sun microsystems

Hardware Engineering Manual
for the
2060 CPU Board

Confidential and Proprietary Material

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1386-13
{Rev 1 of 10 May 1987} **CONFIDENTIAL!**

Acknowledgements

Thanks to **Karl Bizjak, Joe Murphy, Jim Ludemann, Bruce Smith, Kevin Mobley, and Niel Hanes**, without whose explanation and assistance this manual would never have been written.

Contents

Preface xxiii

Chapter 1 An Overview of the Sun-3 Architecture 3

1.1. CPU 3
1.2. System DVMA 3
1.3. VME Slave User DVMA 3
1.4. Refresh 4
1.5. DVMA Controller 4
1.6. Control Space Devices — FC3 4
1.7. Memory Management Unit 4
1.8. Device space 4
1.9. Memory Space (TYPE0 space) 5
Parity Main Memory 5
Video Memory 5
1.10. I/O Devices (TYPE1 space) 5
1.11. VME Master (TYPE2 and TYPE3 Space) 6
1.12. Interrupts 6
On-Board Interrupts 6
VME Vectored Interrupts 7
1.13. CPU Resets and Timeout 7

Chapter 2 VME Compliance 11

2.1. Options 11
2.2. Performance Parameters 11

Chapter 3 Block Diagram **15**
3.1. Data Paths 16

Chapter 4 Mechanical Specifications **19**
4.1. Board Form Factor 19
4.2. Connectors 19
4.3. Switches 19

Chapter 5 68020, 68881 Floating Point Coprocessor, and Associated Circuitry **23**
5.1. Processor Data Buffers — U105:2 23
U105:2 Data Flow 23
5.2. Cache Disable — J100 23
5.3. CPU Space PALs — U106 and U107 24
5.4. U106 PAL 26
U106 Input Signals 26
U106 Output Signals 27
5.5. U107 PAL 30
U107 Input Signals 31
U107 Output Signals 31

Chapter 6 Power-on Circuitry **37**

Chapter 7 Response Synchronizer — U206 ..... **41**

Chapter 8 Reset Pal U201 and User Reset Switch U205 ..... **45**
8.1. U205 User Reset Switch 45
8.2. U201 Reset PAL 46
U201 Input Signals 46
U201 Output Signals 47

Chapter 9 U202 and U203 Bus Error PAL and Register **51**
9.1. U202 Inputs 51
9.2. Pinout of U202 PAL 52

9.3. U202 Output Signals 53
9.4. U203 Bus Error Register 56

Chapter 10 U204 DSACK PAL **59**
10.1. Pinout of U204 PAL 59
10.2. U204 Input Signals 60
Bus Transfer Size 60
Offset Bits 60
10.3. U204 Outputs 62

Chapter 11 Interrupt Circuitry — U301:U300, J300 ..... **67**
11.1. Interrupt Priority 67
11.2. U301:0 Interrupt Enable Registers 68
11.3. J300 69

Chapter 12 Interrupt Circuitry — U302–U304 PALs, U305 Register **73**
12.1. Interrupt Request Cycle 73
12.2. Interrupt Acknowledge Cycle 74
12.3. Priority 77
12.4. Two-Level Priority Encoding 77
12.5. U302 Lower-Priority Encoder 78
U302 Pinout 79
U302 Input Signals 79
U302 Output Signals 80
12.6. U303 Higher-Priority Encoder 84
U303 Pinout 84
U303 Input Signals 84
U303 Output Signals 85
12.7. Second-Level Interrupt Priority Encoder — U304 89
Pinout of U304 PAL 89
U304 Input Signals 90
U304 Output Signals 90
Sample Interrupt Cycle 92

Spurious Interrupt 92
Ethernet Controller and Spurious Interrupts 93
12.8. U305 Latch 94

Chapter 13 ATE Pulldowns — U407 ..... **97**

Chapter 14 Clock Generation — U400–U406 ..... **101**
14.1. Pinout of U400 Clock PAL 101
14.2. U400 Input Signals 102
14.3. U400 Output Signals 103
14.4. U401 Flip-Flops 104
14.5. U402 Flip-Flops 104

Chapter 15 Pal U408 ..... **107**
15.1. Pinout of U408 PAL 107

Chapter 16 Sun-3 Memory Management Unit (MMU) **111**

Chapter 17 Context Register — U509 ..... **115**
17.1. U509 Pinout 116
17.2. U509 Input Signals 116
17.3. U509 Output Signals 117

Chapter 18 Segment Map — U500:08 ..... **121**
18.1. Segment Map Read and Write Cycles 122
Segment Map RAM Read Cycle 122
Segment Map RAM Write Cycle 123
Truth Table for the U508 Buffer 123
18.2. Segment Map RAM Control Signals 123

Chapter 19 Page Map RAM **127**

Chapter 20 Statistics Control PAL — U611 ..... **133**
20.1. U611 Input Signals 133

20.2. U611 Output Signals 134

Chapter 21 MMU Validation and Decode PAL — U612 ..... **139**
21.1. U612 Input Signals 139
21.2. U612 Output Signals 140

Chapter 22 P2 Bus Control and Address Buffers **147**
22.1. U700 Comparator 147
22.2. U703:01 P2 Address Buffers 147
22.3. U704 Control Signal Buffer 148
22.4. Aliases 148

Chapter 23 Parity Circuitry **151**
23.1. Parity Address Latch — U811:08 151
23.2. Parity Generator/Checkers — U807:04 151
23.3. U803 Multiplexer 152
23.4. Parity Control and Parity Check PALs — U802 and U812 152
23.5. U812 Parity Check PAL 153
U812 Input Signals 153
U812 Output Signals 153
23.6. U802 Parity Control PAL 156
U802 Input Signals 156
U802 Output Signals 157
Pinout of U802 PAL 159
23.7. Memory Error Register — U801 161
23.8. Byte Select Buffer (and Address Bit Driver) — U813 162
23.9. Parity Data Buffer — U3112 162

Chapter 24 MOS Bus Devices **165**
24.1. U900 MOS Enables PAL 166
U900 Pinout 167
U900 MOS Write Enable — moswren 167
U900 MOS Read Enable — mosrden 168
Diagnostic Cycle — diagcy 169

24.2. U901 MOS SACK State Machine 169
Pinout of U901 PAL 170
U901 MOS Read/Write Control 172
U901 SCC Interrupt — SYNCWAIT State 176
24.3. U904 MOS Read/Write Strobe Decoder 179
U904 Pinout 179
U904 Input Signals 180
U904 Output Signals 180
24.4. U902 MOS Write and U903 MOS Read Buffers 184
U902 MOS Write Buffer 184
U903 MOS Read Buffer 184
24.5. MOS Read and Write Cycles 185
MOS Read Cycle 186
MOS Write Cycle 186
24.6. Mouse and Keyboard SCC 186
U405 and U2207 Baud Rate Clock 187
Transmit Data Path 187
Receive Data Path 187
24.7. Serial Ports A and B — ttya and ttyb 187
Transmit Data Path 188
Receive Data Path 188
24.8. EEPROM and EPROM 188
EEPROM 190
EPROM 190
24.9. Time of Day (TOD) Clock 190
TOD Oscillator Circuit 190

Chapter 25 TTL Bus Accesses **195**
25.1. TTL Bus Read/Write Cycle 195
25.2. U1400 TTL Bus Sack State Machine 195
U1400 Pinout 196
U1400 State Machine Outputs 196
TTL Bus DTACK State Machine Diagram 198

TTL Bus Cycle Timing 203
25.3. U1401 TTL Bus Device Decoder 204
U1401 Pinout 205
U1401 Input Signals 205
Output Signals of the U1401 PAL 206
25.4. U1402 MMU Decoder 210
U1402 Pinout 212
U1402 Input Signals 212
U1402 Output Signals 213
25.5. U1403 (Miscellaneous) CPU Signal TTL Bus Decoder 218
Pinout of U1403 PAL 219
U1403 Input Signals 219
U1403 Output Signals 220
25.6. Ethernet Control Register 223
U1405 Ethernet Control Write Buffer 223
U1407 Ethernet Control Read Buffer 224
25.7. System Enable Register 224
U1406 System Enable Write Register 224
U1408 System Enable Read Register 225
25.8. U1410 Diagnostics Register 225
25.9. U1409 ID PROM 225
25.10. U1404 P2-to-TTL Data Buffer 225
25.11. U2905 and U2906 User DVMA Enable Register 226
25.12. U203 Bus Error Register 226
25.13. U509 Context Register 226
U509 Pinout 227
Inputs and Outputs of U509 Context Register 227

Chapter 26 Video Circuitry **233**
26.1. Video Cycle Timing 233
26.2. U1504 Video Select Decoder 234
U1504 Pinout 235
U1504 Input and Output signals 235

26.3. U1502 Video Control Decoder 236
U1502 Pinout 237
U1502 Input Signals 237
U1502 Output Signals 237
26.4. P2 Interface State Machine — U1503, U1605/07 241
U1503 Pinout 241
U1503 Inputs 241
U1503 Outputs 242
26.5. VARB and Video Side State Machines 242
Video Read 243
Video Write 247
Video Write Timing Diagrams: A Real Example 249
26.6. U1501 Byte Decode PAL 250
U1501 Pinout 250
U1501 Output signals 251
U1500 Buffer and U1505 DIP 253
26.7. U1608-U1603 Video Controller 253
26.8. U1700-01 Video RAS/CAS Latches 254
26.9. Frame Buffer RAM 255
26.10. ECL Circuitry 256
ECL Clock 256
26.11. Horizontal and Vertical Synch State Machines 257

Chapter 27 VMEbus — Performance **263**
27.1. 2060 VME Implementation 263

Chapter 28 VME Arbiter and Requester **269**
28.1. Terminology for VME Arbiter and Requester 269
28.2. U2704 VME Arbiter and Requester 271
State Machine as Arbiter and Requester 272
Transitions from BUSREQ State 273
Transitions from MASTER State 274
Transitions from MASTER_NG State 276

Transitions from BUSGRANT State 276
28.3. State Machine as Requester Only 277
Transitions from IDLE State 277
Transitions from BUSREQ State 278
Transitions from MASTER State 279
Transitions from MASTER_NG State 280
Transitions from the BUSGRANT State 281

Chapter 29 VME Master Interface 285
29.1. VME Select and Freeze PAL, U2701 285
Pinout for the U2701 PAL 286
Terminology for the VME Select and Freeze PAL 287
CPU Reruns on VME "Short Timeouts" 288
CPU Reruns on Deadlocks 288
29.2. VME Select and Freeze State Diagram 288
Normal Operation 288
Deadlock Resolution 289
VME Short Timeouts 290
VME Long Timeouts 290
29.3. VME Master Controller PAL U2806 291
Terminology for VME Master Controller 293
VME Master Controller State Machine 293
CPU Retains Control of VMEbus at End of Cycle 294
CPU Relinquishes Control of VMEbus at End of Cycle 295
CPU Freeze Cycles 295
Address Modifiers and P_BLWORD 295
Non-Aligned Master Cycles 296

Chapter 30 VME Slave Interface 299
30.1. VME Slave Address Latches, U2901-2 and U2911-13 299
30.2. VME Slave Address Decoder U2907 299
Terminology for the VME Slave Address Decoder, U2907 301
30.3. User DVMA Enable (U2905-6) and Context Registers (U509) 302

30.4. VME Slave Address Multiplexers (U2910:09) 302
30.5. VME Slave Request PAL (U2904) 302
Terminology for VME Slave Request PAL 305
VME Slave Request State Machine 306

Chapter 31 VME Data Buffers, U3000 to U3006 ..... **311**
31.1. 16-Bit Operation 311
31.2. 32-Bit Operation 312
31.3. CPU Cycles 312
31.4. DVMA Cycles 312

Chapter 32 Direct Virtual Memory Access **315**
32.1. A Generic DVMA Cycle 315
32.2. Optimizations to the DVMA Cycle 316
Back-to-Back DVMA 316
Ethernet Hold 316
VMEbus Lock 316
32.3. Refresh as a Special Case 316
32.4. The DVMA Strobe PAL (U2410) 317
Input and Output Signals 319

Chapter 33 Sample Cycles **323**
33.1. VME Master Cycles 323
CPU Access of Idle VMEbus 323
CPU Access of a Busy VMEbus 324
CPU Access of VMEbus, Currently Bus Master 325
CPU Rerun During VME Access 325
Relinquishing the VMEbus 326
33.2. VME Slave Cycles 327
VME Device Accesses P2 Bus 327
Lock Mode Cycles 328
VME Device Initiates P2 Bus Lock 328
VME Device Ends P2 Bus Lock 329

VME Device Not Fast Enough to Initiate P2 Bus Lock 329
33.3. Ethernet Cycles 329
33.4. Refresh Cycles 330

Chapter 34 RAS Decode PALs — U3100 and U3102 ..... **335**
34.1. U3100 and U3102 Pinouts 335
34.2. U3100 and U3102 Input Signals 336
34.3. U3100 and U3102 Outputs 336
U3100 Output Signals 337
U3102 Output Signals 339

Chapter 35 CAS Decode PAL — U3104 ..... **345**
35.1. U3104 Pinout 345
35.2. U3104 Input Signals 346
35.3. U3104 Output Signals 346

Chapter 36 Control Buffers — U3105 and U3115 ..... **351**

Chapter 37 Row and Column Address Multiplexers — U3110:07 ..... **355**

Chapter 38 Memory RAM — *Pages 32 and 33* **359**
38.1. Memory Read 359
38.2. Memory Write 359
38.3. Processor Data Acquisition 359

Appendix A Figures and Timing Diagrams **363**

Tables

Table 1 Component Designators xxv

Table 1-1 Control Space Devices and Their Addresses 4
Table 1-2 I/O Devices and Their Addresses 5
Table 1-3 TYPE2 Space 6
Table 1-4 TYPE3 Space 6
Table 1-5 On-Board Interrupts 7

Table 5-1 U105:2 Processor Data Buffers — Data Flow 22
Table 5-2 J100 — Cache Disable and Enable 23
Table 5-3 Sun-3 Function Code Address Space 24
Table 5-4 CPU Space Cycles 25
Table 5-5 Coprocessor Designation 25

Table 10-1 Data Size Encodings: (p2_siz[1:0]) 60
Table 10-2 Base Offset Encodings: (p2_a[01:00]) 62
Table 10-3 Dynamic Bus Sizing: How dsack[1:0] Decode Port Size 63

Table 11-1 Interrupt Register — Signal Designations 68

Table 12-1 Low Priority Acknowledge Bit Encodings: lp_ack(1:0) 77
Table 12-2 High Priority Acknowledge Bit Encodings: hp_ack(1:0) 78
Table 12-3 Type of Interrupts Encoded by hp_ack(1:0) and lp_ack(1:0) 78
Table 12-4 Lower Priority Acknowledge Signals 81

Table 12-5 Lower Interrupt Priority Encode Signals 83
Table 12-6 Higher-Level Priority Acknowledge Signals 86
Table 12-7 Higher-Level Interrupt Priority Encode Signals 88
Table 12-8 U304: Second-Level Acknowledge Signals 91
Table 12-9 U304: Second-Level Interrupt Priority Level Signals 91
Table 12-10 Spurious Ethernet Interrupts 93

Table 17-1 U509 Context Register — Description 115

Table 18-1 U508 Segment Map Buffer — Data Flow 123

Table 19-1 Format of a Page Map Entry 127
Table 19-2 MMU Statistics Bits 127
Table 19-3 Type-Bit Decode 128
Table 19-4 Byte Selection in the Page Map RAM 128

Table 21-1 MMU Protection Bits — Decode of the Inputs 141
Table 21-2 Truth Table for MMU Protection Bits 141

Table 22-1 U700 Comparator 147

Table 23-1 Parity State Diagram — State Values 157
Table 23-2 Parity State Diagram — Description of the States 158
Table 23-3 Memory Error Register — U801 162

Table 24-1 Map for TYPE1 Space 165
Table 24-2 U901 Control Counter States 172
Table 24-3 U901 Wait Counter States 172
Table 24-4 EPROM Jumpering 190

Table 25-1 TTL Bus DTACK State Machine States 200
Table 25-2 Byte Selection in Parity Address Selection 204
Table 25-3 Byte Decode of the Parity Error and Address Registers 206
Table 25-4 Byte Selection in the Page Map RAM 211

Table 25-5 Contents of the ID PROM ..... 2?
Table 25-6 U1404 P2-to-TTL Data Buffer — Data Flow ..... 22?

Table 29-1 U2806 Transfer Logic 294
Table 29-2 Address Modifier Bits on the 2060 Board 296

Table 32-1 MC68020 Data Size Output Encodings 320
Table 32-2 MC68020 Function Code Output Encodings 320

Table 38-1 Memory Data Buffers — Data Flow 359

Figures

Figure 3-1 2060 Block Diagram 15

Figure 5-1 Virtual Address Space 25
Figure 5-2 U106 Pinout 26
Figure 5-3 U107 Pinout 30

Figure 8-1 Sun-3 Connectors on the 2060 CPU Board 45
Figure 8-2 U201 Pinout 46

Figure 9-1 U202 Pinout 52

Figure 10-1 U204 Pinout 59
Figure 10-2 Byte Data Alignment Within the 32-bit Bus Space† 61
Figure 10-3 Word Data Alignment Within the 32-bit Bus Space† 61
Figure 10-4 3-Byte Data Alignment Within the 32-bit Bus Space† 61
Figure 10-5 Longword Data Alignment Within the 32-bit Bus Space† 61
Figure 10-6 Dynamic Bus Sizing — Transfer Offsets 62

Figure 12-1 Interrupt Request Cycle 74
Figure 12-2 Interrupt Acknowledge Cycle 75
Figure 12-3 Interrupt and Acknowledge Cycle 76
Figure 12-4 U302 Pinout 79
Figure 12-5 U303 Pinout 84
Figure 12-6 U304 Pinout 89

Figure 14-1 U400 Pinout 101
Figure 14-2 Clock Stretch (cs4 — cs4.5) State Diagram 102

Figure 15-1 U408 Pinout 107

Figure 16-1 Sun-3 Address Translation 111
Figure 16-2 Sun-3 MMU 112

Figure 17-1 Context Bits and Virtual Address 115
Figure 17-2 U509 Pinout 116

Figure 18-1 Segment Map RAM — U507:00 121

Figure 23-1 U802 Pinout 159
Figure 23-2 U802 Parity Control PAL — State Diagram 160

Figure 24-1 MOS Decoders 166
Figure 24-2 U900 Pinout 167
Figure 24-3 U901 MOS Control State Machine Pinout 170
Figure 24-4 MOS Control State Machine 171
Figure 24-5 MOS DTACK PAL Wait State Counter 175
Figure 24-6 U904 Pinout 179
Figure 24-7 U902 MOS Write Data Buffer 184
Figure 24-8 U903 MOS Read Data Buffer 185

Figure 25-1 U1400 Pinout 196
Figure 25-2 TTL Bus DTACK (SACK) State Machine Diagram 199
Figure 25-3 U1401 Pinout 205
Figure 25-4 U1402 Pinout 212
Figure 25-5 U1403 Pinout 219
Figure 25-6 U509 Pinout 227

Figure 26-1 U1504 Pinout 235
Figure 26-2 U1502 Pinout 237

Figure 26-3 U1503 Pinout 241
Figure 26-4 Handshaking in the P2 Interface State Machine 245
Figure 26-5 U1501 Pinout 250
Figure 26-6 Data Path — From Frame Buffer to CRT 256

Figure 28-1 U2704 Pinout 271
Figure 28-2 VME MASTER State — Relationship of S4SEL to B_SSEL 275

Figure 29-1 U2701 Pinout 286
Figure 29-2 U2806 Pinout 292

Figure 30-1 U2907 Pinout 300
Figure 30-2 U2904 Pinout 304

Figure 32-1 U2410 Pinout 318

Figure 34-1 U3100 and U3102 Pinouts 335

Figure 35-1 U3104 Pinout 345

Figure 37-1 RAS/CAS Decode Bit Assignments for 2 and 4 Mbyte Systems 355

Preface

This document provides a detailed explanation of the 2060 board hardware. The Sun-3/2060 board uses positive logic.

Signal Names

Both capitalized and uncapitalized names of signals are used in this text; they are synonymous. For instance, readen- and READEN- are names for the same low active signal (see the glossary for definition of "low active").

Glossary

A few terms are used throughout this document which, without explanation, may seem confusing.

- Positive Logic — positive logic means that the asserted level (see below) of a signal is a logic 1 (see below also), 2.8 to 4.5 volts for a TTL gate.
- Asserted — when we say that a signal is "asserted," we mean that it is in it ACTIVE, or true, state. In positive logic this means that a signal like READ, when asserted, is equal to its most positive state. When a signal like WRITE*, WRITE-, or WRITE\ (the three are synonymous) is asserted it is equal to its most negative state.
- Activated — means the same as "asserted."
- Logic 1 — in positive logic, a logic 1 stands for the more positive of the two voltage levels. A logic 1 in negative logic stands for the more negative of the two voltage levels.
- Logic 0 — in positive logic, a logic 0 stands for the more negative of the two voltage levels. A logic 0 in negative logic stands for the more positive of the two voltage levels.
- Set — means the same as logical 1.
- Clear — means the same as a logical 0.
- High Active — refers to the level a signal is when it is "true"; in positive logic, a high active signal is true at its most positive state. For instance, p2_rw, the processor read and write signal, has a high active read state.
- Low Active — refers to the level a signal is when it is "true"; in positive logic, a low active signal is true at its most negative state. For instance, p2_rw, the processor read and write signal, has a low active write state.

- ON — when it refers to a switch (or switch section) setting, is synonymous with CLOSED. This means that the signal at the input of the switch (or switch section) is shorted to its output.
- OFF — when it refers to a switch (or switch section) setting, is synonymous with OPEN. This means that the signal at the input of the switch (switch section) is NOT SHORTED (signal is not passed) to its output.
- CLOSED — when it refers to a switch (or switch section) setting, is synonymous with ON. This means that the signal at the input of the switch (switch section) is shorted to its output.

  When referring to a latch, it means that data is *not* flowing through the latch.
- OPEN — when it refers to a switch (or switch section) setting, is synonymous with OFF. This means that the signal at the input of the switch (switch section) is NOT SHORTED (signal is not passed) to its output.

  When referring to a latch, it means that data is flowing through the latch.
- LATCHED — means that the data is held in the latch (that is, the latch is closed).
- DIP — stands for Dual In-line Package, and refers to the physical geometry of the chip (rectangular, with pins on the two longer sides).
- DIP Switch — a multi-sectioned switch which has DIP geometry.
- Switch — a device for making or breaking an electrical circuit. A switch may have one or more sections, each of which may control a circuit.
- 0x — hexadecimal prefix; the number following this prefix is in hexadecimal.
- PCB — printed circuit board
- TTL — transistor-to-transistor logic.
- RX — receiver
- TX — transmitter
- A(17:12) — indicates a range of signals, in this case address lines 17 through 12.
- R/W — read/write

Signal Levels

The 2060 board uses positive logic; low active signal names can have either a suffix or a prefix, depending upon the convention with which the engineer was most familiar.

For instance, the low active "write" signal from the 68020 might be labelled:

- p_write-
- /p_write
- p2_write_ *(usually found in PAL listings)*

- p_write\
- p_write*.

All of these labels refer to the same signal and mean the same thing.

Component Designators

Table 0-1 *Component Designators*

Designation	*Component*
U	Integrated circuit resistor dips
C	capacitor
R	discrete resistor
J	jumpers/DIPswitches
P	connector

Each component is assigned one of these designators. The designator is one of the above letters followed by four decimal digits.

1. The first two digits generally refer to the page number on which the component is found. In the case of packages which contain more than one component (for example, F00) the first two digits indicate the page on which the package is first used.
2. The last two digits distinguish components on the same page. Sometimes these values will "skip." Generally the numbers are assigned in columns which go from left to right.

Pullups/Pulldowns

Pullups are indicated by "pu" followed by a functional designator, such as "puv7," which indicates a pullup used in the video section. Other pullups are indicated by "h" followed by a numerical designator, such as "h0."

Pulldowns are indicated by an "pd" followed by a numerical designator, such as "pd0."

PALs/PAL Listings

Inputs to PALS are shown (in the schematics) on the left side, outputs on the right.

All of the inputs and outputs are declared at the beginning of the listing. Negative true inputs and outputs are usually preceded by a foreslash (for example, "/write") or succeeded by a trailing underscore (for example, "write_"). These two write signals are synonymous.

In the equations, the negation of a signal is shown by preceding the signal with '!'. A logical AND is designated in the PAL equation by a star "*" or an ampersand "&"; a logical OR may be either a plus sign "+" or a pound sign "#."

1

An Overview of the Sun-3 Architecture

An Overview of the Sun-3 Architecture 3

1.1. CPU 3
1.2. System DVMA 3
1.3. VME Slave User DVMA 3
1.4. Refresh 4
1.5. DVMA Controller 4
1.6. Control Space Devices — FC3 4
1.7. Memory Management Unit 4
1.8. Device space 4
1.9. Memory Space (TYPE0 space) 5
Parity Main Memory 5
Video Memory 5
1.10. I/O Devices (TYPE1 space) 5
1.11. VME Master (TYPE2 and TYPE3 Space) 6
1.12. Interrupts 6
On-Board Interrupts 6
VME Vectored Interrupts 7
1.13. CPU Resets and Timeout 7

1

An Overview of the Sun-3 Architecture

This section is divided into five parts:

- the first describes the CPU and DVMA devices,
- the next two describe the Control Space (68020 extensions and MMU) and all devices which must be accessed through the MMU,
- the last two describe interrupts, resets, and timeouts.

NOTE *In this Chapter, references to the SUN-3 architecture manual (Rev. 2.0) are enclosed within square brackets [].*

1.1. CPU

The processor is a 68020 running at 16.67 MHz. All bus cycles incur a minimum of 1.5 wait states (except FPA and 68881 cycles). S4 is stretched by 30 nsec to cause the half wait state.

In conjunction with the CPU is an optional 68881 Floating Point Processor. A separate clock is available to allow the FPP to run asynchronously to the 68020.

All CPU space cycles are implemented as in section [3.1]. Disabled (System Enable register bit D6=0) FPP coprocessor cycles are terminated with an immediate bus error. All other coprocessor addresses and accesses to an enabled but uninstalled FPP result in a Timeout bus error. Interrupt Acknowledge cycles and installed and enabled FPP cycles terminate normally with DSACK or are aborted with a synchronous bus error.

1.2. System DVMA

The two system DVMA devices are the Ethernet Interface (Intel 82586) [5.14] and the VME Slave System DVMA [6.2]. Both use supervisor data function codes and are implemented as in the SUN-3 architecture manual.

The Ethernet Interface has one feature which other DVMA devices do not implement — FIFO operation of the 82586 requires that the Ethernet Interface be able to retain bus mastership, therefore the 82586 can issue a HOLD signal along with bus request.

1.3. VME Slave User DVMA

This is implemented as described in the SUN-3 architecture manual section [6.2]. User DVMA is performed in user data function codes. There is no response to any attempted access to a disabled context.

4. Refresh

The refresh timer requests the bus via the DVMA controller just like the other DVMA devices. Once the bus has been obtained for a refresh operation, the controller does not execute a DVMA cycle but instead executes a refresh cycle. A refresh strobe, REFR, is issued instead of AS– so that the refresh cycle will not conflict with any other address space cycle.

1.5. DVMA Controller

DVMA/CPU device priority is as follows:

1. Refresh — nothing can stop a refresh cycle
2. Ethernet — can issue bus hold to lock out priorities 3 and 4
3. VME System/User DMA — dynamic bus hold feature to lock out 4
4. 68020/68881

1.6. Control Space Devices — FC3

The following Control Space devices are implemented [4.1]; all devices are byte-read and byte-write except for the bus error register which is byte-read only. The ID PROM, Page Map, and Segment Map are implemented as an array of bytes. This allows word and longword accesses via the 68020 dynamic bus sizing capability.

Table 1-1 *Control Space Devices and Their Addresses*

ADDRESS	DEVICE
[0x00000000] + Virtual	ID PROM
[0x10000000] + Virtual	Page Map
[0x20000000] + Virtual	Segment Map
[0x30000000]	Context Register
[0x40000000]	System Enable Register
[0x50000000]	User Enable Register
[0x60000000]	Bus Error Register
[0x70000000]	Diagnostic Register
[0x80000000] *to* [0xE0000000]	Non-responding addresses which will cause a timeout bus error
[0xF0000000]	MMU bypass access to Serial Port for diagnostics

1.7. Memory Management Unit

The MMU will be implemented as described in the SUN-3 Architecture Manual, section [4.3], with the following exception: cache bits are read back as zero since they are not implemented.

1.8. Device space

The device space includes everything accessed through the MMU — main memory, video memory, the system bus, and the I/O devices.

1.9. Memory Space (TYPE0 space)

Parity Main Memory

Main memory is implemented as described in section [5.2]. A positive acknowledge scheme is used so that non-existent memory locations result in a timeout bus error.

120 nsec 256K-by-1 DRAMs are used to implement the parity memory. Accesses to this memory incur 1.5 wait states on reads and writes. There is a minimum of two megabytes of memory on the CPU board (a maximum of 4 megabytes) and additional memory on the Expansion boards.

Video Memory

Video memory is a 128 Kbyte block of memory starting at location 0xFF000000. Copy Mode (if enabled) causes any write operation to a 128 Kbyte block of memory, starting at location 0x00100000, to also be written into the video memory.

Video display format is of two types. The first is a 1152-by-900 pixel format and the second is a 1024-by-1024 (1K-by-1K) format. The Vertical rate will be 67 Hz, and the pixel rate will be 10 nsec per pixel.

Outputs to the video monitor are as follows:

1. Serial Video — differential ECL
2. Horizontal Sync — positive TTL pulse, sync on rising edge
3. Vertical Sync — positive TTL pulse, sync on rising edge

1.10. I/O Devices (TYPE1 space)

The following devices are implemented in TYPE1, 21-bit address space as per the Sun 3 Architecture Manual [5.2, 5.4 to 5.14]:

Table 1-2 *I/O Devices and Their Addresses*

ADDRESS	DEVICE
[0x00000000]	Keyboard/Mouse interface
[0x00020000]	Serial I/O ports
[0x00040000]	EEPROM
[0x00060000]	Time of Day Clock
[0x00080000]	Parity Error registers
[0x000A0000]	Interrupt register
[0x000C0000]	Ethernet Control register
[0x000E0000]	Non-responding address which will cause a timeout bus error
[0x00100000]	EPROM
[0x00120000] *to* [0x001A0000]	Non-responding addresses which will cause a timeout bus error
[0x001C0000]	Encryption processor
[0x001E0000]	Non-responding addresses which will cause a timeout bus error

The Parity Error registers, EPROM, and the EEPROM appear as an array of bytes. This allows usage of the 68020 dynamic bus sizing capability in accessing these devices.

The Encryption processor is an option. To comply with U.S. Customs law, both the 9518 DCP and support PAL will reside in sockets. The absence of the PAL will cause a timeout error.

The Time of Day clock provides the level 7 non-maskable interrupt. The same interrupt can also provide a level 5 interrupt [5.7: Int. reg].

The EEPROM has a 10 msec per byte write overhead. It is software's responsibility not write to the EEPROM faster than 10 msec/byte.

1.11. VME Master (TYPE2 and TYPE3 Space)

CPU accesses to the VMEbus will be through TYPE2 space for 16-bit data transfers and TYPE3 space for 32-bit data transfers. The 32-bit address will be decoded to supply the VME Address Modifier bits and define the VME address size.

Table 1-3 *TYPE2 Space*

TYPE2		
32-bit Address	*VMEbus with 16-bit data*	*AM5:3 (H) Address Modifiers*
[0x00000000]	VME 32-bit address space	(L L H)
[0xFF000000]	VME 24-bit address space	(H H H)
[0xFFFF0000]	VME 16-bit address space	(H L H) I/O access only

Table 1-4 *TYPE3 Space*

TYPE3		
32-bit Address	*VMEbus with 32-bit data*	*AM5:3 (H) Address Modifiers*
[0x00000000]	VME 32-bit address space	(L L H)
[0xFF000000]	VME 24-bit address space	(H H H)
[0xFFFF0000]	VME 16-bit address space	(H L H) I/O access only

1.12. Interrupts

On-Board Interrupts

On-board interrupts are autovectored on all levels except for level 6 where the 8530 SCCs provide a vector.

Table 1-5 *On-Board Interrupts*

LEVEL	DEVICES
7	NMI- Real Time Clock and Parity Error
6	All Serial Controllers (8530As)
5	Real Time Clock
4	Video vertical interrupt
3	Ethernet, System enable register 3
2	System enable register 2
1	System enable register 1

VME Vectored Interrupts

VME interrupts are vectored and at lower priority than on-board interrupts.

1.13. CPU Resets and Timeout

1. Power On Reset: see [7.0].
2. Watchdog Reset: see [7.0]. A user-accessible panic button (RESET) will also force a watchdog reset.
3. CPU Reset: see [7.0]. In addition, access to the VMEbus will be inhibited for the 200 msec min. SYSRESET- period.
4. CPU Board Timeout: Minimum of one refresh period, maximum of two. All non-responding addresses and devices will result in a timeout bus error.

2

VME Compliance

VME Compliance **11**

2.1. Options 11
2.2. Performance Parameters 11

2

VME Compliance

2.1. Options

1. Multiple Arbiters: A jumper is provided; if installed, it gives arbitration control to another VME device.
2. Arbiter Option: ONE, Only BR3- will be monitored.
3. Requester Option: ROR, Release on request
4. Timeouts: Two VME Master timeouts are provided. The first is a "retry" period of 2.88 µsec at which time the VME interface "freezes" and other DVMA devices (Refresh, Ethernet) can obtain the local bus. After 256 retries (737 µsec), a timeout error will occur. This provides a timeout when the CPU board is master. No timeout will be provided for VME Slave or VME User mode since it is each master's responsibility to provide its own timeout.
5. Backoff Mechanism: If the CPU initiates an access to the VME at the sam time that a VME device accesses the P2, the CPU cycle will back off and be re-run.
6. Non-implemented features: Since multiprocessing will not be allowed on our systems, READ-MODIFY-WRITE is not implemented. The ACFAIL- timing during power down will not meet spec.

2.2. Performance Parameters

The following performance parameters assume a 60 nsec clock and 1.5 wait states on read and write cycles.

1. CPU to VME latency (assume ideal VME device)

not currently bus master	600-660 nsec
currently VMEbus master	420-480 nsec

2. CPU to VME bandwidth (assume ideal VME device)

burst rate	8.3-9.5 MBytes/sec 480-420 nsec/longword

3. VME to P2 latency (not currently P2 bus master, assume idle P2 bus)

AS to DTACK	570-630 nsec

4. VME to P2 bandwidth (assume P2 bus locked)

bandwidth	6.3-8.9 MBytes/sec 635-450 nsec/longword

5. VME to VME transfer

time to acquire VMEbus	70-155 nsec
bandwidth	limited by VME spec and VME devices

3

Block Diagram

Block Diagram **15**

3.1. Data Paths 16

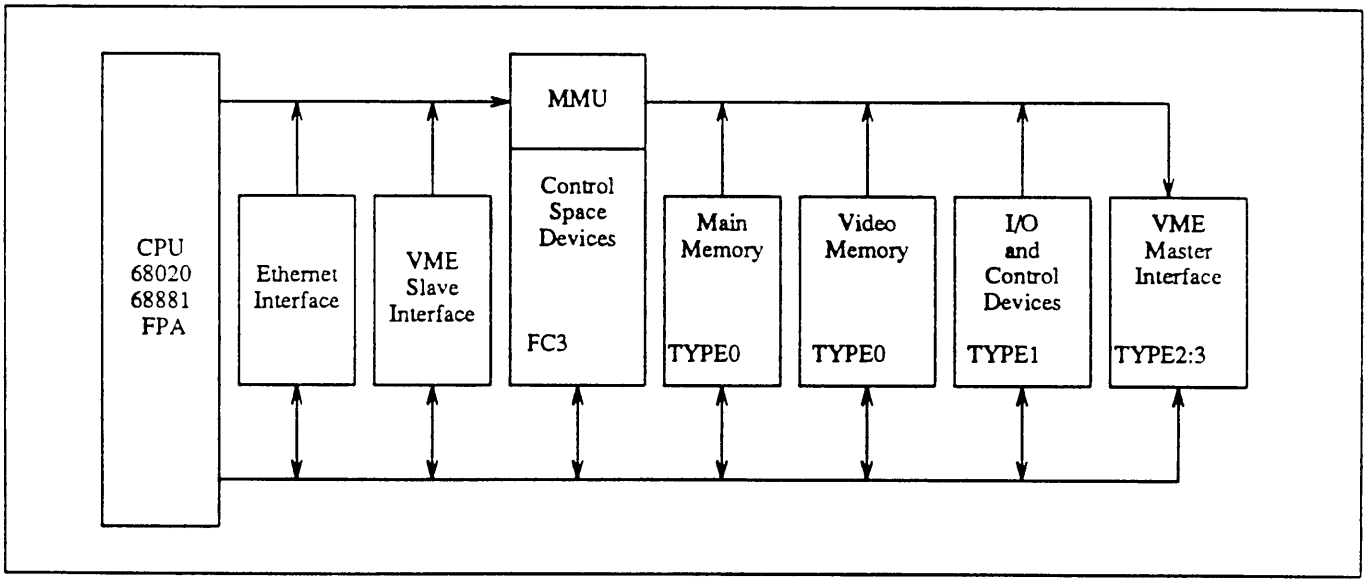
3

Block Diagram

The following figure, "2060 Block Diagram" shows a simplified block diagram of the SUN 2060 system. CPU and DVMA devices are on the left side. They supply a virtual address to the MMU and arbitrate for control via the DVMA controller. Control space devices are located in the center. These are the CPU extensions and are accessed in FC3 space. The MMU translates the virtual address into a physical address (Device Space) that is used by the devices accessed through the MMU. This Device Space is divided into four types:

- type0 for main and video memory,
- type1 for I/O and Control devices, and
- type2:3 for the VME Master interface.

Figure 3-1 *2060 Block Diagram*



3.1. Data Paths

See the figure, "2060 Data Busing," which provides details about data bus connections. There are two bus sizes, a 32-bit and an 8-bit. The 32-bit bus provides a high-bandwidth path between the CPU, DVMA devices and main memory. An 8-bit bus size is used to reduce board routing problems. This works because most of the Control and Device Space devices are 8 bits. The Parity Latch and Page Map interface bandwidth will be lower than maximum because of the 8-bit bus restriction but accesses to these devices will be infrequent and the loss of bandwidth not noticeable. The dynamic bus sizing capability of the 68020 is used so that longword moves can be made to these two devices.

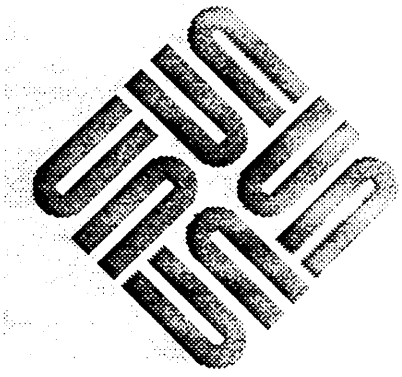
To segregate the MOS devices from the TTL devices, two separate 8-bit buses are used. The two types of devices have different data bus interfacing capabilities: MOS devices have weak bus drivers and are sensitive to undershoot while TTL devices have the opposite characteristics. The separation of the two technologies thus improves system reliability.

4

Mechanical Specifications

Mechanical Specifications **19**

4.1. Board Form Factor 19
4.2. Connectors 19
4.3. Switches 19



4

Mechanical Specifications

4.1. Board Form Factor

The CPU and Expansion boards will conform to the triple height Eurocard specification. This will allow either board to plug into a 75 or 160 chassis.

Eurocard dimensions are:

```
Height   366.67 mm
Width    400.00 mm
```

4.2. Connectors

There are eight connectors on the CPU board.

1. P1, P2 and P3 — 96-pin VMEbus connectors.
2. 9-pin video output,
3. 15-pin Ethernet,
4. two 25-pin serial ports, and
5. 15-pin long distance keyboard and mouse connector.

The basic Expansion board has the three VMEbus connectors: P1, P2, and P3. Additional connectors depend on what other functions (besides memory) are on the board.

4.3. Switches

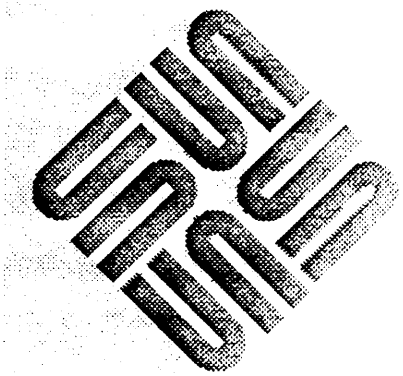
There are two user-accessible switches. One is the diagnostic switch which is used to enter and exit diagnostic mode. The other is the user reset switch, which causes a watchdog reset.

5

68020, 68881 Floating Point Coprocessor, and Associated Circuitry

68020, 68881 Floating Point Coprocessor, and Associated Circuitry 23

5.1. Processor Data Buffers — U105:2 23
U105:2 Data Flow 23
5.2. Cache Disable — J100 23
5.3. CPU Space PALs — U106 and U107 24
5.4. U106 PAL 26
U106 Input Signals 26
U106 Output Signals 27
5.5. U107 PAL 30
U107 Input Signals 31
U107 Output Signals 31



5

68020, 68881 Floating Point Coprocessor, and Associated Circuitry

Page one of the schematics contains the 68020 microprocessor and its 68881 floating point coprocessor. Detailed information about these can be found in their respective User's Manuals.

5.1. Processor Data Buffers — U105:2

To the left of the 68881 are four processor data buffers, U105:2, which isolate the P2 data bus from the data inputs of the 68020/68881; they also serve to reduce the loading on the data inputs of the 68020/68881. These data buffers are bidirectional; output is enabled from them when p_bufen- goes active (low).

Output of these buffers is normally enabled (p_bufen- is low) except during two situations:

1. the 68881 is performing a cycle, or
2. a DMA cycle is being performed.

This enabling/disabling action is controlled by the p_bufen- signal from U107 CPU space PAL.

U105:2 Data Flow

Direction of the data flow is controlled by the processor read/write signal, p_rw; when p_rw is high (a read), data flows from the P2 data bus to the processor; when it is low (write) data flows from the processor to the P2 data bus. The truth table for the U105:2 buffers is:

Table 5-1 *U105:2 Processor Data Buffers — Data Flow*

Gate *p_bufen-*	Direction *p_rw-*	Which way the data will flow
0	0	processor to P2 data bus (B -> A)
0	1	P2 data bus to processor (A -> B)
1	X	*outputs are tri-state*

5.2. Cache Disable — J100

Jumper J100 on page 1 of the schematics serves to enable/disable the cache memory. When J100 is IN, cache is disabled; when J100 is OUT, cache is enabled. Usually this jumper is not installed (cache is enabled).

Table 5-2 *J100 — Cache Disable and Enable*

Jumper *In or Out?*	Cache Status *Enabled or Disabled?*
In	Cache disabled
Out	Cache enabled

5.3. CPU Space PALs — U106 and U107

Each of these PALs, U107 and U106, decode address lines and function codes to generate the space-select and control signals. The three function code bits, FC2:0, decode to one of eight address spaces:

Table 5-3 *Sun-3 Function Code Address Space*

Function Code			Address Space
FC2	FC1	FC0	
0	0	0	Reserved
0	0	1	Device Space (User Data)
0	1	0	Device Space (User Program)
0	1	1	Control Space
1	0	0	Reserved
1	0	1	Device Space (Supervisor Data)
1	1	0	Device Space (Supervisor Program)
1	1	1	CPU Space

Looking at this table, you will notice that address space is divided into three kinds of space:

1. Device Space — Function Codes 1, 2, 5, and 6
2. Control Space — Function Code 3
3. CPU Space — Function Code 7.

NOTE *Attempted access to address space for function codes 0 and 4 is illegal.*

U107:6 PALs issue the appropriate control signals for CPU space cycles — that is, the address space selected by FC7. Two kinds of CPU space cycles are used in the Sun-3 architecture:

1. coprocessor (FPA/68881), and
2. interrupt acknowledge.

Address bits A17 and A16 are used to select between coprocessor and interrupt acknowledge cycles.

Table 5-4 *CPU Space Cycles*

Address Bits *A17*	*A16*	Type of CPU Space Cycle
1	0	Coprocessor
1	1	Interrupt Acknowledge

Address bits A15:13 are used to select from one of eight possible coprocessors. However only one coprocessor is a legal designee: the floating point chip, 68881. It is designated by A15:13 = 001 — see the table below.

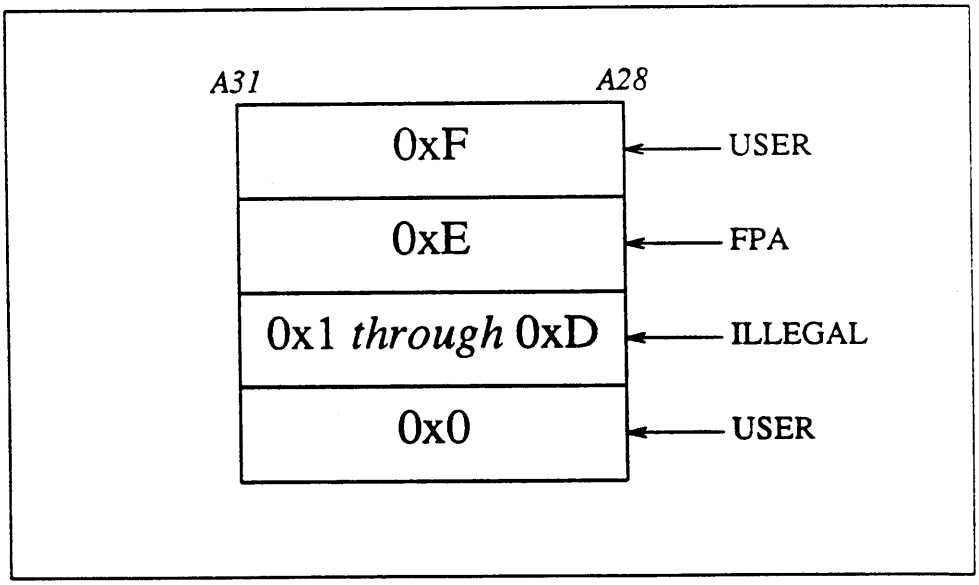
Table 5-5 *Coprocessor Designation*

Address Bits *A15*	*A14*	*A13*	Coprocessor Selected
0	0	1	68881 (FPP)

Although all 32 processor address bits have meaning (in that they are decoded), only the lower 28 address bits are translatable through the MMU. Thus, virtual address space is defined as sixteen contiguous 256 Mbyte (28-bit) virtual address spaces. These sixteen address spaces are decoded by the four high order address bits not translated through the MMU, A(31:28).

User space is defined as either A31:28 = 0x0, 0xE (in the case where you are trying to access the FPA), or 0xF. If a user program attempts to access one of the areas in between — that is A31:28 = 0x1 through A31:28 = 0xD — the processor will timeout and no acknowledge will be returned.

Figure 5-1 *Virtual Address Space*

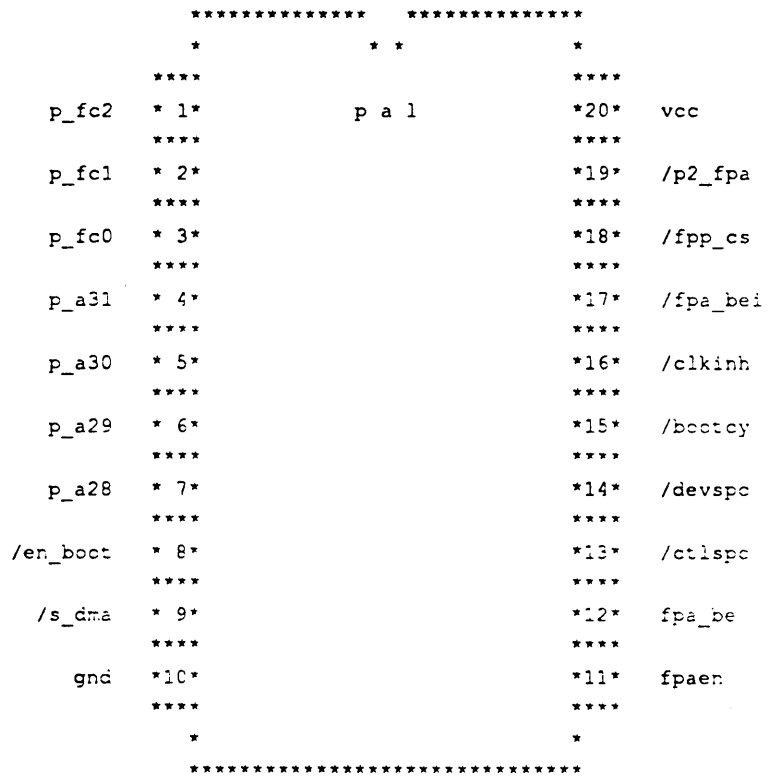


U106 PAL

U106 decodes the four most significant address lines, A31:28, the function code bits FC2:0, and other control and enable signals, to issue various control signals.

Pinout of the U106 PAL is:

Figure 5-2 *U106 Pinout*



U106 Input Signals

Inputs to the U106 PAL are:

```
p_fc[2:0]  = unbuffered processor function codes

p_a[31:28] = unbuffered processor virtual address bits

en_boot-   = special boot state - must go to EPROM

s_dma-     = cycle is a DVMA cycle
                (timing must be same as p_fc[2:0])

fpaen      = FPA is enabled (from System Enable Register)

fpp_cs-    = processor is doing an FPP (68881) cycle.
```

U106 Output Signals

Outputs from the U106 PAL are:

```
bootcy-  =  indicates a supervisor program access in boot state

devspc-  =  processor is doing a device space cycle,
              FC1, 2, 5, or 6

ctlspc-  =  processor is doing a control space cycle,
              FC3

p2_fpa-  =  processor is doing an FPA cycle

fpa_berr =  attempting access to FPA that has not
              been enabled

clkinh-  =  inhibit clock (multiphase) clock generator; used
              for clock stretch through PAL U400
```

The PAL equation for the bootcy- signal is:

```
bootcy-  = /s_dma*en_boot * p_fc2*p_fc1*/p_fc0 *
           /p_a31*/p_a30*/p_a29*/p_a28 +

           /s_dma*en_boot * p_fc2*p_fc1*/p_fc0 *
            p_a31* p_a30* p_a29
```

This equation indicates that the bootcy- signal will be active (low) when:

- you are not in a DMA cycle,
- you are in boot state,
- you are in FC6, which indicates a superviser program access in device space, and
- the uppermost nibble of address, A31:28, equals 0x0, 0xE or 0xF — accessing a valid portion of the virtual address space.†

†Remember, only A31:28 = 0x0, 0xE (FPA), and 0xF, are legal accesses.

