register. Otherwise it does not change, except by this command.

The command below loads the five most significant bits of the 13-bit memory address from the multibus data bus:

I/O Address	R/W	Command
21(HEX)	W	Load Memory Address MSB

The format of the data byte is:

```
            7   6   5   4   3   2   1   0
           ┌──────────┬──────────────────┐
Multibus   │   Not    │  5 MSB of 13-bit │
Bit        │   Used   │  Memory Address  │
Bits 8-15                                │
Unused)    └──────────┴──────────────────┘
```
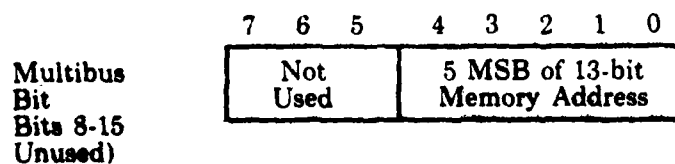
where BIT 4 = 0 selects buffer memory 1 and BIT 4 = 1 selects buffer memory 2.

The command below writes the least significant byte of the multibus data bus to the I/O buffer memory at the previously specified address:

24

I/O Address	R/W	Command
22 (HEX)	W	Write Memory Data Byte

The data byte format is:

```
                        7   6   5   4   3   2   1   0       Bit
          Multibus    ┌───────────────────────────────┐
          Bit         │      Data Byte Written         │
          (Bits 8-15  │     To I/O Buffer Memory       │
          Unused)     └───────────────────────────────┘
```
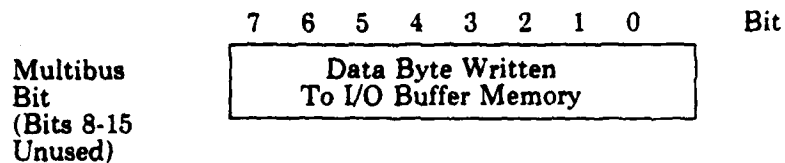
The command below reads onto the least significant byte of the multibus data byte the contents of the I/O buffer memory at the previously specified address:

I/O Address	R/W	Command
22 (HEX)	R	Read Memory Data Byte

The resulting data byte format is:

```
                      7   6   5   4   3   2   1   0       Bit
          Multibus  ┌───────────────────────────────┐
          Bit       │    Data Byte Read from I/O     │
          Byte      │        Buffer Memory           │
                    └───────────────────────────────┘
```
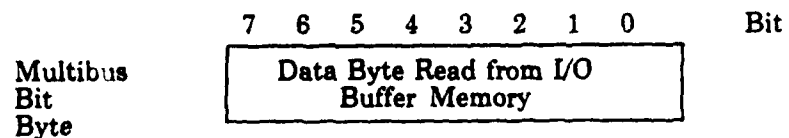
The command below writes the least significant byte of the multibus data bus into the I/O buffer command register:

I/O Address	R/W	Command
23 (HEX)	W	Write Command Register

The data format is:

```
                    7        6  5  4  3  2  1         0    Bit
                 +--------+----------------------+-------+
Multibus         |  Not   |    Not Specified     |   E   |
Bit              |  Used  |                      |   N   |
(Bits 8-15       |        |                      |   E   |
Unused)          |        |                      |   X   |
                 |        |                      |   C   |
                 +--------+----------------------+-------+
```

where

ENEXC = 0   external control disabled (All control signals to the I/O
            buffer board are disabled except those from the multibus)

      = 1   enable external control

            [External control signals to transfer PIP
            data (from timing generator) are enabled.
            Only the multibus command register command
            should be used. Other multibus commands can provide
            erroneous results.]

The external controls are generated as clocks on the timing generator board
and by the PIP chip.

TASK 2--INITIAL CCD CHARACTERIZATION

The initial device characterization work was performed using Honeywell's PIP chip #2214, which had been fabricated earlier in 1980 before the start of the contract. Only one run of wafer processing was made at that time, resulting in a very limited sample of devices. However, a thorough enough analysis of the 2214 chip was made that detailed changes were seen to be necessary.

The following subsection describes the array and cell design of the 2214 chip. Measurements from these chips are then described to indicate the need for the changes that were made. Finally, the changes to the array and cell are described.

Original PIP Array and Cell Design

A prototype cell was used to generate the whole array. Access to the array is through a fill-and-spill serial input at the top left, a four-phase serial I/O buried-channel shift register (with 90° corner gate) along the left and bottom edges of the array, and, finally, a floating diffusion sense amplifier with reset at bottom right.

A polysilicon clock-bus set of lines runs down the right side of the array to connect the repeated metal clock lines which run left-right through the array. These are visible in the photograph of the chip shown in Figure 6.

A copy of the Calma plot of the cell is shown in Figure 7, together with a photograph of the cell on the chip. The cell contains three separately controlled storage areas labeled ST1, ST2, and ST3 in Figure 7. ST2 and ST3 have one common additional gate, XS, which is used to control access to these

27
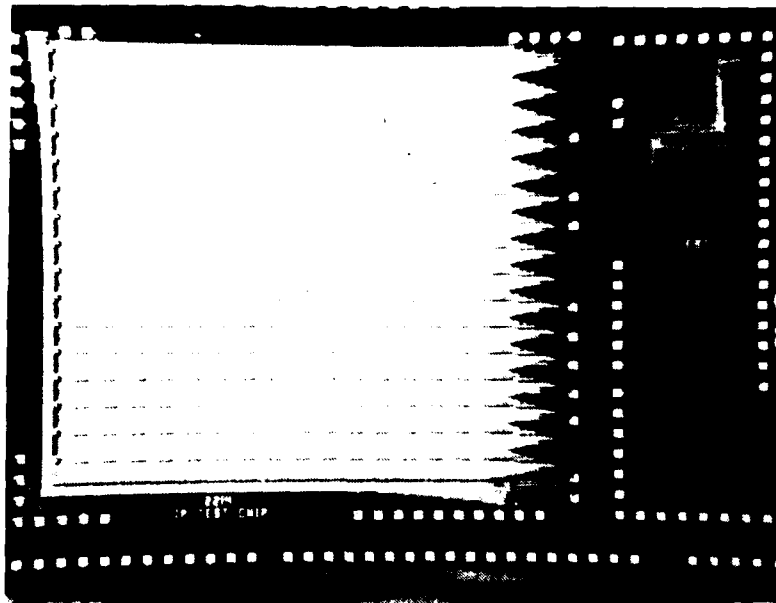
Figure 6. Photograph of Original #2214 CCD Chip

stores. Also contained in the cell is a surface-channel fill-and-spill input
section labeled Regenerator. The inputs to the regenerator are obtained from
two floating-gate detectors labeled Y and Z. Each floating gate has a reset
transis.or source control--RSI and RSNI--but the gates of the transistors are
tied together and controlled by FGR. The Z detector is loaded by the
multiplier capacitor whose control gate is labeled MG. The multiplier in this
design is equivalent to a surface-channel gated-diode structure in which the
diode is connected to the floating-gate detector. By biasing the MG gate
towards surface inversion, the total capacitive load on the detector is
increased. The sensitivity of the detector is correspondingly decreased.

28

Figure 7. Calma Plot of the 2214 Cell, Together with a Photograph
of the Actual Cell (Poly I is shaded)
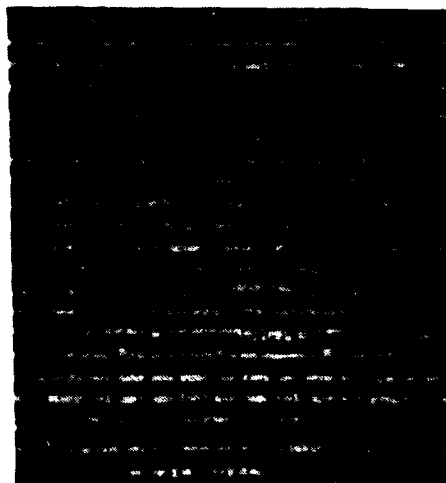
Finally, the cell contains a sink diode, ARV, for erasing data from the array in parallel.

The periphery of each cell is occupied by two three-phase buried-channel shift registers with four distinct phases in groups (A,B,C) and (B,C,D). The top right corner is occupied by the matrix corner-turning phase, labeled X.

The remaining area of the cell is used to obtain clock phase interconnection with two polysilicon levels and one metal level. Approximately 30% of the cell area excluding shift registers is used for this purpose in the present conservative design. The 2214 cell dimensions are 265 x 265 μm^2 (10.43 x 10.43 mil^2) and the chip overall dimensions are 230 x 200 mil^2.

Operation of the regenerator is shown schematically in Figures 8(a) and 8(b). The fill-and-spill diode is connected to the C clock, which is the phase into which the generated charge flows directly. This scheme has been demonstrated at RSRE, Malvern, England (Reference 2), to give a linearity equally as good as the more conventional method while eliminating the separate dio. e clock pulse. The SCCD to BCCD transition occurs under the extension of the C-phase gate. This provides a convenient potential barrier to prevent charge sloshing back into the regenerator when the shift register is used independently.

It should be noted that the edges of the array are terminated only on the left and bottom by shift registers. At the top and right there is nowhere for charge to be shifted upwards or to the right. The significance of this was not apparent until the devices were operated. This is described in the following subsection, on measurements.

a. Cross-section through regenerator

b. Operating states

Figure 8. Regenerator Operation

31

The modifications to the cell, described later, were aimed at eliminating problems with performance degradation rather than changing the functionality of the cell or the array.

Initial Measurements

The clock timing for operating PIP is evidently quite intricate. The cell shift registers are used bidirectionally to maneuver charge packets into and out of the ports of the cell. An example of a maneuver to cause regeneration of the contents of ST1 is shown in Figure 9. Since the number of clocks for control is so large, it is necessary to use a simple model of the cell to observe in real time the action of a given logical state.

The example primitive R_1 shown in Figure 9 illustrates the machine code constraints on hierarchical software structure. It is necessary for the final state to be identical to the initial state so that primitives can be concatenated. This is not necessarily the most efficient way to use the array since a "next destination" may already be available in the front end of a subsequent primitive. These questions are addressed in more detail in Task 5 on Software.

The basic operation of the array can be observed with simple input conditions. For example, if the serial input is off the only charge that can be read out of the array is regenerator offset and leakage. Figure 10 shows the output from such an operating condition with the regenerator controls, RSI and R'VI, adjusted to give no offset, with a frame period of 23 msec. Notice that the peaks are at the leading end of each line, and there are 16 peaks. This charge should ideally have been shifted off the end of the array during the I/O sequence, but instead has summed at the righthand cell in any convenient "on" gate. The charge merely spills to the

32
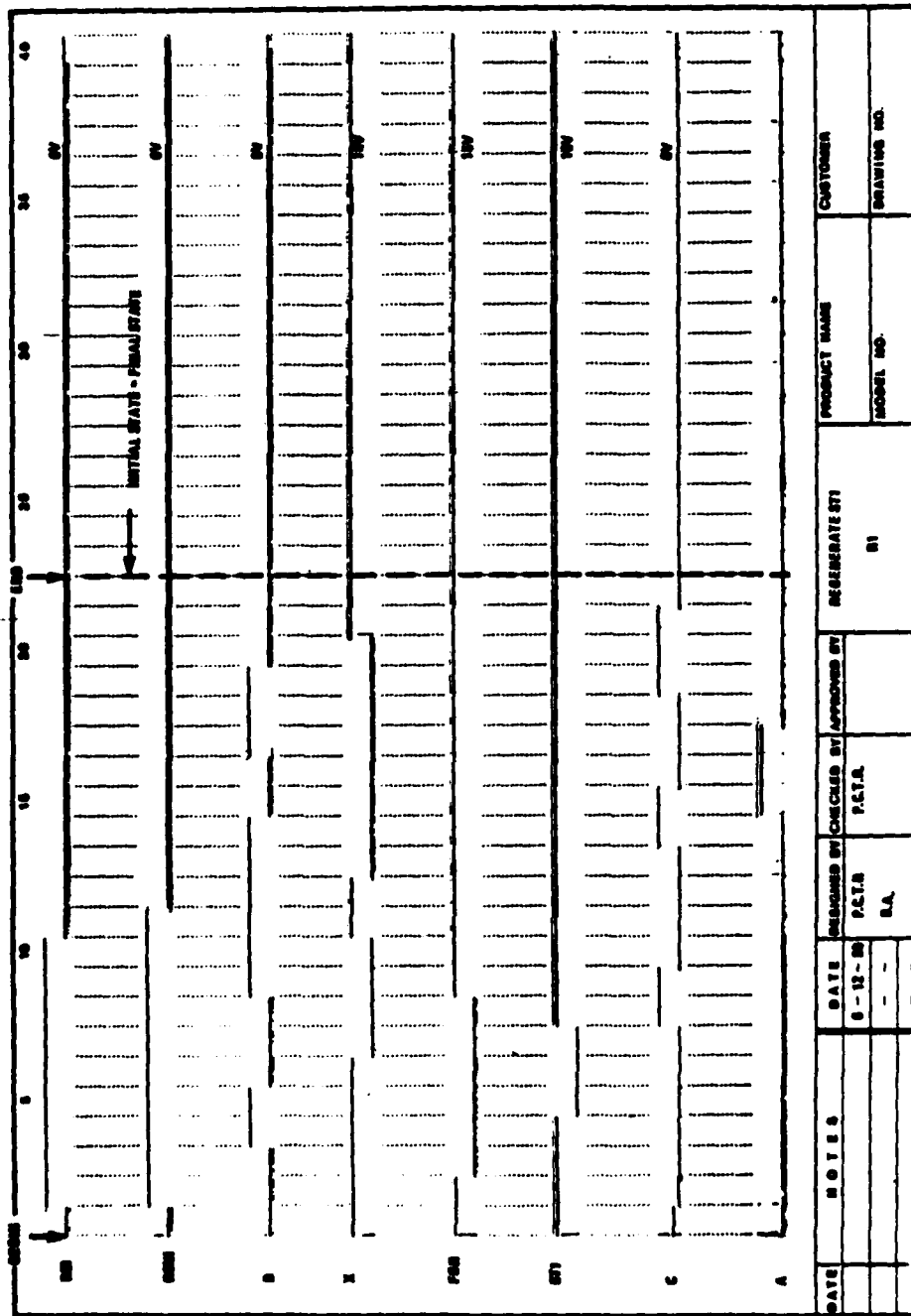
Figure 9. Clock Waveform Timing Diagram for One Primitive (Phases not shown are at rest in initial state.)

33
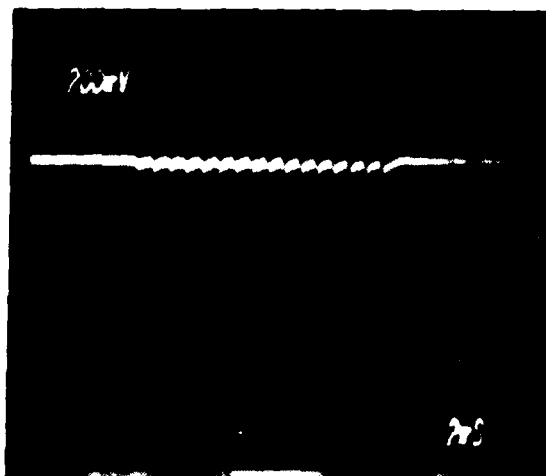
Figure 10. Leakage Residual Buildup at Frame Period of
23 msec, 25°C (2214)

lowest potential if there is no gate to enter normally. This defect can be
eliminated by providing extra cell gates at the top and righthand edge of
the array plus sink diodes to remove the charge.

The linearity and sensitivity of the serial input I/O four-phase register
and sense amplifier are shown in Figure 11 with a triangular input.

These basic functions do not involve arithmetic operations of the cell. A
rather complex operation using all the features of the array is shown in
Figures 12 and 13. The array is loaded with a 6 x 6 uniform pattern and
then the following sequence is performed:

1. ERASE ST1 ⎫
2. ERASE ST2 ⎬   (input data retained in X)
3. ERASE ST3 ⎭
4. LOAD ST1

34
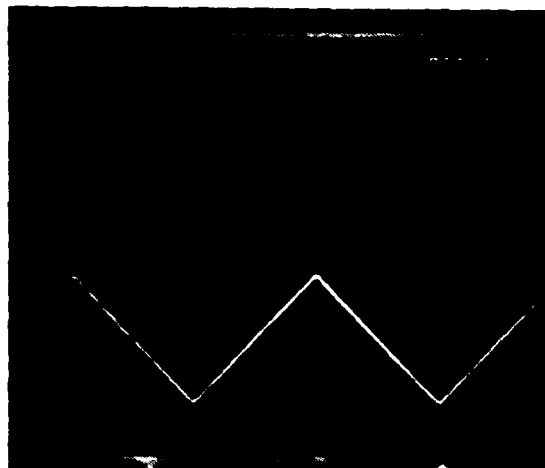
Figure 11. Input Fill and Spill to Sense 7-amp Performance Using
Serial I/O Registers Only

Figure 12. XY Display Using the Routine:
E1 E2 E3 L1 R1 L2 R1 L2 U2
SU SU SU SL SL SL L2 M2

35

Figure 13.   Video Waveform at Top Line of 6 x 6 Test Pattern
(Zero reference at center of screen.)

   5.   REGENERATE ST1 (copy of ST1 retained in X)

   6.   LOAD ST2

   7.   REGENERATE ST1 (copy of ST1 retained in X again)

   8.   LOAD ST2

At this point register addition has occurred.

   9.   UNLOAD ST2

  10.   SHIFT UP 3 TIMES

  11.   SHIFT LEFT 3 TIMES

At this point the double copy of the original pattern is now shifted to the upper left corner of its original position. The original data remains in ST1 and in the original position.

  12.   LOAD ST2

  13.   SUBTRACT ST2 FROM ST1

This last primitive causes the difference between the charge in ST2 and ST1 to be generated and placed in X. The result is then observed by executing an I/O parallel-to-serial readout sequence.

The result of this sequence of primitives should be a 6 x 6 square with one 3 x 3 corner reduced by subtraction. All other output locations should be zero. Notice, however, that the shift operations in both axes have caused the leakage charge to appear both at the bottom of the display and the right side. By cooling the array it is possible to eliminate these spurious outputs almost entirely. The major importance of this result is that the array is able to perform the primary functions in a manner very close to that desired.

Figure 14 shows a typical output video from the original tester buffer circuits directly after the CCD. The circuits are simply impedance and offset buffers to provide drive for the S/H amplifier driving the data logging system. No filtering or integration has been applied. The low gain of the 2214's regenerator is evident when compared to an unregenerated output signal of over 1V. Figure 14(a) shows an entire frame (256 pixels) and Figure 14(b) shows several lines (16 pixels each) in which data is nonzero. Nonuniformity is clearly visible.

Leakage current and transfer inefficiency data have been analyzed, together with various uniformity performance data. Typical data are shown in Figures 15 to 31, which have been logged, plotted, and analyzed using an on-line HP-IB desktop computer system. Since the quantity of the data is so large for two-dimensional arrays (even a small 16 x 16), only the major points will be covered.
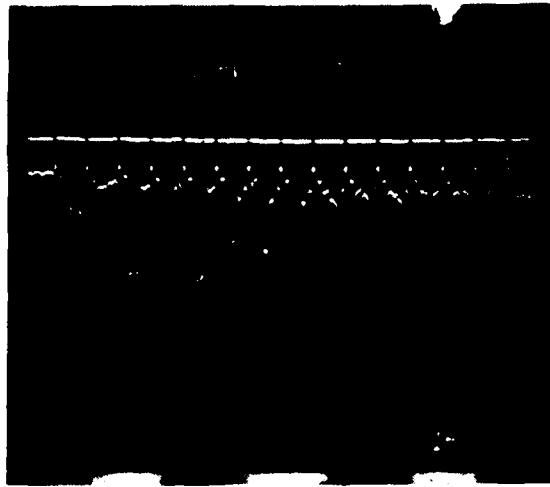
a. Whole frame video (256 pixels) of SUBTRACTOR
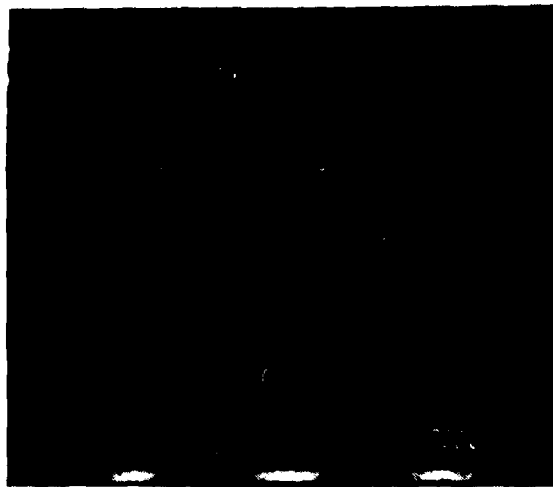
b. Details of three lines (16 pixels each)

Figure 14. Typical Video Output After Buffer

38

Figure 15 shows the result of shifting zero data to the right using the instruction set E_1, E_2, E_3, I/O (load) I/O (unload). We expect to see an accumulation of leakage charge in the 16th column as a result of the lack of terminations to the array. However, although there is evidently about 60 mV output at the appropriate positions for the 16th column (pixel #1 is bottom right, pixel #16 is bottom left, etc., pixel #256 is top left), the output is drastically smaller than that shown in Figure 16. For Figure 16 the instruction set was changed to E_1, E_2, E_3 SU(16) I/O (unload). This shows the expected 800 mV output corresponding to the top row lack of termination.

The anomaly was resolved by the presence of an observed layout error in the 16th column shift register. There is a phase sequence connection fault on the 16th column giving an (X, X, B, C, B, D, ...) instead of the intended (X, D, C, B, D,...). This simple isolated error was corrected on the #3022 chip mask set.

Figures 17 through 27 show the leakage charge accumulated in each cell store when the routines

E_2 E_3 U_1 I/O (unload),
E_1 E_3 U_3 I/O (unload), and
E_1 E_2 U_2 I/O (unload)

are performed both at room temperature ($\sim 25^{\circ}$C) and cooled ($\sim ^{\circ}$C). The integration time in each case is about 23 msec.

The gate area of ST1 = 6.9×10^{-6} cm^2,
          of ST2 = 7.2×10^{-6} cm^2, and
          of ST3 = 7.74×10^{-6} cm^2.

SHIFT RIGHT LEAKAGE 1/9/81
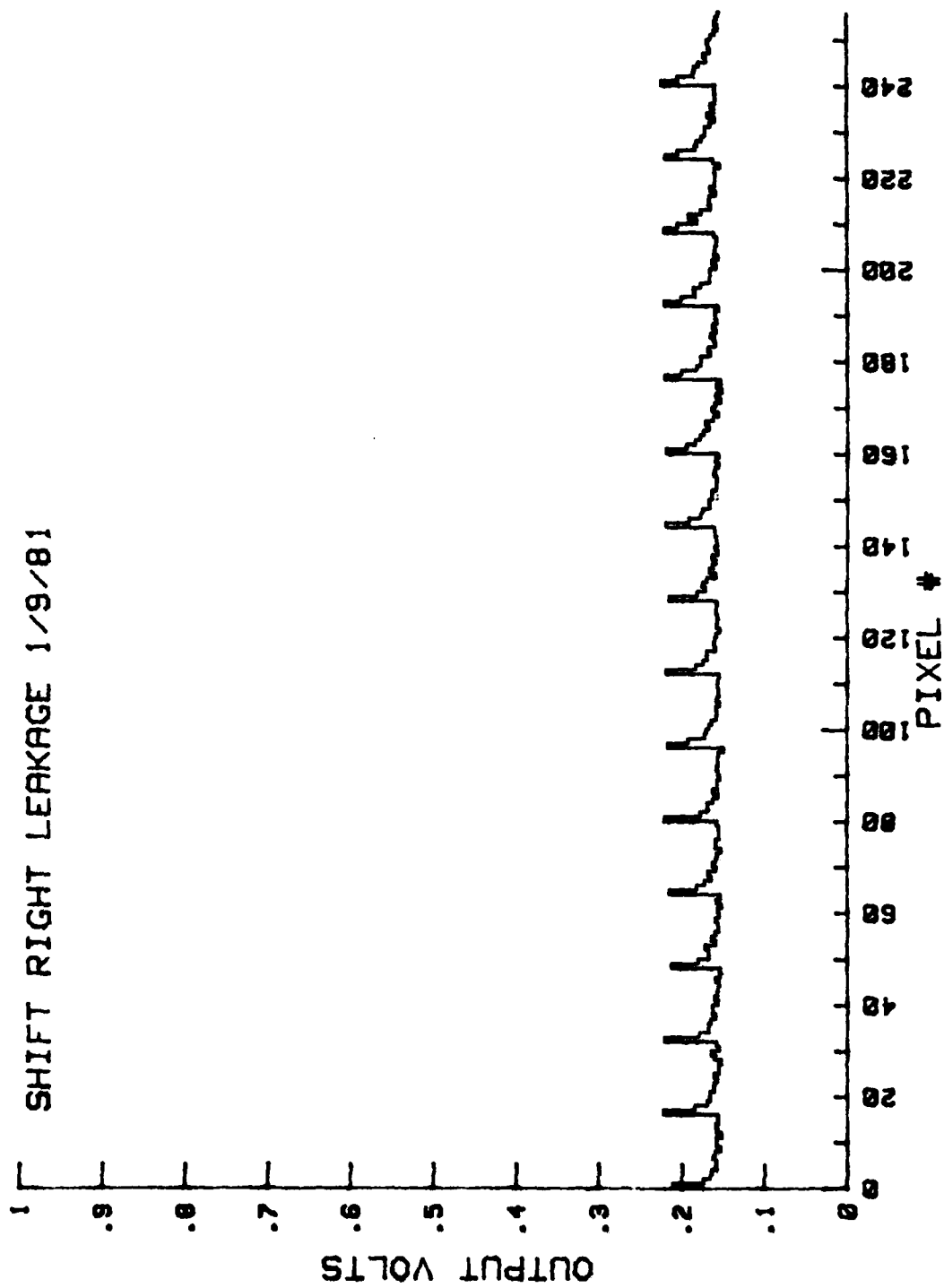
Figure 15. Zero Data Leakage Measurement: Shift Right (2214)

40
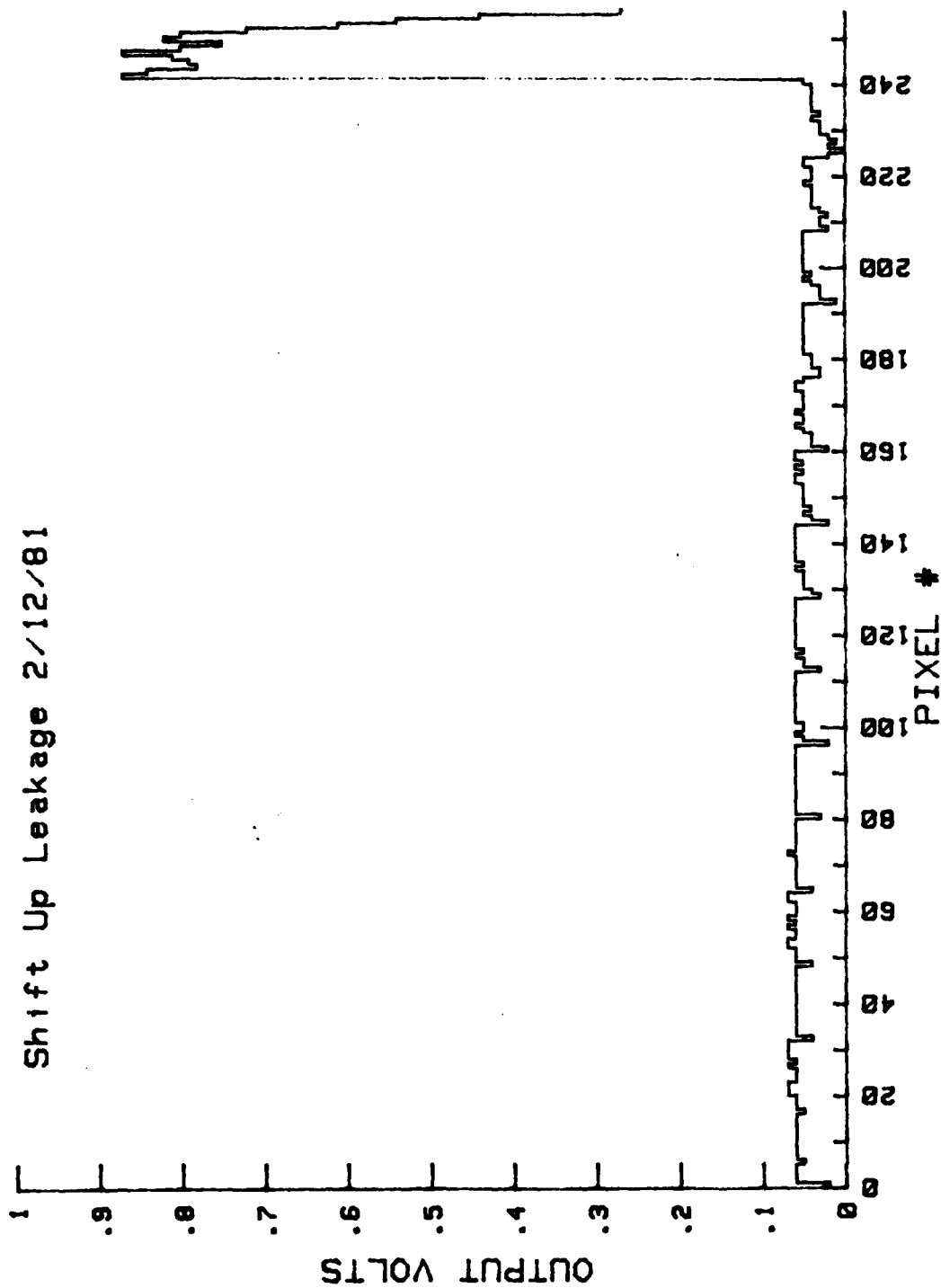
Figure 16. Zero Data Leakage Measurement: Shift Up (2214)
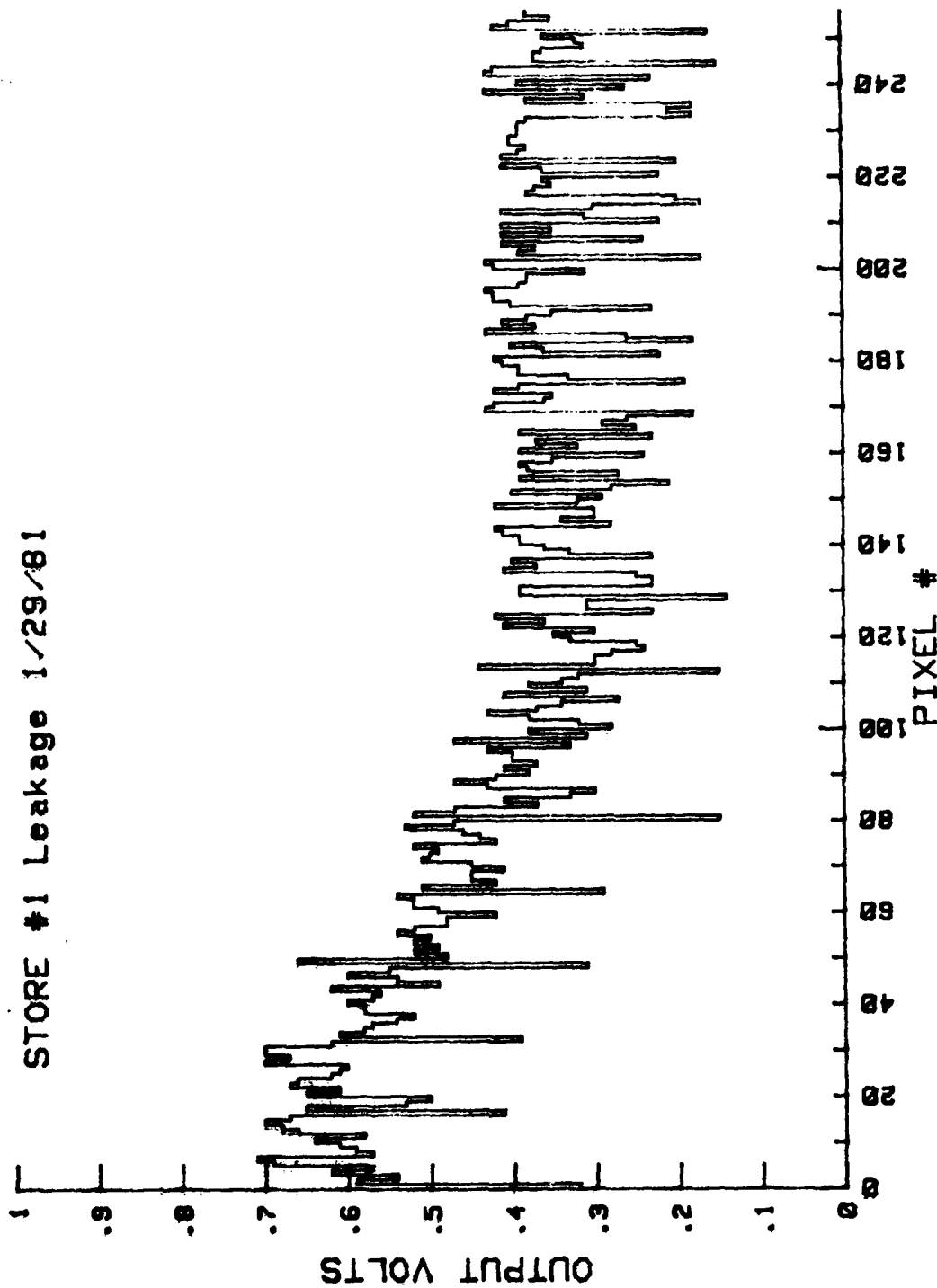
Figure 17. Integration Mode Leakage of ST1:   Room Temperature
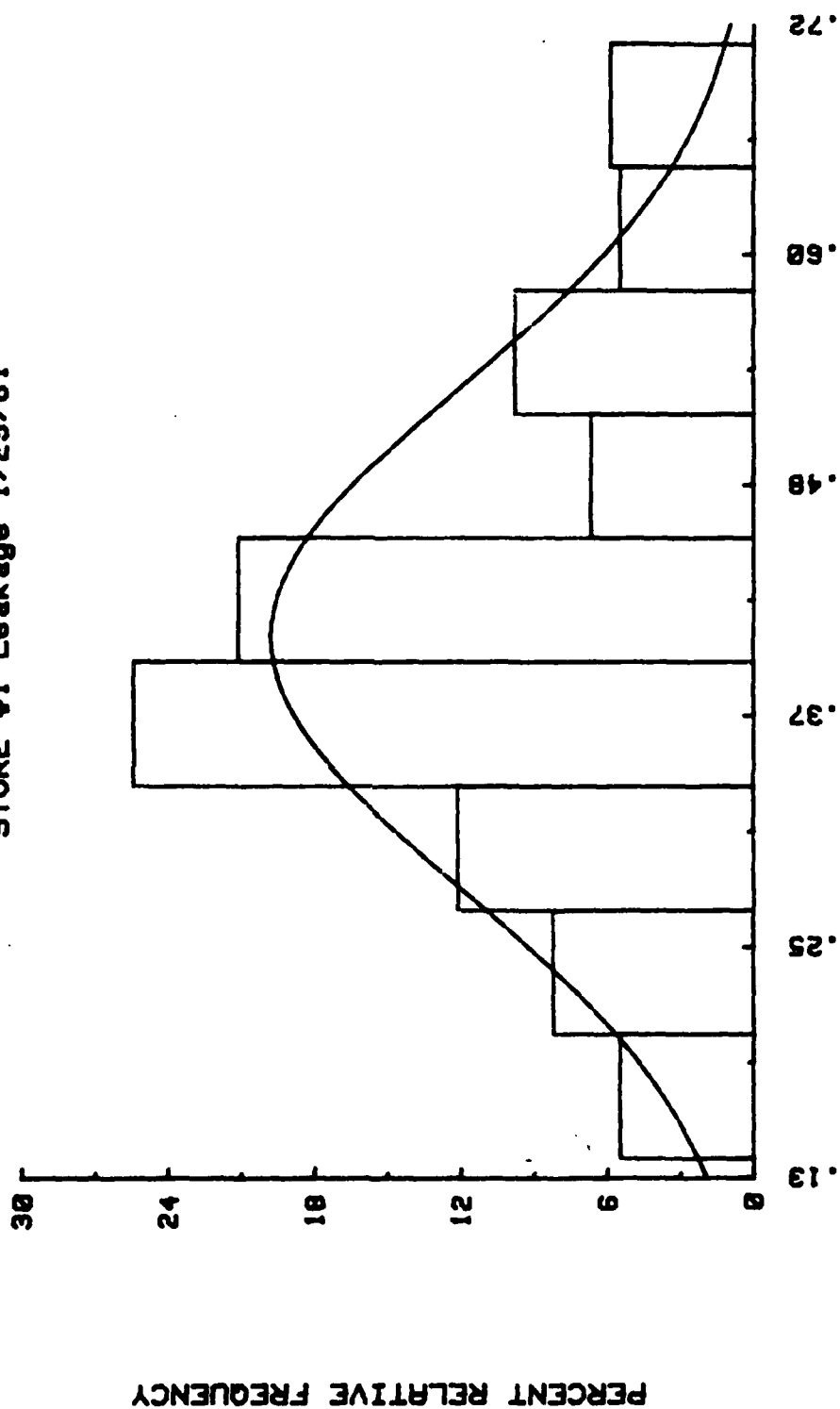
Figure 18. Histogram of Data in Figure 17

43
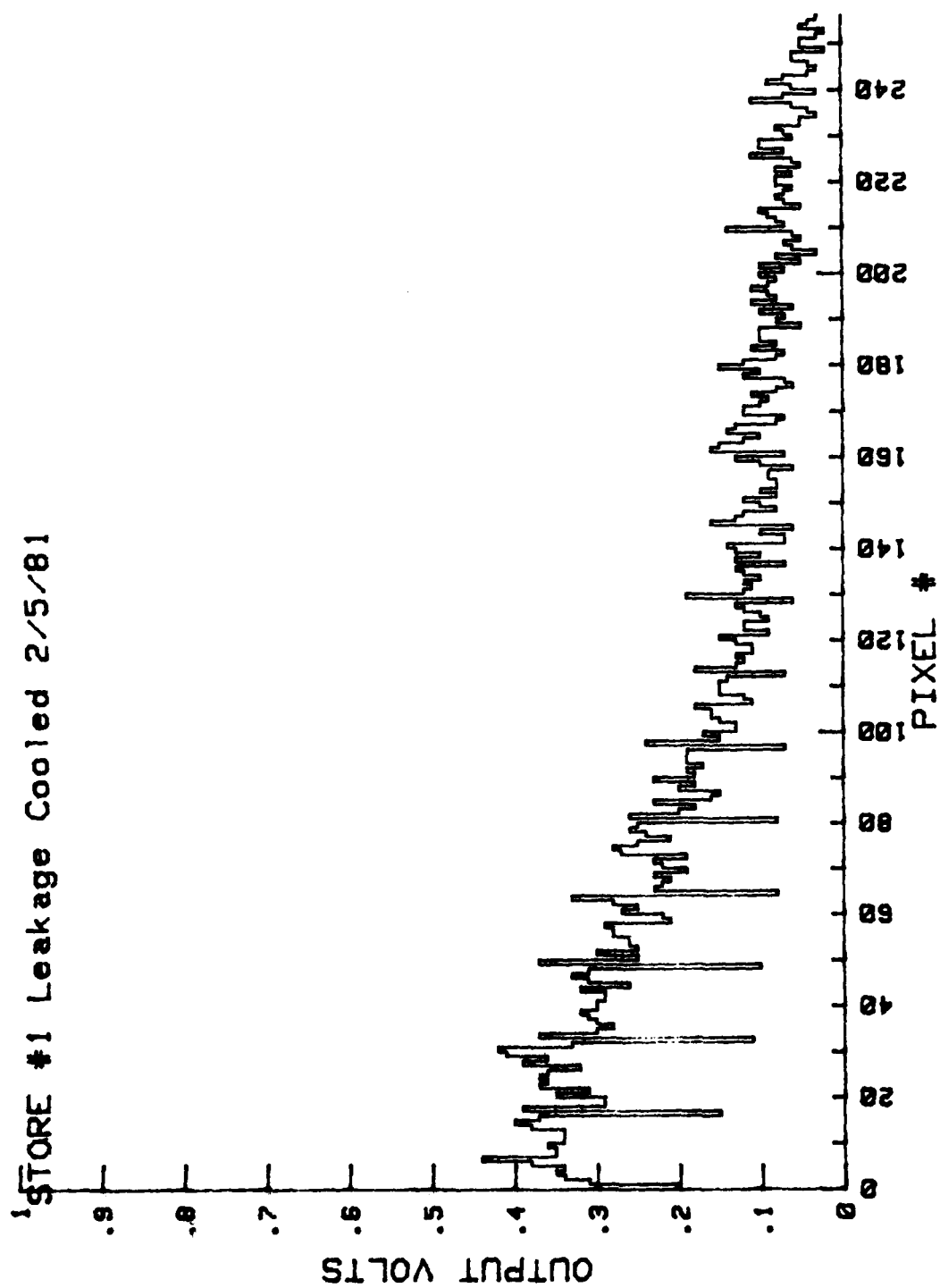
Figure 19. Integration Mode Leakage of ST1: Cooled ~ 0°C
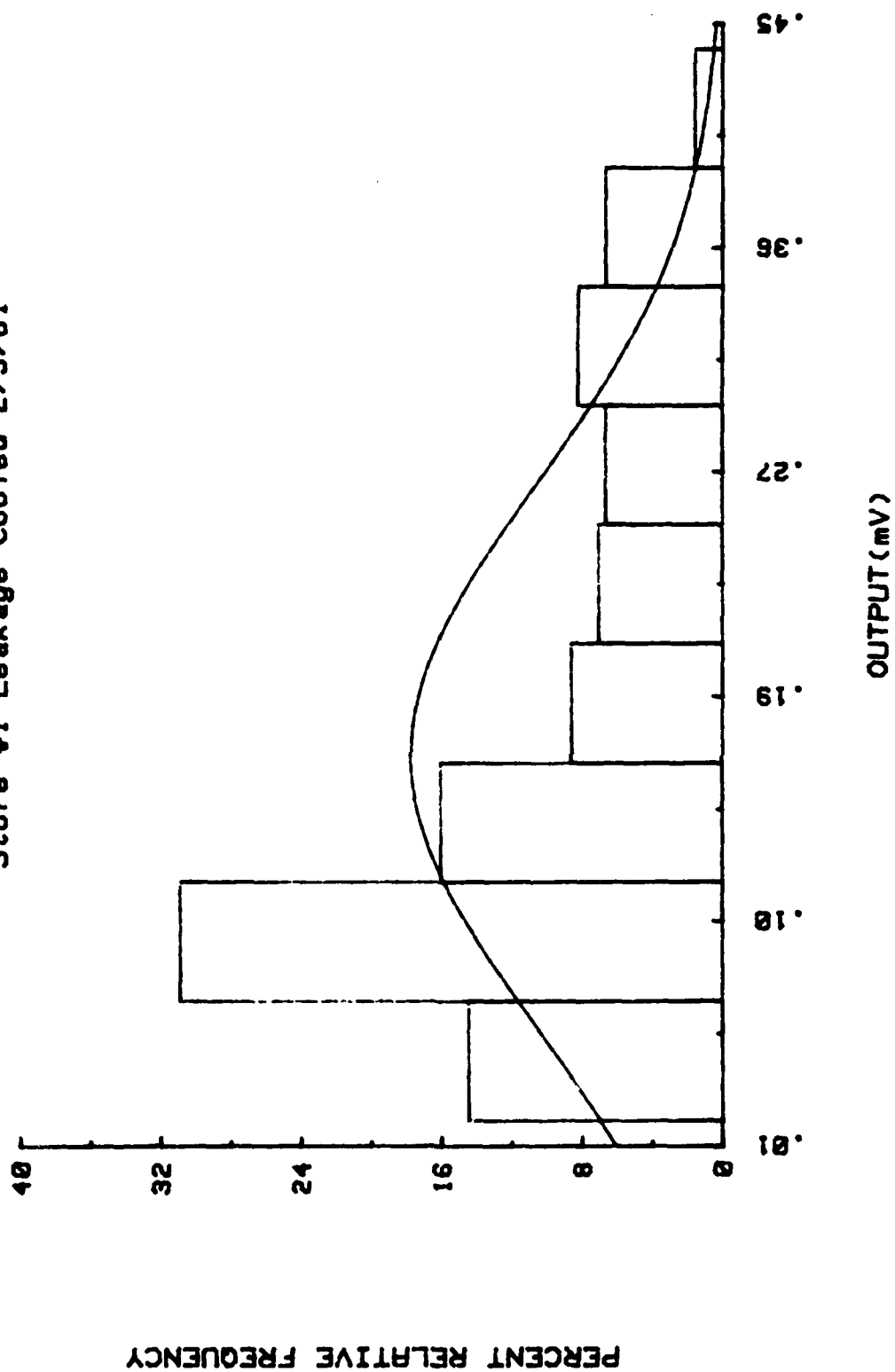
Figure 20. Histogram of Data in Figure 19