The PAL equation for the devspc- signal is:

```
devspc  =   /p_fc2*p_fc1*/p_fc0 * /p_a31*/p_a30*/p_a29*/p_a28 +

               /p_fc1 *  p_fc0 * /p_a31*/p_a30*/p_a29*/p_a28 +

          /en_boot*p_fc1*/p_fc0 * /p_a31*/p_a30*/p_a29*/p_a28 +

            /p_fc2*p_fc1*/p_fc0 *  p_a31* p_a30* p_a29* p_a28 +

               /p_fc1 *  p_fc0 *  p_a31* p_a30* p_a29* p_a28 +

          /en_boot*p_fc1*/p_fc0 *  p_a31* p_a30* p_a29* p_a28
```

This equation says you may do a valid device space access when:

- the function codes = (1, 2, 5 or 6) and p_a<31:28> = (0x0 or 0xF) as long as you are *not* in boot state (as long as en_boot- is false), or
- the function codes = (1, 2, or 5) and p_a<31:28> = (0x0, or 0xF) while you are in boot state.

The PAL equation for the ctlspc- signal is:

```
ctlspc  = /s_dma * /p_fc2 * p_fc1 * p_fc0
```

This equation says you may do a valid control space access when:

- you are not in a DMA cycle, and
- the function code equals FC3.

The PAL equation for the p2_fpa signal is:

```
p2_fpa  = /s_dma*fpaen * /p_fc2   * p_fc1 * /p_fc0 *
           p_a31*p_a30*p_a29*/p_a28 +

          /s_dma*fpaen * /p_fc1 *  p_fc0 *
           p_a31*p_a30*p_a29*/p_a28 +

          /s_dma*fpaen * /en_boot * p_fc1 * /p_fc0 *
           p_a31*p_a30*p_a29*/p_a28
```

This equation says you may do an FPA cycle when:

- you are not in a DMA cycle,
- the FPA is enabled,
- you are in function code 1, 2, 5, or 6 (when not in a boot state; 1, 2, or 5 when in boot state), and
- the high order address nibble is equal to 0xE (FPA space).

The PAL equation for the fpa_bei- signal is:

```
/fpa_be   = /fpa_beit
```

```
fpa_bei  = /s_dma*/fpaen * /p_fc2 *   p_fc1*/p_fc0 *
           p_a31*p_a30*p_a29*/p_a28 +

           /s_dma*/fpaen * /p_fc1* p_fc0 *
           p_a31*p_a30*p_a29*/p_a28 +

           /s_dma*/fpaen * /en_boot * p_fc1*/p_fc0 *
           p_a31*p_a30*p_a29*/p_a28
```

This equation says you generate a bus error when you attempt to access the FPA when the FPA is not enabled (fpaen is false).

The PAL equation for the clock inhibit signal, clkinh, is:

```
clkinh  = /s_dma*fpaen * /p_fc2   * p_fc1 * /p_fc0 *
          p_a31*p_a30*p_a29*/p_a28 +

          /s_dma*fpaen *          /p_fc1 *  p_fc0 *
          p_a31*p_a30*p_a29*/p_a28 +

          /s_dma*fpaen * /en_boot * p_fc1 * /p_fc0 *
          p_a31*p_a30*p_a29*/p_a28 +

          fpp_cs
```

†Note the /fpa_be = /fpa_bei equation; this output is the wrong polarity, and a demorganized version would not fit. So we provide an inverted output, and then invert it again for the correct polarity.

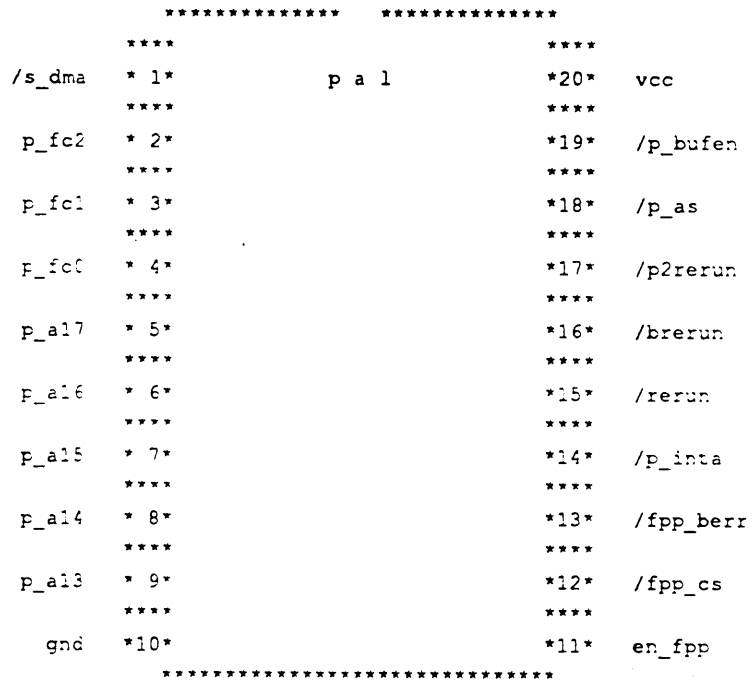
This signal disables the clock generator (U400 and U406) on zero-wait-state cycles (68881 and FPA cycles). This disables states cs4 through cs7, but still allows clock edges at cs2 and cs3.

5.5. U107 PAL

The second half of the CPU Space decoder PALs is U107. It decodes function code bits FC2:0, address bits A17:13, a pair of rerun signals, address strobe, and the FFP enable signal to generate 5 CPU space control signals.

Pinout of the U107 PAL is:

Figure 5-3 *U107 Pinout*



U107 Input Signals

Inputs to the U107 PAL are:

```
s_dma-       =   cycle is a DVMA cycle

p_fc[2:0]    =   unbuffered processor function codes

p_a[17:13]   =   unbuffered processor virtual address

en_fpp       =   FPP enable from System Enable Register

p_as-        =   processor address strobe - acts as a qualifier

b_rerun-     =   rerun request from DVMA logic

sp2_rerun-   =   synchronized rerun request from FPA board
                    via p2 bus
```

U107 Output Signals

Outputs from the U107 PAL are:

```
p_inta-      =  interrupt acknowledge cycle

fpp_berr-    =  quick berr when FPP is not enabled

p_bufen-     =  output enable for bidirectional
                 P2 <-> 68020 data buffers

fpp_cs-      =  chip select for 68881

rerun-       =  logical OR of b_rerun- and sp2_rerun-
```

Remember that CPU space is divided into either

- a coprocessor access, or
- an interrupt acknowledge cycle,

as decoded by address bits A17 and A16. Interrupt acknowledge equals binary 11; coprocessor cycle is decoded by binary 10. A pair of macros in the PAL equation cover these:

```
define INTA_SPACE    = p_a17*p_a16

define COPROCESSOR   = p_a17*/p_a16
```

Also, remember that address bits A15:13 can be decoded to one of eight coprocessors. The only legal coprocessor is the 68881, decoded as 001. A macro in the PAL logic defines this too:

```
define FPP_ID  =  /p_a15*/p_a14*p_a13
```

The p_inta- signal is active (low) when you are doing a CPU space access (FC7) and A17:16 equal 0x3 (both are high). The PAL equation for the interrupt acknowledge cycle signal, p_inta-, is:

```
p_inta  = CPUSPACE*INTA_SPACE
```

A bus error signal is generated immediately if a 68881 cycle is attempted without first enabling the FPP. The PAL equation for the bus error signal, fpp_berr, is:

```
fpp_berr = CPUSPACE*COPROCESSOR*FPP_ID*/en_fpp
```

The signal p_bufen is the output enable (gating signal) for the P2 data buffers, U105:2. The signal is activated every cycle (p_as is valid) unless the FPP is selected, or unless it is a DVMA cycle. The signal is then turned off to avoid the obvious conflict between signals already on the P2 bus and signals being driven onto the P2 bus by the 68881.

```
p_bufen  = /s_dma * /(CPUSPACE*COPROCESSOR*en_fpp) * p_as
```

The p_bufen signal is deasserted to disable the buffers for all coprocessor cycles.

```
p_bufen = /s_dma * p_as * /p_fc2 +
          /s_dma * p_as * /p_fc1 +
          /s_dma * p_as * /p_fc0 +
          /s_dma * p_as * /p_a17 +
          /s_dma * p_as *  p_a16
```

The fpp_cs signal is a chip select for a 68881 cycle. It is valid as long as you are doing a CPU space access (function code equals FC7), you are addressing a coprocessor (A17:16 equal binary 10), address bits A15:13 decode to the FPP (equal a binary 001), the FPP is enabled, and you are not in a DVMA cycle.

```
fpp_cs  = CPUSPACE*COPROCESSOR*FPP_ID*en_fpp*/s_dma
```

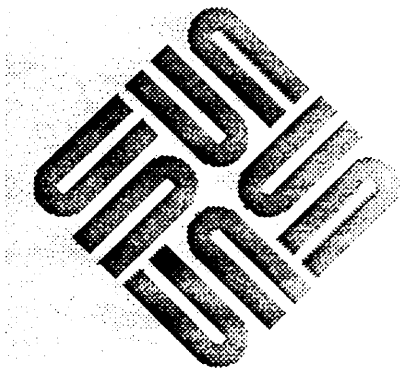
We OR the two sources of rerun (DVMA and FPA) here inside this PAL simply because the necessary pins were available.

```
rerun   = brerun + p2rerun
```

6

Power-on Circuitry

Power-on Circuitry 37



6

Power-on Circuitry

NOTE *The power-on and reset generator is in the upper left-hand portion of page two of the schematics.*

The 2060 board includes a power-on/power-off reset generator that provides an accurate reset pulse. The circuit uses a dual comparator LM393 (U200), a 1.2 volt reference voltage LM385 diode (D201), charge capacitor K200, and resistor network R107:0.

The top comparator forms a power-on reset generator by comparing the voltage from the charge capacitor with the reference voltage. This comparator asserts its output until the voltage across the charge capacitor corresponds to a VCC of 4.5 volts.

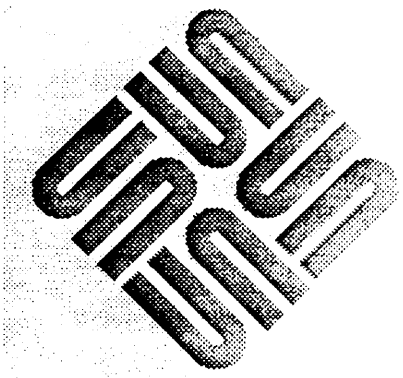
The bottom comparator forms a power-off reset generator by comparing the +5V supply with the reference. This comparator asserts its output when the +5V supply voltage is below 4.5 volt without the charge delay incurred by the first comparator.

The output of both comparators is wire ORed so that signal power-on-reset (por-) is active when either comparator asserts its output.

7

Response Synchronizer — U206

Response Synchronizer — U206 41



7

Response Synchronizer — U206

The two response synchronizer flipflops synchronize the P2 bus error and P2 rerun signals to c60 system clock. The outputs, synchronized P2 bus error (sp2_berr) and synchronized P2 rerun (sp2_rerun-) are activated during cs2 (when cs2 signal to the presets is a high) and the D input to the flip-flops is active (low).

Note that sp2_berr is a high active signal (for software use), so it is taken off the Q/ output of the flip-flop. The sp2_rerun- signal, being low active, is taken off the non-inverting output (Q) of U206.

Both p2_berr- and p2_rerun- are used exclusively by the FPA board.

8

Reset Pal U201 and User Reset Switch U205

Reset Pal U201 and User Reset Switch U205 ..... **45**

8.1. U205 User Reset Switch 45
8.2. U201 Reset PAL 46
U201 Input Signals 46
U201 Output Signals 47

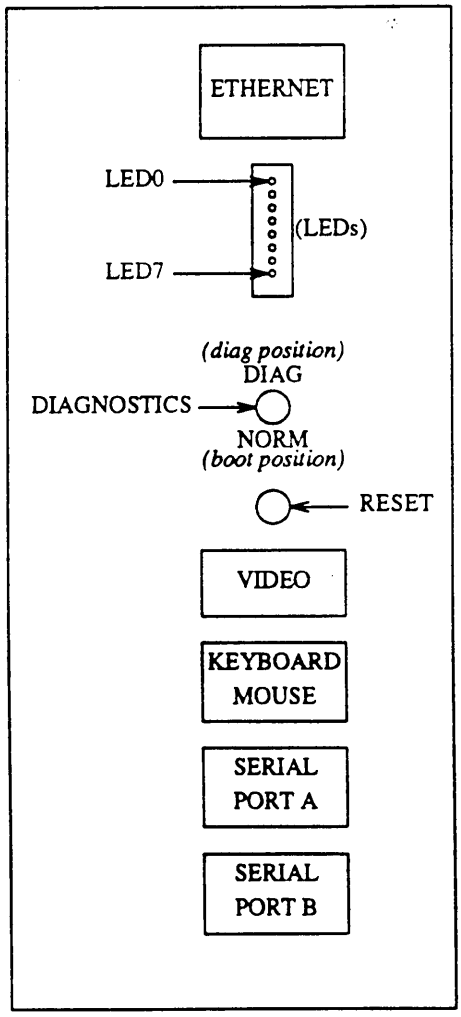
8

Reset Pal U201 and User Reset Switch U205

8.1. U205 User Reset Switch

U205 user reset switch is on the back of the board between the DIAG/NORM switch and the video connector. Pressing this switch forces a watchdog reset, which drops you down into the monitor program.

Figure 8-1 *Sun-3 Connectors on the 2060 CPU Board*

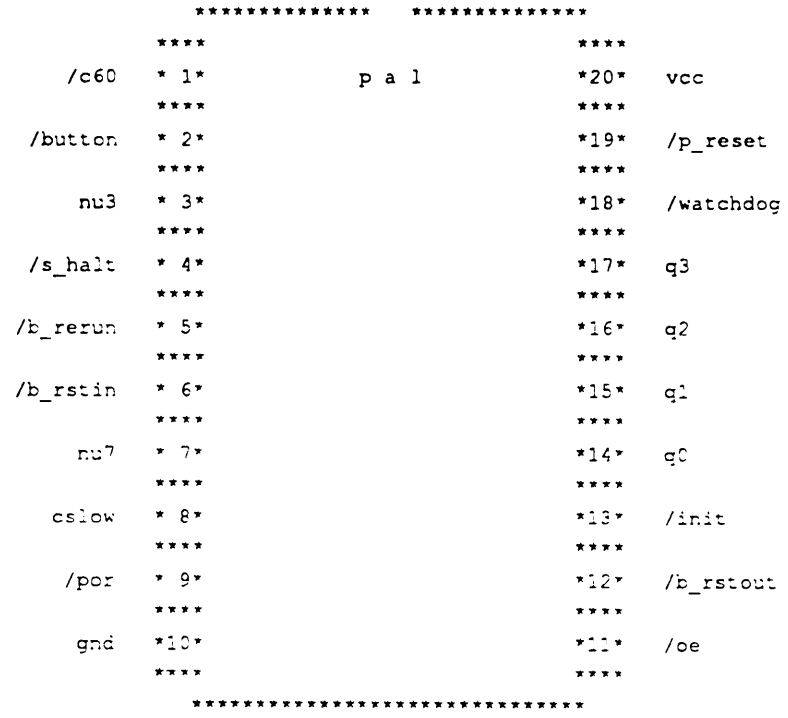


2. U201 Reset PAL

The U201 reset PAL resolves contention between different types of resets.

Pinout of the U201 PAL is:

Figure 8-2 *U201 Pinout*



U201 Input Signals

Inputs to the U201 PAL are listed below.

The following signals are used for watchdog resets (drops you into the monitor PROM):

```
p_halt-   =  Processor halt - either rerun or
             double bus error

b_rerun-  =  Doing cycle rerun - else p_halt- triggers
             watchdog signal

button-   =  Indicates user reset switch has been pushed,
             which is the same as a watchdog reset
```

A very slow clock signal, cslow, is used to generate

- a VME reset pulse of greater than 200 msec, and
- a p_reset signal that lasts longer than 512 cycles.

Since this signal is sampled a lot, the signal is synchronous.

```
cslow = output of refresh address counter, U2404.  It
        has a period approximately 62 msec.
```

The following input signals are "externally" caused resets. The p_reset term triggers a long VME reset.

```
por-       = Power on reset

b_rstin-   = Reset coming from the VMEbus

init_in-   = OR of (por- and b_rstin) feedback

p_resetin- = p_reset - bidirectional - input indicates
               a software reset
```

The p_resetin- and p_resetout- signals are connected to the same bidirectional pin. They are mapped together in a post-processing stage.

U201 Output Signals

Outputs from the U201 PAL are:

```
init-     = board-level initialize signal

watchdog- = watchdog reset occurred (drops you into the monitor)
            used as a status bit in the bus error register

b_rstout- = VMEbus reset

p_reset-  = processor reset
```

These signals are derived from a complex series of state equations which you can find in the PAL listings for U201.

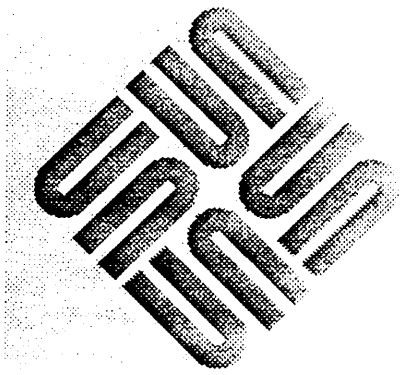
The watchdog- signal is connected to U1403 PAL, where it is decoded as TTL data bit 0 for software readback. To allow this, the least significant bit of data from register U203, t_d[0], is not connected to the data bus; rather t_d[0] is supplied at pin 23 of U1403 PAL.

9

U202 and U203 Bus Error PAL and Register

U202 and U203 Bus Error PAL and Register **51**

9.1. U202 Inputs 51
9.2. Pinout of U202 PAL 52
9.3. U202 Output Signals 53
9.4. U203 Bus Error Register 56



9

U202 and U203 Bus Error PAL and Register

The bus error PAL handles all the bus errors, the most common of which come from the MMU.

9.1. U202 Inputs

Inputs to U202 PAL are:

```
mmu_verr   =   page referenced through MMU is invalid

mmu_perr   =   page is valid, but protected (you are
               trying to reference a protected page)

tout       =   timeout; nothing responded to this cycle

b_berr     =   bus error from the VMEbus

fpp_berr-  =   quick berr when access to the FPP is
               attempted, but FPP has not been enabled

cint_berr- =    spurious interrupt bus error.  If an
               interrupt is asserted and then deasserted
               before an interrupt acknowledge
               can be run, this is known as a
               ''spurious interrupt.''  The system traps
               on a spurious interrupt, which can occur
               on any level except Level 3, the Ethernet
               chip; Intel Ethernet chips can arbitrarily assert
               and deassert their interrupts†.
```

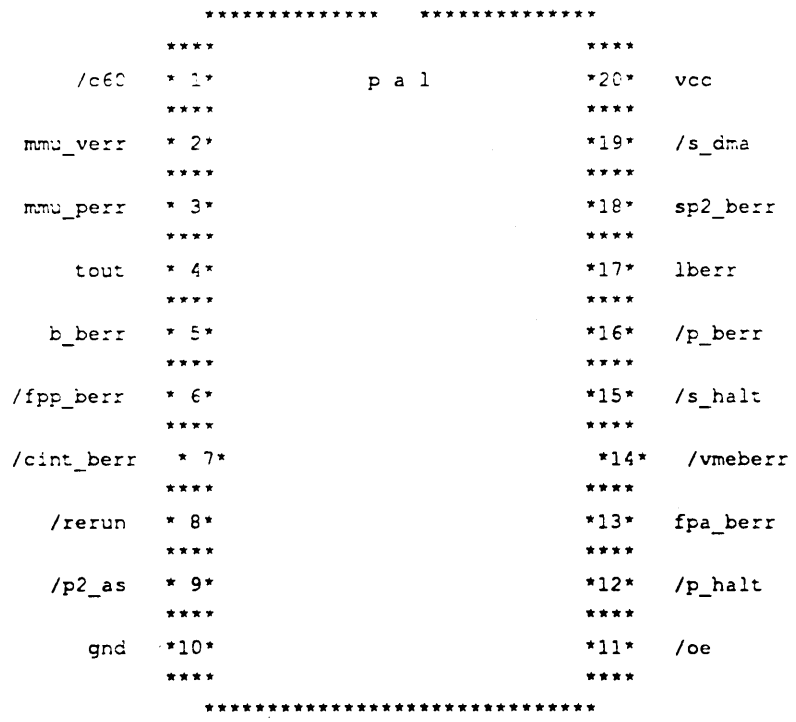
†This is known as a "feature."

```
rerun-      =    rerun the bus cycle; usually necessary
                 because of an arbitration deadlock between
                 the CPU and VME.  The processor is notified
                 that it should rerun the cycle
                 by a simultaneous assertion
                 of p_halt- and p_berr- signals.

p2_as-      =    processor address strobe (acts as a qualifier)

s_dma-      =    indicates the cycle is a DMA cycle

p2_berr     =    bus error on P2 bus from FPA

FPA_berr    =    bus error as result of attemped access to
                 disabled FPA
```

9.2. Pinout of U202 PAL

Pinout of the U202 PAL is:

Figure 9-1 *U202 Pinout*



9.3. U202 Output Signals

Output signals from the U202 PAL are:

```
p_halt-     =    bidirectional processor halt signal

lberr       =    latch bus error status

p_berr-     =    processor bus error

s_halt-     =    syncronized halt signal (internal use only)

vmeberr-    =    bus error for the VME (includes only memory
                 MMU)
```

Halt is asserted for reruns. The signal s_halt is a synchronized version of p_halt; p_halt is the "open collector" version. Since p_halt is bidirectional, R209 pullup resistor is connected to this pin to place the line in a tri-state when the signal is deasserted.

The PAL equation for the p_halt- signal is:

```
if( s_halt ) p_halt = s_halt
s_halt  :=   p2_as * /s_dma * rerun
```

The p_berr signal indicates a processor bus error. It is asserted when one of the following occurs (all are qualified by processor address strobe to indicate a valid read/write cycle):

```
p_berr :=

    p2_as * /s_dma * lberr +
    indicates one of the latched bus errors is active (see lberr below)

    p2_as * /s_dma * fpp_berr +
    access to the 68881 attempted when it has not been enabled

    p2_as * /s_dma * int_berr +   spurious interrupt

    p2_as * /s_dma * rerun +      cycle rerun

    p2_as *  s_dma * mmu_verr +   DVMA attempted to access an invalid page

    p2_as *  s_dma * mmu_perr +   DVMA attempted to access a protected page

    p2_as *  s_dma * tout         DVMA timed out
```

The lberr signal latches certain of the bus error signals into the bus error latch, U203. Since there were not enough OR terms for p_berr, bus errors had to be partitioned into 2 groups — those that have to be latched into the bus error register and those that don't. This is the term used to latch those errors. Errors that are not latched into the bus error register are:

- FPP errors,
- spurious interrupts, and
- reruns.

They vector automatically, and thus do not have to be latched (except for rerun, which is not vectored).

Equation for the bus error register latch signal, lberr-, is:

```
lberr  :=      /p2_as +   valid read/write cycle
                s_dma +          a DMA cycle is not being run
               /mmu_verr *       cycle is invalid
               /mmu_perr *       permissions incorrect
               /tout *           timeout
               /b_berr *         VMEbus error (presyncd)
               /p2_berr *        error from the FPA
               /fpa_berr         FPA is disabled
```

Bus errors that go to the 68020 are latched, and bus errors that are caused by VME cycles excluded.

```
vmeberr :=
          p2_as *  s_dma * mmu_verr +    cycle is invalid
          p2_as *  s_dma * mmu_perr +    permissions incorrect
          p2_as *  s_dma * tout +        timeout
          p2_as *  vmeberr               hold till end of cycle
```

4. U203 Bus Error Register

Some of the bus error signals are latched into U203 to allow software to access them and determine what kind of bus error was asserted. Note that the LSB of the register, t_d[0], is not connected. This bit is the watchdog reset bit, which is supplied later by PAL U1403.

FPP and interrupt bus errors vector automatically to their individual traps, so they do not need to be read by software through the bus error register.

The output enable for the bus error register is supplied by the assertion of rd_berr-, a low active signal supplied by U1403 PAL. Data is loaded and latched into the register by the upward transition of lberr, supplied by U202 bus error PAL.

NOTE *The bus error register latch signal, lberr, is activated by each bus error; thus only the status of the* latest *bus error is latched into the bus error register.*

10

U204 DSACK PAL

U204 DSACK PAL **59**

10.1. Pinout of U204 PAL 59
10.2. U204 Input Signals 60
Bus Transfer Size 60
Offset Bits 60
10.3. U204 Outputs 62

10

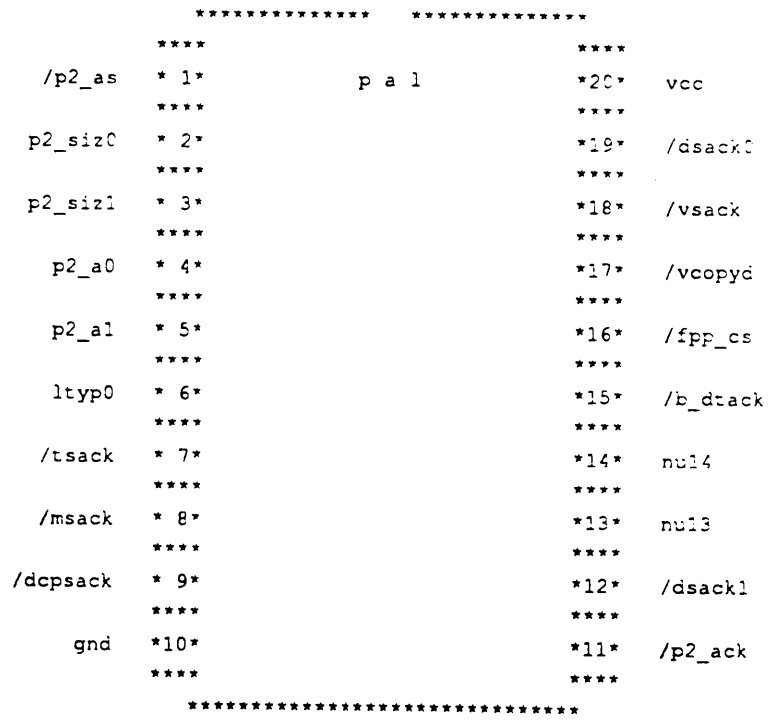
U204 DSACK PAL

The dsack PAL acts mainly as an OR gate, encoding acknowledges arriving from all over the board through the PAL to drive the two dsack output lines. The dsack output lines are used to inform the 68020 of the I/O data port size.

10.1. Pinout of U204 PAL

Pinout of the U204 PAL is:

Figure 10-1 *U204 Pinout*



10.2. U204 Input Signals

```
p2_as-        =  processor is doing a cycle

p2_siz[1:0]   =  size bits (for VME dynamic bus sizing)

p2_a[01:00]   =  address bits (for VME dynamic bus sizing)

ltyp0         =  16-bit or 32-bit VME transfer

tsack-        =  dtack from IO type space - TTL devices

msack-        =  dtack from IO type space - MOS devices

dcpsack-      =  dtack from IO type space - DCP chip

p2_ack-       =  open collector dtack from main memory

vsack-        =  dtack from video frame buffer

vcopydet-     =  copy mode cycle - wait for vsack

fpp_cs-       =  FPP cycle - float outputs

b_dtack-      =  dtack from VME type space
```

Bus Transfer Size

The P2 size bits, p2_siz[1:0], determine the size of the data transfer which is to be made by the processor over the data bus. This "data size" is decoded in the following table.

Table 10-1 *Data Size Encodings: (p2_siz[1:0])*

P2 Size Bits		Size of Transfer
p2_siz[1]	*p2_siz[0]*	
0	0	longword (32 bits)
0	1	byte (8 bits)
1	0	word (16 bits)
1	1	3-byte (24 bits)

Offset Bits

The two low order P2 address bits, p2_a[01:00], are used to determine the offset from the base of the transfer. The "base" of a transfer is the bit at which the least significant bit of the data being transferred lines up in the 32-bit bus transfer space.

- A byte transfer is normally (assuming a zero-byte offset) transferred on lines 24-31 on the data bus, with its LSB aligning on bit 24.

- A word transfer is normally passed over lines 16-31 of the data bus (again assuming a zero-byte offset), with its LSB aligning with bit 16.
- A three-byte transfer occupies data bus lines 8-31, LSB starting at bit 8.
- A longword transfer occupies all 32 data lines.

Figure 10-2 *Byte Data Alignment Within the 32-bit Bus Space†*

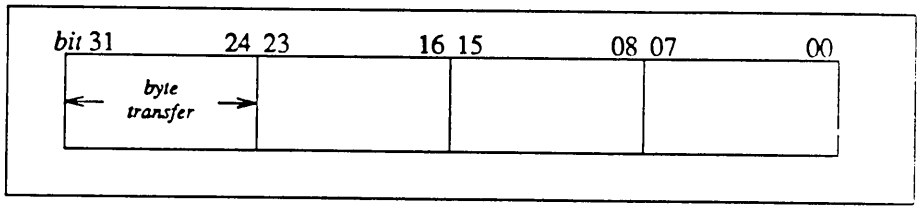


Figure 10-3 *Word Data Alignment Within the 32-bit Bus Space†*

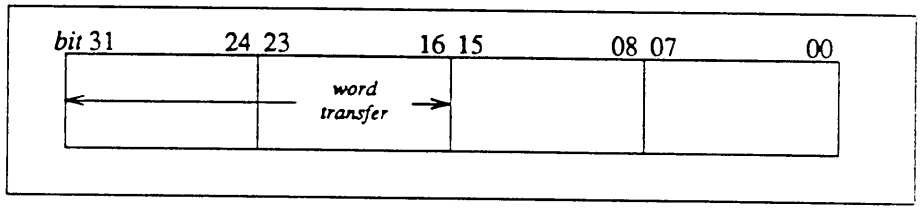


Figure 10-4 *3-Byte Data Alignment Within the 32-bit Bus Space†*

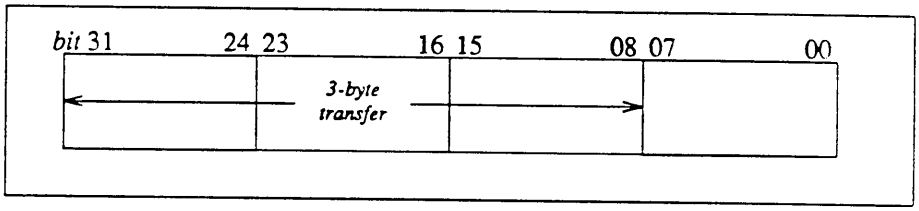
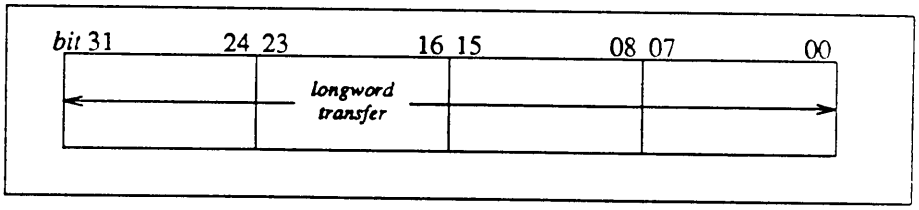


Figure 10-5 *Longword Data Alignment Within the 32-bit Bus Space†*



Remember that in a write cycle, the 68020 drives all 32 lines of the data bus *regardless of the actual size of the transfer*. This means that all 32 lines are driven even though there may only be a byte (8 bits) of data actually being transferred.

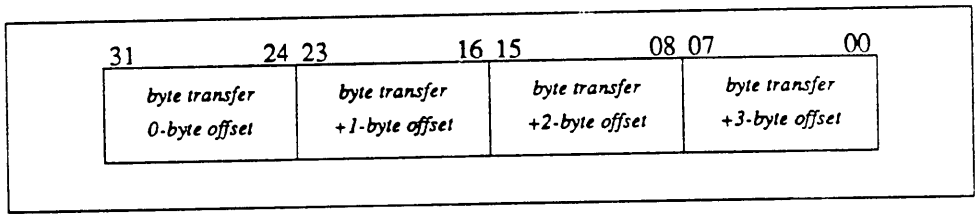
†These alignments are valid for data with a zero-byte offset only.

However, the data can be *offset* from its base — that is, moved one or more bytes to the right in the 32-bit data space.

- For instance, as described above, a byte transfer with no offset is transferred on lines 24 through 31.
- A byte transfer with 1-byte offset will be shifted one byte to the right — that is, transferred over lines 16 through 23.
- A byte transfer with a 2-byte offset will be shifted 2 bytes to the right of the norm — that is, transferred on lines 8 through 15.

The figure below illustrates the different offsets that a byte-transfer may have within a 32-bit longword.

Figure 10-6 *Dynamic Bus Sizing — Transfer Offsets*



And so on. The table below decodes the different offsets.

Table 10-2 *Base Offset Encodings: (p2_a[01:00])*

P2 Address Bits		Size of Offset
p2_a[01]	*p2_a[00]*	
0	0	0 bytes
0	1	+1 bytes
1	0	+2 bytes
1	1	+3 bytes

For more information on transfers and offsets, please the MC68020 User's manual.

10.3. U204 Outputs

Outputs of U204 DSACK PAL are the two dsack bits, dsack[1:0]. They are used as inputs by the processor to determine the port size to or from which it will make an I/O transfer. Port size can be 32, 16 or 8 bits.

Table 10-3 *Dynamic Bus Sizing: How dsack[1:0] Decode Port Size*

DSACK Bits		Size of Port
dsack[01]	*dsack[00]*	
0	0	32 bits
0	1	16 bits
1	0	8 bits
1	1	No size—inserts wait states into current cycle.

Since every device has a unique address, there should never be an occasion when more than one acknowledge is returned at the same time. However should more than one acknowledge be returned simultaneously, the DSACK PAL will output a "no acknowledge" (dsack[1:0] are equal to ones), and wait states are inserted into the current cycle until a timeout bus error is incurred.

11

Interrupt Circuitry — U301:U300, J300

Interrupt Circuitry — U301:U300, J300 67

11.1. Interrupt Priority 67
11.2. U301:0 Interrupt Enable Registers 68
11.3 J300 69

11

Interrupt Circuitry — U301:U300, J300

The interrupt circuitry includes

- two interrupt buffer/registers, U300 and U301,
- the VME interrupt connector, J300,
- the interrupt priority encoder PALs, U304:2, and
- the interrupt priority latch, U305.

This chapter covers the the two interrupt registers and the VME interrupt connector.

11.1. Interrupt Priority

Interrupt priority runs from level 0 (lowest priority) to level 7 (highest priority). Interrupt levels are encoded to the three interrupt priority code bits, p_ipl(2:0), which are connected to the interrupt priority level pins ipl(2:0) on the 68020.

- Interrupt level zero (low active ipl[2:0] bits are all high) indicates that no interrupt service is presently requested.
- Interrupt levels one through six are "maskable" interrupts — they are validated after comparison with three interrupt status bits in the MC68020 status register.
    1. If the value in the status register is greater (its interrupt priority is higher) than this latest interrupt request, the latest interrupt request is ignored.
    2. If the value in the status register is less than or equal to (its interrupt priority is equal to or lower than) this latest interrupt, the latest interrupt is "pending." This means it will be serviced as soon as the present interrupt cycle is completed.
- Interrupt level seven (low active ipl[2:0] bits are all low) is non-maskable. This means that it can not be inhibited by the interrupt priority mask (three interrupt status bits in the MC68020 status register). An interrupt request is generated any time a request level changes from some lower level to level seven.

.2. U301:0 Interrupt Enable Registers

The interrupt registers latch interrupt enable status bidirectionally. Write status is held in U300, software readback status is done through U301. The register is cleared on power-up by the assertion of init- from the U201 Reset PAL.

The interrupt register allows you to selectively generate one of three software interrupts — levels 1, 2, or 3. Interrupt level 0 is used as a global interrupt enable (high active en_int signal). Signal descriptions from the registers are given below.

- EN.INT — This bit enables all interrupts, including those recorded in the Memory Error register. If this bit is off, no interrupts can occur.
- EN.INT(3:1) — These bits cause software interrupts on the corresponding level. The interrupt request caused by an EN.INT(3:1) bit stays active until software clears the corresponding bit.
- EN.INT4 — enables video interrupt requests on level 4. When enabled, a level 4 interrupt request is set on the rising edge of vertical retrace. A level 4 interrupt request is cleared by momentarily turning off the EN.INT4 bit.

NOTE *This bit, EN.INT4, has no effect in implementations which have no memory frame buffers.*

- EN.INT5 — enables clock interrupt requests on level 5. When enabled, a level 5 interrupt request is set on the rising edge of the clock interrupt output. The level 5 interrupt request is cleared by momentarily turning off the EN.INT5 bit.
- EN.INT6 — is a reserved bit. It can be read from and written to but has no effect.
- EN.INT7 — enables clock interrupt requests on level 7. When enabled, a level 7 interrupt request is set on the rising edge of the clock interrupt output. The level 7 interrupt request is cleared by momentarily turning off the EN.INT7 bit.

Table 11-1 *Interrupt Register — Signal Designations*

BIT	NAME	TYPE	MEANING
D<0>	EN.INT	read-write	Enable all Interrupts
D<1>	EN.INT1	read-write	Software Interrupt Level 1
D<2>	EN.INT2	read-write	Software Interrupt Level 2
D<3>	EN.INT3	read-write	Software Interrupt Level 3
D<4>	EN.INT4	read-write	Enable Video Interrupt Level 4
D<5>	EN.INT5	read-write	Enable Clock Interrupt Level 5
D<6>	EN.INT6	read-write	(reserved)
D<7>	EN.INT7	read-write	Enable Clock Interrupt Level 7

The write interrupt register, U300, has data written into it from the TTL data bus when a positive transition of wr_int- arrives from U1401 PAL (we use the positive edge of a negative active signal as clock here to allow for latch setup time). The register is cleared to lows by the assertion of the processor reset signal, p_reset-.

The readback register is an ALS244 buffer whose output is enabled with the assertion of rd_int- (a low active signal) to pins 1 and 19.

The register latches seven levels of interrupt enable and an eighth signal, vinten, vertical interrupt enable, which clears the vertical interrupt flipflop, U2209, in the video section of the board.

11.3. J300

J300 jumper allows you to individually enable or disable interrupts coming from the VMEbus. This is a convenience in situations where you are using more than one CPU, by allowing you to assign individual interrupts to separate CPUs.

12

Interrupt Circuitry — U302–U304 PALs, U305 Register

Interrupt Circuitry — U302–U304 PALs, U305 Register 73

12.1. Interrupt Request Cycle 73
12.2. Interrupt Acknowledge Cycle 74
12.3. Priority 77
12.4. Two-Level Priority Encoding 77
12.5. U302 Lower-Priority Encoder 78
U302 Pinout 79
U302 Input Signals 79
U302 Output Signals 80
12.6. U303 Higher-Priority Encoder 84
U303 Pinout 84
U303 Input Signals 84
U303 Output Signals 85
12.7. Second-Level Interrupt Priority Encoder — U304 89
Pinout of U304 PAL 89
U304 Input Signals 90
U304 Output Signals 90
Sample Interrupt Cycle 92
Spurious Interrupt 92
Ethernet Controller and Spurious Interrupts 93
12.8. U305 Latch 94

12

Interrupt Circuitry — U302–U304 PALs, U305 Register

This Chapter covers the three interrupt PALs, U304:2, on page three of the schematics. It also describes the interrupt priority latch, U305.

Interrupts can come from the

- VMEbus (through J300 header)
- on-board devices
- interrupt enable register (U300), or be
- spurious (false).

NOTE *A spurious interrupt is an interrupt which is asserted and then deasserted before the processor can complete an interrupt acknowledge cycle. Typically, an interrupt level is asserted but no dsack(1:0) bits or autovector signal (p_avec-) is returned to conclude the interrupt acknowledge cycle.*

There are two parts to an interrupt cycle:

1. a request cycle, and
2. an interrupt cycle.

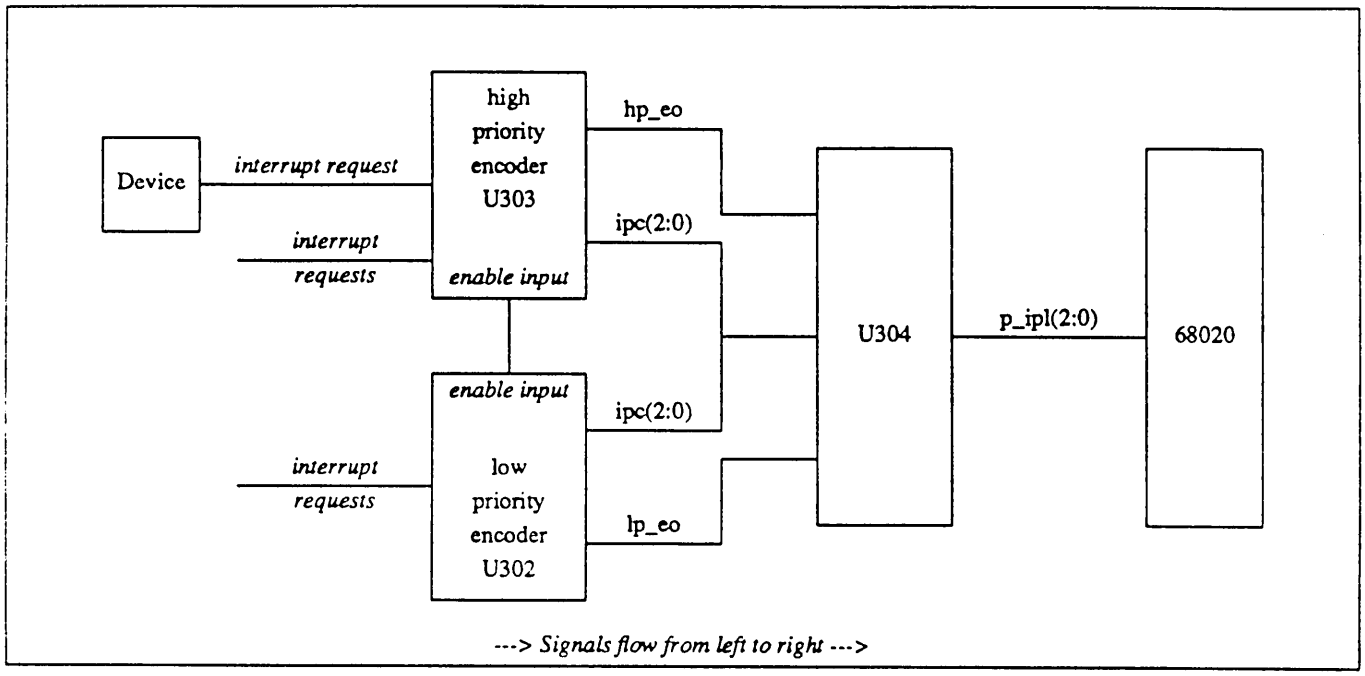
12.1. Interrupt Request Cycle

On interrupt request cycles, a device or interrupt request is asserted to the either the high or low priority encoder (U302 or U303). The interrupt request level is then output from the appropriate PAL on the three interrupt priority code lines, ipc(2:0), to PAL U304.

This interrupt priority code level is then encoded by U304 onto the three interrupt priority level lines p_ipl(2:0) which are connected to the processor.

This concludes the interrupt request cycle.

Figure 12-1 *Interrupt Request Cycle*



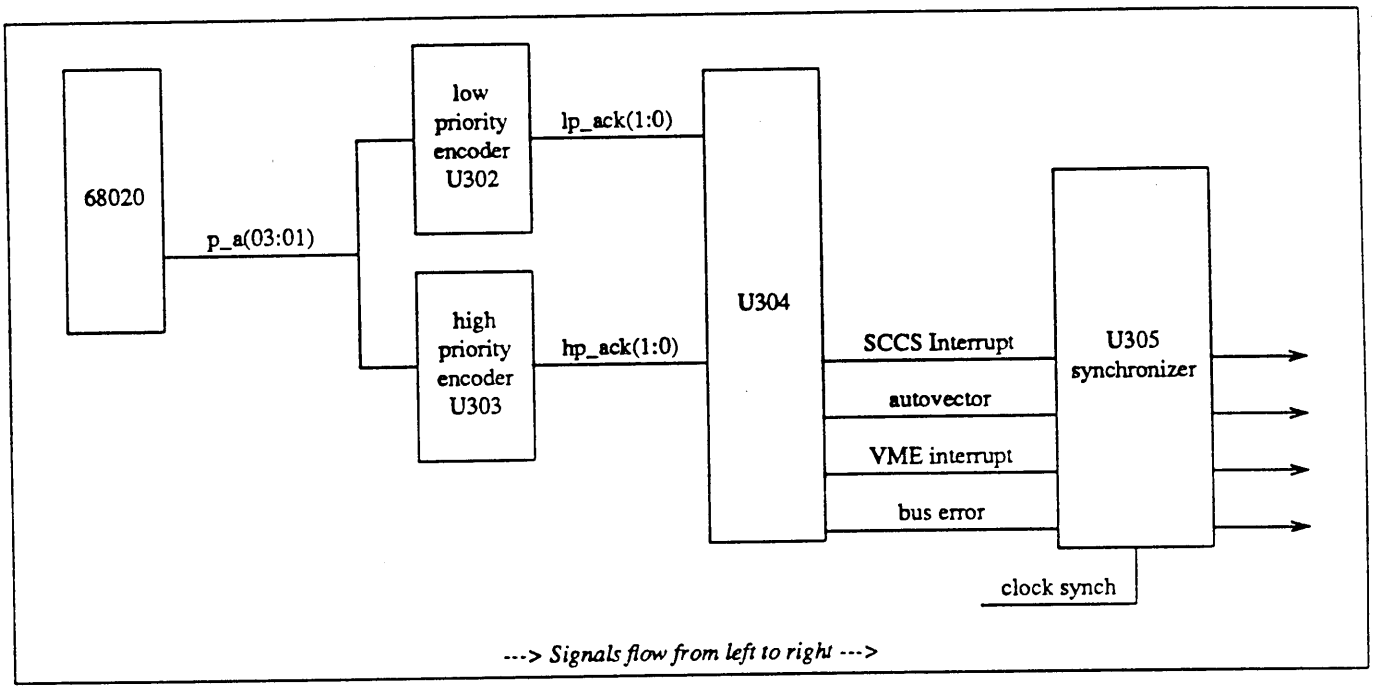
The processor completes the second half of an interrupt cycle, the interrupt acknowledge cycle, by issuing the acknowledged priority level on three processor address lines, p_a(03:01), which are connected to the upper and lower priority encoders, U302 and U303. The appropriate encoder issues a 2-bit acknowledge signal to U304 on either

- hp_ack(1:0) — high priority encoder, or
- lp_ack(1:0) — low priority encoder.

U304 then decodes this 2-bit acknowledge and issues the appropriate interrupt acknowledge signal:

- interrupt bus error
- VMEbus interrupt
- SCC interrupt acknowledge, or
- autovector.

Figure 12-2 *Interrupt Acknowledge Cycle*



Before issuing any sort of interrupt signals of their own, U303 and U302 make certain that there are no interrupts pending of a higher priority than that they are about to service.

- If a higher-priority interrupt is pending, the interrupt acknowledge cycle is allowed to complete before this new one is initiated.
- If no higher-level interrupt is pending (or the interrupt pending is of an equal or lower priority than the earlier interrupt about to be serviced) the earlier interrupt cycle is completed.

Thus, the progression through an interrupt cycle is:

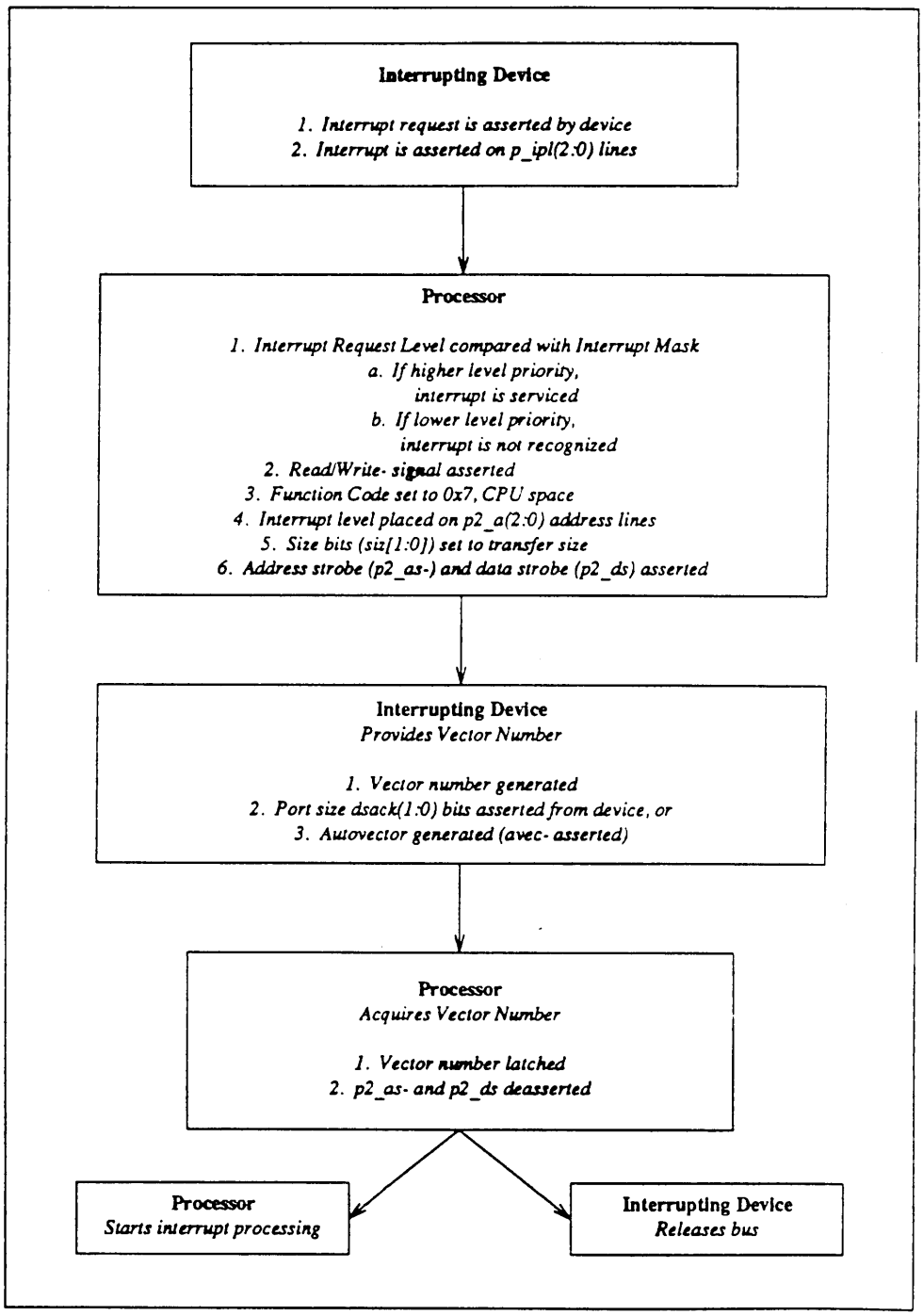
```
Request --> Pending --> Acknowledge --> Service
```

These acknowledge signals encode various types of interrupts:

- autovector — the p_avec- signal is asserted by most on-board devices, forcing the processor to generate an internal vector number;
- vectored on-board interrupt — vectors to the SCCs;
- vectored interrupts from the VMEbus;
- finally, there may be no response, which indicates that no interrupts are pending (spurious interrupt).