45
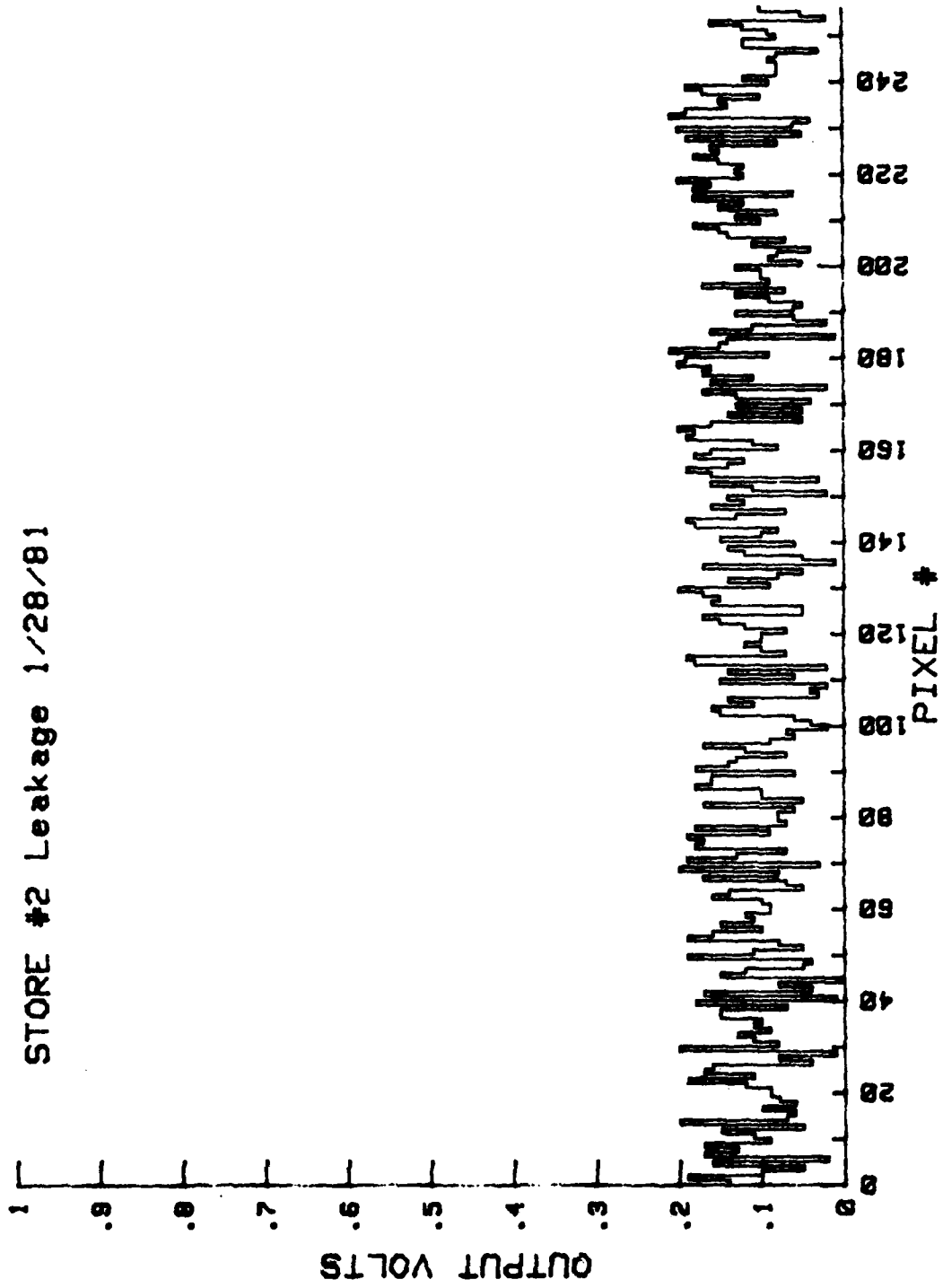
Figure 21. Integration Mode Leakage of ST2: Room Temperature
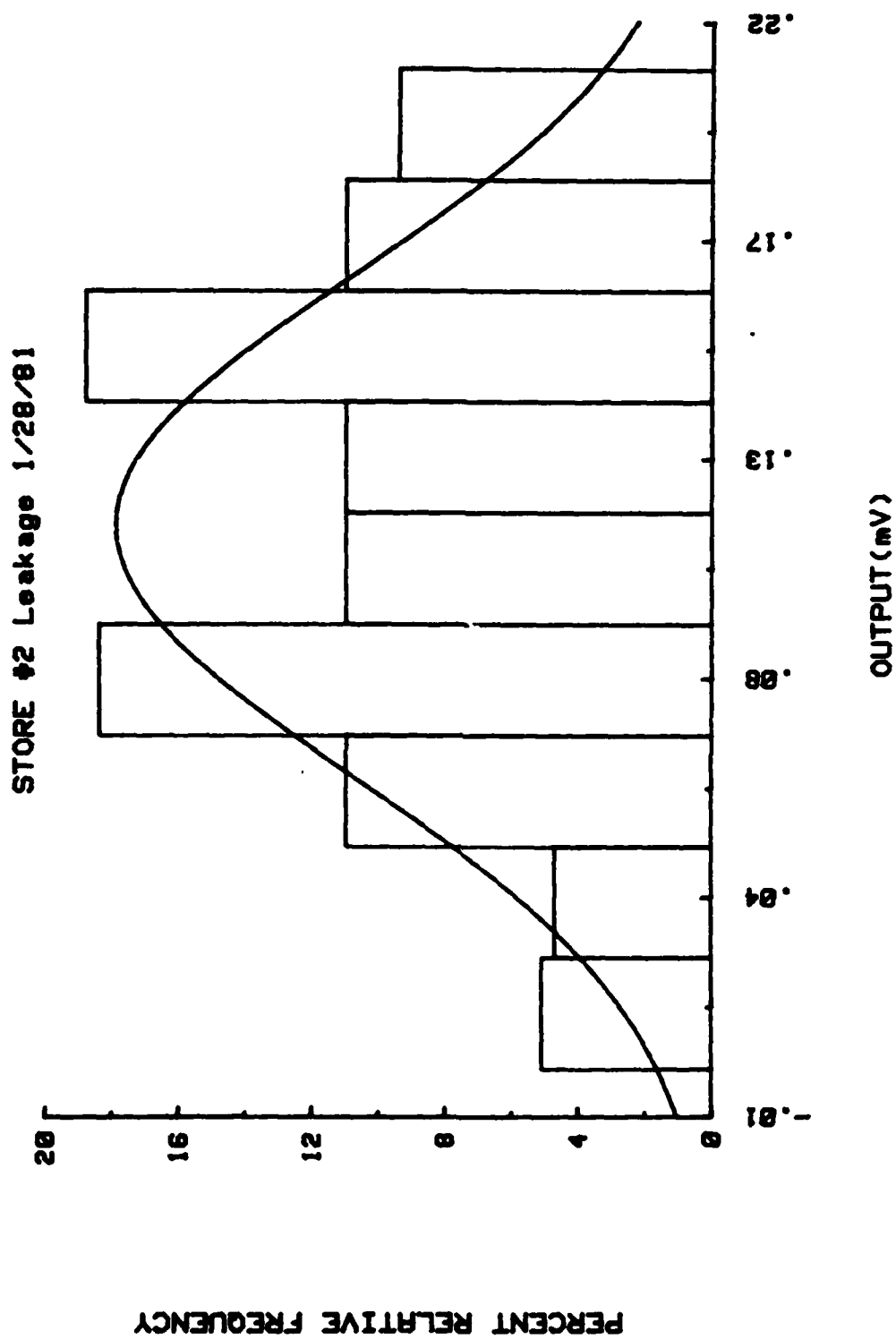
Figure 22. Histogram of Data in Figure 21

47

STORE #2 Leakage Cooled 2/6/81

OUTPUT VOLTS

PIXEL #
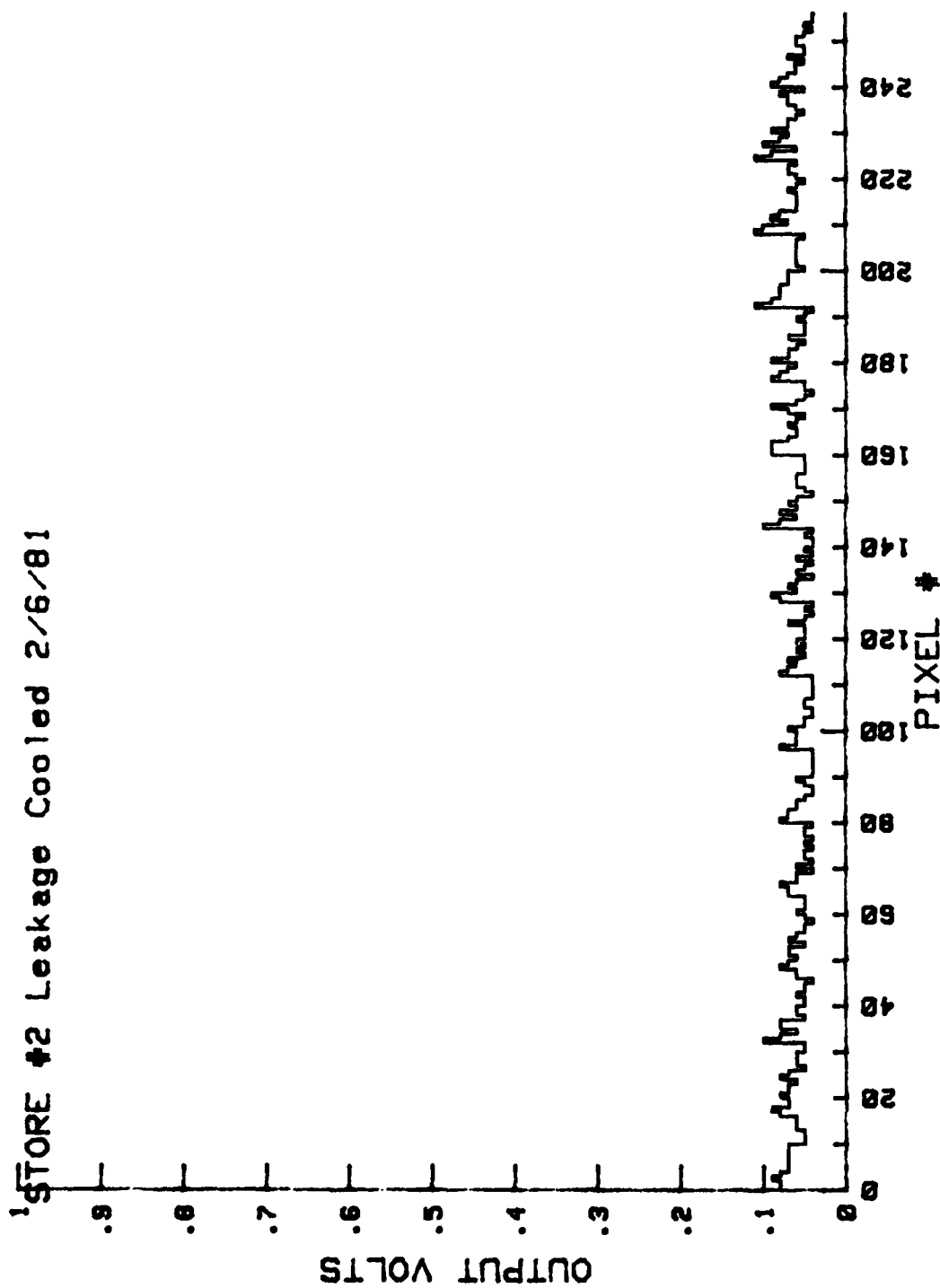
Figure 23. Integration Mode Leakage of ST2: Cooled ~ 0°C

48

STORE #3 Leakage 2/28/81

OUTPUT VOLTS

PIXEL #
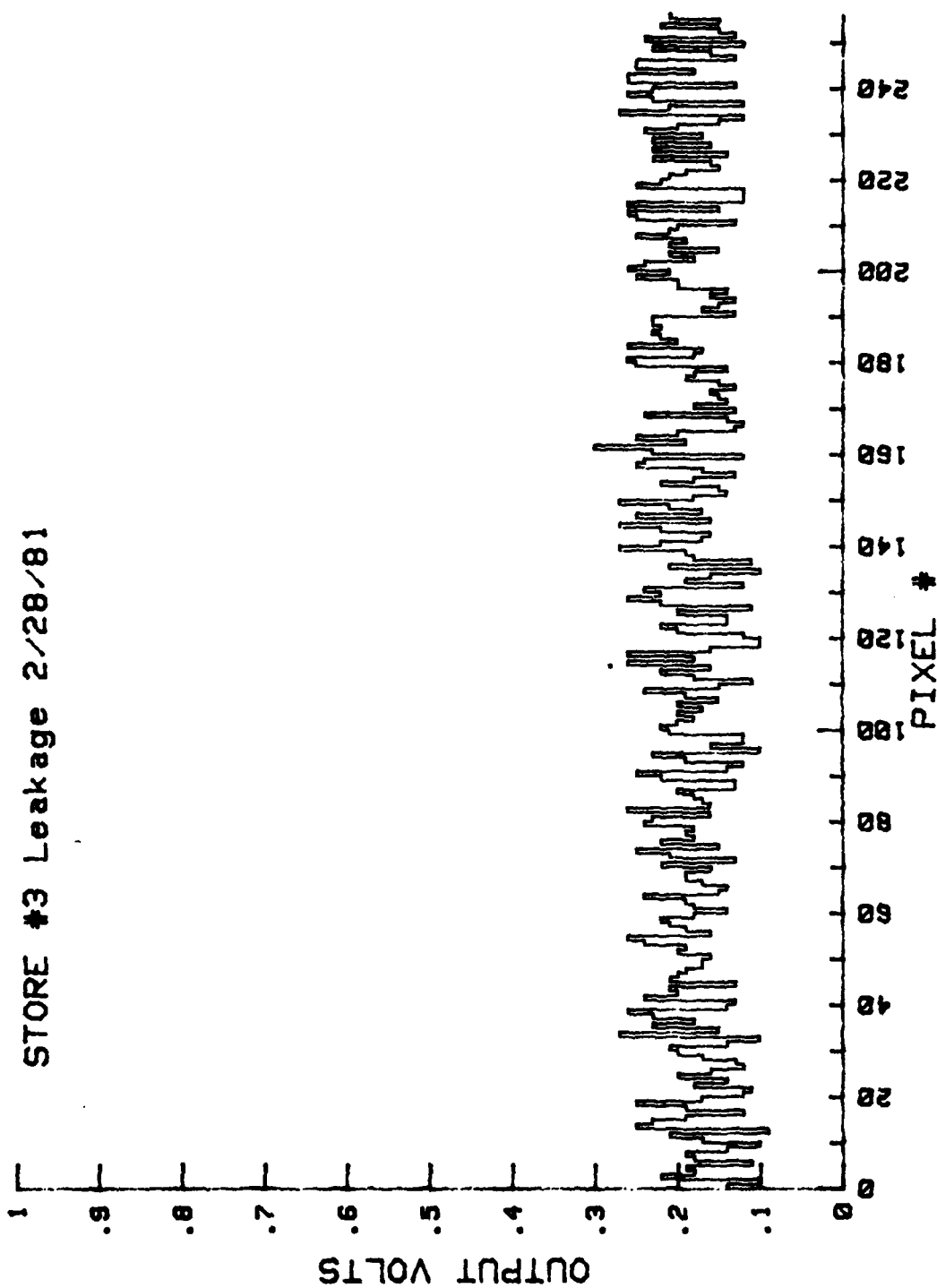
Figure 24. Integration Mode Leakage of ST3: Room Temperature

STORE #3 Leakage 1/28/81

PERCENT RELATIVE FREQUENCY
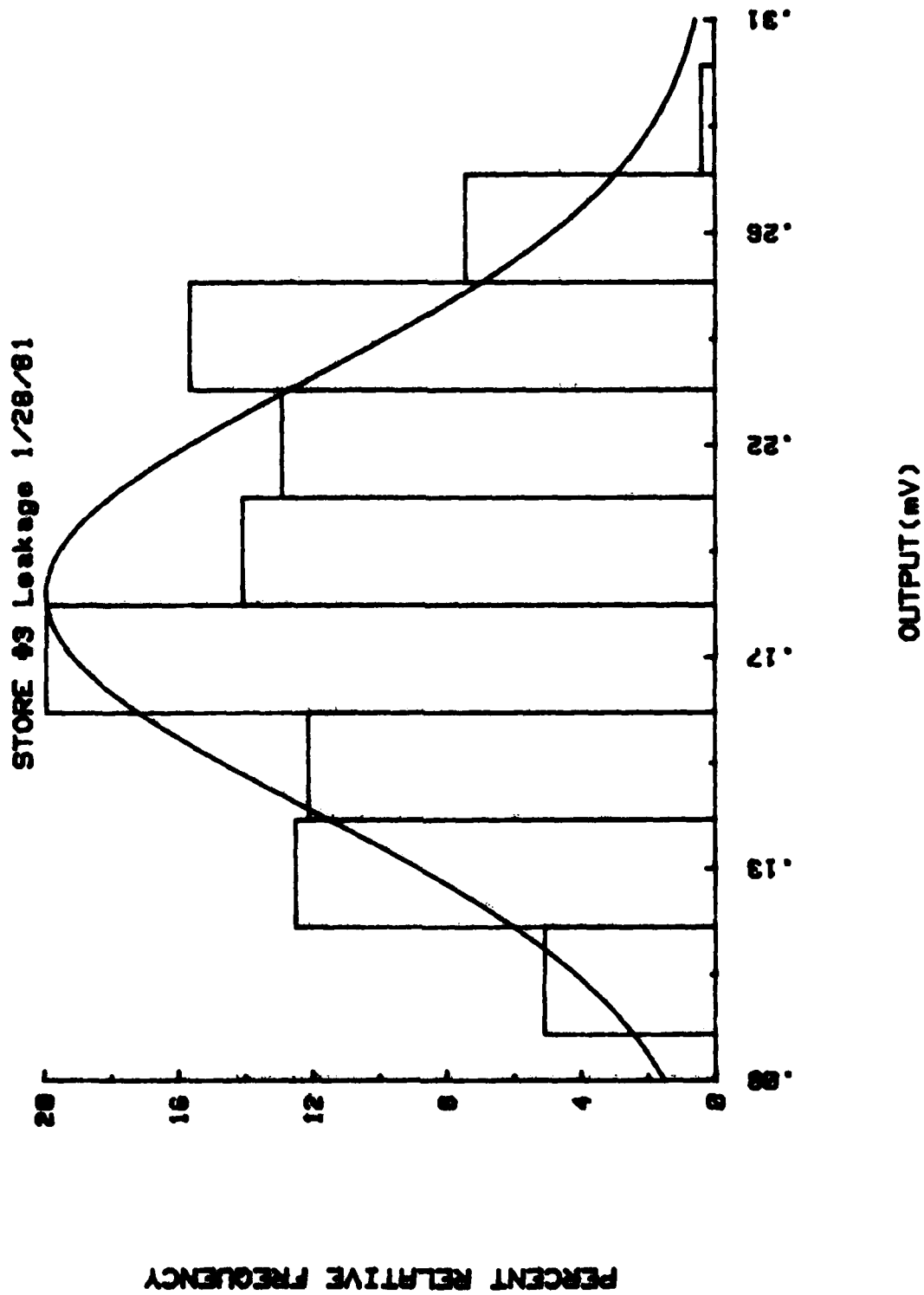
OUTPUT(mV)

Figure 25. Histogram of Data in Figure 24
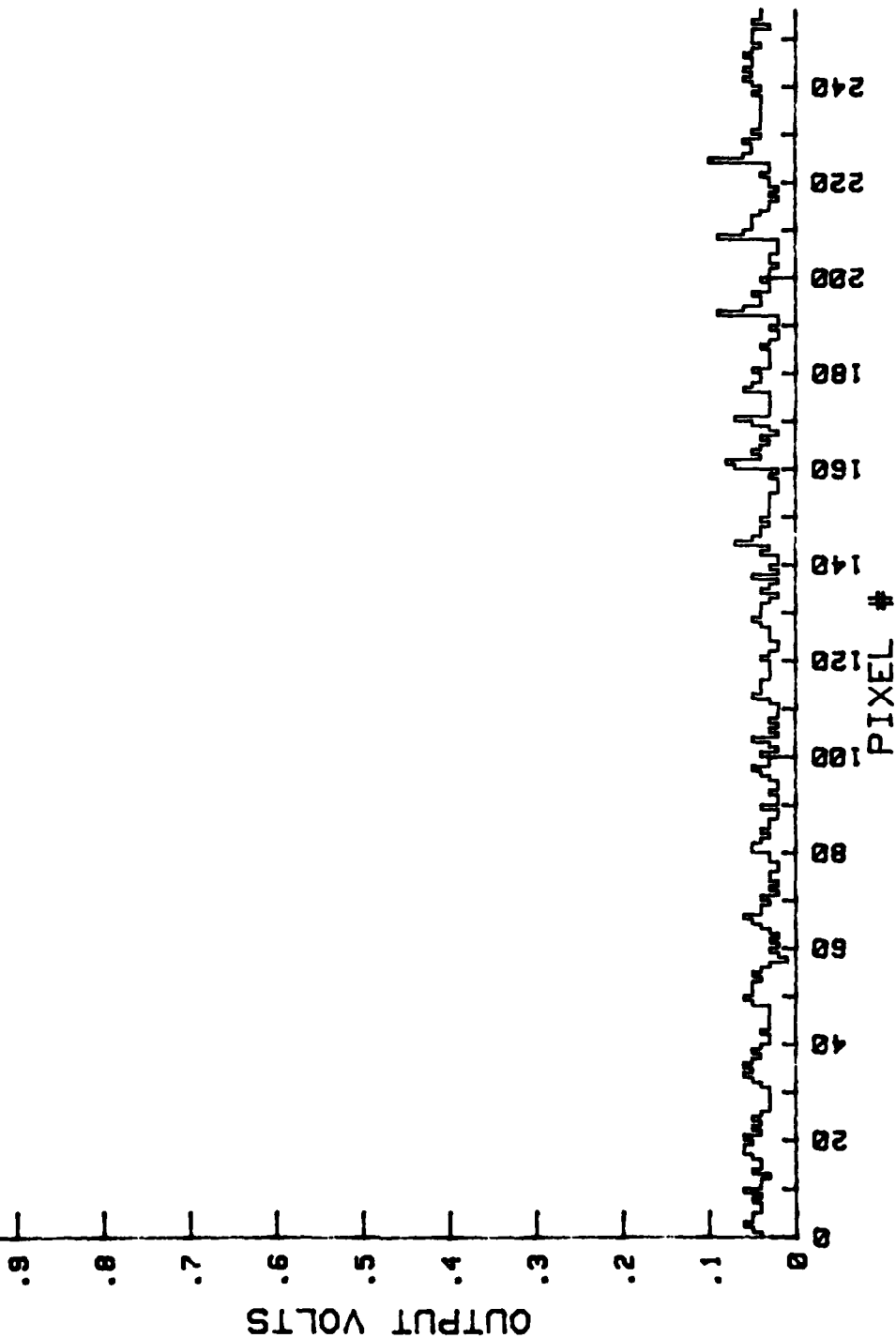
Figure 26. Integration Mode Leakage of ST3: Room Temperature
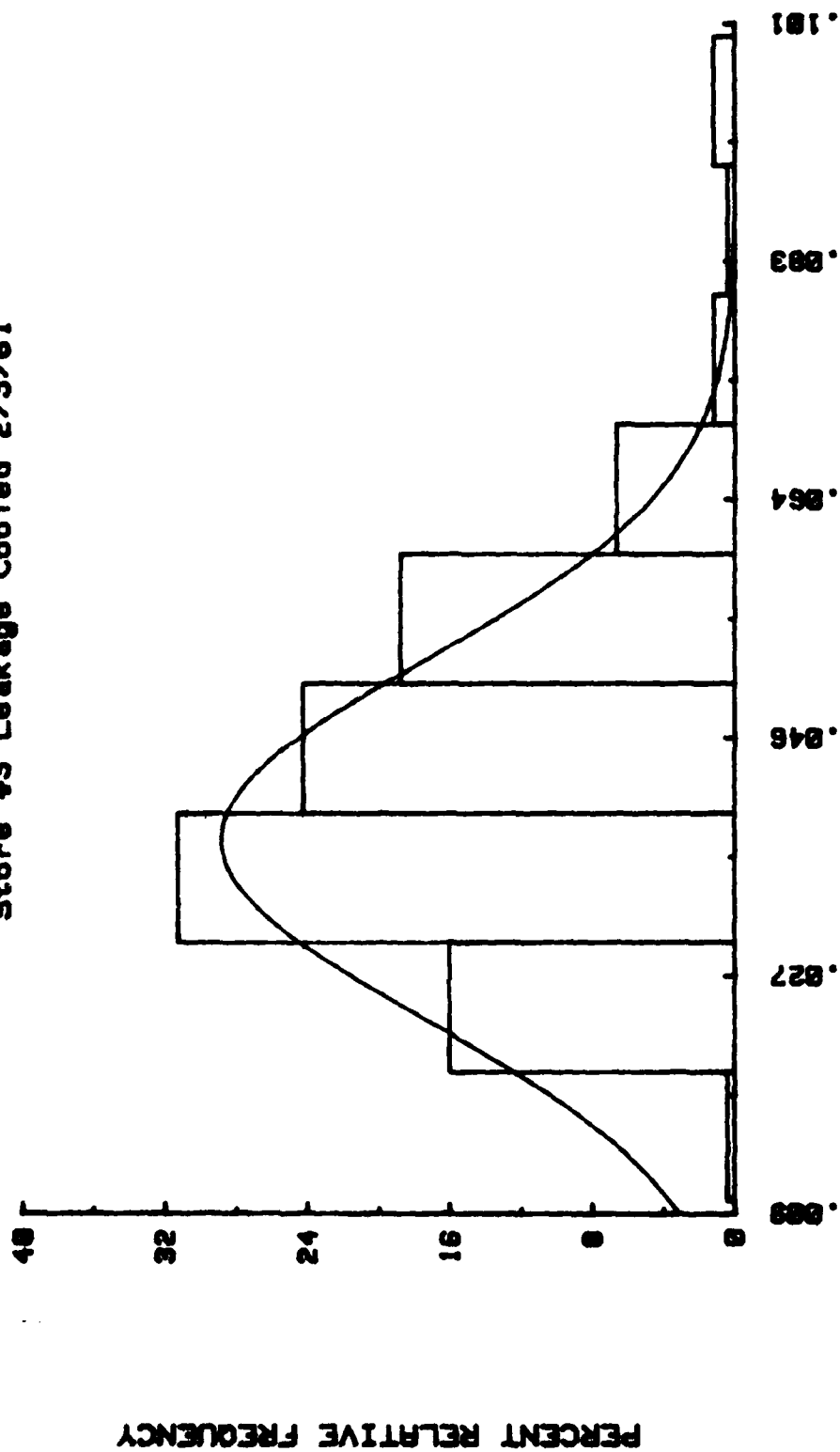
Figure 27. Histogram of Data in Figure 26

One volt output corresponds to about 1 pC charge packet.

Evidently, ST1 uniquely suffers from a nonuniform leakage current. The magnitude of the leakage current at 25°C is about $2.5\,\mu A\;cm^{-2}$ and greatly exceeds the expected value of 10 to 100 $nA\;cm^{-2}$. ST2 and ST3 are close to the expected leakage at about 300 $nA\;cm^{-2}$ and are uniformly distributed (gaussian).

The only physical correlation with the leakage behavior of ST1 that could account for a nonuniform signature is a nonuniform proximity of the diffusion feeding the C clock to the regenerator fill-and-spill diode. We have searched for such a variation without much success using an SEM on actual chips. The diode-poly spacing was, however, designed to be $4\mu m$, which is too close for comfort with short channel V_t effects evident in FETs at that gate length. The 3022 chip has double that spacing, which should eliminate the effect entirely. It should be noted that in normal operation of the array, we erase all stores immediately after loading the array using I/O (load). The leakage should not, therefore, be a problem even at its evident magnitude at 25°C.

Histogram plots of each set of measurements are shown in sequence with the pixel distribution.

Figure 28 shows a plot of the 2214's response to simple loading and unloading of the perimeter I/O registers with a 6 x 6 block of unit data.

Transfer inefficiency effects will be the only contribution to distortion here. Evidently nothing very significant has occurred with the 2214's (16 x 8 and 16 x 8 + 4 = 260) transfers. Figure 29 is a plot of the same data as Figure 28, to show the fat zero level of ~ 8% and the trailing inefficiency product of about 25 mV. This corresponds to an inefficiency, ϵ, of about 8 x 10^{-5} (α = 0.99992) at a clock rate of about 100 kHz--the expected performance of a BCCD.
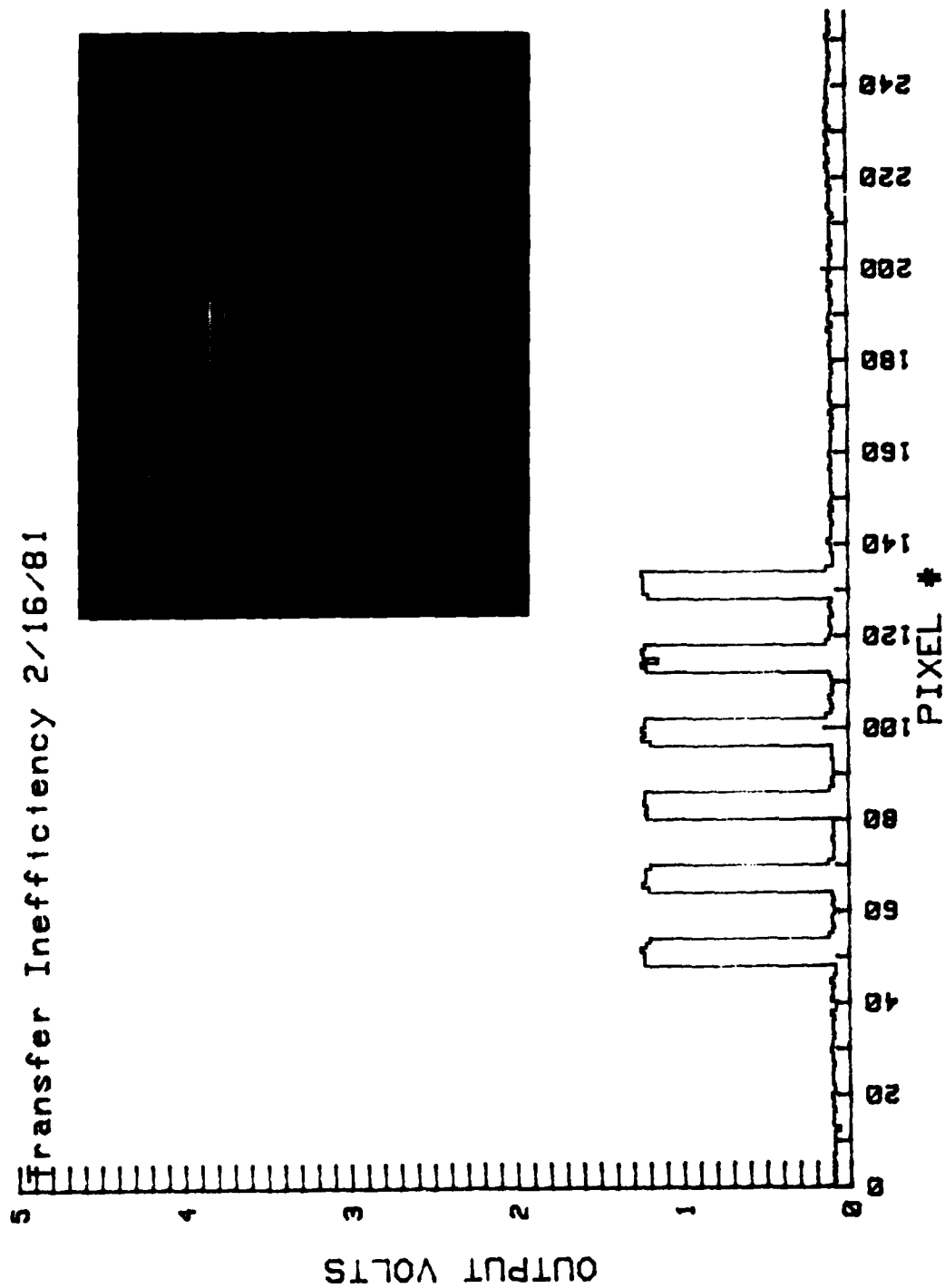
Figure 28. Transfer Inefficiency Effects in I/O Registers

54
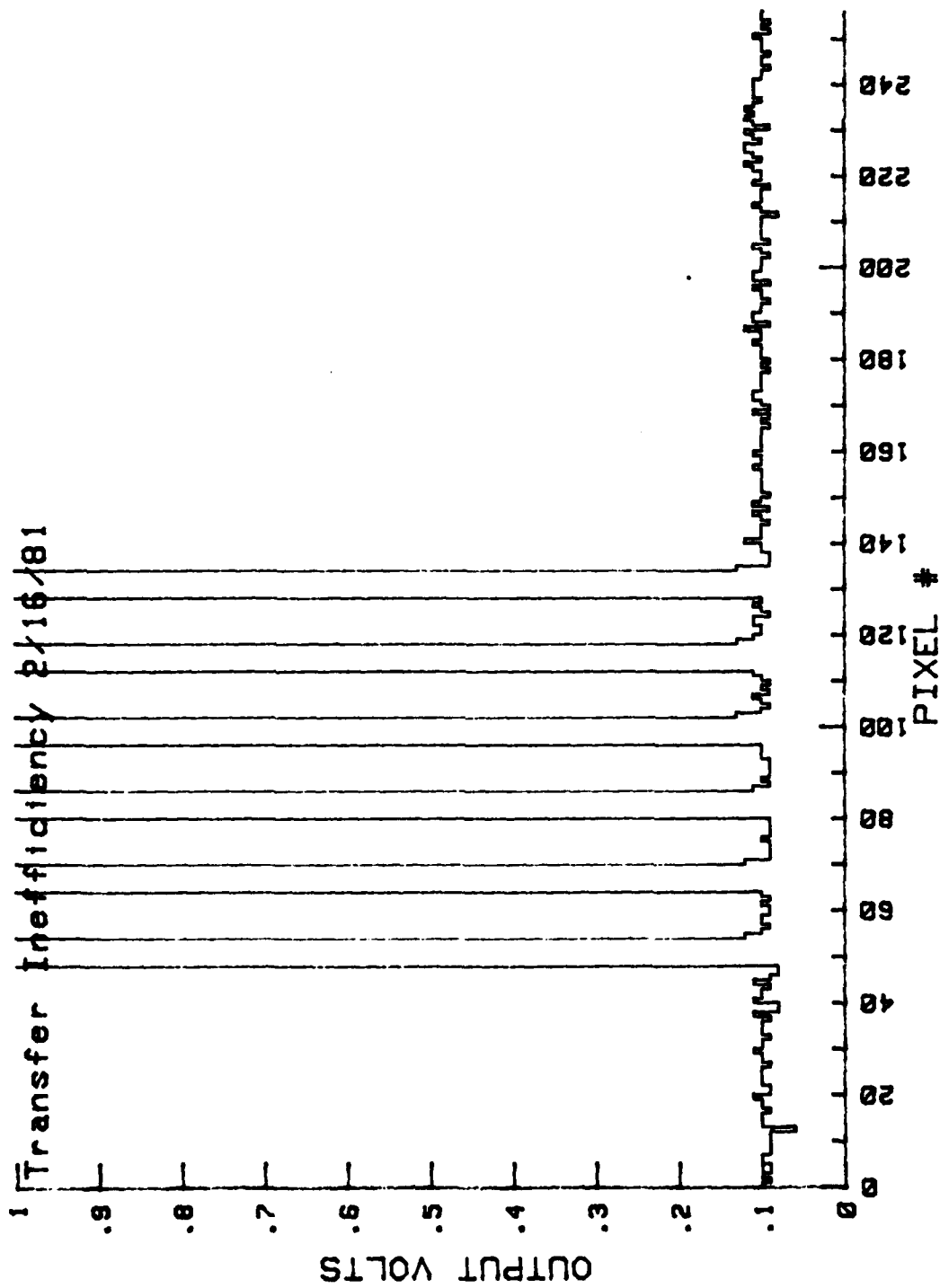
Figure 29. Magnified Scale of Data in Figure 28

Figure 30 shows the same test as Figure 28, but this time the array is loaded
in addition to shifting through the I/O registers. Now, we observe the
leakage accumulation outputs (16th column missing) and again a good ϵ of
about the same value. The number of transfers in this case varies depending
on the location of the data, but is roughly equivalent to the previous case.
Evidently, corner-turning maneuvers are no problem with the 2214.

There are no unexplained phenomena in the testing of the 2214 concerning
leakage, transfer inefficiency, and uniformity. These results and the
analysis of the defective properties were taken into account in the redesign
of the cell and array termination, described next.

TASK 3--DESIGN ITERATION AND FABRICATION OF 16 x 16 ARRAY

Design Modifications

Several major modifications were made to the cell and one was made to the
array termination.

Cell--

1. Regenerator gate structure was changed to reduce loading effects on
   floating gates Z and Y.

2. Charge-charge sensitivity of the regenerator was intended to be
   increased from presently observed ˜ 0.2 to ˜ 1.0.

3. Input gates to the regenerator were screened from C-clock
   breakthrough.

4. Spacing between ST1 and C-clock diode was increased to reduce
   "leakage" into ST1.
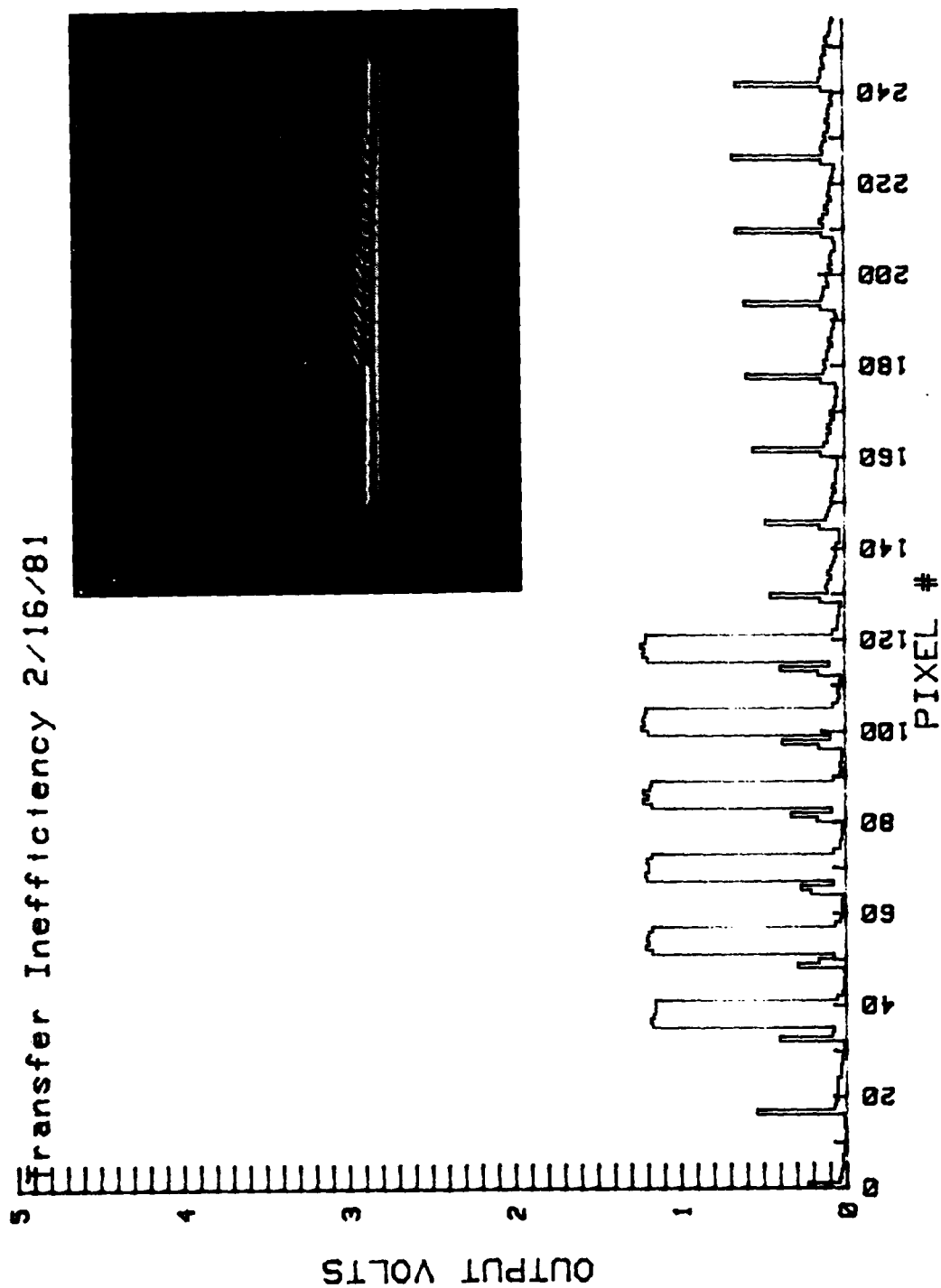
Figure 30. Transfer Inefficiency Effects in I/O and Array Registers

57
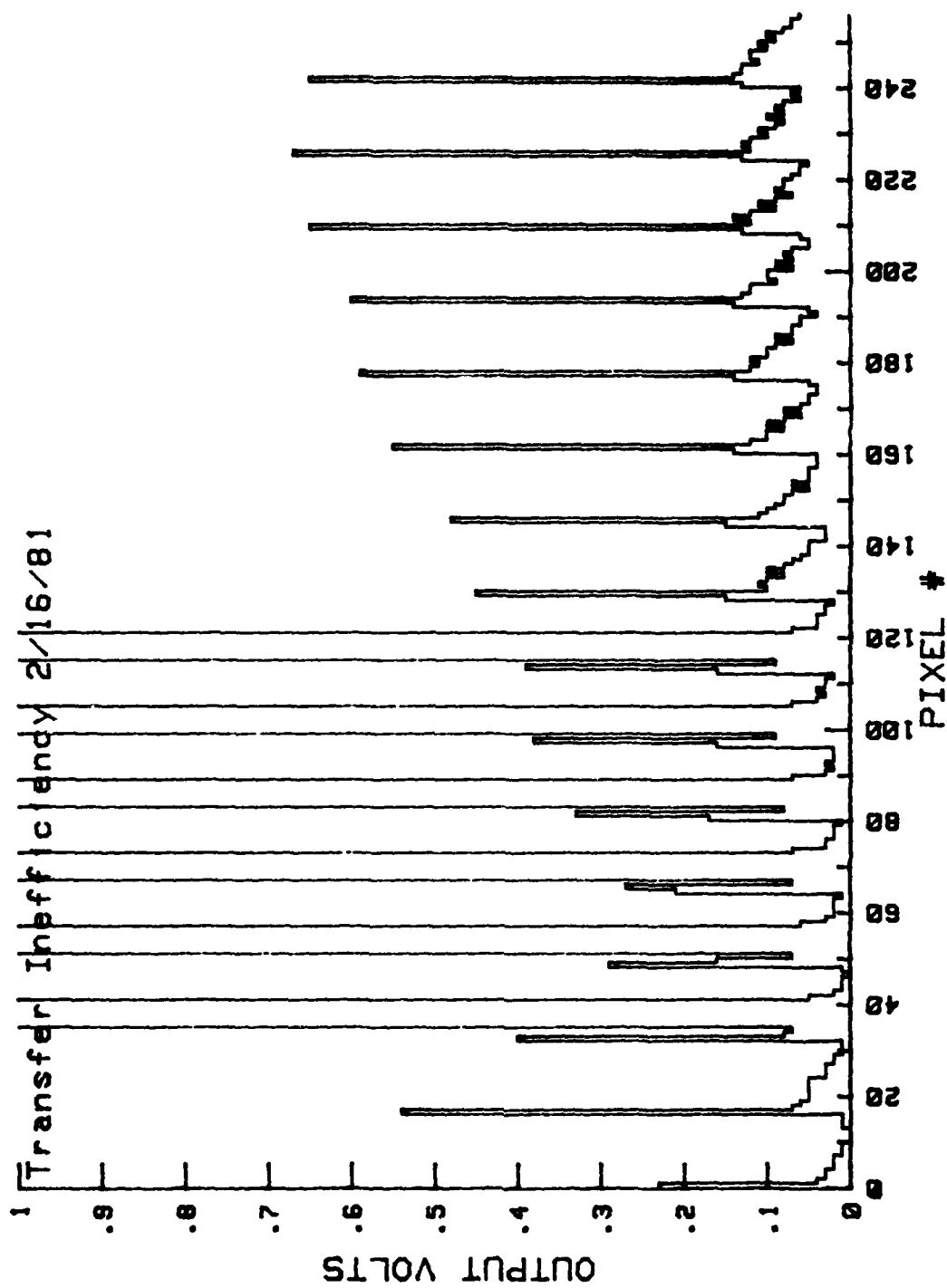
Figure 31. Magnified Scale of Data in Figure 30

5. The multiplier was changed to produce a linear transfer function in the regenerator. The present design gives "square law" distortion products. This new Honeywell design has been described in another RADC Contract, the CCD Applications Study (Reference 3).