A flow chart of the interrupt acknowledge cycle is provided below.

Figure 12-3 *Interrupt and Acknowledge Cycle*



12.3. Priority

Priority goes to the on-board interrupts on both encoding and acknowledging.

12.4. Two-Level Priority Encoding

The two-level priority encoder uses three PALs — U304:U302 — and a latch, U305. The first level of encoding is done by the two PALs, U302 and U303. Second level encoding is done by U304.

It is necessary to use two levels of encoding because there are too many inputs to be handled by a single PAL. Therefore the pair of PALs, U302 and U303, execute a first-level encoding; their outputs are encoded by the second-level PAL, U304, whose outputs are the final interrupt and acknowledge signals.

The first level of encoding is split between

- U302 — which handles the lower priority interrupts and
- U303 — which handles the higher priority interrupts.

When one of the two encoders processes an interrupt, it issues an interrupt priority code to the second-level encoder, U304, over the ipc(2:0) lines.

The two first-level encoders, U302 and U303, are daisy chained together by the output enable signal /hp_eo, which arbitrates between U302 and U303. If U303 deasserts the low active /hp_eo signal, it:

1. notifies U304 that U303 will have control of the ipc(2:0) lines, and
2. disables U302, the lower-level priority encoder. This "lock-out" ensures that the highest priority interrupt request takes precedence, preventing bus contention.

Otherwise, U302 asserts /lp_eo, indicating that it (U302) will have output on the ipc(2:0) lines.

The 2-bit lp_ack(1:0) and hp_ack(1:0) acknowledge bits encode the three types of interrupt acknowledge:

Table 12-1 *Low Priority Acknowledge Bit Encodings: lp_ack(1:0)*

Priority Address Bits		Type of Vector
lp_ack[01]	*lp_ack[00]*	
1	1	Autovector
1	0	Not used
0	1	VME vector
0	0	Spurious Interrupt!†

†If lp_ack(1:0) = 00 and the p2_a(3:1) address bits equal 0x7, 0x6, or 0x5, this code does not indicate a spurious interrupt. It merely indicates an interrupt on that respective level — 5, 6, or 7, which is of no interest to this encoder.

Table 12-2 *High Priority Acknowledge Bit Encodings: hp_ack(1:0)*

Priority Address Bits		Type of Vector
hp_ack[01]	*hp_ack[00]*	
1	1	Autovector
1	0	SCC interrupt
0	1	VME vector
0	0	Spurious interrupt!‡

Concatenation of these two tables, lp_ack and hp_ack, result in the following derivations:

Table 12-3 *Type of Interrupts Encoded by hp_ack(1:0) and lp_ack(1:0)*

Priority Address Bits				Type of Vector
hp_ack[01]	*hp_ack[00]*	*lp_ack[01]*	*lp_ack[00]*	
0	0	0	0	spurious interrupt
0	0	0	1	vmevec - low priority section
0	0	1	0	sccvec - low priority section - ILLEGAL
0	0	1	1	autovec - low priority section
0	1	X ††	X	vmevec - high priority section
1	0	X	X	sccvec - high priority section
1	1	X	X	autovec - high priority section

12.5. U302 Lower-Priority Encoder

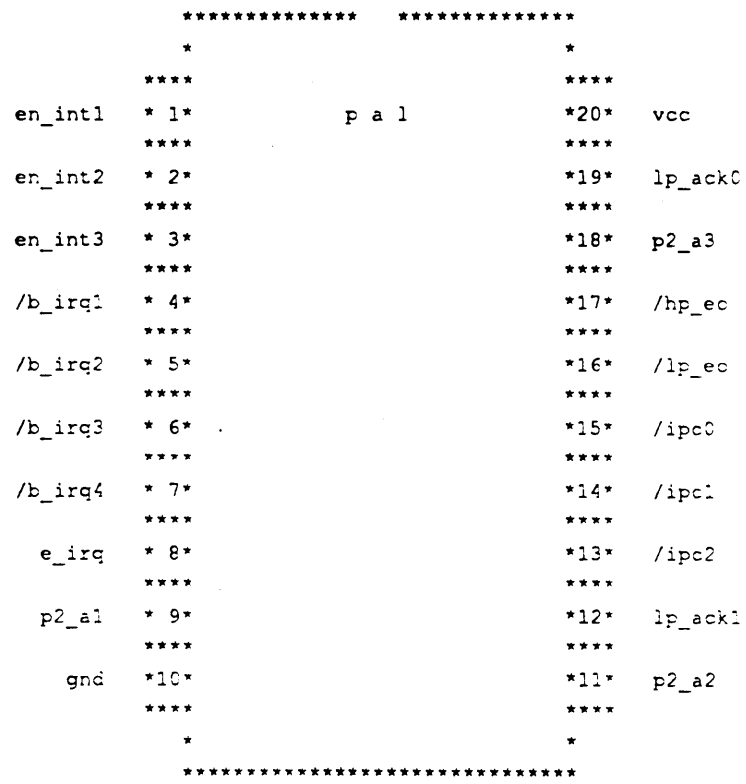
U302 encodes interrupt levels 4, 3, 2, 1 and 0; the lower-level interrupts. It also encodes the lower-priority acknowledges.

‡If hp_ack(1:0) = 00 and the p2_a(3:1) address bits equal 0x3, 0x2, 0x1, or 0x0, this code does not indicate a spurious interrupt. It merely indicates an interrupt on that respective level — 3, 2, 1, or 0, which is of no interest to this encoder.
††"X" means "don't care."

U302 Pinout

Pinout for the U302 PAL is:

Figure 12-4 *U302 Pinout*



U302 Input Signals

Inputs to the U302 PAL are:

```
en_int[3:1] = software interrupt enables from TTL data bus

b_irq[4:1]  = interrupt requests from the VMEbus

e_irq       = Ethernet chip interrupt request

p2_a[03:01] = P2 address bits which are used by the processor
              to indicate which level of interrupt is
              being acknowledged

hp_eo       = high-level priority encoder enable output, which
              is used to disable U302 to avoid bus contention
              with U303.
```

'302 Output Signals

Two types of signals are decoded through the lower-level priority encoder:

- acknowledge encode
- priority encode.

Acknowledge encode signals, /lp_ack1 and /lp_ack0, are the result of the following equations:

```
/lp_ack1  = p2_a3 +
            /e_irq * /en_int3 * b_irq3 * p2_a2 * p2_a1 +
            /en_int2 * /p2_a1 +
            /en_int1 * /p2_a2 +
            /p2_a2 * /p2_a1

/lp_ack0  = p2_a3 * p2_a2 +
            p2_a3 * p2_a1 +
            /b_irq4 * /p2_a2 * /p2_a1 +
            /en_int2 * /b_irq2 * p2_a2 * /p2_a1 +
            /en_int1 * /b_irq1 * /p2_a2 * p2_a1 +
            /p2_a3 * /p2_a2 * /p2_a1
```

The table below contains the PAL logic from which each output signal is derived.

Table 12-4 *Lower Priority Acknowledge Signals*

Inputs											Outputs		Comments
b_irq4-	e_irq	en_int3	b_irq3-	en_int2	b_irq2-	en_int1	b_irq1-	p2_a3	p2_a2	p2_a1	lp_ack1	lp_ack0	
X†	X	X	X	X	X	X	X	1	1	1	0	0	level 7
X	X	X	X	X	X	X	X	1	1	0	0	0	level 6
X	X	X	X	X	X	X	X	1	0	1	0	0	level 5
0	X	X	X	X	X	X	X	1	0	0	0	1	vectored VME interrupt, level 4
1	X	X	X	X	X	X	X	1	0	0	0	0	spurious interrupt, level 4
X	1	X	X	X	X	X	X	0	1	1	1	1	level 3 autovector (Ethernet)
X	0	1	X	X	X	X	X	0	1	1	1	1	level 3 autovector (system enable interrupt)
X	0	0	0	X	X	X	X	0	1	1	0	1	vectored VME interrupt, level 3
X	0	0	1	X	X	X	X	0	1	1	1	1	special case—spurious Ethernet interrupts
X	X	X	X	1	X	X	X	0	1	0	1	1	level 2 autovector to system enable interrupt
X	X	X	X	0	0	X	X	0	1	0	0	1	level 2 VME vectored interrupt
X	X	X	X	0	1	X	X	0	1	0	0	0	spurious level 2 interrupt
X	X	X	X	X	X	1	X	0	0	1	1	1	level 1 autovector (system enable interrupt)
X	X	X	X	X	X	0	0	0	0	1	0	1	level 1 VME vectored interrupt
X	X	X	X	X	X	0	1	0	0	1	0	0	spurious level 1 interrupt
X	X	X	X	X	X	X	X	0	0	0	0	0	level 0 interrupt

†"X" means "don't care."

The priority encode signals, ipc[2:0] and lp_eo, are the result of the following PAL equations:

```
ipc2  = hp_eo * b_irq4 +
        hp_eo * e_irq +
        hp_eo * en_int3 +
        hp_eo * b_irq3

ipc1  = hp_eo * b_irq4 +
        hp_eo * e_irq +
        hp_eo * /en_int3 * /b_irq3 * en_int2 +
        hp_eo * /en_int3 * /b_irq3 * b_irq2

ipc0  = hp_eo * b_irq4 +
        hp_eo * /e_irq * en_int3 +
        hp_eo * /e_irq * /b_irq3 * en_int2 +
        hp_eo * /e_irq * /b_irq3 * /b_irq2 * en_int1

lp_eo  = /hp_eo + cannot be asserted when hp_eo is deasserted
         /b_irq4 * /e_irq * /en_int3 * /b_irq3 *
            /en_int2 * /b_irq2 * /en_int1 * /b_irq1
```

The following table contains the PAL logic from which each output signal is derived.

Table 12-5 *Lower Interrupt Priority Encode Signals*

Inputs									Outputs				Comments
hp_eo-	b_irq4-	e_irq	en_int3	b_irq3-	en_int2	b_irq2-	en_int1	b_irq1-	ipc2-	ipc1-	ipc0-	lp_eo-	
1	X†	X	X	X	X	X	X	X	1	1	1	0	interrupts disabled
0	1	0	0	1	0	1	0	1	1	1	1	0	no interrupts
0	0	X	X	X	X	X	X	X	0	0	0	1	VME vector - level 4 - VME level[4]
0	1	1	X	X	X	X	X	X	0	0	1	1	Autovector - level 3 - ethernet
0	1	0	1	X	X	X	X	X	0	1	0	1	Autovector - level 3 - system enable int
0	1	0	0	0	X	X	X	X	0	1	1	1	VME vector - level 3 - VME level[3]
0	1	0	0	1	1	X	X	X	1	0	0	1	Autovector - level 2 - system enable int
0	1	0	0	1	0	0	X	X	1	0	1	1	VME vector - level 2 - VME level[2]
0	1	0	0	1	0	1	1	X	1	1	0	1	Autovector - level 1 system enable int
0	1	0	0	1	0	1	0	0	1	1	1	1	VME vector - level 1 - VME level[1]

†"X" means "don't care."

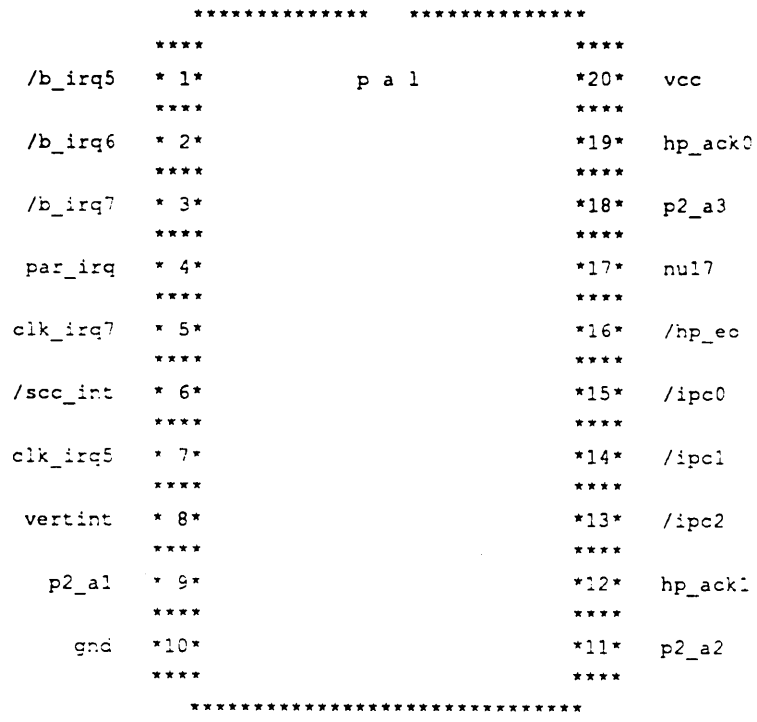
12.6. U303 Higher-Priority Encoder

This section contains the signal description for U303 higher-priority encoder.

U303 Pinout

Pinout for the U303 PAL is:

Figure 12-5 *U303 Pinout*



U303 Input Signals

Inputs to the U303 PAL are:

```
b_irq[7:5] = interrupt request from the VMEbus

par_irq    = parity circuitry interrupt request

clk_irq7   = time-of-day clock interrupt on level 7

scc_int    = serial controller interrupt

clk_irq5   = time-of-day clock interrupt on level 5

vertint    = interrupt from the vertical state machine
             in the video section

p2_a[02:00] = P2 address bits (for readback of present interrupt
              level being acknowledged)
```

U303 Output Signals

Just as in U302, two types of signals are decoded through the higher-level priority encoder:

- acknowledge encode
- priority encode.

Acknowledge encode signals, /hp_ack1 and /hp_ack0, are the result of the following equations:

```
/hp_ack1  = /par_irq * /clk_irq7 * p2_a2 * p2_a1 +
            /scc_int * p2_a2 * /p2_a1 +
            /clk_irq5 * /p2_a2 * p2_a1 +
            /vertint * /p2_a2 * /p2_a1 +
            /p2_a3
```

```
/hp_ack0  = /par_irq * /clk_irq7 * /b_irq7 * p2_a2 * p2_a1 +
            scc_int * p2_a2 * /p2_a1 +
            /b_irq6 * p2_a2 * /p2_a1 +
            /clk_irq5 * /b_irq5 * /p2_a2 * p2_a1 +
            /vertint * /p2_a2 * /p2_a1 +
            /p2_a3
```

The table below contains the PAL logic from which each output signal is derived.

Table 12-6 *Higher-Level Priority Acknowledge Signals*

Inputs											Outputs		Comments
par_irq	clk_irq7	b_irq7-	scc_int-	b_irq6-	clk_irq5	b_irq5-	vertint	p2_a3	p2_a2	p2_a1	hp_ack1	hp_ack0	
1	X†	X	X	X	X	X	X	1	1	1	1	1	Autovector - level 7 - parity error
0	1	X	X	X	X	X	X	1	1	1	1	1	Autovector - level 7 - system clock
0	0	0	X	X	X	X	X	1	1	1	0	1	VME vector - level 7 - VME level[7]
0	0	1	X	X	X	X	X	1	1	1	0	0	spurious - level 7
X	X	X	0	X	X	X	X	1	1	0	1	0	sccvec - level 6 - serial chips
X	X	X	1	0	X	X	X	1	1	0	0	1	VME vector - level 6 - VME level[6]
X	X	X	1	1	X	X	X	1	1	0	0	0	spurious - level 6
X	X	X	X	X	1	X	X	1	0	1	1	1	Autovector - level 5 - system clock
X	X	X	X	X	0	0	X	1	0	1	0	1	VME vector - level 5 - VME level[5]
X	X	X	X	X	0	1	X	1	0	1	0	0	spurious - level 5
X	X	X	X	X	X	X	1	1	0	0	1	1	Autovector - level 4 - video interrupt
X	X	X	X	X	X	X	0	1	0	0	0	0	spurious - level 4
X	X	X	X	X	X	X	X	0	1	1	0	0	level 3
X	X	X	X	X	X	X	X	0	1	0	0	0	level 2
X	X	X	X	X	X	X	X	0	0	1	0	0	level 1
X	X	X	X	X	X	X	X	0	0	0	0	0	level 0

†"X" means "don't care."

Higher-level interrupt priority encode signals, hp_eo and ipc[2:0], are the result of the following equations:

```
if (/hp_eo ) ipc2  = par_irq +
                     clk_irq7 +
                     b_irq7 +
                     scc_int
```

```
if (/hp_eo ) ipc1  = par_irq +
                     clk_irq7 +
                     /b_irq7 * /scc_int * b_irq6 +
                     /b_irq7 * /scc_int * clk_irq5
```

```
if (/hp_eo ) ipc0  = par_irq +
                     /clk_irq7 * b_irq7 +
                     /clk_irq7 * /scc_int * b_irq6 +
                     /clk_irq7 * /scc_int * /clk_irq5 * b_irq5
```

```
hp_eo  = /par_irq * /clk_irq7 * /b_irq7 * /scc_int *
         /b_irq6 * /clk_irq5 * /b_irq5 * /vertint
```

The table below contains the PAL logic from which each output signal is derived.

Table 12-7 *Higher-Level Interrupt Priority Encode Signals*

Inputs								Outputs				Comments
par_irq	clk_irq7	b_irq7-	scc_int-	b_irq6-	clk_irq5	b_irq5-	vertint	ipc2-	ipc1-	ipc0-	hp_eo-	
0	0	1	1	1	0	1	0	1	1	1	0	no interrupts
1	X	X	X	X	X	X	X	0	0	0	1	Autovector - level 7 - parity error
0	1	X	X	X	X	X	X	0	0	1	1	Autovector - level 7 - system clock
0	0	0	X	X	X	X	X	0	1	0	1	VME vector - level 7 - VME level[7]
0	0	1	0	X	X	X	X	0	1	1	1	sccvec - level 6 - serial chips
0	0	1	1	0	X	X	X	1	0	0	1	VME vector - level 6 - VME level[6]
0	0	1	1	1	1	X	X	1	0	1	1	Autovector - level 5 - system clock
0	0	1	1	1	0	0	X	1	1	0	1	VME vector - level 5 - VME level[5]
0	0	1	1	1	0	1	1	1	1	1	1	Autovector - level 4 - video interrupt

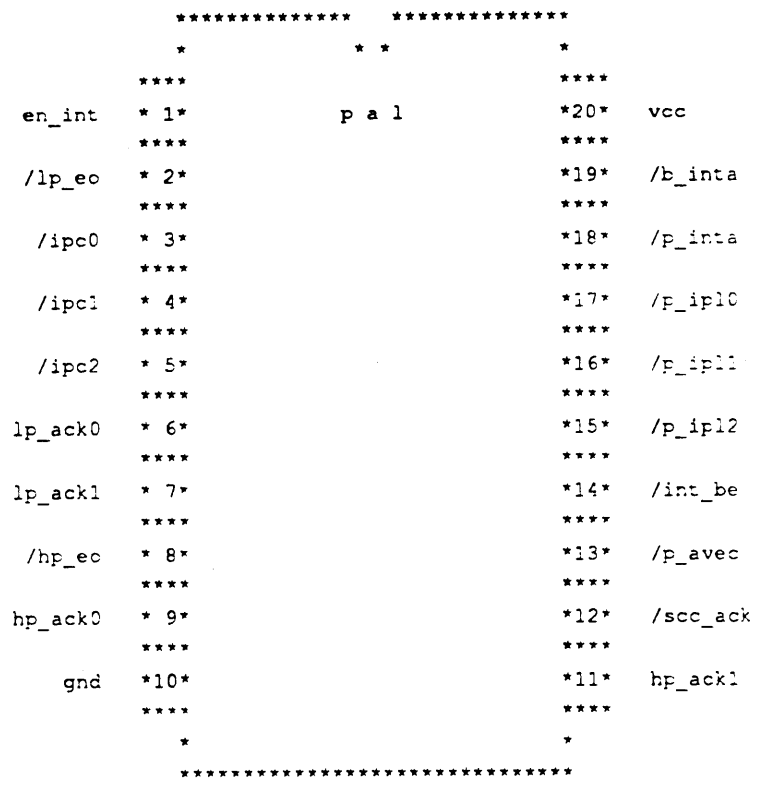
12.7. Second-Level Interrupt Priority Encoder — U304

The second-level priority encoder takes the outputs of the two first-level priority encoder PALs, U302 and U303, and encodes them to issue as interrupt priority and acknowledge signals.

Pinout of U304 PAL

Pinout of the U304 PAL is:

Figure 12-6 *U304 Pinout*



304 Input Signals

```
en_int      = global interrupt enable

/hp_eo      = enable output from U303 (locks out U302)

/lp_eo      = enable output from U302 (cannot be asserted
              when /hp_eo is deasserted)

/ipc(2:0)   = interrupt priority code from either U302
            or U303 (/hp_eo signal arbitrates source)

/p_inta     = processor interrupt acknowledge

hp_ack(1:0)= encode of vector-type being asserted from
              higher-level priority encoder

lp_ack(1:0)= encode of vector-type being asserted from
              lower-level priority encoder
```

U304 Output Signals

Output signals for the U304 PAL are:

```
/p_ipl(2:0) = interrupt priority level sent back to the
              processor to notify what level interrupt
              is being asserted

int_berr-   = bus error interrupt

b_inta-     = VMEbus interrupt acknowledge

scc_ack-    = SCC interrupt acknowledge

p_avec-     = tells the processor to generate an autovector
```

The table below contains the PAL logic from which the acknowledge and bus error signals are derived.

Table 12-8 *U304: Second-Level Acknowledge Signals*

Inputs					Outputs				Comments
p_inta-	hp_ack1	hp_ack0	lp_ack1	lp_ack0	int_berr-	b_inta-	scc_ack-	p_avec-	
1	X	X	X	X	1	1	1	1	not doing an interrupt cycle
0	0	0	0	0	0	1	1	1	spurious interrupt
0	0	0	0	1	1	0	1	1	vmevec - low priority section
0	0	0	1	0	0	1	1	1	sccvec - low priority section - ILLEGAL
0	0	0	1	1	1	1	1	0	autovec - low priority section
0	0	1	X	X	1	0	1	1	vmevec - high priority section
0	1	0	X	X	1	1	0	1	sccvec - high priority section
0	1	1	X	X	1	1	1	0	autovec - high priority section

The table below contains the PAL logic from which the interrupt level signals, ipl(2:0), are derived.

Table 12-9 *U304: Second-Level Interrupt Priority Level Signals*

Inputs						Outputs			Comments
en_int	hp_eo-	lp_eo-	ipc2-	ipc1-	ipc0-	p_ipl2-	p_ipl1-	p_ipl0-	
0	X	X	X	X	X	1	1	1	interrupts disabled
1	0	0	X	X	X	1	1	1	no interrupts active
1	1	X	0	0	0	0	0	0	autovec - level 7 - parity error
1	1	X	0	0	1	0	0	0	autovec - level 7 - system clock
1	1	X	0	1	0	0	0	0	vmevec - level 7 - VME level[7]
1	1	X	0	1	1	0	0	1	sccvec - level 6 - serial chips
1	1	X	1	0	0	0	0	1	vmevec - level 6 - VME level[6]
1	1	X	1	0	1	0	1	0	autovec - level 5 - system clock
1	1	X	1	1	0	0	1	0	vmevec - level 5 - VME level[5]
1	1	X	1	1	1	0	1	1	autovec - level 4 - video interrupt
1	0	1	0	0	0	0	1	1	vmevec - level 4 - VME level[4]
1	0	1	0	0	1	1	0	0	autovec - level 3 - ethernet
1	0	1	0	1	0	1	0	0	autovec - level 3 - system enable int
1	0	1	0	1	1	1	0	0	vmevec - level 3 - VME level[3]
1	0	1	1	0	0	1	0	1	autovec - level 2 - system enable int
1	0	1	1	0	1	1	0	1	vmevec - level 2 - VME level[2]
1	0	1	1	1	0	1	1	0	autovec - level 1 - system enable int
1	0	1	1	1	1	1	1	0	vmevec - level 1 - VME level[1]

ample Interrupt Cycle

This subsection runs you through a sample interrupt cycle. Let's say that a video interrupt is being asserted — level 4 autovector.

The first step in the interrupt cycle is for the interrupting device (the video circuitry) to issue an interrupt request. It does this by asserting the vertical retrace interrupt signal, vertint, from U2209 vertical interrupt flip-flop.

- Vertical interrupt is coupled to U303 PAL.
- U303 deasserts hp_eo-, indicating to U304 that it (U303) will take control of the ipc(2:0) bus it shares in common with U302. This also locks U302 off of the ipc(2:0) bus.
- Interrupt level four is encoded onto ipc(2:0) lines from U303 into U304.
- U304 issues the video interrupt level over the ipl(2:0) lines to the 68020.

This concludes the interrupt request half of the cycle.

Next, the processor runs an interrupt acknowledge cycle.

- If no higher-level interrupt request is pending, the 68020 processor issues the video interrupt acknowledge over its address lines, p_a(3:1), which are coupled to PAL U303.
- The p2_a(3:1) address bits are set to the interrupt level, level 4. These are decoded through U303 to select either an autovectored video interrupt, or a spurious interrupt. Differentiation between these two is made by the assertion or deassertion of the vertint signal — vertical retrace interrupt. If vertint is false, then the system initiates an interrupt bus error and vectors to the spurious interrupt trap. If vertint is valid (high), the interrupt is acknowledged on the two higher-level encoder acknowledge lines, hp_ack(1:0).
- The processor interrupt acknowledge signal, p_inta-, is input to U304.
- U304 makes certain that the processor interrupt acknowledge, p_inta-, from U107 is true before issuing the p_avec- (processor autovector signal).

Remember that during an interrupt acknowledge cycle the interrupt level, level 4, is asserted on address lines A03:01. All the rest of the address lines, A31:04, are driven high; the three function code bits, FC2:0, are also driven high. This uniquely identifies the current processor cycle as an interrupt acknowledge cycle. Decoding these address and function code bits, the U107 CPU space PAL issues the low active processor interrupt acknowledge signal, p_inta-.

Spurious Interrupt

A spurious interrupt occurs when an interrupt signal is asserted, an interrupt cycle is run, but the interrupting device has taken away the request. When this happens, U304 deasserts all its outputs except for int_berr-, interrupt bus error, which is coupled to the U202 bus error PAL. One example of a spurious interrupt is to run a VME interrupt acknowledge cycle and then have the device requesting the interrupt not respond to the vector fetch cycle.

There is a special case, though, in which symptoms that normally would indicate a spurious interrupt are not recognized. This is the case of the asynchronous Ethernet chip, described below.

Ethernet Controller and Spurious Interrupts

Look at the PAL logic (the table below) for the Ethernet controller.

Table 12-10 *Spurious Ethernet Interrupts*

Inputs											Outputs		Comments
b_irq4-	e_irq	en_int3	b_irq3-	en_int2	b_irq2-	en_int1	b_irq1-	p2_a3	p2_a2	p2_a1	lp_ack1	lp_ack0	
X	1	X	X	X	X	X	X	0	1	1	1	1	level 3 autovector (Ethernet)
X	0	0	1	X	X	X	X	0	1	1	1	1	special case— spurious Ethernet interrupts

The Ethernet controller interrupts on level 3, p2_a(3:1) = 0x3, while e_irq- (Ethernet controller interrupt request) is asserted. This causes a level 3 autovector code to be transmitted over lp_ack(1:0) outputs as normal. The spurious interrupt capability is also accounted for on this level. However notice what happens when the symptoms of a spurious interrupt occur (taken from the table above):

- e_irq is low (false)
- en_int3 is low (false)
- b_irq3- is high (false)

This indicates that neither an Ethernet controller interrupt (e_irq is false), nor a system enable interrupt (en_int3 is false), nor a VME interrupt (b_irq3- is false) are requesting an interrupt. And yet a level 3 interrupt has been asserted over p2_a(3:1). A classic case of the spurious interrupt?

No.

The Ethernet controller has a "feature" built into it which allows it to raise and lower its interrupt asynchronously. Thus the processor may see an interrupt raised, but by the time it has gone out to acknowledge that interrupt it may find that the interrupt request has temporarily disappeared. When this happens on interrupt level 3, the system does not vector to the spurious interrupt trap as it would on any other level; instead it *takes for granted that this disappearing interrupt was asserted by the Ethernet controller.* And U302 asserts valid Ethernet autovector signals on its lp_ack(1:0) output lines.

The cause of this is the Intel Ethernet controller, which can raise and lower its interrupt request line asynchronously in order to protect on-chip status information. The Intel Ethernet chip has no on-chip status registers to synchronously latch status data; status data can be updated any time. However any update to the status register of the Ethernet controller chip is presaged by the deassertion and then assertion of an interrupt request; since status updating can occur asynchronously, it follows that the interrupt request can also rise and fall asynchronously.

This is why, if the conditions for a spurious interrupt are true on a level 3 interrupt request, the processor *takes it for granted* that this spurious condition was caused by the Ethernet controller, and instead acts as if it is a normal Ethernet

interrupt request.

12.8. U305 Latch

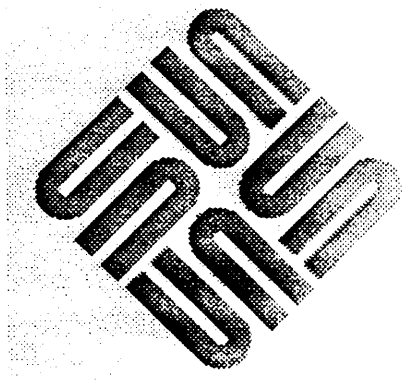
Since the outputs of the three PALs, U304:2, are combinatorial (that is to say, asynchronous), certain of their outputs need to be latched to preserve their state. This is done in U305.

- When cs3- is high, U305 acts as a transparent latch, that is, data at its inputs pass through within a propagation delay on the latch's outputs. Any change of data at the inputs will appear (within the bounds of this same propagation delay) at the latch's outputs.
- When cs3- is low, data is also latched into the latch. However data output is not enabled until the assertion of a low active output enable signal. Since the outputs are permanently enabled by the connection of pin 1 (output enable) to a pulldown resistor, the only limitation (outside of the inherent propagation delay) on data transmission through U305 are the input setup time requirements.

13

ATE Pulldowns — U407

ATE Pulldowns — U407 97



13

ATE Pulldowns — U407

Instead of tying unused signals to ground, they are tied to active pulldown resistors, through U407 octal line drivers. These line drivers are inverters, with outputs permanently enabled by having low active output enables at pins 1 and 19 tied to ground.

Active pulldowns allow ATE overrides for test fixtures.

14

Clock Generation — U400–U406

Clock Generation — U400–U406 ..... **101**

14.1. Pinout of U400 Clock PAL 101
14.2. U400 Input Signals 102
14.3. U400 Output Signals 103
14.4. U401 Flip-Flops 104
14.5. U402 Flip-Flops 104

14

Clock Generation — U400–U406

A single crystal supplies clock signals for the 2060 board:

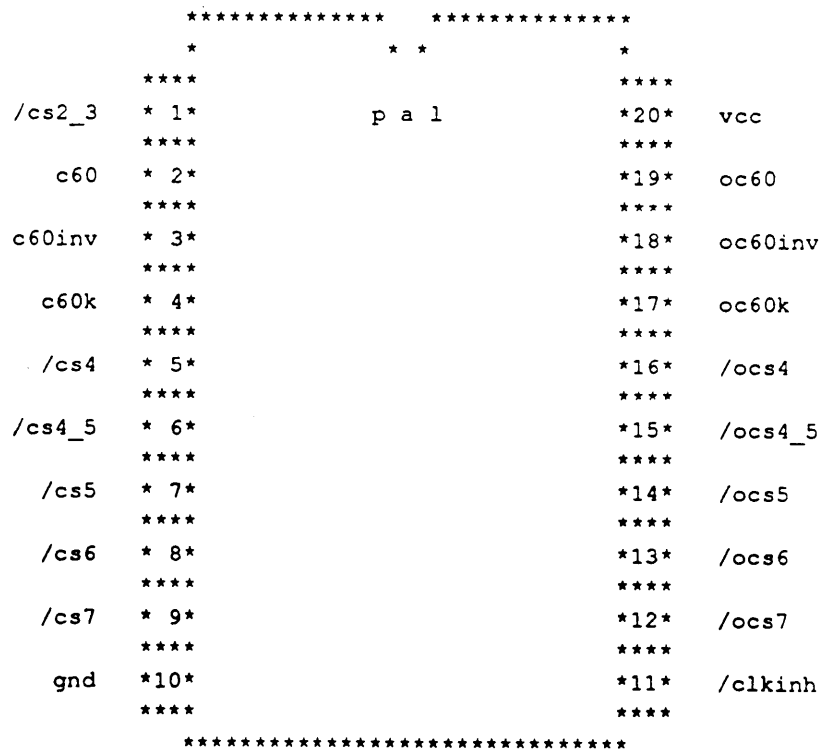
- 33.33 (*ad nauseam*) MHz U403 supplies 16.667 MHz (60 nsec) clock.

This clock may be supplied independently to both the 68020 processor and the 68881 coprocessor by way of the J400 jumper, U400 and U406.

14.1. Pinout of U400 Clock PAL

Pinout of U400 clock PAL is:

Figure 14-1 *U400 Pinout*



14.2. U400 Input Signals

If you look at the schematics on page 4, you will notice that many of the inputs to the U400 clock generator PAL are actually outputs from the state machine.

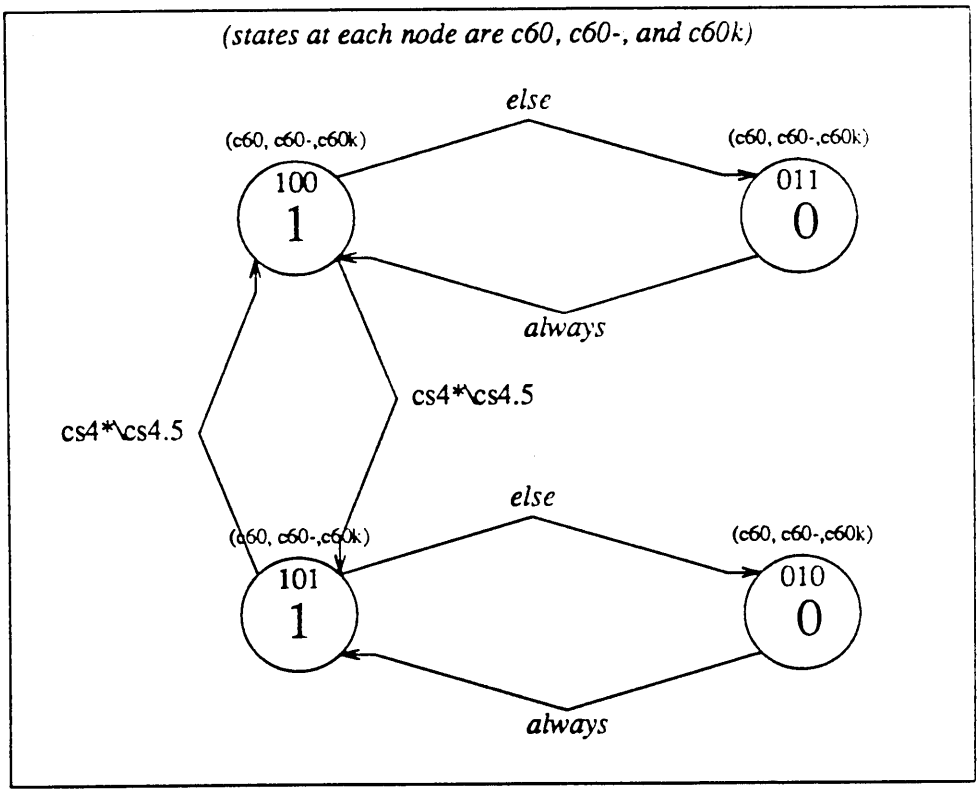
The PAL can be divided into two sections: the top three outputs are all 60 nsec system clock:

- c60 clock
- inverted c60 clock
- a constant version of c60 clock, labelled c60k.

This c60k signal is unlike c60 and c60-. These last two can be stretched (making c4.5, etc.).

The state diagram below illustrates the generation of c60, c60-, and c60k. The "stretch" occurs in the transition from state 100 to 101, or 101 to 100.

Figure 14-2 *Clock Stretch (cs4 — cs4.5) State Diagram*



The bottom five inputs to U400 are the resultant clock states: cs4, cs4.5, cs5, cs6, and cs7.

Inputs to U400 PAL are:

```
cs2_3    =  signal occurring between cs2 and cs3 which is
              used to start cs4
c60      =  60 nsec system clock
c60inv   =  inverted 60 nsec clock
c60k     =  60 nsec constant clock
/cs4     =  clock state 4
/cs4_5   =  stretched cs4 (30 nsec wait state inserted)
/cs5     =  clock state 5
/cs6     =  clock state 6
/cs7     =  clock state 7
/clkinh  =  inhibits clock stretch during FPA or 68881
              bus cycles
```

14.3. U400 Output Signals

Output signals from the U400 PAL are latched in U406; these output signals can be separated into two categories:

- system clock — c60, c60-, and c60k
- phases of system clock — cs4-, cs4.5-, cs5-, cs6-, and cs7.

The PAL logic from which these output signals are derived is given below. (Thi is the code which goes with the clock stretch state diagram given above.)

```
/oc60     = c60 * /c60inv * cs4_5 + 60 nsec system clock
            c60 * /c60inv * /cs4

/oc60inv  = /c60 +  inverted 60 nsec system clock
            /cs4_5 * cs4 +
             c60inv

/oc60k    = /c60 * c60inv * c60k + 60 nsec "constant" clock
             c60 * /c60inv * c60k
```

The remainder are basically the result of a shift register that generates an edge for states 4-7 of the processor clock.

Clock stretch is eliminated on cycles to the FPA or 68881 because the FPA and 68881 operate at zero wait states. This elimination is accomplished by having the clock inhibit signal suppress cs4 and thus never allow a transition from state 101 to 100 (or vice versa) in the state machine.

```
ocs4     =  /clkinh * cs2_3 */cs7    assert for s4 -> s8

ocs4_5   =   cs4 * cs2_3  +          set by cs4 if stretching
             cs4_5 * cs2_3           hold through cs2_3 deassertion

ocs5     =   cs4_5 * cs2_3  +        set by cs4_5 if stretching
             cs5 * cs2_3             hold through cs2_3 deassertion

ocs6     =   cs5 * cs2_3  +          set by cs5
             cs6 * cs2_3             hold through cs2_3 deassertion

ocs7     =   cs6 * cs2_3  +          set by cs6
             cs7 * cs2_3             hold through cs2_3 deassertion
```

Inputs to U406 are latched by the rising edge of master clock. Output of the latches are permanently enabled by connecting the output control at pin 1 to a pulldown.

14.4. U401 Flip-Flops

U401 flip-flops issue synchronized versions of cs2 and cs3. When processor address strobe, p_as-, is asserted, the first positive-going edge of c60 clock asserts cs2- from pin 5 of U401.

This cs2- output is used as an input for the second flip-flop; on the next positive edge of c60- clock, cs3- is asserted from pin 9 of U401.

These two flip-flops are preset by the assertion of system-wide init- signal.

14.5. U402 Flip-Flops

U402 pin 5 is used to generate 80 nsec clock.

U402 pin 9 generates cs2_3-, which causes ocs4- to go active 60 nsecs after cs2.

15

Pal U408

Pal U408 ..... **107**

15.1. Pinout of U408 PAL 107

15

Pal U408

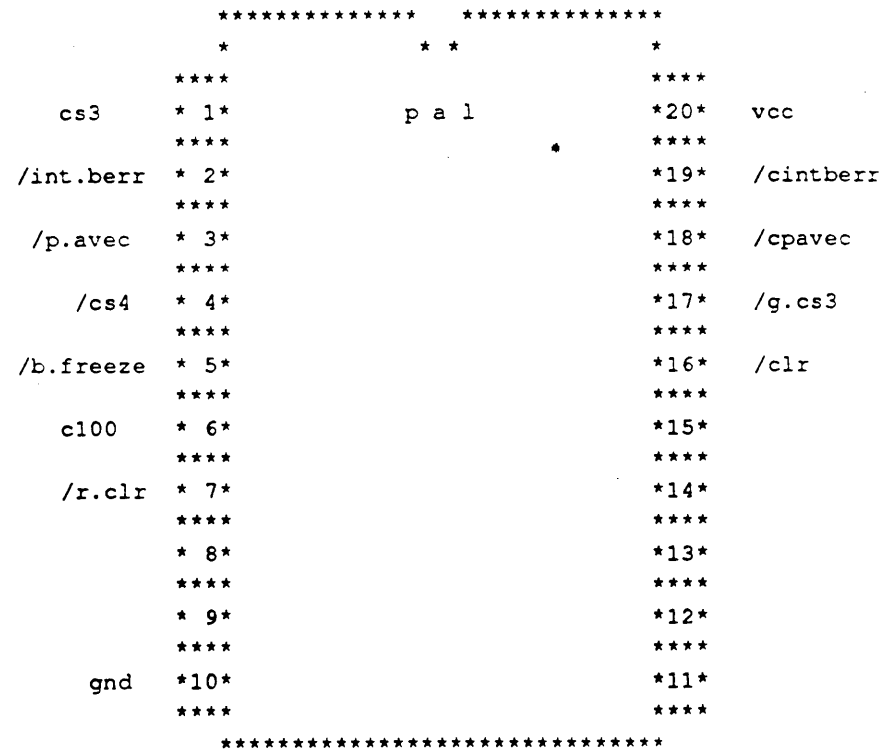
This PAL was added in a later modification to decode and generate the following signals:

- cintberr-
- cpavec-
- G.cs3-
- clr-.

15.1. Pinout of U408 PAL

Pinout of U408 PAL is:

Figure 15-1 *U408 Pinout*



16

Sun-3 Memory Management Unit (MMU)

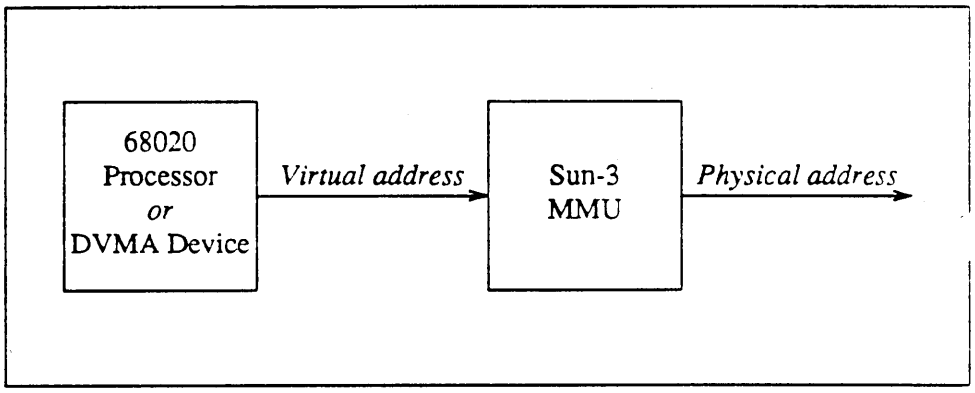
Sun-3 Memory Management Unit (MMU) **111**

16

Sun-3 Memory Management Unit (MMU)

Sun-3 architecture splits the large physical address space into smaller "virtual" address spaces. The 2060 board uses the Sun-3 MMU to generate physical addresses from the virtual addresses is receives from the processor or DVMA devices.

Figure 16-1 *Sun-3 Address Translation*



32 bits of address can define 4 gigabytes (4096 Mbytes) of physical address. However this much physical address is not actually resident on the board; most of it lies in secondary memory devices such as disk and tape drives. To access this memory, virtual addresses from the processor or DVMA devices must be translated into physical addresses. To envision this address space, imagine the virtual address as 32 bits from the processor, each with its own 3-bit context (contexts are also called "processes").

This translation of virtual addresses into physical addresses occurs in the Memory Management Unit (MMU). The Sun-3 MMU consists of three main parts:

- Context Register
- Segment MAP
- Page Map.

This list of MMU parts is hierarchical; that is, each successor further translates (multiplies) its predecessor. For instance, the context register multiplies address

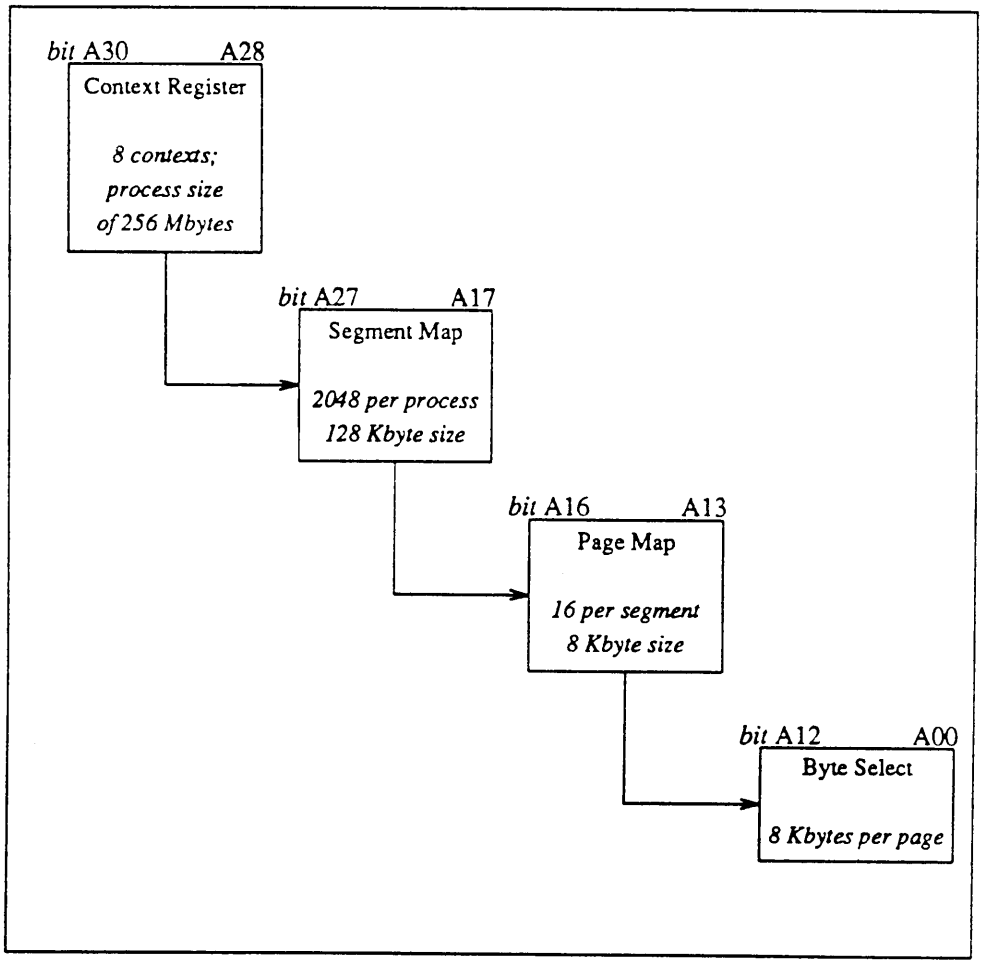
space into one-of-eight contexts; the segment map splits each one of these contexts into one-of-2K segment areas; the page map splits each of these segment areas into one-of-16 pages.

- Each page contains 8 Kbytes (A12:00, 13 bits of address).
- Each segment contains 128 Kbytes (A16:13, another 4 bits of address).
- Each context contains 256 Mbytes (A27:17, another 11 bits of address).

This total of 28 address bits defines virtual address space.

The figure below illustrates this progression from context to segment to page to individual byte location.

Figure 16-2 *Sun-3 MMU*

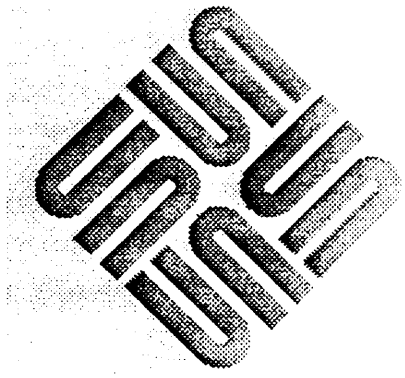


17

Context Register — U509

Context Register — U509 ..... **115**

17.1. U509 Pinout 116
17.2. U509 Input Signals 116
17.3. U509 Output Signals 117

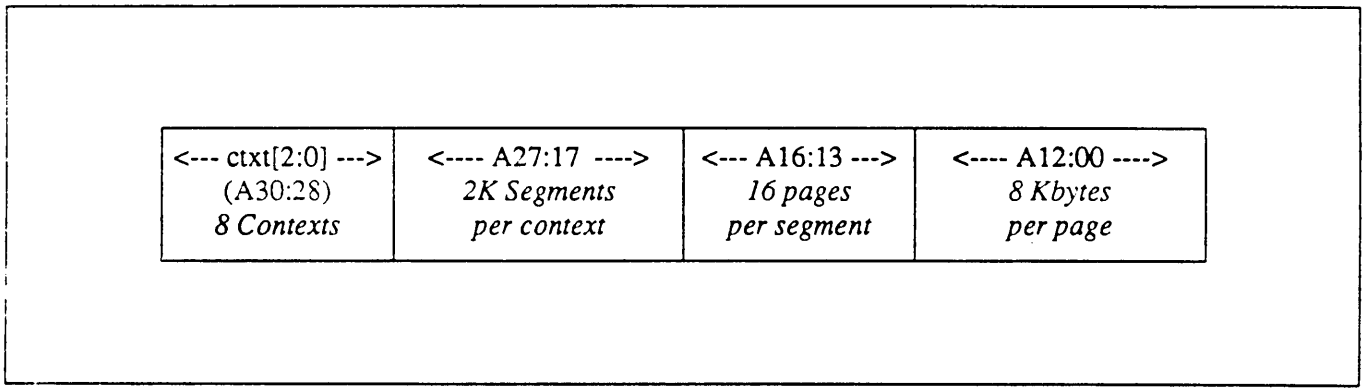


17

Context Register — U509

The Sun-3 MMU multiplies address space into eight distinct 256 Mbyte (28-bit virtual address space) "contexts," by means of the three context address bits ctxt(2:0) encoded within PAL U509. The current context is selected by means of these 3 bits. The same context applies to both user and supervisor state. The figure below illustrates this division of Sun-3 address space.

Figure 17-1 *Context Bits and Virtual Address*



The context register has read and write controls — rd_ctxt- and wr_ctxt- — but no initialization.