Array--

1. Dummy cell gates (A,B) and new phase T were added to the right edge termination, and a diode was connected to ARV. Similar dummy gates (B,C), new phase T, and an ARV diode were added to the upper edge termination. This will eliminate the major effects of thermal leakage charge buildup observed in measurements.

The new generator structure is shown in cross-section in Figure 32. Introduction of screen gate connected to RSI gives a slightly different arithmetic operation. The output consists of three terms:

1. $Q_{OUT} = K_1 Q_Z - K_2 Q_Y + K_3 Q_Y$

$$= K_1 Q_Z - K_2^1 Q_Y$$

2. K_1 and K_2 are the floating-gate charge-to-charge sensitivities.

3. K_3 is an extra charge input due to quasi-DC screen gate RSI.

Figures 33 to 36 show the new chip cell layout (#3022) and, as a comparison, the old cell layout (#2214). The shaded regions are Poly 1 in both Figures 33 and 34, and the metal interconnect is shown separately in Figures 35 and 36. Figure 36 indicates the location of the two extra control lines introduced to provide the improved linear multiplier (MD + MG2) and also the exchange of memory control (Xm). The shift registers are not altered from the original design save for the different locations of the contact windows and Poly 1, Poly 2 interconnect. The new regenerator is seen to have appropriate screen gates as described above.

Figure 32. Modified Regenerator Gate Structure Showing Reduced Loading, Increased Sensitivity, and C-Clock Screening
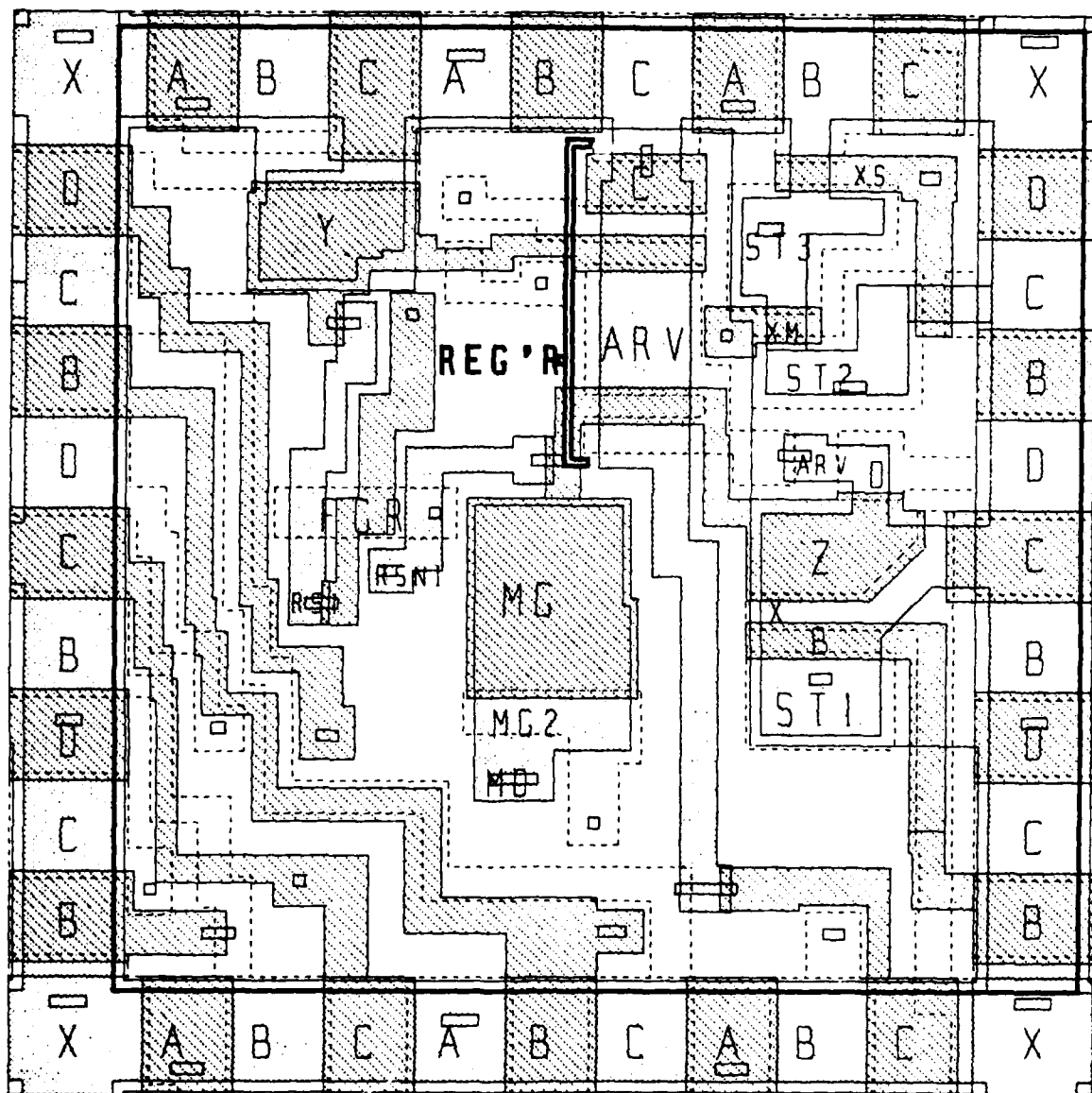
Figure 33.   CALMA Plot of 3022 Cell

61
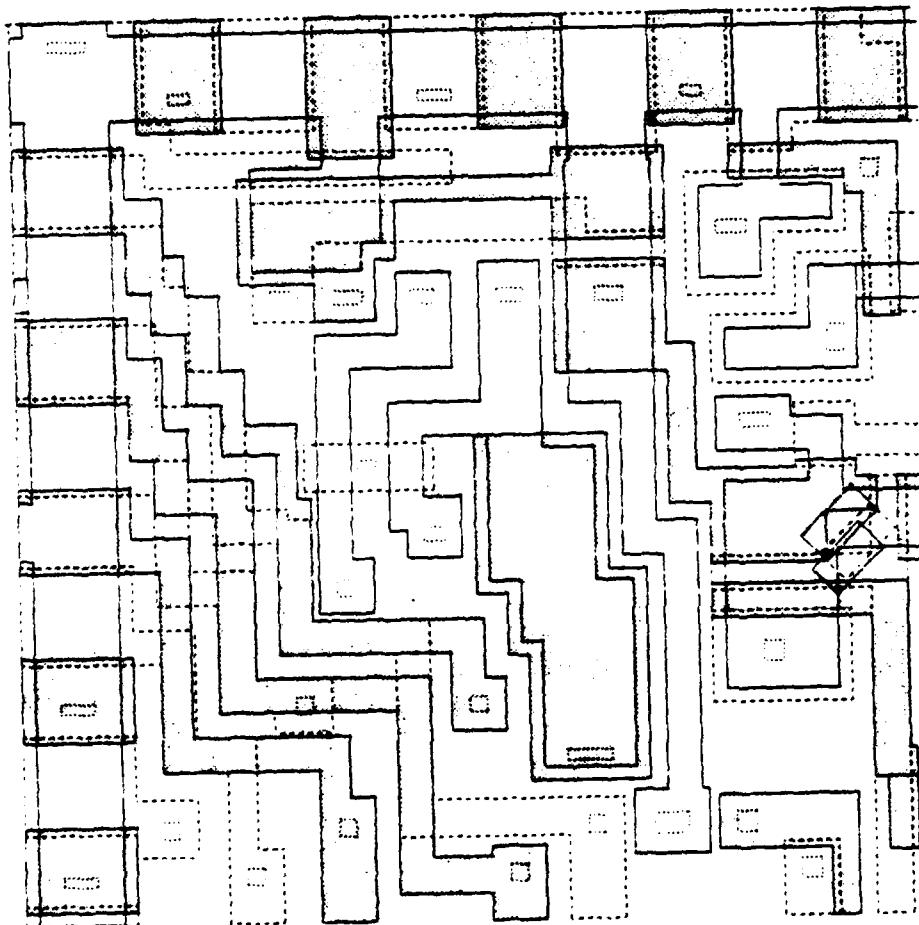
Figure 34. CALMA Plot of 2214 Cell
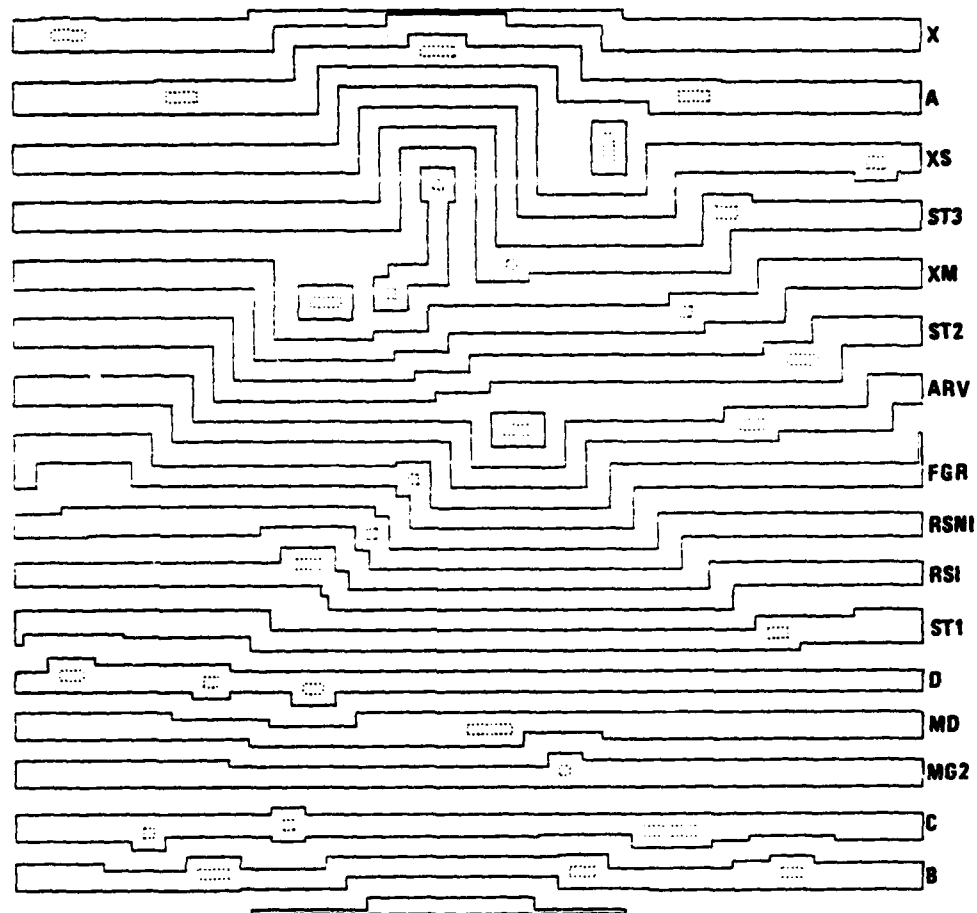
Figure 35. CALMA Plot of Metal Tracks on 3022 Cell
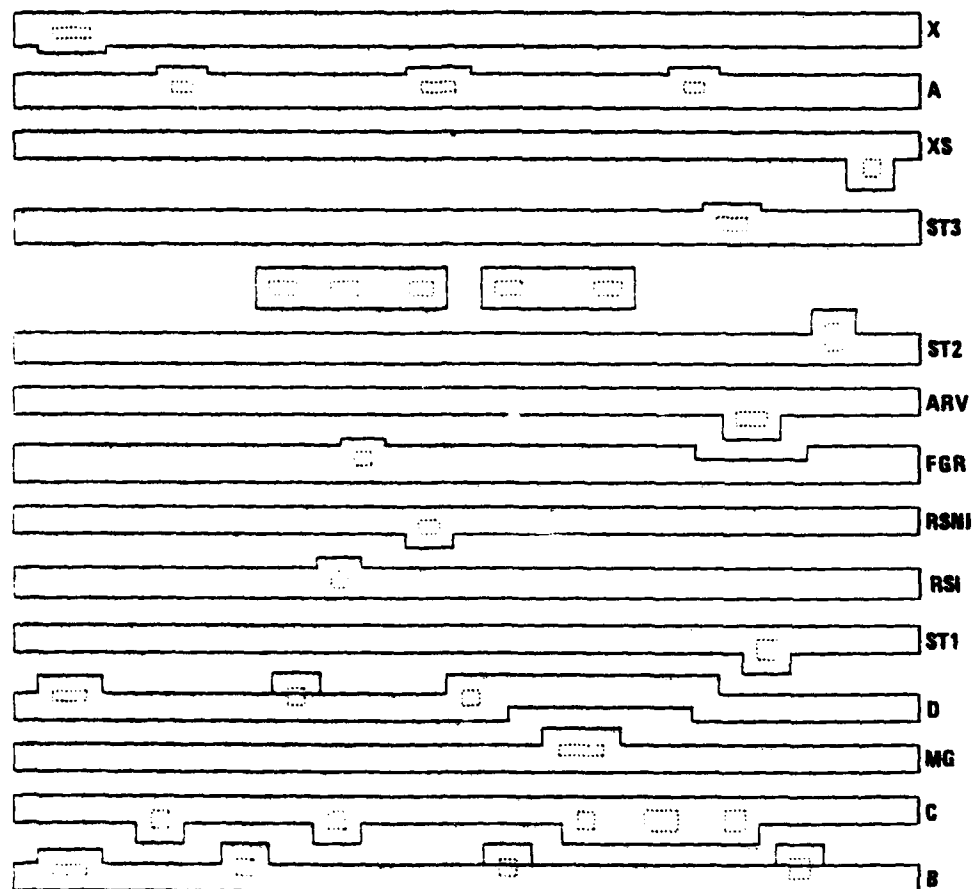
Figure 36. CALMA Plot of Metal Tracks on 2214 Cell

Figure 37 shows the modified (approximately to scale) 40-pin DIP assignments. Also noticeable is a more uniform pad placement, which allows rectangular placement of the die in the DIP cavity. Test FETs are retained to provide V_T uniformity data for comparison with the array regenerator uniformity measurements, but these are not bonded out to the package.

Although not shown in the cell plots, the array is terminated by dummy phase gates and a sink diode at each column/row end. This ensures that attempts to shift leakage/data charge off the array will result in zero charge remaining in those peripheral rows/columns.

The terminations use separate diodes and gates in the same manner as the SINK node in the cell. An extra phase, T, was introduced to perform this.

The sequence of clock states for a Shift Right (SR) is shown diagramatically in Figure 38 and for shift up (SU) in Figure 39.

Improved Functionality

The #3022 cell architecture is essentially the same as the #2214. However, there are two major differences in functionality. The first is the new linear regenerator, which was described above. The second is the extra control clock XM, which connects ST2 and ST3. This allows two new primitive command types: rotate the contents of X, ST2, and ST3 (bidirectional); swap the contents of ST2 and ST3. "Move" primitives (MV) are also introduced to provide the capability of performing memory swaps during computation of a modulus. The updated list of commands is shown in Table 4. This is not an exhaustive list of the Level 1 (primitive) commands for chip #3022. Controller memory allocation has been defined and performed under Task 1 as part of the set-up of the Exerciser unit.
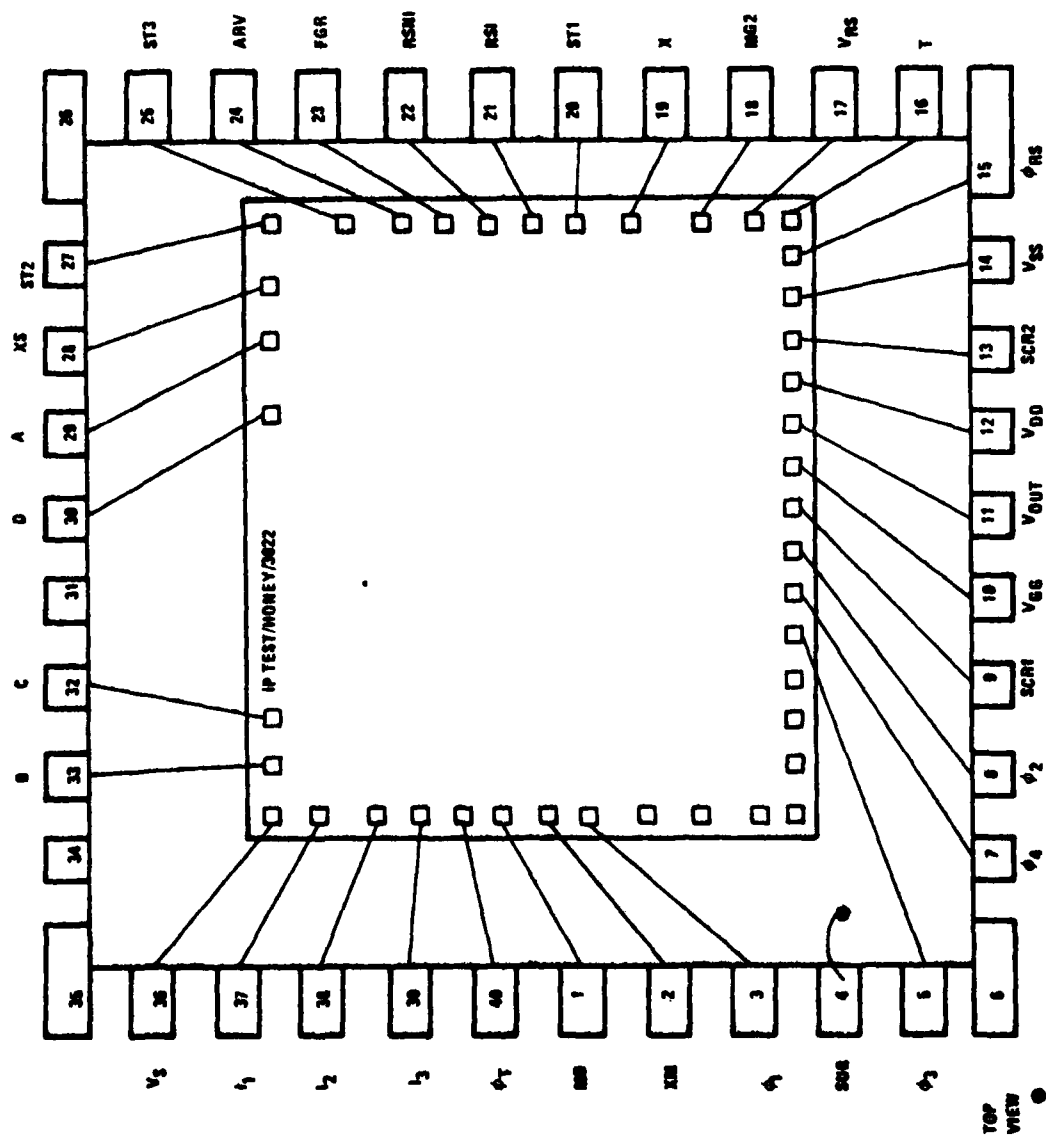
Figure 37. Revised 40-Pin DIP Pin Assignments for #3022 Chip

66
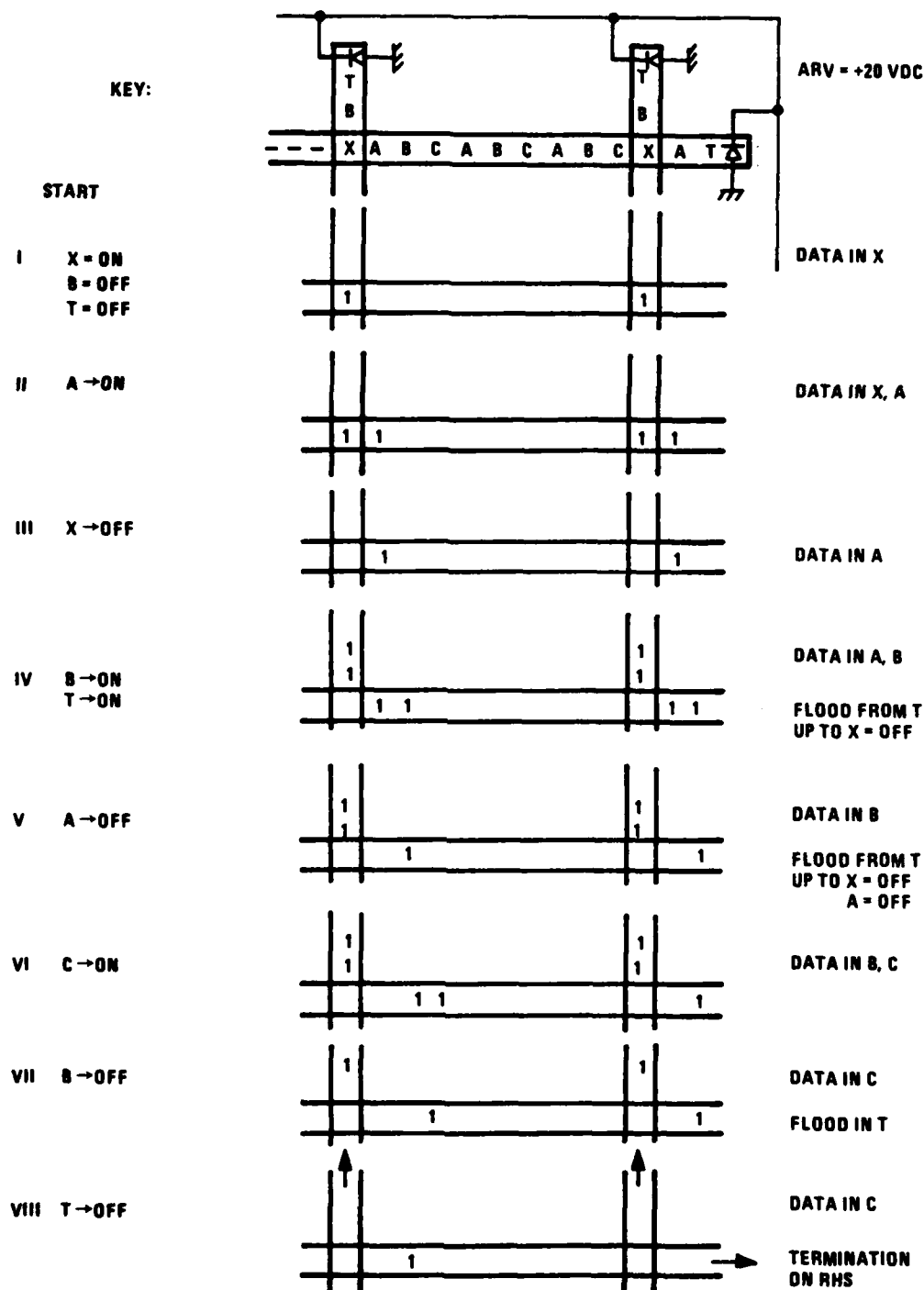
Figure 38.   Shift Right (SR) Clock Sequence with Termination Phase T
as Now Implemented

67

DATA IN X

DATA IN X, B

I X = ON

II B → ON

DATA IN B

DATA IN B

DATA IN B, C

**FLOOD IN T, B
UP TO X = OFF**

**FLOOD IN T, B
UP TO X = OFF**

III X → OFF

IV T → ON

IV C → ON

DATA IN C

FLOOD IN T

TERMINATION

DATA IN C

IV B → OFF

VII T → OFF
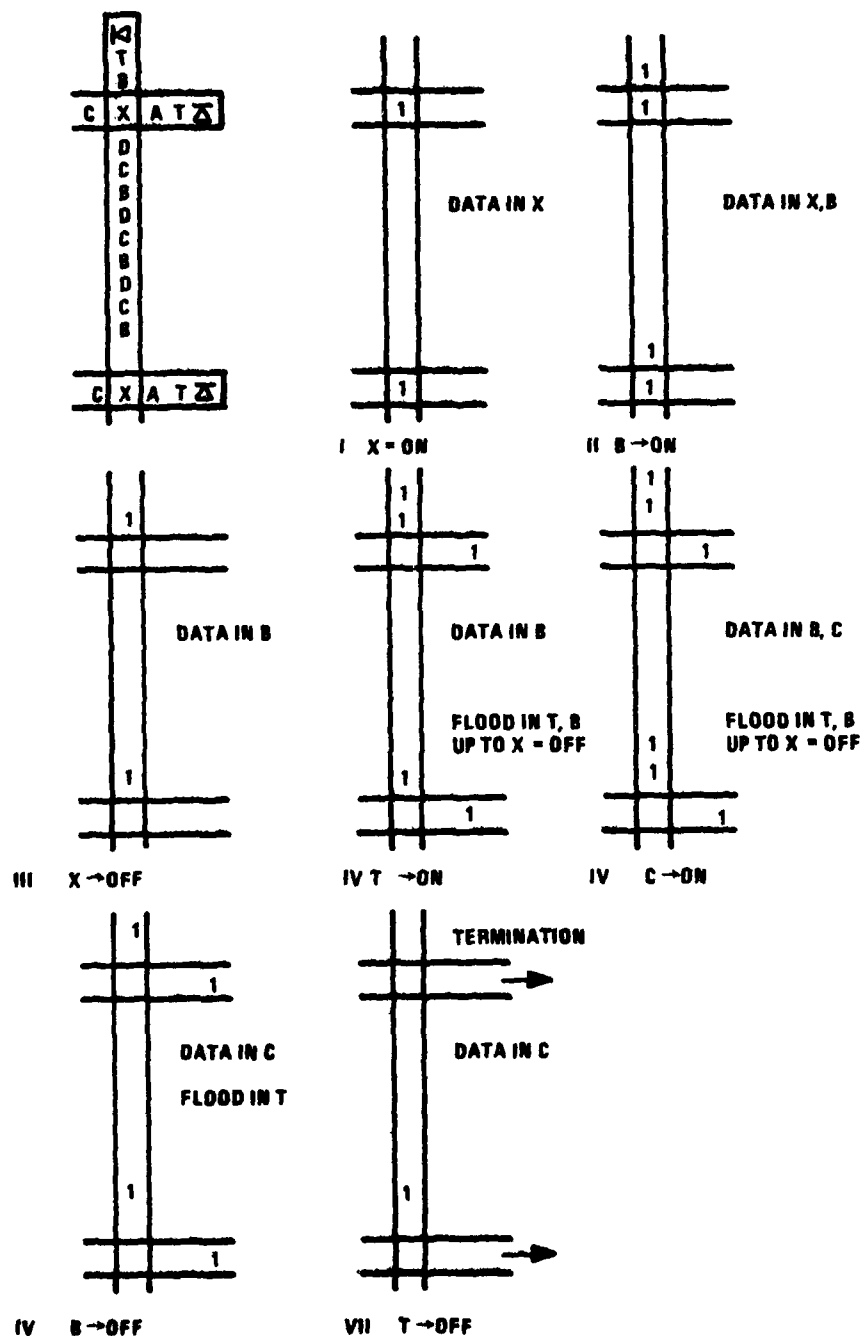
Figure 39.   Shift Up (SU) Clock Sequence with Termination Phase T
as Now Implemented

TABLE 4. CCD MATRIX PROCESSOR PRIMITIVES (CHIP #3022): LEVEL 1 COMMANDS

Function	Mnemonic	Number of States	Initial Conditions	
			Start	End
Load/unload 1 stage (2 bits 4φ, serial register)	IB, OB	17, 17	Yes	Yes
Load entire array	IA	4688	Yes	Yes
Unload entire array	OA	4688	Yes	Yes
Array shift right (xη)	SR	21	Yes	Yes
Array shift left (xη)	SL	21	Yes	Yes
Array shift up (xη)	SU	21	Yes	Yes
Array shift down (xη)	SD	21	Yes	Yes
--All shifts: X goes to X once per pass, η passes called				
--SL, SD: limited η; SR, SU: unlimited η				
Load store #1 from X	L1	18	Yes	Yes
Load store #2 from X	L2	9	Yes	Yes
Load store #3 from X	L3	9	Yes	Yes
Unload store #1 to X	U1	14	Yes	Yes
Unload store #2 to X	U2	7	Yes	Yes
Unload store #3 to X	U3	8	Yes	Yes
--"Load" performs register addition in named store				
--"Unload" performs register addition with X				
Erase store #1	E1	7	Yes	Yes
Erase Store #2	E2	19	Yes	Yes
Erase store #3	E3	55	Yes	Yes
Rotate X23	RF	14	Yes	Yes
Rotate X32	RB	17	Yes	Yes
Swap 32	SW	31	Yes	Yes
Move store #1 to store #2	MV	14	Yes	Yes
Move Y to X	MX	17	No	Yes
Regenerate store #1 to X	R1	23	Yes	Yes
Regenerate store #2 to X	R2	29	Yes	Yes
--original store preserved				
Difference ST#1 - ST#3	D3	29	Yes	No
Difference ST#1 - X	D4	30	Yes	No
Difference ST#1 - ST#2	M2	68	Yes	Yes
--D3, D4: Result goes to ST#3; M2: result goes to X				
--D3, D4: Destroys ST#1; M2: ST#1, ST#2 conserved				

The #3022 architecture, although deficient in storage, can perform a simple edge enhancement algorithm: the so-called Roberts Cross operator. This is a simple modulus sum on a 2 x 2 window as defined by:

$$R = |a - d| + |b - c|$$ (negative results become zero)

The set of primitives necessary to perform this is included in Table 4.

The sequence of major stages is illustrated in Figure 40, and the program is shown below.

START Assume I/O to load array has just occurred, and E1, E2, E3 cleared the stores, leaving X intact.

     I. L1
    II. R1, L3, R1, SU(1), SL(1), MV1-2, L1
   III. D3, generates sum SL(1), ST1 - ST3; ST1:= NULL
    IV. MV Y-X, L1
     V. R2
    VI. SU(1), SL(1), generates "d" element
   VII. D4, generates sum ST1-X; ST1:= NULL
  VIII. MV Y-X ⎤
    IX. L1, E1 ⎦   clears "d" from array
     X. R2, SL(1), L1, R2, SU(1), generates "b" and "c"
    XI. D4 ⎤
   XII. MV Y-X, L1, R2 ⎬
  XIII. SL(1) ⎭ regenerates "b" for last difference
   XIV. D4, MV Y-X, L1, E1, generates R

This routine has yet to be executed since only fully functional #3022 chips can perform the new commands. XM is used to provide temporary access to ST3 while charge is shifted along the registers.

THE WINDOW IS 2 x 2 USING THE SET
{a, b, c, d} WITH a,b AS THE TOP ROW
OF THE MATRIX.

DEFINED QUANTITIES:

$\triangle 1 = (Ka \cdot Kd)$
$\triangle 2 = (Kd \cdot Ka)$
$\Sigma 1 = \triangle 1 + \triangle 2 = K(1a \cdot d1)$
$\triangle 3 = (Kb \cdot Kc)$
$\triangle 4 = (Kc \cdot Kb)$
$R = \triangle 1 + \triangle 2 + \triangle 3 + \triangle 4$
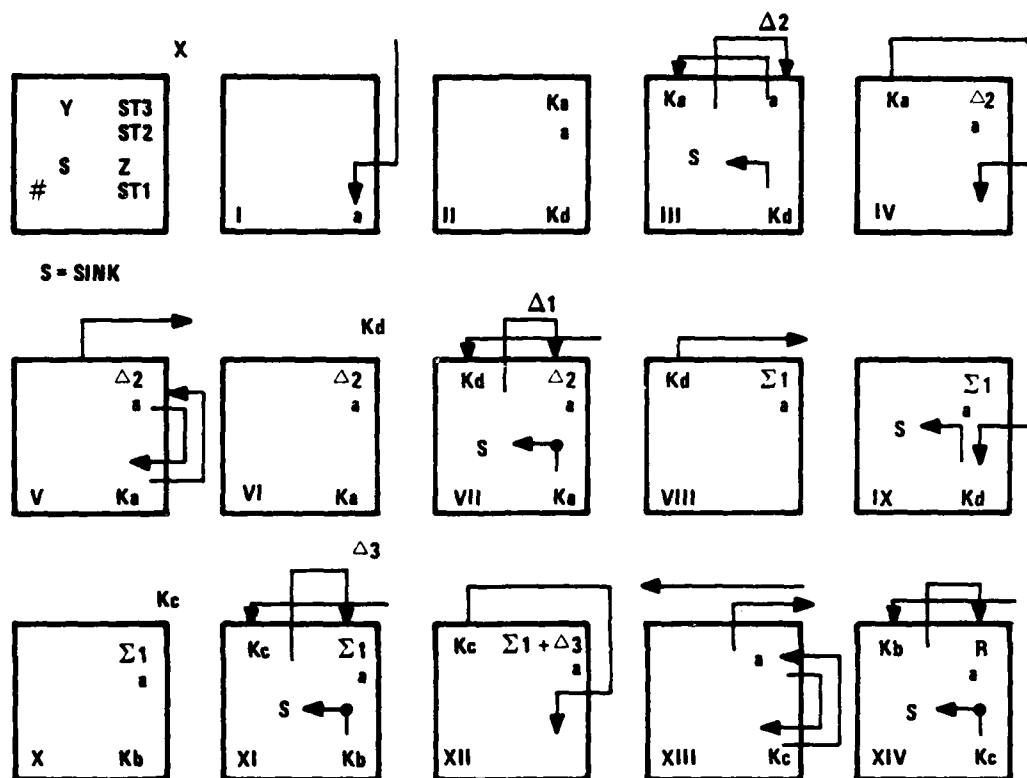
Figure 40.   Sequence of Primitives for #3022 to Generate Roberts Cross,
             R, and Save Original Image, a, for Further Processing

71

When using the #3022 chip with the correct termination, T, the extra clock drive must be coded as appropriate. This has been performed also as a part of Task 1. A photograph of the entire chip is shown in Figure 41.

Processing runs (3022-1, 3022-2, 3022-3) were executed through November 1981 and resulted in discovery of some mask errors missed on the redesign. This resulted in failure of the first and second wafer runs (12 wafers each). The third process run was successfully completed and detailed measurements made at the probe card level to select parts for packaging.

Since the first two wafer runs were lost, two further wafer runs were started (3022-4, 3022-5) at the end of November 1981. Run 4 was completed and received for testing with some parts selected for packaging. At the time of writing Run 5 remains to be tested.

Packaged parts from both Run 3 and Run 4 have been delivered to RADC with further parts to be sent as they become available. It is expected that quality and performance of later parts will be better than the first samples. Honeywell is anxious to provide as good a selection of parts as possible.

The results of Run 3 parts measurements are described in the next subsection.

TASK 4--FINAL 16 x 16 CCD CHARACTERIZATION

Chip Tests and Wafer Tests of #3022 CCDs

Wafer testing has been performed on four process runs with good parts obtained from the third and fourth runs. Further process runs will be characterized later on Honeywell's internal programs.
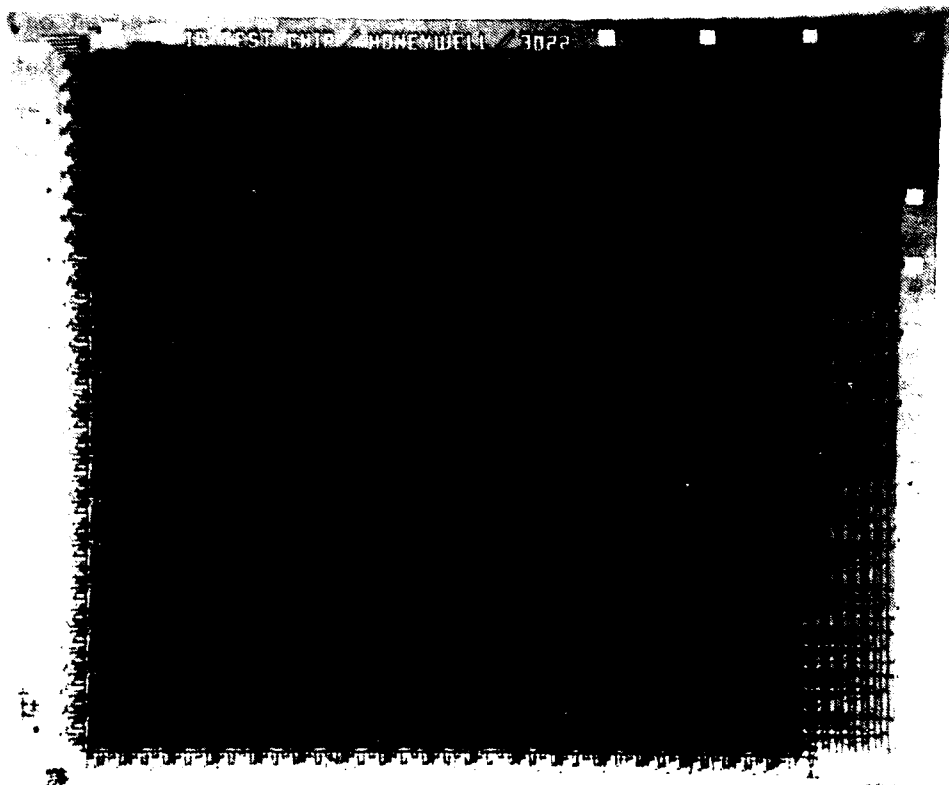
Figure 41. Photograph of Entire 3022 Chip

The results from run 3 wafers, some parts of which have been packaged and delivered to RADC, are shown in Figures 42 through 58. The elementary functions of shifting data through the external four-phase shift register only and then also through the three-phase matrix shift registers were used as the "go/no go" test criteria.

Yields of > 90% for the external four-phase registers were observed on all wafers (these registers have the least complex interconnect). The internal matrix registers showed about 40% yield with the major failure mode apparently due to open-circuit clock phases with a random fault distribution. Of those devices that passed this second level of tests, there were almost 100% passes to the next level of testing. This next level operates not only the external registers and matrix shift registers, but it also loads and unloads each of the stores.

Figures 42 and 43 show array shifting with zero data for two cases: (1) using "Erase" primitives and "Swap 2←→3" primitive; (2) using only input array (IA) and output array (OA) subroutines. The primitives used are shown on the TTY print-out in each figure. These measurements were executed using the RADC Exerciser unit adjusted manually to give a slow frame time of about 47 msec in order to show leakage charge effects on these packaged parts.

Satisfactory performance is evident in these photographs--the leakage charge ramps slightly from right to left per line of 16 bits due to build-up during the IA subroutine (expected and normal), and ramps very slightly from left to right, per whole frame of 256 bits during OA subroutine (expected and normal). It should be noted that a "corner turn" is executed by IA, OA subroutines by choice--a different choice with no corner turn is available if required, but the subroutines have not been included in the EPROM set.

*P
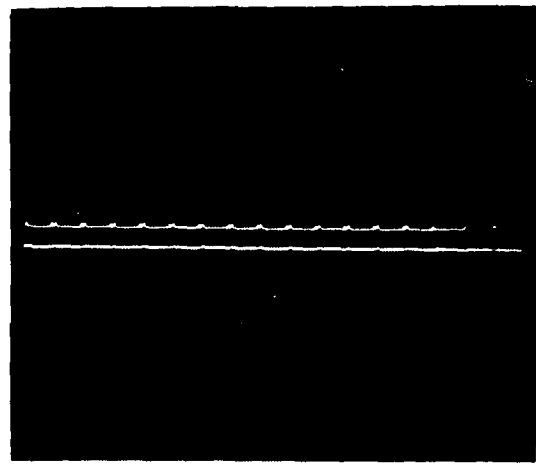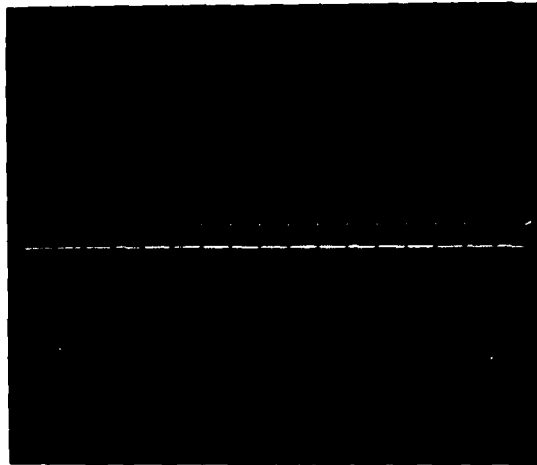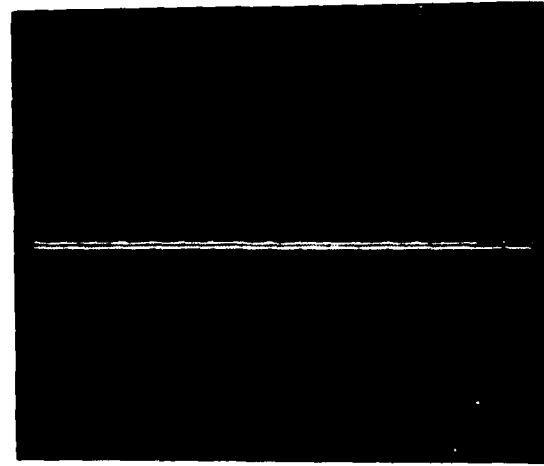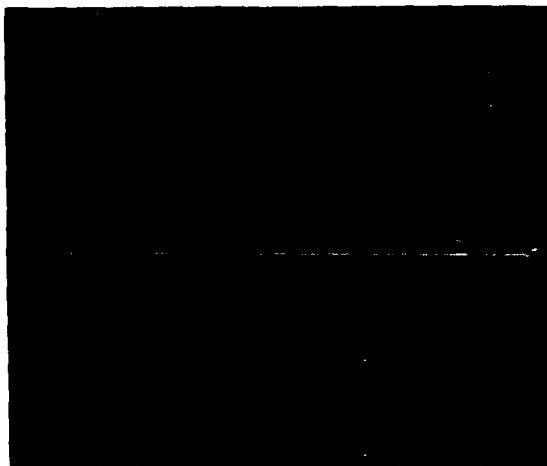
ENTER PRIMITIVES

E1
E2
SW
E2
OA
CO

*P

ENTER PRIMITIVES

IA
OA
CO

Figure 42. Leakage Current with Zero Data (25^OC, 47 msec)

Figure 43. Leakage Current with Zero Data, Without Erasing Stores

The same program was executed but with a 6 x 6 test pattern in Figures 44 and 45. Again, satisfactory performance has been obtained. Note also that transfer efficiency on this device (3022-3-8 chip #64) is good. An estimated $N_\epsilon \sim 0.1$ is evident, where

$$N \sim 1/2(16 \times 8) + 1/2(16 \times 10) + 1/2(16 \times 10)$$

$$+ 1/2(16 \times 8) + 4 = 292 \text{ transfers}$$

This suggests a mean transfer inefficiency of 3×10^{-4} ($\alpha = 0.9997$), which is acceptable performance for a BCCD. A slightly better transfer efficiency is expected from the best devices which should be as good as the original "best performance" #2214 parts used earlier.

Loading and unloading the three stores is shown in Figures 46 through 48. The "swap memory" routine SW was tested in Figure 49, with partial success. The cause for the degradation in the case where L2 is executed followed by SW and U3 is as yet unexplained. A software error may be present: it is suspected because the other choice (L3, SW, U2) is quite satisfactory. There is a significant difference in charge flow with the L2, SW, U3 choice requiring all controls to be correct, whereas L3, SW, U2 is a simpler use. This is still under investigation with more testing of run 4 parts.