Table 17-1 *U509 Context Register — Description*

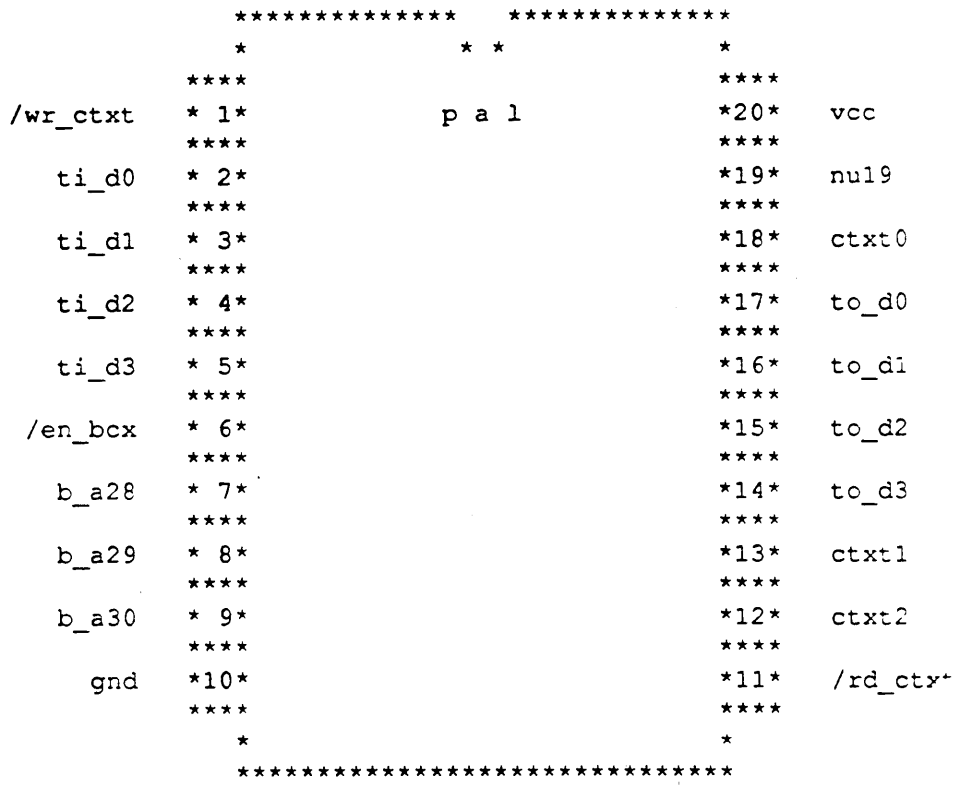
NAME	A<31..28>	SIZE	TYPE
CONTEXT REGISTER	0x3	BYTE	R/W

The U509 context register is actually a PAL that holds the current usr/supervisor context in a register and multiplexes between this present context and the "user context" provided from the VME section of the 2060 board. This PAL latches/outputs data to and from the TTL data bus.

.1. U509 Pinout

Pinout of U509 context register PAL is:

Figure 17-2 *U509 Pinout*



17.2. U509 Input Signals

Inputs to U509 PAL are:

```
ti_d[3:0]     =   TTL data bus (inputs for writes
                    to context register)

en_bcx-       =   enable bus context - from VME section
                    of the 2060

b_a[30:28]    =   ''user context'' information from VMEbus.
```

17.3. U509 Output Signals

Outputs to U509 PAL are:

```
ctxt[3:0]  =   context value that is input to the
                   segment map rams

to_d[3:0]  =   TTL data bus (outputs for reads from
                   context register)
```

PAL logic from which each of these output signals are derived is given below.

The TTL data bus signals, /ti_d[3:0], need to be inverted because outputs are inverted.

```
/to_d[3:0]  =  /ti_d[3:0]
```

The following equations define the multiplexing between the "user context" presented on the VMEbus and the context presented on the TTL data bus inputs.

```
/ctxt0 =  en_bcx * /b_a28 + select VME input
         /en_bcx * /to_d0   select TTL data input

/ctxt1 =  en_bcx * /b_a29 + select VME input
         /en_bcx * /to_d1   select TTL data input

/ctxt2 =  en_bcx * /b_a30 + select VME input
         /en_bcx * /to_d2   select TTL data input
```

Generally, these equations indicate that if the en_bcx- signal from U2904 VME slave request PAL is asserted, then the inputs at b_a(30:28) are selected (VME user context). If en_bcx- is deasserted, the data in the context register are selected.

18

Segment Map — U500:08

Segment Map — U500:08 ..... **121**

18.1. Segment Map Read and Write Cycles 122
Segment Map RAM Read Cycle 122
Segment Map RAM Write Cycle 123
Truth Table for the U508 Buffer 123
18.2. Segment Map RAM Control Signals 123

18

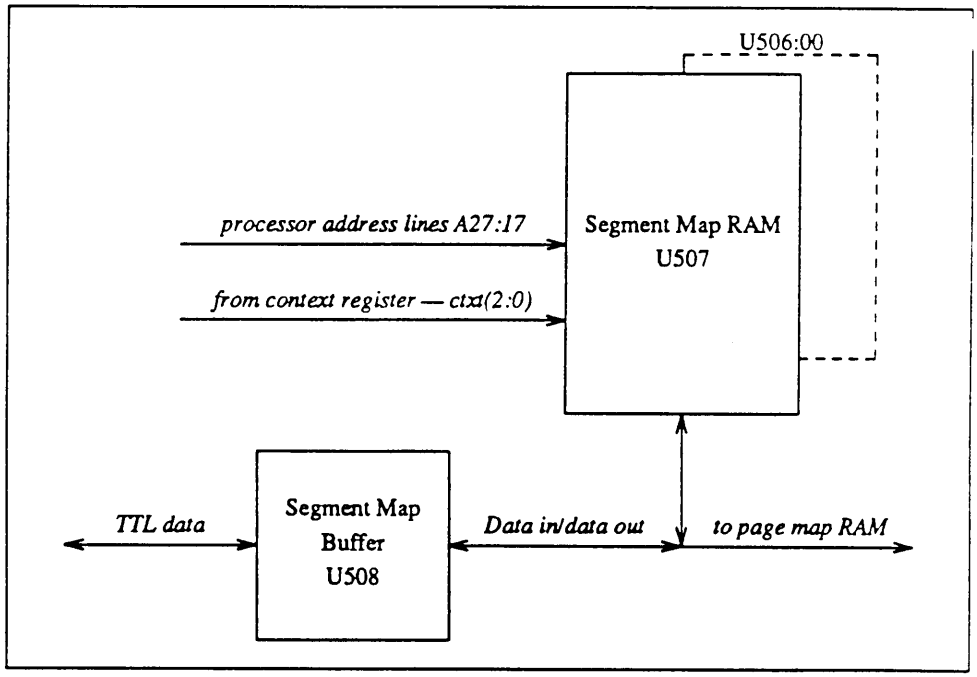
Segment Map — U500:08

The segment map RAM, U507:00, circuitry also includes the segment map transceiver, U508.

The eight segment map RAM chips are individually organized as 16K-by-1. Fourteen address lines — eleven processor address bits (p_a27:17) and three context bits (ctxt[2:0]) — decode to the individual bit-level in each RAM; the eight-bit-wide organization of the RAM array results in an 8-bit byte being read from or written to the segment/page map bus, labelled ia(24:17) on page 5 of the schematics.

The figure below presents a simplified illustration of segment map data flow.

Figure 18-1 *Segment Map RAM — U507:00*



Data can be read from or written to the segment map RAM, depending upon the state of the read and write control signals. There are three signals which control

data flow in this circuit:

```
mmu_weseg- = segment map write enable

mmu_gtseg- = gates (enables output) from U508 bidirectional
             transceiver

p2_rw      = read/write from P2 bus
```

A fourth control signal, chip enable to the RAMs, is permanently asserted by way of a pulldown.

18.1. Segment Map Read and Write Cycles

NOTE *Some of the following information can also be found in the discussion of U1402, the CPU MMU Decoder PAL.*

Data inputs of the segment map RAM chips are wire-ORed to the data outputs (DI to DO). In normal operation, the outputs of the segment map RAM chips are ON (output-enabled) by the deassertion of the mmu_weseg- signal (signal is high). Therefore there is a potential for bus conflict if the read and write timings are not carefully managed; whenever you are writing data from the TTL data bus to the segment map you must make certain that data out from the RAM have first been disabled.

An I/O cycle to the segment map RAM begins with the RAM in a read state — that is, output data is present on their DI/DO pins. The segment map data buffer, U508, is output-enabled by the gate signal mmu_gtseg- from U1402.

Segment Map RAM Read Cycle

A read of the segment map RAM will start with the normal deassertion of the segment RAM write strobe mmu_weseg- (high), the p2_rw signal going high, then mmu_gtseg- going low. The assertion of these two signals enables read data at the DO data output pins of the segment map RAM through U508 onto TTL data bus t_d(7:0).

If you take a look at the TTL bus read timing diagram, you will see that a read cycle begins with cs4 going low, followed by the mmu_gtseg doing likewise. This enables data at the output of the RAM onto the TTL data bus. Since, in the absence of any other operation, data is *always* enabled onto the output of the RAM — CE tied to a pull-down, and write enable false (mmu_wseg- is high). The mmu_gtseg- signal stays true until ttlrdend- is true (see the decode of the U1402 PAL for an explanation of this), whereupon U503 shuts down.

Segment Map RAM Write Cycle

We initiate a write cycle to these RAM chips by first asserting the write strobe mmu_weseg- to the RAM (see mmu_we signal in the TTL BUS WRITES timing diagram), which tri-states the outputs of the RAM chips. Only then is the data buffer (U508) output-enabled onto the segment map RAM data bus, ia(24:17).

A write (p2_rw is low) to the buffer, in combination with mmu_gtseg- going low, enables data on the TTL bus through the U508 buffer onto the data inputs of the segment map RAM.

Truth Table for the U508 Buffer

Thus the truth chart for the U508 buffer is:

Table 18-1 *U508 Segment Map Buffer — Data Flow*

Gate *mmu_gtseg-*	Direction *p2_rw*	Which Way The Data Will Flow
0	0	TTL data bus to RAM (B -> A)
0	1	RAM to TTL data bus (A -> B)
1	X	*tri-state*

The gate signal to the U508 buffer has to be deactivated at the same time as the write enable to the RAM in order to prevent read data (now automatically enabled onto the ia[24:17] bus when the RAM chips go from write to read) from conflicting with the buffer's write data also enabled onto the bus. In other words, you can't have the RAMs enabling data onto the bus at the same time as the buffer enables *its* data onto the bus.

18.2. Segment Map RAM Control Signals

Equations for the segment map RAM control signals are:

```
mmu_weseg = ctlspc*/p_a31*/p_a30* p_a29*/p_a28*
            /p2_rw * /ttlwend*cs2
```

The mmu_weseg- signal is active when you are doing a control space access to the segment map RAM, you are in a write cycle during clock state 2, and you have not entered the end of the write cycle (ttlwend is false/high).

With the write enable strobe issued, the outputs of the RAM go into a tri-state. Note that the RAM go tri-state *before* the U508 buffer is output-enabled (mmu_gtseg- is issued) onto the segment map RAM data bus. The mmu_weseg- signal is issued in cs2; mmu_gtseg- is issued during cs4.

```
mmu_gtseg = ctlspc*/p_a31*/p_a30*p_a29*/p_a28*
                p2_rw*cs4 +

            mmu_gtseg* p2_rw*/ttlrdend +

            ctlspc*/p_a31*/p_a30*p_a29*/p_a28
                 *cs4* /p2_rw * /ttlwend
```

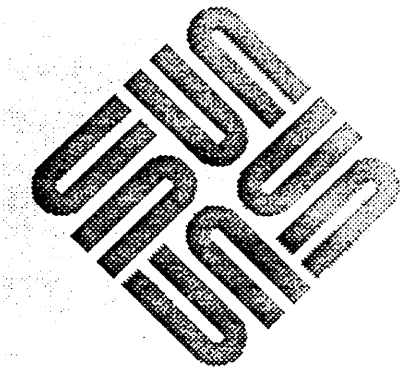
This ensures that there is no buffer conflict between the RAM and the U508 buffer on the segment RAM bus (ia[24:17]).

When mmu_gtseg- is issued in conjunction with p2_rw being low (write cycle), data from the TTL bus is gated through U508 and coupled to the outputs of the segment map RAM. This data is loaded into the RAM on the rising edge of mmu_weseg-, the deactivation of the RAM write enable signal. When mmu_weseg- goes high, the RAM are READ out-enabled, but there is no conflict, because the data output is the same as the data just written in.

19

Page Map RAM

Page Map RAM **127**



19

Page Map RAM

The page map contains 4096 32-bit page entries each mapping to an 8 Kbyte page. Page map entries are composed of a valid bit, protection field, don't cache bit, type field, accessed and modified bits, and a page number.

The page map is divided into 256 sections of 16 entries each. Each section is pointed to by a segment map entry and is called a page map entry group, or pmeg.

Table 19-1 *Format of a Page Map Entry*

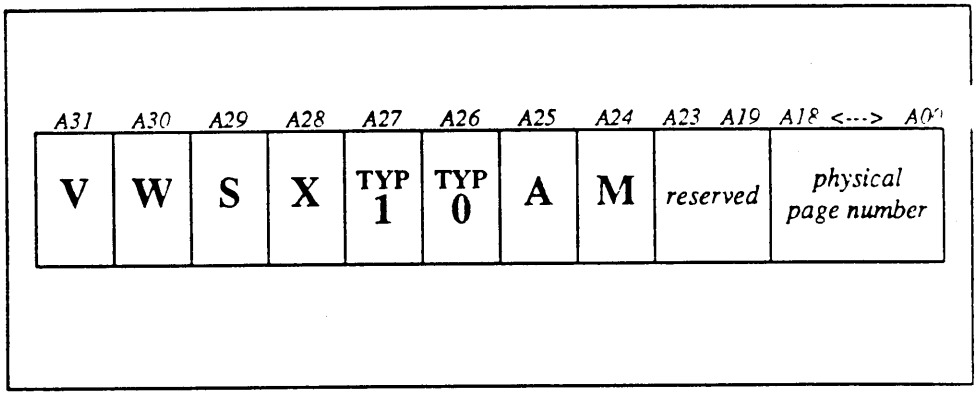


Table 19-2 *MMU Statistics Bits*

Bit	Meaning
v	valid bit, implies read access
w	write access bit
s	system access bit
x	don't cache bit
a	accessed bit
m	modified bit

Table 19-3 *Type-Bit Decode*

Decode of Type Bits		Access
typ(01)	*typ(00)*	
0	0	main memory
0	1	Sun-3 I/O space
1	0	VMEbus 16-bit data
1	1	VMEbus 32-bit data

Since the page map is a 32-bit value, it has to be broken into four 8-bit bytes to fit onto the 8-bit data bus. Address bits p2_a(1:0), decoded through U1402, act as the byte-select mechanism.

Table 19-4 *Byte Selection in the Page Map RAM*

Term	Address Bits		Write Enable Strobe Asserted
	A1	*A0*	
BYTE24	0	0	mmu_we24
BYTE16	0	1	mmu_we16
BYTE08	1	0	mmu_we08
BYTE00	1	1	mmu_we00

The page map RAM operate similarly to the segment map RAM. The page map RAM are normally output enabled—read data is latched to their outputs (since the write enable signal, mmu_we[24/16/08/00] signals are normally high). To write to the page map RAM we must first shut down their outputs by issuing the appropriate write enable strobe. This tri-states the page map's output pins. One of the four U610:07 transceivers is then output-enabled by one of the four gating signals. Disabling the outputs of the RAM occurs during cs2; gating of the transceiver occurs later, in cs4.

An example of the decode of a write enable signal (in this case mmu-we24) is given below:

```
mmu_we24 = ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
           /p2_a01*/p2_a00*/p2_rw*/ttlwend*cs2
```

This signal (mmu_we24) is the write enable for byte 31:24 of the page map RAM. It is active when you are doing a control space access (ctlspc- true) to the page map RAM (pa_31:28 = 0001), the p2_a01:0 address bits decode to BYTE24 (00), you are in a write cycle (p2_rw is low), you have not ended the write cycle (ttlwend is high/false) and you are in clock state two (cs2 is true).

After the write enable disables the output of the RAM at cs2, the appropriate gate signal (mmu_we[24/16/08/00]) is issued during cs4. For instance, decode of the mmu_gt16 signal, which gates U607, reveals:

```
mmu_gt16 = ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
                /p2_a01*p2_a00*p2_rw*cs4 +

           mmu_gt16*p2_rw*/ttlrdend +

           ctlspc*/p_a31*/p_a30*/p_a29* p_a28*
                /p2_a01*p2_a00*cs4*/p2_rw*/ttlwend
```

This mmu_gt16 signal is asserted when:

1. you are doing a control space access and address bits pa_31:28 decode to the page map RAM (0001), the p2_a01:0 address bits decode to BYTE16 (01), you are doing a read cycle (p2_rw is high), and you are in clock state four (cs4 is valid); or
2. mmu_gt16 is asserted (this is a self-latching mechanism, needed to extend the read cycle beyond the deassertion of cs4), p2_rw indicates a read cycle (signal is high), and you have not entered the end of the read cycle (tlrdend is still high); or
3. you are doing a control space access to the device decoded by address bits pa_31:28 to be the page map RAM (0001), p2_a01:0 decode to BYTE16 (01), you are in clock state four (cs4 is valid), you are in a write cycle (p2_rw is low) and you have not yet entered the end of the write cycle (ttlwend is false).

The mmu_gate signal couples data through the appropriate buffer, onto the MMU bus; this data is then written into the page map RAM on the rising edge (deactivation) of the write enable signal.

For more information on these control signals, please see the explanation of PAL U1402.

20

Statistics Control PAL — U611

Statistics Control PAL — U611 ..... **133**

20.1. U611 Input Signals 133
20.2. U611 Output Signals 134

20

Statistics Control PAL — U611

This PAL generates and updates various control and status bits for the MMU.

20.1. U611 Input Signals

Inputs to the U611 PAL are:

```
i_mod        =   modified bit from MMU

i_acc        =   accessed bit from MMU

i_typ[1:0]   =   type bits from MMU

we24-        =   write enable strobe for control
                   space access

p2_rw        =   doing a read/write operation

ttlwend-     =   terminate write cycle for above

cs4-         =   state clock - used to time
                   statistics updating

cs4_5-       =   state clock - used to time
                   statistics updating

cs5-         =   state clock - used to time
                   statistics updating

cs7-         =   state clock - used to time
                   statistics updating

val-         =   cycle is a legal device space
                   cycle - do update
```

The following clock states are used in statistics update timing:

```
s4      = latch output of RAM

s4_5    = assert we- on the static RAM
            (turns off RAM drivers)

s5      = turn on PAL to get updated version of
            the statistics bits

s7      = deassert we- and turn off the PAL outputs
```

20.2. U611 Output Signals

Outputs from the U611 PAL are:

```
stwe-     =     write enable for MMU RAM

o_typ1    =     updated version of stat bit
o_typ0    =     updated version of stat bit
o_acc     =     updated version of stat bit
o_mod     =     updated version of stat bit

lmod      =     latched version of modified bit
```

Write strobe — there are two sources of writes to the static RAM in the MMU. The processor needs to be able to read/write, and the statistics bits must be updated when a device space cycle occurs.

- Processor writes are terminated with ttlwend since the decode path for we24 is quite slow.
- On device space updates we assert stwe- at cs4_5 and deassert at cs7.

```
stwe  =     we24 * /ttlwend +   control space cycles

            val * cs4_5 * /cs7  device space cycles
```

If we define output enable as:

```
#define OE      val*cs5*/cs7
```

then the updated version of statistics bits are a result of:

```
if ( TYPEOE ) /o_typ1 =  /i_typ1
```

```
if ( TYPEOE ) /o_typ0 =  /i_typ0
```

```
if ( OE ) /o_acc  =  gnd      always a 1
```

```
if ( OE ) /o_mod  =  /lmod  latched version of modified bit
```

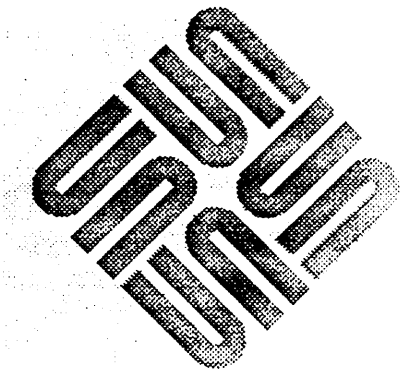
```
/lmod  = /i_mod * p2_rw * /cs4 + set if write or already set
         /lmod * cs4                hold as long as cs4
```

21

MMU Validation and Decode PAL — U612

MMU Validation and Decode PAL — U612 ..... **139**

21.1 U612 Input Signals 139
21.2 U612 Output Signals 140



21

MMU Validation and Decode PAL — U612

U612 PAL decodes various MMU validation type bits and system control signals to issue MMU control and error signals.

21.1. U612 Input Signals

Inputs to U612 are:

```
mmu_typ[1:0] =  type bits determine physical
                  address space

mmu_s        =  page is supervisor access only

mmu_w        =  page is writable

mmu_v        =  page is valid

p2_rw        =  read/write-

cs4-         =  cs4 - output of MMU is stable by now

cs4_5-       =  cs4_5 - feedback is stable - freeze

devspc-      =  doing a device space (physical
                  address) cycle

p2_fc[2]     =  function code 2 - doing a
                  supervisor cycle

p2_as-       =  used to end cycles (especially for
                  mmu_ram-)
```

..2. U612 Output Signals

Outputs from U612 are:

```
ltyp0      =     latched version of type0

mmu_vme-   =     valid VME master cycle (to type2
                   or type3 space)

mmu_io-    =     valid IO cycle (to type1 space)

         The above signals are asynchronous state machines
         that latch when asserted, and stay latched through
         p_as (end of cycle). This is done because the
         the type bits may glitch during statistics updating.


mmu_ram-   =     valid RAM cycle (type0 space -
                   includes video)

mmu_verr   =     page is invalid

mmu_perr   =     page is valid, but protection is bad

mmu_val-   =     physical address cycle's protection is
                   ok, allow statistics updating
```

The ltyp0- signal provides a latched version of the type[0] bit for VME dsack signals (input to U204 dsack PAL).

```
/ltyp0 := p2_as * cs4 * /cs4_5 * /typ0 +   sets the latch

          p2_as * /ltyp0   hold until p2_as is no longer valid
```

The protection bits are derived from the following truth table.

Table 21-1 *MMU Protection Bits — Decode of the Inputs*

Input Signals			Meaning
devspc-	*	mmu_v	enable cycle
p2_rw	*	mmu_w	read protected
p2_fc2	*	mmu_s	supervisor protected

Table 21-2 *Truth Table for MMU Protection Bits*

Inputs						Outputs			Meaning	Status
devspc-	mmu_v	p2_rw	mmu_w	p2_fc2	mmu_s	mmu_verr	mmu_perr	mmu_val-		
1	-	-	-	-	-	0	0	1	*not device space cycle*	
0	0	-	-	-	-	1	0	1	*page is marked invalid*	
READS										
0	1	1	-	-	0	0	0	0	*usr,supv <- usr*	
0	1	1	-	0	1	0	1	1	*usr <- supv*	*ERROR*
0	1	1	-	1	1	0	0	0	*supv <- supv*	
WRITES										
0	1	0	0	-	-	0	1	1	*usr,supv -> prot*	*ERROR*
0	1	0	1	-	0	0	0	0	*usr,supv -> usr*	
0	1	0	1	0	1	0	1	1	*usr -> supv*	*ERROR*
0	1	0	1	1	1	0	0	0	*supv -> supv*	

The mmu_val- term produces the following set of validation equations:

```
mmu_val- =

devspc * mmu_v * p2_rw * /mmu_s +

devspc * mmu_v * p2_rw * p2_fc2 +

devspc * mmu_v * mmu_w * /mmu_s +

devspc * mmu_v * mmu_w * p2_fc2
```

These basic equations are used for mmu_val-, mmu_vme-, mmu_io-, and mmu_ram-. They are further qualified by clock edges (cs4, cs5) and also by type space decode bits (typ[1:0]).

```
/mmu_verr  = /cs4 +     qualified by cs4

             /devspc +

             mmu_v
```

```
/mmu_perr  = /cs4 +       qualified by cs4

             /devspc +

             /mmu_v +

             p2_rw * /mmu_s +

             p2_rw * p2_fc2 +

             mmu_w * /mmu_s +

             mmu_w * p2_fc2
```

All terms qualified by cs4

```
mmu_val = cs4 * devspc * mmu_v * p2_rw * /mmu_s +

          cs4 * devspc * mmu_v * p2_rw * p2_fc2 +

          cs4 * devspc * mmu_v * mmu_w * /mmu_s +

          cs4 * devspc * mmu_v * mmu_w * p2_fc2
```

Enabled at s4, and frozen at s5. VME cycles are type = (2 or 3). Signal is held valid till cycle ends, indicated by deassertion of p2_as.

```
mmu_vme := p2_as * cs4*/cs4_5 * devspc *
             mmu_v * p2_rw * /mmu_s * typ1 +

           p2_as * cs4*/cs4_5 * devspc * mmu_v *
             p2_rw * p2_fc2 * typ1 +

           p2_as * cs4*/cs4_5 * devspc * mmu_v *
             mmu_w * /mmu_s * typ1 +

           p2_as * cs4*/cs4_5 * devspc * mmu_v *
             mmu_w * p2_fc2 * typ1 +

           p2_as * mmu_vme  hold through end of cycle
```

Enabled at s4, and frozen at s5. IO cycles are type = (1). Signal is held valid till cycle ends, indicated by p2_as being deasserted.

```
mmu_io := p2_as * cs4*/cs4_5 * devspc * mmu_v *
            p2_rw * /mmu_s * /typ1 * typ0 +

          p2_as * cs4*/cs4_5 * devspc * mmu_v *
            p2_rw * p2_fc2 * /typ1 * typ0 +

          p2_as * cs4*/cs4_5 * devspc * mmu_v *
            mmu_w * /mmu_s * /typ1 * typ0 +

          p2_as * cs4*/cs4_5 * devspc * mmu_v *
            mmu_w * p2_fc2 * /typ1 * typ0 +

          p2_as * mmu_io
```

The mmu_ram signal is not qualified by cs4; it is sampled by the memory CAS decoder and the video interface hardware. RAM cycles are type = (0).

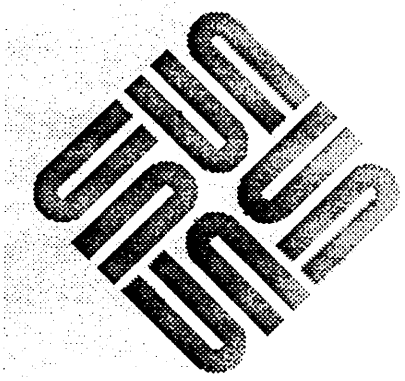
```
mmu_ram := p2_as * devspc * mmu_v * p2_rw *
             /mmu_s * /typ1 * /typ0 +

           p2_as * devspc * mmu_v * p2_rw *
             p2_fc2 * /typ1 * /typ0 +

           p2_as * devspc * mmu_v * mmu_w *
             /mmu_s * /typ1 * /typ0 +

           p2_as * devspc * mmu_v * mmu_w *
             p2_fc2 * /typ1 * /typ0
```

22

P2 Bus Control and Address Buffers

P2 Bus Control and Address Buffers **147**

22.1. U700 Comparator 147
22.2. U703:01 P2 Address Buffers 147
22.3. U704 Control Signal Buffer 148
22.4. Aliases 148



22

P2 Bus Control and Address Buffers

Page 7 of the schematics includes the

- U700 comparator — decodes address bits to indicate an access to bottom 32 Mbytes of memory
- U703:01 P2 address buffers
- U704 Control signal buffer.

22.1. U700 Comparator

This comparator issues the signal that indicates an access is being made to the bottom 32 Mbytes of memory, the amount of physical memory addressable by A24:00.

In the comparator, mmu_a(31:25) address bits are compared to the ground inputs at P(7:0) and when the comparison is equal — mmu_a(31:25) address bits are all zeros — the low active signal p2_bot32M- is issued. This signal is used in the 3104 CAS decoder PAL as indication that an access is is being made to the bottom 32 Mbytes.

Input to the comparator is permanently enabled by the connection of a pulldown to pin 1, the gating signal. The table below summarizes the logic.

Table 22-1 *U700 Comparator*

Inputs		Output
G- (Input enable)	*mmu_a(31:25)*	*p2_bot32M-*
Low	All low (equal GND)	Low
Low	High (not equal to GND)	High
High	Doesn't matter	High

22.2. U703:01 P2 Address Buffers

U703:01 buffer the MMU address bits mmu_a(23:13) coming from the page map RAM along with unbuffered processor address bits p_a(12:00) to the P2 address bus. The two output enable pins of these three buffers are permanently gated ON by being connected to a pulldown, and address bits at the inputs are coupled immediately to the P2 address bus (minus propagation delay, of course).

3. U704 Control Signal Buffer

U704 operates in the same manner as the three address buffers described above. Control signals asserted at the inputs are driven onto the P2 bus (except for the p1_sysc system clock signal from U2505 crystal, which is available only if J2502 is IN). Notice also that mmu_a(24), address bit 24 from the page maps, is connected to U704; there wasn't enough room in U703:01 for it.

Just like U703:01 buffers, U704 has its two output enables constantly gated ON by being permanently connected to a pulldown.

Signals buffered by U704 are:

- mmu_a(24) — upper/lower 32 Mbyte select bit;
- p_fc(2) — unbuffered high order function code bit from the processor;
- c62 — 62 nsec clock from the 16 MHz crystal on page 25 of the schematics, U2505. This signal is available only if J2502 is IN.
- refr- — refresh signal from U2409 DVMA controller;
- p_siz(1:0) — unbuffered processor size bits, indicating the size of the data to be transferred over the bus;
- p_as — address strobe, whose assertion and deassertion define the beginning and end of a cycle;
- p_rw — processor read/write signal.

4. Aliases

The signals in the lower right hand of page 7 are merely aliases. Thus, devspc- is the same as p2_devspc-. And so on.

```
devspc    <-> p2_devspc-
cs2-      <-> p2_uas-
cs3     <-> p2_mux-
mmu_ram- <-> p2_ram-
cs4-      <-> p2_cas-
cs6-      <-> p2_endras-
```

23

Parity Circuitry

Parity Circuitry **151**

23.1. Parity Address Latch — U811:08 151
23.2. Parity Generator/Checkers — U807:04 151
23.3. U803 Multiplexer 152
23.4. Parity Control and Parity Check PALs — U802 and U812 152
23.5. U812 Parity Check PAL 153
U812 Input Signals 153
U812 Output Signals 153
23.6. U802 Parity Control PAL 156
U802 Input Signals 156
U802 Output Signals 157
Pinout of U802 PAL 159
23.7. Memory Error Register — U801 161
23.8. Byte Select Buffer (and Address Bit Driver) — U813 162
23.9. Parity Data Buffer — U3112 162

23

Parity Circuitry

23.1. Parity Address Latch — U811:08

In the event of a parity error, the parity address latches will latch up all 28 bits of virtual address for the MMU to translate. Since the TTL data bus is only 8 bits wide, this address must be multiplexed. Multiplexing is accomplished by the individual assertion of one-of-four read byte parity strobes, rd_pad(24/16/08/00), to one of the four parity address latches.

The latches are loaded by an upward transition of the parity address strobe signal, par_as, from U802 parity control PAL. Bytes of address are output sequentially onto the TTL data bus, t_d(7:0), by the individual assertion of the following read parity address signals:

- rd_pad(00) — outputs the low order byte of address, bits 7:0 from U811
- rd_pad(08) — outputs the next highest byte of address, bits 15:8 from U810
- rd_pad(16) — outputs the next highest byte of address, bits 23:16 from U809
- rd_pad(24) — outputs the high order byte of address, bits 31:24 from U808.

U808 also latches up the three context bits, for use by the MMU in translating this virtual address to an physical address.

Finally, U808 latches up the s_dma signal, indicating whether or not a DMA cycle was being executed.

23.2. Parity Generator/Checkers — U807:04

The 2060 board uses typical F280 9-bit parity generator/checkers. Each of the four chips accept 9 bits of parity generator/check data, detect whether an odd or even number of these are high, and issue a parity data bit at the SUM ODD output.

- If the number of high inputs is ODD (1, 3, 5, 7, or 9) then the SUM ODD output is high.
- If the number of high inputs is EVEN (0, 2, 4, 6, or 8) then the SUM ODD output is low.

Outputs of the parity checker/generators are connected to U812 parity check PAL, and also U801 memory error register.

The F280s operate in both read (generation) and write (check) modes.

Software can test the operation of the parity chips by setting the ninth parity bit (through U803 multiplexer), par_test. By setting this bit, you can force bad parity on every byte and check for the resultant parity error.

23.3. U803 Multiplexer

The software test bit, par_test, is one of the inputs multiplexed through U803 F158. Output is always enabled from the mux since pin 16 is permanently tied to a pulldown. The output of the mux is inverted; thus, setting the parity test bit, par_test, will add a low to the 9-bit input of the parity checker/generators, forcing bad parity.

Multiplexing is done by the assertion of the processor read/write signal, p2_rw.

- When p2_rw is high (indicating a read cycle), the inputs at port A are selected for transmission. In this case a read cycle selects the parity test bit, par_test.
- When p2_rw is low (indicating a write cycle), the inputs at port B are selected for transmission. In this case a write cycle asserts the P2 parity byte-select bits, p2_par(24:16:08:00), from U801 memory error register.

23.4. Parity Control and Parity Check PALs — U802 and U812

These two PALs work in tandem; U812 looks at the parity check bits from the parity checkers, the size and offset bits from the processor, and asserts

- a parity error signal (parerr-), if appropriate,
- a VMEbus error signal, s_error, and
- one of the four parity error byte select signals, par_err(24:16:08:00).

A read cycle from memory uses the size (siz[1:0]) and offset (p_a[1:0]) bits to determine exactly which byte(s) was (were) transferred. Decoding these bits through U812 allows erroneous data to be located down to an individual byte.

The parity error signal, parerr-, is connected to the U802 PAL. U802 asserts a sample-window signal, sample-, to indicate at what time during the read or write cycle it is valid to check parity; on the next following edge of c60 clock U812 checks parity and indicates status to U802 via the parerr- signal. If parerr- is asserted, U802 will decode it and issue the parity interrupt request signal, par_irq, if appropriate.

23.5. U812 Parity Check PAL

U812 Input Signals

Inputs to U812 PAL are:

```
/sample      =  indicates a valid sample period
                  for checking parity error

/vmeberr     =  VMEbus error

p2_siz(1:0)  =  size of transfer, used to determine
                  individual byte that generated
                  the parity error

p2_a(01:00)  =  size of byte offset

par_d(24:16:08:00)  =  parity check bits
```

U812 Output Signals

Outputs from U812 parity check PAL are derived below.

```
par_err(24) := /sample * /p2_a01 * /p2_a00 * par_d24 +

               sample * par_err(24)
```

The first term of the above equation says that the parity error signal for the most significant byte, D(31:24), is asserted when the sample signal is not true (not in state cs8_cs0), offset bits indicate 0 byte offset, and the parity check bit comes from the high order data byte, p2_d(31:24).

The second term of the above equation is a self-latching function; it is true during the sampling period defined in U802 PAL — cs8_cs0.

The parity error signal for byte (23:16) is defined below.

```
par_err(16) := /sample * p2_siz1 * /p2_a01 * par_d16 +

       /sample * /p2_siz0 * /p2_a01 * par_d16 +

       /sample * /p2_a01 * p2_a00 * par_d16 +

       sample * par_err(16)
```

This equation defines the parity error signal for data bits D(23:16), the check bit from U806. The first three terms indicate that par_err(16) will be asserted whenever the check bit from U806 parity checker is valid, you are NOT in the sampling period, and your data is offset 1, 2, or 3 bytes.

The last term is self-latching; par_err(16) is valid during the sampling period.

The parity error signal for byte (15:08) is defined below.

```
par_err(08) := /sample * /p2_siz1 * /p2_siz0 *
                  /p2_a00 * par_d08 +
                  longword transfer, 0 or +2 byte offsets

               /sample * p2_siz1 * p2_siz0 *
                 /p2_a00 * par_d08 +
                 3-byte transfer, 0 or +2 byte offsets

               /sample * p2_siz1 * /p2_a01 *
                  p2_a00 * par_d08 +
                  1 or 3-byte transfers, +1 byte offset

               /sample * /p2_siz0 * /p2_a01 *
                  p2_a00 * par_d08 +
                  Word or longword transfer, +1 byte offset

               /sample * p2_a01 * /p2_a00 * par_d08 +
                  +2 byte offset

               sample * par_err(08) self-latching
```

The first five terms indicate that par_err(08) can be indicated during a non-sampling period whenever the par_d08 signal is asserted from U805. The last term is self-latching; it holds par_err(08) valid during the sampling period.

The parity error signal for byte (07:00) is defined below.

```
par_err(00) := /sample * /p2_siz1 * /p2_siz0 *
                  par_d00 + longword transfer

               /sample * p2_siz1 * p2_siz0 *
                  p2_a00 * par_d00 +
                  3 byte transfer, 0 or +2 byte offset

               /sample * p2_siz1 * p2_a01 * par_d00 +
                2 or 3 byte transfer; 2 or 3 byte offset

               /sample * p2_a01 * p2_a00 * par_d00 +
                  3 byte offset

                sample * par_err(00)
                   self-latching during sample
```

The s_error signal is defined below.

```
/s_error := /vmeberr * /p2_siz0 * /p2_a00 *
              /par_d24 * /par_d16 * /par_d08 * /par_d00 +

            /vmeberr * p2_siz1 * /p2_siz0 * /p2_a01 *
              /p2_a00 * /par_d24 * /par_d16 +

            /vmeberr * /p2_siz0 * p2_a01 * /p2_a00 *
              /par_d08 * /par_d00 +

            /vmeberr * /p2_siz1 * p2_siz0 * /p2_a01 *
              /p2_a00 * /par_d24 +

            /vmeberr * /p2_siz1 * p2_siz0 * /p2_a01 *
               p2_a00 * /par_d16 +

            /vmeberr * /p2_siz1 * p2_siz0 * p2_a01 *
              /p2_a00 * /par_d08 +

            /vmeberr * /p2_siz1 * p2_siz0 * p2_a01 *
              p2_a00 * /par_d00 +

            /sample * /vmeberr
```

All the parity errors are ORed together into one signal, parerr-, and connected to the parity control PAL, U802. Since this output is clocked, the signal is not valid until 2 states after you see sample-.

```
parerr :=  par_err(24) +
           par_err(16) +
           par_err(08) +
           par_err(00)
```

23.6. U802 Parity Control PAL

The parity control PAL works in tandem with U812, parity check PAL, to generate the parity interrupt request signal, par_irq, during the appropriate sampling period (between cs8 and cs0). Also generated are:

- output enable signal for the high-order byte of parity address (and context bits), rd_pad24-, which clears parity errors,
- latch strobe for the four parity address registers, par_as, and
- the sample- signal, which tells U812 when to check for parity errors.

U802 Input Signals

Inputs to U802 PAL are:

```
par_ien   =    Parity Interrupt Enable (from
                 memory control reg) allows software to
                 selectively enable or disable the parity
                 interrupt.

par_chk   =    Parity Check Enable (from memory
                 control reg) indicates whether you are in
                 parity check or parity generation.

p2_rw     =    Indicates a read or write cycle (used to
                 indicate read here).


p2_ack-   =    Positive acknowledge from memory.

devspc-   =    Cycle is a device space cycle (not FPA).

pad24-    =    Select from TTL decode logic (used to
                 unfreeze).

parerr-   =    Indicates the Parity Checker PAL has
                 detected an error.
```

U802 Output Signals

Outputs of U802 PAL are:

```
par_irq  =  Parity Error Interrupt Request.

par_as   =  Parity Address Strobe - latch virtual address.

sample-  =  Tell Parity Checker PAL to check for error

rd_pad24 =  Read most significant byte of parity address
```

Equations of each of these signals are given below. The first is rd_pad24:

```
rd_pad24 = pad24 * p2_rw
```

pad24 valid from U1401 TTL Bus decoder, during a read cycle

A state diagram illustrates the derivation of the remaining three signals, sample-, par_as, and par_irq. First, the states of the state diagram are:

Table 23-1 *Parity State Diagram — State Values*

Control State Bits *q2*	*q1*	*q0*	State
0	0	0	nu0
0	0	1	nu1
0	1	0	freeze
0	1	1	sendresults
1	0	0	nu4
1	0	1	genparity
1	1	0	latchparity
1	1	1	idle

Next, the meaning of these states is given in the following table:

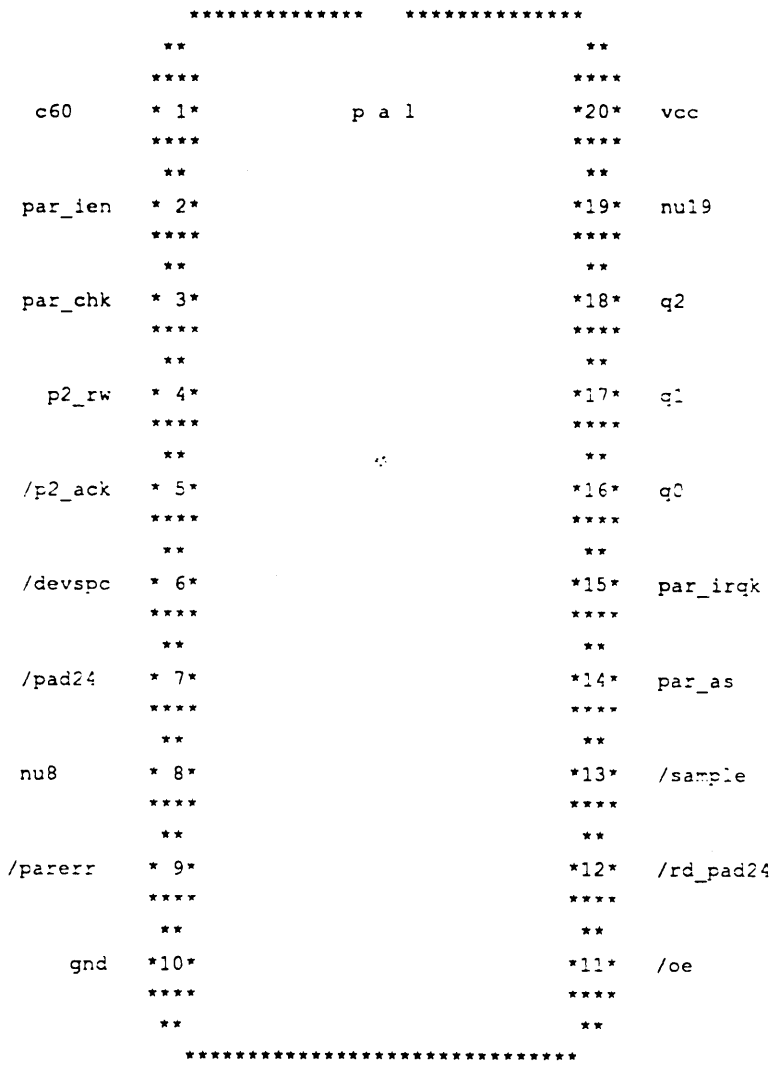
Table 23-2 *Parity State Diagram — Description of the States*

States	Description
nu0	State 0, not used
nu1	State 1, not used
freeze	Parity error - freeze latch and parity check bits
sendresults	Parity Checker sends back the results of check
nu4	State 4, not used
genparity	Parity Checker generates parity check bits
latchparity	Up address and tell Parity Checker to check
idle	Not enabled + Not a read + Not acknowledged + Not devspc access

Pinout of U802 PAL

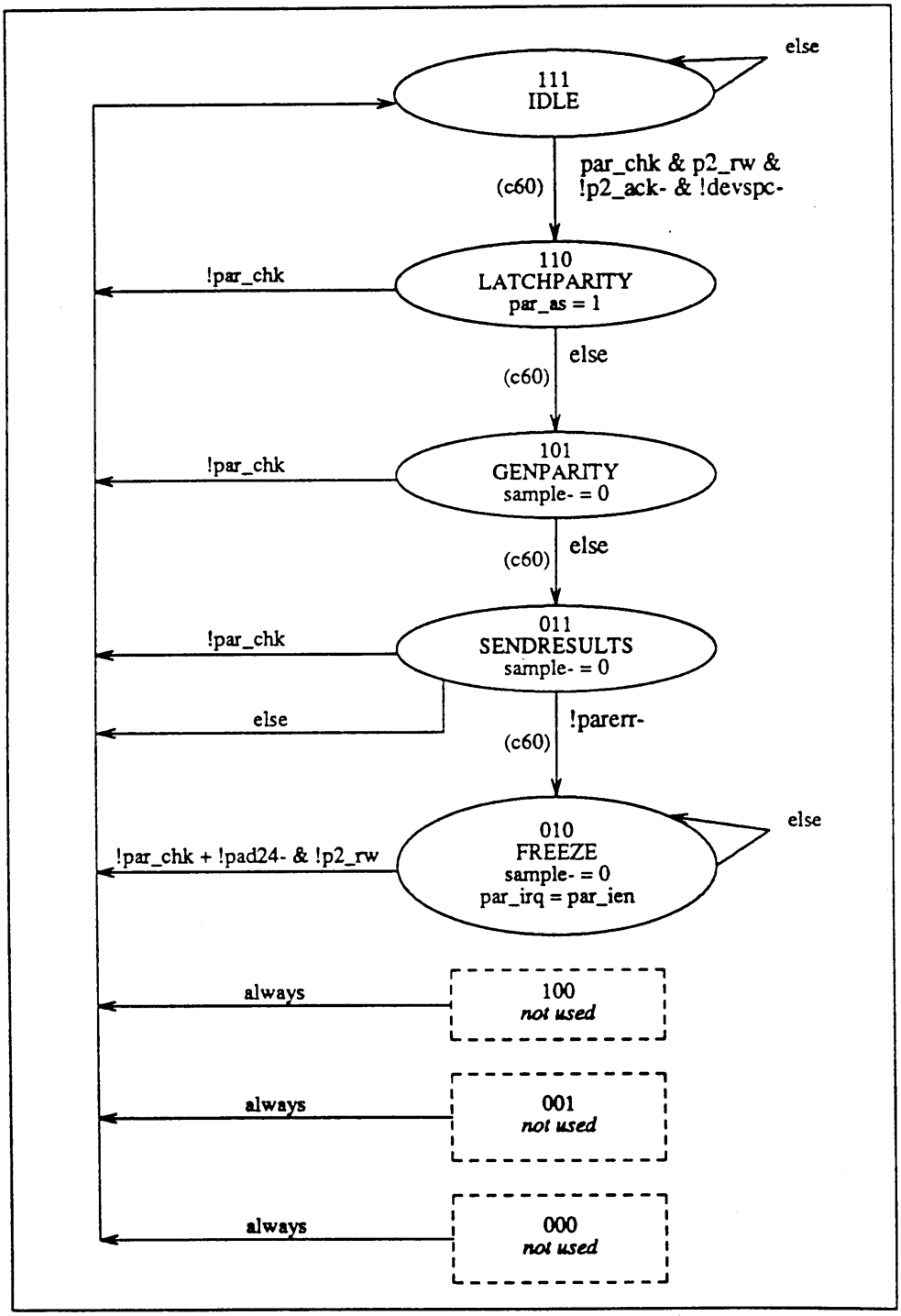
Pinout of the U802 PAL is:

Figure 23-1 *U802 Pinout*



The following is the state diagram for the 2060 parity control PAL.

Figure 23-2 *U802 Parity Control PAL — State Diagram*



The sample- signal defines the time period when a valid interrupt sample may be taken.

```
sample  := q1 * /q0 * /pad24 * par_chk +
           q1 * /q0 * p2_rw * par_chk +
           /q2 * q1 * q0 * parerr * par_chk +
           q2 * /q1 * q0 * par_chk +
           q2 * q1 * /q0 * par_chk
```

Parity address strobe clocks the four parity address registers.

```
/par_as  := /q2 +
            /q1 +
            /q0 +
            /devspc +
            /p2_rw +
            /par_chk +
            /p2_ack
```

Parity interrupt request raises a non-maskable level 7 interrupt.

```
/par_irq := /q0 * pad24 * /p2_rw +
            q0 * /parerr +
            /par_ien +
            /q1 +
            /par_chk +
            q2
```

23.7. Memory Error Register — U801

The memory error register provides the necessary control and information to deal with parity errors. It stores information on which byte(s) caused the parity error, sets a pending parity error interrupt, and provides functions to test parity error checking.

- When rd_par- signal is asserted, the four parity control bits are enabled.
- When the p2_rw signal is low, indicating a write cycle, the four parity error bits are asserted.

NOTE *The parity control and error bits can be asserted and deasserted independently.*

Table 23-3 *Memory Error Register — U801*

Memory Error Register for Parity Memory				
BIT	**NAME**	**TYPE**	**MEANING**	**GATING SIGNAL**
D<0>	PARITY ERROR 00	read-only	Parity Error, bits D07:D00	rd_par- (U813)
D<1>	PARITY ERROR 08	read-only	Parity Error, bits D15:D08	rd_par- (U813)
D<2>	PARITY ERROR 16	read-only	Parity Error, bits D23:D16	rd_par- (U813)
D<3>	PARITY ERROR 24	read-only	Parity Error, bits D31:D24	rd_par- (U813)
D<4>	PARITY CHECK	read-write	Enable parity checking	rd_par- (U801)
D<5>	PARITY TEST	read-write	Test by inverting parity	rd_par- (U801)
D<6>	PARITY INT ENBL	read-write	Parity interrupt enable	rd_par- (U801)
D<7>	PARITY INTRPT	read-only	Parity interrupt (level 7)	rd_par- (U801)

The memory error bits are described below.

- The four parity error bits are set when a parity error is detected in the corresponding byte. These come from the F280 checkers.
- Parity check bit is set to enable parity checking on memory read cycles.
- Parity test bit is set in order to write parity with the inverse polarity to test the operation of the parity error circuitry. With parity test off, correct parity is generated on all memory write cycles.
- Parity interrupt enable enables level 7 interrupts if a parity error is detected. (Remember, level 7 interrupts are non-maskable.)
- Parity interrupt is true if a parity interrupt is pending.

23.8. Byte Select Buffer (and Address Bit Driver) — U813

Only half of this ALS240 buffer is actually used in the parity circuitry; the four parity error byte-select bits (par_err[24:16:08:00]) are gated out onto the TTL data bus by the assertion of the rd_par- signal. These parity error signals indicate which byte in the 32-bit data space actually caused the parity error.

The remainder of the buffer is used to drive the four most-significant address bits during a DMA cycle. Address bits 31:28 are driven high during a DMA cycle (indicated by the assertion of s_dma-); since the ALS240 is an inverting driver, the addresses are derived from ground on pins 2, 4, 6, and 8 of the input side.

23.9. Parity Data Buffer — U3112

The four parity check bits, p2_par(24:16:08:00), are gated through U3112 as soon as they arrive at the inputs (output is permanently enabled by connection to a pulldown).

The other half of the buffer drives the memory parity out bits (m_po(24:16:08:00) from each byte of memory. These are gated out of the buffer by the assertion of the m_parrd- signal from U3104 PAL.

24

MOS Bus Devices

MOS Bus Devices **165**

24.1. U900 MOS Enables PAL 166
U900 Pinout 167
U900 MOS Write Enable — moswren 167
U900 MOS Read Enable — mosrden 168
Diagnostic Cycle — diagcy 169
24.2. U901 MOS SACK State Machine 169
Pinout of U901 PAL 170
U901 MOS Read/Write Control 172
U901 SCC Interrupt — SYNCWAIT State 176
24.3. U904 MOS Read/Write Strobe Decoder 179
U904 Pinout 179
U904 Input Signals 180
U904 Output Signals 180
24.4. U902 MOS Write and U903 MOS Read Buffers 184
U902 MOS Write Buffer 184
U903 MOS Read Buffer 184
24.5. MOS Read and Write Cycles 185
MOS Read Cycle 186
MOS Write Cycle 186
24.6. Mouse and Keyboard SCC 186
U405 and U2207 Baud Rate Clock 187
Transmit Data Path 187

Receive Data Path 187
24.7. Serial Ports A and B — ttya and ttyb 187
Transmit Data Path 188
Receive Data Path 188
24.8. EEPROM and EPROM 188
EEPROM 190
EPROM 190
24.9 Time of Day (TOD) Clock 190
TOD Oscillator Circuit 190

24

MOS Bus Devices

Seven MOS I/O devices share an 8-bit bus amongst themselves. These MOS devices are:

- User-accessible EAROM (EEPROM)
- Boot PROM (EPROM)
- Real Time Clock (RTC)
- Serial Ports (Port A and Port B)
- Mouse
- Keyboard

MOS devices are accessed and controlled through a series of PALs and a read/write decoder, shown on page 9 of the schematics.

MOS devices occupy TYPE1 space, and are located at:

Table 24-1 *Map for TYPE1 Space*

Device	Address bits	Address
KEYBDMOUSE	/p2_a20*/p2_a19*/p2_a18*/p2_a17	0x000000
SERIALIO	/p2_a20*/p2_a19*/p2_a18* p2_a17	0x020000
EEPROM	/p2_a20*/p2_a19* p2_a18*/p2_a17	0x040000
TOD	/p2_a20*/p2_a19* p2_a18* p2_a17	0x060000
EPROM	p2_a20*/p2_a19*/p2_a18*/p2_a17	0x100000

You can access the MOS bus using one of three cycles:

1. MOS read cycle, or
2. MOS write cycle, or
3. Diagnostic cycle.

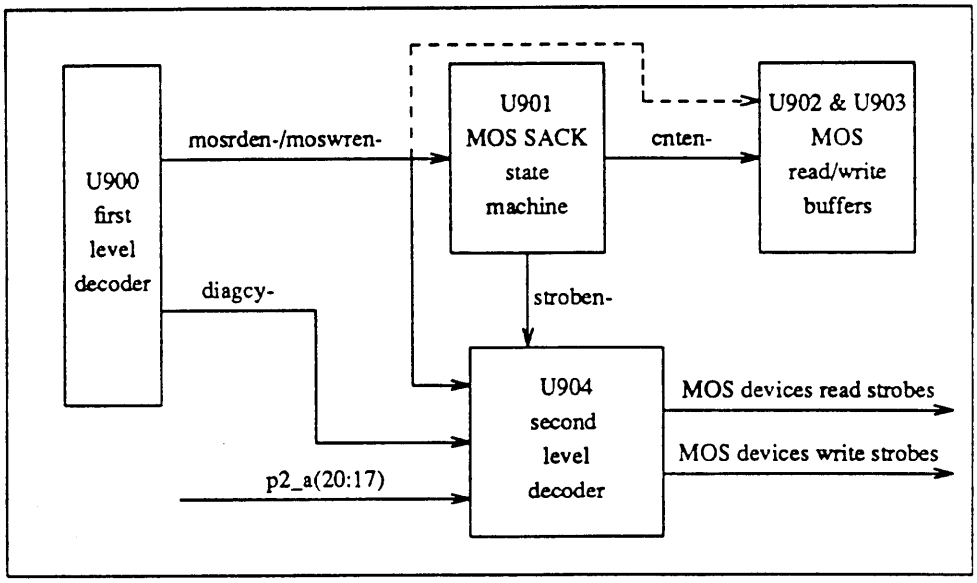
Each of these accesses are defined by an output from the first-level MOS decoder, U900. The diagnostic cycle bypasses the MMU (allowing you to access a terminal even with a bad MMU).

MOS devices are physically accessed through:

- a MOS enable PAL, U900, which generates the MOS read, MOS write, and the diagnostic cycle signals,
- a MOS SACK (synchronized acknowledge) state machine, U901, which generates MSACK (SACK for the MOS devices), strobe enable, and count enable signals,
- and the MOS read/write strobe decoder, U904, which issues read/write strobes to the individual MOS devices.

The figure below illustrates this:

Figure 24-1 *MOS Decoders*



24.1. U900 MOS Enables PAL

U900 is the first-level decoder for the MOS devices. Its outputs function as follows:

- moswren: this low-active signal enables the write data buffers (U902) for the MOS devices. The DCP is not included. This signal also starts the dtack state machine and indicates to U904 whether you are in a read or a write cycle.

NOTE *A write to the EPROM is available to enable diagnostics to check the statistics bits on a write to TYPE1 space without actually affecting any devices.*

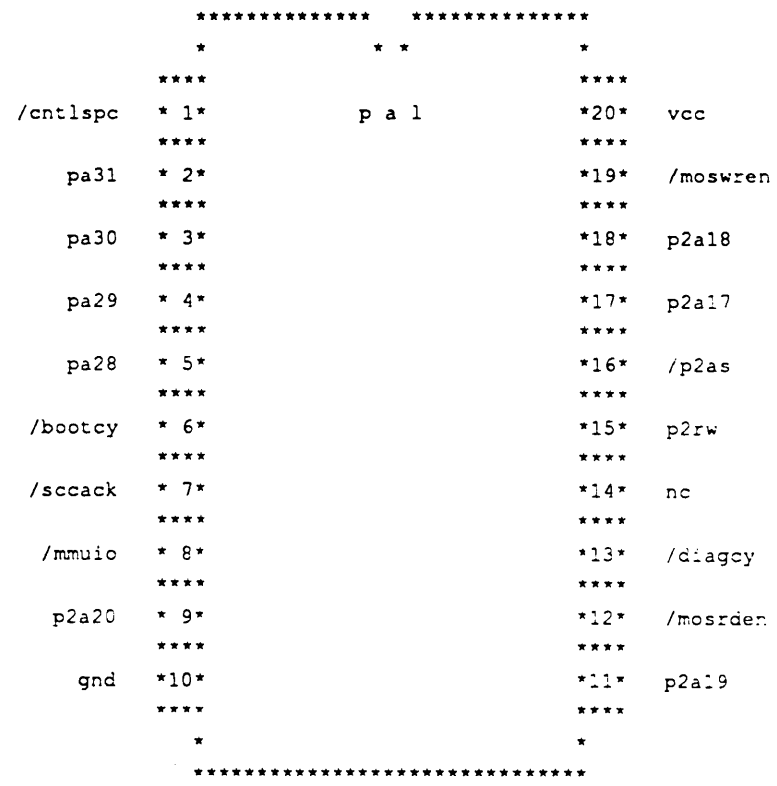
- mosrden: this low active signal enables the read output buffers (U903) to drive the P2 data bus for all MOS devices. This signal also starts the dtack state machine and indicates to U904 whether you are in a read or a write cycle.
- diagcy: special serial port SCC access indicator. Diagnostics cycle mode bypasses the MMU logic so that diagnostics can get access to a terminal with an inoperative MMU.

NOTE *The signal sccack (serial controller acknowledge) is included due to a three OR-term limit in the rd_serial output of U904.*

U900 Pinout

Pinout of the U900 high-level MOS decoder PAL is:

Figure 24-2 *U900 Pinout*



MOS read or MOS write is set by the state of the P2 read/write signal, p2_rw. A low indicates a write cycle; high indicates a read cycle. When the mmuio- signal on pin 8 of U900 is pulled low (active) the P2 address bits A20-A17 will decode to a TYPE1 space (indicating an I/O device) MOS device. When p_a(31:28) are all high and the Control space signal, ctlspc-, is active, this indicates a diagnostic cycle (bypassing the MMU).

U900 MOS Write Enable — moswren

When the mmuio- signal on pin 8 of U900 is pulled low (active) the P2 address bits A20-A17 will decode to a MOS device. The state of the p2_rw signal determines whether this will be a read cycle or a write cycle. Remember that:

```
p2_rw = high = read cycle

p2_rw = low = write cycle
```

```
moswren = /p2rw * p2as * mmuio * /p2a20 * /p2a19 +              ; writable mos
          /p2rw * p2as * cntlspc * pa31 * pa30 * pa29 * pa28 + ; diagcycle
          /p2rw * p2as * mmuio * p2a20*/p2a19*/p2a18*/p2a17    ; for diagnostic
                                                               ; EPROM write
```

Inputs to U900 decoding to one of the following three events will force a MOS write cycle:

1. a MOS device write cycle — P2 read/write signal is low (indicating a write cycle), low active p2_as (address strobe) is asserted, low active MMUIO signal is asserted, and the two high order address bits for TYPE1 space, A20 and A19, are held low, indicating a TYPE1 device (keyboard, mouse, serial I/O port, EEPROM, or time of day clock) is going to be selected, or
2. a Control space diagnostic cycle — P2 read/write signal is low (indicating a write cycle), low active p2_as (address strobe) is asserted, low active Control space (cntlspc-) is asserted, and address bits A28-A31 are high, or
3. a diagnostic write cycle to the EPROM — a write to the EPROM is essentially a write to a bit bucket. The diagnostics write cycle enables diagnostics to check statistics logic without actually affecting any devices.

U900 MOS Read Enable — mosrden

```
mosrden = p2rw * p2as * mmuio * /p2a20 * /p2a19 +              ; readable mos
          p2rw * p2as * mmuio * p2a20*/p2a19*/p2a18*/p2a17 +   ; eprom
          p2rw * p2as * cntlspc * pa31 * pa30 * pa29 * pa28 +  ; diagcycle
          p2rw * p2as * bootcy +                               ; boot eprom
          p2rw * p2as * sccack                                 ; scc vect'd int
```

mosrden- is issued on pin 12 of U900 following one of five input events:

1. a MOS device read cycle — P2 read/write is high, indicating a read cycle; P2 address strobe is low (active); mmuio- is low (active); P2 address bits A20 and A19 are low (indicating one of the four devices: keyboard, mouse, TOD, or EEPROM, in TYPE1 space); or
2. the EPROM is going to be read — P2 read/write is high, indicating a read cycle; P2 address strobe is low (active); mmuio- is low (active); P2 address bits A20-A17 select the EPROM at 0x100000, or

3. a diagnostic cycle read — P2 read/write is high, indicating a read cycle; P2 address strobe is low (active); a Control space (as opposed to MMU I/O) access is indicated by the low active assertion of ctlspc-; and address bits A31-A28 are all high, indicating that the MMU will be bypassed, allowing direct diagnostic access of the serial controllers, or
4. a boot PROM read cycle — P2 read/write is high, indicating a read cycle; P2 address strobe is low (active); and the low active signal bootcy- (decoded from U106 PAL) is asserted, or
5. read of the serial controller's vectored interrupt — P2 read/write is high, indicating a read cycle; P2 address strobe is low (active); and sccack- (serial controller acknowledge) is issued from U304 PAL. The serial controllers' interrupts are daisy-chained; once the interrupt is acknowledged by the assertion of sccack-, the software is supplied a vector by the interrupting controller.

Diagnostic Cycle — diagcy

```
diagcy  = p2as * cntlspc * pa31 * pa30 * pa29 * pa28 +        ; diag scc access
          p2rw * p2as * sccack                                ; scc vect'd int
```

diagcy- is issued on pin 13 of U900 by one of the following two events:

1. serial controller access bypassing the MMU — P2 address strobe is asserted; control space (ctlspc-) is asserted; and address bits A31-A28 are set (indicating the UART bypass), or
2. read of the SCC vectored interrupt — P2 read asserted (p2_rw is high); P2 address strobe is asserted, and sccack- is asserted. It is ORed in U900 instead of U904 because the latter had a three-term OR limit, and they were all used.

24.2. U901 MOS SACK State Machine

U901 MOS SACK state machine supplies three signals:

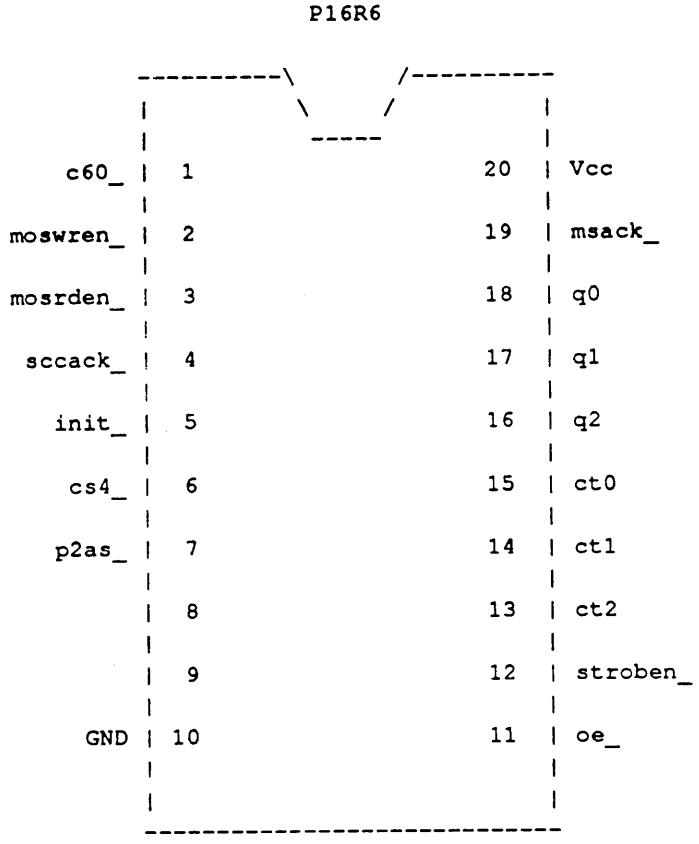
1. MOS synchronous acknowledge, msack-, to the U204 DSACK PAL,
2. clock, cnten-, to the U903 MOS read buffer, and
3. strobe enable, stroben-, to the U904 MOS read/write strobe decoder.

U901 uses moswren-, mosrden-, and sccack- to start the state machine to issue stroben-, cnten-, and msack- at the appropriate times.

Pinout of U901 PAL

Pinout of the U901 State Machine is:

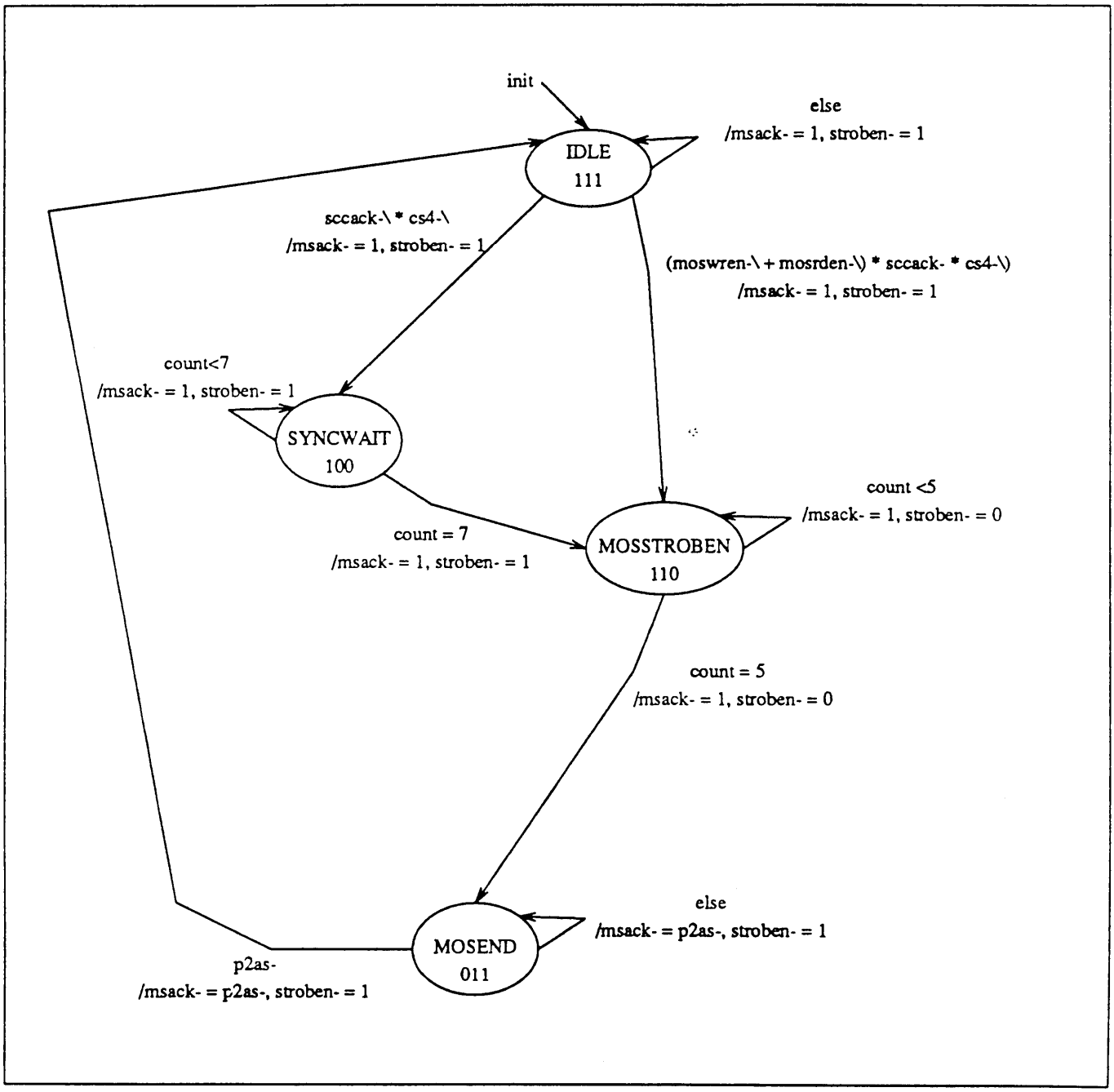
Figure 24-3 *U901 MOS Control State Machine Pinout*



NOTE *For the following explanation, see the MOS Bus read cycle timing diagram in the appendix.*

The following figure illustrates the MOS control state machine.

Figure 24-4 *MOS Control State Machine*



There are two paths through the MOS control state machine:

1. MOS read/write, and
2. SCC interrupt.

U901 MOS Read/Write Control

The MOS read/write path starts in the IDLE state. IDLE state (111) is defined by a control state counter, which in turn influences a wait state counter. Every time the least significant bit (q0, also known as cnten — counter enable) of the control state counter goes from a one to a zero, the wait state counter is enabled and begins incrementing. Both of these counters are internal to the U901 state machine. The states in each of these counters are defined in the following two tables:

Table 24-2 *U901 Control Counter States*

Control State	Counter Value		
	q2	q1	q0
idle	1	1	1
syncwait	1	0	0
mosstroben	1	1	0
mosend	0	1	1
nostate0	0	0	0
nostate1	0	0	1
nostate2	0	1	0
nostate5	1	0	1

Table 24-3 *U901 Wait Counter States*

Control State	Counter Value		
	ct2	ct1	ct0
cnt0	0	0	0
cnt1	0	0	1
cnt2	0	1	0
cnt3	0	1	1
cnt4	1	0	0
cnt5	1	0	1
cnt6	1	1	0
cnt7	1	1	1

Note that some of the states in the control state counter are labelled "nostates." These are values in the control state counter which do not have control states assigned to them: they are defined in case the counter should happen to hold one of these stateless values when you power up. For instance, as shown below, a control state register value of 000 is defined as nostate0.

```
Assign control states

...
nostate0    =  000
nostate1    =  001
nostate2    =  010
nostate5    =  101
```

If you *should* happen to power up in one of these "nostates," the state machine will go immediately to IDLE. For instance, powering up in nostate0 will cause you to go to IDLE state and force the outputs msack- and stroben- to their inactive (high) state because of the following statements:

```
state  nostate0:   msack_ = h;
                   stroben_ = h;

                   goto idle;
```

```
state   nostate1:       msack_ = h;
                        stroben_ = h;

                        goto idle;
```

```
state  nostate2:   msack_ = h;
                   stroben_ = h;

                   goto idle;
```

```
state  nostate5:   msack_ = h;
                   stroben_ = h;

                   goto idle;
```

U901 is set to the IDLE state at initialization, and the control counter is set to 111. If you look at the MOS read timing diagram, you will see that msack- and stroben- are false (high) during IDLE. This is also illustrated in the state diagram equation:

```
state  idle:       msack_ = h;
                   stroben_ = h;
```

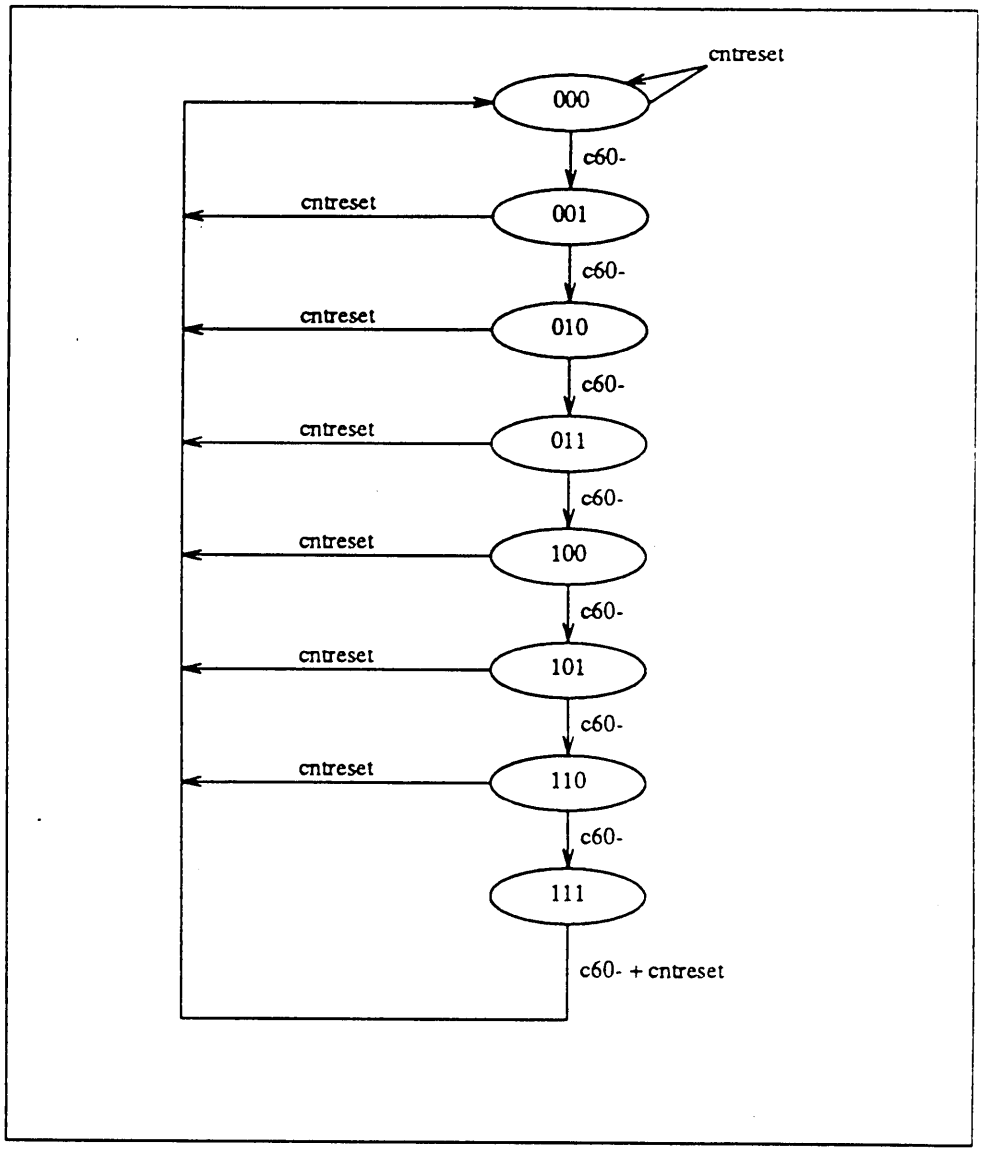
To go from the IDLE state to MOS strobe enable (MOSSTROBEN) state either MOS read or MOS write must go true (low), scc_ack- stay high, and you must be

in state 4 (cs4- asserted). The other path, to SYNCWAIT, is selected if both sccack- and cs4- are asserted (low). Otherwise you remain in IDLE state.

```
state_diagram  [q2,q1,q0] "control state machine"

state  idle:   msack_ = h;
               stroben_ = h;

          if !sccack_&!cs4_ then syncwait
           else
            if (!moswren_#!mosrden_)&!cs4_&sccack_
             then mosstroben
              else
               idle;
```

In MOSSTROBEN state the control state counter goes from 111 (IDLE) to 110 (MOSSTROBEN). Remember, when the least significant bit (q0) of the control state counter goes from one to zero it enables the wait state counter (ct2-ct0) to begin incrementing, unless or until the signal cnten- goes high, resetting this counter. Once in MOSSTROBEN state the wait state counter increments every c60- (60 nsec) clock.

Figure 24-5 *MOS DTACK PAL Wait State Counter*



As long as the wait state counter is less than 5 (101), U901 remains in MOSSTROBEN state. When the count reaches five, 540 nsecs into the MOS read/write cycle, U901 enters MOSEND state. This is designated by 011 in the control state counter.

```
state   mosstroben:     msack_ = h;
                        stroben_ = l;

                if !init_ then idle
                 else
                  if count == 5 then mosend
                   else
                    mosstroben;
```

```
state   mosend:         msack_ = p2as_;
                        stroben_ = h;

                if !init_ then idle
                 else
                  if p2as_ then idle
                   else
                    mosend;
```

Recall that the least significant bit (LSB) of the control state counter enables and disables the wait state counter. When U901 enters MOSEND state, the LSB (q0) goes to a one, (control state counter goes from 110 to 011) which disables cnten-, and the wait state counter is reset to 000 on the next clock. This counter remains reset until cnten- goes low again.

The stroben- signal goes high, disabling it, and msack- goes low, pacing the p2as- address strobe. When the p2as- signal goes high msack- also goes high, still pacing p2as-. On the next clock you reenter IDLE state.

If you look at the timing diagram, MOS bus cycle begins in the IDLE state with both msack- and stroben- high. IDLE state continues until 180 nsecs into the cycle, S5 of c60 clock. MOSSTROBEN state continues until count = 5, 540 nsecs into the cycle, at which time U901 enters MOSEND state. When msack- goes low, the processor begins to end the bus cycle. When p2_as- goes high (end of the bus cycle) the IDLE state is entered on the next clock (720 nsec) and remains in IDLE.

U901 SCC Interrupt — SYNCWAIT State

The other path through the state machine is that of SYNCWAIT, the interrupt acknowledge cycle. SYNCWAIT is necessary because of the long set-up times needed for MOS chips; it's essentially a lot of wait states inserted into an interrupt cycle.

SYNCWAIT state is entered whenever sccack- and cs4- are asserted.

```
state   idle:   msack_ = h;
                stroben_ = h;

         if !sccack_&!cs4_ then syncwait
          else
           if (!moswren_#!mosrden_)&!cs4_&sccack_
            then mosstroben
             else
              idle;
```

When sccack- and cs4- are asserted, msack- and stroben- are still deasserted. The control state counter goes from 111 (IDLE) to 100 (SYNCWAIT). The least significant bit (q0) of the control state counter goes from a one to a zero (counter goes from 111 to 100) and this enables the wait state counter, which begins to increment. As long as the wait counter is less than seven (111), U901 remains in SYNCWAIT state; as soon as the wait state counter reaches seven U901 enters MOSSTROBEN state, a MOS read/write cycle.

```
state   syncwait:  msack_ = h;
                   stroben_ = h;

          if !init_ then idle
           else
            if count == 7 then mosstroben
             else
              syncwait;
```

These seven wait states allow

1. synchronization to PCLK baud rate clock on the serial communication controller (205 nsecs),
2. the daisy-chained interrupt enable output from the first SCC chip to get out of the chip (250 nsecs), and
3. the interrupt enable input to set up in the next chip (100 nsecs).

In other words, these seven wait states give the two SCCs time to decide which of the SCCs will issue the interrupt. At this point the read cycle starts (enter MOSSTROBEN state) to retrieve the interrupt vector. The wait state counter simply rolls over from 111 to 000.

MOSSTROBEN state operates the same in SYNC as in MOS read/write cycles. The control state counter's LSB remains a zero (110), enabling the wait state counter, which continues to increment. As long as the wait state counter is less than five (101), U901 remains in MOSSTROBEN state. When the count reaches five, 1020 nsecs into the SYNC cycle, U901 enters MOSEND state. This is designated by 011 in the control state counter.

```
state   mosstroben:  msack_ = h;
                     stroben_ = l;

         if !init_ then idle
          else
           if count == 5 then mosend
            else
             mosstroben;
```

```
state   mosend:    msack_ = p2as_;
                   stroben_ = h;

         if !init_ then idle
          else
           if p2as_ then idle
            else
             mosend;
```

When U901 enters MOSEND state the control state counter goes from 110 to 011. When the LSB (q0) goes to a one cnten- is disabled and the wait state counter is reset to 000. This counter remains reset until cnten- goes low again. 1030 nsecs into the cycle stroben- goes high, disabling it, and msack- goes low. 1080 nsecs into the cycle the wait state counter is reset to zero (000). When p2_as address strobe goes high, you reenter IDLE state on the next clock.

The timing for the states is:

```
IDLE         0 to 180 nsecs
SYNCWAIT     until 660 nsecs
MOSSTROBEN   until 1020 nsecs
MOSEND       until 1200 nsecs
```

(at which time you go back to idle)

Thus, the interrupt acknowledge cycle for the MOS devices is 1200 nsecs long.

MOS devices have very slow output disable times. A write cycle to a MOS device following a read cycle will cause a buffer conflict unless time is allowed at the end of a read cycle for the outputs to be cleared. The 8530 SCC gives one solution to this problem — a status register read cycle will not provide stable data for the length of the read cycle. To avoid unknown metastable conditions in the 68020, the read data should be latched and time allowed for the data to settle. Latching the data for all MOS reads allows the MOS read strobes to deactivate early, allowing time for the buffers to turn off before the next cycle and also provides stable data to the 68020.

MOS devices also have data and address hold time requirements which require the write strobes to terminate before the end of the cycle.

The stroben- signal is used for providing the short R/W strobes for hold time requirements and buffer turn-off time. In addition, the rising edge of cnten- is used to clock the 74LS374 data hold latch (U1203).

24.3. U904 MOS Read/Write Strobe Decoder

The U904 PAL generates the read and write strobes for all the MOS devices except the DCP. The devices for which strobes are generated are:

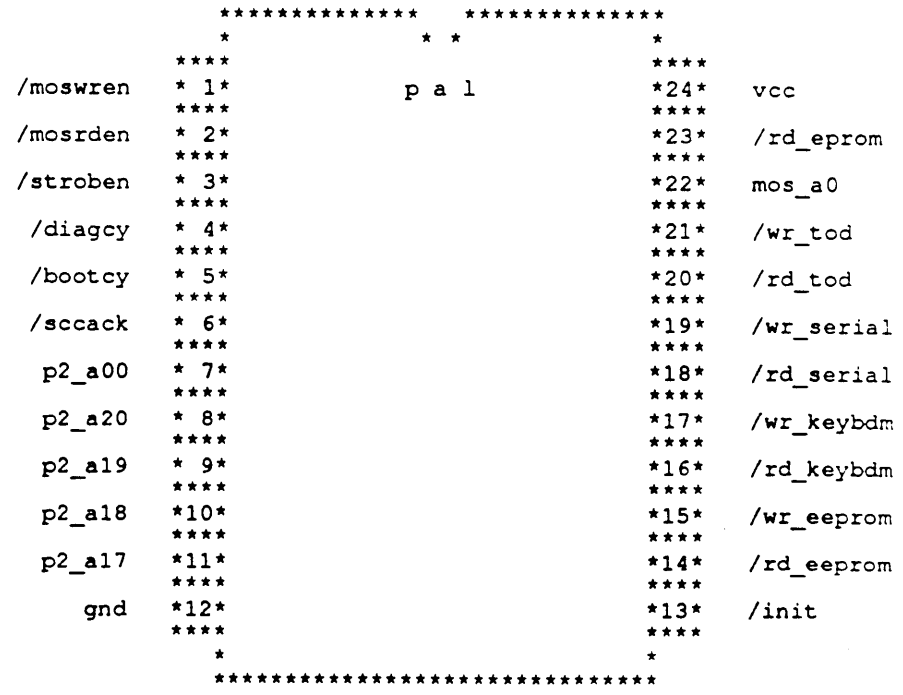
- read/write Time of Day clock
- read/write serial ports
- read/write keyboard and mouse
- read/write EEPROM, and
- read EPROM.

The read strobes for the 8530 SCC devices are active for normal reads and interrupt acknowledge cycles. In addition to normal accesses, the serial port SCC read and write strobes are active during a diagnostic cycle. A diagnostic cycle access bypasses the MMU (is mapped into Control space as "UART Bypass") so that diagnostic programs can get access to a serial port while using a minimum of hardware.

U904 Pinout

Pinout for U904 is:

Figure 24-6 *U904 Pinout*



U904 Input Signals

Description of the input signals:

- /moswren: indicates a valid TYPE1 space write cycle
- /mosrden: indicates a valid TYPE1 space read cycle
- /stroben: enable read/write strobes; avoids buffer conflicts
- p2_a<20:17>: physical address from MMU which decodes to a specific MOS device — see the table which defines TYPE1 space devices, earlier.
- p2_a00: physical address from MMU
- /bootcy: doing special virtual address boot cycle
- /diagcy: doing special virtual address diag cycle or sccack
- /sccack: serial port and keyboard/mouse interrupt acknowledge
- /init: resets the SCC's

U904 Output Signals

Description of the output signals:

- /rd_eprom: read EPROM (boot)
- /rd_eeprom: read EEPROM
- /wr_eeprom: write EEPROM
- /rd_keybdm: read keyboard/mouse scc
- /wr_keybdm: write keyboard/mouse scc
- /rd_serial: read tty[ab] scc
- /wr_serial: write tty[ab] scc
- /rd_tod: read TOD chip
- /wr_tod: write TOD chip
- mos_a0: address bit A0 on the MOS bus is gated to reduce noise injection into 7170 TOD chip. If this line is not gated, transitions on the address bus are injected into the TOD clock, making it run fast. For more information, see the section on the 7170 TOD clock chip.

Three macros are used in the output equations:

```
#define VALIDRD = mosrden*stroben*/reset

#define VALIDWT = moswren*stroben*/reset

#define IO      = /bootcy*/diagcy
```

These define:

1. a valid read cycle as being an active MOS read enable, an active MOS strobe enable, and init (reset) inactive;
2. a valid write cycle as being an active MOS write enable, an active MOS strobe enable, and init (reset) inactive;
3. a valid I/O cycle as being neither a boot cycle nor a diagnostics cycle. In other words, a valid I/O cycle does *not* bypass the MMU.

Decodes for the output signals are:

```
rd_eprom  =       VALIDRD*EPROM*IO +
                        bootcy*VALIDRD
```

This equation explains how to get access for a read of the EPROM. To access it, you must be in a valid read cycle (defined above), address bits A20:17 must decode to the EPROM in TYPE1 space —

```
EPROM = p2_a20*/p2_a19*/p2_a18*/p2_a17 = 0x100000
```

and you must be in an I/O cycle, *or* you must be in a bootcycle with a valid read cycle.

The remainder of the outputs are defined below.

```
rd_eeprom =       VALIDRD*EEPROM*IO
```

To read the EEPROM, you must be in a valid read, have the correct A20:17 address bits for TYPE1 space decode to the EEPROM —

```
EEPROM = /p2_a20*/p2_a19*p2_a18*/p2_a17 = 0x040000
```

and you must be in an I/O cycle.

```
wr_eeprom =       VALIDWT*EEPROM*IO
```

The only difference here is that you are in a valid write cycle.

NOTE *Notice the reset term in the following serial equations; they cause both the read and write strobes to be active simultaneously, which resets the SCCs.*

```
rd_keybdm =        VALIDRD*KEYBDMOUSE*IO +
                          sccack*VALIDRD +
                          reset
```

To read the keyboard or mouse you must be in a valid read cycle, have the TYPE1 space address for the keyboard/mouse —

```
KEYBDMOUSE = /p2_a20*/p2_a19*/p2_a18*/p2_a17  = 0x000000
```

and you must be in a valid I/O cycle. *Or* you can access it through an interrupt acknowldge cycle (sccack and valid read of the interrupt vector).

```
wr_keybdm =        VALIDWT*KEYBDMOUSE*IO +
                          reset
```

A write to the keyboard/mouse is similar, except that there is no valid write during an interrupt acknowledge cycle.

```
rd_serial =        VALIDRD*SERIALIO*IO +
                          VALIDRD*diagcy +
                          reset
```

To read the serial I/O port you must have a valid read, the TYPE1 address for the serial I/O port —

```
SERIALIO = /p2_a20*/p2_a19*/p2_a18* p2_a17 = 0x020000
```

and a valid I/O cycle. *Or* you can access it during a reset (init- asserted) cycle. *Or* you can access it during a valid read and a diagnostic cycle. If you remember back to U900, a diagnostic cycle is defined either as a control space access (bypassing the MMU) or read of the vectored interrupt.

```
wr_serial =       VALIDWT*SERIALIO*IO +
                        VALIDWT*diagcy +
                        reset
```

A write to the serial ports is the same as a read, except that a valid write is asserted in place of valid read.

```
rd_tod    =       VALIDRD*TOD*IO
```

Notice that the TOD clock can only be accessed through the MMU — no diagnostic cycle.

```
wr_tod    =       VALIDWT*TOD*IO
```

```
/mos_a0   =       mosrden*TOD*IO*/p2_a00 +
                  moswren*TOD*IO*/p2_a00
```

The A0 bit is gated through U904 to reduce the number of transitions coupled to the TOD clock. Transitions coupled to the oscillator circuit cause the TOD clock to run fast; to control this, mos_a0 is gated through U904 only when there is a TOD access, which is rare enough not to affect the TOD clock.

As shown above, the /mos_a0 bit is asserted during either a MOS read or a MOS write, and when the TYPE1 address for the TOD clock is selected:

```
TOD = /p2_a20*/p2_a19* p2_a18* p2_a17 = 0x060000
```

when the MOS circuitry is in an I/O cycle, and P2 address bit A0 is low.

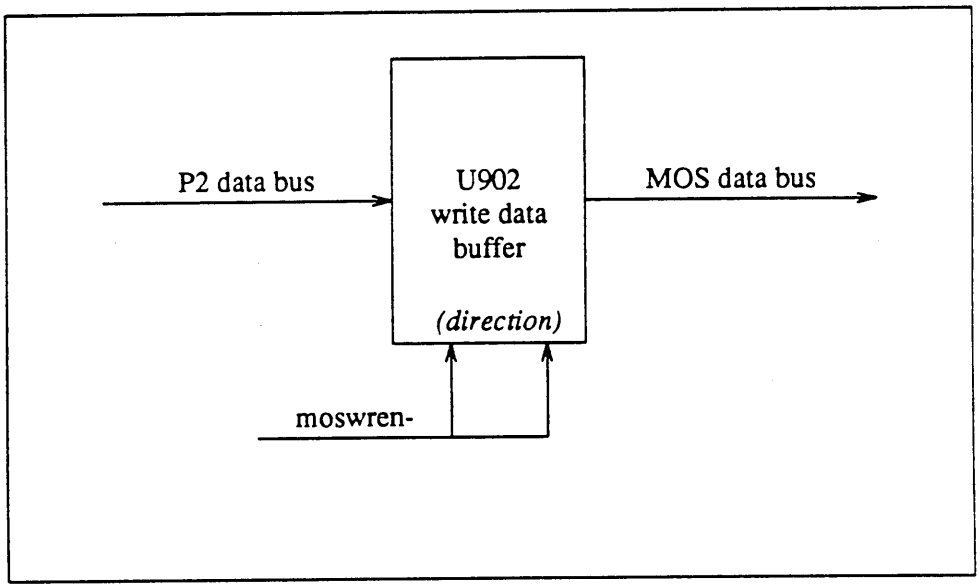
24.4. U902 MOS Write and U903 MOS Read Buffers

The U902 write buffer and U903 read buffer couple data bidirectionally from the processor bus (P2 data) to the MOS data bus.

U902 MOS Write Buffer

U902 write buffer is a HCMOS chip used to minimize undershoot to its connected MOS devices. When moswren- (from U900) goes active, data from the P2 data bus is coupled to the MOS data bus.

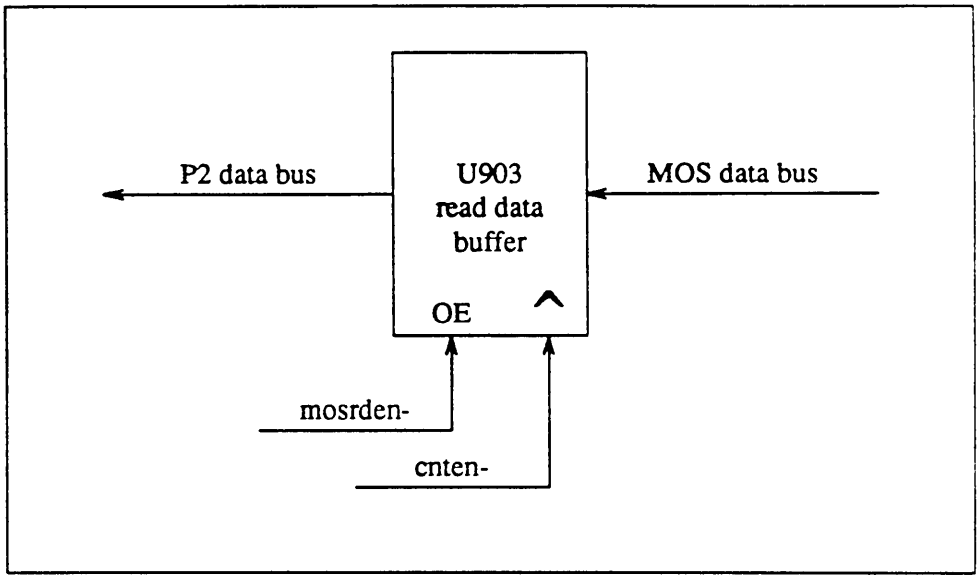
Figure 24-7 *U902 MOS Write Data Buffer*



U903 MOS Read Buffer

On the rising edge of cnten-, data from the MOS bus (m_d[7:0]) are latched onto the P2 data bus. The cnten- signal is also the LSB (q0) of the control state counter in U904. This rising edge — going from a binary zero to a binary one — of cnten- also signals the transition from MOSSTROBEN (110) state to MOSEND (011) state.

Figure 24-8 *U903 MOS Read Data Buffer*



If you look at the timing diagram for MOS Bus Read in the appendix, you will see that cnten (q0) goes high at 545 nsecs, and then mosreden- at 667 nsecs. The delay between cnten- and mosrden- is intentional, for two reasons:

1. MOS devices have very slow output disable times. A write cycle immediately following a read cycle could conceivably result in a buffer conflict — the read cycle data may not have been cleared from the MOS device's output. This delay between clocking data into the U903 buffer and enabling it out is inserted into the timing cycle to make certain that the MOS device has had time to clear itself of any read data.
2. When you do a status read from any of the 8530 serial communication controllers, the data is not guaranteed to be stable in the SCC's status register for the entire length of the MOS read cycle. This is because status registers in the SCCs are updated asynchronously and there is no provision for latching this data inside the SCC.

To avoid metastable conditions and allow enough time for the MOS device to turn off after data is read, a delay of a little over 100 nsecs between data latch and output enable is built into the timing for U903.

24.5. MOS Read and Write Cycles

Both the read and the write cycles use much of the same timing; the major difference is that the write cycle uses U902 data buffers and the read cycle uses U903 data buffers. U902 is enabled by the write line; U903 is enabled by the read line.

MOS Read Cycle

The MOS read cycle starts out in IDLE state, with the control state counter in U904 set to 111. Both msack- and stroben- are high (false); mosrden- goes true and moswren- stays high (false).

Whenever mosrden- or moswren- go low, *and* cs4- is true *and* sccack remains false (we are not doing an interrupt acknowledge cycle), the state machine moves from IDLE to MOSSTROBEN. Both msack- and stroben- remain high.

In MOSSTROBEN state, the three-bit wait state counter begins to increment from 000, using 60 nsec clock (c60). The msack- signal remains false, but stroben- goes low around 220 nsecs into the cycle.

When the wait state counter reaches five (101), the state machine moves from MOSSTROBEN to MOSEND state; msack- still remains false (high) and stroben- remains true.

In MOSEND state, data is clocked into the U903 read buffer by cnten- going high (false) at 545 nsecs into the read cycle. At the same time cnten- going from a zero to a one disables the U904's wait state counter. 580 nsecs into the cycle, mosend- goes low (true).

In MOSEND state, msack- paces p2_as-, going low at 580 nsecs, and stroben- goes high at this same time.

When the address strobe p2_as- goes high, the read cycle reenters IDLE state.

MOS Write Cycle

The MOS write cycle is much like the MOS read cycle. The MOS write cycle starts out in IDLE state, with the control state counter in U904 set to 111. Both msack- and stroben- are high (false). When mosrden- stays false (high) and moswren- goes low (true) a write cycle is signalled; cs4- dropped at 120 nsecs and ccack- stays high (false) since this is not an interrupt acknowledge cycle. The write cycle enters MOSSTROBEN state.

At this time msack- remains high but stroben- drops low, causing the appropriate MOS write strobes to be issued from U904. The wait state counter is enabled and begins incrementing every 60 nsecs (c60 clock) from an initial count of 000. When it reaches five (101) the write cycle enters MOSEND state. When p2_as- goes high, the cycle ends and the MOS circuitry returns to IDLE state.

24.6. Mouse and Keyboard SCC

The mouse and keyboard ports are both enclosed in the U1000 8530 serial communications controller.

1. serial I/O for the keyboard is on SCC port A;
2. serial I/O for the mouse is on SCC port B.

Inputs to the U1000 SCC are:

- m_d[7:0] — MOS data bus bits 0-7
- /iei_scc — interrupt enable input, daisy-chained from the serial port SCC.
- /scc_ack — SCC acknowledge used in the interrupt acknowledge cycle
- mos_a1:0 — MOS bus address bits, terminated with serial resistors R901 and R902. These come from an F244 on the other side of the board and are

susceptible to undershoot; R901:2 are added to compensate.

- /rd_keybdm, /wr_keybdm — read/write.

U405 and U2207 Baud Rate Clock

U405 19.8608 MHz crystal and U2207 divider make up the baud rate clock for the U1000 UART. It also provides 100 nsec clock (c100) to the refresh period counter on page 24.

J1001 is normally IN, connecting the clock crystal to the UART. 200 nsec clock (c200) from the divide-by-four (QB) output of the U2207 divider/counter is connected to the PCLK input of U1000 UART.

Transmit Data Path

Transmit data enters the parallel data port (D7:0) of U1000 from the MOS data bus, bits m_d[7:0]. Data comes out serially on pin 15 (keyboard) and pin 25 (mouse). TX data is inverted through U1104 to set it to the correct polarity, fed through U1003 driver to J1000 DB-15 Mouse/Keyboard connector. TX data for the keyboard is connected to pin 3 (txkeybd); TX data for the mouse is connected to pin 7 (txmouse).

Notice that U1003 driver has VEE grounded. This driver normally operates at +/- 5 VDC; grounding VEE makes the LS29 act like a normal TTL driver, operating between 0 and +5 VDC.

C1003:2 47 pF caps are connected across the mouse and keyboard data outputs to slow the transmit signal's edge rate to between 1 and 2 microseconds; this reduces radiation.

Receive Data Path

RX data enters the 2060 board on pin 1 (keyboard) and pin 5 (mouse) of J1000. Receive data lines are linked to 4.7 KΩ pullup resistors (necessary because the mouse uses open collector circuitry) and then connect to U1002 differential receiver with hysteresis. The internal hysteresis circuitry provides noise immunity, and R1000 and R1001 bias network moves the input threshhold up to 1.5 VDC, also as a protection against the effects of noise. Threshhold voltage comes in on the plus inputs (A+ and D+) of U1002; the RX data comes in on the minus inputs (A- and D-). This causes the receiver to act as an inverter.

Keyboard data of corrected polarity is connected to port A of the SCC (RXDA, pin 13); mouse data is connected to port B (RXDB, pin 27). From there it is converted from serial to parallel data format inside the SCC, and put out on the MOS data bus, m_d[7:0].

24.7. Serial Ports A and B — ttya and ttyb

Serial Ports A and B use an 8530 SCC identical to the one used by the keyboard and mouse. The inputs to the SCC are nearly the same as those for the keyboard/mouse SCC — with the exception, of course, that read and write serial signals are used instead of their keyboard/mouse correspondents. Also many of the modem/terminal control signals are used in the serial port circuit.

Transmit Data Path

Transmit data enters the parallel data port (D7:0) of U1100 from the MOS data bus, bits m_d[7:0]. Data comes out serially on pin 15 (port A) and pin 25 (port B). TX data is inverted through U1105:4 inverters to put the two signals in the correct polarity. Next the data is fed through U1107:6 drivers to J1100 DB-15 Port A connector and J1101 Port B connector. TX data for the port A is connected to pin 2 of J1100; TX data for port B is connected to pin 2 of J1101.

Notice that U1007:6 drivers do *not* have VEE grounded (as U1003 was). This driver operates at +/- 12 VDC and thus supplies valid RS-423 signal levels.

C1007:0 47 pF caps are connected across the output signal lines to slow these signals' edge rate to between 1 and 2 microseconds; this reduces radiation.

Serial port control signals (request to send, data terminal ready) are supplied by the SCC to both serial ports also.

Receive Data Path

RX data and control signals enter the 2060 board on the serial port connectors, J1100 and J1101. The data and control lines are linked to U1003:1 differential receivers with hysteresis; this internal hysteresis circuitry provides noise immunity. The plus inputs are connected to ground and the signals enter the minus inputs; this inverts the signals.

NOTE *This inversion means that the control signals are, by default, ACTIVE!*

The 4.7 KΩ resistors connected serially to the synch (SYNA and SYNB) and TXCA/TXCB transmit clock lines are used as current limiters.