Leakage charge collected by the stores during IA + OA frame time is unloaded and read out for the photographs in Figures 50 through 52. There is no major difference between the three stores and an acceptably low leakage is attained: about 10% full well in the 47 msec IA + OA frame time. Evidently the signature of ST1 on #2214 chips has been successfully eliminated on the #3022 chip.

Figure 44. Performance with Data
(70% full well, 0%
fat zero) Showing
Transfer Efficiency

Figure 45. Performance with Data,
Without Erasing Stores
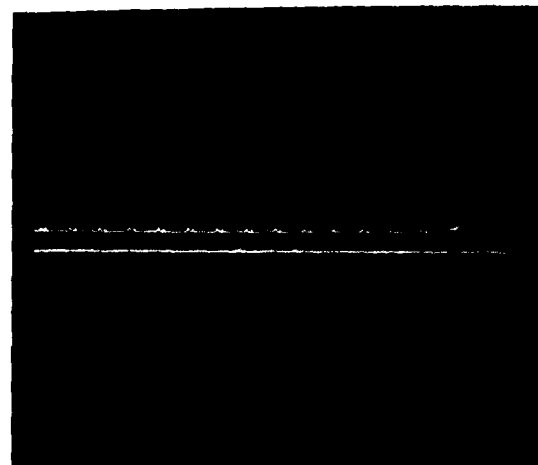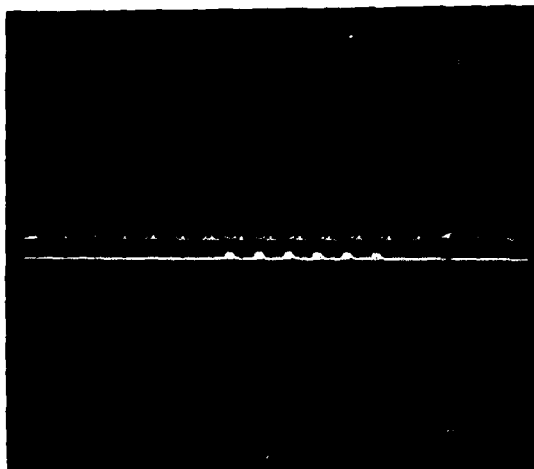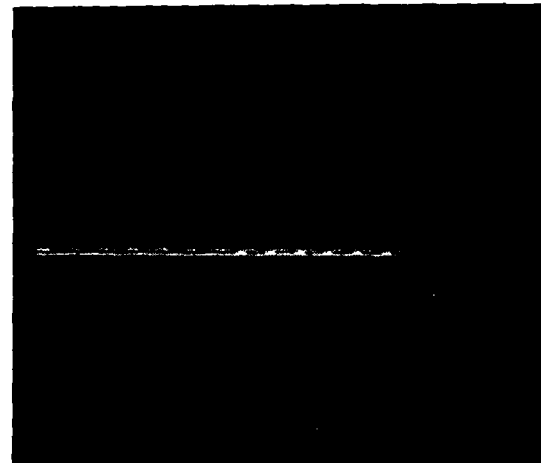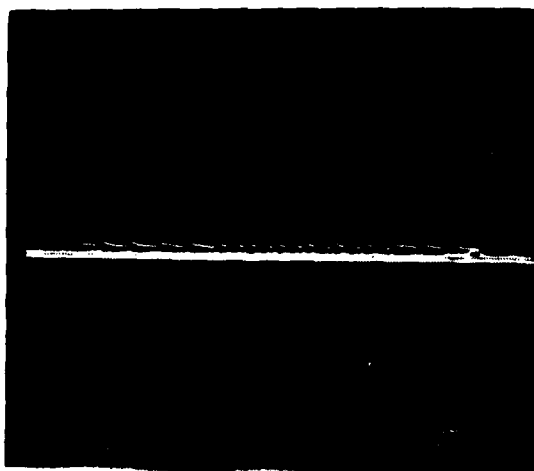
```
*P
              ENTER PRIMITIVES
    IA
    E1
    E2
    SW
    E2
    L1
    U1
    OA
    CO
```

```
*P
              ENTER PRIMITIVES
    IA
    E1
    E2
    SW
    E2
    L2
    U2
    OA
    CO
```

Figure 46.   Loading and Unloading ST1   Figure 47.   Loading and Unloading ST2
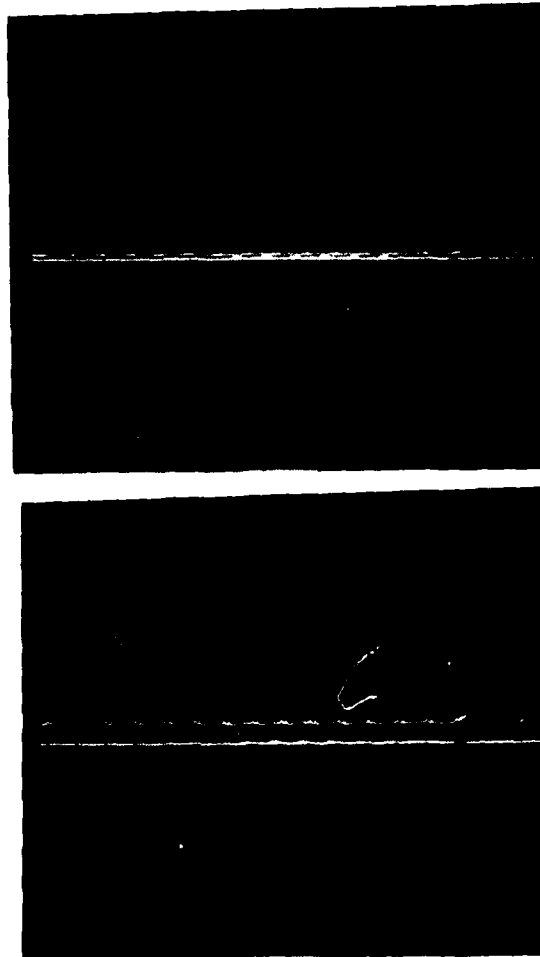
```
*P
        ENTER PRIMITIVES
IA
E1
E2
SW
E2
L3
U3
OA
CO
```

Figure 48. Loading and Unloading ST3
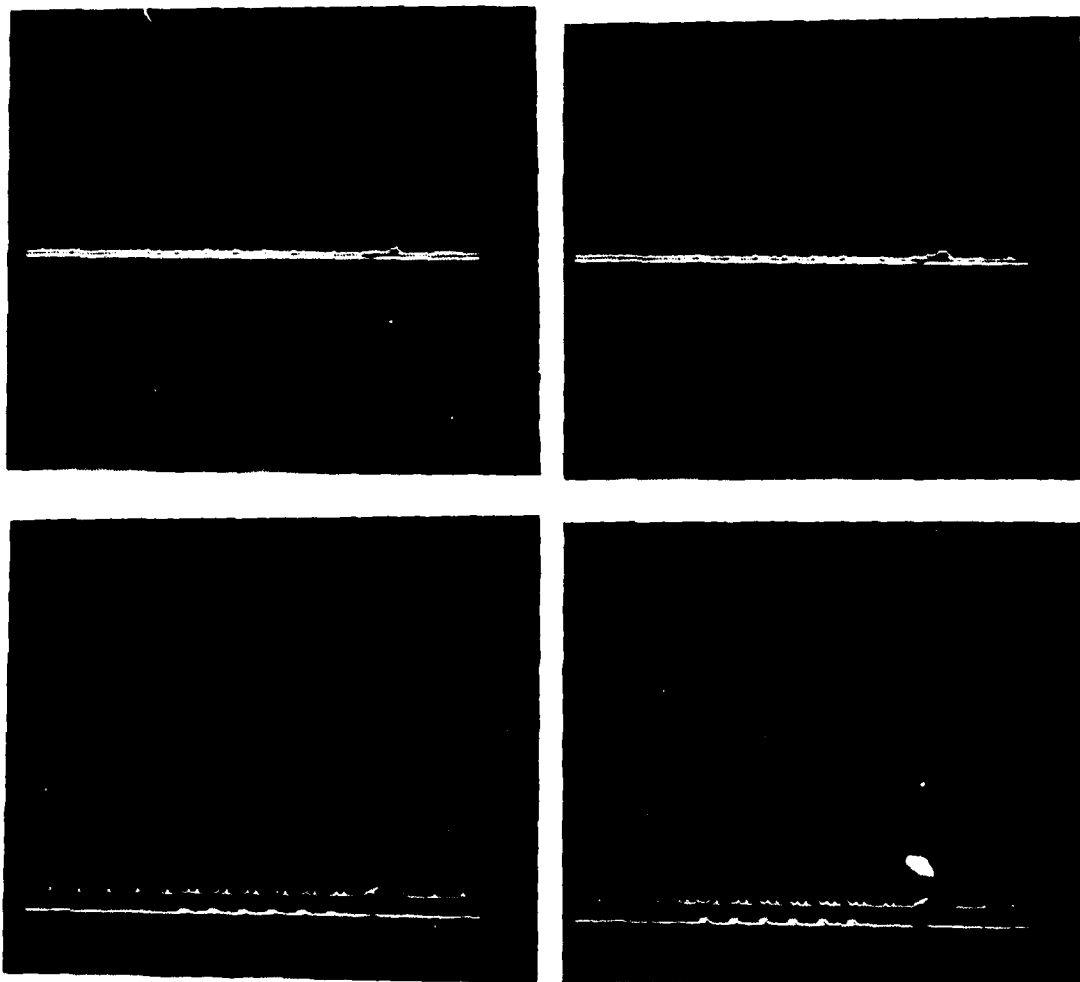
79

Figure 49. Swapping Data (ST2, ST3); Software Error Suspected
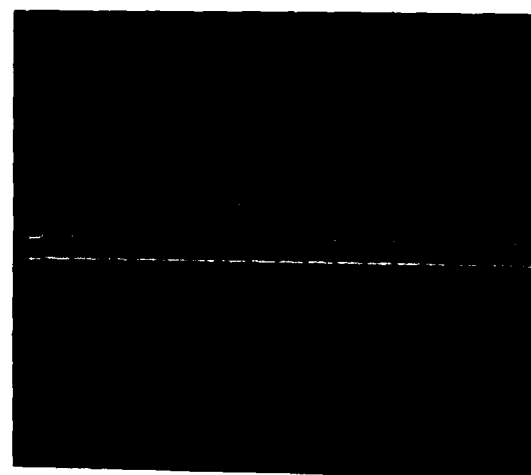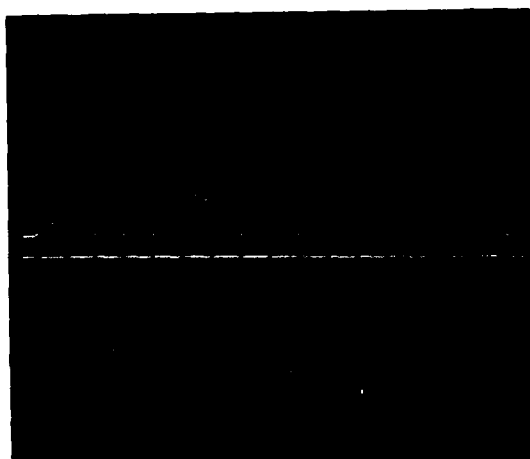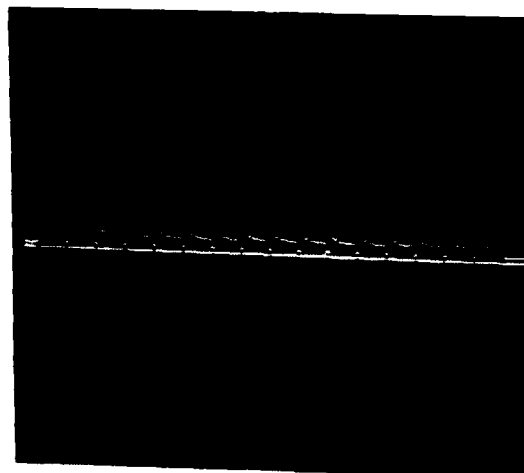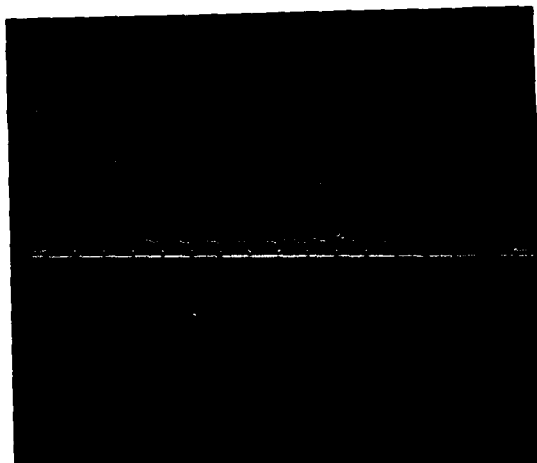
```
*p
        ENTER PRIMITIVES                    *p
U1                                                  ENTER PRIMITIVES
0A                                          U2
CO                                          0A
                                            CO
```

Figure 50.   Leakage from ST1   Figure 51.   Leakage from ST2
             (25°C, 47 msec)   (25°C, 47 msec)

81

*P
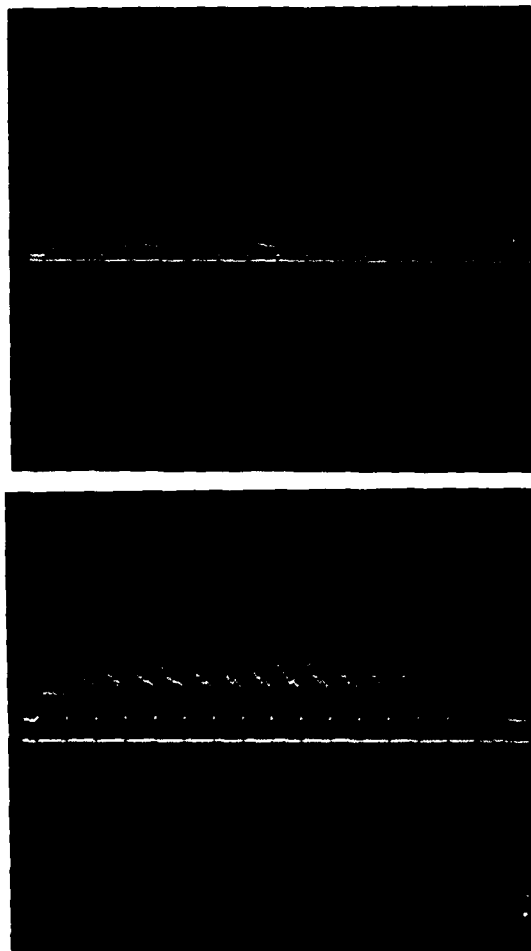
ENTER PRIMITIVES
U3
0A
CO

Figure 52. Leakage from ST3 (25°C, 47 msec)

The most crucial test is that of using the FGA as a regenerator. The output from the array after R1 has been executed is shown in Figure 53. The Exerciser was set to operate the CCD at a faster rate to reduce leakage charge effects. As a calibration, the subroutine was extended to store this regenerated output in ST2 and then unload ST1. This then shows the charge that was loaded into the FGA input Z. Comparing the two outputs in detail (see Figures 53 and 54), it can be seen that:

1. Relatively good uniformity is obtained apart from a baseline offset nonuniformity.

2. The charge-charge unity gain is approximately attained, as desired.

3. There is no discernible 1/f noise-smearing of the output pulses.

This is a rather pleasing result that validates the design modifications introduced to the 3022 chip. A further test on the regenerator, using the R1 primitive again, is shown in Figure 55. The 6 x 6 test pattern has been set (by TTY command *E and data location/value) to give a linear "staircase" in place of the simple "box" of unit amplitude used previously. The data is read out in the same manner as just described.

Notice that

4. A linear response has been obtained up to a limiting amplitude--offset effects should be considered separately.

This validates the other design changes to the FGA and is as computed from the model given in Appendix A.

The multiplier structure is the remaining feature to be tested. The operation of the multiplier (see Figure 56) is briefly as follows. The FGA control, RSNI, is assumed to have been at OFF and then switched ON ready to "float" the Z and Y gates by switching off FGR. At this point, MG2 must be set to give the desired loading factor on the FGA input, Z.
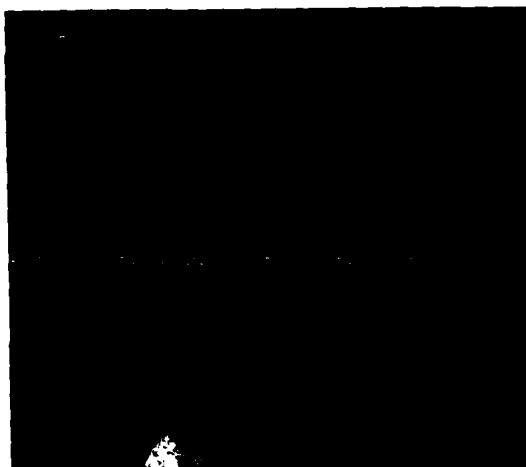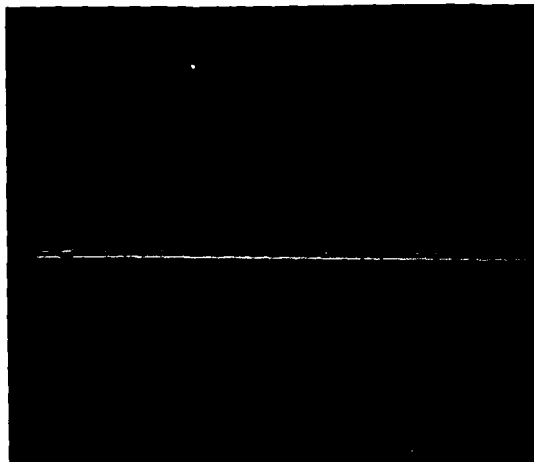
IA	SHOWING	IA
E1	NOMINAL	E1
E2	UNITY	E2
SW	GAIN	SW
E2		E2
L1		L1
R1		R1
OA		L2
CO		U1
		OA
		CO

Figure 53. Regenerator Performance —Output Charge   Figure 54. Regenerator Performance —Input Charge

SELECT FROM LIST P
E

ELEMENT/SHADE?   662
ELEMENT/SHADE?   674
ELEMENT/SHADE?   686
ELEMENT/SHADE?   698
ELEMENT/SHADE?   6AB
ELEMENT/SHADE?   6BD
ELEMENT/SHADE?   762
ELEMENT/SHADE?   774
ELEMENT/SHADE?   786
ELEMENT/SHADE?   798
ELEMENT/SHADE?   7AB
ELEMENT/SHADE?   7BD
ELEMENT/SHADE?   862
ELEMENT/SHADE?   874
ELEMENT/SHADE?   886
ELEMENT/SHADE?   898
ELEMENT/SHADE?   8AB
ELEMENT/SHADE?   8BD
ELEMENT/SHADE?   962
ELEMENT/SHADE?   974
ELEMENT/SHADE?   986
ELEMENT/SHADE?   998
ELEMENT/SHADE?   9AB
ELEMENT/SHADE?   9BD
ELEMENT/SHADE?   A62
ELEMENT/SHADE?   A74
ELEMENT/SHADE?   A86
ELEMENT/SHADE?   A98
ELEMENT/SHADE?   AAB
ELEMENT/SHADE?   ABD
ELEMENT/SHADE?   B62
ELEMENT/SHADE?   B74
ELEMENT/SHADE?   B86
ELEMENT/SHADE?   B98
ELEMENT/SHADE?   BAB
ELEMENT/SHADE?   BBD
ELEMENT/SHADE?

ENTER PRIMITIVES
IA
E1
E2
SW
E2
L1
R1
L2
R1
L2
U2
OA
CO

Figure 55.   Regenerator Performance Showing Linearity

85
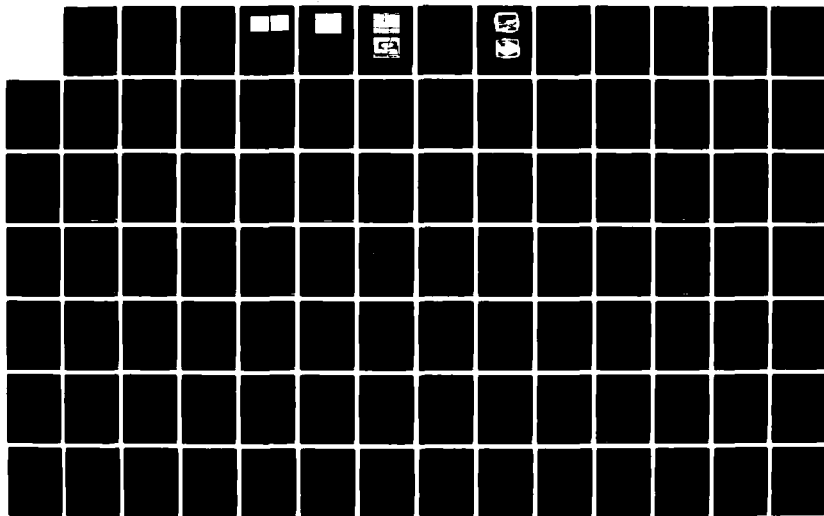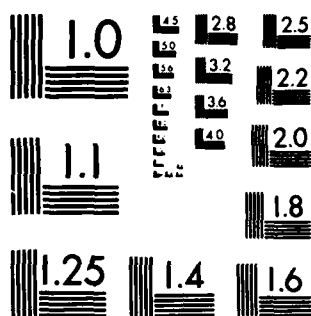
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A
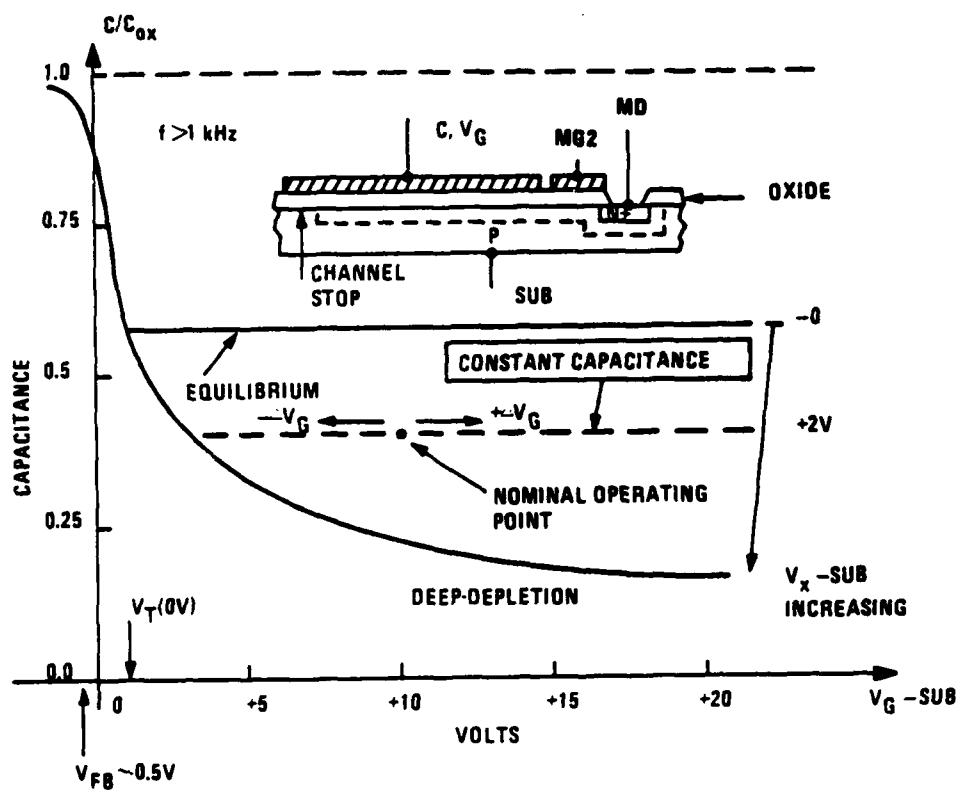
Figure 56. Physical Basis for Multiplier Gain Control (Patent applied for: Honeywell #a4109674-US)

Having set MG2 to an analog bias, MD is then switched from its default OFF (0V) to its active ON (+15V) state. This allows excess channel charge from under MG2 to flow out to the MD diode sink. The surface potential under the multiplier gate (the large Poly I gate visible in Figure 34) is thereby set to the desired value and the Z loading factor is fixed. FGR can be switched OFF next and then, in a subsequent clock state, signal charge can be loaded into the Z floating gate as usual.

We have tested the Rl primitive with MD inactive at default = +3.5V and MG2 adjusted to give a "low" gain and then a "high" gain manually to observe operation.

5. The gain control is completely effective.

Figures 57 and 58 show the output from an Ll Rl subroutine with the DC setting of MG2 and MD shown as the lower two oscilloscope traces. MG2 is set high in Figure 57 and low in Figure 58.

Further testing of this primary function is in progress to observe dynamic gain setting. No data is available at the time of writing. When data is available, however, the results will be sent to RADC through the present contract monitor (Mr. Paul Pellegrini).

The measurements show that all the basic components of the 3022 cell function as designed. It remains to be seen, however, where software, drive waveform, and fabrication defects limit the performance of the 3022 chips in executing a complete threshold and edge extract algorithm. SEM photographs of the 3022 cell and a couple of the contact areas are shown in Figures 59 and 60. The diagonal edge of the Z FGA input gate is visible at bottom right in Figure 59. Figure 61 shows the connection of RSNI reset transistor source diffusion to the Poly I gate, which is the multiplier-to-FGA noninverting input gate connection. Figure 60 shows the metal track for the ST3 bus directly above the RSNI contact on the
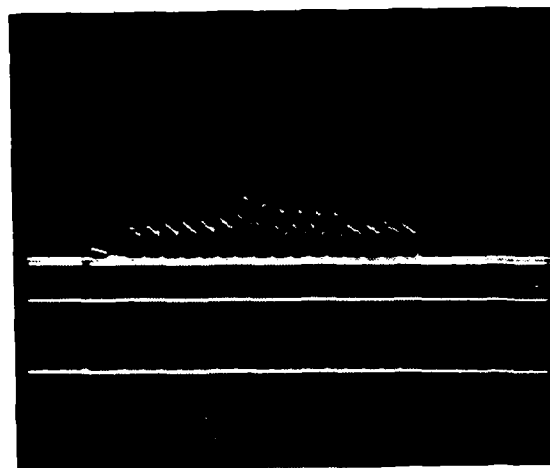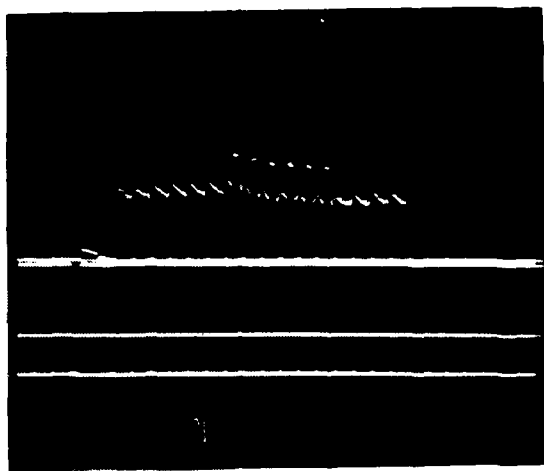
87

```
                        *P
                              ENTER PRIMITIVES
                        IA
                        E1
                        E2                              LOW GAIN
                        SW                              *
    HIGH GAIN           E2
    (UNITY)             L1                              LOW GAIN
    MG2 HIGH            R1                              (x.5)
                        OA                              MG2 LOW
                        CO
```

TOP TRACE = V OUT   ZERO = CENTER OF PICTURE
  MIDDLE TRACE = MG2   ZERO = BOTTOM OF PICTURE
   BOTTOM TRACE = MD (INACTIVE)

Figure 57. Multiplier Performance   Figure 58. Multiplier Performance
            (MG2 High)   (MG2 Low)
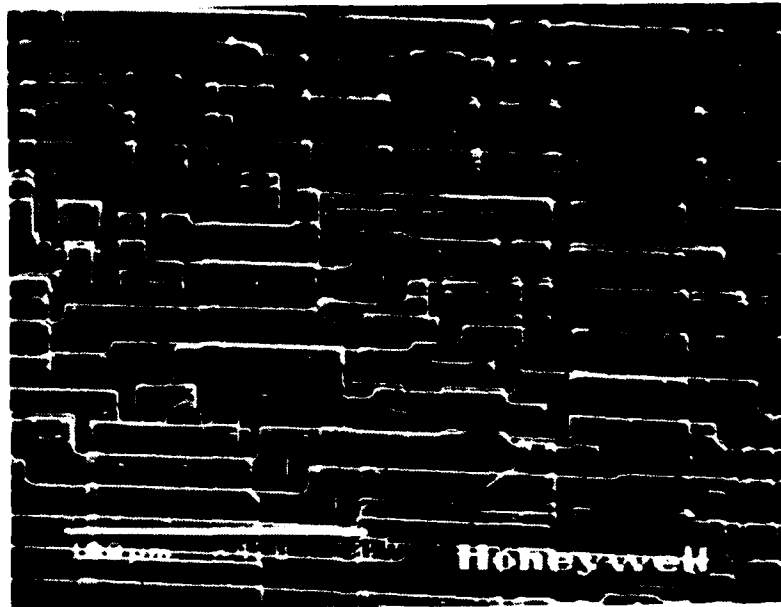
Figure 59. SEM Photograph of 3022 Cell

third track above it. Notice that step coverage and alignment are
excellent. A "spike inhibit" undoped poly layer is applied after contact
windows are etched and before metal is applied (Cu-doped Al). It is
possible that the low temperature (450°C) sinter for making the ohmic
contacts to the gates is not completely effective and that this may be
the cause of the low yield of functional chips. This is under
investigation.

System Tests with Original #2214 CCDs

During the course of the RADC contract execution a related contract from
Eglin AFB, #F08635-80-C-0246 to Honeywell's Avionics Division, was
provided with samples of Honeywell's #2214 CCDs together with operation
software for all primitives plus simple subroutines. The subroutines
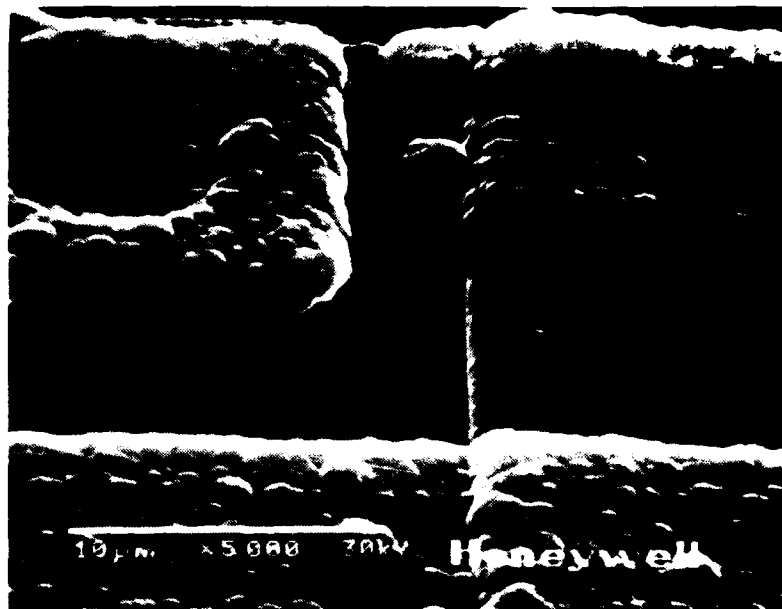
Figure 60. ST3 Metal Bus and Contact to Cross-Over Screen on 3022 Chip
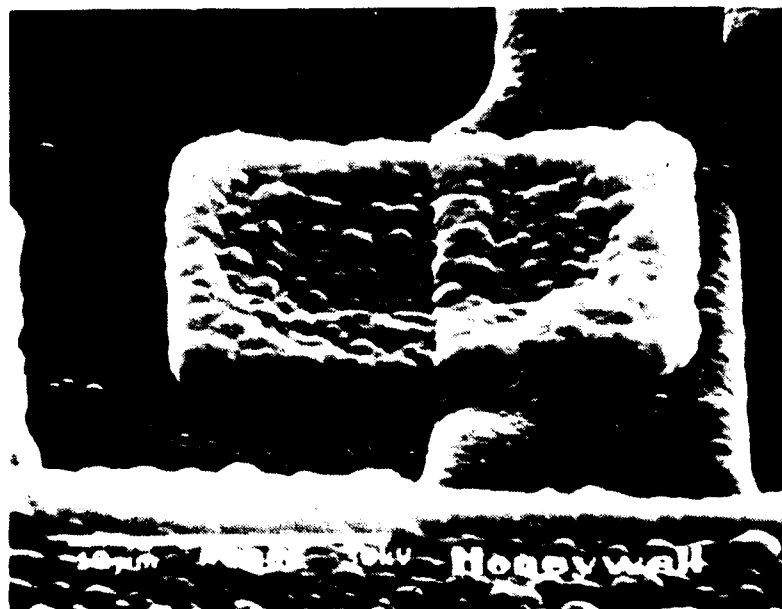
Figure 61. RSNI to Poly I Contact on 3022 Chip

90

were developed partly on Honeywell funding and partly on this present
RADC contract. At the direction of RADC (Dr. Freeman Shepherd) the
necessary information to operate the devices was supplied to the Eglin
contract to coordinate the two programs and enhance the yield of
results. There was no implication that fully functional parts were
available, however. It was already known that significant deficiencies,
which were being corrected in the present development program, existed in
the #2214 chip design.

A somewhat impressive image processing performance was nonetheless
attained by approximately "tuning" the 2214 drive waveforms. The
limiting factors on the chip were as already described in detail in Task
2. By using the Eglin DCIU controller, however, it became possible to
mask out border degradation from the resulting image and also to limit
the dynamic range of the final display to mask the nonuniformities
generated by the 2214 chip defects. The result on the simple threshold
and partial edge algorithms is shown using FLIR imagery (source:
NV&EOL/Honeywell-IRL frame #01926) in Figures 62 and 63. A complete set
of photographs has been supplied to RADC showing various test conditions.

In Figure 62 the Tracking Gate mode has been invoked and a selected
target located manually through the TTY. In Figure 63 the upper left
quadrant (#0) displays a "thresholded" version of the target image using
the subroutine:

        IA El E2 E3 L1 M2 SL L1 M2 SL L1 U1

This enhanced contrast image (stored in RAM) is then further operated on by
the 2214 chip to extract the bottom/right edges shown in the upper right
quadrant (#1) using the subroutine:

        IA El E2 E3 L1 R1 L3 R1 L3 U1 SU SL L1 R1 L2 R1 L2 US El L1 M2

Figure 62. FLIR Image To Be Processed by 2214 Chip
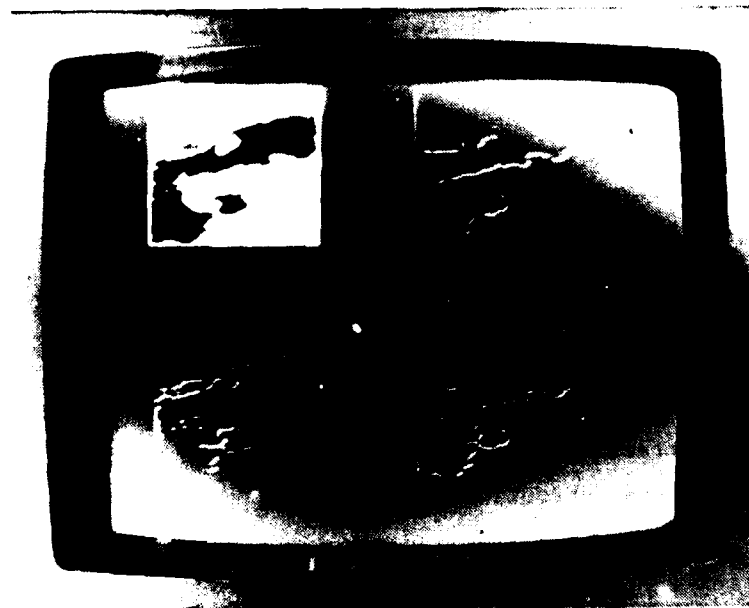
Figure 63. Processed Images Using Threshold and Subtract Subroutines

Similarly, the upper/left edges are extracted and displayed in the bottom left quadrant (#2) using the subroutine:

IA El E2 E3 Ll Rl L3 Rl L3 Ul SD SR Ll Rl L2 Rl L2 U3 El Ll M2

Finally, the #1 and #2 quadrant images are added by the microprocessor in the controller to produce the result shown in #3.

The results are remarkable considering the low performance of the 2214 chip. They suggest that in practice significant capability can be obtained without the requirement for "perfect" CCD performance. The algorithms provide an image processing gain of S/N, which can in some cases tolerate considerable nonidealities.

The performance of a functioning 3022 chip is expected to greatly exceed that of the 2214, however. Not only should the analog arithmetic be dramatically better, but the 3022 will provide in one subroutine the complete threshold and Roberts Cross full edge extractor processed images. The 3022 will provide as output options:

a. Original image only

b. Thresholded image only

c. Edge image (2 x 2 Roberts) only

d. Sum of (a) and (c): edge-enhanced original image

e. Sum of (b) and (c): edge-enhanced thresholded image

And the 3022 will do these entirely without the need to use the microprocessor of the controller in the RADC Exerciser to do partial sums.

The subroutine to perform (c) is:

```
IA  El  E2  SW  E2  Ll
Rl  L3  Rl  SU  SL  MV  Ll
D3
MX  Ll
R2
SU  SL
D4
MX  SD  SD
Ll  El
R2  SL  Ll  R2  Su
D4
R2  SL
D4  MX  Ll  El
```

Roberts Cross, R, placed in position "a" in ST3.

To continue to produce (d) uses:

```
all of the above, plus
U3  Ul
```


This is a full real-world algorithm, and is the maximum window size on which
the 3022 can perform edge extraction. Notice that this is a full set of two
"modulus" operations. The 3 x 3 Sobel Edge is a "better" algorithm but must
wait for the second-generation chip architecture before a demonstration is
possible since this architecture requires a fourth storage site.

TASK 5--SYSTEM ARCHITECTURE AND SOFTWARE DEFINITION

In this subsection we will assess the architectural strengths and weaknesses
of PIP in performing image processing algorithms. An appropriate starting
point is the discussion of the required classes of operations in target

94

acquisition and tracking. (The advantages of a parallel architecture will become readily apparent.) We will then discuss implementation of each of the algorithm classes above in the PIP by using specific examples. We felt that analysis of a system example using a series of algorithms would be an appropriate way to provide the inputs needed for the new cell design in Task 6. Toward that end we picked a chain of algorithms very similar to those used on an existing Honeywell seeker program and obtained a set of specific improvements incorporated in the new cell. Additional input for the new design came from modeling of the existing PIP mnemonic set; that will also be described in this subsection.

In this subsection we also outline the basic form of PIP software and discuss some strategies possible in PIP control.

Target Acquisition and Tracking Algorithms

Acquisition Subsystems--Acquisition encompasses a number of approaches, ranging from simple matched filters to advanced target screening/cueing subsystems. The simple algorithms closely parallel the tracking algorithms themselves (like the matched filters), and are effective only in special circumstances that still require man-in-the-loop interaction and are ineffective in high clutter environments. These are typical of acquisition schemes found in current trackers. They are generally designed to reacquire a target following loss of track, or to lock on a target in the window positioned approximately in the target trajectory by an operator. Automatic target screening/cueing approaches, on the other hand, are more effective for target acquisition, although they were originally developed for operator

cueing. Their performances in high-clutter, low signal/noise environments
are superior to the simpler target acquisition schemes found in today's
trackers. Some of these schemes can also recognize targets; this is an
asset because it enables the prioritization of targets, an essential
function in autonomous fire control.

Automatic target acquisition and recognition architectures and hardware have
been developed by several contractors, including Honeywell (Prototype
Automatic Target Screener built for the Army NV&EOL and the Imaging Sensor
Autoprocessor being designed for AFAL), Rockwell (the ISA above), Hughes
(the Autocuer for the ATAC FLIR), Texas Instruments, and Westinghouse.
Although not all of these are in hardware form, the functional approac
are similar. All employ a sequence of successive subfunctions shown i   ..?e
block diagram of Figure 64.

The video image is first segmented into smaller regions based on intensity
thresholds, edges, or texture. These regions are then associated into
objects or object components. More measurements (features) are computed on
these extracted objects. The features are then used to classify the objects
as targets/nontargets and into a target type. Target recognition has the
added benefit of being able to prioritize the object to be tracked.
Numerous algorithmic variations in the subfunctions are possible, of
course. The algorithms do, however, fall into seven classes:

1. Smoothing Algorithms--Linear boxcar filters, 3 x 3 and 5 x 5
   median filters

2. Edge Algorithms--Sobel, Laplacian, gradient, Roberts Cross,
   directional masks, etc.

3. Thresholding Schemes--Auto threshold, Superslice,
   two-dimensional histograms
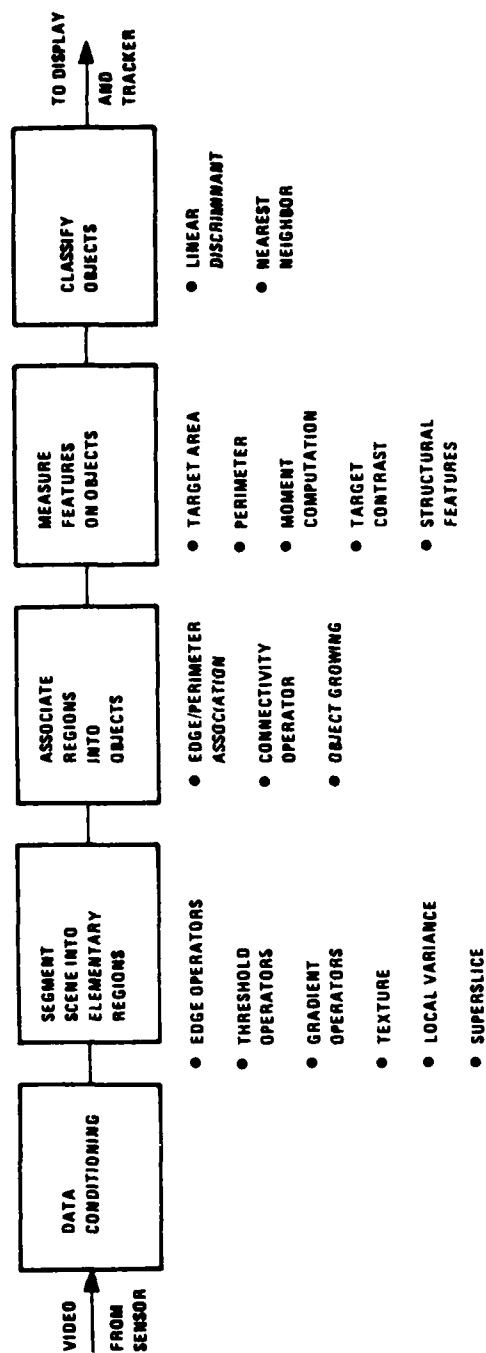
Figure 64. Functional Block Diagram of Generic Automatic Target Screening/Cueing Approaches

4. Association of Classified Pixels—Object growing, connected components, edge/perimeter match

5. Edge Thinning—Thickening, nonmaximum suppression, perimeter extraction, binary noise removal

6. Feature Extraction Functions—Moment computation, perimeter extraction area, texture, average and peak intensity

7. Target Classification Algorithms—Linear discrimination, nearest neighbor classifiers, etc.

In later subsections we will examine specific implementation in PIP of representative algorithms from each class.

Tracking Algorithms—Commonly used tracking algorithms can be categorized as:

o   Centroid
    --Centroid of brightness
    --Centroid of area
    --Area balance

o   Edge and Generalized Edge
    --Adaptive gate
    --Leading and trailing edge averages

o   Correlation
    --Mean absolute difference
    --Product correlator
    --Gradient correlators

o   Feature Tracker
    --Feature matching (Honeywell concept)

o   Matched Filter
    --Fixed filter
    --Adaptive filter

All operations such as convolution, correlation, minimum absolute distance correlation, matched filter, edge operators, etc., fall into the arithmetic category. The basic arithmetic functions are similar to the point operations but involve more variables and include add, subtract, multiply, and absolute value.

Sliding window operations involving binary logic include operations on binary images—usually in acquisition algorithms for connectivity analysis, noise point removal, feature extraction, and perimeter extraction. In these functions, every resultant image point is a fixed boolean function of the original image points in a fixed neighborhood.

Sliding window operations requiring compare (or sort) operations include the median filter, local peak detector to find the maximum of the correlation surface, and nonmaximum suppressor operators for edge thinning.