Serial port A data of corrected polarity is connected to port A of the SCC (RXDA, pin 13); serial port B data is connected to port B (RXDB, pin 27). It is then converted from serial to parallel format inside the SCC, and put out on the MOS data bus, m_d[7:0].

The signal -5vr (negative 5 volts regulated) is supplied at pin 25 of both connectors for use by selected modems.

24.8. EEPROM and EPROM

The EEPROM and EPROM reside in TYPE1 space and are connected to the MOS bus. Their respective addresses are:

```
EEPROM = /p2_a20*/p2_a19* p2_a18*/p2_a17 = 0x040000

EPROM  = p2_a20*/p2_a19*/p2_a18*/p2_a17  = 0x100000
```

The EEPROM can be both read and written; the EPROM is read-only. Thus strobes to the EEPROM and rd_eeprom and wr_eeprom; strobe to the EPROM is rd_eprom. If you recall, these three strobes were among those issued by U904; the rd_eprom strobe was defined:

```
rd_eprom = (mosrden*stroben*/reset*p2_a20*/p2_a19*/p2_a18*/p2_a17*/bootcy*/diagcy) +
       (bootcy*mosrden*stroben*/reset)
```

In other words, a read EPROM cycle can be made when

1. MOS read is enabled, AND
2. strobe enable is true, AND
3. it is NOT a reset cycle, AND
4. P2 address bit A20 is high, AND
5. P2 A19 is low, AND
6. P2 A18 is low, AND
7. P2 A17 is low, AND
8. it is NOT a boot cycle, AND
9. it is NOT a diagnostic cycle.

Or, a read of the EPROM can be made when

1. it's a boot cycle, AND
2. a MOS read cycle, AND
3. strobe enable is true, AND
4. it's not a reset cycle.

Read of and write to the EEPROM are defined similarly:

```
rd_eeprom = mosrden*stroben*/reset*/p2_a20*/p2_a19*p2_a18*/p2_a17*/bootcy*/diagcy
```

```
wr_eeprom = moswren*stroben*/reset*/p2_a20*/p2_a19*p2_a18*/p2_a17*/bootcy*/diagcy
```

Both the EEPROM and the EPROM use addresses directly from the processor (p_a addresses, virtual addresses) and not those off the MOS bus. This is because the EEPROM and EPROM are physically located close to the processor, and are susceptible to undershoot (as all MOS devices are). Since the 68020 closely controls undershoot, the processor's address lines are used.

EEPROM

The EEPROM is a read/write device, organized into 2K 8-bit bytes. Eleven bits of processor address (p_a[10:00]) select a byte from within this 2K space.

Chip write enable is controlled by the signal wr_eeprom-; when the signal is low, data on the MOS bus (m_d[7:0]) is written into the chip at the byte address selected by p_a[10:00]. When wr_eeprom- is high, write to the chip is disabled.

Chip output enable (EEPROM read) is controlled by the signal rd_eeprom-; when the signal is low, data in the chip is written to the MOS data bus (m_d[7:0]). When rd_eeprom- is high, output from the chip is disabled.

The 180Ω series termination resistors on the read and write were added to control undershoot, since U904 is located on the far side of the board.

EPROM

The EPROM can be either 256K or 512K, organized as either 32K or 64K 8-bit bytes. 15 address bits (p_a[00:14]) are used to access a byte in the 256K EPROM; 16 address bits are used in the 512K EPROM. The high order address bit (p_a[15]) for the 512K EPROM is jumpered at J1201. If the 256K EPROM is used, J1200 is jumpered. The table below gives this configuration.

Table 24-4 *EPROM Jumpering*

EPROM Size	Jumper IN or OUT?	
	J1200	J1201
256K	IN	OUT
512K	OUT	IN

The rd_eprom line has a 180Ω series termination resistor (R907) to handle undershoot.

24.9. Time of Day (TOD) Clock

The TOD clock is controlled by an Intersil 7170 real-time clock chip. Its 8-bit bidirectional data bus is connected to the MOS data bus (m_d[7:0]); 5 bits of MOS address (mos_a[4:0]) select from among 18 on-chip calendar functions. The m_d[7:0] bus is susceptible to undershoot (due to the length of the bus) so their source, U902, is an HCMOS device.

Also connected to the 7170 are the MOS read and write strobes: wr_tod- and rd_tod-, both low active signals issued from U904.

TOD Oscillator Circuit

The U1204 37.268 KHz crystal is part of a parallel resonant oscillator circuit. 200 KΩ R1203 is in series with the crystal as a limiting resister because the crystal can only handle about 10 µwatts. R1203 also acts as a wave-shaper.

C1201 adjustable capacitor on the output side of the 7170, together with C1200 loading capacitor (on the input of the 7170) allow you to adjust the crystal frequency to the clock chip.

U1202 battery backup is connected to VCC to power the TOD chip when power (VCC) is off.

- In battery backup mode, a switch inside the 7170 chip disconnects ground at pin 11 and connects the battery to pin 14. When ground is disconnected the circuit is isolated and there is no possibility of battery current leakage off-chip.
- When the system is powered up (VCC available) the battery is disconnected by the 7170 and ground is reconnected. When the battery is disconnected, there is a very high input impedance to the 7170 at pin 14, and thus there is practally no current drain. However the battery needs some sort of current drain, so R1202 470 Ω bleeder resister has been added to the circuit to allow a controlled discharge, keeping the battery stabilized around 3 volts.

The TOD interrupt is issued at pin 12 of the 7170 chip. The low active interrupt signal (int-) is connected to a 4.7 K pull-up resistor, inverted through U1104 inverter with hysteresis.† The signal is then connected as clock to the two interrupt flip-flops, U1205-1 and U1205-2.

U1201-2 (top flip-flop on extreme right of page 12 of the schematics) is interrupt request level 7 flip-flop; U1201-1 beneath it serves as interrupt request flip-flop for level 5. Level 7 is a non-maskable interrupt, and is provided for software profiling. When either flip-flop is cleared, the appropriate interrupt request is asserted.

E33 test point is used for calibrating the 7170 TOD chip.

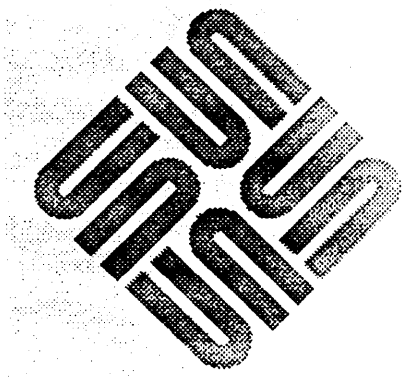
† Hysteresis is necessary because the positive edge of the interrupt signal has an *extremely* slow edge, and without hysteresis could trigger false clocks out to the interrupt flip-flops.

25

TTL Bus Accesses

TTL Bus Accesses **195**

25.1. TTL Bus Read/Write Cycle 195
25.2. U1400 TTL Bus Sack State Machine 195
U1400 Pinout 196
U1400 State Machine Outputs 196
TTL Bus DTACK State Machine Diagram 198
TTL Bus Cycle Timing 203
25.3. U1401 TTL Bus Device Decoder 204
U1401 Pinout 205
U1401 Input Signals 205
Output Signals of the U1401 PAL 206
25.4. U1402 MMU Decoder 210
U1402 Pinout 212
U1402 Input Signals 212
U1402 Output Signals 213
25.5. U1403 (Miscellaneous) CPU Signal TTL Bus Decoder 218
Pinout of U1403 PAL 219
U1403 Input Signals 219
U1403 Output Signals 220
25.6. Ethernet Control Register 223
U1405 Ethernet Control Write Buffer 223
U1407 Ethernet Control Read Buffer 224
25.7. System Enable Register 224



U1406 System Enable Write Register 224
U1408 System Enable Read Register 225
25.8. U1410 Diagnostics Register 225
25.9. U1409 ID PROM 225
25.10. U1404 P2-to-TTL Data Buffer 225
25.11. U2905 and U2906 User DVMA Enable Register 226
25.12. U203 Bus Error Register 226
25.13. U509 Context Register 226
U509 Pinout 227
Inputs and Outputs of U509 Context Register 227

25

TTL Bus Accesses

Page 14(a) of the schematics covers decoders U1400-U1403. These are:

- U1400 — TTL bus read/write, sack-, and buffer signal decoder
- U1401 — read/write strobe decoder to individual TTL devices
- U1402 — MMU read/write/control signal decoder
- U1403 — miscellaneous TTL-bused CPU control signal decoder.

25.1. TTL Bus Read/Write Cycle

The TTL bus decoder PALs U1403:1 are combinatorial, that is, based on the signals present at their inputs they asynchronously issue various output signals to start the I/O cycle.

The end of the I/O cycle occurs synchronously, however, triggered by signals from U1400 PAL. These synchronous signals are the read and write END signals.

Thus the beginning of an I/O cycle is *not* dependent upon U1400, but the end of the cycle *is* dependent.

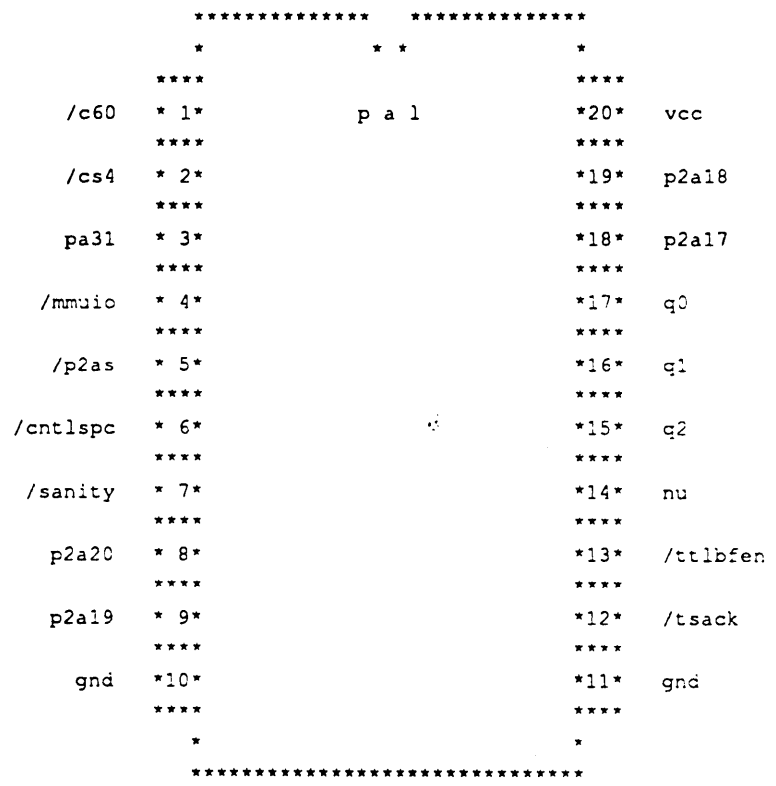
25.2. U1400 TTL Bus Sack State Machine

This PAL controls the TTL bus transceiver and generates the TTL bus dsack signal, labelled tsack. q2, q1, and q0 are the dsack state machine control counter bits. The other combinatorial output is ttlbfen-, the TTL bus data transceiver (U1404) output enable. The output direction of the data transceiver is controlled by the p2_rw signal. The signal ttlbfen- is active for all valid device and control space devices. Note that ttlwend- is the same as counter bit q0 and ttlrdend- is the same as control counter bit q1. Also output are the TTL read and write state indicators — ttlrdend (TTL read end) and ttlwend (TTL write end).

U1400 Pinout

Pinout of the U1400 PAL is:

Figure 25-1 *U1400 Pinout*



U1400 State Machine Outputs

Outputs from the state machine are:

- ttlbfen- : enables the output of the TTL buffers,
- tsack- : synchronous interrupt acknowledge for the TTL bus,
- q1/ttlrdend- : signals the end of the TTL read cycle; this signal is coupled to the q1 bit of the state machine's internal control counter,
- q0/ttlwend- : signals the early end of the TTL write cycle; this signal is coupled to the q0 bit of the state machine's internal control counter).

NOTE *For the following description, please consult both the state diagram and the read and write timing diagrams for the TTL bus.*

These output signals are derived through the following equation:

```
ttlbfen = cntlspc * /pa31 * cs4 +

          mmuio*p2as*/p2a20*p2a19*/p2a18 +

          mmuio*p2as*/p2a20*p2a19*/p2a17 +

          ttlbfen * q1
```

This equations tells us that ttlbfen- (TTL buffer's output is enabled) is asserted when:

1. you are doing a control space access, processor address bit 31 is low, and you are in clock state four, or
2. you are accessing device space through the MMU, address strobe is asserted, and the address bits A20:18 decode to either the parity error or interrupt registers in TYPE1 address space, or
3. you are accessing device space through the MMU, address strobe is asserted, and the address bits A20:19, A17 decode to either the parity error or Ethernet controller registers in TYPE1 address space, or
4. TTL buffer's output is enabled (ttlbfen- stays low) until TTLEND state (q1 bit goes low). This self-latching mechanism is necessary to enable the output of the TTL buffer (U1404) until the end of the bus cycle, because cs4 deactivates during TTLWREND state.

```
tsack  = /q2 * /q0 * p2as +

         /q2 * q1 * /sanity * p2as
```

The tsack- (TTL bus synchronous acknowledge) signal is asserted when:

1. the q2 bit of the control state counter is low and q0 (ttlwend-, signalling the end of the TTL write cycle) is low, and address strobe is asserted, or
2. the q2 bit of the control state counter is low and q1 is high (indicating the TTL read cycle is NOT completed), sanity (init-) is not asserted, and address strobe is asserted.

```
/q2 := /q2 * q1 * /q0 +
       /q2 * q1 * /sanity +
       q1 * q0 * /sanity * ttlbfen

/q1 (ttlrdend) := /q2 * q1 * /q0

/q0 (ttlwend) := /q2 * q1 * /q0 +
                 /q2 * q1 * /sanity
```

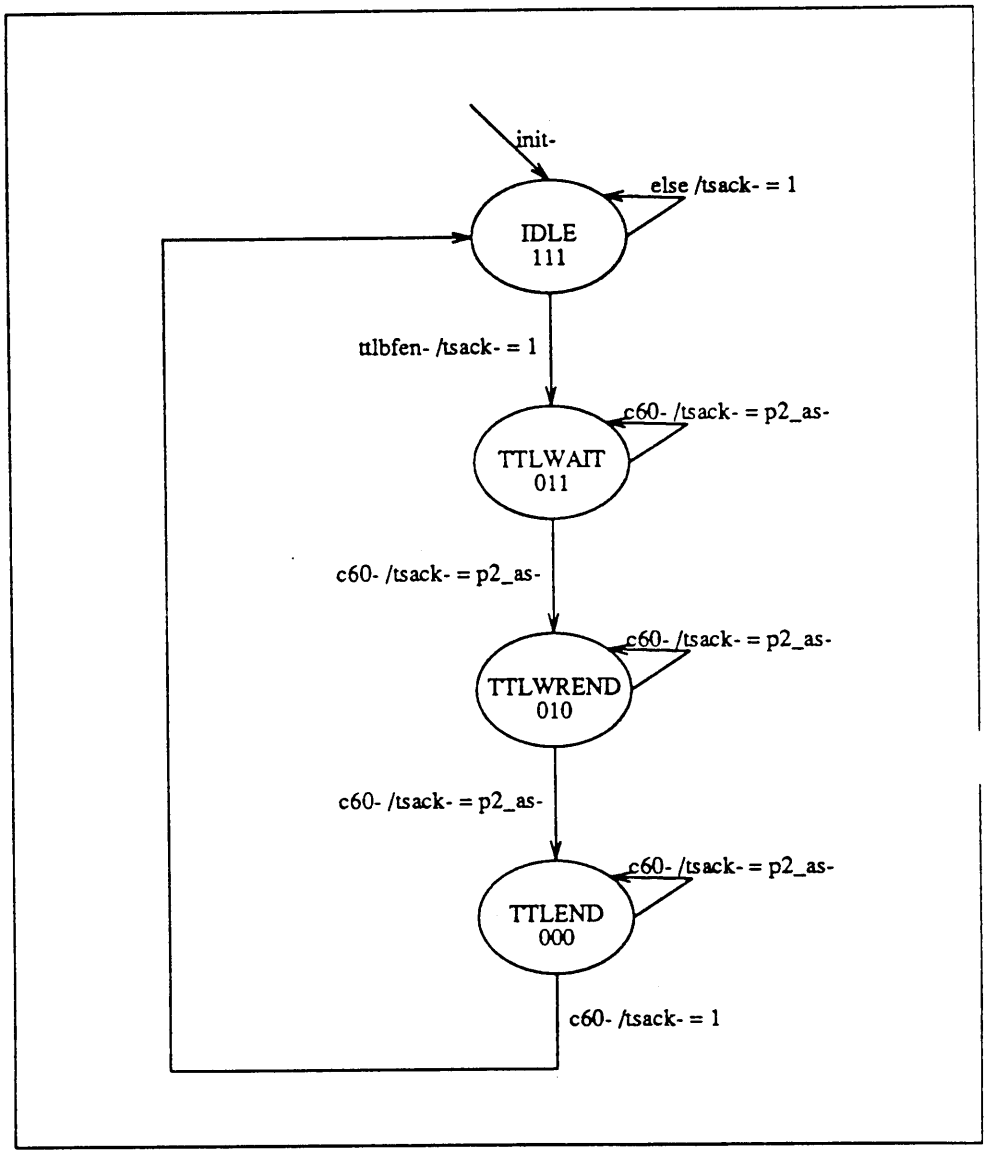
The three internal control state counter bits are derived from the equations above.

TTL Bus DTACK State Machine Diagram

States defined for U1400 state machine are:

```
ttlend     =    end of TTL bus cycle
nostate1   =    not used state 1
ttlwrend   =    write strobe dissable
ttlwait    =    TTL bus wait state
nostate4   =    not used state 4
nostate5   =    not used state 5
nostate6   =    not used state 6
idle       =    idle state
```

Figure 25-2 *TTL Bus DTACK (SACK) State Machine Diagram*



There are four states used in the state machine, and their lengths in the cycle are given below:

Table 25-1 *TTL Bus DTACK State Machine States*

State	*Length the State runs in the cycle*
IDLE	0 to 180 nsecs
TTLWAIT	180 to 240 nsecs
TTLWREND	240 to 300 nsecs
TTLEND	300 to 360 nsecs
	return to IDLE state

The init- signal (also known euphemistically as the system "sanity" signal because it sets the system to a known condition) sets this state machine to the IDLE state, with the control state counter at 111 (q2, q1, and q0 = 111). In IDLE state, the interrupt acknowledge signal, tsack-, remains high (inactive).

IDLE is defined as:

```
state idle        No valid TTL bus access.  Wait here in idle.

!sanity-
    -> state idle,              power on reset

                    tsack- = 1;
        else

        !ttlbfen-
        -> state ttlwait,   valid TTL bus access: goto wait state

                    tsack- = 1;
        else

        always  -> state idle,        no valid cycle: stay in idle

                    tsack- = 1;
```

When ttlbfen- goes low, (actually on the next edge of c60 clock following the assertion of ttlbfen-) you enter TTLWAIT state (q2, q1 and q0 = 011). This is 180 nsecs into the TTL bus cycle; the assertion of ttlbfen- signals the beginning of a TTL access. In TTLWAIT state, tsack- goes low and paces the state of the address strobe signal, p2_as-.

TTLWAIT is defined as:

```
state ttlwait

Valid TTL bus cycle start. Insert a
wait state by spending one
clock in this state. Issue tsack-.

        !sanity-
                -> state idle,          power on reset

                   tsack- = 1;
        else

        always  -> state ttlwrend,      enough! Issue tsack- and go to
                                        ttlwend-
                   tsack- = p2as-;
```

On the next c60 clock, U1400 enters TTL write end, TTLWREND, state (q2, q1 and q0 = 010). Notice that the LSB, q0, has gone from a one to a zero; remember that this bit is also the signal ttlwrend. Thus, ttlwrend- is asserted and issued on the output of U1400. If you look at the timing diagram, you will see that this occurs 245 to 255 nsecs into the bus cycle. The tsack- signal is still pacing address strobe (still low) because the acknowledge signal must stay valid until address strobe is deasserted.

TTLWREND issues an "early end" to the write cycle, necessary to guarantee that the data and address hold times are met. The ttlwrend- signal is connected to U1403:1 to deassert the write strobes coming from these three PALs.

TTLWREND is defined as:

```
state ttlwrend

One wait state has passed. Issue ttlwend- (q0 bit) to
shut down the write strobe to the various TTL devices.

!sanity-
        -> state idle,          power on reset

           tsack- = 1;

always  -> state ttlend,        stay here for one clock

           tsack- = p2as-;
```

TTLEND state occurs 300 nsecs into the TTL bus cycle, on the next c60 clock. The control state counter goes from 010 to 000 — q1 goes from a one to a zero. Recall that q1 is also the ttlrdend signal; thus when you enter TTLEND, ttlrdend- goes low (true) which signals the end of the TTL read cycle, deactivating any read strobes issued to the bus by U1401:3. The tsack signal goes high (is deasserted) following p2_as, 307 nsecs into the bus cycle.

TTLEND is defined as:

```
state ttlend

 This is the last state in the TTL bus cycle.
Here both ttlwend- and ttlrdend- are active.

!sanity-
        -> state idle,      power on reset

           tsack- = 1;

always  -> state idle,      cycle over, go back to idle

           tsack- = p2as-;
```

On the next c60 clock you re-enter IDLE state. The tsack- signal remains high (inactive).

Unused states are defined below:

```
Not-used states. Do not go here. If you get here on a power on reset
leave immediately or you will be shot for loitering.†

state nostate1

        always  -> state idle,

                   tsack- = 1;
```

†Don't blame us for the editorial comments; they were included in the PAL listings by the design engineer. Personally, I would take any threats of bodily harm with a large grain of salt.

```
state nostate4

        always  -> state idle,

                   tsack- = 1;
```

```
state nostate5

        always  -> state idle,

                   tsack- = 1;
```

```
state nostate6

        always  -> state idle,

                   tsack- = 1;


end
```

TTL Bus Cycle Timing

This section refers to the timing diagram for TTL bus reads and writes.

1. In IDLE state, ttlbfen- occurs sometime during cs4, depending upon whether it is a control space access (labelled ttlbfen[c]) or an access through the MMU (labelled ttlbfen[mmu]).
2. When ttlbfen- occurs during cs4, the state machine enters TTLWAIT state on the next negative edge of c60 clock (s5).
3. The tsack- signal drops low 190 to 220 nsecs into the cycle.
4. On the next clock (s7), 245 to 255 nsecs into the cycle, ttlwend- goes true (low). The cs4 signal goes high.
5. On the next clock (s9), ttlrdend- goes active, 305 to 315 nsecs into the cycle. Now both TTL read and write cycles are ended, and you are in TTLEND state. Address strobe goes high, and tsack- does likewise.
6. On the next clock IDLE state returns.

25.3. U1401 TTL Bus Device Decoder

This PAL generates the read and write strobes for the TTL devices in p2_type1 (device) space. The parity latches are arranged as an array of five bytes (four address and one data) so that the dynamic bus sizing of the 68020 can be used, hence the BYTExx term in the rd_padxx equations below. For example, the BYTE00 term selects the low order byte of the parity address register, bits 00 through 07. BYTE08 selects the next significant byte of the address register, bits 08 to 15. And so on.

Table 25-2 *Byte Selection in Parity Address Selection*

Term	Address Bits *A1*	*A0*	Signal Enabled
BYTE24	0	0	pad24
BYTE16	0	1	rd_pad16
BYTE08	1	0	rd_pad08
BYTE00	1	1	rd_pad00

In order to insure data and address hold times, the /ttlwend signal is incorporated into the /p2_rw*mmuio*/ttlwend definition so the write strobes go inactive before the end of the bus cycle.

The devices for whom this PAL issues read and write strobes are listed below, along with their addresses (A31:28):

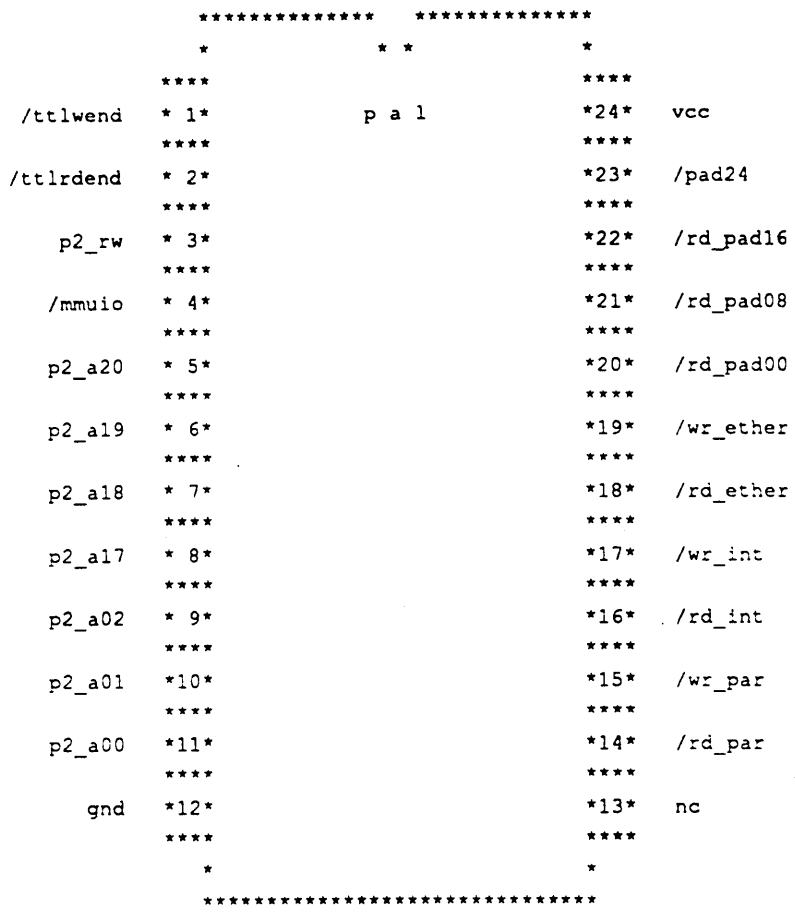
```
Parity error register = /p2_a20*p2_a19*/p2_a18*/p2_a17 = 0x080000

Interrupt register = /p2_a20*p2_a19*/p2_a18*p2_a17 = 0x0A0000

Ethernet control register = /p2_a20*p2_a19*p2_a18*/p2_a17 = 0x0C0000
```

These are all mapped through the MMU, in TYPE1 space.

U1401 Pinout

Pinout of U1401 is:

Figure 25-3 *U1401 Pinout*



U1401 Input Signals

Inputs to the U1401 PAL are:

```
/mmuio        =  valid type 1 space access

/ttlrdend     =  ends read strobe to avoid buffer conflict

/ttlwend      =  write strobe end for data/address hold time

p2_a<20:17>   =  physical address from MMU,
                  to select the registers

p2_a<2:0>     =  physical address from MMU,
                 to select bytes from the array of
                 parity registers

p2_rw         =  processor read/write- signal
```

The p2_a(2:0) address bits decode decode down to one of five bytes of the parity error and address registers. There is a single byte of parity error register and there are four bytes of parity address, which are decoded as follows ("X" means "Don't Care"):

Table 25-3 *Byte Decode of the Parity Error and Address Registers*

Address bits			Signal	Register
A2	*A1*	*A0*	Enabled	Selected
0	X	X	rd_par *or* wr_par	parity error (U801 and U813)
1	0	0	pad24	parity address (U811:08)
1	0	1	rd_pad16	parity address (U811:08)
1	1	0	rd_pad08	parity address (U811:08)
1	1	1	rd_pad00	parity address (U811:08)

Output Signals of the U1401 PAL

Output signals from the U1401 PAL are:

```
/rd_ether = read strobe to Ethernet control register

/wr_ether = write strobe to Ethernet control register

/rd_int = read strobe to interrupt enable register

/wr_int = write strobe to interrupt enable register

/rd_par = read strobe to parity error reg

/wr_par = write strobe to parity error reg

pad24 = read/write strobe to parity address latch <31:24>

/rd_pad16 = read strobe to parity address latch <23:16>

/rd_pad08 = read strobe to parity address latch <15:08>

/rd_pad00 = read strobe to parity address latch <07:00>
```

As explained above, all of the devices except for the parity address latch are single-byte latches. The parity address latch is 32 bits wide; the TTL bus is 8 bits wide. Therefore the address is broken up into four 8-bit registers. The CPU board uses the 68020's dynamic bus-sizing capability to execute four successive reads of the 32-bit parity address off the 8-bit TTL bus.

```
rd_ether = p2_rw*mmuio*/ttlrdend*
           /p2_a20*p2_a19*p2_a18*/p2_a17
```

This signal is the read strobe to the Ethernet control register, U1407. The signal simultaneously clocks and enables output from the Ethernet controller onto the TTL data bus. To assert rd_ether you must be in a read cycle (p2_rw high), accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), and have selected the ethernet control register via bits A20:17, which is equal to 0x0C0000 in TYPE1 space.

```
wr_ether = /p2_rw*mmuio*/ttlwend*
       /p2_a20*p2_a19*p2_a18*/p2_a17
```

This signal is the write strobe to the Ethernet control register, U1405. The signal clocks data from the TTL bus onto the data inputs of the Ethernet controller. To assert wr_ether you must be in a write cycle (p2_rw low), accessing the device

through the MMU (mmuio- true), must NOT be in TTLWREND state (ttlwend- must NOT be low), and have selected the ethernet control register via bits A20:17, which is equal to 0x0C0000 in TYPE1 space.

```
rd_int = p2_rw*mmuio*/ttlrdend*
     /p2_a20*p2_a19*/p2_a18*p2_a17
```

This signal is the read strobe to the interrupt register, U301. To assert rd_int you must be in a read cycle (p2_rw high), accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), and have selected the ethernet control register via bits A20:17, which is equal to 0x0A0000 in TYPE1 space.

```
wr_int = /p2_rw*mmuio*/ttlwend*
     /p2_a20*p2_a19*/p2_a18*p2_a17
```

This signal is the write strobe to the interrupt register, U300. The signal clocks data from the TTL bus onto the data inputs of the priority encoder, U302, and the interrupt enable bus. To assert wr_int you must be in a write cycle (p2_rw low), accessing the device through the MMU (mmuio- true), must NOT be in TTLWREND state (ttlwend- must NOT be low), and have selected the ethernet control register via bits A20:17, which is equal to 0x0A0000 in TYPE1 space.

```
rd_par = p2_rw*mmuio*/ttlrdend*
     /p2_a20*p2_a19*/p2_a18*/p2_a17*/p2_a02
```

This signal is the read strobe to the parity error register, U801 and U813. To assert rd_par you must be in a read cycle (p2_rw high), accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), have selected the parity error register via bits A20:17, which is equal to 0x080000 in TYPE1 space, and the p2_a02 address bit must be LOW (deselecting the parity address registers).

```
wr_par = /p2_rw*mmuio*/ttlwend*
     /p2_a20*p2_a19*/p2_a18*/p2_a17*/p2_a02
```

This signal is the write strobe to the parity error register, U801 and U813. To assert wr_par you must be in a write cycle (p2_rw low), accessing the device through the MMU (mmuio- true), must NOT be in TTLWREND state (ttlwend- must NOT be low), and have selected the ethernet control register via bits A20:17, which is equal to 0x080000 in TYPE1 space, and the p2_a02 address bit must be LOW (deselecting the parity address registers).

```
pad24  =  mmuio*/ttlrdend*
      /p2_a20*p2_a19*/p2_a18*/p2_a17*
      p2_a02*/p2_a01*/p2_a00
```

This signal generates the read strobe rd_pad24 (through U802 parity control PAL), to the high order parity address register, U808. To assert pad24 you must be accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), and have selected the parity register via bits A20:17, which is equal to 0x080000 in TYPE1 space, and the p2_a02:0 address bits must be equal to 100 (BYTE24 term in the PAL listings).

```
rd_pad16 = p2_rw*mmuio*/ttlrdend*
       /p2_a20* p2_a19*/p2_a18*/p2_a17*
       p2_a02*/p2_a01*p2_a00
```

This signal generates an output enable to the parity address register, U809. To assert rd_pad16 you must be in a read cycle (p2_rw high), accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), and have selected the parity register via bits A20:17, which is equal to 0x080000 in TYPE1 space, and the p2_a02:0 address bits must be equal to 101 (BYTE16 term in the PAL listings).

```
rd_pad08 = p2_rw*mmuio*/ttlrdend*
       /p2_a20* p2_a19*/p2_a18*/p2_a17*
       p2_a02*p2_a01*/p2_a00
```

This signal generates an output enable to the parity address register, U810. To assert rd_pad08 you must be in a read cycle (p2_rw high), accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), and have selected the parity register via bits A20:17, which is equal to 0x080000 in TYPE1 space, and the p2_a02:0 address bits must be equal to 110 (BYTE08 term in the PAL listings).

```
rd_pad00 = p2_rw*mmuio*/ttlrdend*
       /p2_a20*p2_a19*/p2_a18*/p2_a17*
       p2_a02*p2_a01*p2_a00
```

This signal generates an output enable to the parity address register, U811. To assert rd_pad00 you must be in a read cycle (p2_rw high), accessing the device through the MMU (mmuio- true), must NOT be in TTLEND state (ttlrdend- must NOT be low), and have selected the parity register via bits A20:17, which is equal to 0x080000 in TYPE1 space, and the p2_a02:0 address bits must be equal to 111 (BYTE00 term in the PAL listings).

NOTE *There are a pair of "generic" signals—one in each timing digram—which stand for the read and write strobes issued by U1401. The generic read signal is ttlspcrden- on the TTL BUS READS timing diagram; the generic write signal is ttlspcwren- on the TTL BUS WRITES timing diagram.*

25.4. U1402 MMU Decoder

This PAL generates the page and segment map RAM write strobes and the data transceiver (U508 and U610:7) output enable signals.

- The write strobes are enabled by cs2 being true and ttlwend- false.
- The gate outputs are enabled by cs4 being true and ttlwrend- false (write cycles) or by cs4 and latched until ttlrdend- is true (read cycles). This allows time for the write signal to disable the outputs of the RAM output buffers before the data transceivers' outputs are enabled.
- The direction of the data transceivers is controlled by p2_rw.

The page map is arranged as an array of bytes so the dynamic bus sizing capability of the 68020 can be used, hence the BYTExx term in the mmu_gtxx- and mmu_wexx- equations.

Table 25-4 *Byte Selection in the Page Map RAM*

Term	Address Bits *A1*	*A0*	Signal Enabled
BYTE24	0	0	mmu_we24
BYTE16	0	1	mmu_we16
BYTE08	1	0	mmu_we08
BYTE00	1	1	mmu_we00

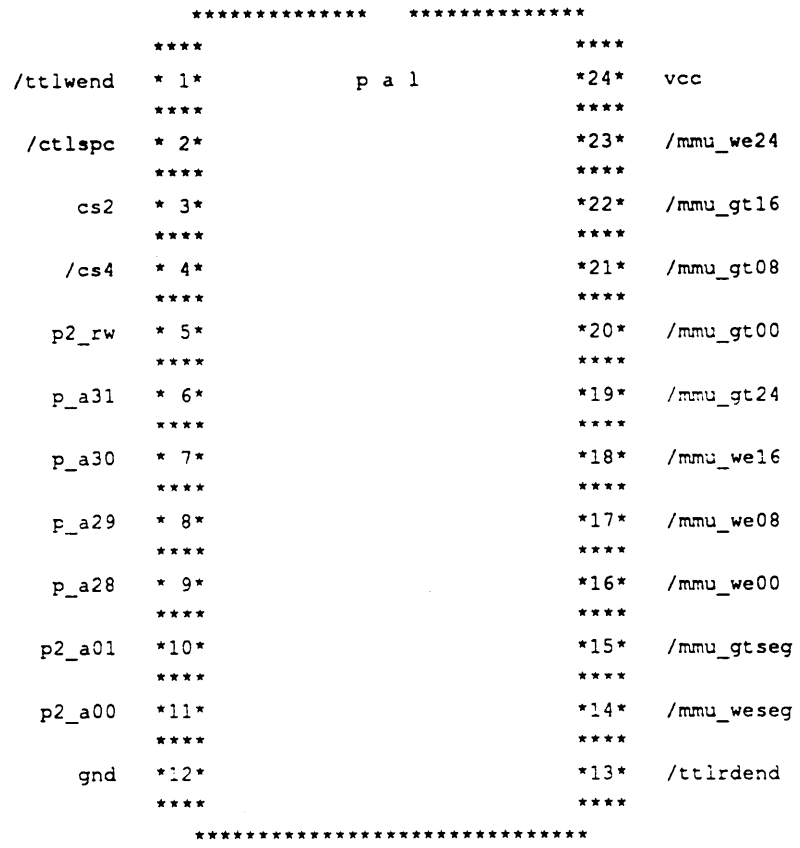
Finally, in order to insure data and address hold times, the ttlwend- signal is incorporated into the /p2_rw*/ttlwend definition so the write strobes go inactive before the end of the bus cycle.

```
WRITE = /p2_rw * /ttlwend
```

U1402 Pinout

Pinout of U1402 is:

Figure 25-4 *U1402 Pinout*



U1402 Input Signals

Inputs to the U1402 PAL are:

```
/ttlrdend    =   ends read strobe for buffer conflict avoidance

/ttlwend     =   ends write strobe for data/address hold time

/ctlspc      =   indicates the processor is doing a
                   control space cycle

p_a<31:28>   =   unbuffered virtual address (for register decode)

p2_a<1:0>    =   buffered virtual address (for byte decode)

cs2          =   used to enable the write strobes

/cs4         =   used to enable the gates

p2_rw        =   buffered processor read/write- signal
```

U1402 Output Signals

Outputs from the U1402 PAL are:

```
/mmu_gt24 =   gate buffer for byte 0 of page map
                  (prot and id bits)

/mmu_gt16 =   gate buffer for byte 1 of page map - page<18:16>

/mmu_gt08 =   gate buffer for byte 2 of page map - page<15:8>

/mmu_gt00 =   gate buffer for byte 3 of page map - page<7:0>

/mmu_we24 =   write enable byte 0 of page map -
                  (prot and id bits)

/mmu_we16 =   write enable byte 1 of page map - page<18:16>

/mmu_we08 =   write enable byte 2 of page map - page<15:8>

/mmu_we00 =   write enable byte 3 of page map - page<7:0>

/mmu_gtseg =  gate buffer for segment RAM

/mmu_weseg =  write enable segment RAM
```

Equations for U1402's output signals are below:

```
mmu_gt24 = ctlspc*/p_a31*/p_a30*/p_a29* p_a28 *
                /p2_a01*/p2_a00*p2_rw*cs4 +

           mmu_gt24*p2_rw*/ttlrdend +

           ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
                /p2_a01*/p2_a00*cs4*/p2_rw*/ttlwend
```

This signal gates the buffer for byte 0 of the page maps. It is asserted when:

1. you are doing a control space access and address bits pa_<31:28> decode to the page map RAM (0001), the p2_a(01:00) address bits decode to BYTE24 (00), you are doing a read cycle (p2_rw is high), and you are in clock state four (cs4 is valid); or
2. mmu_gt24 is asserted (this is the self-latching mechanism, needed to extend this read cycle beyond the deassertion of cs4), p2_rw indicates a read cycle (signal is high), and you have not entered the end of the read cycle (ttlrdend is still high); or
3. you are doing a control space access to the device decoded by address bits pa_<31:28> to be the page map RAM (0001), p2_a901:00) decode to BYTE24 (00), you are in clock state four (cs4 is valid), you are in a write cycle (p2_rw is low) and you have not yet entered the end of the write cycle (ttlwend is false).

```
mmu_gt16 = ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
                /p2_a01*p2_a00*p2_rw*cs4 +

           mmu_gt16*p2_rw*/ttlrdend +

           ctlspc*/p_a31*/p_a30*/p_a29* p_a28*
                /p2_a01*p2_a00*cs4*/p2_rw*/ttlwend
```

This signal gates the buffer for byte 1 of the page maps. It is asserted when:

1. you are doing a control space access and address bits pa_<31:28> decode to the page map RAM (0001), the p2_a(01:00) address bits decode to BYTE16 (01), you are doing a read cycle (p2_rw is high), and you are in clock state four (cs4 is valid); or
2. mmu_gt16 is asserted (this is the self-latching mechanism, needed to extend this read cycle beyond the deassertion of cs4), p2_rw indicates a read cycle (signal is high), and you have not entered the end of the read cycle (ttlrdend is still high); or

3. you are doing a control space access to the device decoded by address bits pa_<31:28> to be the page map RAM (0001), p2_a(01:00) decode to BYTE16 (01), you are in clock state four (cs4 is valid), you are in a write cycle (p2_rw is low) and you have not yet entered the end of the write cycle (ttlwend is false).

```
mmu_gt08 = ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
             p2_a01*/p2_a00*p2_rw*cs4 +

           mmu_gt08*p2_rw*/ttlrdend +

           ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
             p2_a01*/ p2_a00*cs4*/p2_rw*/ttlwend
```

This signal gates the buffer for byte 2 of the page maps. It is asserted when:

1. you are doing a control space access and address bits pa_<31:28> decode to the page map RAM (0001), the p2_a(01:00) address bits decode to BYTE08 (10), you are doing a read cycle (p2_rw is high), and you are in clock state four (cs4 is valid); or
2. mmu_gt08 is asserted (this is the self-latching mechanism, needed to extend this read cycle beyond the deassertion of cs4), p2_rw indicates a read cycle (signal is high), and you have not entered the end of the read cycle (ttlrdend is still high); or
3. you are doing a control space access to the device decoded by address bits pa_<31:28> to be the page map RAM (0001), p2_a(01:00) decode to BYTE08 (10), you are in clock state four (cs4 is valid), you are in a write cycle (p2_rw is low) and you have not yet entered the end of the write cycle (ttlwend is false).

```
mmu_gt00 = ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
             p2_a01* p2_a00*p2_rw*cs4 +

           mmu_gt00*p2_rw*/ttlrdend +

           ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
             p2_a01*p2_a00*cs4*/p2_rw*/ttlwend
```

This signal gates the buffer for byte 3 of the page maps. It is asserted when:

1. you are doing a control space access and address bits pa_<31:28> decode to the page map RAM (0001), the p2_a(01:00) address bits decode to BYTE00 (11), you are doing a read cycle (p2_rw is high), and you are in clock state four (cs4 is valid); or
2. mmu_gt00 is asserted (this is the self-latching mechanism, needed to extend this read cycle beyond the deassertion of cs4), p2_rw indicates a read cycle (signal is high), and you have not entered the end of the read cycle (ttlrdend is still high); or
3. you are doing a control space access to the device decoded by address bits pa_<31:28> to be the page map RAM (0001), p2_a(01:00) decode to BYTE00 (11), you are in clock state four (cs4 is valid), you are in a write cycle (p2_rw is low) and you have not yet entered the end of the write cycle (ttlwend is false).

```
mmu_we24 = ctlspc*/p_a31*/p_a30*/p_a29*p_a28*
           /p2_a01*/p2_a00*/p2_rw*/ttlwend*cs2
```

This signal (mmu_we24) is the write enable for byte 0 of the page map RAM. It is active when you are doing a control space access (ctlspc- true) to the page map RAM (pa_<31:28> = 0001), the p2_a(01:00) address bits decode to BYTE24 (00), you are in a write cycle (p2_rw is low), you have not ended the write cycle (ttlwend is high/false) and you are in clock state two (cs2 is true).

```
mmu_we16 = ctlspc* /p_a31*/p_a30*/p_a29* p_a28*
           /p2_a01* p2_a00* /p2_rw * /ttlwend*cs2
```

This signal (mmu_we16) is the write enable for byte 1 of the page map RAM. It is active when you are doing a control space access (ctlspc- true) to the page map RAM (pa_<31:28> = 0001), the p2_a(01:00) address bits decode to BYTE16 (01), you are in a write cycle (p2_rw is low), you have not ended the write cycle (ttlwend is high/false) and you are still in clock state two (cs2 is true).

```
mmu_we08 = ctlspc* /p_a31*/p_a30*/p_a29* p_a28*
           p2_a01*/ p2_a00* /p2_rw * /ttlwend*cs2
```

This signal (mmu_we08) is the write enable for byte 2 of the page map RAM. It is active when you are doing a control space access (ctlspc- true) to the page map

RAM (pa_<31:28> = 0001), the p2_a(01:00) address bits decode to BYTE08 (10), you are in a write cycle (p2_rw is low), you have not ended the write cycle (ttlwend is high/false) and you are still in clock state two (cs2 is true).

```
mmu_we00 = ctlspc* /p_a31*/p_a30*/p_a29* p_a28 *
             p2_a01* p2_a00* /p2_rw * /ttlwend*cs2
```

This signal (mmu_we00) is the write enable for byte 3 of the page map RAM. It is active when you are doing a control space access (ctlspc- true) to the page map RAM (pa_<31:28> = 0001), the p2_a(01:00) address bits decode to BYTE00 (11), you are in a write cycle (p2_rw is low), you have not ended the write cycle (ttlwend is high/false) and you are still in clock state two (cs2 is true).

```
mmu_gtseg = ctlspc*/p_a31*/p_a30*p_a29*/p_a28*
                p2_rw*cs4 +

            mmu_gtseg* p2_rw*/ttlrdend +

            ctlspc*/p_a31*/p_a30*p_a29*/p_a28
                 *cs4* /p2_rw * /ttlwend
```

This signal (mmu_gtseg) gates the buffer for the segment map RAM. It is active when:

1. you are doing a control space access (ctlspc- true) to the segment map RAM (pa_<31:28> = 0010), and you are in a read cycle (p2_rw is high) during clock state 4 (cs4 is true), or;
2. mmu_gtseg is true (this is the self-latching term, ensuring that this read strobe lasts the entire length of the read cycle, beyond the time cs4 is deactivated), you are in a read cycle (p2_rw is high) and you have not entered the end of the read cycle (ttlrdend is false), or;
3. you are doing a control space write cycle to the segment map RAM (pa_<31:28> = 0010), you are in clock state four (cs4 is true) and you have not entered the end of the write cycle (ttlwend is false/high).

```
mmu_weseg = ctlspc*/p_a31*/p_a30* p_a29*/p_a28*
              /p2_rw * /ttlwend*cs2
```

This signal (mmu_weseg) is the write enable strobe to the eight segment map RAM chips. It is active when you are doing a control space access to the segment map RAM, you are in a write cycle during clock state 2, and you have not entered the end of the write cycle (ttlwend is false/high).

Note the self-latching terms in the read cycle signals; since cs4 is deasserted before the end of the read cycle, the MMU gate signal is fed back into itself, self-latching. This lasts until ttlrdend- goes true, indicating end of the read cycle.

25.5. U1403 (Miscellaneous) CPU Signal TTL Bus Decoder

This PAL generates the read and write strobes for Control Space devices which are not included in the Page and Segment Map RAM. Of special note are two signals:

1. the ttlwend- signal, which is used to disable the write strobes before the end of the bus cycle to insure data and address hold time, and
2. the ttlrdend- signal which disables the read strobes so there is no end-of-cycle buffer conflict. The read output strobes are latched inside U1403 since cs4- ends earlier than the read cycle.

The Control Space devices (and their addresses) to which this PAL sends read/write strobes are:

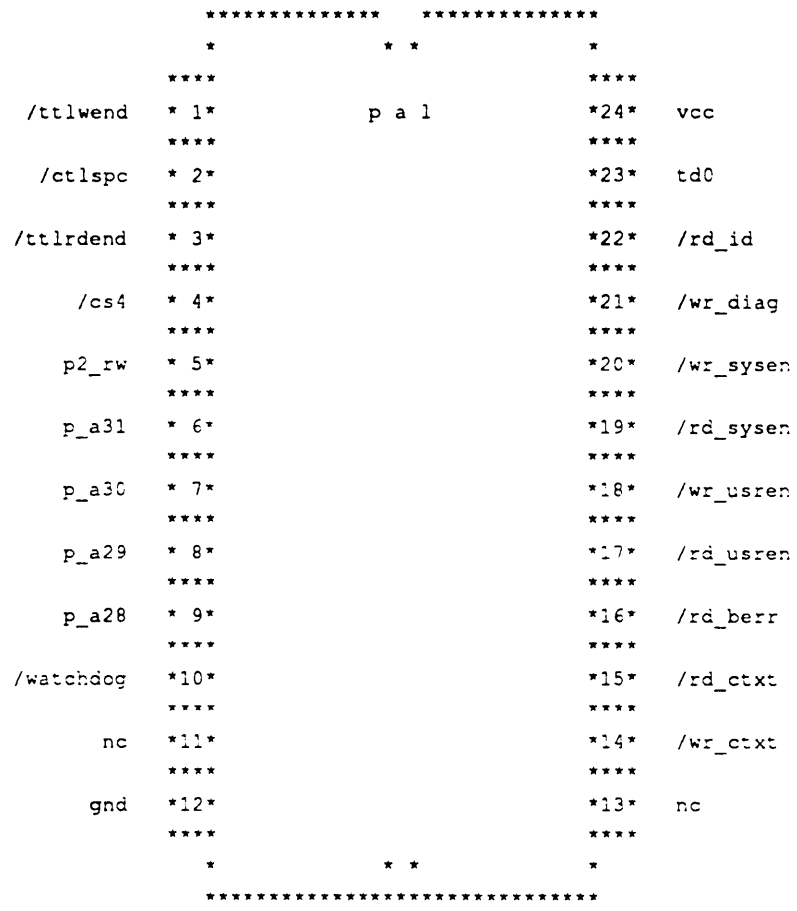
```
IDPROM      =   /p_a31*/p_a30*/p_a29*/p_a28   = 0
CONTEXT     =   /p_a31*/p_a30* p_a29* p_a28   = 3
SYSENABLE   =   /p_a31* p_a30*/p_a29*/p_a28   = 4
USERENABLE  =   /p_a31* p_a30*/p_a29* p_a28   = 5
BUSERROR    =   /p_a31* p_a30* p_a29*/p_a28   = 6
DIAG        =   /p_a31* p_a30* p_a29* p_a28   = 7
```

Note an interesting characteristic of all of these devices: address bit p_a31 is low. This is an easy one-bit way of selecting or deselecting all of these devices simultaneously; in fact, this method is used in the state machine.

Pinout of U1403 PAL

Pinout of U1403 is:

Figure 25-5 *U1403 Pinout*



U1403 Input Signals

Input signals to the U1403 PAL are:

```
/ttlwend    = end write strobes for
              data/address hold time (see U1400)

/ctlspc     = processor is doing a control space cycle

p_a<31:28>  = unbuffered virtual address (for decoding
              to specific control register)

/cs4        = used to enable read/write strobes

/ttlrdend   = buffered processor address strobe; defines
              the end of the TTL read cycle (see U1400)

p2_rw       = buffered processor read/write- signal

/watchdog   = watchdog reset bit
```

U1403 Output Signals

U1403 output signals are:

```
/rd_id      = read strobe for the ID prom

/rd_ctxt    = read strobe for the context reg

/wr_ctxt    = write strobe for the context reg

/rd_sysen   = read strobe for the system enable reg

/wr_sysen   = write strobe for the system enable reg

/rd_usren   = read strobe for the user DVMA enable reg

/wr_usren   = write strobe for the user DVMA enable reg

/rd_berr    = read strobe for the bus error reg

/wr_diag    = write strobe for the
                   diagnostic register (LEDs)

td0         = watchdog readback bit driver
                   onto TTL data bus
```

- A read cycle is defined as a cycle in which the read signal is active (p2_rw is high) and you are in cs4 (cs4 is low).
- A write cycle is defined as a cycle in which the write signal is active (p2_rw is low), you are still in a TTL write cycle (ttlwend is high) and you are in cs4 (cs4 is low).