Thus far we have talked only in general terms about the types of operations required for acquisition and tracking. A system solution, of course, requires implementation of the algorithms. That is the subject of the next subsection.

Processor Architectures for Signal and Image Processing

There are two generic approaches to real-time image processing hardware: serial and parallel. Present approaches are overwhelmingly serial for two reasons. First, sensor output formats have been generally serial; despite the two-dimensional nature of image data, the general approach has been serial image scanning. Even current staring arrays provide a multiplexed, serial video data stream. Second, the vast majority of processing algorithms have been developed and simulated on serial computers. Even parallel operators like M x M sliding windows and convolutions have generally been implemented in serial fashion with M - 1 FIFOs and $(M - 1)^2$

single-pixel delays. The weighted sum of all M^2 pixels in the window must be computed at video rates. This is often a difficult, power-consuming task.

Serial architectures have several drawbacks; they are long on hardware and short on speed and flexibility. Even hardwired serial modules often cannot operate at 875-line video rates and when algorithms change, either the entire module structure must be changed or a new module developed.

The desired architecture for image processing must be flexible and yet handle the high throughputs needed for image processing. Parallel processing, however, can handle high throughputs even when using low-power, slower technology. A dramatic example is the human visual system, which does parallel processing orders of magnitude faster than the fastest serial computer despite a basic processing element (the cortex neuron) much slower than standard TTL.

In addition, many of the image operations we have discussed are local and parallel in nature; they can be decomposed into simple sequences of fast parallel primitives, as we shall shortly see. Unfortunately, digital implementations of fully parallel approaches have not yet been successful. The PIP architecture, however, is both flexible and fast, can potentially be mated directly with analog sensors, is compact, and consumes very little power. The features of the PIP chip were described earlier, but we will now discuss them from an architectural perspective.

An interesting point to note is that parallelism has long been associated with only the "high-throughput" functions in Figure 64--data conditioning and segmentation. We can, however, also cast other functions in parallel form.

For example, let us examine the processing sequence for a perimeter extraction. We have an object consisting of binary "ones" and wish to eliminate superfluous "ones" to form just a boundary.

100

B	C	D
E	F	G
H	J	K

A simple operation for doing this is merely:

$$F = F \cdot \overline{(B \cdot C \cdot D \cdot G \cdot K \cdot J \cdot H \cdot E)}$$

The parallel processor does this as follows:

Operation	Notes
1. SD (1)	Bring C over the center (m,n)
2. A → A	A = 1
3. A · I → A	A = C
4. SL (1)	Bring D over the center
5. A · I → A	A = C · D
.	
.	
.	
17. A · I → A	A = B · C · D · G · K · J · H · E
18. $\overline{A}$ → A	A = $\overline{B \cdot C \cdot D \cdot G \cdot K \cdot J \cdot H \cdot E}$
19. SR (1)	Bring F over (m,n)
20. A · I → A	

We require only 20 operations.

Matrix Processor Control

The rationale for parallel processing and the type of instructions our CCD matrix processor can execute have been summarized. Use of the device, however, requires that we discuss control. In this subsection we will outline the control philosophy of the processor and provide details on the specific set of commands that have been used throughout this program.

The CCD matrix processor operates from a number of control lines, as do all CCDs. These control lines provide either DC levels or clocking waveforms that generally assume a high or low state. (A few lines could require a continuum of signals; these will be treated shortly.) In the case of the 2214 we must provide 21 clocked waveforms. One set of possible values of these clocks can be considered a clock state; we can consider each state as a digital word in which each bit is the high or low state of a particular clock line. Control of the matrix processor is then effected by a series of digital words that specify system states; that is, a finite-state machine. (The digital words must be shifted and buffered for compatibility with the CCD, but that is of no consequence from a control standpoint.) Specifying each control state is a time-consuming task, since detailed manipulation of the charge packets requires device knowledge.

We can, however, concatenate these digital words into short routines that are used repeatedly and are themselves concatenated to form algorithms. This first level of abstraction will be called a "primitive." (As an example, we now can request a "store 1" command, or ST1, which takes data stored under the X node and transfers that charge packet for storage under ST1.) We have now freed the software developer from one responsibility. He need only know the arithmetic capability of the cell, not _how_ the cell does the desired operation. Coding of the processor can be considerably more rapid. A higher level of abstraction would refer to blocks of matrix processor primitives as macros; an example of this is given later in this subsection.

In choosing a primitive set for the 2214 chip, we considered a number of factors. The basic tradeoff is between large memory requirements, a large number of primitives, and high speed versus smaller storage requirements, a core subset of primitives, and lower speed. We opted for the smaller set, but this tradeoff should be made in a more systematic fashion when the new cell is fabricated. Table 5 lists all the original matrix processor mnemonics and their functions; several have been and are being added to this list. Appendix C provides a complete listing of all clock states for each of these mnemonics.

Before proceeding to an example in the next subsection, we will discuss some additional control issues that are not yet resolved. All of the current primitives require that the start and end of each primitive correspond to the same state; that is, there is no restriction in concatenation of primitives. While this simplifies programming, it may sacrifice speed, because states that are useless in a particular sequence have been added for compatibility among primitives. Alternatives to the current approach might be explored.

A similar issue arises in control of the racetrack storage loop (part of the new cell that will be described later). The simplest control structure would, in effect, label each storage site. (At the beginning and end of any instruction that accessed the loop we would have charge packets in the same relative locations.) This may lose speed, however, since we might again make meaningless shifts just to get to initial and final conditions prior to using the same storage locations again. Worse yet, we would not be able to use the loop in a stack architecture mode by shifting the stack one location at a time either forward or backward. The penalty for such versatility is more complex control logic that must keep track of the storage loop position before each instruction. This issue requires closer examination.

TABLE 5.   2214 MATRIX PROCESSOR MNEMONICS FOR FUNCTION IMPLEMENTATION

Mnemonic	Function	Execution Time (clock periods)
I/O	READ IN, SHIFT, READ OUT	17
SR	SHIFT RIGHT X → X	21
SL	SHIFT LEFT X → X	21
SU	SHIFT UP X → X	21
SD	SHIFT DN X → X	21
L1	LOAD MEMORY 1 X → 1	18
L2	LOAD MEMORY 2 X → 2	9
L3	LOAD MEMORY 3 X → 3	9
R1	REGENERATE $(S_1 \cdot K_1)$ COPY 1 to X	22
R2	REGENERATE $(S_2 \cdot K_1)$ COPY 2 to X	26
U1	UNLOAD MEMORY 1 → X	14
U2	UNLOAD MEMORY 2 → X	7
U3	UNLOAD MEMORY 3 → X	8
E1	ERASE MEMORY 1	7
E2	ERASE MEMORY 2	19
E3	ERASE MEMORY 3	55
M2	SUBTRACT 2 FROM 1	67
MOD	MODULUS $|2 - 1| = (2 - 1) + (1 - 2) \rightarrow 3$	138

A final control issue is again more relevant to the new cell. In the 2214
many clock lines serve multiple functions. This reduces the number of control
lines but also practically eliminates the possibility of overlapping
instructions; that is, starting a second instruction before its predecessor
is complete. In the new matrix processor cell design the control lines are
more independent; this affords the possibility of a control architecture
where two instructions are executed slightly faster than in a conventional
single-stack-pointer architecture. Again, more work is needed here.

PIP Architecture

The 2214 chip is a fully parallel processor. Each arithmetic unit cell, under
external control, performs exactly the same sequence of operations as all
other unit cells. Therefore, we can examine the sequence of operations at a

single location and realize that 256 such sequences are simultaneously occurring. We will refer to quantities like A(i,j) and imply a 16 x 16 array. The cell contains three dedicated storage locations (ST1(i,j), ST2(i,j), and ST3(i,j)) which can also serve as accumulators. We can perform shift operations from cell (i,j) to cell (i+m, j+n); if we define the corner location as X(i,j), this implies that X(i+m, j+n) $\rightarrow$ X(i,j). This notation implies that X(i,j) is the destination register for a PIP result. The heart of the cell is the regenerator whose output is given by:

$$K1Z(i,j) - K1Y(i,j) \rightarrow R(i,j)$$

(Charge can be moved from any site to Z or Y.)

Numerous operations (including subtraction, initialization, multiplication, and absolute value) can be derived from this useful function. The arithmetic capabilities can be summarized as follows (STX(i,j) represents any of the three stores):

- Add $\qquad$ $STX(i,j) + I(i,j) \rightarrow STX(i,j)$

- "Subtract" $\qquad \begin{cases} K_1(Z(i,j) - Y(i,j)) & \text{if } Z - Y > 0 \rightarrow R(i,j) \\ 0 & \text{if } Z - Y < 0 \rightarrow R(i,j) \end{cases}$

- Multiply by a constant $\quad K_1 Z(i,j) \rightarrow R(i,j)$

- Clear $\qquad 0 \rightarrow STX(i,j)$

- Initialization $\qquad \alpha \rightarrow R(i,j)$

- Absolute difference $\qquad X(i,j) - I(i,j) \rightarrow X(i,j)$

- Store $\qquad I(i,j) \rightarrow STX(i,j)$

The modified subtraction performed by PIP is very useful in practice; the algorithm example presented later will explain why. In addition to the arithmetic operations above, certain binary operations can be performed in the cell by changing the usual operation of the regenerator. Binarization of an image, for example, is done as follows:

o   Define binarization as $\begin{cases} 1 \text{ if } ((i,j) > \alpha + X(i,j) \\ 0 \text{ if } I(i,j) \leq \alpha + X(i,j) \end{cases}$

o   Generate the local threshold α as shown in initialization above

o   Perform the single-sided subtraction shown above

o   Perform analog addition between the result of the subtraction and itself N times to increase the gain.

o   Use charge skimming on that result using an appropriate MOS gate within the PIP cell as a reference surface potential and an adjacent MOS gate as the store of the input charge. The excess charge spills over the reference barrier and is either stored in a third gate or sent directly to a sink diode. An external D/A can be used to control the reference level of the chosen clock phase. (The D phase, for example, could be used for this.)

This function greatly enhances the ability of PIP to perform several of the classes of image processing operations mentioned earlier. Certain other operations require AND, OR, and NOT logical operations and/or comparison operations.

Logical operations are somewhat difficult to implement in direct PIP form, although following binarization of two arrays (as indicated above) we could use the regenerator to perform certain logical functions. For example, the XOR function is achieved by matching the gain on the inverting and noninverting inputs and performing the absolute difference operations. The AND function is mechanized by using a store as an accumulator and binarizing the result. Comparison operations of the form:

$$I_1 + I_2 \text{ if } I_2 > I_1$$

can be considered a special case of the threshold operation and are readily done in PIP. Another class of comparison operations (needed for median filtering) takes the form:

$$I_1 \to I_2 \quad \text{if} \quad I_2 > I_1 \text{ and } I_2 < I_3$$

This operation could be designed into a new cell but is not readily implemented in PIP.

Table 6 lists a number of image processing algorithms and summarizes the types of operations required to implement each.

A Matrix Processor Example

To appreciate the actual implementation of algorithms in PIP we have chosen a system example that performs representative algorithms from Table 5. The application is a missile seeker, and PIP performs both target detection and computation of centroid information for tracking.

The sequence of operations is shown in Table 7. The background filter provides 5 x 5 smoothing window with unity weighting. The output response $A(i,j)$ with an input of $I(i,j)$ is:

$$A(i,j) = \sum_{k=-2}^{2} , \sum_{l=-2}^{2} I(i+k, j+l)$$

The subtract and threshold operation that follows this uses the background estimate above to generate a binary map as follows:

$$B(i,j) = \begin{cases} 1 & \text{if } |I(i,j) - B(i,j)| > \alpha \\ 0 & \text{if } |I(i,j) - B(i,j)| \leq \alpha \end{cases}$$

If we look at the operations that follow this, however, we can justify a modified procedure. The PIP provides a single-sided subtraction that may provide equivalent (or even better) results than either straight binarization or pure gray-level processing. Conventional image processing systems have retained two image maps: one binary and one with gray scale. PIP can easily provide the following function:

TABLE 6. PIP FUNCTIONS NEEDED FOR IMPLEMENTING TRACKER AND ACQUISITION ALGORITHMS

Tracker and Acquisition Algorithms	Arithmetic Operations					Logical Operations	Comparison Operations		Shift Operations
	Add/Subtract $A - A \pm I$	Multiply by Constant $A - QA$	Absolute Value $A - \lvert A - I \rvert$	Threshold/ Binarize 0/1	Memory	Binary Operations AND, OR, NOT	If $A < I$ $A \leftarrow I$	If $A < I$, and $I_1 < I_2$, $A - I_1$	
Convolution (M x M)	M^2	M^2			2				M^2
Correlation (M x M)	M^2	M^2			2				M^2
Minimum Absolute Distance Correlation (M x M)			M^2		2				M^2
Sobel Edge 3 x 3	18	18	2	2	2			x	
Gradient (Maryland)	16		2		2				16
Background Estimate (M²)	M^2	M^2			2				M^2
Matched Filter (M x M)	M^2	M^2			2		1		M^2
Texture (Rosenfeld)	M^2		1		2				M^2
Median Filter (5 x 5)					3			25	25
Peak Correlation Detection (M x M)					2		M^2		M^2
Nonmaximum Suppression (3 x 3)					2	9	9		9
Edge Thinning					2	9			9
Binary Noise Filter					2				9
Centroid (M x M)	M^2	M^2			2				M^2
Edge/Perimeter Coincidence						1	1		
Superslice/Maryland				10	2				
Threshold/Binarize				1	2				
Reference Update	1	2			2				

TABLE 7. PROCESSING SEQUENCE/MISSILE SEEKER

```
o    Background average (5 x 5)

o    Subtract and threshold (5 x 5)

o    Template matching

o    X-centroid

o    Y-centroid

o    Sum of absolute differences tracking
```

$$B(i,j) = \begin{cases} |I(i,j) - B(i,j)| - \alpha & \text{if } |I(i,j) - B(i,j)| > \alpha \\ 0 & \text{if } |I(i,j) - B(i,j)| \leq \alpha \end{cases}$$

Note the following ("target" implies here only a "region of interest"):

1. Binarization presumably separates "targets" from "nontargets," but in the steps to follow we would get sharper peaks in the filtering operations with gray-level information.

2. Gray-level processing does not separate targets from nontargets.

3. PIP's subtraction operation both isolates targets and retains gray-level processing capability (albeit with a constant subtracted from the result) for targets.

We would recommend simulation of this concept to verify it, but a binary thresholder may well be unnecessary in this application.

109

The next algorithmic step is template matching. The image $M(i,j)$ generated in the previous step is passed through a two-dimensional FIR filter; that is,

$$C(i,j) = \sum_{k=-2}^{2}, \sum_{l=-2}^{2} \alpha_{kl} M(i+k, j+l)$$

This is followed by two operations that feed a tracking processor. If one or more points exceed an external threshold for a target, we use the M array to do centroiding in the X and Y directions as follows:

$$\bar{i} = \frac{1}{2S} \sum_{i=-2}^{2} \sum_{j=-2}^{2} i\, M(i,j)$$

$$\bar{j} = \frac{1}{2S} \sum_{i=-2}^{2}, \sum_{j=-2}^{2} j\, M(i,j)$$

When implemented in PIP we perform each computation in fully parallel fashion and search the external data stream for a maximum in each case. The final tracking step computes minimum absolute difference correlation.

Let $I_0(i,j)$ be the old frame and $I_N(i,j)$ be the new frame. We must find $D(k,l)$ such that:

$$D_k l = M,n\ (D_{m,n})$$

and $D_{m,n} = \sum_i \sum_j |I_0(i,j) - I_N(i-m,j-n)|$

In other words, we shift the two images successively, subtract, sum over the array, and find the position (k,l) that yields the minimum value of this sum of differences. We are now ready to tackle the PIP implementation of this algorithm sequence.

A variety of ways exist to implement a two-dimensional filter in PIP. The five basic classes are illustrated in Figure 65 along with the numbers of shifts required for each method. The method to use is dependent on M, the number of pixels in the window, but for all practical cases methods (b) and (c) are superior. Note that for separable filters the partial sums can be used in filter calculations at more than one pixel; this is why the expressions seem to give lower numbers than the illustrations would indicate.

To compute the background average (a separable filter) we use technique (e) from Figure 65. The PIP mnemonic sequence would go as follows:

PIP Mnemonic	Function
IA	Load Array
L2	; X(Image In X) $\rightarrow$ REG2
E3	; 0 $\rightarrow$ REG3
REGENERATE (S_2,1)	; Standard Regenerate
SUM	; SUM Macro (See Below)
REPEAT 4 TIMES	; LOOP
SR	; Shift Right
SUM	
END REPEAT	; END OF LOOP
U3	; Horiz. Sum in REG3 $\rightarrow$ X
SUM	; Compute Total Sum
REPEAT 4 TIMES	
SD	; Shift Down
SUM	
END REPEAT	; Sum in REG3
U3	ST3 (averaged image I) $\rightarrow$ X
SU	X up one position
SU	X up two positions
SL	X left one position
SL	X left two positions
L1	X $\rightarrow$ ST1
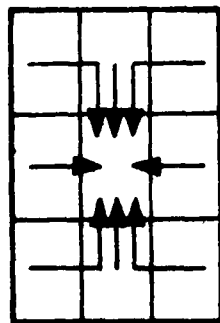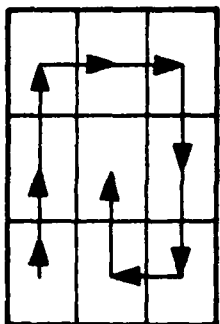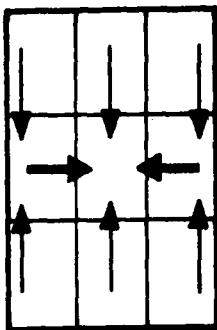
$$\frac{(M\sqrt{M} - 7M + 7\sqrt{M} + 15)}{4\sqrt{M}}$$ SHIFTS PER PIXEL

c. Separable, without propagation of partial sums

$$\frac{M + \sqrt{M} - 1}{\sqrt{M}}$$ SHIFTS PER PIXEL

e. Separable, propagation of partial sums in both dimensions

M SHIFTS

b. Nonseparable, propagation of partial sums

$$\frac{M^2 - 7M + 18}{2}$$ SHIFTS

a. Nonseparable, single sum

$$\frac{11M - 24\sqrt{M} + 45}{8\sqrt{M}}$$ SHIFTS PER PIXEL

d. Separable, propagation of partial sums in one dimension

Figure 65. Five Techniques for Computing Two-Dimensional Filters in PIP

SUM MACRO

Mnemonic	Function
START SUM	; $X \rightarrow REG1$
L1	
DOUBLE REGENERATE $(S_1, .2)$	
L3	; $X + REG3 \rightarrow REG3$
U1	; $REG1 \rightarrow X$
END SUM	

We next perform the subtract and threshold operation. This can be done very simply in PIP as a variation on the absolute difference instruction described earlier. We need only apply a gate voltage to RSI that gives the desired offset from zero. The regenerator function will then clip off the input to the Z gate detector and leave the result. We have designated R1 as "regenerate the contents of ST1 and store the result in X." We can now define "$R1 - \alpha$" as the clipping operation described above. Similarly, M31 indicates (ST3 - ST1), so we will define:

$$M31 - \alpha = (ST3 - ST1) - \alpha$$

(Note that M31 is equivalent to the existing M2 instruction in function.) The sequence for implementation of the subtract and threshold algorithm is given below.

Mnemonic	Function
MOD	$|I - N| \rightarrow ST3$
E1	$0 \rightarrow ST1$
GEN1 α	$\alpha \rightarrow ST1$
M31	$|I - N| - \alpha \rightarrow X$
E1	$0 \rightarrow ST1$
L1	$(|I - N| - \alpha) \rightarrow ST1$

As a starting point for implementation of the remaining algorithms we will review the contents of the cell storage locations. In all cases the subscript indicates frame 1.

$$B_1 = (|I_1 - A_1| - \alpha) \rightarrow ST1, \text{ where } I_i = \text{"raw" image, } A = \text{averaged image}$$

$$I_1 \rightarrow ST2$$

$$|I_1 - A_1| \rightarrow ST3$$

$$0 \rightarrow X$$

We consider each of the first two algorithms in this sequence as separate entities for implementation. To minimize the number of required storage locations, however, we should consider the remaining three algorithms as a group.

The template-matching filter can use the B array to perform filtering. We will assume that the window size for the computation of the template-matching filter is 5 x 5 and that the weighting coefficients are arbitrary. We will also assume a similar size for the centroid computation. Note that 5 x 5 may not be adequate; the actual window size must be larger than the largest target expected. The template-matching operation will compute 256 two-dimensional filtering functions in parallel and ideally will indicate the presence or absence of a target that will then trigger centroiding and tracking. The template-matching filter need only provide an output when one or more cells indicate a computation larger than some threshold. In the existing chip such a determination could be made only after reading out the entire array. A random-access feature is incorporated in the new cell design.

If we unload the entire array and find one template filter point above the target threshold, then we must proceed with centroiding. However, since we need the array in ST1 to do centroiding as well, we may be better off combining some or all of the steps in the two operations.

114

The template-matching operation will be nearly identical to the background averaging operation. We assume that the array to be used is $B_1 = \emptyset I_1 - A_1' - \alpha)$ stored in ST1. The operation to be performed is:

$$C(i,j) = \sum_{k=1}^{5} \sum_{l=1}^{5} \beta_{k,l} B(i-k, j-l)$$

As noted earlier, we cannot assume that the $\beta_{k,l}$ are separable and will not do so here. Table 8 presents the nonseparable implementation in PIP of the above function. We will assume that the $\beta_{k,l}$ are scaled properly to preclude an overflow condition with any of the $C(i,j)$. Note that the SUM macro also includes computation of the centroid and stores the running sums for $\bar{i}$ and $\bar{j}$ in ST4 and ST5, respectively. We require four extra storage registers in this implementation, which minimizes the number of shift operations. In addition, we could perform the minimum absolute difference operation simultaneously; we have included that in Table 8 as well. We have assumed here that the output stream of the template-matching operation is thresholded off-chip (a trivial operation). As noted, the results of window operations will not be centered but instead will be stored in the lower right pixel of the window. Since all window results are to be read out anyway, we do not shift them up and left prior to readout.

Note that minimum absolute difference correlation requires data from a previous frame. We shall assume that the image used for this operation is the image B_i. We must assume that no desired data is destroyed; one of the four extra storage registers required must be used for this.

Although the sequence shown in Table 8 is efficient with regard to minimizing shifting operations, it is worth exploring an alternative which requires fewer storage registers. Table 9 gives the PIP instruction sequence for the same seeker algorithms, but each of the four summing computations required (template match, X-centroid, Y-centroid, and tracking) is done separately. We now require only four storage registers rather than seven.

115

TABLE 8. NONSEPARABLE SEEKER ALGORITHMS, TEMPLATE/CENTROID/TRACKING IMPLEMENTATION

PIP Mnemonic	Function
1. R1 (0.2)	$0.2\ B(i-2,j-2) \rightarrow X$
2. E4	Clear X-centroid register
3. L4	$0.2\ B(i-2,j-2) \rightarrow ST4$
4. R1 (0.2	$0.2\ B(i-2,j-2) \rightarrow X$
5. E5	Clear Y-centroid register
6. L5	$0.2\ B(i-2,j-2) \rightarrow ST5$
7. U6	$B_{m-1}(i,j) \rightarrow X$
8. L2	$B_{m-1}(i,j) \rightarrow ST2$
9. MOD	$\|B_{m-1}(i,j) - B_m(i,j)\| \rightarrow ST3$
10. U3	$\|B_{m-1}(i,j) - \beta_m(i,j)\| \rightarrow X$
11. E7	$0 \rightarrow ST7$
12. L7	$\|B_{m-1}(i,j) - B_m(i,j)\| \rightarrow ST7$
13. U2	$B_{m-1}(i,j) \rightarrow X$
14. L6	$B_{m-1}(i,j) \rightarrow ST6$
15. R1($\beta_{i-2,j-2}$)	$\beta_{i-2,j-2}\ B(i-2,j-2) \rightarrow X$
16. E3	$0 \rightarrow ST3$
17. L3	$\beta_{i-2,j-2}\ B(i-2,j-2) \rightarrow ST3$
18. R1 (1)	$B(i-2,j-2) \rightarrow X$
19. SHIFT	See SHIFT macro below
20. OUTPUT ST3	

At this point the output template filter results are externally thresholded to determine whether further processing steps (centroid determination and tracking) are warranted. If so, then the following steps are performed:

21. OUTPUT ST4	$\overline{i}$ (X-centroid)
22. OUTPUT ST5	$\overline{j}$ (Y-centroid)
23. OUTPUT ST6	Tracking results
24. U1	$B_{m-1} \rightarrow X$
25. L6	$B_{m-1} \rightarrow ST6$

116

PIP Mnemonic	Function
SHIFT Macro	
1. REPEAT 5 times	
2. SR	Shift right
3. SUM	See SUM macro below
4. END REPEAT	
5. SD	Shift down
6. SUM	
7. REPEAT 4 times	
8. SL	Shift left
9. SUM	
10. END REPEAT	
11. SD	
12. SUM	
13. REPEAT 4 times	
14. SR	
15. SUM	
16. END REPEAT	
17. SD	
18. SUM	
19. REPEAT 4 times	
20. LS	
21. SUM	
22. END REPEAT	
23. SD	
24. SUM	
25. REPEAT 4 times	
26. SR	
27. SUM	
28. END REPEAT	

TABLE 8. NONSEPARABLE SEEKER ALGORITHMS, TEMPLATE/CENTROID/TRACKING
IMPLEMENTATION (concluded)

PIP Mnemonic	Function
SUM Macro	
1. E2	$0 \rightarrow ST2$
2. L2	$B_m(i-k,j-1) \rightarrow ST2$
3. R2 $(\beta_{i-k,j-1})$	$\beta_{i-k,j-1}\, B_m(i-k,j-1) \rightarrow X$
4. L3	Template sum $\rightarrow$ ST3
5. R2 (i/5)	X-centroid term $\rightarrow$ X
6. L4	X-centroid sum $\rightarrow$ ST4
7. R2 (j/5)	Y-centroid term $\rightarrow$ X
8. L5	Y-centroid sum $\rightarrow$ ST5
9. MOD (2,6,7)	$ST2 - ST6 \rightarrow ST7$ or $\left\| B_{m-1}(i,j) - B_m(i-k,j-1) \right\| + ST7 \rightarrow ST7$
10. U2	$B_m(i-k,j-1) \rightarrow X$

Summary of Example

In a conventional digital processor most instructions require the same
execution time in a given processor. (There are obvious exceptions, of
course: the double-length and single-length instructions in many
microprocessors form two classes.) In PIP, however, execution of each
sequence for each mnemonic requires a different number of clock states. This
implies that not only the number of instructions but also the types are
important in determining total throughput. Table 10 summarizes the actual
execution times required, grouped by instruction type. These are expressed
both in numbers of clock states and in percentage of the total execution
time. (Even with no changes, however, PIP can perform all calculations in 59
msec at a 1 MHz clock rate.) In addition to possible major cell changes, we
wanted to identify missing mnemonics. A summary of comments regarding the
adequacy of the existing instruction set follows.

PIP Mnemonic	Function
Template Match	
1. Rl $(\beta_{1-2,j-2})$	$\beta_{i-2,j-2}$ B(i-2,j-2) → X
2. E3	0 → ST3
3. L3	$\beta_{i-2,j-2}$ B(i-2,j-2) → ST3
4. Rl (1)	B(i-2,j-2) → X
5. SHIFT*	SHIFT macro in Table 6
6. OUTPUT ST3	

If threshold results justify further processing, then we perform the following steps:

X-Centroid	
7. Rl (0.2)	0.2 B(i-2,j-2) → X
8. E3	Clear X-centroid register
9. L3	0.2 B(i-2,j-2) → ST3
10. Rl (1)	B(i-2,j-2) → X
11. SHIFT†	
12. OUTPUT ST3	
Y-Centroid	
13. Rl (0.2)	0.2B(i-2,j-2) → X
14. E3	Clear Y-centroid register
15. L3	0.2 B(i-2,j-2) → ST3
16. Rl (1)	B_m(i-2,j-2) → X
11. SHIFT‡	
12. OUTPUT ST3	

*Replace SUM macro with SUM1

†Replace SUM macro with SUM2

‡Replace SUM macro with SUM3

119

TABLE 9. SEPARABLE SEEKER ALGORITHMS, TEMPLATE/
CENTROID/TRACKING IMPLEMENTATION (continued)

PIP Mnemonic	Function
Tracking	
19. U4	$B_{m-1} \rightarrow X$
20. E2	$0 \rightarrow ST2$
21. L2	$B_{m-1} \rightarrow ST2$
22. MOD	$\left\| B_{m-1}(i,j) - B_m(i,j) \right\| \rightarrow ST3$
23. R1 (1)	$B_m(i,j) \rightarrow X$
24. SHIFT*	
25. OUTPUT ST4	
26. U1	$B_m(i,j) \rightarrow X$
27. E4	$0 \rightarrow ST4$
28. L4	$B_m(i,j) \rightarrow ST4$
SUM1 Macro	
1. E2	$0 \rightarrow ST2$
2. L2	$B_m(i-k,j-1) \rightarrow ST2$
3. R2 $(\beta_{i-k,j-1})$	$\beta_{i-k,j-1} B(i-k,j-1) \rightarrow X$
4. L3	Template sum $\rightarrow ST3$
5. U2	$B_m(i-k, j-1) \rightarrow X$
SUM2 Macro	
1. E2	$0 \rightarrow ST2$
2. L2	$B_m(i-k,j-1) \rightarrow ST2$
3. R2 (i/5)	$(i/5) B_m(i-k,j-1) \rightarrow X$
4. L3	X-centroid sum $\rightarrow ST3$
5. U2	$B_m(i-k,j-1) \rightarrow X$

*Replace SUM macro with SUM4

PIP Mnemonic	Function
SUM3 Macro	
1. E2	$0 \rightarrow ST2$
2. L2	$B_m(i-k, j-1) \rightarrow ST2$
3. R2 (j/5)	$(j/5)\ B_m(i-k, j-1) \rightarrow X$
4. L3	Y-centroid sum $\rightarrow ST3$
5. U2	$B_m(i-k, j-1) \rightarrow X$
SUM4 Macro	
1. E2	$0 \rightarrow ST2$
2. L2	$B_m(i-k, j-1) \rightarrow ST2$
3. MOD (2,4,3)	$|ST2 - ST4| \rightarrow ST3$ or $\left| B_{m-1}(i,j) - B_m(i-k, j-1) \right| \rightarrow ST3$
4. U2	$B_m(i-k, j-1) \rightarrow X$

TABLE 10.   PIP EXECUTION TIMES FOR MISSILE SEEKER ALGORITHMS (2214)

Instruction Type	Clock States Required	Percent of Time
I/O (Load Array, Unload Array)	48,730	83.0
Shift (SL, SU, SD)	777	1.3
Erase (E1, E2, E3)	640	1.1
Load/Unload (L1, L2, L3, U1, U2, U3)	2,565	4.4
Regenerate (R1, R2)	2,275	3.9
Modulus	3,726	6.3
	58,713	100.0

o   Background Filter--A fourth storage cell is necessary for rapid implementation. Multiple passes through the regenerator are necessary to achieve some scaling factors.

o   Subtract and Threshold--The modulus instruction can be modified (as mentioned earlier) to directly obtain the desired result.

o   Template Match--Random-access output is desirable.

o   Centroid--No problem, except that even more storage registers (up to six) are needed.

o   Minimum Absolute Difference Correlation--No additional functions are needed.

The final data required before we can discuss architectural improvements in the new cell comes from cell modeling. We describe that in the next subsection.

PIP Modeling

Modeling of the PIP cell can be done in many ways. We initially generated a register transfer model of PIP (see Figure 66) which treats the PIP cell as a distributed processor architecture. Although useful in some contexts, this approach has a weakness; it is not a physical model, so some operations shown occur simultaneously and throughput calculations are impossible. We chose then to perform physical modeling.

The modeling of PIP is considerably more complex than modeling of CCD delay lines, filters, or imagers because the destination of spurious dark-current charge and signal charge separated from the main packet by charge transfer inefficiency cannot be readily ascertained. This occurs for two reasons. Both the temporal and spatial characteristics of such charge can be modeled in a more conventional CCD. In PIP, however, the charge left behind because of charge-transfer inefficiency must wait a variable amount of time for a trailing packet to come along. (This provides more time, in general, for

122

Figure 66. PIP Cell Model

NOTES: 1. DRO = DESTRUCTIVE READ-OUT
2. NDRO = NON-DRO
3. Σ IN REGISTER MEANS SUMMING REGISTER
4. PORTS ARE CIRCLED
5. ONLY ONE PORT PER REGISTER MAY BE ACTIVE AT ANY TIME

123

trapped charge to rejoin the main packet and, except in the case of the intercell shift registers, would probably provide a higher effective transfer efficiency within the cell.) Also PIP gates can be turned on and off with no signal charge present and the dark current generated during the ON cycle has no clear destination. An example of this is the set of B and C gates that are not adjacent to any of the corner nodes X. When a "shift-right" operation is performed the signal charge packet moves from one X node to the next. The B and C gates between the nodes, however, are on during part of the transfer and then off when charge reaches the X node. The dark current generated while they are on must go somewhere when the gates are turned off; it redistributes itself in areas of lowest potential. The exact redistribution could not be determined without solving a complex two-dimensional potential problem for every possible PIP instruction sequence—a tedious exercise of questionable value. An additional PIP nuance is the varying gate size, which leads to varying numbers of dark-current carriers being generated per unit time under the various electrodes. Nevertheless, we have generated a first-order model that can predict the performance degradation within a cell resulting from dark-current generation and charge transfer inefficiency. It can be extended easily to include the I/O instructions, shift right, shift up, etc. To generate a first-order model, several simplifying assumptions were necessary as follows:

1. A generation rate of 10 n_A/cm^2 was assumed throughout the active channel regions.

2. When dark-current charge was redistributed among various potential wells, it was assumed to stay entirely within the cell being modeled. The charge was assumed to distribute itself among charge packets of interest (ST1, ST2, ST3, X) in a way proportional to both their areas and the amount of time each gate was on (i.e., serving as a potential well).

3. Dark current was assumed conserved within the cell. (In practice, this means that no dark current was lost through sink ARV.)

4. Despite the varying gate sizes of the PIP cell, we assumed an average transfer inefficiency ϵ per transfer.

5. The regenerator circuit was modeled using the equation derived in an earlier quarterly RADC report, namely:

    Regenerator output $= K_1 A - K_2 B$, where
    A = charge in Z
    B = charge in Y

6. Second-order effects ϵ^2 terms and dark-current transfer inefficiency) are ignored.

The model equations for each instruction within the cell are shown in Table 11. Note that τ is the time for a single clock state in microseconds. Everything is reduced to numbers of electrons. The model divides the behavior into ideal and non-ideal terms to aid in predicting ideal performance for long instruction sequences. Note that noise terms are omitted, since we would expect dark current (particularly for long instruction sequences including input/output) to dominate. Nonlinearity in the regenerator performance can be readily incorporated. (Primed quantities indicate results after an instruction; unprimed quantities are the original charge packets.) This model plus the data from the seeker example give us the inputs necessary to discuss our "wish list" for design of a new cell.

Prepared Architectural Improvements

We can divide the architectural discussion into three parts as follows:

o   Input/output

o   Storage

o   Regeneration/scaling

These are discussed below.

TABLE 11.   MODEL OF THE PIP INTRACELL INSTRUCTION SET

Mnemonic	Register	Ideal Result	Degradation (Add to Ideal Result)
E1	ST1' =	0	$+0.4\epsilon$ * ST1 + 7.4τ
	ST2' =	ST2	$+1.0\epsilon$ * ST1 + 18.8τ
	ST3' =	ST3	$+1.0\epsilon$ * ST1 + 18.3τ
	X' =	X	$+1.5\epsilon$ * ST1 + 26.1τ
E2	ST1' =	ST1	$+2.5\epsilon$ * ST2 + 58.8τ
	ST2' =	0	$+0.6\epsilon$ * ST2 + 15.2τ
	ST3' =	ST3	$+2.0\epsilon$ * ST2 + 48.2τ
	X' =	X	$+2.9\epsilon$ * ST2 + 67.7τ
E3	ST1' =	ST1	$+3.9\epsilon$ * ST3 + 5.9ϵ * X + 180.7τ
	ST2' =	ST2	$+3.1\epsilon$ * ST3 + 4.6ϵ * X + 142.6τ
	ST3' =	0	$+0.6\epsilon$ * ST3 + 1.0ϵ * X + 29.6τ
	X' =	X	$+4.3\epsilon$ * ST3 - 11.5ϵ X + 199.5τ
R1	ST1' =	ST1	$+(-4.2\epsilon + 1.8K_1\epsilon)$ * ST1 + 69.7τ
	ST2' =	ST2	$+(1.2\epsilon + 1.2 K_1\epsilon)$ * ST1 + 45.6τ
	ST3' =	ST3	$+(1.2\epsilon + 1.2 K_1\epsilon)$ * ST1 + 45.6τ
	X' =	K_1*ST1	$+(1.8\epsilon - 7.2 K_1\epsilon)$ * ST1 + 69.1τ
R2	ST1' =	ST1	$+(3.1\epsilon + 1.3 K_1\epsilon)$ * ST2 + 59.4τ
	ST2' =	ST2	$+(-10.0\epsilon + 1.7K_1\epsilon)$ * ST2 + 77.6τ
	ST3' =	ST3	$+(2.9\epsilon + 1.2 K_1\epsilon)$ * ST2 + 56.3τ
	X' =	K_1*ST2	$+(3.8\epsilon - 11.4 K_1\epsilon)$ * ST2 + 76.7τ
L1	ST1' =	X	$+(-3.5\epsilon)$ * X + 100.9τ
	ST2' =	ST2	$+1.7\epsilon$ * X + 37.8τ
	ST3' =	ST3	$+1.7\epsilon$ * X + 37.8τ
	X' =	0	$+0.2\epsilon$ * X + 3.5τ
U1	ST1' =	0	$+0.1\epsilon$ + ST1 + 2.5τ
	ST2' =	ST2	$+2.0\epsilon$ * ST1 + 35.0τ
	ST3' =	ST3	$+2.0\epsilon$ * ST1 + 35.0τ
	X' =	ST1	$+ (-4.1\epsilon)$ * ST1 + 67.5τ
U2	ST1' =	ST1	$+0.7\epsilon$ * ST2 + 12.7τ
	ST2' =	0	$+0.1\epsilon$ * ST2 + 1.8τ
	ST3' =	ST3	$+0.9\epsilon$ * ST2 + 15.8τ
	X' =	ST2	$+(-1.7\epsilon)$ * ST2 + 39.7τ
U3	ST1' =	ST1	$+1.4\epsilon$ * ST3 + 27.3τ
	ST2' =	0	$+0.8\epsilon$ * ST3 + 15.1τ
	ST3' =	ST3	-0.2ϵ * ST3 + 3.4τ
	X' =	ST2	$+(-2.3\epsilon)$ * ST3 + 39.7τ
M2	ST1' =	ST1	$+(1.6\epsilon + 1.6\epsilon K_1)$ * ST1 + 177.6τ $+(7.0\epsilon - 1.6\epsilon K_2)$ * ST2 + 2.4ϵ * ST3
	ST2' =	ST2	$+(1.9\epsilon + 1.9\epsilon K_1)$ * ST1 + 207.3τ $+(5.6\epsilon - 1.9\epsilon K_2)$ * ST2 + 2.8ϵ * ST3
	ST3' =	0	$+0.1\epsilon$ * ST2 + 2.0τ
	X' =	K_1*ST1-K_2*ST2	$+(-16.5\epsilon K_1)$ * ST1 + 6.5ϵK_2 * ST2 + 273.1τ

Input/Output--This function clearly dominates the current statistics on execution times. We will discuss input and output separately.

Input speed can obviously be increased tremendously by using fully parallel array loading. If a focal plane array is mated with the PIP array using bump interconnection, then we can readily solve the input problem if we include in each PIP cell a node for inputting signal charge and add a gate for controlling that input within the focal plane unit cell. We need only include in the PIP cell a region around the node large enough for bump interconnection. The ARV region is a fine candidate region for direct sensor input data.

Outputting from the array in serial fashion cannot be improved, but in general we do not really need serial output data for the entire array. Instead, locations of selected subarrays and the data therein are more interesting. A random-access feature in PIP would be ideal for this. The rationale for including a random-access feature in PIP has been explored in a contract with Eglin AFB. Three random-access features are desirable as follows:

o   Indicating that a threshold excession has occurred somewhere in the array.

o   Providing the controller with an address (addresses) of threshold excessions.

o   Allowing the controller to randomly address the array to obtain analog data.

Although in many applications only the first level is required (decision point for further processing), all three types are desirable. For example, the missile seeker summary in Table 10 is dominated by the time required to repeatedly unload the entire array when only a portion of it is needed.

127

Storage--Tables 10 and 11 indicate several interesting things about the
existing storage architecture. First, since storage locations each must be
accessed differently, the degradations resulting from erasing each of three
registers in turn are not identical. (This was also clear in the second
quarterly report when leakage values from the three stores were presented.)
Next, although not readily apparent, portions of the time required for
regeneration and modulus functions are necessary just to clear a path for
desired results by temporarily storing the value at location X. Finally,
data loading/unloading probably consumes more time than any other operation
except I/O.

Consideration of all these factors has led to proposal of a ring store
architecture that consists of a racetrack CCD (see Figure 67). This
approach offers numerous advantages over the present storage architecture.
Some of these include the following:
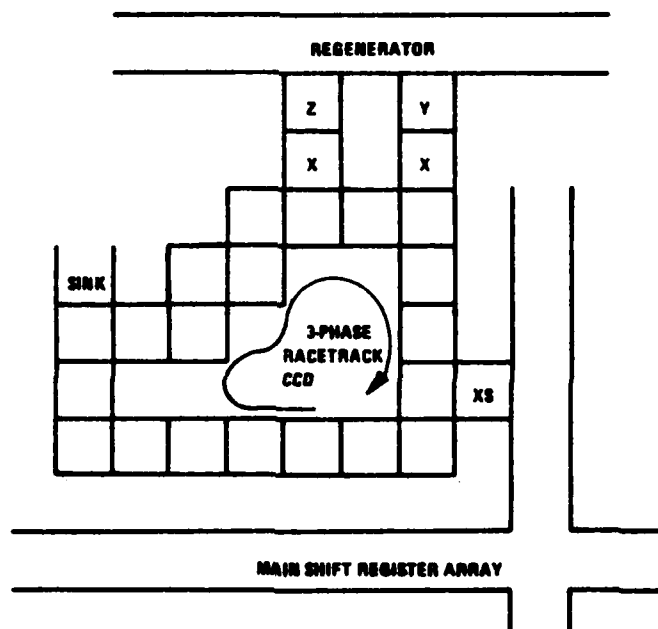
Figure 67. Ring Store Architecture

128

o   Mnemonics can be readily modified to have any source and
    destination registers. The only software change is
    minor--involving control of a few extra shifts around the racetrack.

o   The input/output node of the racetrack can be placed between the
    regenerator output and node X. This eliminates the time wasted to
    temporarily buffer the contents of X to go "around the corner" to
    stores 1 and 2.

o   The storage loop can be operated as a FIFO or LIFO, which allows,
    if desired, a "stack" architecture in addition to the independent
    mode used now.

o   Total throughput is greatly increased.

o   Additional storage locations require no additional control gates.

Regeneration/Scaling--The regenerator was significantly improved on the 3022
chip. Nevertheless, we feel that even more can be done to help regeneration
and scaling. The existing regenerator has only one multiplier; it is
associated with the noninverting input. Scaling of the inverting input is
not possible directly.

If a second multiplier is added (associated with the inverting input), then
this problem can be corrected. In addition, the seeker example illustrated
that the limited dynamic range of the regenerator's multiplying constant
leads to repeatedly regenerating a result to obtain proper scaling. This
can be eliminated by adding a coarse scaling function (based on charge
splitting by area) that can quickly scale to a value close to that desired.
Only one regeneration is needed for any scaling operation.

To illustrate the throughput improvements, we computed the number of clock
states required for two of the PIP mnemonic sets. For M2, the subtraction
instruction, the new architecture (assuming five stores) requires only 36

states versus 65 with the 3022. Erase instructions with the new architecture require only 9 to 13 states; the old architecture requires 8 to 53. The overall throughput improvement is conservatively estimated at 40% within a cell.