The equations for the output signals are given below. Note that the second OR term of the various read strobes are feedback loops which latch the output to the

input, making the read strobe last until the end of the TTL read cycle (until ttlrdend goes low). This feedback loop is necessary because cs4 does not last the length of the entire read cycle, therefore the individual device's read strobe must be latched until the end of the read cycle (ttlrdend- comes true). Write cycle strobes do not have to be self-latching since cs4 lasts as long as the write cycle. Thus the write strobe can be deactivated by the deassertion of cs4.

The watchdog bit is a reset bit from U201 which signals various system errors or someone has pressed the user RESET switch. The signal on pin 23 of U1403, t_d(0), the LSB of the TTL data bus, acts as a 1-bit watchdog register. If you are doing a read of the bus error register (U203), by asserting rd_berr-, this same rd_berr- signal will force a read of the watchdog bit from U1403. Thus the status of the watchdog error bit is available on the TTL bus every time you read the bus error register. This is derived from the following equation:

```
if(rd_berr) /td0 = /watchdog
```

This equations says that if rd_berr is true then data bit zero of the TTL bus (td0) is the same state as the watchdog bit.

Other outputs from U1403 are defined below:

```
rd_id = ctlspc*/p_a31*/p_a30*/p_a29*/p_a28*p2_rw*cs4 +

        rd_id*/ttlrdend
```

This equation tells us that the read strobe for the ID PROM is derived one of two ways:

1. a control space access to the device addressed by bits A31:28 as 0000 while doing a read (p2_rw high) during clock state 4 (cs4 true), or
2. the signal rd_id is asserted and ttlrdend is NOT asserted (high).

This second OR term is the self-latching mechanism discussed earlier. The self-latching of the rd_id signal is defeated when ttlrdend goes low (is true).

```
rd_ctxt = ctlspc*/p_a31*/p_a30*p_a29*p_a28*p2_rw*cs4 +

          rd_ctxt * /ttlrdend
```

This equation defines a read from the context registers as:

1. a control space access to the device selected by address bits A31:28 when they are 0011 (0x3), and assertion of the read strobe during cs4, or
2. the read context signal latched and enabled by the deassertion of the ttlrdend signal (rd_ctxt is true as long as ttlrdend is false).

```
wr_ctxt = ctlspc*/p_a31*/p_a30*p_a29*p_a28*/p2_rw*/ttlwend*cs4
```

This equation defines a write to the context registers as a control space access to the device selected by address bits A31:28 when they are 0011 (0x3), and assertion of the write strobe during cs4 as long as you have not entered ttlwend state (ttlwend signal is high).

Notice that there is no self-latching needed for write strobes; the TTL bus write cycle ends when cs4 does, thus ensuring hold times for the address and data.

```
rd_sysen = ctlspc*/p_a31* p_a30*/p_a29*/p_a28 * p2_rw*cs4 +
           rd_sysen * /ttlrdend
```

This equation defines the read strobe to the system enable register, at A31:28 0100 (0x4). It is similar to the other read strobes explained above, and has the read self-latching mechanism.

```
wr_sysen = ctlspc*/p_a31*p_a30*/p_a29*/p_a28*/p2_rw*/ttlwend*cs4
```

This defines a write to the system enable register.

```
rd_usren = ctlspc*/p_a31* p_a30*/p_a29* p_a28 * p2_rw*cs4 +
           rd_usren * /ttlrdend
```

This defines a read of the user DVMA enable register, at A31:28 = 0101 (0x5). The read signal is self-latching.

```
wr_usren = ctlspc*/p_a31*p_a30*/p_a29*p_a28*/p2_rw*/ttlwend*cs4
```

This defines a write to the user DVMA enable register.

```
rd_berr = ctlspc*/p_a31* p_a30* p_a29*/p_a28 * p2_rw*cs4 +
          rd_berr * /ttlrdend
```

This defines a read of the bus error register, at A31:28 0110 (0x6). The read signal is self-latching.

```
wr_diag = ctlspc*/p_a31*p_a30*p_a29*p_a28*/p2_rw*/ttlwend*cs4
```

This defines a write to the diagnostics register, which are really the LEDs, at 0111 (0x7).

25.6. Ethernet Control Register

The Ethernet control register is on page 14(b) of the schematics, and includes U1405 and U1407. U1405 operates as a write buffer; U1407 as a read buffer.

U1405 Ethernet Control Write Buffer

The TTL devices connected to the TTL bus all use the same sort of circuitry: ALS 273s latch write data from the TTL bus for the particular device, which is clocked on the rising edge of the device's individual write strobe issued by U1401 TTL bus decoder PAL.

U1405 buffer is used for write data storage to the Ethernet controller. Data at the input(s) to the buffer (t_d[7:0]) is clocked to the Ethernet controller by the rising edge (deassertion) of the wr_ether- signal, issued by U1401 PAL, and which occurs at the end of a write cycle. If you look at the TTL BUS WRITES timing diagram, this signal is lumped under the generic label "ttlspcwren" which is asserted 138-186 nsecs into the write cycle, and deasserted 260-305 nsecs into the write cycle.

U1405 write buffer latches four control signals to the Ethernet controller:

1. e_inten- : interrupt enable to the controller
2. e_ce : chip enable to the controller
3. e_loopb- : sets the Ethernet chip into loopback mode

4. e_reset- : resets the Ethernet controller.

U1405 read buffer's outputs are held static to the Ethernet controller until the buffer is either cleared or new data is clocked into it by the rising edge of wr_ether-. U1405 is cleared (all Q outputs are set to zero) by the assertion of the init- signal.

U1407 Ethernet Control Read Buffer

U1407 ALS 373 acts as a read-back register to the write outputs of the U1405 write register. The four control signals from U1405 are connected to U1407:

1. e_inten- : interrupt enable to the controller
2. e_ce : chip enable to the controller
3. e_loopb- : sets the Ethernet chip into loopback node
4. e_reset- : resets the Ethernet controller.

Also connected to the U1407 read register are the signals:

1. e_int : interrupt signal from the Ethernet controller
2. e_err : error signal from the Ethernet chip

Since the interrupt and error signals occur asynchronously to the processor, they must be synchronized through U1407. Data at the input(s) of U1407 are latched to the TTL data bus (t_d[7:0]) by the assertion of the rd_ether- strobe from U1401. When the rd_ether- strobe is high (deasserted) the U1407 read buffer is set to a high impedance state.

25.7. System Enable Register

The U1406 and U1408 system enable register operates the same as the Ethernet control register. Write data is clocked into U1406, and U1408 acts as a read-back register.

U1406 System Enable Write Register

Write data is clocked from the TTL data bus into the U1406 register on the rising edge of the wr_sysen- signal, which goes inactive at the end of the write cycle. Write signals output from U1406 are:

1. fpaen : enable signal to the FPA
2. encopy : enables copy mode to the video buffer
3. envideo : enables output from the video section
4. en_cache: enables cache memory
5. en_sdvma : enables supervisor DVMA
6. en_fpp : enables the floating point processor
7. en_boot- : enables boot state.

U1408 System Enable Read Register

U1408 is used as a read-back register for system enable data. When rd_sysen- goes low, system enable data is latched by U1408 to the TTL data bus. Since the user diagnostic switch can be pushed at any time (asynchronous), an ALS 373 is used to synchronize this event. The init- signal clears the Q outputs of the register to zeroes on power up.

25.8. U1410 Diagnostics Register

The user diagnostics register clocks data from the TTL data bus into the bank of 8 LEDs when wr_diag_ goes from low to high at the end of the write cycle. The init- signal clears the Q outputs of the register to zeroes on power up.

25.9. U1409 ID PROM

The ID PROM is a read-only device, which outputs byte-wide data onto the TTL data bus. The PROM is organized 32 bytes by 8 bits, thus five address lines (p2_a[04:00]) are used to access any one of these 32 bytes.

Information contained in the PROM includes machine type, a unique serial number for software licensing, a unique Ethernet address, the date of machine manufacture, and a checksum. Additionally, the ID PROM stores configuration information for the machine. The ID PROM is located in Control Space, at address A31:28 = 0x0, in other words:

```
IDPROM = /p_a31*/p_a30*/p_a29*/p_a28 = 0
```

Contents of the PROM are organized in the table below:

Table 25-5 *Contents of the ID PROM*

Entry Number	Contents†	Length (in bytes)
1	Format	1
2	Machine Type	1
3	Ethernet Address	6
4	Date	4
5	Serial Number	3
6	Checksum	1
7	Reserved	16

Output of the PROM is enabled when the rd_id- signal is active (low).

25.10. U1404 P2-to-TTL Data Buffer

U1404 buffers P2 data to and from the TTL data bus. The "A" side is connected to the TTL data bus, while the B side is connected to the P2 data bus, bits 31:24. Output is enabled by the assertion of ttlbfen- from U1400; the direction of the data flow is controlled by the p2_rw signal. When the p2_rw signal is high (read) and ttlbfen- is true, data flows from the TTL data bus to the P2 data bus. When p2_rw is low (write cycle) and ttlbfen- is true, data flows from the P2 data bus onto the TTL data bus. The table below codifies this:

†These contents are described further in the Sun-3 Architecture manual.

Table 25-6 *U1404 P2-to-TTL Data Buffer — Data Flow*

Gate *ttlbfen-*	Direction *p2_rw*	Which way the data will flow
0	0	P2 data bus to TTL bus (B -> A)
0	1	TTL data bus to P2 bus (A -> B)
1	X	*tri-state*

25.11. U2905 and U2906 User DVMA Enable Register

The User DVMA register has a write section (U2905) and a read section (U2906). The write register has data at its inputs clocked out of it by the rising edge of wr_usren- (which goes high at the end of the write cycle). Data clocked out is connected to the enable context bus, en_cx(7:0). When asserted, the init- signal clears all the Q outputs to a low.

U2906 read register has data at its inputs (from the enable context bus) latched and output-enabled with the assertion of rd_usren-. This data is coupled to the TTL data bus, t_d(7:0). U2906 thus operates as a read-back register of the enable context bus.

25.12. U203 Bus Error Register

U203 is a read-only bus error register, which latches the status of the various error signals to the TTL data bus. Data at the input to the register is stored on the low-to-high transition of lberr (load bus error), and this data is coupled to the TTL data bus when the output enable signal, rd_berr-, goes low. When rd_berr- is high, the register goes to a high impedance tri-state.

25.13. U509 Context Register

U509 Context register holds present status as either user or superviser status, and multiplexes between this present status and the "user context" provided by the VME section of the 2060 board. U509 latches data from and outputs data to the TTL data bus.

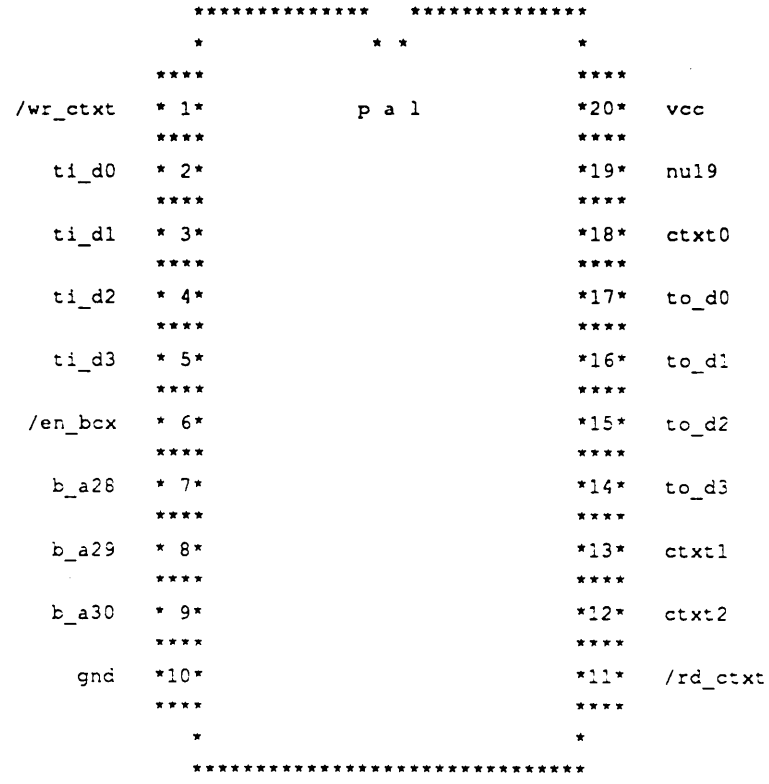
Instead of a buffer/register, the context register consists of a PAL. Data at the I(3:0) inputs are clocked into the PAL by the rising edge of wr_ctxt-, at the end of the write cycle, and this data is latched into an internal register; rd_ctxt- enables this output from the PAL. Outputs O(3:0) are held static until output is disabled — when rd_ctxt- is deasserted.

The en_bcx signal multiplexes between the two input data ports, t_d(3:0) and b_a(30:28). When the en_bcx- signal is true, it enables the user context data at b_a(30:28) onto the ctxt(2:0) bus. When en_bcx- signal is false, the normal TTL bus data, t_d(3:0), is enabled onto the ctxt(2:0) bus.

U509 Pinout

Pinout of the U509 PAL is:

Figure 25-6 *U509 Pinout*



Inputs and Outputs of U509 Context Register

Inputs to the context register are:

```
t_d[3:0]    = TTL data bus
                (inputs for writes to context register)

en_bcx-     = enable context bus (from VME circuitry)

b_a[30:28] = "user context" information from VMEbus
```

Outputs of the context register are:

```
ctxt[3:0] = multiplexed context value that is input
            to the segment map rams

to_d[3:0] = TTL data bus (outputs for reads
            from context register)
```

Eight user contexts are decoded from input bits b_a[30:28]; these bits are aliased in the PAL equations as:

```
UCTXT0 = b_a28
UCTXT1 = b_a29
UCTXT2 = b_a30
```

The Q3:0 outputs are latched externally to the I3:0 inputs, as represented in the PAL equations:†

```
/to_d0  :=      /ti_d0
/to_d1  :=      /ti_d1
/to_d2  :=      /ti_d2
/to_d3  :=      /ti_d3
```

These signals must be inverted because outputs of the PAL are inverted.

The derivation of the three context bits UCTXT2:0 is given below:

```
/ctxt0  =       en_bcx * /UCTXT0 +
               /en_bcx * /to_d0

/ctxt1  =       en_bcx * /UCTXT1 +
               /en_bcx * /to_d1

/ctxt2  =       en_bcx * /UCTXT2 +
               /en_bcx * /to_d2
```

For instance, ctxt(0) is asserted when

†These labels differ somewhat from the schematics; the PAL equations differentiate between the input and output TTL bus signals.

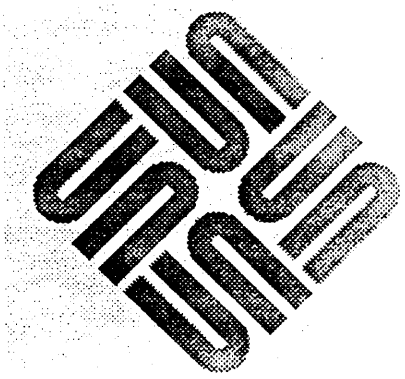
1. en_bcx- is true (low) and UCTXT0 (b_a28 address bit) is also low, or
2. en_bcx- is NOT true (en_bcx is high) and to_d0 (TTL bus data bit 0) is low.

26

Video Circuitry

Video Circuitry 233

26.1. Video Cycle Timing 233
26.2. U1504 Video Select Decoder 234
U1504 Pinout 235
U1504 Input and Output signals 235
26.3. U1502 Video Control Decoder 236
U1502 Pinout 237
U1502 Input Signals 237
U1502 Output Signals 237
26.4. P2 Interface State Machine — U1503, U1605/07 241
U1503 Pinout 241
U1503 Inputs 241
U1503 Outputs 242
26.5. VARB and Video Side State Machines 242
Video Read 243
Video Write 247
Video Write Timing Diagrams: A Real Example 249
26.6. U1501 Byte Decode PAL 250
U1501 Pinout 250
U1501 Output signals 251
U1500 Buffer and U1505 DIP 253
26.7. U1608-U1603 Video Controller 253
26.8. U1700-01 Video RAS/CAS Latches 254



26.9. Frame Buffer RAM 255
26.10. ECL Circuitry 256
ECL Clock 256
26.11. Horizontal and Vertical Synch State Machines 257

26

Video Circuitry

NOTE *Please refer to the video block diagram in the Appendix as you follow the description below.*

Address and control signals from the P2 bus are connected to the U1701:00 address and control latches and the P2 interface state machine (U1503). U1503 issues vsack back to the processor (through the dsack PAL), and also handshakes with the video memory controller on page 16, using the vreq-/busy signals. U1503 state machine also issues the buffer latch signal, vlatch (which is the complement of the vreq- signal), to the P2 data transceiver latches (on pages 18-21).

Processor column address to video memory comes from U1701:00, multiplexed by row or column address strobe enable signals to the output control pins. RAS/CAS/WE signals to the frame buffer are issued by the video memory controller on page 16. Read or write data is connected to/from the P2 data bus through the P2 data transceiver latches (LS 652s on pages 18-21), whose read/write direction is determined by the assertion of vack (read data) or vlatch (write data) signals.

The video refresh counter on page 17 issues the address of the location in the frame buffer which is to be output to the display. It is comprised of a counter (U1705:04) whose output is incremented and latched in a pair of buffers (U1703:02) by either RAS (row address strobe) or CAS (column address strobe). This address is then coupled to video memory on the rcaddr(7:0) bus.

64 bits of data are latched into or out of the LS 652 data transceivers, which are connected to the P2 data bus through the transceivers, or to the video display terminal through the ALS534s and the ECL shifters on page 23 of the schematics.

The video synchronization signals (horizontal and vertical) are issued by their respective state machines on page 22.

The video control state machine includes the two PROMs (U1608 and U1603) and their latches (U1601 and U1604).

26.1. Video Cycle Timing

The video cycle timing is given in the timing diagram labelled "2060 FRAME BUFFER CONTROL."

A video cycle has sixteen 40-nsec states, lasting 640 nsec. The cycle is broken into two halves:

1. the first 320 nsecs services any processor requests (issuing from U1503, etc.),
2. the final 320 nsecs services video update requests.

The processor half is conditional — may or may not occur — but the video update always occurs every video cycle (all the time, or else you will see video trash).

If you are doing a CPU read/write access, the processor issues RAS (row address strobe) from states 2 through 6 and CAS (column address strobe) from states 5 to 8. The processor read/write cycle is completed — data read or written to the frame buffer, vack issued, data latched into data transceivers (if a processor read) and the U1607 busy flipflop is cleared — and the video cycle enters its second half, video update.

If you are *not* doing a CPU read/write access, only the video update cycle occurs.

A processor video access cycle starts with the enable request (enreq) line sampling the busy signal through U1600 AND gate (on page 16), which sets the sreq signal through U1607-1 flip-flop in state 1 of the video cycle. The enreq signal then goes low. The sreq signal stays set until state 9, when the vack- signal from U1604 clears flip-flop U1607-1.

26.2. U1504 Video Select Decoder

Video memory can be located either in a physically-separate frame buffer (address 0xFF000000) or in main memory (address 0x00100000). In either case it takes up 128 Kbytes. The frame buffer can be read-from or written-to directly, just like ordinary memory. However, when a "copy-mode" write is executed, the enable copy bit from the system enable register (U1406) is set. Information being written into the specific 128 Kbyte "copy region" set aside in main memory for this purpose (starting at address 0x00100000), is also written into the separate frame buffer. A read from the copy region returns data from main memory, and does not affect video memory.

U1504 is an address space decoder; from its address bit inputs, mmu_a(31:17), one of two signals are issued:

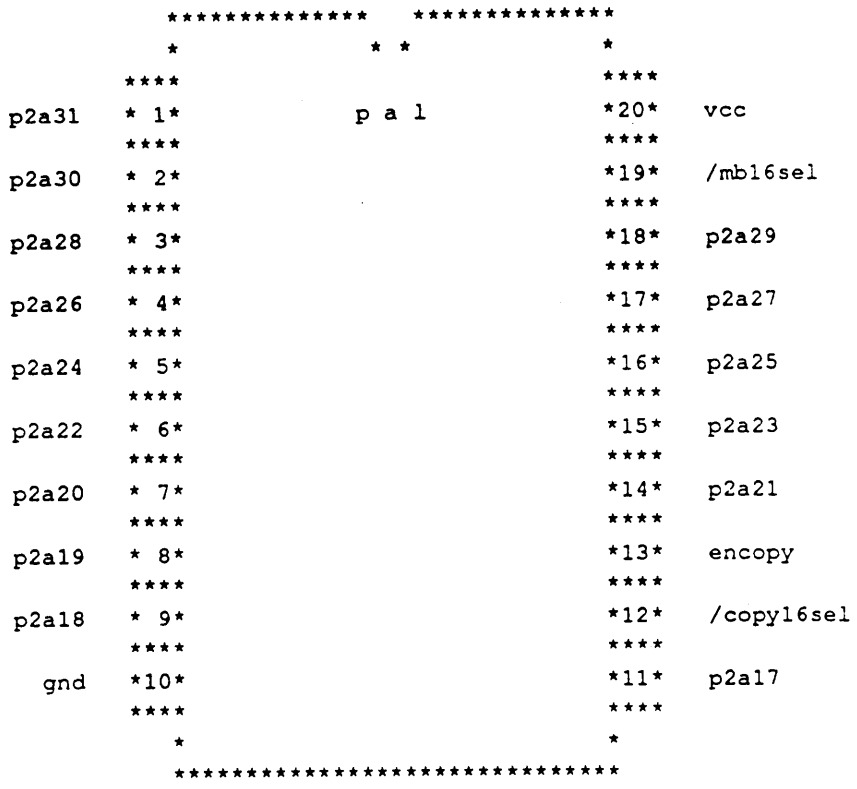
- mb16sel- : indicates an access is to be made to the top 16 Mbyte space (which is where the frame buffer resides) of the 4 Gbyte physical address space and is used to qualify reads from the video memory and VME accesses. This signal is used for normal video reads and writes.
- copy16sel- : indicates an access is to be made either to the top 16 Mbyte space of the 4 Gbyte physical address space, or to the 1 Mbyte + 128K copy mode space. This signal is used to qualify writes to the video memory.

This PAL must be of the 15 nsec variety.

U1504 Pinout

Pinout of the U1504 PAL is:

Figure 26-1 *U1504 Pinout*



U1504 Input and Output signals

Inputs to the U1504 Decoder are:

```
mmu_a(31:17)

encopy
```

Outputs of the U1504 decoder are:

```
/copy16sel

/mb16sel
```

Equations for the two output signals are:

```
mb16sel = mmu_a31*mmu_a30*mmu_a29*mmu_a28*mmu_a27*mmu_a26* mmu_a25*mmu_a24
```

The above equation says that the mb16sel- signal is asserted if mmu_a(31:24) bits are all high (top 16 Mbytes of the 4 Gbyte address space selected), meaning the address is 0xFFxxxxxx.

```
copy16sel = mmu_a31*mmu_a30*mmu_a29*mmu_a28*mmu_a27*mmu_a26*mmu_a25*mmu_a24 +

            /mmu_a31*/mmu_a30*/mmu_a29*/mmu_a28*/mmu_a27*/mmu_a26*/mmu_a25*/mmu_a24*
            /mmu_a23*/mmu_a22*/mmu_a21*mmu_a20*/mmu_a19*/mmu_a18*/mmu_a17*encopy
```

The copy16sel- signal is asserted if:

1. mmu_a(31:24) bits are all high, 0xFFxxxxxx, (top 16 Mbytes of the 4 Gbyte address space selected), or if
2. 0x001xxxxx (mmu_a20 high and all others in the equation are low) address space in main memory is selected, and copy mode is enabled (encopy true).

The copy16sel signal is actually an OR of the mb16sel term with a copy-mode enable term.

26.3. U1502 Video Control Decoder

The two select signals from U1504 are connected to (among others) the U1502 video control decoder, which is a multi-purpose strobe decoder. The signals p2rden0 and p2rden1 are output enables to gate either

- bits 63:32 (p2rden0) or
- bits 31:00 (p2rden1)

of the video memory onto the p2_d(31:00) data bus.

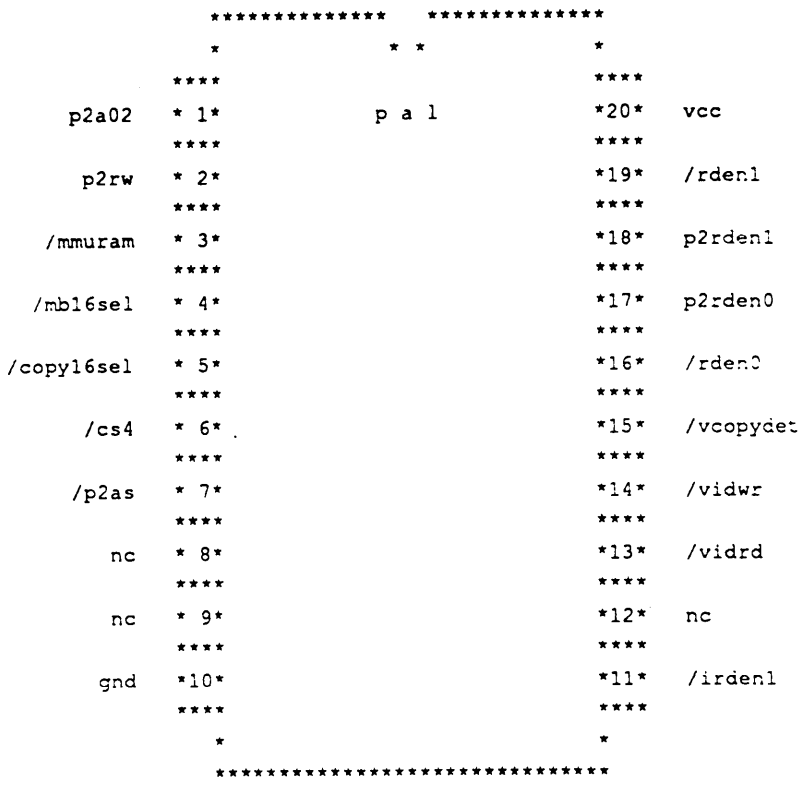
During any read of the video memory, all 64 bits of the video memory are latched. The p2_a02 input signal determines which 32-bit word is to be read, upper or lower — when p2_a02 is low, it selects the upper data word, when high it selects the lower data word.

The vcopydet- signal is used by the 68020 DSACK generator. When vcopydet- is asserted, DSACK will not be asserted until both p2 memory and video memory respond to a copy mode write.

U1502 Pinout

Pinout of the U1502 PAL is:

Figure 26-2 *U1502 Pinout*



U1502 Input Signals

The two video select signals decoded by U1504, copy16sel- and mb16sel-, are connected to the U1502 video control decoder. Other inputs are:

```
p2rw      = read/write strobe

/mmuram   = indicates you are accessing
            memory through the MMU

p2_a02    = bit used to select upper or lower
            data word from 64-bit video bus
```

U1502 Output Signals

Outputs from the U1502 decoder are:

```
p2rden0   = read enable for upper video data word

p2rden1   = read enable for lower video data word

/vcopydet = video copy detect signal, used to
            make certain both P2 memory and video
            memory are ready before processor
            issues a DSACK.
```

If you look on pages 18-21 of the schematics, you will find the video memory RAM. Notice that the read output enables on the data latches are connected to either p2rden0 (for the upper data word) or p2rden1 (lower data word).

The equations for these two signals, p2rden0 and p2rden1, are below.

```
/p2rden0 = /rden0

   (since the output enable of the LS652
    is active high, rden0 must be inverted)
```

```
rden0 =  mb16sel * p2rw * mmuram * /p2a02 * cs4 +

         rden0' * p2as
```

This equation tells us that p2rden0, the read enable strobe for the upper 32 bits of video data, is issued when rden0 is issued. The rden0 signal is issued when

1. the mb16sel signal is true (you are doing an access to the upper 16 Mbytes of the 4 Gbyte physical address space); you are in a read cycle (p2rw is high); you are accessing a TYPE0 space device — memory and the frame buffer; mmuram- is true; p2a02 is low; and you are in clock state 4 (address and control signals are stable); or,
2. rden0 is true and until p2as- goes false (self-latching mechanism to ensure the read cycle lasts beyond the deassertion of cs4). When p2as- goes false (high), this self-latching mechanism is aborted.

```
/p2rden1 = /irden1

    (since the output enable of the LS652
    is active high, rden0 must be inverted)
```

```
rden1 =  mb16sel * p2rw * mmuram *  p2a02 * cs4 +

         irden1 * p2as
```

This equation tells us that p2rden1, the read enable strobe for the lower 32 bits of video data, is issued when irden1 is issued. The irden1 signal is issued when

1. the mb16sel signal is true (you are doing an access to the upper 16 Mbytes of the 4 Gbyte physical address space); you are in a read cycle (p2rw is high); mmuram- is true; p2a02 is high; and you are in clock state 4 (address and control signals are stable); or,
2. irden1 is true and until p2as- goes false (self-latching mechanism to ensure the read cycle lasts beyond the deassertion of cs4). When p2as- goes false (high), this self-latching mechanism is aborted.

The video copy detect signal is issued when you are going to execute a copy-mode write—writing to both video memory and main memory. Should the frame buffer be busy when you want to initiate a copy-mode write, vcopydet- will be issued, which in turn holds off the CPU from issuing a dsack until vsack is issued. This ensures both video and main memory are done (vsack and p2_ack have been issued to U204 DSACK PAL) before you end a copy-mode write.

```
vcopydet = copy16sel * /mb16sel * /p2rw * mmuram * cs4 +

           vcopydet * p2as
```

This equation says that video copy detect is true when

1. copy16sel is true and mb16sel is NOT true (in essence, you are in copy mode to the 0x00100000 address space); you are in a write cycle (p2rw low) to memory (mmuram- is true) during clock state 4 (cs4 true); or
2. the vcopydet- signal is true until p2as address strobe goes false.

This latter term is a self-latching mechanism to make sure the vcopydet- lasts as long as long as the address strobe is not deactivated when cs4 goes inactive. When p2as goes false (high), this self-latching mechanism is aborted.

The logical AND of copy16sel*/mb16sel means that access to the top 16 Mbytes is NOT activated (mb16sel term is removed from copy16sel term), which leaves just the copy term asserted.

```
vidrd = mb16sel * p2rw * mmuram * cs4 +

        vidrd * p2as
```

The equation above indicates that video read (vidrd-) is active (low) for the length of the video read cycle. Deactivation of vidrd- indicates that the video read cycle has been completed. Thus vidrd- is true when:

1. mb16sel is true (doing an access to the top 16 Mbytes of the 4 Gbyte physical address space); p2rw is high (indicated a read cycle); mmuram- is true (accessing a TYPE0 space device — main memory or frame buffer) during clock state 4; or
2. video read is true until p2as address strobe goes false.

This latter term is a self-latching mechanism to make sure vidrd- is not deactivated when cs4 goes inactive. When p2as goes false (high), this self-latching mechanism is aborted.

```
vidwr = copy16sel * /p2rw * mmuram * cs4 +

        vidwr * p2as
```

Video write (vidwr-) is active for the length of the video write cycle. Note that copy16sel is used instead of mb16sel, because you can do writes to both the frame buffer video memory and the copy region of main memory. Therefore, vidwr- is true when:

1. copy16sel is true; during a write cycle (p2rw low); mmuram- is true (accessing a TYPE0 space device — main memory or frame buffer) during clock state 4; or,
2. video write is true until p2as address strobe goes false.

This latter term is a self-latching mechanism to make sure vidwr- is not deactivated when cs4 goes inactive. When p2as goes false (high), this self-latching mechanism is aborted.

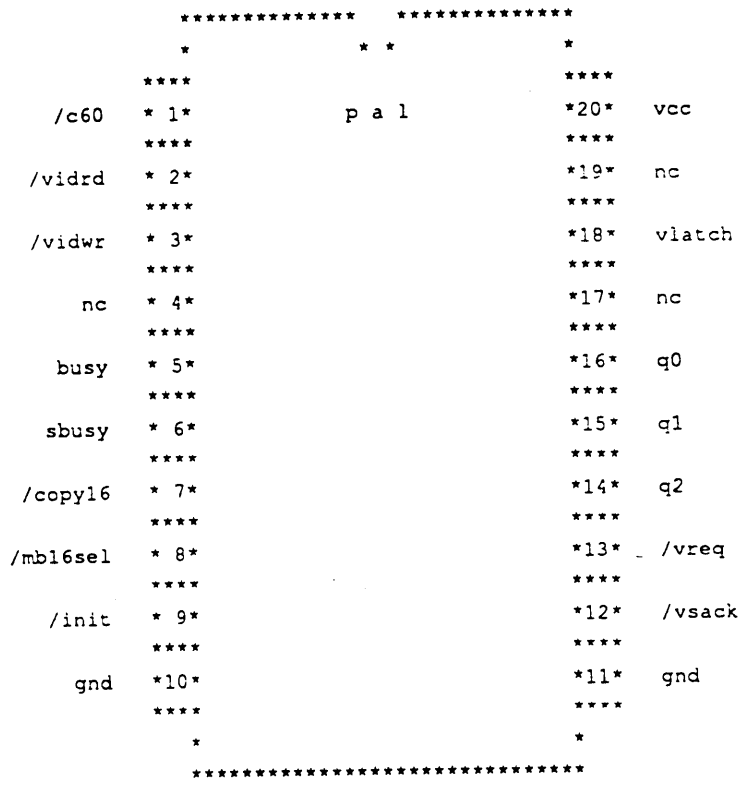
26.4. P2 Interface State Machine — U1503, U1605/07

U1503 issues the control signals which interface the video circuitry to the P2 bus. It has a state machine internal to itself (labelled "VARB") which handshakes with another (labelled "Video Side") which consists of flip-flops U1605 and U1607. This handshaking mechanism is described below.

U1503 Pinout

Pinout of the U1503 PAL is:

Figure 26-3 *U1503 Pinout*



U1503 Inputs

Inputs to the PAL are:

```
vidrd-  = Valid video read strobe

vidwr-  = Valid video write strobe

busy    = Video frame buffer busy

sbusy   = Synchronized version of vbusy to 68020 clock

init-   = Reset input

p2_as-  = not used
```

503 Outputs

Outputs from the PAL are:

```
vsack- = Video dsack†

vreq-  = 68020 video operation request†

vlatch = combinatorial output; vlatch is the
         inversion of vreq-, used to latch
         various buffer/registers.
```

26.5. VARB and Video Side State Machines

Take a look at the two state machines, labelled "VARB" and "Video Side." They work in conjunction with each other, through a trio of handshaking signals — vreq and busy/sbusy.

A cycle begins with both state machines in IDLE states.

† These outputs are not latched and are associated with state as in the Moore model.

```
state idle

!init-                          power on reset
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

!vidrd-
        -> state rdservreq,

           vsack- = 1,
           vreq- = !(!vidrd- & !busy);
                video read request and NOT busy
else

!vidwr-
        -> state wrserv,

           vsack- = vidwr-,
           vreq- = !(!vidwr- & !busy);
                video read request and NOT busy
else

always
        -> state idle,

           vsack- = vidwr-,
           vreq- = !((!vidrd- + !vidwr-) & !busy);
```

The init- signal (labelled "sanity" in the state diagram), is a power-on reset which sets the state machines to IDLE.

In IDLE, U1503 (VARB state machine) has vsack- pacing the condition of vidwr- (both are high/inactive). The vreq- signal goes active (low) when vidrd- or vidwr- and NOT busy are true, indicating a processor video cycle request.

While in IDLE, U1503 is going to receive either a video read (vidrd-) or a video write (vidwr-) from U1502, which signals the start of a bus cycle. When either of these two signals goes active (low) the state machine goes out of IDLE state into a read or a write service state. Let's say the signal received was a video read.

Video Read

In a video read cycle, U1502 issues a vidrd- signal to U1503. This moves the state machine to RDSERVRQ state, which does two things:

1. keeps vsack- at a one (deactivates it), and
2. keeps vreq- at the state of vidrd- and NOT busy, which is a low (indicating that the processor wants to do a video cycle).

This is described in the following equation:

```
state rdservreq

!init-                          power on reset
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

sbusy
        -> state rdserv,    video side has received a
            request so go to readservice state
           vsack- = 1,
           vreq- = 1;
else

always
        -> state rdservreq,  wait for video side to
             receive a request
           vsack- = 1,
           vreq- = !(!vidrd- & !busy);
```

The vreq- signal is part of half of the handshaking mechanism that connects the VARB state machine with the Video Side state machine. If you follow vreq- out of U1503, you will see it connects to pin 2, the D input of U1605-0 flip-flop.

Power on reset (init-) has set U1605-0 flip-flop, putting a low at its inverted output (Q NOT, pin 6). This same init- signal has cleared U1607-0. Both flip-flops are clocked by 40 nsec video clock (vclk40), and U1605-0 stays set until vreq- goes low. This is the IDLE state of the Video Side state diagram.

When vreq- goes low and is presented to the D input of U1605-0, this flip-flop is cleared at the next 40 nsec clock and the inverted output (a logical high) is taken from pin 6. This is the SYNC state of Video Side state diagram, named because it synchronizes the vreq- to the 40 nsec video clock.

The next 40 nsec vclk40 clocks this high through U1607-0 flip-flop, which asserts the busy signal at pin 6. This puts the Video Side state machine in the BUSY state.

Busy itself stays true (high) and is qualified through U1600 AND gate by the assertion of enreq — request enable — by the U1603 video memory controller. The output of U1600 AND gate is clocked through U1607-1 by 40 nsec vclk40, and emerges as synchronized request, sreq, which is connected to the video memory controller state machine, U1603. The enabling of sreq through the AND gate by the enreq signal moves the Video Side state machine into SERVICE state. U1607 flip-flops are cleared by vack- from U1604:3, and this takes the Video Side state machine out of SERVICE state and moves it back to IDLE.

Since busy is clocked through U1607 by 40 nsec vclk40, it is essentially asynchronous to the U1503 PAL, which is operated by 60 nsec clock. Therefore busy must be synchronized to the PAL, by c60 in U1605-1.

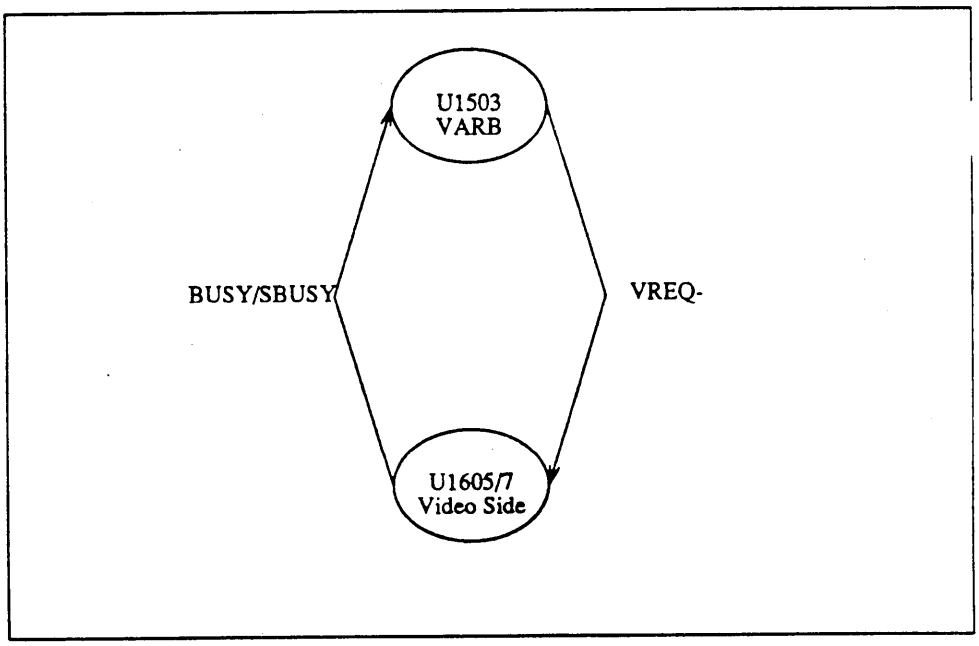
In the meantime, the VARB state machine has been patiently waiting to receive the high active busy signal from the Video Side state machine, U1607-0. This high active busy is connected to U1503, which immediately deactivates the vreq- signal, setting vreq- to a high.

To recapitulate thusfar:

1. a video read, vidrd-, has been issued to U1503;
2. vreq- is issued by U1503
3. vreq- is clocked through U1605, and then U1607 to emerge as busy
4. busy connects back to U1503, deactivating vreq-
5. the VARB state machine is in RDSERVRQ state.

This then is the handshaking mechanism between the two state machines: vreq- from VARB to Video Side, and busy/sbusy from Video Side back to VARB.

Figure 26-4 *Handshaking in the P2 Interface State Machine*



The vreq/busy handshaking is used to make certain that vreq- is held active until busy is issued, which means the read request is being serviced.

When sbusy occurs, the VARB state machine goes from RDSERVRQ state to RDSERV state.

```
state rdserv

!init-                              power on reset
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

!sbusy
        -> state rdend,    NOT busy so go to readend state

           vsack- = vidrd-,
           vreq- = 1;
else

always
        -> state rdserv,   wait for NOT busy

           vsack- = !(!busy),
           vreq- = 1;
```

During RDSERV state, the state machine waits for the return of the NOT busy signal and the Video Side state machine to progress through BUSY to SERVICE to IDLE. The vsack- signal goes active when NOT busy goes true. The vreq- signal is still a high (false). When NOT busy occurs, the data is stable on the bus, and the state machine moves into RDEND (read end) state.

```
state rdend

!init-                          power on reset
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

vidrd-
        -> state idle,  read inactive (cycle over) so go to idle state

           vsack- = 1,
           vreq- = 1;
else

always
        -> state rdend,  read cycle still active so wait

           vsack- = vidrd-,
           vreq- = 1;
```

Remain in RDEND until the read request is gone then go to the IDLE state. The vreq- signal is still inactive (high). Also, vsack is still the same level as vidrd-, which is active until p2_as goes high, which indicates the end of the read cycle. At this time the state machine returns to IDLE, with vsack and vreq high (inactive).

Video Write

The video write cycle begins with the state machine in IDLE. The busy signal is not active (it is low) and vreq and vsack are high (false). When a vidwr- comes into U1503 PAL, vreq- is issued, and clocked through U1605-0 flip-flop. This starts handshaking between the VARB and Video Side state machines (see the description of handshaking in the Video Read section, above). The vsack- signal is issued to end the write bus cycle. Write data and address are latched on vlatch (which is inverted vreq-) so the processor need not wait until the data is written into the video RAM.

```
vlatch = /vreq-
```

The vlatch signal is an inversion of the vreq- signal. It must be inverted because it is used to latch data into ALS 374 buffers, and these need a low-to-high transition. The vlatch signal latches the size and address bits along with the p2_rw signal in U1500 ALS 374; it also latches address bits into U1700 and U1701, and latches write data into the LS 652 transceivers on pages 18-21.

The vidwr- signal also causes the state machine to enter WRSERV state, and the Video Side state machine goes from IDLE through the following states:

1. SYNC : in which vreq is synchronized to 40 nsec vclk40 in flip-flop U1605-0;
2. BUSY : in which the busy signal is clocked out of U1607-0;
3. SERVICE: in which the busy signal is enabled through U1600 AND gate by the enable request signal (enreq) from U1603/4;
4. the flip-flops are cleared by vack-, and the Video Side state machine returns to IDLE.

The busy signal is re-synchronized to 60 nsec clock through flip-flop U1607-1 and emerges as sbusy, which is used back in U1503 VARB state machine.

The VARB state machine enters WRSERV by the following equation:

```
state wrserv

!init-                                    power on reset
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

vidwr- & sbusy
        -> state wrend,    write done and busy so
                  go to the write end state
           vsack- = 1,
           vreq- = 1;
else

always
        -> state wrserv,

           vsack- = vidwr-,
           vreq- = !(!vidwr- & !busy);
```

In WRSERV state, vsack- paces vidwr- (both are low). An immediate vsack- is issued from U1503; vreq- stays low because busy is false (low) and vidwr- is low (true). When the busy signal is coupled back to U1503, it disables vreq- (vreq- goes high)

```
state wrend
 !init-                    power on reset
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

!sbusy                     busy so go to the idle state
        -> state idle,

           vsack- = 1,
           vreq- = 1;
else

always                     NOT busy so wait
        -> state wrend,

           vsack- = 1,
           vreq- = 1;
```

When vidwr- goes high and sbusy is true, the state machine enters WREND (write end) state, indicating the end of the write cycle. Both vsack- and vreq- are false. The state machine remains in WREND until the frame buffer has serviced the write request (vack- returns true from the frame buffer, the Video Side state machine goes to IDLE, and the Video Side flip-flops are cleared). When sbusy goes low (busy from U1607-0 has been cleared by vack- and is clocked through U1607-1) indicating that the video state machine is no longer tied up doing a write operation, the VARB state machine returns to IDLE.

```
state idle

       vsack- = 1
       vreq- = 1
```

Should the processor start another video write cycle while in WREND state, vsack- and vreq- are suppressed and wait states occur until the video write is done and the data/address latches are available for the waiting cycle.

Video Write Timing Diagrams: A Real Example

An actual logic analyzer tracing of a video write cycle is available in the appendix. Note that the c60 clock is really *inverted* c60 clock — c60 bar. Therefore a transition from clock state to clock state occurs on a low-to-high edge.

A write cycle starts during cs4 (the stretched portion of c60). The vidwr- signal is issued from U1502 during this clock state and one PAL delay (15 nsec) later both vsack- and vreq- drop (go true). The state machine is still in IDLE.

The next upward transition of c60 clock the state machine enters WRSERV state. The busy signal comes back from U1607-0, and invalidates vreq-. The high active sbusy signal is clocked out of U1607-1 into U1503. When vidwr- goes high (inactive) vsack- is disabled and on the next c60 clock (rising edge in the timing diagram) the state machine enters WREND. The vack- signal is issued by the frame buffer which clears the busy flip-flop, which in turn clears the sbusy flip-flop. When sbusy goes low (inactive) the state machine returns to IDLE.

Note that a second processor write cycle starts (sample 99) while the video state machine is still doing the first write. In this case, vreq- and vsack- are not issued until the write is finished (IDLE state).

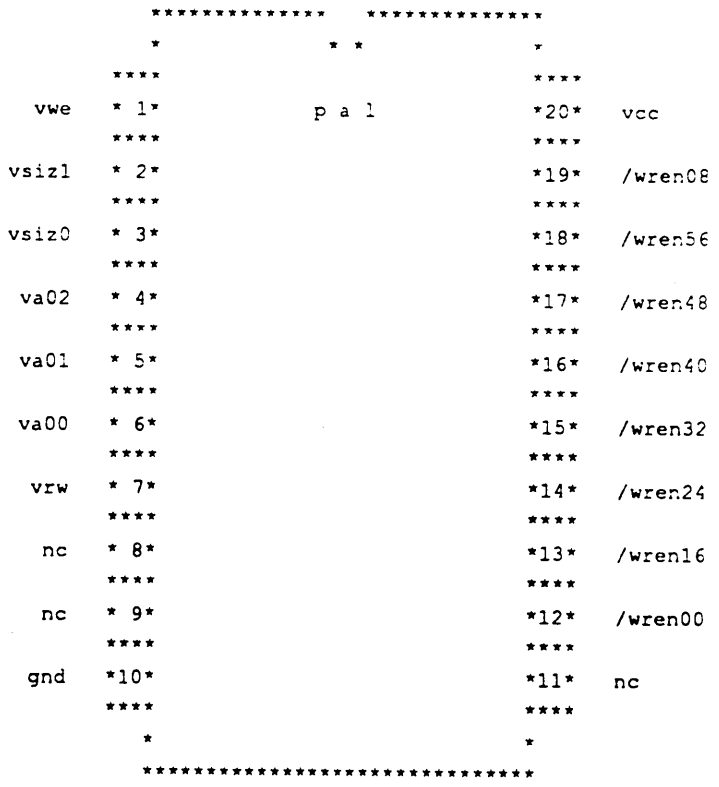
26.6. U1501 Byte Decode PAL

This PAL generates the write strobes for the video memory 4416 rams. Since the video memory is 64 bits wide, va02 (latched version of p2_a02) is used to select the four write enable signals wren(24:16:08:00) or wren(56:48:40:32) and accounts for the symmetry of the equations. These strobes will only be enabled if the memory write signal, vwe, is true and the latched version of p2_rw, vrw, is low indicating a write cycle. The size (vsiz1:0) and address line (va01:00) decode equations allow byte, word, 3-byte, and longword transfers, with offset.

U1501 Pinout

Pinout of the U1501 PAL is:

Figure 26-5 *U1501 Pinout*



U1501 Output signals

Output signals of the U1501 PAL are:

```
wren00 = vwe * /vrw * /va02 * /vsiz1 * /vsiz0 +
         vwe * /vrw * /va02 * va01 * va00 +
         vwe * /vrw * /va02 * vsiz1 * vsiz0 * va00 +
         vwe * /vrw * /va02 * vsiz1 * va01
```

The wren00- signal write enables data bits [39:32] in U1908.

```
wren08 = vwe * /vrw * /va02 * va01 * /va00 +
         vwe * /vrw * /va02 * /vsiz1 * /vsiz0 * /va01 +
         vwe * /vrw * /va02 * vsiz1 * vsiz0 * /va01 +
         vwe * /vrw * /va02 * vsiz1 * /va01 * va00
```

The wren08- signal enables data bits [47:40] in U1907.

```
wren16 = vwe * /vrw * /va02 * /va01 * va00 +
         vwe * /vrw * /va02 * vsiz1 * /va01 +
         vwe * /vrw * /va02 * /vsiz0 * /va01
```

The wren16- signal enables data bits [55:48] in U1808.

```
wren24 = vwe * /vrw * /va02 * /va01 * /va00
```

The wren24- signal enables data bits [63:56] in U1807.

```
wren32 = vwe * /vrw *  va02 * /vsiz1 * /vsiz0 +
         vwe * /vrw *  va02 * va01 * va00 +
         vwe * /vrw *  va02 * vsiz1 * vsiz0 * va00 +
         vwe * /vrw *  va02 * vsiz1 * va01
```

The wren32- signal enables data bits [07:00] in U2108.

```
wren40 = vwe * /vrw *  va02 * va01 * /va00 +
         vwe * /vrw *  va02 * /vsiz1 * /vsiz0 * /va01 +
         vwe * /vrw *  va02 * vsiz1 * vsiz0 * /va01 +
         vwe * /vrw *  va02 * vsiz1 * /va01 * va00
```

The wren40- signal enables data bits [15:08] in U2107.

```
wren48 = vwe * /vrw *  va02 * /va01 * va00 +
         vwe * /vrw *  va02 * vsiz1 * /va01 +
         vwe * /vrw *  va02 * /vsiz0 * /va01
```

The wren48- signal enables data bits [23:16] in U2008.

```
wren56 = vwe * /vrw *  va02 * /va01 * /va00
```

The wren56- signal enables data bits [31:24] in U2007.