When the entire array is considered (including I/O) the improvement is even more dramatic. We performed an analysis of the missile seeker algorithm sequence as implemented with a PIP cell that includes all of these features. The results are listed in Table 12. In Section III we will discuss the design of new matrix processor cells.

TASK 6--INITIAL DESIGN OF 32 x 32 ARRAY CELL

The layout of a full capability cell is shown in Figures 68, 69, and 70. The elements contained in the cell are as required from the analysis of the

TABLE 12. EXECUTION TIMES FOR SEEKER ALGORITHMS (New Cell Functions)

Instruction Type	Clock Periods	Percent of Total
I/O	90	1.4
Shift	777	12.1
Erase	480	7.5
Load/Unload	1,881	29.4
Regenerate	1,820	28.5
Modulus	1,350	21.1
	6,398	100.0
Improvement in execution time vs old cell: 91%		
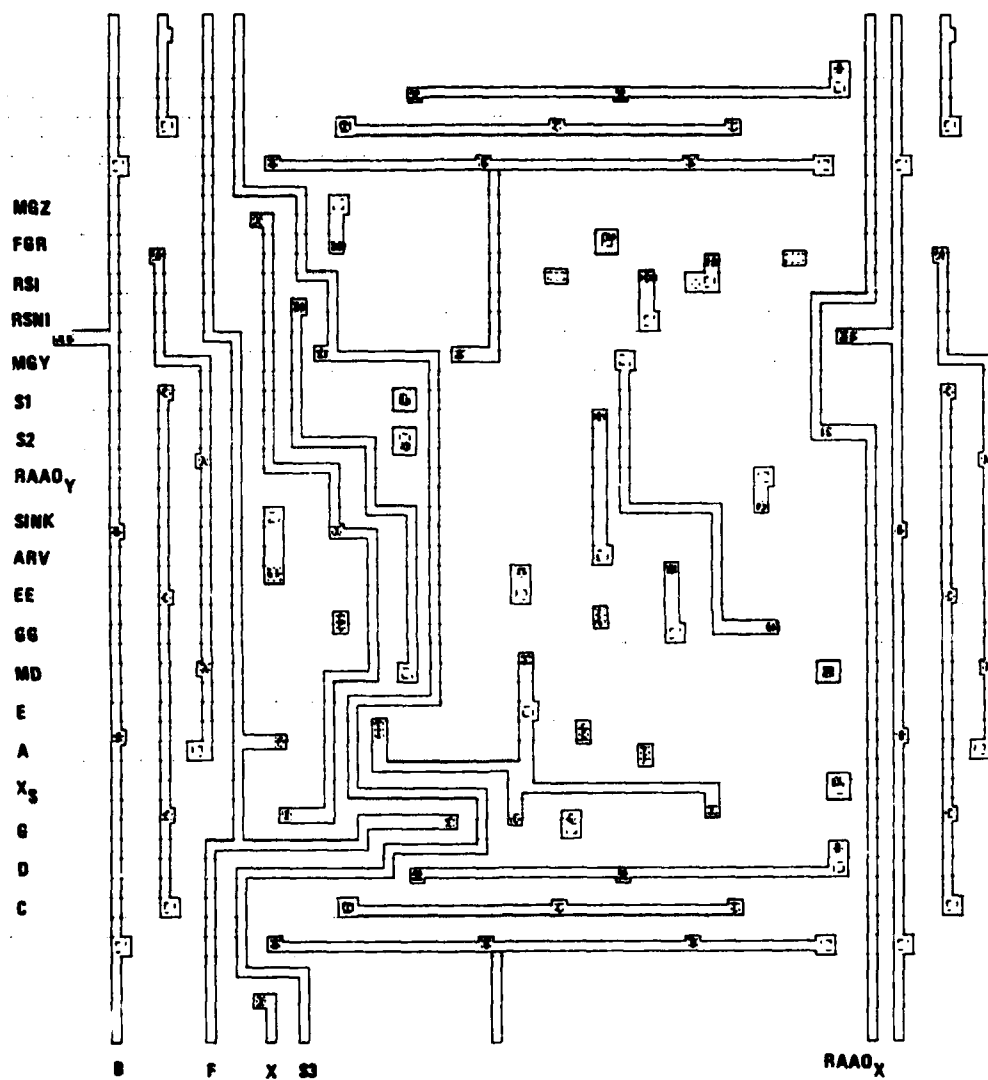
Figure 68. Clock Phase Two-Metal Interconnect CALMA Plot of
Second-Generation Matrix Processor
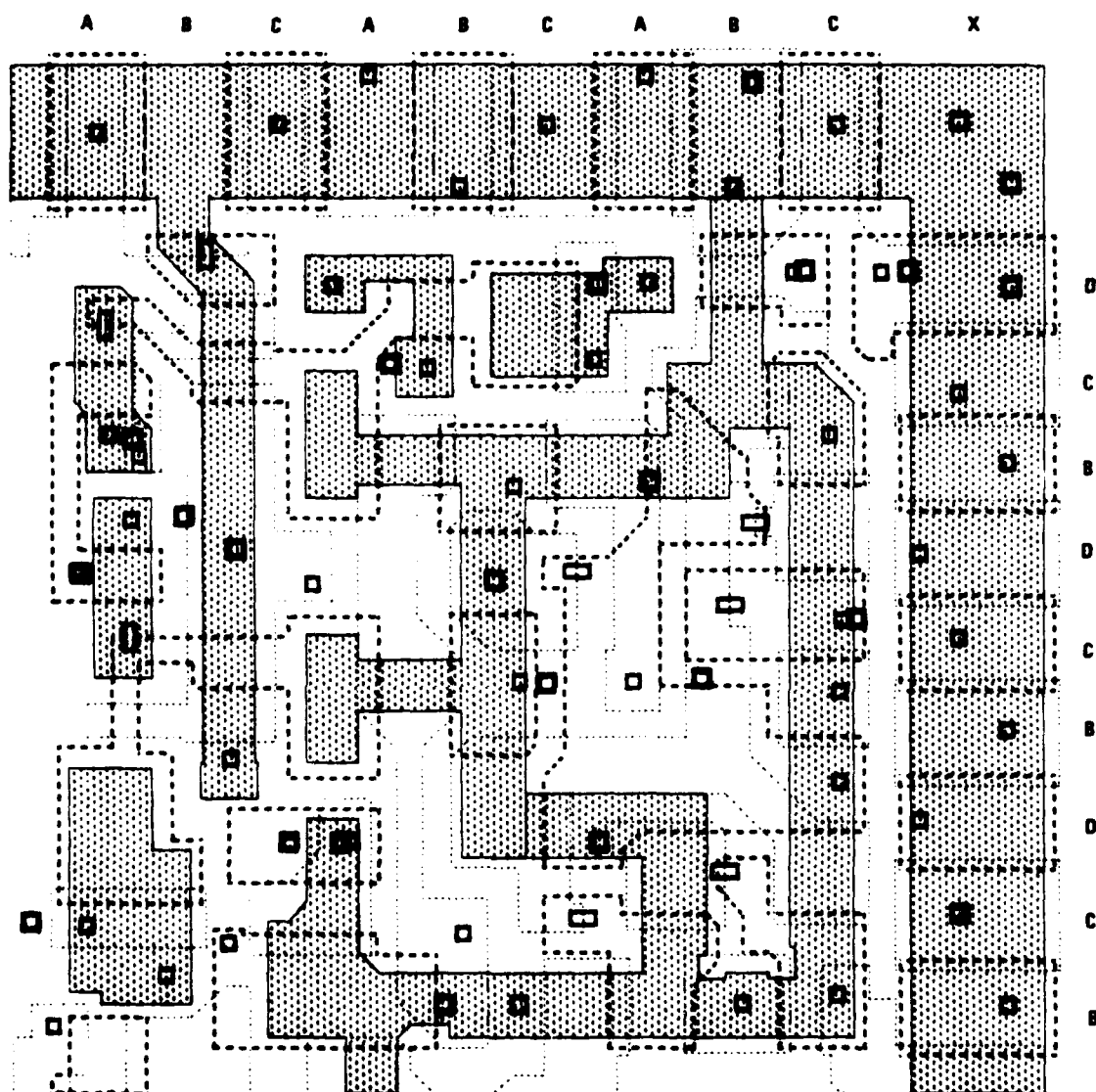
Figure 69.   Field Cut and Two-Poly Layer CALMA Plot of Second-Generation
Matrix Processor (Note: Racetrack internal to cell)

132
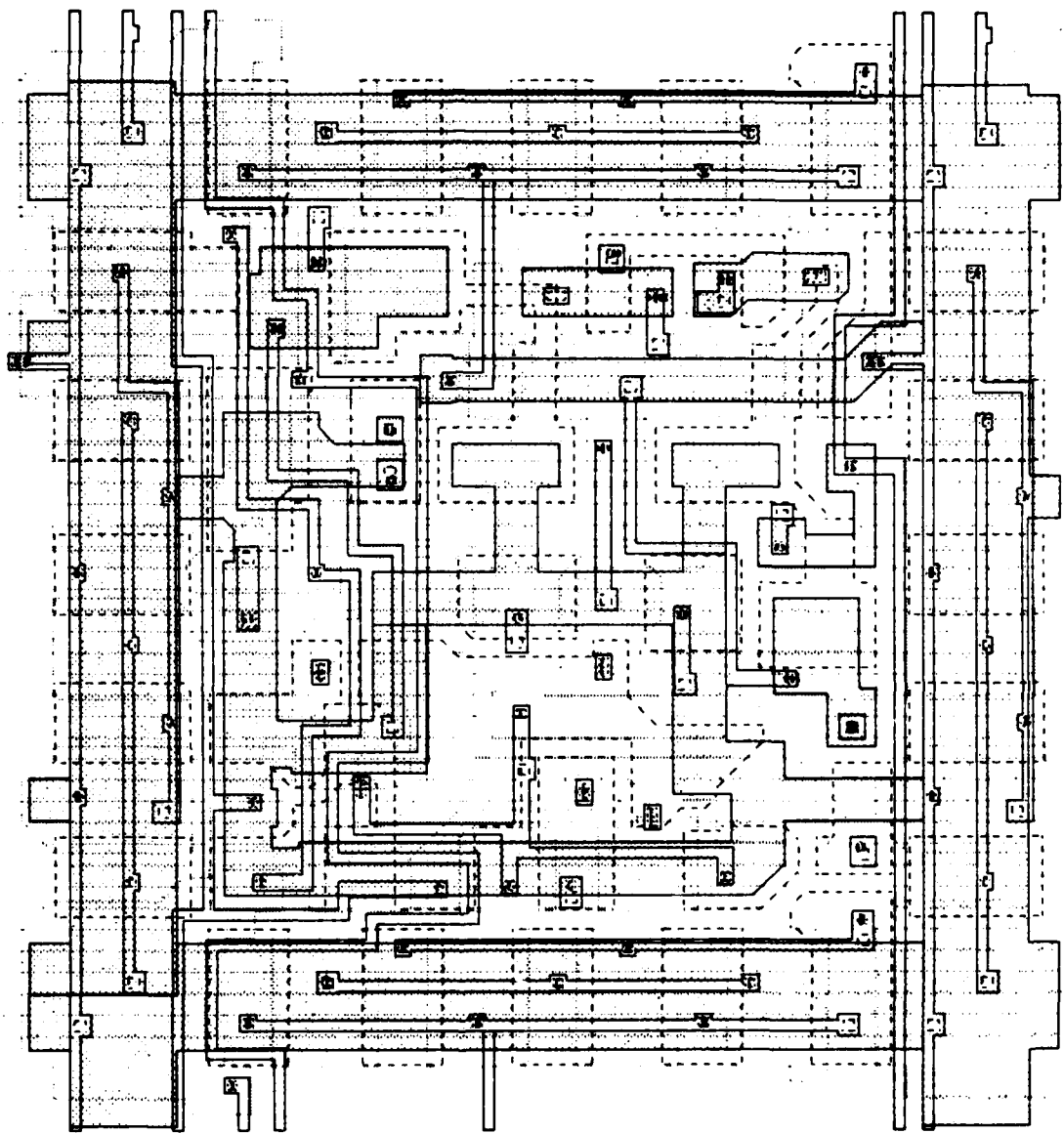
Figure 70. Composite CALMA Plot of Second-Generation Matrix Processor

133

algorithms discussed in Task 5 on Software and System Architecture. Each
component and its function is listed below.

1. Ring Store

    This is a re-entrant three-phase CCD delay line that is entirely
    within the cell and does not interfere with the cell-cell shift
    registers except for sharing the C and X phases for erase and FGA
    input functions.

    There is communication to the horizontal cell-cell shift register
    through the gate labeled "XS" that is similar to the XS function on
    the 3022/2214 chip. The racetrack has five distinct phase gates
    (three plus two extra for loading/unloading the FGA ports): E, F, G,
    EE, and GG.

    There is access to a SINK/INPUT diffusion at the left-middle part of
    Figure 69 controlled by Phase C. The access to the FGA Z and Y ports
    is through phases EE and GG, respectively, and can be seen in the
    center of Figure 69. Entry to Z and Y is controlled by phase X as
    before in the 3022/2214 chips.

2. FGA

    This is the same basic design as used on the 3022 except that each
    input Z and Y have identically loaded multipliers with independent MG
    controls: MGZ, MGY. It is visible near the top of the cell in
    Figure 69.

3. 0.2X Coarse Splitter/Scaler

This is a new component located to the left of the FGA, and it
comprises a simple three-gate potential equilibration charge
divider. This has been demonstrated to work extremely well on a
Honeywell CCD/HCT IR imager chip. Details are available for that
chip if needed. The operation is very simple. Charge is loaded into
the larger gate region (S3) from the cell-cell shift register using
gate A. Both S2 and S1 gates are at ON potentials at this time, so
charge from A equilibrates in channel potential following A Switch
OFF. The time constant is diffusion-limited (L^2/D), where the
longest path is perhaps two shift gates. Phase S2 is then switched
OFF and divides the charge into two regions under S3 and S1. The
areas are chosen to give an accurate 0.2 ratio of charge in S1 to
charge in S3. Having completed that division, the X control is
switched ON, opening access to the SINK/INPUT diffusion as used for
the Ring Store. S3 can then empty the excess charge into the sink
followed by X Switch OFF. The scaled charge can then be returned to
the A phase and continue in further processing. The net effect of
this is that virtually any sensitivity of charge-charge can be
programmed for the regenerate primitive. The MGZ and MGY controls
can be set to any appropriate analog voltage and then multiple splits
of 0.2 each applied to that gain. It is usual in algorithms to
require a simple set of values for threshold and smoothing, so that
the controller need only store a few coefficients in the primitive
routine. A D/A is then called for on the controller VLSI chip.

4. Random Access Analog Output Transistor

This is an FET located at the mid-righthand side of the cell. The
gate of this FET is a small extension of the Y input gate to the FGA
and links the FGA to the MGY multiplier structure. The source and

drain of this RAAO-FET are then simultaneously accessible in the
X-direction and Y-direction at the edges of the array.

The structure therefore resembles a conventional DRAM except that the
data bit is not influenced by the word and bit-line capacitance save
for a small gate-diode overlap capacitance. A simple MOS comparator
must then be supplied with a common reference voltage. Depending on
the gain of the simple comparator—programmable if desired—either a
digital or an analog output can be obtained in one or two clock
states (e.g., 0.5 μsec total) for an entire array or any selection
of lines of the array.

The selection of lines is controlled by the other axis. Each of the
word-lines is to be driven either ON or OFF with a simple drive.
This can be some small overhead to the matrix processor or N pins can
be added to the list of the total pin-outs for the chip to allow the
controller chip (or Exerciser unit) to drive.

The significance of this component is to be considered in terms of
the higher level algorithms for adaptive sequences. The output is
not only very fast from the entire array, but it is nondestructive.
This means that conditional jumps can be executed by the
controller/sequencer depending on the presence or absence of events
in the data matrix.

A truly smart sensor can therefore be implemented that has virtually
no constraints on programmability.

5. Sink/Input Diffusion

This is an input node to the cell of a DC sink node, depending on the
application of the array. It is intended that a sensor chip (e.g., a

136

silicon imager) can be indium-bumped to give a direct parallel input to the cell. This sensor must provide the mode select switches for program control. A simple pair of FETs are required on that chip (or other device; e.g., tactile input) to select between "Read Image" (charge) or "Sink data" to reset voltage.

These functions are all provided without in any way interrupting the bit-serial I/O presently available on the 3022/2214 chips. The design has been laid out to CALMA using VLSI design rules for both a $9 \times 9 \text{ mil}^2$ cell and a $3 \times 3 \text{ mil}^2$ cell. The $9 \times 9 \text{ mil}^2$ cell uses the same rules as have been used on the VLSI PtSi imager chip #3318 presently under development at Honeywell. The $3 \times 3 \text{ mil}^2$ cell uses a state-of-the-art fabrication process that is not yet readily available. A simple scaling factor can be applied to the design to shrink the cell to the appropriate cell pitch required by the system application. The smallest cell achievable on a given process is determined by the second metal (including via) pitch. There are 20 horizontal second-metal lines per cell, as seen in Figure 68.

TASK 7--DESIGN OPTIMIZATION REVIEW WITH RADC

Three major reviews were held with RADC: one was early in the program at SRC when the design corrections and modifications to the original chip were discussed and agreed upon, the second was when the initial 32 x 32 cell design was presented to RADC, and the final review for customer acceptance was at the end of the contract. At each of these reviews care was taken to consider in detail not only the actual chip design but also the system implications of that design. RADC requested that the CCD be capable of operating on as broad a range of applications as feasible, including non-image SIMD operations.

All the requirements of RADC and Honeywell have been met in concept. It remains only to verify the actual chip performance in a real application.

The Exerciser unit delivered to RADC is sufficient to operate all the
presently conceived generations of the architecture. Sufficient expansion has
been provided so that even extra clock phases for, say, Parallel RAAO can be
accommodated with only software development effort.

SECTION III

PROJECTIONS OF FUTURE DEVELOPMENTS OF MATRIX
PROCESSOR CCDs WITH PARALLEL I/O

In this section we present an estimate of the follow-on development options
that seem both desirable and feasible in the near future and perhaps out to
the 1987-88 time frame.

Current CCD and sensor technology, and near-term advanced technology
development, are considered for a feasibility demonstration of a complete
subsystem, possibly by late 1985.

Controller and software/memory optimization will be described in the context
of a realizable three- or four-chip assembly: sensor, processor, controller,
and optional host CPU.

SENSOR/PROCESSOR

Initial interest is for a minimum offset sensor that provides an existing
capability of integrated FET switches for multiplexing, or CCD line-parallel
(least desirable) addressing to mate with a second-generation matrix processor
array. An attractive candidate is the PtSi monolithic CCD imager developed by
RADC and currently under advanced development both by Honeywell and other
manufacturers in the US.

Other candidates that may possibly be usable include InSb, HCT, and
pyro-electric (on a silicon chip), which are maturing rapidly, and in some
cases have demonstrated monolithic CCD structures (in the technical
literature) for self-scanned arrays. The worst nonuniformities of spectral
responsivity with PV operation of, say, HCT would be the most difficult to

deal with in a matrix processor, although we think not impossible. An exception may be automatic compensation methods from off-chip references and shutterless compensation algorithms techniques executable by the processor, as mentioned in Task 6.

Thermal mismatch for cooled sensor hybrids must be considered in detail when using non-silicon sensor arrays. Considerable experience has already been gained by Honeywell (SRC/EOO) with indium bump-connected HCT and LTT PV arrays. This technology has been under development using IR&D funding and also on several contracts dating from about 1976, and is now at a sufficiently advanced stage for use on matrix processor development.

A special PtSi imager chip would be designed to allow either line-parallel readout to line-parallel input circuitry on the new matrix processor or, more ambitiously, a full two-dimensional parallel accessed array. Even with limited design rules, an $8 \times 8 \text{ mil}^2$ processor cell seems practical with current technology. This implies a 32 x 32 array on a $320 \times 300 \text{ mil}^2$ chip with a full set of peripheral circuitry for RAAO readout. The operation of such an array would initially be using the Exerciser unit already developed since this gives greatest flexibility and serves also as an algorithm development system for parallel operators. However, early design of at least part of the final chip for control of the matrix processor will be undertaken on Honeywell's IR&D funding to finalize the processor-controller partitioning of functions.

SOFTWARE AND CONTROL

Maximum benefit from the matrix processor concept will only accrue when we miniaturize the controller as well as the matrix processor/sensor array. Although an Exerciser unit serves valuable functions in both validating the matrix processor concept and testing various combinations of primitives, the unit is too large (and, frankly, unnecessarily powerful) for a system

application. We feel that design of a controller chip is an important step toward development of a very compact sensor/processor/controller module which could revolutionize the approach to applications requiring moderate resolution and high throughput.

The control architecture being used in the Exerciser box is conventional--a 2910 microprogram sequencer steps through successive words of ROM. No new primitive may begin until its predecessor is complete. We foresee improvements in two areas of the controller that would facilitate its integration on a single chip: architecture and storage.

Honeywell has developed (as part of the DoD VHSIC program) an innovative approach to microprogram sequencer design that incorporates two stacks--one address stack and one counter stack. The counter stack concept allows considerable flexibility in vesting of unfinished loops. Such a feature would be a great asset in upgrading the controller to respond to commands like "SR5," which would require repetition of one primitive and also repetition of a subprimitive within it. In other words, "SR" includes a partial sequence that controls the A, B, and C clocks; that sequence is repeated three times within the primitive. We need only store that sequence once and loop through it three times. We can also use the same sequence for "SL" but decrement addresses rather than increment.

Another architectural improvement would use the greater flexibility of the new cell design developed in this program. In Section II, Task 5, we outlined the possibility of overlapping instructions to enhance throughput. This may now be possible because operations in different parts of the cell can be carried out simultaneously with little or no interference. (In the 2214/3022 design, each clock often performed functions at several locations within the cell).

141

Storage of primitives in the present Exerciser unit is in 48-bit words, where up to 32 bits may be allocated for matrix processor clock states. This is desirable from an experimentalist's point of view, since changes in primitives can be readily inserted. In a system application, however, we need not waste so much storage when it is highly likely that only a few bits change each time. By either coding only differences in states from cycle to cycle (similar to image bandwidth compression) or using ROM compression techniques, we could substantially reduce the memory size required to implement a set of primitives in a system application. Such a reduction obviously makes a single-chip implementation simpler.

Honeywell is using 1982 internal funding to investigate controller architectural and storage improvements. This work will culminate in a design and (if contract funding can be identified) fabrication of a single-chip controller that would take a major step toward system applications for the matrix processor concept.

SECTION IV

SUMMARY AND CONCLUSIONS

An architecturally novel Parallel CCD Matrix Processor has been developed.
The processor is fully programmable and in its most elaborate form can
exceed the throughput of virtually any available image processor presently
known. At the heart of the device is a unique, linear, programmable
floating gate amplifier (FGA), which is small enough to fit into a 3 x 3
mil^2 pixel together with six storage sites, a fixed-ratio scaler, and
SINK/INPUT node. One axis of the two-input FGA is used additionally as a
control for periphery-access in a random access analog output readout mode.
This cell with a full complement of features represents the state-of-the-art
of packing density for a general-purpose signal and image processing
algorithm implementing IC.

Although not completely debugged, the two chips fabricated and tested have
shown all the basic design concepts and parameters to function as intended.
It remains to exercise a "perfect" chip and demonstrate a single-frame,
high-order image processing algorithm for contrast enhancement using
threshold and edge-extraction subroutines. The first demonstration of
partial operations has been achieved, albeit with assistance from the
host/controller CPU. There are no known design concept barriers to
implementing the first fully programmable smart sensor system. Fabrication
limits to uniformity of offset in the FGA have as yet prevented a complete
demonstration, but this is expected to be accomplished in the near future.

A flexible Exerciser and Algorithm Development System (microprocessor-based)
has been constructed and exercised using a multiplicity of signal/image
sources. Both real-time and freeze-frame modes of activity are accommodated
by this powerful unit, which provides all the necessary drive voltages to
operate both first- and second-generation CCD matrix processor chips.

143

A cheap, robust, multispectral subsystem of a few watts power requirement is foreseen as a fabrication goal. Leading up to that goal, construction of a second-generation CCD chip and an appropriate image sensor is seen as a necessary stage of development. Included in that development is the design and possible fabrication of a programmable microcontroller/clock-driver VLSI CMOS chip.

Unique analog SIMD algorithms are now provided with hardware to speed software development. Many unique arithmetic features of the CCD processor provide new opportunities for efficiency and speed of execution of ultra-high throughput parallel operators. There remains a great deal to be researched and tested with this new processor but construction is demonstrated to be feasible on a standard IC CCD process line to yield full-capability arrays of perhaps 32 x 32. An advanced CCD fabrication line would permit up to 128 x 128 arrays with only geometry scaling required from the larger cell design.

APPENDIX A

HP41C POCKET COMPUTER LISTING

The following pages give the HP41C pocket computer listing for general
single-gate electrode calculations of CCD and Floating Gate CCD
one-dimensional approximations.

PRP "DATA"

```
01◆LBL "DATA"
02 FIX 3
03 RCL 02
04 "VFB="
05 ARCL 02
06 PROMPT
07 STO 02
08 RCL 06
09 "VGA="
10 ARCL 06
11 PROMPT
12 STO 06
13 SCI 4
14 RCL 04
15 "AG."
16 ARCL 04
17 PROMPT
18 STO 04
19 "9:11,EX,S:10,12"
20 PROMPT
21 RCL 07
22 RCL 04
23 *
24 "QS="
25 ARCL X
26 PROMPT
27 XEQ "QSIG"
28 XEQ "N"
29 "RDY"
30 AVIEW
31 RTN
```

SET UP CONSTANTS

```
32◆LBL "VO"
33 RCL 00
34 1/X
35 RCL 10
36 *
37 "OX:"
38 ARCL X
39 PROMPT
40 1/X
41 RCL 10
42 *
43 STO 00
44 PSE
45 1/X
46 X↑2
47 RCL 16
48 "NA."
49 ARCL 16
50 PROMPT
51 ENTER↑
52 STO 16
53 *
54 RCL 11
55 RCL 12
56 *
57 *
58 STO 01
59 FIX 3
60 "VO="
61 ARCL X
62 SCI 4
63 PROMPT
64 RTN
```

COMPUTE C_{OX}, V_O

t_{OX} (Cm)

$$V_O = \frac{q\varepsilon_O \varepsilon_{Si} N_A}{(C_{OX})^2}$$

STORES

00	=	$C_{OX} = \varepsilon_O \varepsilon_{OX}/t_{OX}$					
01	=	V_O	11	=	q, electron	21	= X_O'
02	=	V_{FB}	12	=	$\varepsilon_O \varepsilon_{Si}$	22	= X_O
03	=	V_G'	13	=	β	23	= X_1
04	=	A_G	14	=	γ	24	= X_1'
05	=	A_G/C_S	15	=	$\sqrt{\beta^2 - 4\gamma}$	25	= F
06	=	V_{FG}^O	16	=	N_A	26	= E
07	=	Q_S/A_G	17	=	N_D	27	= D
08	=	$V_O^2 + 2V_O V$	18	=	X_J	28	= G
09	=	C_S (PF)	19	=	$\Delta V_{FG} - Q_S/A_G C_{OX}$	29	= ΔV_{FG}
10	=	$\varepsilon_O \varepsilon_{OX}$	20	=	$-X_M$	30	= B

ASN'S: DATA, V_O, V_Z, PSI, BVFG, SVFG, VZQ, QSIG, SDVFG

```
65◆LBL "PSI"
66 XEQ "QSIG"
67 RCL 00
68 /
69 RCL 02
70 -
71 ENTER↑
72 "VG?"
73 PROMPT
74 +
75 STO 03
76 "V↑="
77 ARCL X
78 AVIEW
79 XEQ 02
80 SQRT
81 CHS
82 RCL 01
83 +
84 RCL 03
85 +
86 FIX 4
87 "PSI="
88 ARCL X
89 SCI 4
90 PROMPT
91 RTN
```

COMPUTE
ψ_S

$$V'_G = V_G - V_{FB} + Q_S/A_G C_{OX}$$

$$\psi_S = V'_G + V_O - \left[V_O^2 + 2V_O\,V'_G\right]^{1/2}$$

147

92◆LBL "SVFG"	COMPUTE ΔV_{FG}	139 RCL 05
93 "CS.PF:CLX"		140 RCL 00
94 PROMPT		141 *
95 1 E-12	EITHER KEY	142 X↑2
96 *	C_S (PF)	143 *
97 1/X	OR	144 STO 14
98 RCL 04	XEQ M THEN CLX	145 4
99 *	THEN R/S	146 *
100 STO 05		147 RCL 13
101 RCL 00		148 X↑2
102 X↑2		149 -
103 *		150 CHS
104 RCL 01		151 SQRT
105 *		152 STO 15
106 RCL 06		153 RCL 13
107 "VG0:"		154 CHS
108 ARCL 06		155 +
109 PROMPT	NEW VALUE?	156 2
110 STO 06		157 /
111 RCL 02		158 SCI 2
112 -	$V = V_{FG}^O - V_{FB}$	159 "<>VFG:"
113 XEQ 03		160 ARCL X
114 STO 08		161 SCI 4
115 SQRT		162 PROMPT
116 RCL 00		163 RCL 07
117 *		164 RCL 00
118 +		165 /
119 RCL 07		166 -
120 +		167 STO 19
121 RCL 05		168 RCL 01
122 XEQ 01		169 XEQ 01
123 STO 13		170 RCL 00
124 RCL 08		171 -
125 SQRT		172 CHS
126 RCL 07		173 SQRT
127 RCL 00		174 CHS
128 /		175 RCL 06
129 +		176 +
130 X↑2		177 RCL 02
131 RCL 07		178 -
132 RCL 00		179 RCL 19
133 /		180 -
134 RCL 01		181 RCL 01
135 XEQ 01		182 +
136 RCL 08		183 "PSI="
137 +		184 ARCL X
138 -		185 PROMPT
		186 RTN

"ΔV_{FG} = VALUE"

"ψ_S = VALUE"

AT NEW V_{FG}

$$\beta = \frac{2A_G}{C_S} \left\{ \frac{Q_S}{A_G} + C_{OX} \left[V_O^2 + 2V_O \left(V_{FG}^O - V_{FB}\right) \right]^{1/2} + \left(\frac{A_G}{C_S}\right) C_{OX}^2 V_O \right\}$$

$$\gamma = \left(\frac{A_G}{C_S}\right)^2 \left[\frac{Q_S}{A_G} + C_{OX} \left[V_O^2 + 2V_O \left(V_{FG}^O - V_{FB}\right) \right]^{1/2} \right]^2$$

$$- C_{OX}^2 \left[V_O^2 + 2V_O \left(V_{FG}^O - V_{FB} + \frac{Q_S}{A_G C_{OX}}\right) \right] \right\}$$

$$V_{FG}^O - V_{FG}^Q = \Delta V_{FG} = \frac{-\beta + \sqrt{\beta^2 - 4\gamma}}{2} \quad , \quad Q_S = 0 \rightarrow \gamma = 0 \text{ and } \Delta V_{FG} = 0$$

148

187•LBL "QSIG" | SET Q_SIG
188 "QS?"
189 PROMPT | pC
190 ABS
191 CHS
192 RCL 04
193 "AG="
194 ARCL X
195 AVIEW
196 /
197 1 E-12
198 *
199 STO 07
200 RTN

201•LBL "SDVFG" | SLOPE SCCD
202 RCL 13
203 RCL 15
204 /
205 1
206 -
207 RCL 05
208 *
209 RCL 07
210 RCL 08
211 SQRT
212 RCL 08
213 *
214 +
215 RCL 08
216 RCL 01
217 *
218 -
219 2
220 *
221 RCL 05
222 X↑2
223 *
224 RCL 15
225 /
226 -
227 RCL 04
228 /
229 "4=" | SLOPE = VALUE
230 ARCL X
231 AVIEW
232 RTN

233•LBL "M" | SET LOAD FACTOR
234 "M=?"
235 PROMPT | $M = \dfrac{C_S}{A_G C_{OX}}$
236 RCL 00
237 RCL 04
238 *
239 *
240 1 E11
241 *
242 STO 09
243 RTN

244•LBL 01
245 *
246 2
247 *
248 RTN

249•LBL 02
250 RCL 03

251•LBL 07
252 RCL 01
253 XEQ 01
254 RCL 01
255 X↑2
256 + | $\left[V_O^2 + 2\, V_O\, V_G' \right]$
257 RTN

$$\text{SLOPE} = \frac{1}{C_S}\left[\sqrt{\frac{\beta}{\beta^2 - 4\gamma}} - 1\right] - \frac{2}{\sqrt{\beta^2 - 4\gamma}}\left(\frac{A_G}{C_S^2}\right)\left[\frac{Q_S}{A_G} + C_{OX}\left[V_O^2 + 2\,V_O\,(V_{FG}^O - V_{FB})\right]^{1/2} - 2\,V_O C_{OX}\right]$$

149

```
258◆LBL "VZ"        ┌──────────────────┐
259 XEQ "QSIG"      │ FIXED GATE V_Z   │
260 SF 02           └──────────────────┘
261 XEQ 13
262 CF 02
263 RCL 06
264 SF 00
265 XEQ 06
266 SQRT
267 XEQ 08
268 CF 00
269 -
270 CHS
271 STO 24
272 "DEP="          ┌────────────────────────────────────┐
273 ARCL X          │ SURFACE DEP. WIDTH = VALUE CM      │
274 AVIEW           └────────────────────────────────────┘
275 PSE
276 XEQ "VZ0"
277 RTN

278◆LBL "VZ0"       ┌──────────────────────┐
279 RCL 18          │ FLOATING GATE V_Z    │
280 RCL 20          └──────────────────────┘
281 +
282 RCL 24
283 -
284 X↑2
285 RCL 30
286 *
287 "VZ="
288 ARCL X
289 PROMPT          ┌────────────────┐
290 RTN             │ V_Z = VALUE    │
                    └────────────────┘
```

$$B = \frac{q\, N_D\, (N_A + N_D)}{2\, \epsilon_O\, \epsilon_{Si}\, N_A}$$

BURIED CHANNEL
FLOATING GATE

COMPUTE ΔV_{FG}

INITIALIZE

```
291◆LBL "SVFG"
292 CF 00
293 CF 01
294 XEQ "DATA"
295 XEQ "V0"
296 RCL 18
297 "XJ="
298 ARCL 18
299 PROMPT
300 STO 18
301 RCL 17
302 "ND "
303 ARCL 17
304 PROMPT
305 STO 17
306 RCL 16
307 +
308 RCL 16
309 /
310 RCL 17
311 *
312 RCL 11
313 *
314 RCL 12
315 /
316 2
317 /
318 STO 3A

319◆LBL 13
320◆LBL "GO"
321 RCL 06
322 "VGA="
323 ARCL X
324 PROMPT
325 STO 06
326 RCL 07
327 RCL 17
328 / /
329 RCL 11
330 /
331 STO 20
332 CHS
333 "XM="
334 ARCL X
335 AVIEW
336 CHS
337 XEQ 04
338 STO 21
339 FS? 02
340 RTN
341 0
342 XEQ 04
343 STO 22
344 RCL 06
345 CF 00
346 CF 01
```

$X_M = \dfrac{Q_S}{q\,N_D A_G}$

FIXED GATE V_Z

SET $X_M = 0$

COMPUTE X_1' OR X_1

```
347◆LBL 00
348 XEQ 06
349 SQRT
350 XEQ 09
351 -
352 CHS
353 FC? 00
354 STO 23
355 FS? 00
356 STO 24
357 RCL 24
358 X<0?
359 GTO 09
360 CLD
361 0
362 SF 00
363 XEQ 06
364 STO 25
365 XEQ 10
366 RCL 26
367 RCL 23
368 *
369 4
370 /
371 RCL 21
372 +
373 RCL 26
374 *
375 RCL 27
376 +
377 RCL 28
378 *
379 RCL 25
380 +
381 SQRT
382 CHS
383 RCL 26
384 RCL 23
385 *
386 2
387 /
388 +
389 RCL 23
390 -
391 RCL 21
392 +
393 RCL 26
394 *
395 FC? 01
396 STO 12
397 "DVFG="
398 SCI 2
399 ARCL X
400 AVIEW
401 SCI 4
```

X_O' OR X_O

COMPUTE E, D, G
$= q\,N_D A_G/C_S$

$D = V_{FG}^O - EX_1$

$G = 2\epsilon_0\epsilon_{Si}/(2B\epsilon_0\epsilon_{Si})$

$F = X_O'^2 + G\,(V_{FB} - B\,(X_J - X_M))$

ΔV_{FG} = VALUE

$$V_{FG}^Q - V_{FG}^O = \Delta V_{FG} = E\left\{ X_O' - X_1 + \frac{EG}{2} - \left(DG + 2C\,X_O' + \frac{E^2G^2}{4} + F\right)^{1/2}\right\}$$
$$\leq 0$$

402*LBL 12
403 STO 29 ΔV_{FG}
404 FS?C 01
405 GTO 11   SECOND PASS?
406 RCL 05   STOP
407 +
408 SF 00   USE X_M
409 SF 01
410 XEQ 00   SECOND PASS
               COMPUTE X'_1
411*LBL 04   COMPUTE X'_0 OR X_0
412 RCL 18
413 +
414 RCL 30
415 XEQ 01
416 RCL 00
417 *
418 XEQ 05
419 +
420 RCL 12
421 *
422 RCL 06
423 /
424 XEQ 07
425 RTN

426*LBL 05   COMPUTE qN_D
427 RCL 11
428 RCL 17
429 *
430 RTN

431*LBL 06 COMPUTE
432 RCL 18
433 FS? 00 $x^2 + \dfrac{2\epsilon_0\,\epsilon_{Si}\,(V + V_{FB} - B(X) - X_M)^2}{2\,B\epsilon_0\epsilon_{Si} - qN_D}$
434 RCL 29
435 FS? 00
436 +
437 X↑2
438 RCL 30
439 *
440 -
441 RCL 02
442 +
443 RCL 12
444 XEQ 01
445 XEQ 07
446 XEQ 00 X'_0 OR X_0
447 X↑2
448 +
449 RTN

450*LBL 07   COMPUTE
451 RCL 12 $\div\ (2B\epsilon_0\epsilon_{Si} - qN_D)$
452 RCL 30
453 XEQ 01
454 XEQ 05
455 -
456 /
457 RTN

458*LBL 03   RETURN X'_0 OR X_0
459 FS? 00
460 RCL 21
461 FS? 00
462 RCL 22
463 RTN

464*LBL 09   ON ERROR
465 "XDEP<0"   DISPLAY
466 AVIEW   XDEP < O
467 PSE
468 0
469 STO 24   RESET $X_1 = 0$
470 XEQ "BVFG" RESTART
471 RTN

472*LBL 10   COMPUTE E, D, G
473 XEQ 05
474 RCL 09
475 1 E-12
476 *
477 /
478 RCL 04
479 *
480 STO 26
481 RCL 23
482 *
483 CHS
484 RCL 06
485 +
486 STO 27
487 RCL 12
488 2
489 *
490 XEQ 07 $\div\ (2B\epsilon_0\epsilon_{Si} - qN_D)$
491 STO 28
492 RTN

493*LBL 11
494 CF 00   BREAK LOOP
495 END   AND STOP

ANALYSIS OF BCCD FLOATING GATE SENSING

Refer to Figure A1. The analysis starts by assuming charge balance in conditions (a) and (b):

So,

$$Q_{FG}^O + Q_{D1}^O = 0 \tag{A-1}$$

$$Q_{FG}^Q + Q_{D1}^Q = 0 \tag{A-2}$$

where Q_{D1} = surface space-charge in X_1, X_1^1.

Conservation of charge yields:

$$C_S V_{FG}^Q + Q_{FG}^Q = C_S V_{FG}^O + Q_{FG}^O \tag{A-3}$$

From Haken (Reference 4), we know that

$$Q_{FG}^O = -qN_D X_1 A_{FG} \tag{A-4}$$

$$Q_{FG}^Q = -qN_D X_1' A_{FG} \tag{A-5}$$

where

$$X_1 = X_O - \left[X_0^2 - \frac{2\varepsilon_{Si} (AX_J^2 - V_{FG}^O + V_{FB})}{2\varepsilon_{Si}A - qN_D} \right]^{1/2} \tag{A-6}$$

and X_1^1 is of the same form as (A-6) but X_0^1 replaces X_0 and $X_J - X_M$ replaces X_J.

$$X_O = \frac{\varepsilon_{Si}(2AX_J C_{OX} + q N_D)}{C_{OX}(2\varepsilon_{Si} - q N_D)} \tag{A-7}$$

$$X_O = \varepsilon_{Si} \frac{(2A(Xj - Xm) C_{OX} + qN_D)}{C_{OX} (2\varepsilon_{Si} - qN_D)}$$
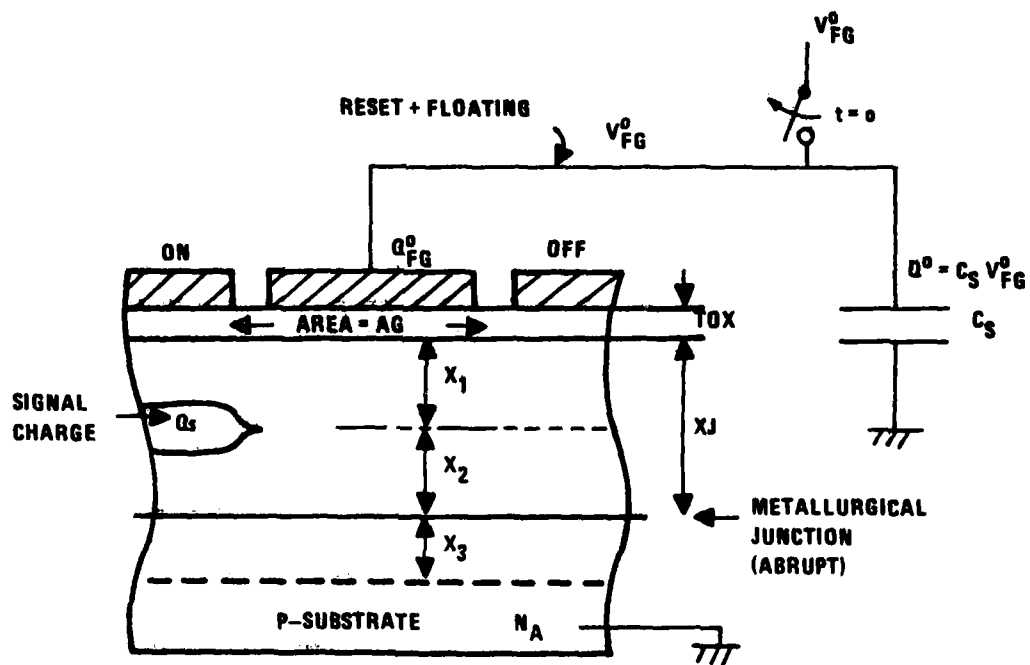
a. Reset mode with no channel charge
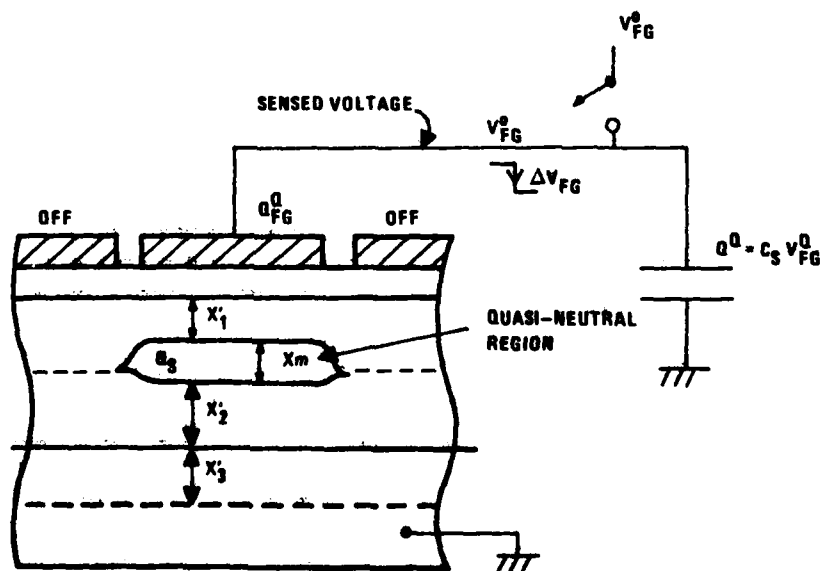
b. Sensed mode with changed V_{FG}

Figure A1. BCCD Floating-Gate Sensing Model for Analysis of Regenerator

154

Combining the above equations gives a simple expression for V_{FG}^Q:

$$V_{FG}^Q = V_{FG}^O - \frac{qN_D X_1 A_{FG}}{C_S} + \frac{qN_D X_1' A_{FG}}{C_S} \qquad (A-8)$$

By defining the following quantities, a solution for $V_{FG}^Q - V_{FG}^O$ is found:

(Notice X_1^1 is a function of V_{FG}^Q.)

$$D = V_{FG}^O - \frac{qN_D X_1 A_{FG}}{C_S}$$

$$E = \frac{qN_D A_{FG}}{C_S}$$

$$F = X_O'^2 - \frac{2\varepsilon_{Si}\left(A(Xj - Xm)^2 + V_{FB}\right)}{2\varepsilon_{Si}A - qN_D}$$

$$G = \frac{2\varepsilon_{Si}}{2\varepsilon_{Si}A - qN_D}$$

Then (A-8) becomes: $V_{FG}^Q = D + EX_O' - E\left[F + GV_{FG}^Q\right]^{1/2} \qquad (A-9)$

which solves for V_{FG}^Q as:

$$\Delta V = -EX_1 + EX_O' + (E^2 G/2) - \sqrt{DE^2 G + (E^3 GX_O') + (E^4 G^2/4) + E^2 F} \qquad (A-10)$$

This is the desired solution for constant load capacitance C_S.

155

For the #3022 chip the total load capacitance with $V_{FG} = 10^V$ (substrate = -3V) is 0.154 pF and the floating gate active area is $1093.3 \mu m^2$.

With these parameters in the BASIC program "FGBCCD," which computes ΔV_{FG} and percent change in slope (not given here), the responses shown in Figures B1 and B2 (Appendix B) were generated.

APPENDIX B

HP9845C DESKTOP COMPUTER BASIC PROGRAM

The following pages give the HP9845C desktop computer BASIC program for several *cases* of one-dimensional CCD and floating-gate CCD approximations. The programs in most instances produce hard-copy graphics over user-specified ranges of input variables.

```
10    '   THE NAME OF THIS PROGRAM IS "SRFCCH".   IT COMPUTES THE CHANGE IN THE
          FLOATING GATE VOLTAGE, Delvfg, FOR A SURFACE N-CHANNEL DEVICE.   THE
20    '   VARIALBE PARAMETERS ARE INPUT FROM THE KEYBOARD.
30    Q=1.602E-19         !  C/e-
40    PRINT " TYPE IN THE VALUES OF THE PARAMETERS Na, Tox, Lfg, Wfg, Cs, Qsig. V
ofg, Vfb"
50    PRINT " "
60    PRINT "          Na in units of 1E15 cm^-3"
70    PRINT " "
80    PRINT "          Tox in ANGSTROMS (= # MICRONS * 10,000)"
90    PRINT " "
100   PRINT "          Lfg in MICRONS (= # ANGSTOMS * 1E-4 = # CM * 1E4)"
110   PRINT " "
120   PRINT "          Wfg in MICRONS"
130   PRINT " "
140   PRINT "          Cs in picoFARADS"
150   PRINT " "
160   PRINT "          Qsig in picoCOULOMBS"
170   PRINT " "
180   PRINT "          Vofg in VOLTS"
190   PRINT " "
200   PRINT "          Vfb in VOLTS"
210   INPUT Na,Tox,Lfg,Wfg,Cs,Qsig,Vofg,Vfb
220   Na=Na*1E15        !   cm^-3
230   Tox=Tox*1E-8      !   cm
240   Lfg=Lfg*1E-4      !   cm
250   Wfg=Wfg*1E-4      !   cm
260   Cs=Cs*1E-12       !   F
270   Afg=Lfg*Wfg*1E8   !   um^2
280   Qsigum2=Qsig/Afg  !   pC/um^2
290   Qsig=Qsig*1E-12   !   C
300   Electrons=-Qsig/Q !   e-
310   Electum2=-Qsig/(Afg*Q)   (   e-/um^2
320   Eo=8.86E-14       !   F/cm
330   Esi=11.7
340   Eox=3.9
350   Cox=Eo*Eox/Tox    !   F/cm^2
360   Afg=Lfg*Wfg       !   cm^2
370   N=Cs/(Afg*Cox)
380   Vo=Q*Na*Esi*Eo/Cox^2
390   Beta=2*Afg/Cs*((Qsig/Afg+Cox*(Vo^2+2*Vo*(Vofg-Vfb))^.5+Afg*Cox^2*Vo/Cs)
400   G1=(Qsig/Afg+Cox*SQR(Vo^2+2*Vo*(Vofg-Vfb)))^2
410   G2=Cox^2*(Vo^2+2*Vo*(Vofg-Vfb+Qsig/Afg/Cox))
420   Gamma=(Afg/Cs)^2*(G1-G2)
430   Disc=SQR(Beta^2-4*Gamma)
440   Delvfg=(-Beta+Disc)/2
450   Tox=Tox*1E8       !   Angstroms
460   Afg=Afg*1E8       !   um^2
470   Cs=Cs*1E12        !   pF
480   Qsig=Qsig*1E12    !   pC
490   Cox=Cox*1E4       !   pF/um^2
500   PRINTER IS 7,1
510   PRINT "Program: SRFCCH"
520   FLOAT 1
530   PRINT "Na (Cm^-3)";SPA(17);Na
540   FIXED 0
550   PRINT "Tox (Angstroms)";SPA(12);Tox
560   FIXED 1
570   PRINT "Afg (Microns^2)";SPA(12);Afg
580   FIXED 5
590   PRINT "Cs (Picofarads)";SPA(12);Cs
600   FIXED 2
610   PRINT "Qsig (Picocoulombs)";SPA(8);Qsig
620   FLOAT 3
630   PRINT "Qsig (picoCoulombs/um^2)";SPA(3);Qsigum2
640   FIXED 1
```

158

```
650    PRINT "Vofg (Volts)";SPA(15);Vofg,"Vfb (Volts)";SPA(2);Vfb
660    FLOAT 4
670    PRINT "Cox (Picofarads/Micron^2)";SPA(2);Cox
680    FIXED 2
690    PRINT "N (loading factor)";SPA(9);N
700    FLOAT 3
710    PRINT "Electrons";SPA(18);Electrons
720    FIXED 1
730    PRINT "Electrons/Micron^2";SPA(9);Electum2
740    FIXED 3
750    PRINT "Beta";SPA(23);Beta
760    PRINT "Gamma";SPA(22);Gamma
770    PRINT "Delvfg (Volts)";SPA(13);Delvfg
780    PRINTER IS 16
790    END
```

```
10    PRINT "The name of this program is 'GLSFCL' (General SurFace ChanneL).   It
computes    Delvfg, the change in the     "
20    PRINT "floating gate voltage Vfg for a surface N-channel device.  It also c
alculates"
30    PRINT "the partial derivative of Delvfg with respect to Qsig.  Graphical ou
tput is"
40    PRINT "optional.  An INPUT asking for '(Y/N)' means type Y for a Yes respon
se and N for"
50    PRINT "a No response.  Any response other than Y or N may result in erroneo
us operation"
60    PRINT "of the program."
70    PRINT " "
80    OPTION BASE 1
90    DIM Qsig(51),Qsigpc(51),Delvfg(51),Pdervq(51),Pvq(51)
100   !
110   !
120   PRINT "The first section of the program calculates the change in Vfg as a f
unction of"
130   PRINT "Qsig for N values of Qsig defined on an interval Qsig(1) to Qsig(N)
via a"
140   PRINT "FOR-NEXT loop.  A single point (N=1) computation is allowed.  Press
CONTINUE to"
150   PRINT "access the first section of the program."
160   PAUSE
170   PRINT LIN(4)
180   Q=1.602E-19          !  C
190   PRINT "TYPE IN THE VALUES OF THE PARAMETERS Na, Tox, Lfg, Wfg, Cs, Vofg, Vf
b"
200   PRINT " "
210   PRINT "              Na in units of 1E15 cm^-3"
220   PRINT " "
230   PRINT "              Tox in ANGSTROMS (= # MICRONS * 10,000)"
240   PRINT " "
250   PRINT "              Lfg in MICRONS (= # ANGSTOMS * 1E-4 = # CM * 1E4)"
260   PRINT " "
270   PRINT "              Wfg in MICRONS"
280   PRINT " "
290   PRINT "              Cs in picoFARADS"
300   PRINT " "
310   PRINT "              Vofg in VOLTS"
320   PRINT " "
330   PRINT "              Vfb in VOLTS"
340   INPUT Na,Tox,Lfg,Wfg,Cs,Vofg,Vfb
350   Na=Na*1E15            !  cm^-3
360   Afg=Lfg*Wfg           !  um^2
370   PRINTER IS 7,1
380   PRINT "Program: GLSFCL"
390   FLOAT 1
400   PRINT "Na (Cm^-3)";SPA(12);Na
410   FIXED 0
420   PRINT "Tox (Angstroms)";SPA(7);Tox
430   FIXED 1
440   PRINT "Afg (Microns^2)";SPA(7);Afg
450   FIXED 5
460   PRINT "Cs (Picofarads)";SPA(7);Cs
470   FIXED 1
480   PRINT "Vofg (Volts)";SPA(10);Vofg,"Vfb (Volts)";SPA(2);Vfb
490   Tox=Tox*1E-8          !  cm
500   Lfg=Lfg*1E-4          !  cm
510   Wfg=Wfg*1E-4          !  cm
520   Cs=Cs*1E-12           !  F
530   Eo=8.86E-14           !  F/cm
540   Esi=11.7
550   Eox=3.9
560   Cox=Eo*Eox/Tox        !  F/cm^2
570   Afg=Lfg*Wfg           !  cm^2
```

160

```
580    Lf=Cs/(Afg*Cox)
590    FLOAT 4
600    PRINT "Cox (F/cm^2)";SPA(10);Cox
610    FIXED 2
620    PRINT "LF (loading factor)";SPA(3);Lf
630    PRINT LIN(2)
640    PRINTER IS 16
650    Vo=Q*Na*Esi*Eo/Cox^2      ! V
660    INPUT "N,Qsig(1) pC ,Qsig(N) pC ",N,Qsiglo,Qsighi
670    REDIM Qsig(N),Qsigpc(N),Delvfg(N),Pdervq(N),Pvq(N)
680    Qsiglo=Qsiglo*1E-12      ! C
690    Qsighi=Qsighi*1E-12      ! C
700    PRINT "Qsig (pC)","Delvfg (V)"
710    PRINT " "
720          FOR I=1 TO N STEP 1
730    IF N=1 THEN Qsig(I)=Qsiglo      ! C
740    IF N=1 THEN 760
750    Qsig(I)=Qsiglo+(Qsighi-Qsiglo)*(I-1)/(N-1)      ! C
760    Beta=2*Afg/Cs*(Qsig(I)/Afg+Cox*(Vo^2+2*Vo*(Vofg-Vfb))^.5+Afg*Cox^2*Vo/Cs  !V
770    G1=(Qsig(I)/Afg+Cox*SQR(Vo^2+2*Vo*(Vofg-Vfb)))^2      ! C^2 cm^4
780    G2=Cox^2*(Vo^2+2*Vo*(Vofg-Vfb+Qsig(I)/Afg/Cox))      ! C^2/cm^4
790    Gamma=(Afg/Cs)^2*(G1-G2)      ! V^2
800    Disc=SQR(Beta^2-4*Gamma)      ! V
810    Delvfg(I)=(-Beta+Disc)/2      ! V
820    FIXED 3
830    Qsigpc(I)=Qsig(I)*1E12      ! pC
840    PRINT Qsigpc(I),Delvfg(I)
850          NEXT I
860    PRINT "End of Output"
870    !
880    !
890    PRINT LIN(2)
900    PRINT "Press CONTINUE to access the next section of the program which calcu
lates the"
910    PRINT "partial derivative of Delvfg with respect to Qsig (the slope of the
Delvfg vs"
920    PRINT "Qsig graph) using the parameters and Qsig array from the previous se
ction."
930    PAUSE
940    PRINT LIN(5)
950    PRINT "Qsig pC","Pdervq (in units of 1E+12 V/C)"
960          FOR I=1 TO N STEP 1
970    Beta=2*Afg/Cs*(Qsig(I)/Afg+Cox*(Vo^2+2*Vo*(Vofg-Vfb))^.5+Afg*Cox^2*Vo/Cs  !V
980    G1=(Qsig(I)/Afg+Cox*SQR(Vo^2+2*Vo*(Vofg-Vfb)))^2      ! C^2 cm^4
990    G2=Cox^2*(Vo^2+2*Vo*(Vofg-Vfb+Qsig(I)/Afg/Cox))      ! C^2/cm^4
1000   Gamma=(Afg/Cs)^2*(G1-G2)      ! V^2
1010   X=1/SQR(Beta^2-4*Gamma)      ! V^-1
1020   Y=(Afg/Cs)^2*(2*(Qsig(I)/Afg+Cox*(Vo^2+2*Vo*(Vofg-Vfb))^.5-2*Vo*Cox)/V^2 C
1030   Pdervq(I)=1/Afg*(.5*(-1+Beta*X)*(2*Afg/Cs)-X*Y)      ! V/C
1040   Pvq(I)=Pdervq(I)/1E12      ! V/C
1050   FIXED 3
1060   PRINT Qsigpc(I);SPA(10);Pvq(I)
1070         NEXT I
1080   PRINT "End of Output"
1090   PRINT LIN(10)
1100   INPUT "Do you desire a graphical output of this data? (Y N)",Ans1$
1110   IF Ans1$="N" THEN 1760
1120   INPUT "Do you want the graphical output on the 9872A or 9845C? (72A/45C)",A
ns2$
1130   IF Ans2$="45C" THEN 1790
1140   ! 9872A PLOTTER ROUTINE--------------------------------------------------
1150   PRINT "BE SURE THERE IS PAPER ON THE PLOTTER  then press CONTINUE"
1160   PAUSE
1170   PRINT PAGE
1180   INPUT "Do you need to draw new axes? (Y/N)",Ans3$
1190   IF Ans3$="N" THEN 1580
```

```
1200 PLOTTER IS 7,5,"9872A"
1210 LIMIT 20,260,20,190          ! mm
1220 LOCATE 10,120,10,97          ! GDUs
1230 INPUT "ENTER THE MAX. VALUE TO BE PLOTTED ON THE X AXIS,Y AXIS",Xmax,Ymax
1240 SCALE 0,-Xmax,0,Ymax
1250 PEN 1          ! black
1260 LINE TYPE 1
1270 AXES 1/10,1/10,0,0,10,10,2
1280 ! ---------------------------- LABEL ROUTINE
1290 ! -----  X axis title          !
1300 DEG                            !
1310 MOVE .5,-.2                    !
1320 CSIZE 5,.5                     !
1330 LORG 6                         !
1340 LDIR 0                         !
1350 LABEL USING "K";"Qsig (pC)"    !
1360 ! -----  X axis numbers        !
1370 CSIZE 3,.5                     !
1380 LORG 9                         !
1390 FOR A=0 TO -Xmax STEP .1       !
1400 MOVE A,-.1                     !
1410 LABEL USING "K";"-";A          !
1420 NEXT A                         !
1430 ! -----  Y axis title          !
1440 MOVE -.08,1.5                  !
1450 CSIZE 4,.5                     !
1460 LORG 6                         !
1470 LDIR 90                        !
1480 LABEL USING "K";"Delvfg (V)"   !
1490 ! -----  Y axis numbers        !
1500 CSIZE 3,.5                     !
1510 LORG 8                         !
1520 LDIR 0                         !
1530 FOR A=0 TO Ymax STEP 1         !
1540 MOVE -.01,A                    !
1550 LABEL USING "K";A              !
1560 NEXT A                         !
1570 ! ----------------------------
1580 PEN 4          ! red
1590 Qsigpc(1)=-Qsigpc(1)          ! pC
1600 PLOT Qsigpc(1),Delvfg(1),-2
1610 FOR J=2 TO N STEP 1
1620 Qsigpc(J)=-Qsigpc(J)          ! pC
1630 PLOT Qsigpc(J),Delvfg(J),-1
1640 NEXT J
1650 ! -----  tox label
1660 Tox=Tox*1E8                   ! Angstroms
1670 PEN 1          ! black
1680 CSIZE 2,.5
1690 LORG 2
1700 MOVE -Xmax+.0005,Delvfg(N)
1710 LDIR 0
1720 LABEL USING "K";"tox=";Tox;"A"
1730 ! -----
1740 PENUP
1750 PEN 0
1760 INPUT "Do you wish to run the program again with a new set of parameters? (
Y/N)",Ans4$
1770 IF Ans4$="Y" THEN 180
1780 STOP
1790 ! 9845C CRT GRAPHICS ROUTINE-----------------------------------------------
1800 PRINT PAGE
1810 INPUT "Do you need to draw new axes? (Y/N)",Ans5$
1820 IF Ans5$="N" THEN 1940
1830 PLOTTER IS "GRAPHICS"
1940 ! GRAPHICS
```

162

```
1850 LIMIT 0,140,0,140              ! mm
1860 PEN 2        ! red
1870 LINE TYPE 1
1880 FRAME
1890 LOCATE 10,97,10,97             ! GDUs
1900 INPUT "ENTER THE MAX. VALUE TO BE PLOTTED ON THE X AXIS,Y AXIS",Xmax,Ymax
1910 SCALE 0,-Xmax,0,Ymax
1920 PEN 1        ! white
1930 AXES 1/10,1/10,0,0,10,10,2
1940 Qsigpc(1)=-Qsigpc(1)           ! pC
1950 PLOT Qsigpc(1),Delvfg(1),-2
1960 FOR J=2 TO N STEP 1
1970 Qsigpc(J)=-Qsigpc(J)           ! pC
1980 PLOT Qsigpc(J),Delvfg(J),-1
1990 NEXT J
2000 PENUP
2010 PEN 0
2020 INPUT "Do you wish to clear the graphics display? (Y/N)",Ans6$
2030 IF Ans6$="N" THEN 2060
2040 SETGU
2050 GCLEAR
2060 INPUT "Do you wish to run the program again with a new set of parameters?
Y/N)",Ans7$
2070 IF Ans7$="Y" THEN 180
2080 STOP
2090 END
```

```
10    PRINT "The name of this program is 'BURCHL'.  It computes the maximum burie
d channel   potential, Vz, of a buried N-channel device if we know Qs and Vg OR"
20    PRINT "it finds the channel charge, Qs, of the same device if we know Vz an
d Vg."
30    FLOAT 4
40    PRINT " "
50    PRINT " "
60    PRINT "TYPE IN THE VALUES OF THE PARAMETERS Nd, Na, Tox, Xn, Vg, Vfb"
70    PRINT " "
80    PRINT "donor concentration, Nd, in units of  E15 / CENTIMETER^3"
90    PRINT " "
100   PRINT "acceptor concentration, Na, in units of  E15 / CENTIMETER^3"
110   PRINT " "
120   PRINT "oxide thickness, Tox, in ANGSTROMS (= # Microns * 10,000)"
130   PRINT " "
140   PRINT "channel depth, Xn, in MICRONS (= # Angstroms * 1E-4 = # Cm * 1E4)"
150   PRINT " "
160   PRINT "gate voltage, Vg, in VOLTS"
170   PRINT " "
180   PRINT "flat-band voltage, Vfb, in VOLTS"
190   INPUT Nd,Na,Tox,Xn,Vg,Vfb
200   Nd=Nd*1E15
210   Na=Na*1E15
220   Tox=Tox*1E-8    !   cm
230   Xn=Xn*1E-4      !   cm
240   Q=1.602E-19     !   cb
250   Eo=8.86E-14     !   F/cm
260   Eox=3.9
270   Esi=11.7
280   Cox=Eo*Eox/Tox  !  F/cm^2
290   B=Q*Nd*(Nd+Na)/(2*Eo*Esi*Na)
300   Vgprim=Vg-Vfb
310   PRINT " "
320   PRINT " "
330   PRINT " "
340   PRINT " "
350   PRINT "To compute Vz, type 'Vz  and press CONTINUE; to compute Qs, type 'Qs
'  and press CONTINUE; to stop, press STOP"
360   INPUT Section$
370   IF Section$="Vz" THEN 400
380   IF Section$="Qs" THEN 760
390   ! ---------------------------------------------------------------
400   ! Vz(Qs,Vg)
410   INPUT "Enter the value of Qs in picoCOULOMBS / MICRON^2",Qs
420   Qs=Qs*1E-4       !   C/cm^2
430   Xz=Qs/(Q*Nd)     !   cm
440   X=Xn-Xz          !   cm
450   Xoprim=Eo*Esi*(2*B*X*Cox+Q*Nd)/(Cox*(2*B*Eo*Esi-Q*Nd))
460   X1=Xoprim-SQR(Xoprim^2-2*Eo*Esi*(B*X^2-Vgprim)/(2*B*Eo*Esi-Q*Nd))
470   PRINT "X1=",X1
480   IF X1<0 THEN 700
490   Vz=B*(X-X1)^2
500   Tox=Tox*1E8 !  Angstroms
510   Xn=Xn*1E4       !  Microns
520   Cox=Cox*1E4 !  pF/um^2
530   Qs=Qs*1E4   !  pC/um^2
540   PRINTER IS 7,1
550   PRINT "Program: BURCHL    Section: Vz(Qs,Vg)"
560   PRINT "Nd (Cm^-3)",Nd
570   PRINT "Na (Cm^-3)",Na
580   PRINT "Tox (Angstroms)",Tox
590   PRINT "Qs (Picocoulombs/Micron^2)",Qs
600   PRINT "Xn (Microns)",Xn
610   PRINT "Vg (Volts)",Vg
620   PRINT "Vfb (Volts)",Vfb
630   PRINT "Cox (Picofarads/Micron^2)",Cox
```

164

```
640   PRINT "Vz (Volts)=",Vz
650   PRINTER IS 16
660   Tox=Tox*1E-8   !   cm
670   Xn=Xn*1E-4     !   cm
680   Cox=Cox*1E-4   !   F/cm^2
690   GOTO 350
700   INPUT "X1 < 0 (not allowed)  Do you want to enter new data?  (Y/N)",A$
710   IF A$="Y" THEN 60
720   PRINT "I won't accept no for an answer!  Press CONTINUE"
730   PAUSE
740   GOTO 60
750   ! -------------------------------------------------------------------
760   ! Qs(Vz,Vg)
770   INPUT "Enter the value of Vz in VOLTS",Vz
780   A=2*Eo*Esi/(2*B*Eo*Esi-Q*Nd)
790   C=SQR(Vz/B)-Xn
800   D=A*B
810   E=A*Vgprim
820   F=Eo*Esi/(Cox*(2*B*Eo*Esi-Q*Nd))
830   G=C+2*B*F*Cox*Xn+F*Q*Nd
840   H=E-D*Xn^2
850   I=(1-2*B*Cox*F)/(Q*Nd)
860   J=2*B*F*Cox*Xn+F*Q*Nd
870   K=2*B*F*Cox/(Q*Nd)
880   L=G^2-J^2-H
890   M=2*K*J+2*G*I-2*D*Xn/(Q*Nd)
900   N=I^2-K^2+D/(Q*Nd)^2
910   P=M/N
920   R=L/N
930   Disc=P^2-4*R
940   PRINT "Disc=",Disc
950   IF Disc>=0 THEN 980
960   PRINT " Disc = P^2-4*R is < 0 =>SQR(Disc) is not real.  ENTER NEW
DATA "
970   GOTO 60
980   Qs=(-P-SQR(Disc))/2    !  C/cm^2
990   Qsabs=ABS(Qs)
1000  IF Qsabs<=1.35000000000E-17 THEN Qs=0
1010  Tox=Tox*1E8    !  Angstroms
1020  Xn=Xn*1E4      !  Microns
1030  Cox=Cox*1E4    !  pF/um^2
1040  Qs=Qs*1E4      !  pC/um^2
1050  PRINTER IS 7,1
1060  PRINT "Program: BURCHL    Section: Qs(Vz,Vg)"
1070  PRINT "Nd (Cm^-3)",Nd
1080  PRINT "Na (Cm^-3)",Na
1090  PRINT "Tox (Angstroms)",Tox
1100  PRINT "Qs (Picocoulombs/Micron^2)",Qs
1110  Qs=Qs*1E-4     !  C/cm^2
1120  PRINT "Qs (Coulombs/cm^2)",Qs
1130  PRINT "Xn (Microns)",Xn
1140  PRINT "Vg (Volts)",Vg
1150  PRINT "Vfb (Volts)",Vfb
1160  PRINT "Cox (Picofarads/Micron^2)",Cox
1170  PRINT "Vz (Volts)=",Vz
1180  PRINT "Disc=",Disc
1190  X1=Xoprim-SQR(Xoprim^2-2*Eo*Esi*(B*X^2-Vgprim)/(2*B*Eo*Esi-Q*Nd))
1200  PRINT "X1=",X1
1210  PRINTER IS 16
1220  IF X1<0 THEN 1240
1230  STOP
1240  INPUT "X1 < 0; Do you wish to enter new data?  (Y/N)",Choice$
1250  IF Choice$="Y" THEN 60
1260  STOP
1270  END
```

```
10    PRINT "The name of this program is 'FGBCCD' (Floating Gate/Buried channel"
20    PRINT "CCD).  It computes:"
30    PRINT "     (1) the change, Delvfg (=Vqfg - Vofg), in the floating gate"
40    PRINT "         voltage, Vfg, of a buried N-channel CCD"
50    PRINT "     (2) the partial derivative of Delvfg with respect to Qsignal"
60    PRINT "     (3) the percent change in the partial derivative"
70    PRINT "     (4) the maximum channel potential, Vz"
80    PRINT "The derivation of the equations may be found in the 21 May 1981"
90    PRINT "memo by James Joseph entitled 'Buried-Channel Floating-Gate Sensing"
100   PRINT "With Constant Capacitive Load'.  The required parameters are"
110   PRINT "entered via INPUT statements."
120   PRINT "Note: We calculate the quantity Delvfg which by physical argument"
130   PRINT "must be a negative number since Vofg is always greater than Vqfg"
140   PRINT "for this type of system.  However, rather than label the Delvfg"
150   PRINT "axis with negative numbers, we plot the absolute value of Delvfg"
160   PRINT "and label the vertical axis '-(delta)Vfg'.  Also, Xm must be a"
170   PRINT "positive number since it is simply the width of that portion of the"
180   PRINT "depletion region which has been charge-neutralized by the signal"
190   PRINT "charge.  Therefore, although it is defined as Qs/(Q*Nd) in the"
200   PRINT "memo, we must define it as ABS(Qsig/(Afg*Q*Nd)) since our Qsig"
210   PRINT "values are negative."
220   PRINT "(press CONTINUE)"
230   PAUSE
240   PRINT PAGE
250   INPUT "Is this program being run on the HP9835A or HP9845C? (A/C)",Computer
$
260   IF Computer$="A" THEN 290
270   IF Computer$="C" THEN 290
280   GOTO 250
290   OPTION BASE 1
300   DIM Qsig(101),Qsigpc(101),Electrons(101),Megaelectrons(101),Delvfg(101),Pde
rvq(101),Pvq(101),Pderpercent(101),Vz(101),Xvar(101),Yvar(101)
310   !
320   !
330   PRINT "The first section of the program calculates Delvfg as a function of"
340   PRINT "Qsig for N values of Qsig defined on an interval Qsig(1) to Qsig(N)"
350   PRINT "via a FOR-NEXT loop.  A single point (N=1) computation is allowed."
360   PRINT "The maximum value of N is 101 unless the initial DIM statement is"
370   PRINT "revised before running the program."
380   PRINT "Press CONTINUE to access the first section of the program."
390   PAUSE
400   PRINT LIN(4)
410   Q=1.602E-19       ! C
420   Eo=8.86E-14       ! F/cm
430   Esi=11.7
440   Eox=3.9
450   PRINT "ENTER THE VALUE OF:"
460   PRINT " "
470   PRINT "            Na in units of 1E15 cm^-3"
480   PRINT " "
490   PRINT "            Nd in units of 1E15 cm^-3"
500   PRINT " "
510   PRINT "            Tox in ANGSTROMS (= # MICRONS * 10,000)"
520   PRINT " "
530   PRINT "            Xj in MICRONS"
540   PRINT " "
550   PRINT "            Lfg in MICRONS (= # ANGSTROMS * 1E-4 = # CM * 1E4)"
560   PRINT " "
570   PRINT "            Wfg in MICRONS"
580   PRINT " "
590   PRINT "            Vofg in VOLTS"
600   PRINT " "
610   PRINT "            Vfb in VOLTS"
620   INPUT "Na, Nd, Tox, Xj",Na,Nd,Toxang,Xjum
630   ! Na=4
640   Na=Na*1E15
```

166

```
650   ! Nd=50
660   Nd=Nd*1E15
670   ! Toxang=200
680   Tox=Toxang*1E-8
690   ! Xjum=.3
700   Xj=Xjum*1E-4
710   INPUT "Lfg, Wfg, Vofg, Vfb",Lfgum,Wfgum,Vofg,Vfb
720   ! Lfgum=40
730   Lfg=Lfgum*1E-4
740   ! Wfgum=20
750   Wfg=Wfgum*1E-4
760   Afgum2=Lfgum*Wfgum
770   Afg=Lfg*Wfg
780   ! Vofg=10
790   ! Vfb=-.9
800   Cox=Eo*Eox/Tox        ' F/cm^2
810   Coxpf=Cox*1E12        ! pF/cm^2
820   INPUT "Which do you know: LF or CS in pF? (LF/CS) , Enter its value.",Ans$,
Value
830   IF Ans$="LF" THEN 860
840   IF Ans$="CS" THEN 910
850   GOTO 820
860   Lf=Value
870   ! Lf=.1
880   Cs=Lf*Cox*Afg
890   Cspf=Cs*1E12
900   GOTO 950
910   Cspf=Value
920   ! Cspf=.25130
930   Cs=Cspf*1E-12
940   Lf=Cs/(Afg*Cox)
950   IF Computer$="A" THEN PRINTER IS 7,1
960   IF Computer$="C" THEN PRINTER IS 0
970   PRINT "Program: FGBCCD"
980   FLOAT 1
990   PRINT "Na (Cm^-3)";SPA(12);Na
1000  FLOAT 1
1010  PRINT "Nd (Cm^-3)";SPA(12);Nd
1020  FIXED 0
1030  PRINT "Tox (Angstroms)";SPA(7);Toxang
1040  FIXED 1
1050  PRINT "Xj (Microns)";SPA(10);Xjum
1060  FIXED 1
1070  PRINT "Afg (Microns^2)";SPA(7);Afgum2
1080  FIXED 1
1090  PRINT "Vofg (Volts)";SPA(10);Vofg,"Vfb (Volts)";SPA(2);Vfb
1100  FLOAT 4
1110  PRINT "Cox (F/cm^2)";SPA(10);Cox
1120  FIXED 2
1130  PRINT "LF (loading factor)";SPA(3);Lf
1140  FIXED 5
1150  PRINT "CS (Picofarads)";SPA(7);Cspf
1160  PRINTER IS 16
1170  INPUT "Which do you know: Qsig in pC OR number of e-? (Q/E)",Charge$
1180  IF Charge$="E" THEN 1210
1190  IF Charge$="Q" THEN 1370
1200  GOTO 1170
1210  INPUT "N, Electrons(1), Electrons(N)",N,Electronslo,Electronshi
1220  ! N=11
1230  ! Electronslo=0
1240  ! Electronshi=6E6
1250  Qsiglopc=-Electronslo*Q*1E12
1260  Qsighipc=-Electronshi*Q*1E12
1270  IF Computer$="A" THEN PRINTER IS 7,1
1280  IF Computer$="C" THEN PRINTER IS 0
1290  FIXED 0
```

167

```
1300 PRINT "N";SPA(12);N
1310 FIXED 1
1320 PRINT "Qsig(1) pC",Qsiglopc,"Qsig(N) pC",Qsighipc
1330 FLOAT 3
1340 PRINT "Electrons(1)",Electronslo,"Electrons(N)",Electronshi
1350 PRINTER IS 16
1360 GOTO 1520
1370 INPUT "N,Qsig(1) pC ,Qsig(N) pC ",N,Qsiglopc,Qsighipc
1380 ! N=11
1390 ! Qsiglopc=0
1400 ! Qsighipc=-1
1410 Electronslo=-Qsiglopc*1E-12/Q
1420 Electronshi=-Qsighipc*1E-12/Q
1430 IF Computer$="A" THEN PRINTER IS 7,1
1440 IF Computer$="C" THEN PRINTER IS 0
1450 FIXED 0
1460 PRINT "N";SPA(12);N
1470 FIXED 1
1480 PRINT "Qsig(1) pC",Qsiglopc,"Qsig(N) pC",Qsighipc
1490 FLOAT 3
1500 PRINT "Electrons(1)",Electronslo,"Electrons(N)",Electronshi
1510 PRINTER IS 16
1520 REDIM Qsig(N),Qsigpc(N),Electrons(N),Megaelectrons(N),Delufg(N),Pderuq(N),P
uq(N),Pderpercent(N),Vz(N),Xvar(N),Yvar(N)
1530 Qsiglo=Qsiglopc*1E-12
1540 Qsighi=Qsighipc*1E-12
1550 PRINT "Qsig (pC)","Millions of e-",CHR$(198);"Vfg (V)"
1560 PRINT " "
1570 Nn=N          ! so we can redefine Nn without changing N
1580 A=Q*Nd*(Nd+Na)/(2*Eo*Esi*Na)
1590 B=-1/(Afg*Q*Nd)        ! PD of Xm wrt Qsig
1600 Y=Eo*Esi/(2*A*Eo*Esi-Q*Nd)
1610 C=-Y*2*A        ! PD of Xoprime wrt Xm
1620 Xo=Y*(2*A*Xj*Cox+Q*Nd)/Cox
1630 X1=Xo-SQR(Xo^2-2*Y*(A*Xj^2-Vofg+Vfb))
1640 D=Vofg-Q*Nd*X1*Afg/Cs
1650 E=Q*Nd*Afg/Cs
1660 G=2*Y
1670          FOR I=1 TO Nn STEP 1
1680 IF Nn=1 THEN Qsig(I)=Qsiglo
1690 IF Nn=1 THEN 1710
1700 Qsig(I)=Qsiglo+(Qsighi-Qsiglo)*(I-1)/(Nn-1)          ! C
1710 Electrons(I)=-Qsig(I)/Q          ! e-
1720 Xm=ABS(Qsig(I)*B)          ! >0
1730 Xoprime=Y*(2*A*(Xj-Xm)*Cox+Q*Nd)/Cox
1740 F=Xoprime^2-2*Y*(A*(Xj-Xm)^2+Vfb)
1750 IF F+D*G+E+G*Xoprime+E^2*G^2/4<0 THEN 4800
1760 Delufg(I)=-E*(X1-Xoprime-E*G/2+SQR(F+D*G+E*G*Xoprime+E^2*G^2/4))          ! V
1770 Vqfg=Vofg+Delufg(I)
1780 X1prime=Xoprime-SQR(Xoprime^2-2*Y*(A*(Xj-Xm)^2-Vqfg+Vfb))
1790 IF X1prime<0 THEN 4950
1800 FIXED 3
1810 Qsigpc(I)=Qsig(I)*1E12
1820 Megaelectrons(I)=Electrons(I)*1E-6 ! millions of e-,need array for plotting
1830 PRINT Qsigpc(I),Megaelectrons(I),Delufg(I)
1840          NEXT I
1850 PRINT "End of Output"
1860 Qpcum2=Qsigpc(Nn)/Afgum2
1870 Kelectronsum2=Electrons(Nn)/Afgum2*1E-3
1880 IF Computer$="A" THEN PRINTER IS 7,1
1890 IF Computer$="C" THEN PRINTER IS 0
1900 PRINT "Qsig(Nn)/Afg (maximum allowable signal charge per gate area) pC/um^2
    ";Qpcum2
1910 PRINT "Thousands of e- per um^2 corresponding to Qsig(Nn)     ";Kelectronsum
2
1920 PRINT LIN(2)        ! seperates hardcopy output
```

```
1930 PRINTER IS 16
1940 !
1950 !
1960 PRINT "The second section of the program calculates:"
1970 PRINT "    (1) the partial derivative of Delvfg with respect to Qsig"
1980 PRINT "    (2) the percent change in the partial derivative"
1990 PRINT "    (3) the maximum channel potential, Vz"
2000 PRINT "using the parameters and Qsig array from the previous section."
2010 PRINT "Press CONTINUE to access the second section of the program."
2020 PAUSE
2030 PRINT LIN(6)
2040 PRINT "Qsig pC","Slope(x1E11 V/C)","% slope change","Vz"
2050     FOR I=1 TO Nn STEP 1
2060 Xm=ABS(Qsig(I)*B)
2070 Xoprime=Y*(2*A*(Xj-Xm)*Cox+Q*Nd)/Cox
2080 F=Xoprime^2-2*Y*(A*(Xj-Xm)^2+Vfb)
2090 H=(D*G+F)*E^2+G*Xoprime*E^3+G^2*E^4/4
2100 Vqfg=Vofg+Delvfg(I)
2110 X1prime=Xoprime-SQR(Xoprime^2-2*Y*(A*(Xj-Xm)^2-Vqfg+Vfb))
2120 Pdervq(I)=E*B*C-.5*E^2*B*C/H^.5*(E*G+2*Xoprime-2*(Xj-Xm))
2130 Pvq(I)=Pdervq(I)*1E-11        ! need array for plotting
2140 Pderpercent(I)=(Pdervq(I)-Pdervq(1))/Pdervq(1)*100
2150 Vz(I)=A*(Xj-Xm-X1prime)^2
2160 IF Vz(I)<Vqfg THEN 5090
2170 FIXED 3
2180 PRINT Qsigpc(I),Pvq(I),Pderpercent(I),Vz(I)
2190     NEXT I
2200 PRINT "End of Output"
2210 PRINT LIN(2)
2220 !
2230 !
2240 INPUT "Do you desire a graphical output of this data? (Y/N)",Ans1$
2250 IF Ans1$="Y" THEN 2280
2260 IF Ans1$="N" THEN 3640
2270 GOTO 2240
2280 PRINT "You may choose which quantity you want on the horizontal axis:"
2290 PRINT "    i) Qsig in pC   CODE: Q"
2300 PRINT "    ii) Number of e-   CODE: E"
2310 PRINT "Enter the code of the variable for the X axis."
2320 INPUT "XCODE?",Xcode$
2330 PRINT PAGE
2340 PRINT "You may choose which quantity you want on the vertical axis:"
2350 PRINT "    i) Delta Vfg  CODE: DV"
2360 PRINT "    ii) Partial Derivative  CODE: PD"
2370 PRINT "    iii) % change in PD  CODE: PC"
2380 PRINT "    iv) Vz  CODE: VZ"
2390 PRINT "Enter the code of the variable for the Y axis."
2400 INPUT "YCODE?",Ycode$
2410 PRINT PAGE
2420 PRINT "XCODE = ";Xcode$,"YCODE = ";Ycode$
2430     FOR I=1 TO Nn STEP 1
2440 IF Xcode$="Q" THEN Xvar(I)=Qsigpc(I)
2450 IF Xcode$="E" THEN Xvar(I)=Megaelectrons(I)
2460 IF Ycode$="DV" THEN Yvar(I)=Delvfg(I)
2470 IF Ycode$="PD" THEN Yvar(I)=Pvq(I)
2480 IF Ycode$="PC" THEN Yvar(I)=Pderpercent(I)
2490 IF Ycode$="VZ" THEN Yvar(I)=Vz(I)
2500 Xvarmin=ABS(Xvar(1))
2510 Xvarmax=ABS(Xvar(1))
2520 Yvarmin=ABS(Yvar(1))
2530 Yvarmax=ABS(Yvar(1))
2540 IF I=1 THEN 2590
2550 IF ABS(Xvar(I))<ABS(Xvar(I-1)) THEN Xvarmin=ABS(Xvar(I))
2560 IF ABS(Xvar(I))>ABS(Xvar(I-1)) THEN Xvarmax=ABS(Xvar(I))
2570 IF ABS(Yvar(I))<ABS(Yvar(I-1)) THEN Yvarmin=ABS(Yvar(I))
2580 IF ABS(Yvar(I))>ABS(Yvar(I-1)) THEN Yvarmax=ABS(Yvar(I))
```

169

```
2590       NEXT I
2600 PRINT "Xvarmin= ";Xvarmin,"Xvarmax= ";Xvarmax
2610 PRINT "Yvarmin= ";Yvarmin,"Yvarmax= ";Yvarmax
2620 INPUT "Do you want the graphical output on the 9872A or 9845C? (72A/45C)",A
ns2$
2630 IF Ans2$="72A" THEN 2660
2640 IF Ans2$="45C" THEN 3650
2650 GOTO 2620
2660 ! 9872A PLOTTER ROUTINE----------------------------------------------------
2670 PRINT "BE SURE THERE IS PAPER ON THE PLOTTER  then press CONTINUE"
2680 BEEP
2690 PAUSE
2700 PRINT PAGE
2710 INPUT "Do you need to draw new axes? (Y/N)",Ans3$
2720 IF Ans3$="Y" THEN 2750
2730 IF Ans3$="N" THEN 3440
2740 GOTO 2710
2750 IF Computer$="C" THEN PLOTTER IS 13,"GRAPHICS"
2760 IF Computer$="C" THEN PLOTTER 13 IS OFF
2770 PLOTTER IS 7,5,"9872A"
2780 PLOTTER 7,5 IS ON
2790 LIMIT 20,260,20,190              ! mm
2800 LOCATE 10,97,10,97               ! GDUs
2810 Xmin=INT(Xvarmin)
2820 Xmax=INT(Xvarmax)+1
2830 Ymin=INT(Yvarmin)
2840 Ymax=INT(Yvarmax)+1
2850 SCALE Xmin,Xmax,Ymin,Ymax
2860 PEN 1          ! black
2870 LINE TYPE 1
2880 AXES 1/10,1/10,Xmin,Ymin,10,10,2
2890 ! ------------------------------ AXIS LABELING ROUTINE
2900 ! ----- X axis title                        !
2910 DEG                                          !
2920 MOVE Xmin+.5*(Xmax-Xmin),Ymin-.067*(Ymax-Ymin)!
2930 CSIZE 5,.5                                   !
2940 LORG 6                                       !
2950 LDIR 0                                       !
2960 IF Xcode$="Q" THEN 2980                      !
2970 IF Xcode$="E" THEN 3000                      !
2980 LABEL "Qsig (pC)"                            !
2990 GOTO 3020                                    !
3000 LABEL "Electrons (x1E+6)"                    !
3010 GOTO 3020                                    !
3020 ! ----- X axis numbers                       !
3030 CSIZE 3,.5                                   !
3040 LORG 6                                       !
3050 MOVE 0,Ymin-.03*(Ymax-Ymin)                  !
3060 LABEL USING "K";"0"                          !
3070 FOR A=.5 TO Xmax STEP .5                     !
3080 MOVE A,Ymin-.03*(Ymax-Ymin)                  !
3090 IF Xcode$="Q" THEN 3110                      !
3100 IF Xcode$="E" THEN 3130                      !
3110 LABEL USING "K";"-";A                        !
3120 GOTO 3150                                    !
3130 LABEL USING "K";A                            !
3140 GOTO 3150                                    !
3150 NEXT A                                       !
3160 ! ----- Y axis title                         !
3170 MOVE Xmin-.07*(Xmax-Xmin),Ymin+.5*(Ymax-Ymin) !
3180 CSIZE 4,.5                                   !
3190 LDIR 90                                      !
3200 IF Ycode$="DV" THEN 3240                     !
3210 IF Ycode$="PD" THEN 3260                     !
3220 IF Ycode$="PC" THEN 3280                     !
3230 IF Ycode$="VZ" THEN 3340                     !
```

```
3240 LABEL "-";CHR$(198);"Vfg (V)"                      !
3250 GOTO 3350                                          !
3260 LABEL "SLOPE (x1E+11 V/C)"                         !
3270 GOTO 3350                                          !
3280 IF Pderpercent(Nn/2)>0 THEN 3300                   !
3290 IF Pderpercent(Nn/2)<0 THEN 3320                   !
3300 LABEL "% CHANGE IN SLOPE"                          !
3310 GOTO 3350                                          !
3320 LABEL "- % CHANGE IN SLOPE"                        !
3330 GOTO 3350                                          !
3340 LABEL "Vz (V)"                                     !
3350 ! -----  Y axis numbers                           !
3360 CSIZE 3,.5                                         !
3370 LORG 8                                             !
3380 LDIR 0                                             !
3390 FOR A=0 TO Ymax STEP 1                             !
3400 MOVE Xmin-.003*(Xmax-Xmin),A                       !
3410 LABEL USING "K,X";A                                !
3420 NEXT A                                             !
3430 ! ------------------------------------------------
3440 PEN 4        ! red
3450 PLOT ABS(Xvar(1)),ABS(Yvar(1)),-2
3460 FOR J=2 TO Nn STEP 1
3470 PLOT ABS(Xvar(J)),ABS(Yvar(J)),-1
3480 NEXT J
3490 ! -----  locus label
3500 PEN 1        ! black
3510 CSIZE 2.5,.5
3520 LORG 2
3530 MOVE ABS(Xvar(Nn))+.01,ABS(Yvar(Nn))
3540 ! LABEL USING "X,K";"tox=";Toxang;CHR$(208);" LF=";Lf
3550 LABEL "LF= ";Lf
3560 ! -----
3570 PRINT "LETTER command activated.  Press CONTINUE to deactivate.  DO NOT"
3580 PRINT "LEAVE THE PLOTTER IN LETTER MODE (pen will dry out)!!!"
3590 CSIZE 2.5,.5
3600 LETTER
3610 PRINT PAGE
3620 PENUP
3630 PEN 0
3640 GOTO 4710
3650 ! 9845C CRT GRAPHICS ROUTINE------------------------------------------------
3660 PRINT PAGE
3670 INPUT "Do you need to draw new axes? (Y/N)",Ans4$
3680 IF Ans4$="Y" THEN 3710
3690 IF Ans4$="N" THEN 4410
3700 GOTO 3670
3710 PLOTTER IS 7,5,"9872A"
3720 PEN 0
3730 PLOTTER 7,5 IS OFF
3740 PLOTTER IS 13,"GRAPHICS"
3750 IF Computer$="C" THEN GRAPHICS
3760 LIMIT 0,125,0,125            ! mm
3770 LOCATE 10,99,10,99           ! GDUs
3780 Xmin=INT(Xvarmin)
3790 Xmax=INT(Xvarmax)+1
3800 Ymin=INT(Yvarmin)
3810 Ymax=INT(Yvarmax)+1
3820 SCALE Xmin,Xmax,Ymin,Ymax
3830 PEN 1        ! white
3840 LINE TYPE 1
3850 AXES 1/10,1/10,Xmin,Ymin,10,10,1
3860 ! ------------------------------  AXIS LABELING ROUTINE
3870 ! -----  X axis title                        !
3880 DEG                                           !
3890 MOVE Xmin+.5*(Xmax-Xmin),Ymin-.067*(Ymax-Ymin)!
```

171

```
3900 CSIZE 5,.5
3910 LORG 6
3920 LDIR 0
3930 IF Xcode$="Q" THEN 3950
3940 IF Xcode$="E" THEN 3970
3950 LABEL "Qsig (pC)"
3960 GOTO 3990
3970 LABEL "Electrons (x1E+6)"
3980 GOTO 3990
3990 ! -----  X axis numbers
4000 CSIZE 3,.5
4010 LORG 6
4020 MOVE 0,Ymin-.03*(Ymax-Ymin)
4030 LABEL USING "K";"0"
4040 FOR A=.5 TO Xmax STEP .5
4050 MOVE A,Ymin-.03*(Ymax-Ymin)
4060 IF Xcode$="Q" THEN 4080
4070 IF Xcode$="E" THEN 4100
4080 LABEL USING "K";"-";A
4090 GOTO 4120
4100 LABEL USING "K";A
4110 GOTO 4120
4120 NEXT A
4130 ! -----  Y axis title
4140 MOVE Xmin-.07*(Xmax-Xmin),Ymin+.5*(Ymax-Ymin)
4150 CSIZE 4,.5
4160 LDIR 90
4170 IF Ycode$="DV" THEN 4210
4180 IF Ycode$="PD" THEN 4230
4190 IF Ycode$="PC" THEN 4250
4200 IF Ycode$="VZ" THEN 4310
4210 LABEL "-";CHR$(198);"Vfg (V)"
4220 GOTO 4320
4230 LABEL "SLOPE (x1E+11 V/C)"
4240 GOTO 4320
4250 IF Pderpercent(Nn/2)>0 THEN 4270
4260 IF Pderpercent(Nn/2)<0 THEN 4290
4270 LABEL "% CHANGE IN SLOPE"
4280 GOTO 4320
4290 LABEL "- % CHANGE IN SLOPE"
4300 GOTO 4320
4310 LABEL "Vz (V)"
4320 ! -----  Y axis numbers
4330 CSIZE 3,.5
4340 LORG 8
4350 LDIR 0
4360 FOR A=0 TO Ymax STEP 1
4370 MOVE Xmin-.003*(Xmax-Xmin),A
4380 LABEL USING "K,X";A
4390 NEXT A
4400 ! -----------------------------------------------
4410 PEN 2      ! red
4420 PLOT ABS(Xvar(1)),ABS(Yvar(1)),-2
4430 FOR J=2 TO Nn STEP 1
4440 PLOT ABS(Xvar(J)),ABS(Yvar(J)),-1
4450 NEXT J
4460 ! -----  locus label
4470 CSIZE 2.5,.5
4480 LORG 2
4490 MOVE ABS(Xvar(Nn))+.01,ABS(Yvar(Nn))
4500 ! LABEL USING "X,K";"tox=";Toxang;CHR$(208);" LF=";Lf
4510 LABEL "LF= ";Lf
4520 ! -----
4530 PRINT "LETTER command activated.  Press CONTINUE to deactivate."
4540 CSIZE 2.5,.5
4550 LETTER
```

172

```
4560 PRINT PAGE
4570 PENUP
4580 PEN 0
4590 INPUT "Do you wish to get a hardcopy printout? (Y/N)",Ans5$
4600 IF Ans5$="Y" THEN 4630
4610 IF Ans5$="N" THEN 4650
4620 GOTO 4590
4630 IF Computer$="C" THEN DUMP GRAPHICS
4640 ! DUMP GRAPHICS #7,1
4650 INPUT "Do you wish to clear the graphics display? (Y/N)",Ans6$
4660 IF Ans6$="Y" THEN 4690
4670 IF Ans6$="N" THEN 4710
4680 GOTO 4650
4690 SETGU
4700 GCLEAR
4710 INPUT "Do you wish to run the program again with a new set of parameters? (
Y/N)",Ans7$
4720 IF Ans7$="Y" THEN 410
4730 IF Ans7$="N" THEN 3750
4740 GOTO 4710
4750 PRINT PAGE
4760 PRINT "--> The program has ended.  To restart the program, press PUN.  --
-"
4770 PRINT LIN(5)
4780 STOP
4790 ! ------------------------------------------------------------------------
4800 Nn=I-1
4810 PRINT "End of Output"
4820 PRINT "A calculation has been made in the execution of the program"
4830 PRINT "resulting in a negative number.  The next statement requests the"
4840 PRINT "square root of that negative number.  This is not a valid operation"
4850 PRINT "so the computations have been stopped and the results computed"
4860 PRINT "up to this point have been printed on the screen as they normally"
4870 PRINT "would be.  The remainder of the program may be executed as if this"
4880 PRINT "had not occurred EXCEPT that now the program will only handle the"
4890 PRINT "results obtained so far and not the N points requested in the INPUT"
4900 PRINT "statement.  The request for the square root of a negative number"
4910 PRINT "indicates that the operator-entered data is incompatable with the"
4920 PRINT "physical situation represented by the program equations."
4930 GOTO 1860
4940 ! ------------------------------------------------------------------------
4950 Nn=I-1
4960 PRINT "End of Output"
4970 PRINT "A calculation has been made in the execution of the program"
4980 PRINT "resulting in a negative value of X1prime.  This is not a physically"
4990 PRINT "realizable situation so the computations have been stopped and the"
5000 PRINT "results computed up to this point have been printed on the screen"
5010 PRINT "as they normally would be.  The remainder of the program may be "
5020 PRINT "executed as if this had not occurred EXCEPT that now the program"
5030 PRINT "will only handle the results obtained so far and not the N points"
5040 PRINT "requested in the INPUT statement.  X1prime will in general be > 0."
5050 PRINT "It will be = 0 when the surface depletion region collapses at"
5060 PRINT "maximum channel charge."
5070 GOTO 1860
5080 ! ------------------------------------------------------------------------
5090 PRINT PAGE
5100 PRINT "Vz < Vqfg"
5110 PAUSE
5120 GOTO 2170
5130 ! ------------------------------------------------------------------------
5140 END
```

173

```
Program: FGBCCD
Na (Cm^-3)              4.0E+15
Nd (Cm^-3)              4.2E+16
Tox (Angstroms)         1100
Xj (Microns)            .56
Afg (Microns^2)         1093.3
Vofg (Volts)            13.0              Vfb (Volts)  -.5
Cox (F/cm^2)            3.1413E-08
LF (loading factor)     .45
CS (Picofarads)         .15440
N          101
Qsig(1) pC              0.0               Qsig(N) pC           -.5
Electrons(1)            0.000E+00         Electrons(N)         3.121E+06
Qsig(Nn)/Afg (maximum allowable signal charge per gate area) pC/um^2   -.000
Thousands of e- per um^2 corresponding to Qsig(Nn)     2.855
```
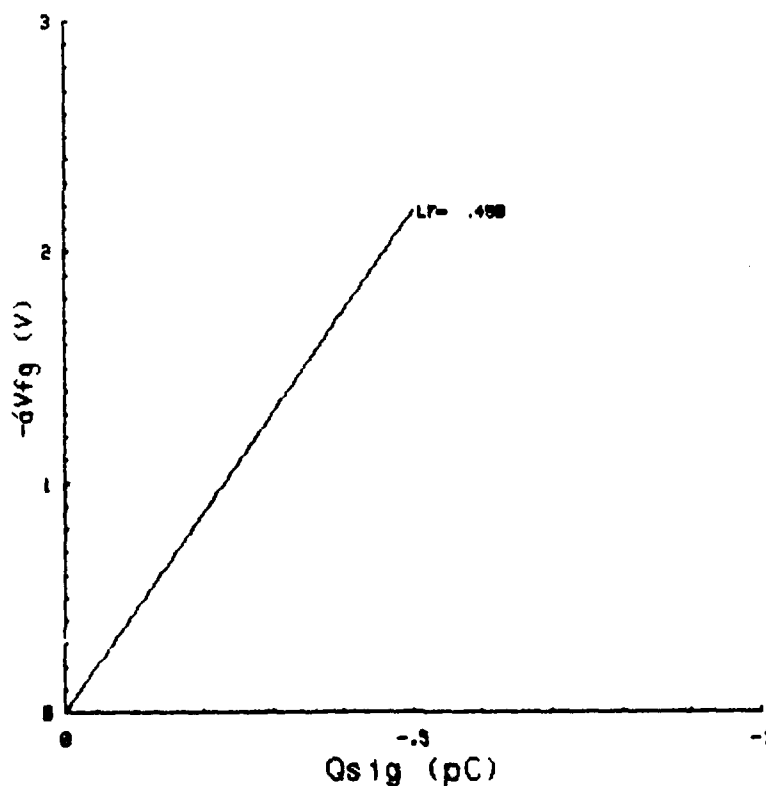

Figure B1.   HP9845C Printout of the Floating-Gate Voltage Response
             for #3022 Matrix Processor Chip (Parameters are listed
             at the top.)

174

```
Program: FG3CCD
Na (Cm^-3)              4.0E+15
Nd (Cm^-3)              4.2E+16
Tox (Angstroms)         1100
Xj (Microns)            .56
Afg (Microns^2)         1093.3
Vofg (Volts)            13.0              Vfb (Volts)  -.5
Cox (F/cm^2)            3.1413E-08
LF (loading factor)     .45
CS (Picofarads)         .15440
N              101
Qsig(1) pC             0.0               Qsig(N) pC             -.5
Electrons(1)           0.000E+00         Electrons(N)           3.121E+06
Qsig(Nn)/Afg (maximum allowable signal charge per gate area) pC/um^2   -.000
Thousands of e- per um^2 corresponding to Qsig(Nn)      2.855
```
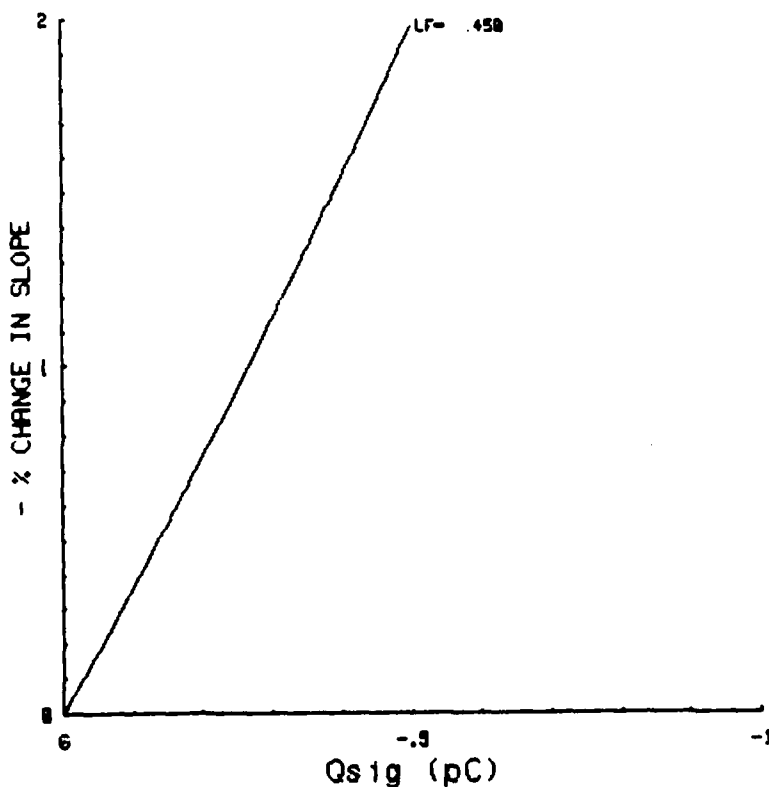

Figure B2. HP9845C Printout of the Deviation from Initial Sensitivity
for the #3022 Chip (Parameters are listed at the top.)

175

SOFTWARE USER'S GUIDE

This appendix is a detailed user's guide for the Exerciser unit delivered to RADC. The following topics are covered:

o   Procedure

o   Hierarchy

o   Memory allocation

o   Startup sequence

o   Description of functions

The RADC Exerciser is supplied with software resident in ROM and is normally to be used with Honeywell-supplied primitives. However, modifications to the set can be implemented by hookup to an MDS (8086 compatible).

A complete listing of PLM operating software can be obtained, if desired, direct from Honeywell SRC (Dr. P.C.T. Roberts, 612/378-4992).

DISCUSSION

Procedure Hierarchy

The software is composed of a collection of procedures. A procedure may have
other procedures embedded within it or it may use procedures that are externally
defined. To show the overall relationship between procedures a hierarchy is
defined. The hierarchy scheme used here is based on whether or not a procedure
is used by any other. If so, it is placed on a level below it in the hierarchy.
If it is used by two different procedures, it is defined to be a utility proce-
dure. These are referenced by their number only and are included as a separate
list below the hierarchy list. This should leave one procedure at the outer
level of the hierarchy; however another type of procedure, namely the interrupt
procedure, is also placed at the outer level. Interrupt procedures are not
referenced by any other procedure directly but rather by the hardware mechanism
that controls the interrupt; therefore, they are placed at the outer level.

With the above definitions the following hierarchy results:

```
    PIPDMO (1,2,4,7,8,9,14,20)
        COMPUTE_TG_BRANCH_ADDR
        SET_BAUD_RATE
        INIT_INTERRUPT_CONT
        SELECT_FROM_MENU
        GREY_SCALE
        TEST_PATTERN (1,2,3,6,7,8,9,12)
                    EDIT_ELEMENT
                            ELEMENT_XY
                    UPDATE_SCREEN
                    INIT_INPUT_ARRAY
        DIGITIZE_FRAME (9)
        TRACK_GATE (2,3,5,6,7,8,9)
                    MOVE_GATE
                            RESTORE_VIDEO
                            SAVE_VIDEO_AND_DRAW_EDGE
                    HOME_GATE
                    VIDEO_TO_PIP
                            XFER_VIDEO_TO_PIP_ARRAY
                            XFER_PIP_ARRAY_TO_VIDEO
                    THRESHOLD
                            DARK_THRESHOLD
                            LIGHT_THRESHOLD
                            BINARY_THRESHOLD
        QUICK_TG_MEMORY_TEST (8,10,11)
                    TG_ADDRESS_FORMATTING
                            PRINT_ERROR_ADDR
        QUICK_IO_MEMORY_TEST (8,10,11)
                    IO_ADDRESS_FORMATTING
                            PRINT_ERROR_ADDR
        TG_UTILITIES (16,17,18,19,20)
                    GET_ADDR
                    IO_BUFFER_BYTE
                    TG_SINGLE_STEP
```

```
                    TG_HALT
                    TG_SEG_ADDR
                    TG_JAM_ADDR
                    TG_VERIFY
                    TG_SAVE
         HALT_TG_INTERRUPT
         DIGITIZE_AT_INTERRUPT
```

Utility Procedures

```
 1)  RGB256_INIT
 2)  INTERPRET_PRIMITIVES
 3)  IOB_MEMIO
 4)  BRANCH_ADDR
 5)  EXECUTE_PRIMITIVES
 6)  PROMPT_AND_GET_CHAR
 7)  SIO_OUT_CHAR
 8)  SIO_OUT_STRING
 9)  SIO_GET_CHAR
10)  SIO_OUT_BYTE
11)  SIO_OUT_WORD
12)  SIO_ASCII_TO_HEX
13)  SIO_CRLF
14)  SIO_MS_DELAY
15)  SIO_GET_STRING
16)  TG_RESET
17)  TG_RUN
18)  TG_MACRO_SELECT
19)  TG_MEMIO
20)  TG_LOAD
```

Memory Allocation

The 8086 CPU address is 20 bits long giving a range of 0 to FFFFF hexidecimal
(0 to 1,048,575 decimal). The single board computer (SBC) has been configured
such that RAM is located at addresses 0 to 7FFF (0 to 32,767 decimal) and EPROM
is located at addresses F8000 to FFFFF (1,015,808 to 1,048,575 decimal). The
following shows the starting location for the Intel supplied monitor program
and scratchpad, the PIP waveform data, the program data, and the program code
and constants.

FIRST ADDRESS HEX (DECIMAL)		DESCRIPTION
0	(0)	Intel 957A Monitor RAM
700	(1,792)	Program Data
2,000	(8,192)*	Program Code
6,000	(24,576)*	Constants and PIP Waveforms
FE000	(1,040,384)	Intel 957A Monitor Code and Constraints

*The Program Code and Constants when in EPROM's are located starting
 at address F8000 (1,015,808). Constants and PIP Waveforms when in
 EPROM's are located starting at address FC000 (1,032,192).

Startup Procedure

From RAM:

When the program is to be executed from RAM the following steps are performed:

The following should be turned on: MDS computer, SBC, camera, monitor and the TI terminal (if it is to be used). The following steps should be performed in the order shown:

1) Put disk "PIP MASTER" in drive 1 and disk "SYSTEM.DBY" in drive 0.
2) Press RESET on MDS.
3) Press RESET on the SBC.
4) Type SUBMIT :F1:PIP
5) When the MDS displays :F1:TALK, press RESET on the SBC.

The program has now been down loaded to the SBC RAM. If the MDS terminal is to be used two control U's must be typed on the MDS keyboard. Now the command G200:2 can be typed to begin program execution.

If the TI terminal is to be the I/O device, its cable must be put on the J2 connector in place of the download cable. The SBC must be reset and two control U's typed on the TI keyboard. Now the command G200:2 can be typed to begin program execution.

From EPROM:

When the program is to be executed from EPROM the following steps are performed:

Turn on the SBC chassis, camera, monitor and the TI terminal or the MDS computer. The cable from the TI terminal or the MDS computer (whichever device is to be used for I/O) is to be connected to the J2 connector on the SBC card.

If the MDS computer is to be used the disk "PIP MASTER" must be placed in drive 1 and disk SYSTEM.DBY in drive 0. The following must be typed:

> :F1:BAUD 112
> :F1:TALK

Now with either terminal the monitor is invoked by pressing two control U's. The interrupt vector pointer locations must now be loaded as follows:

type:	SW A0, <cr>
	21B4,
	F800,
	21D8,
	F800,
	087E,
	F800,
	08DA,
	F800 <cr>

The program is then started by typing GF800:2.

Description of Functions

When the program is started, the following is printed:

        LIST OF AVAILABLE FUNCTIONS

        G = DISPLAY GRAY SCALE
        P = DISPLAY TEST PATTERN
        D = DIGITIZE ONE FRAME
        T = TRACK GATE
        M = TG. MEM. TEST
        O = IO. MEM. TEST
        U = TG UTILITIES
        I = INTERPRET AND EXE
        L = LIST FRAME MEMORY

        SELECT FROM LIST

Each of these functions will be described in the following paragraphs. The
following conventions are used in command syntax descriptions (note that "A"
and "B" represent arbitrary ASCII character strings):

[A]	indicates that "A" is optional
{A B}	indicates that either "A" or "B" must be selected
<A>	indicates that "A" is an ASCII hexadecimal constant
<cr>	indicates that a carriage return is entered
b	indicates that a blank or space is entered

DISPLAY GRAY SCALE:

This function displays on the last 16 lines of the monitor a scale of 16 gray
shades ranging from black on the left to white on the right. It is invoked
by pressing the "G" key on the keyboard.

DISPLAY TEST PATTERN:

This function, when invoked by pressing the "P" key, displays the test pattern
in the upper left quadrant of the monitor. The terminal then responds with
an asterisk prompt. This prompt means that the program is still in the "test
pattern" function and there are remaining options available. The response to
the prompt can be "I", "E", "P", or "carriage return", where:

I	indicates initialize test pattern
E	indicates edit test pattern
P	indicates enter primitives
<cr>	indicates return to SELECT FROM LIST level

The "I" and <cr> need no additional input. The "E" option prints the terminal response "ELEMENT/SHADE?". The required input is 3 hexidecimal characters. The first represents the column of the element that is to be edited; the second character represents the row. The third character is the shade of gray which the element will be assigned, black being a value of "0" and white a value of "F". After each entry, the response is repeated and another element can be changed; if no more changes are necessary, a carriage return is entered.

The "P" option responds with "ENTER PRIMITIVES". The required response is one of the two character mnemonics corresponding to the PIP primitives. These are listed in the description of the INTERPRET AND EXE function. When the list of primitives have been entered, a carriage return executes the primitives and displays the first 256 elements of PIP's write portion of the I/O buffer in the test pattern format. The first time this option is selected, the result is placed in the upper right quadrant of the monitor, the next time in the lower left, and then the lower right. The fourth time, the result will overwrite the upper right quadrant and the cycle repeats. Each time the program comes back to the prompt level and another "P" must be entered.

DIGITIZE ONE FRAME:

This function is invoked by pressing the "D" key.

The program responds with "SELECT SOURCE(S) (0-3)". The correct response is any combination of the numbers 0, 1, 2 or 3. The numbers selected correspond to the four video sources which are connected to the video memory. When a source is selected it will be digitized once every 50 milliseconds for two seconds. Then the next video source selected will be digitized in a similar manner, and so on. To freeze the frame, any key is pressed. The program then returns to the SELECT FROM LIST level.

TRACK GATE:

This function is invoked by pressing the "T" key. The program responds with "GATE SIZE? (1-8)". The single number 1 through 8 is then entered. The number entered determines the size of a side of the gate which is drawn (1 representing 16 pixels and 8 representing 8*16 or 128 pixels). The gate can now be moved up, down, left or right by pressing the "U", "D", "L", or "R" keys. When the gate is positioned in the desired position, the "H" key is pressed to place the gate and its current contents back to the upper left of the monitor. The rest of the monitor is set to a black level.

The asterisk prompt is now printed indicating the program is still in the "track gate" function and there are remaining options available. The response to the prompt can be "P", "S", "A", "E", "T", "B", or "carriage return", where:

P	indicates enter primitives
S	indicates subtract gates
A	indicates average gates
E	indicates edge detect
T	indicates threshold
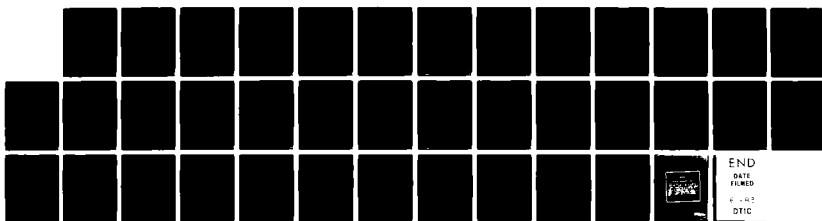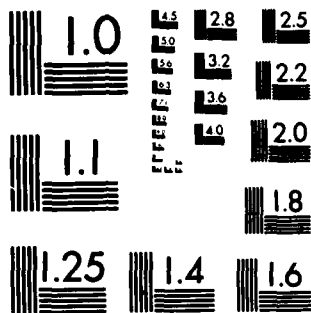
MICROCOPY RESOLUTION TEST CHART

NATIONAL BUREAU OF STANDARDS-1963-A

B   indicates background average

<cr>   indicates return to SELECT FROM LIST level

The program prints "INPUT GATE(S), OUTPUT GATE?" The valid response is two
or three numbers each between 0 and 3 which represent quadrants of the monitor
where the input is to be taken and the output is to appear.

The "P" option responds with "ENTER PRIMITIVES". The required response is one
of the two character mnemonics corresponding to the PIP primitives. These are
listed in the description of the INTERPRET AND EXE function. When the list
of primitives have been entered, a carriage return executes the primitives and
displays the first 256 elements of PIP's write portion of the I/O buffer in
the test pattern format.

The "S" option subtracts the pixel values of the first input gate from the pixel
values of the second input gate and places the results in the output gate.

The "A" option averages the pixel values of the two input gates and places the
results in the output gate.

The "E" option responds with "THRESHOLD?". The valid response is a single hexa-
decimal number 0 (zero) thru F (0 = dark, F = light). The program then uses
the Robert's cross algorithm to perform an edge detect function on the input
gate and places the results in the output gate.

The "T" option requests a threshold type and a threshold value. This option
first responds with "THRESHOLD TYPE (D,L,B)?". The valid responses are "D",
"L", "B" or carriage return where:

    D - (Dark Threshold) will change all pixels with a value greater than
        the threshold value to white

    L - (Light Threshold) will change all pixels with a value less than the
        threshold value to black

    B - (Binary Threshold) will change all pixels with a value less than or
        equal to the threshold value to black and those with a value greater
        than the threshold value to white

  <cr> - (carriage return) will perform no action and allow the user to select
        another TRACK GATE option.

The option then prints "THRESHOLD VALUE (0-F)?". The valid response is a single
hexadecimal digit 0 (zero) thru F.

The "B" option requests a matrix size. The valid response is the odd number
3, 5, or 7.

TG MEM. TEST:

This function is invoked by pressing the "M" key. The timing generator memory
test is executed. If no errors are detected, no response is given and the pro-
gram returns to the "SELECT FROM LIST" level. If an error is found, the bad
memory address and its contents are printed. Successful completion of the
test requires about 7 seconds execution time.

IO MEM. TEST:

This function is invoked by pressing the "O" key. The IO buffer memory test
is executed. The results are similar to the TG MEM. TEST described above.

TG UTILITIES:

This function is invoked by pressing the "U" key. The program responds with
an asterisk prompt.

This utility package allows the user to examine and/or modify the contents of
the timing generator and I/O buffer memories, as well as debug timing generator
microcode using the utility's single step capability.

Load Timing Generator Memory -- Syntax:

L ∅ <start addr>[, <end addr>] <cr>

This command loads timing generator memory with microinstructions from
SBC-86/12. <start addr> and <end addr> specify the range of addresses
of locations in timing generator memory whose contents are to be loaded
(if <end addr> is omitted, then <end addr> = <start addr>). Both <start
addr> and <end addr> must be in the range 0 to 7FF. Data for microword
N are taken from bytes at addresses K+6*N through K+6*N+5, where K is a
constant whose value is determined by the location of the label
"START_OF_TG_BUFFER".

Examples:

L 0,1FF   loads and verifies timing generator address 0 through 1FF
           from SBC

L 200   loads address 200 from SBC

Verify Timing Generator Memory -- Syntax

V ∅ start addr , end addr   cr

This command verifies the contents of timing generator memory against SBC-
86/12. The parameters "<start addr>" and "<end addr>" have the same in-
terpretations as in the load command. If a discrepancy is found during
verification, a string of 6 hexidecimal bytes is printed out for the ad-
dress found to be in error.

184

Example:

    V 5,F   verifies timing generator address 5 through F against SBC

Save Timing Generator Memory -- Syntax:

        SØ <start addr> [, <end addr>] <cr>

This command stores the contents of timing generator memory in SBC RAM.
Data are taken from microwords at locations <start addr> through <end addr>
(if <end addr> is omitted, then <end addr> = <start addr>). The 6 bytes
from microword N are written to the bytes at RAM locations 6*N through
6*N+5. Typically, timing generator microinstructions are written to disk-
ette using the ICE-86 emulator or the SBC monitor upload feature once the
microinstructions have been transferred to SBC RAM using this command.

Example:

    S 0,7FF   saves entire contents of timing generator memory

Display Microinstructions -- Syntax:

        DMØ <start addr> [, <end addr>] <cr>

This command displays timing generator microinstructions as strings of
ASCII hexadecimal bytes on the serial terminal. The address and 6 data
bytes corresponding to each microinstruction between locations <start addr>
and <end addr> are printed out as ASCII hexadecimal constants (if <end
addr> is omitted, then <end addr> = <start addr>).

Example:

    DM 100   displays microinstruction at location 100

Change Microinstruction Byte -- Syntax:

        CMØ <micro addr>,<byte addr>,<data byte><cr>

This command changes a single byte in a timing generator microinstruction
and displays the microinstruction once the change has been made. Byte
<byte addr> of the microinstruction at location <micro addr> is set to
<data byte>. Note that microinstruction bytes are numbered 0 through 5,
with 0 corresponding to the most significant byte.

Example:

    CM 2F,0,AA   sets byte 0 of microinstruction 2F to AA

Display I/O Buffer -- Syntax:

        OBØ <start addr> [, <end addr>] <cr>

This command displays the contents of the PIP I/O buffer between addresses <start addr> and <end addr> (if <end addr> is omitted, then <end addr> = <start addr>). <start addr> and <end addr> must be legal I/O buffer addresses (i.e., between 0 and 1FFF).

Example:

    DB 0,10   displays I/O buffer locations 0 through 10

Change I/O Buffer -- Syntax:

        CBᴘ <start addr> , <nbytes> , <data byte1> ,.... <count> *
        <data byteJ> ,...., <data byte K> <cr>

This command sets one or more bytes of the I/O buffer to specified values; <nbytes> bytes starting with location <start addr> are set to the ASCII hex values specified in the data list following the <nbytes> parameter. Each entry in this data list consists of either a single byte or a replicated byte of the form <count> * <data byte>, the latter being equivalent to <data byte> appearing as a single entry <count> times. The total number of bytes specified in the data list must be <nbytes>.

Examples:

    CB 0,10,0,B,E*FF   sets byte 0 to 0, byte 1 to B, and bytes 2 through
                         F to FF

    DB 0,100,100*0   clears bytes 0 through FF

Change Macro Address Register -- Syntax:

        CAᴘ<macro addr><cr>

This command sets the macro address register on the timing generator to a specified address (the address must be a legal timing generator memory address).

Example:

    CA 300   sets macro address register to 300

Run Timing Generator -- Syntax:

        Rᴘ <clock rate><cr>

This command enables the timing generator sequencer to run continuously at a specified clock rate. Clock rate must be in the range 0 through 7 and selects the clock rate as follows:

        0 = 8.33 MHz
        1 = 4.16 MHz
        2 = 2.08 MHz
        3 = 1.04 MHz

```
                        4 = 500 KHz
                        5 = 250 KHz
                        6 = 125 KHz
                        7 = 63 KHz
```

Example:

    R 0   runs timing generator at 8.33 MHz

Halt Timing Generator -- Syntax:

    H<cr>

This command disables the timing generator sequencer from running continuously.

Single Step Timing Generator -- Syntax:

    G[[T]ɓ[<start addr>,]<end addr>]<cr>

This command single steps the timing generator sequencer either unconditionally or until a specified microinstruction address is encountered. In its simplest form, the command steps the sequencer one instruction; this is the form "G<cr>." The form "G[T]ɓ[<start addr>,]<end addr><cr>" specifies that the sequencer is to be stepped until the microinstruction at location <end addr> has executed. If <start addr> is present, then execution is forced to start at address <start addr>; otherwise, execution starts at whatever address the sequencer is currently generating. The "T" parameter, when present, specifies that a trace be built of those addresses generated by the sequencer until <end addr> is encountered. Each address generated by the sequencer after each single step is written to the trace buffer starting with buffer location 0. The buffer holds up to 256 addresses; if the buffer overflows, single step execution is halted and a message to that effect is printed on the terminal. Once single stepping has halted, either normally or abnormally, a word called the trace pointer gives the index of the last location written in the trace buffer. The contents of both the trace pointer and the trace buffer can be examined using the commands in the Display Trace Pointer/Trace Buffer command.

Examples:

    G   steps once

    G 200   steps until address 200 is executed

    G 300,200   starts execution at address 300 and steps until address 200 is executed

    GT 2FF   enables trace and steps until address 2FF is executed

<u>Display Trace Pointer/Trace Buffer</u> -- Syntax:

   DTP<cr>   display trace pointer

   DT <start addr>[,<end addr>] <cr>   display trace buffer

These commands allow examination of the trace buffer as described in the
Single Step Timing Generator command.

The first command listed above displays the contents of the trace pointer
so that the user can determine how much of the trace buffer contains useful
information.

The second command displays the contents of the trace buffer. In its sim-
plest form, the latter command is "DT<cr>," which displays the entire trace
buffer. If <start addr> and <end addr> are present, the contents of the
trace buffer between locations <start addr> and <end addr> are displayed
(if <end addr> is omitted, then <end addr> = <start addr>).

Examples:

   DT 0,1FF   displays locations 0 through 1FF of trace buffer

   DT 50   displays location 50 of trace buffer

<u>Initialize Timing Generator</u> -- Syntax:

       I<cr>

This command issues a reset pulse to the timing generator and forces the
sequencer to execute the microinstruction at address 0.

<u>Quit</u> -- Syntax:

       <cr>

A carriage return on a line by itself returns the program to the SELECT
FROM LIST level.

Certain control characters have special functions when entered by the user from
the serial terminal. Control S, when entered during any data printout on the
terminal, causes the printout to temporarily halt; entering control Q causes
the printout to resume. Control X, when entered as part of a command line being
typed in by the user, causes that line to be deleted. Control C has the effect
of returning control to the SBC monitor program.

INTERPRET AND EXE:

This function is invoked by pressing the "I" key. The program responds with
"ENTER PRIMITIVES". The correct response is one of following 2 character mne-
monics:

SR	Shift Array Right
SL	Shift Array Left
SU	Shift Array Up
SD	Shift Array Down
L1	Load Store #1 from X
L2	Load Store #2 from X
L3	Load Store #3 from X
U1	Unload Store #1 to X
U2	Unload Store #2 to X
U3	Unload Store #3 to X
E1	Erase Store #1
E2	Erase Store #2
E3	Erase Store #3
M2	Store #1 minus Store #2
R1	Regenerate #1
R2	Regenerate #2
OS	Execute once
CO	Execute continuously
IA	Input entire array
OA	Output entire array
D3	Store #1 minus Store #3
D4	Store #1 minus X
RF	Rotate X, Store #2, Store #3
SW	Swap Store #2, #3
RB	Rotate X, Store #3, Store #2
MV	Move Store #1 to Store #2
MX	Move Y to X

After each mnemonic is entered, the code is interpreted and a *"JUMP TO SUBROUTINE"* 2910 microinstruction is placed in the next available TG memory location. When all primitives have been entered, a carriage return is typed as the first character of a line. This causes the TG sequencer to execute the primitives and returns the program to the "SELECT FROM LIST" level.

LIST FRAME MEMORY:

This option is not complete. Any response from the user will cause a return to the SELECT FROM LIST level. This option is intended to allow the user to list the contents of frame memory to a printer.

PIP PRIMITIVE MNEMONIC: SR
PRIMITIVE DESCRIPTION: SHIFT ARRAY RIGHT
ACTIVE CLOCKS: PHI   T
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: JMD
UPDATED ON: 01/27/82

PHI   PHA   PHB   PHC   PHX   I   PHI

PMI   I   PHX   PHC   PHB   PHA   PMI

08
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21

PIP PRIMITIVE MNEMONIC: SL
PRIMITIVE DESCRIPTION: SHIFT ARRAY LEFT
ACTIVE CLOCKS: PHI   PHX   PHC   PHB   PHA   PHI
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: DBY
UPDATED ON: 8/4/81

PHI	PHX	PHC	PHB	PHA	PHI

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21

PHI   I   PHX   PHD   PHC   PHB   PHI

PHI   I   PHX   PHD   PHC   PHB   PHI

08
03
04
06
07
08
09
10
11
12
13
14
15
16
17
18
20

PIP PRIMITIVE MNEMONIC: L1
PRIMITIVE DESCRIPTION: LOAD STORE #1 FROM X
ACTIVE CLOCKS: RSN1   PHX   PHD   PHC   PHB
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PK/JH
UPDATED ON: 9/08/82

RSN1   PHX   PHD   PHC   PHB

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39

194

PIP PRIMITIVE MNEMONIC: L2
PRIMITIVE DESCRIPTION: LOAD STORE #2 FROM X
ACTIVE CLOCKS: PHX   PHXS   PHC   PHD
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PR/JP
UPDATED ON: 9/09/82

	PHX	PHXS	PHC	PHD
01				
02				
03				
04				
06				
07				
08				
09				
10				
11				
12				
13				
14				

PIP PRIMITIVE MNEMONIC: L3
PRIMITIVE DESCRIPTION: LOAD STORE #3 FROM X
ACTIVE CLOCKS: PHX   PHXS   PHB   PHC
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PR/JP
UPDATED ON: 9/09/82

PHX   PHXS   PHB   PHC

01
02
03
04
05
06
07
08
09
10
11
12
13
14

196

PIP PRIMITIVE MNEMONIC: U1
PRIMITIVE DESCRIPTION: UNLOAD STORE #1 TO X
ACTIVE CLOCKS: ST1   PHX   PHU   PHC   PHB
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PH/JH
UPDATED ON: 9/08/82

ST1	PHX	PHU	PHC	PHB

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

PIP PRIMITIVE MNEMONIC: U2
PRIMITIVE DESCRIPTION: UNLOAD STORE #2 TO X
ACTIVE CLOCKS: ST2   PHU   PHC   PHXS
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PH/JP
UPDATED ON: 9/29/82

ST2   PHU   PHC   PHXS

PIP PRIMITIVE MNEMONIC: U3
PRIMITIVE DESCRIPTION: UNLOAD STORE #3 TO X
ACTIVE CLOCKS: PHB PHC PHX PHXS ST3
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PR/JP
UPDATED ON: 9/09/82

PHB PHC PHX PHXS ST3

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22

199

PIP PRIMITIVE MNEMONIC: E1
PRIMITIVE DESCRIPTION: ERASE STORE #1
ACTIVE CLOCKS: ST1 HSN1 PHX
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PR/JII
UPDATED ON: 9/08/82

PIP PRIMITIVE MNEMONIC: E2
PRIMITIVE DESCRIPTION: ERASE STORE #2
ACTIVE CLOCKS: PHX5   ST2   RSN1   PHX   PHD   PHC   PHB   PHA
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PK/JH   CHANGES: ELIMINATE REG PCHANGE & SINK/SNEAK D
UPDATED ON: 10/6/82

201

PIP PRIMITIVE MNEMONIC: E3
PRIMITIVE DESCRIPTION: ERASE STORE #3
ACTIVE CLOCKS: RSM1   HSI
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: DBY
UPDATED ON: 01/29/82

203

PIP PRIMITIVE MNEMONIC: M2
PRIMITIVE DESCRIPTION: STORE #1 MINUS STORE #2
ACTIVE CLOCKS: ST3   HSNI   HSI   MD   FGR   ST1   ST2   PHXS   PHX   PHD   PHC   PHB   PHA

INITIAL CLOCK STATES: DEFAULT   CHANGES: ELIM PCHARGE
UPDATE BY: PR/JH
UPDATED ON: 10/06/82

	ST3	HSNI	HSI	MD	FGR	ST1	ST2	PHXS	PHX	PHD	PHC	PHB	PHA

001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
039
040
041
042
042
042
043
044
045
046

PIP PRIMITIVE MNEMONIC: R1
PRIMITIVE DESCRIPTION: REGENERATE 1
ACTIVE CLOCKS: ST1   FGR   RSN1   RS1   PHX   PHC   PHB   PHA   MD
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: PR/JH
UPDATED ON: 10/9/82

208

209

PRIMITIVE DESCRIPTION: REGENERATE 2

ACTIVE CLOCKS: PHA
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: LHL
UPDATED ON: 03/15/82

PHA MU FGR RSNI RSI PHX PHD PHB SI2 PHC PHXS

01
02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
DONE

PIP PRIMITIVE MNEMONIC: D3
PRIMITIVE DESCRIPTION: STORE #1 MINUS STORE #3
ACTIVE CLOCKS: ST3   ST1   MD   FGR
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: LRL
UPDATED ON: 03/15/82

PIP PRIMITIVE MNEMONIC: D4
PRIMITIVE DESCRIPTION: STORE #1 MINUS X
ACTIVE CLOCKS: SI3   SII   MD

INITIAL CLOCK STATES: DEFAULT
UPDATE BY: LHL
UPDATED ON: 03/15/82

212

PIP PRIMITIVE MNEMONIC: RF
PRIMITIVE DESCRIPTION: ROTATE X, STORE #2, STORE #3
ACTIVE CLOCKS: PHXM   PHXS   PHX   PHD   PHC   PHB   ST2   ST3
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: JMD
UPDATED ON: 01/27/82

PHXM	PHXS	PHX	PHD	PHC	PHB	ST2	ST3
01							
02							
03							
04							
05							
06							
07							
08							
09							
10							
11							
12							
13							
14							

PIP PRIMITIVE MNEMONIC: SW
PRIMITIVE DESCRIPTION: SWAP,STORE #2 AND #3
ACTIVE CLOCKS: RS1   ST3   ST2
INITIAL CLOCK STATES: DEFAULT
(UPDATE BY: PR/JP
UPDATED ON: 09/09/82

214

PIP PRIMITIVE MNEMONIC: RU
PRIMITIVE DESCRIPTION: ROTATE X, STORE #3, STORE #2
ACTIVE CLOCKS: ST2   PHXM   PHXS   PHX   PHD   PHC   PHB   ST3
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: JMD
UPDATED ON: 01/27/82

PIP PRIMITIVE MNEMONIC: MV
PRIMITIVE DESCRIPTION: MOVE STORE #1 TO STORE #2
ACTIVE CLOCKS: ST2   ST1   PHXS   PHX
INITIAL CLOCK STATES: DEFAULT
UPDATE BY: JMD
UPDATED ON: 01/27/82

ST2   ST1   PHXS   PHX   PHD   PHC   PHB   PHA

01
02
03
04
05
06
07
08
09
10
11
12
13
14

PIP PRIMITIVE MNEMONIC: MX
PRIMITIVE DESCRIPTION: MOVE Y TO X
ACTIVE CLOCKS: RSN1   RS1   PHX   PHD   PHC   PHB   PHA
INITIAL CLOCK STATES: PRESET
UPDATE BY: JMD
UPDATED ON: 01/27/82

RSN1   RS1   PHX   PHD   PHC   PHB   PHA

01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18

218

REFERENCES

1. Narendra, P.M., "Reference-Free Nonuniformity Compensator for IR Imaging Arrays," _SPIE Proceedings: Smart Sensors II_, Vol. 252, August 1980.

2. McCaughan, D.V. and Harp, J.G., "Phase-Referred Input: A Simple New Linear CCD Input Method," _Electronics Letters_, Vol. 12, No. 25, December 9, 1976, pp. 682-83.

3. Lamb, D.R., "CCD Applications Study," Interim Report on RADC Contract No. F19628-79-C-0176, Honeywell Systems and Research Center, Minneapolis, Minnesota, May 5, 1980, pp. 30-36.

4. Haken, R.A., "A Comparison of Static Bulk-Channel Charge-Coupled Device Characteristics Using Uniform, Gaussian, and Measured Impurity Distributions," _Solid-State Electronics_, Vol. 20, 1977, pp. 789-97.