U1500 Buffer and U1505 DIP

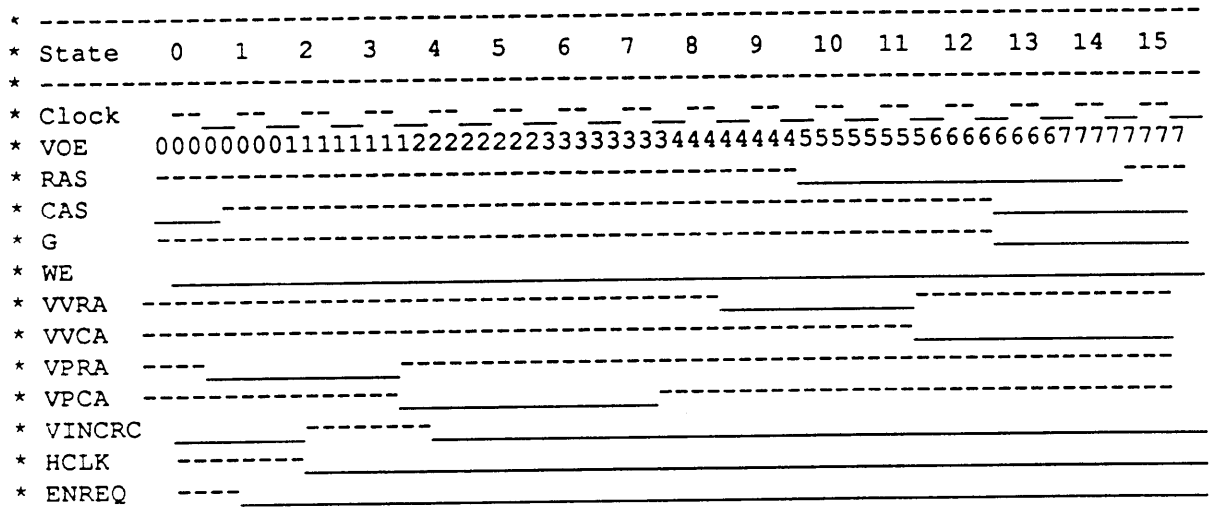
U1505 DIP series-terminates the write enable signals to the frame buffer RAM on pages 18-21. The wren(56:00)† signals are output enables to the LS 652 transceivers, and don't need to be series-terminated. The wren(56:00)† signals output enable data from the P2 data bus onto the video memory data bus (A -> B), to be written into the frame buffer RAM.

U1500 LS 374 latches size, address, and read/write control signals from the P2 bus on the rising edge of rvlatch from U1503 PAL through U1609 RDIP. These size, address and read/write bits are decoded by U1501 to issue a specific read or write enable strobe to the frame buffer.

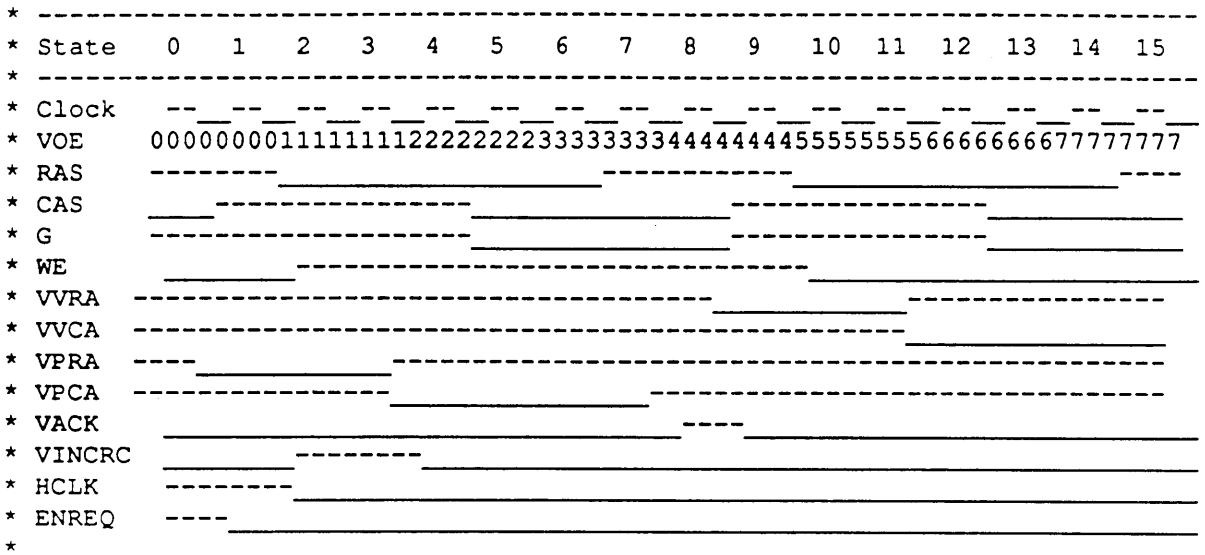
26.7. U1608-U1603 Video Controller

The video state machines perform an optional read or write access to the frame buffer memory followed by a video update read cycle. The basic memory cycle consists of 16 states; the state machine is clocked every 40 nsec and, hence, repeats every 640 nsec.

The following are timing diagrams for the frame buffer; the first is with no CPU read/write access during the CPU portion of the cycle (first 320 nsecs), and the second includes a CPU read/write access.



†These signals are labelled "wen" in the schematics and labelled "wren" in the PAL listings. Don't be confused. They refer to the same signal.



The video control state machine includes the two PROMs (U1608 and U1603) and their latches (U1601 and U1604). Three state bits, labelled state1, state2 and state3 (along with state0, the TTL version of 80 nsec ECL clock) combine to make a 4-bit state counter which determines video cycle states zero through fifteen (state 0-15 in the timing diagrams immediately above). State0, the TTL version of the 80 nsec ECL clock, provides synchronization between the ECL and TTL state machines. Three of these state bits — state1, state2, and state3 — are also used by the 3-to-8 demultiplexer U1602 to generate one of eight video output enable signals for the ALS 534 output buffers connected to the frame buffer RAM.

These four state bits also generate (through U1603) the video RAS, CAS, write enable, horizontal clock (hclk), enable request (enreq, which qualifies the busy signal through U1600 AND gate), vack, and vgen- (which enables read data from the video buffer RAM).

26.8. U1700-01 Video RAS/CAS Latches

There are separate row and column address strobes for each half of the video cycle:

- vpra and vpca are the row and column address strobes for the processor (first) half of the video cycle;
- vvra and vvca are the row and column address strobes for the video update (last) half of the cycle.

U1700 and U1701 hold the processor RAS and CAS addresses used during the processor half of the video cycle:

- U1700 = RAS address (output controlled by vpra-)
- U1701 = CAS address (output controlled by vpca-).

Both of these registers (U1700 and U1701) have the P2 address latched into them by the rising edge of rvlatch (which is vlatch passed through U1609 RDIP)

coming from U1503 and have their outputs enabled by vpra- (U1700) or vpca- (U1701). When either of these go active during the processor half of the video cycle, the P2 address is presented to the video address bus.

The video update half of the timing cycle (last 320 nsecs) begins at state 8, when vack is true. During this half-cycle, the video refresh counter on page 17 (U1705:4) issues a refresh address through either U1702 (row address) or U1703 (column address) buffers. Either the row or the column address is enabled from the appropriate buffers by vvra- (video row address) or vvca- (video column address). The enabled address is latched onto the rcaddr(7:0) bus through U1707 series terminator resistors. The video refresh counter is cleared by the vclr-, from the vertical state machine on page 22 of the schematics. The counter is not incremented during blanking (by qualifying vrcinc with displen through U1600 AND gate).

26.9. Frame Buffer RAM

Data take two paths from the frame buffer RAM:

1. P2 data bus
2. through the ECL circuitry out to the video display.

The output enable for the video RAM chips is the signal mgen(1:0). However if there is a write to the RAM during the processor half of the video cycle, this signal is over-ridden by the appropriate write-enable signal (mwren[56:00]), effectively disabling mgen(1:0) as an output enable.

The vack- signal latches read data into the LS 652 transceivers; vlatch latches write data into the transceivers. These data are then connected either to (on a read cycle) or from (on a write cycle) the P2 data bus.

The video output latches between the 64-bit frame buffer data bus and the 8-bit ECL converters (on page 23) are multiplexed by one of eight video output enable signals (voe[7:0]) asserted by the video control state machine through U1602 3-to-8-line decoder. The ALS 534 latches invert their outputs to compensate for the ECL output shifters. The ECL shifters' logic needs a one to be a video white and a zero to be a video black. Since this is the complement of the logic input to the 534s, the signals are inverted.

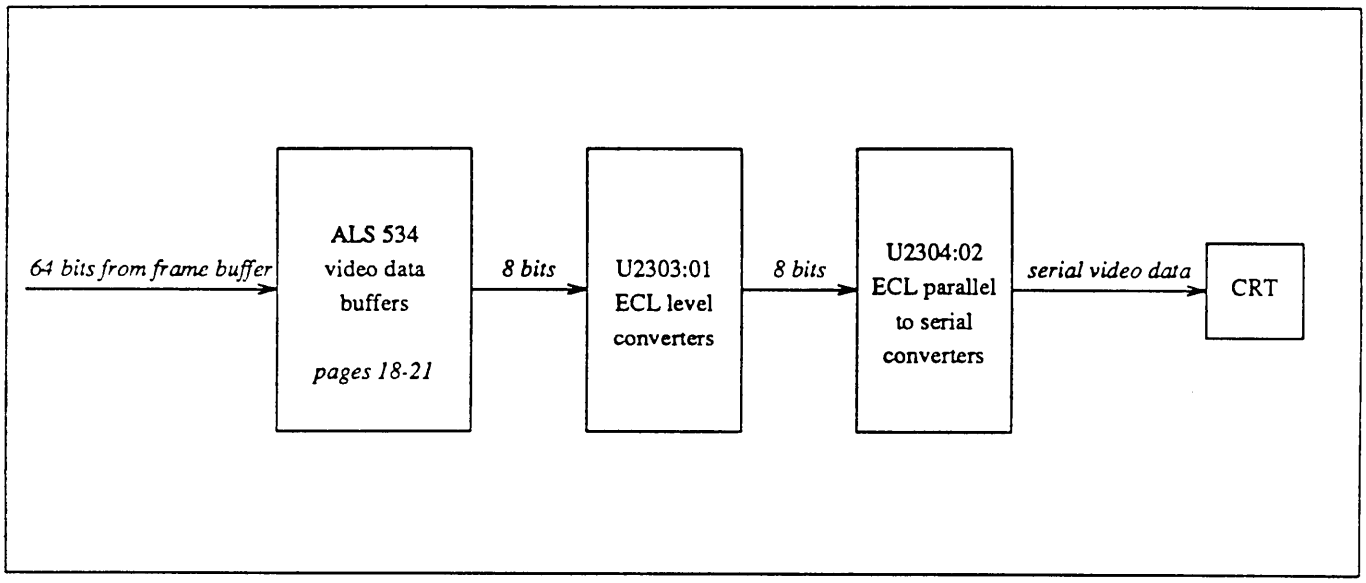
During the last half of the video cycle, 64 bits of data are latched into the 8 ALS 534 latches (at the output of the frame buffer RAM on pages 18-21) by the horizontal video clock signal, hclk, which is active at the end of state 15/beginning of state 0. Only one of these eight buffers will have its output enabled, however, depending upon which output enable signal (voe[7:0]) is issued by the video control machine on page 16. Thus every 640 nsecs 64 bits of data are latched into their appropriate buffers.†

†The fact that this pixel/bit time is 10 nsecs — same as the ECL clock — is a "fundamental wonderfulness" (to quote the engineer) and purely serendipitous.

26.10. ECL Circuitry

Eight bits of TTL video data are shifted out of the frame buffer latch whose output is enabled by one of the eight valid voe[7:0] signals. These eight bits are converted from TTL to ECL signal levels, and then from eight bits of parallel data to serial data, which is fed to the differential inputs of the CRT.

Figure 26-6 *Data Path — From Frame Buffer to CRT*



Serial data taken from the parallel-to-serial converters is passed through a "restoration" flip-flop, U2310, which is toggled at 10 nsec clock rate. Since the 10H141 parallel-to-serial converters are susceptible to noise (which shows up on the monitor as every eighth pixel being slightly darker than the rest), U2310 flip-flop is used to shape and clean up the signal to the differential inputs of the CRT. U2310 also provides the differential signals needed by the CRT to drive the monitor.

R2304 and 2305 resistors are used for differential transmission, and diodes 2304:01 are used for transient suppression — clamp the voltage levels to -5 VDC and ground.

ECL Clock

NOTE *Please see the 2060 video timing diagram (covering the output shifter, clock generator, and blanking) in the appendix.*

The 8 data bits are shifted on 80 nsec boundaries from the 8-bit TTL output buffers (ALS 534s on pages 18-21). These 8 bits are converted to serial data, and shifted serially on 10 nsec boundaries out to the differential inputs of the CRT. The clock for this is derived from U2308 ECL oscillator, with J2301 jumper IN. Output of the oscillator is 10 nsec ECL clock, which is used to generate various clock signals used in video timing, through U2305 and U2312 video clock

generator.

This 10 nsec ECL clock is input to U2305 counter, which is in down-counter mode. ECL 20, 40 and 80 nsec clocks are generated; when its output is 000, the output of the OR gate asserts the ECL load signal, eload-, (true every 80 nsecs) which loads an 8-bit byte into the parallel-to-serial converters. 10 nsec eclk10 is used to shift this byte out, bit by bit.

Also derived from 10 nsec ECL clock is the 40 nsec TTL video clock, vclk40, and the state0 signal. The vclk40 signal is merely the TTL version of eclk40, passed through U2306 ECL-to-TTL converter. The state0 signal is derived from ECL 80 nsec clock, eclk80, and is the complement of the ECL signal (taken from the DIN- input).

To make certain that the ALS 534 output buffers are loaded correctly on 80 nsec boundaries, the state0 signal (complement of 80 nsec ECL clock) is used. This 80 nsec ECL clock (state0 signal) is used to synchronize the video state controller on page 16 with the eload- pulse from U2312 OR gate.

Notice that the chip select input to the U2303:01 level converters is the blank- signal. This is the video blanking signal, generated by the horizontal state machine on page 22 and qualified by the video enable signal from the system enable register on page 14(b). When blank- to the CS inputs of the TTL-to-ECL level converters is true (low), the outputs of the level converters are forced to a zero state (black), which means that nothing will be painted onto the display during retrace period.

During blanking periods the video refresh counter (page 17) is NOT incremented

The timing signals are terminated by R2308 RDIP, which are 120Ω/195Ω pullup/pulldown which give an impedance of about 70Ω, which is about the same impedance of the board traces.

26.11. Horizontal and Vertical Synch State Machines

The horizontal and vertical state machines operate much alike; the horizontal is on the top of page 22 of the schematics, and the vertical is on the bottom. U2203 is the output register for both state machines.

640 nsec hclk starts U2201 counter incrementing, which increments the count into U2202 PROM, the horizontal state machine. The hsynch, display enable, and clear for the horizontal state machine counter are all asserted at the appropriate times by the horizontal timing PROM, U2202.

Horizontal timings are given below, for both the 1152 by 900 pixel display and the 1024 by 1024.

```
Timing:

1 Horizontal state = 64 pixel; 1 pixel = 10 nsec.

             Range      Length    Length  Time
            [State]    [State]   [Pixel] [µsec]
-------------------------------------------------------------
*** 1152 x 900 Display ***
cycle        00..24       25      1600    16.00          HFreq = 62.5 KHz
visible      00..17       18      1152    11.52
invisble     18..24        7       448     4.48
frontporch     ..          0         0        0
hsync        18..19        2       128     1.28
backporch    20..24        5       320     3.20
*** 1024 x 1024 Display ***
cycle        00..24       25      1600    16.00          HFreq = 62.5 KHz
visible      00..15       18      1152    11.52
invisble     16..24        7       448     4.48
frontporch   16..16        1        64     0.64
hsync        17..18        2       128     1.28
backporch    19..24        6       384     3.84
-------------------------------------------------------------
```

The display enable signal, displen, is ANDed at U1600 with the enable video bit from the system enable register to disqualify blanking (when both signals are true). The hsynch signal, which occurs every 16 µsecs (1152 visible plus 448 invisible horizontal pixels times 10 nsec ECL clock), clocks the vertical state machine's counter. Vertical timing is given below:

```
1 state = 1 line = 16.00 µsec  (62.50 KHz)

             Range       Length  Time
            [Lines]     [Lines] [µsec]
-------------------------------------------------------------
*** 1152x900 Display ***
cycle        000..936     937     14992        66.70 Hz
visible      000..899     900     14400
invisble     900..936      37       592
frontporch     ..          0         0
vsync        900..909      10       160
backporch    910..936      27       432
*** 1024x1024 Display ***
cycle        000..1060   1061     16976        58.91 Hz
visible      000..1023   1024     16384
invisble    1024..1060     37       592
frontporch     ..          0         0
vsync       1024..1033     10       160
backporch   1034..1060     27       432
-------------------------------------------------------------
```

U2206 and U2207 vertical state machine counter is incremented by hsync. The count is applied to vertical timing PROM U2208, which asserts various control signals from the vertical state machine: vertical synch, vertical blanking (during vertical retrace), and vclr, which clears the vertical state machine's counter, U2206 and U2207, and also clears the video refresh counter on page 17.

Notice that the vertical blanking signal, vblank, is fed from the output of the state machine register, U2203, back into the horizontal state machine. This causes hblank and vblank to be ORed together. The horizontal state machine's blanking always occurs before vertical blanking; if a vblank is valid after an hblank, it will merely append the vblanking interval to the tail of the horizontal blanking period to produce clean and continuous blanking during vertical retrace.

27

VMEbus — Performance

VMEbus — Performance **263**

27.1. 2060 VME Implementation 263

27

VMEbus — Performance

The 2060 VME interface was designed for the highest data transfer rate possible — given the constraint of the extremely limited printed circuit board space available. Under control of the CPU, the 2060 is capable of transferring up to 8.9 Megabytes/second to or from an external VME device. The 2060 memory is capable of accepting data at a rate of up to 7.8 Megabytes/second when under the control of an external VME master.

27.1. 2060 VME Implementation

MASTER CAPABILITIES

```
Data Bus Size:        D32 MASTER 32/16/8 bit data

Address Bus Size:     A32 MASTER (DYN) 32/24/16 bit addresses

Timeout Option:         TOUT(737) 737 microsecond timeout period

Sequential Access:    None

Interrupt Handler:    IH(1-7)(STAT) Level 1 thru 7,
                        independently jumperable
                        All interrupts use vectors provided by
                        VMEbus interrupters, per the VME spec.

Requester Option:     ROR R(3) Release on request, level 3

Bus Busy Option:      Releases BBSY after AS assertion when
                        releasing bus

Read/Modify/Write:    Will not release VMEbus during
                       Read/Modify/Write cycles
```

SLAVE CAPABILITIES

Data Bus Size: D32 SLAVE (DYN) 32/16/8 bit data

Address Bus Size: A32 SLAVE (DYN) 32/24 bit addresses
(no 16-bit addr.)

Sequential Access: None

Special Access Mode: A high-speed access mode is engaged if the time from DTACK assertion to the next AS and DS assertion is less than 200ns.

Interrupter Options: None

32-bit Slave Addressing: The 2060 responds to the bottom 1 Mbyte by performing DVMA using system function codes and responds to the top 2 Gbytes by performing DVMA using user function codes. Response can be dynamically disabled on 256 Mbyte boundaries.

24-Bit Slave Addressing: The 2060 responds to the bottom 1 Mbyte by performing DVMA, by using the system function codes.

SYSTEM CONTROLLER CAPABILITIES

Clock Option: SYSCLK 16 MHz, jumperable (not used on board)

Arbiter Option: ONE Bus Request/Grant Level 3 only, or External Arbiter

Bus Time Out Module: None

Sysreset Option: SYSRESET MASTER or SYSRESET SLAVE, including manual button

Sysfail Option: Not Monitored

ACfail Option: Not Implemented (ACFAIL is connected to SYSRESET)

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature: 10-40 C

Humidity: 5-90% non-condensing

POWER CHARACTERISTICS

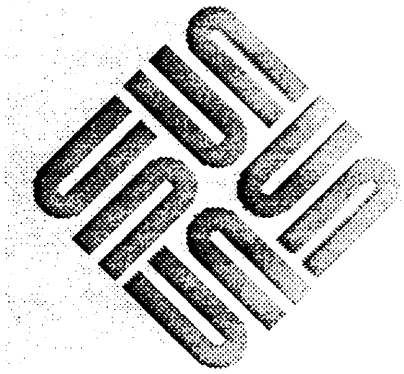
+5 Volts:	14 Amp Max.
-5 Volts:	1 Amp Max.
+12 Volts:	0.5 Amp Max.
-12 Volts:	Not used

28

VME Arbiter and Requester

VME Arbiter and Requester **269**

28.1. Terminology for VME Arbiter and Requester 269
28.2. U2704 VME Arbiter and Requester 271
State Machine as Arbiter and Requester 272
Transitions from BUSREQ State 273
Transitions from MASTER State 274
Transitions from MASTER_NG State 276
Transitions from BUSGRANT State 276
28.3. State Machine as Requester Only 277
Transitions from IDLE State 277
Transitions from BUSREQ State 278
Transitions from MASTER State 279
Transitions from MASTER_NG State 280
Transitions from the BUSGRANT State 281



28

VME Arbiter and Requester

The VME Arbiter and Requester functions are implemented in the synchronous state machine shown on page 27 of the schematics. The Arbiter/Requester state machine (U2704) is responsible for:

- granting control of the VMEbus to the CPU while holding off other devices wishing control of the VMEbus, and
- granting control of the VMEbus to external VME devices when the CPU doesn't wish to access it.

The arbiter and requester functions could have been implemented as separate modules, but this was not done because separating the functions would have introduced extra states into the request process, slowing it significantly and increasing the chip count.

The arbitration function can be locked out by moving shunt J2701 to J2700; this allows the customer to perform arbitration on a separate board, in order that two or more 2060 boards can be installed in the same system for testing or multiple-processor systems. Moving this jumper from J2701 to J2700 leaves the 2060 board operating only as a VMEbus requester, as detailed in the VMEbus Manual. The arbiter/requester state machine operates differently in the two modes, so they are handled separately below. This entire discussion assumes a familiarity with the VMEbus specification, so no attempt is made here to explain the basic workings of the VMEbus.

28.1. Terminology for VME Arbiter and Requester

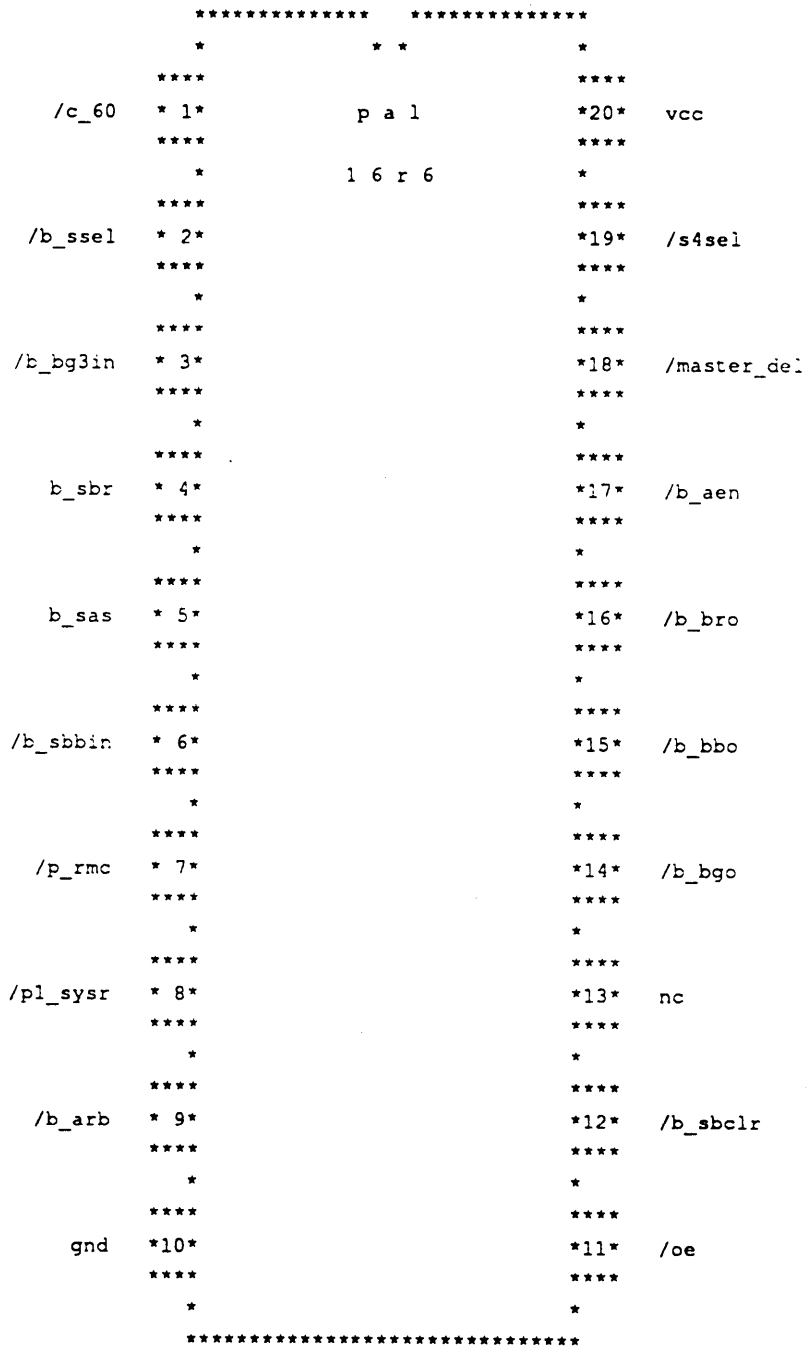
- B_AEN: Originally stood for VME Address Enable, but now the addresses are actually enabled by B_OECPU. B_AEN indicates to the VME Master Controller PAL (U2806) that the 2060 board has control of the VMEbus, causing U2806 to assert B_OECPU.
- B_BBOUT: VMEbus Busy OUT. This signal indicates the 2060 board is driving the VMEbus busy signal, P1_BBSY. It is separated from P1_BBSY by open collector driver U2702.
- B_BG3IN: VMEbus Grant 3 In. When the 2060 is jumpered as the VMEbus arbiter (J2700 out and J2701 in), B_BG3IN is tied to ground to save on terms inside the VME Arbiter/ Requester PAL. When the 2060 is jumpered to be a VME Requester only (J2700 in and J2701 out), this signal is the synchronized form of P1_BG3IN, indicating that the off-board VME Arbiter is granting the VMEbus to the 2060 board.

- B_BGOUT: VMEbus Grant OUT. Electrically the same as signal P1_BG3OUT, this signal indicates that the VMEbus arbiter on the 2060 board is granting control of the VMEbus to an external master. The 2060 has control of this signal only if configured (through jumpers) to be the VMEbus arbiter.
- B_BROUT: VMEbus Request OUT. This signal is buffered by open collector driver U2702 to form P1_BR3, indicating that an external VMEbus master has control of the VMEbus and the 2060 wishes to acquire control.
- B_SBBIN: VME Synchronized Bus Busy IN. P1_BBSY is run through a low-pass filter (C2700 and R2700), then synchronized by c60 clock to form this signal.
- B_SBR: VME Synchronized Bus Request. All four levels of VMEbus request, P1_BR3:0, are logically ORed in U2703, then synchronized by c60 clock to form this signal.
- B_SSEL: Bus Synchronized SELect, indicating that the CPU is accessing the VMEbus. This signal is asserted from U2701 PAL at state 5 if either B_INTA or MMU_VME is active, and is held active during freeze cycles.
- P1_AS: VME Address Strobe
- P1_BBSY: VMEbus BuSY signal, asserted by the current VMEbus master to indicate that it has control of the VMEbus. Control of the VMEbus may not be taken away from this master until it has deactivated P1_BBSY.
- P1_BR3, P1_BR2, P1_BR1, P1_BR0: VMEbus Request signals. The 2060 board issues requests on P1_BR3 only, but will relinquish the bus upon receiving a request on any level. Requests on levels other than 3 will occur in a properly configured system only if the 2060 board is set up so as not to be the VMEbus arbiter.
- P_RMC: Processor Read Modify Cycle. Indicates that the current cycle is part of an indivisible read-modify-write cycle, generally used to coordinate actions between several processors. The VMEbus is not given up as long as this signal is asserted in order that semaphores between multiple processors may be implemented in VMEbus memory.
- S4SEL: State 4 SELect. This signal is asserted approximately at state 4 during a CPU cycle that accesses the VMEbus. This signal is required to make sure that the Arbiter/Requester keeps the VMEbus until the upcoming state 5, because a B_SSEL is about to be asserted. Otherwise it would be possible for Arbiter/Requester to give up the bus just as B_SSEL became asserted, which would mean the VME Master Controller PAL could start its cycle after the VMEbus had already been given up.

28.2. U2704 VME Arbiter and Requester

Pinout of the U2704 PAL is:

Figure 28-1 *U2704 Pinout*



ıte Machine as Arbiter and -quester

This discussion refers to the state diagram labelled "VME Arbiter & Requestor, Arbiter Mode," which is in Appendix A.

State equations are given below; an explanation of the different states is given further on.

```
idle       =  /b_aen*/b_bro*/b_bbo*/b_bgo*/p1_sysr

master     =   b_aen*/b_bro*b_bbo*/b_bgo*/p1_sysr

master_grt =   b_aen*/b_bro*/b_bbo*b_bgo*/p1_sysr

busgrant   =  /b_aen*/b_bro*/b_bbo*b_bgo*/p1_sysr

busreq     =  /b_aen*b_bro*/b_bbo*/b_bgo*/p1_sysr

master_req =   b_aen*b_bro*b_bbo*/b_bgo*/p1_sysr

waitreq    =  /b_aen*b_bro*b_bbo*/b_bgo*/p1_sysr

wait       =  /b_aen*/b_bro*b_bbo*/b_bgo*/p1_sysr

master_ng  =   b_aen*/b_bro*/b_bbo*/b_bgo*/p1_sysr
```

The state machine starts in the IDLE state waiting for B_SSEL or B_SBR.

- B_SSEL indicates that the CPU wants to access the VMEbus, and is asserted from U2701 when signal MMU_VME or B_INTA is asserted, which happens during a read or write to the VMEbus or an interrupt vector acquisition cycle from the VMEbus, respectively.
- B_SBR is asserted when P1_BR3 comes in from an external VME device (through U2401 NAND gate) indicating that it wants the bus. P1_BR2, P1_BR1, and P1_BR0 are monitored for the case where the 2060 board is a requester only.

When B_SSEL is asserted there are three possible state transitions, depending on the state of B_SBBIN (synchronized bus busy in, indicating that an external device is asserting P1_BBSY on the VMEbus) and B_SAS (a synchronized version of VMEbus address strobe).

1. If neither B_SBBIN or B_SAS are asserted, the VMEbus is idle and the next state is MASTER, in which we assert B_BBOUT to lock the VMEbus and B_AEN to signal to the VME master interface that the CPU has been granted the bus. B_AEN is the signal referred to in the VMEbus Manual as Master-Granted-Bus.

The equation MASTER state is:

```
In state IDLE;

if B_SSEL * !B_SBBIN * !B_SAS then state = MASTER
```

2. The second case is when B_SBBIN is not asserted but B_SAS is, which means another device is just about to finish using the VMEbus and has released P1_BBSY so that the arbitration process can go on in parallel with that device's last cycle. In this case we go to the WAIT state, where we assert B_BBOUT to lock the VMEbus, and wait for B_SAS to be negated before jumping to the MASTER state where control is granted to the VME master interface.

The equation for the WAIT state is

```
In state IDLE;

if B_SSEL * !B_SBBIN * B_SAS then state = WAIT
```

3. The third case is when B_SBBIN (synchronized bus busy) is asserted, causing us to jump to the BUSREQ state, where we assert B_BROUT. This is necessary in case there is another Release On Request VMEbus requester out there, because such a device will not release P1_BBSY until it sees a request from another device.

The equation for the BUSREQ state is

```
In state IDLE;

if B_SSEL * B_SBBIN then state = BUSREQ
```

Transitions from BUSREQ State

There are two possible transitions from the BUSREQ state once B_SBBIN has been negated, depending on the state of B_SAS at that point.

1. If B_SAS is negated also, then we jump to the MASTER-REQ state, where we assert B_AEN, B_BBOUT, and continue asserting B_BROUT to guarantee overlap between our assertion of P1_BBSY and our negation of P1_BR3. On the next clock we make the transition to the MASTER state.

The equation for the MASTER-REQ state is

```
In state BUSREQ;

if !B_SAS * !B_SBBIN then state = MASTER-REQ
```

2. The second way of leaving the BUSREQ state is to jump to the WAITREQ state, which occurs if B_SAS is still asserted when B_SBBIN becomes negated. This means that the external device that had control of the

VMEbus is performing its last transfer and has released P1_BBSY, so that arbitration can take place.

The equation for the WAITREQ state is

```
In state BUSREQ;

if B_SAS * !B_SBBIN then state = WAITREQ
```

In the WAITREQ state, we once again assert B_BBOUT and continue asserting B_BROUT to guarantee overlap between the two signals.

At the next clock we

1. jump to the WAIT state if B_SAS is still asserted, or
2. jump to the MASTER state if B_SAS is now negated.

From the WAIT state we wait until B_SAS is negated before moving to the MASTER state. This wait is required by the VMEbus, which dictates that the new VMEbus master mustn't enable its drivers onto the bus until the old master has negated the VME address strobe.

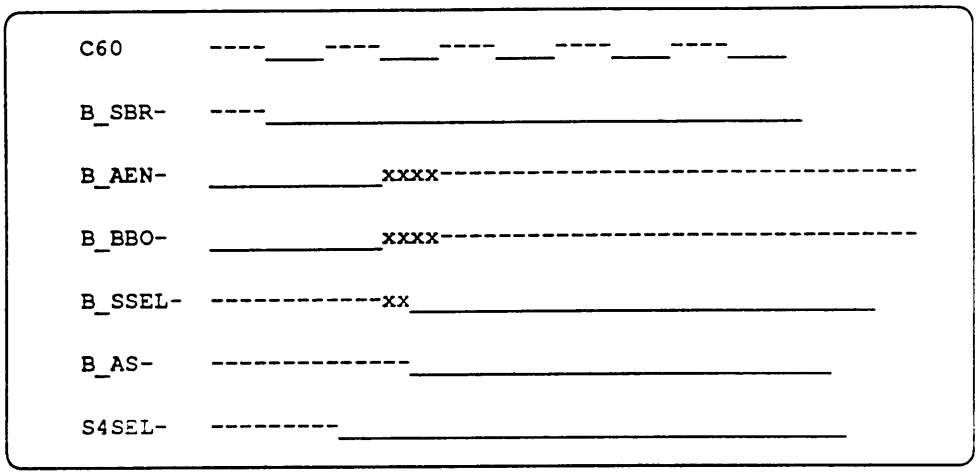
Transitions from MASTER State

The MASTER state is the normal condition of a release-on-request requester, and the only time we leave this state is when an external device has requested control of the VMEbus.

When an external bus request is received (P1_BR3 asserted), we need to look at several conditions.

1. We will stay in MASTER state if P_RMC is asserted, which allows implementation of interprocessor semaphores in VMEbus memory.
2. We will also stay in MASTER state if S4SEL is asserted, which means that B_SSEL is going to be asserted at the next falling edge of the system clock (see terminology, above.) Without S4SEL here, it is possible for the VME Master Controller to assert VME address strobe even though we have made the decision to give up the VMEbus by leaving MASTER state. This is shown in the following timing diagram, in which B_SSEL occurs one clock after B_SBR is received.

Figure 28-2 *VME MASTER State — Relationship of S4SEL to B_SSEL*



If neither P_RMC nor S4SEL is asserted when an external bus request is received, then we look to see whether or not the CPU is using the VMEbus.

- If the CPU is *not* using the VMEbus, then B_SSEL will not currently be asserted and we will make a transition directly to the MASTER_NG2 state, in which we wait until the P1_BBSY that we have asserted onto the VMEbus is no longer being received as B_SBBIN. This is an indeterminate amount of time because of the low-pass filter on P1_BBSY on each 2060 board, and because we don't know how many 2060 boards might be plugged into the backplane.

The equation for the MASTER_NG2 state is

```
In state MASTER;

if !B_SSEL * B_SBR * !P_RMC * !S4SEL then state = MASTER_NG2
```

When B_SBBIN has gone away, we jump to BUSGRANT state, where B_BGOUT is asserted to indicate that the external device requesting the VMEbus has been granted control.

- In the case where the CPU is using the VMEbus and an external bus request is received, then B_SSEL will currently be asserted and we will wait until the 2060 VME Master Controller has asserted P1_AS (which will be received as B_SAS) before we move to the MASTER_NG state.

The equation for the MASTER_NG state is

```
In state MASTER;

if B_SSEL * B_SBR * B_SAS * !P_RMC then state = MASTER_NG2
```

There is a reason for the two separate states, MASTER_NG and MASTER_NG2. If we made a transition from MASTER to MASTER_NG when B_SSEL wasn't asserted and then B_SSEL *was* asserted, we would get stuck forever in MASTER_GRT state waiting for B_SSEL to be deasserted.

Transitions from MASTER_NG State

Once in the MASTER_NG state, we wait for B_SBBIN to go away just as in the transition from MASTER_NG2 state mentioned above. At the point when B_SBBIN goes away we check to see if B_SSEL is still asserted, which means that the CPU is not yet finished with its cycle. If B_SSEL is still asserted we jump to MASTER_GRT state, where we grant the VMEbus to the external device that is requesting it, then wait for B_SSEL to be negated before jumping to BUSGRANT state.

The equation for the MASTER_GRT state is

```
In state MASTER_NG;

if B_SSEL * !B_SBBIN then state = MASTER_GRT
```

The advantage of issuing B_BGOUT before the CPU has finished with its cycle is that this overlaps the end of the CPU cycle with the time required by the bus grant signal to propagate down the bus grant daisy chain. When several boards are installed this can be a significant amount of time, on the order of 90 ns per board. This overlap is allowed by the VMEbus specification because the next device must wait until P1_AS is negated before taking control of the bus. If B_SSEL is not asserted at the point when B_SBBIN goes away, we jump directly from the MASTER_NG state to the BUSGRANT state.

The equation for the BUSGRANT state is

```
In state MASTER_NG;

if !B_SSEL * !B_SBBIN then state = BUSGRANT
```

Transitions from BUSGRANT State

Once we are in the BUSGRANT state, we wait for the external device that has now been granted the bus to acknowledge by asserting P1_BBSY (which we will receive as B_SBBIN). When we receive this signal we either jump to the IDLE state or the BUSREQ state, depending on the state of B_SSEL at that time. If B_SSEL is not asserted when B_SBBIN becomes asserted, that means the CPU does not wish to use the VMEbus and we jump to the IDLE state to await a request from the CPU or another VME device.

The equation for the IDLE state is

```
In state BUSGRANT;

if    (!B_SSEL * B_SBBIN) +
      (!B_SBBIN * !B_SBR) then state =  IDLE
```

If B_SSEL is asserted when B_SBBIN becomes asserted, that means the CPU wants to use the VMEbus and we jump to the BUSREQ state where we assert B_BROUT.

The equation for the BUSREQ state is

```
In state BUSGRANT;

if  B_SSEL * B_SBBIN then state =  BUSREQ
```

28.3. State Machine as Requester Only

When jumper J2701 is removed and jumper J2700 is inserted, the arbiter/requester state machine functions as a requester only, requiring (or allowing) the arbitration function to be taken over by another VME board. Operation as a requester bears a close resemblance to operation as an arbiter/requester except that now the effects of P1_BG3IN, VMEbus Grant Level Three In, must be taken into account, as that is how the external arbiter grants control to the 2060 board. The following explanation refers to the state diagram titled "VME Arbiter/Requester, Requester-Only Mode," which is in Appendix A.

Transitions from IDLE State

The requester state machine starts off in the IDLE state, waiting for either a B_SSEL signal from the CPU indicating a request for control of the VMEbus, or a P1_BG3IN signal from the arbiter indicating the VMEbus has been granted to someone.

The equation for the IDLE state is

```
In state IDLE;

if  !B_SSEL * !P1_BG3IN then state =  IDLE
```

There are four transitions from the IDLE state, depending on the states of the above two signals plus P1_BBSY and B_SAS. These transitions are enumerated below:

1. If the CPU doesn't want the VMEbus when P1_BG3IN is received, then we pass the bus grant down the bus grant daisy chain by jumping to the BUSGRANT state where we assert B_BGOUT, assuming that a device further down the daisy chain is requesting the bus.

The equation for the BUSGRANT state is

```
In state IDLE;

if  !B_SSEL * B_BG3IN then state =  BUSGRANT
```

2. If the CPU wants the VMEbus but no bus grant (P1_BG3IN) has yet been issued, then we jump to the BUSREQ state where we assert B_BROUT.

The equation for the BUSREQ state is

```
In state IDLE;

if  !B_SSEL * B_BG3IN then state =  BUSGRANT
```

3. The third transition from the IDLE state occurs when the CPU wants to access the VMEbus (B_SSEL asserted), a bus grant has been issued (P1_BG3IN asserted), no other device has acknowledged the bus grant (P1_BBSY negated), but the previous bus master is still active (B_SAS still asserted). In this case, we jump to the WAIT state to wait for the negation of B_SAS before making the transition to the MASTER state.

The equation for the WAIT state is

```
In state IDLE;

if B_SSEL * B_BG3IN * !B_SBBIN * B_SAS then state = WAIT
```

4. The final way of leaving the IDLE state is to jump directly to the MASTER state, which happens when the CPU wants to use the VMEbus (B_SSEL asserted), the VMEbus is idle (P1_AS and P1_BBSY negated), and a bus grant has been issued (P1_BG3IN asserted). This happens when the CPU asserts B_SSEL just as a bus grant is being given to another device, so we in effect intercept the bus grant intended for the other device.

The equation for the MASTER state is

```
In state IDLE;

if B_SSEL * B_BG3IN * !B_SBBIN * !B_SAS then state = MASTER
```

Transitions from BUSREQ State

Two transitions out of the BUSREQ state are possible, both requiring B_SBBIN negated and P1_BG3IN asserted:

1. If B_SAS is not asserted when we recognize that B_SBBIN is negated and P1_BG3IN is asserted, that means we can use the VMEbus immediately so we jump to the MASTER-REQ state. In the MASTER-REQ state we assert B_AEN to grant the bus to the CPU, B_BBOUT to inform the arbiter that we are taking the bus, and we continue asserting B_BROUT to guarantee overlap between the bus request and bus grant signals. On the next clock, we jump to the MASTER state.

The equation for the MASTER-REQ state is

```
In state BUSREQ;

if  B_BG3IN * !B_SBBIN * !B_SAS then state = MASTER-REQ
```

2. If B_SAS is still asserted when we recognize that B_SBBIN is negated and P1_BG3IN is asserted, we must wait until the previous bus master relinquishes the bus by negating P1_AS. We do this by jumping to the WAITREQ state, where we assert B_BBOUT and B_BROUT for the reasons mentioned above, but don't yet issue B_AEN to grant the bus to the CPU.

The equation for the WAITREQ state is

```
In state BUSREQ;

if  B_BG3IN * !B_SBBIN * B_SAS then state = WAITREQ
```

If at the next clock B_SAS has gone inactive, we jump to the MASTER state, otherwise we jump to the WAIT state to wait for the negation of B_SAS before moving to the MASTER state.

Transitions from MASTER State

All of the above paths lead to the MASTER state, where we assert B_AEN to inform the CPU that it has been granted the VMEbus, and we assert B_BBOUT to inform the arbiter that we have taken control. It is in this state that the CPU performs its transfers to and from the VMEbus.

We leave the MASTER state when we see another device requesting the VMEbus via P1_BR0, 1, 2, or 3, the OR combination of which is called B_SBR (U2401), as long as P_RMC or S4SEL are not asserted. Just as in the Arbiter/Requester case above, P_RMC holds the VMEbus throughout a CPU read-modify-write cycle, and S4SEL holds the VMEbus if B_SSEL is just about to be asserted. If the CPU is not using the VMEbus when we receive B_SBR (B_SSEL inactive), and P1_BG3IN has been negated, we jump directly to the IDLE state. P1_BG3IN must be inactive to meet the VME specification requirement that the requester must not release bus busy until bus grant in has been removed.

The equation for the IDLE state (from MASTER) is

```
In state MASTER;

if    !B_SSEL * B_SBR * !B_BG3IN
   * !S4SEL * !P_RMC   then state = IDLE
```

If the CPU is still using the VMEbus when B_SBR is received, we wait until P1_AS has been asserted by the 2060 VME Master Controller PAL (and once again check that P1_BG3IN has been negated), then jump to the MASTER_NG state, where we wait for P1_BG3IN to be asserted again.

The equation for the MASTER_NG state is

```
In state MASTER;

if   B_SSEL * B_SBR * !B_BG3IN
   * B_SAS * !P_RMC then state = MASTER_NG
```

Transitions from MASTER_NG State

The purpose of the MASTER_NG state is to allow arbitration to proceed in parallel by releasing P1_BBSY but holding the bus by asserting P1_AS.

From MASTER_NG state are three possible transitions, all of them eventually leading to the BUSGRANT state.

1. If the CPU is finished using the VMEbus before the arbiter issues the bus grant for the next master (B_SSEL negated before P1_BG3IN asserted), we jump to the IDLE state then to the BUSGRANT state when B_BG3IN is received.

The equation for the IDLE state is

```
In state MASTER_NG;

if !B_SSEL * !B_BG3IN then state = IDLE
```

2. If we receive the bus grant at the same time that the CPU finishes using the VMEbus, we jump directly to the BUSGRANT state.

The equation for the BUSGRANT state is

```
In state MASTER_NG;

if !B_SSEL * B_BG3IN then state = BUSGRANT
```

3. If the bus grant is received before the CPU is finished using the VMEbus, we jump to the MASTER_GRT state, where we assert B_BGOUT and await the negation of B_SSEL before proceeding to the BUSGRANT state.

The equation for the MASTER_GRT state is

```
In state MASTER_NG;

if B_SSEL * B_BG3IN then state = MASTER_GRT
```

Transitions from the BUSGRANT State

In the BUSGRANT state we assert B_BGOUT while awaiting the negation of P1_BG3IN. If at that point B_SSEL is not asserted we jump back to the IDLE state, whereas if it is asserted we jump to the BUSREQ state.

The equation for the IDLE state is

```
In state BUSGRANT;

if !B_SSEL * !B_BG3IN then state = IDLE
```

The equation for the BUSREQ state is

```
In state BUSGRANT;

if B_SSEL * !B_BG3IN then state = BUSREQ
```

38

Memory RAM — *Pages 32 and 33*

Memory RAM — *Pages 32 and 33* **359**

38.1. Memory Read 359
38.2. Memory Write 359
38.3. Processor Data Acquisition 359

38

Memory RAM — *Pages 32 and 33*

Eight banks of eighteen 256K-by-1-bit chips make up the memory array on the 2060 board. This provides 4Mbytes of memory, with one bit of parity per byte. Multiplexing of eighteen address bits from the P2 bus, p2_a(21:02), allow access to a bit within each 256K-by-1-bit chip, or an entire word from one of the eight banks.

Bit data is bidirectionally connected to the 32-bit P2 data bus through the U3200, U3202, U3300, U3302 data transceivers. Direction of the data flow through these four transceivers is controlled by the assertion of the memory read/write (m_rw) signal.

38.1. Memory Read

When m_rw is high — a read cycle — and output is enabled through the assertion of the memory buffer enable signal m_ben-, a bit of data from each memory RAM within the selected bank is coupled to the P2 data bus.

38.2. Memory Write

When m_rw is low — a write cycle — and output is enabled through the assertion of the memory buffer enable signal m_ben-, a bit of data is written into each memory RAM (within the selected bank) from the P2 data bus.

Thus the truth table for the memory data buffers is:

Table 38-1 *Memory Data Buffers — Data Flow*

Gate *m_ben-*	Direction *m_rw*	Which way the data will flow
0	1	memory to P2 data bus (A -> B)
0	0	P2 data bus to memory (B -> A)
1	X	*outputs are tri-state*

38.3. Processor Data Acquisition

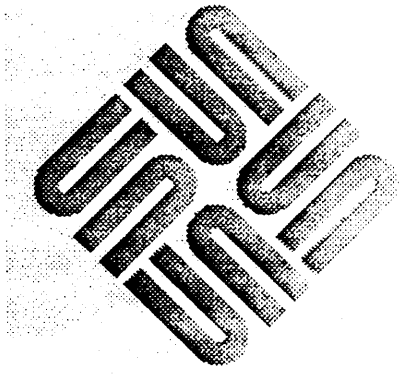
Note that assertion of m_rw and m_ben- are valid *for all four data transceivers simultaneously* and thus memory data is driven onto all 32 bits of the data bus; this leaves it up to the processor (through bus sizing and byte offset capabilities) to access the byte(s) it wants within this 32-bit data space. Manipulation of the size and offset bits by the processor also determines which of the RAS signals will be valid to each bank of memory (see the section on U3100 and U3102 for further explanation).

29

VME Master Interface

VME Master Interface **285**

29.1. VME Select and Freeze PAL, U2701 285
Pinout for the U2701 PAL 286
Terminology for the VME Select and Freeze PAL 287
CPU Reruns on VME "Short Timeouts" 288
CPU Reruns on Deadlocks 288
29.2. VME Select and Freeze State Diagram 288
Normal Operation 288
Deadlock Resolution 289
VME Short Timeouts 290
VME Long Timeouts 290
29.3. VME Master Controller PAL U2806 291
Terminology for VME Master Controller 293
VME Master Controller State Machine 293
CPU Retains Control of VMEbus at End of Cycle 294
CPU Relinquishes Control of VMEbus at End of Cycle 295
CPU Freeze Cycles 295
Address Modifiers and P_BLWORD 295
Non-Aligned Master Cycles 296



29

VME Master Interface

29.1. VME Select and Freeze PAL, U2701

The VME Select and Freeze PAL, U2701, controls the VMEbus select signals B_SSEL and S4SEL, and controls the operation of the VMEbus Master Interface during VME rerun cycles.

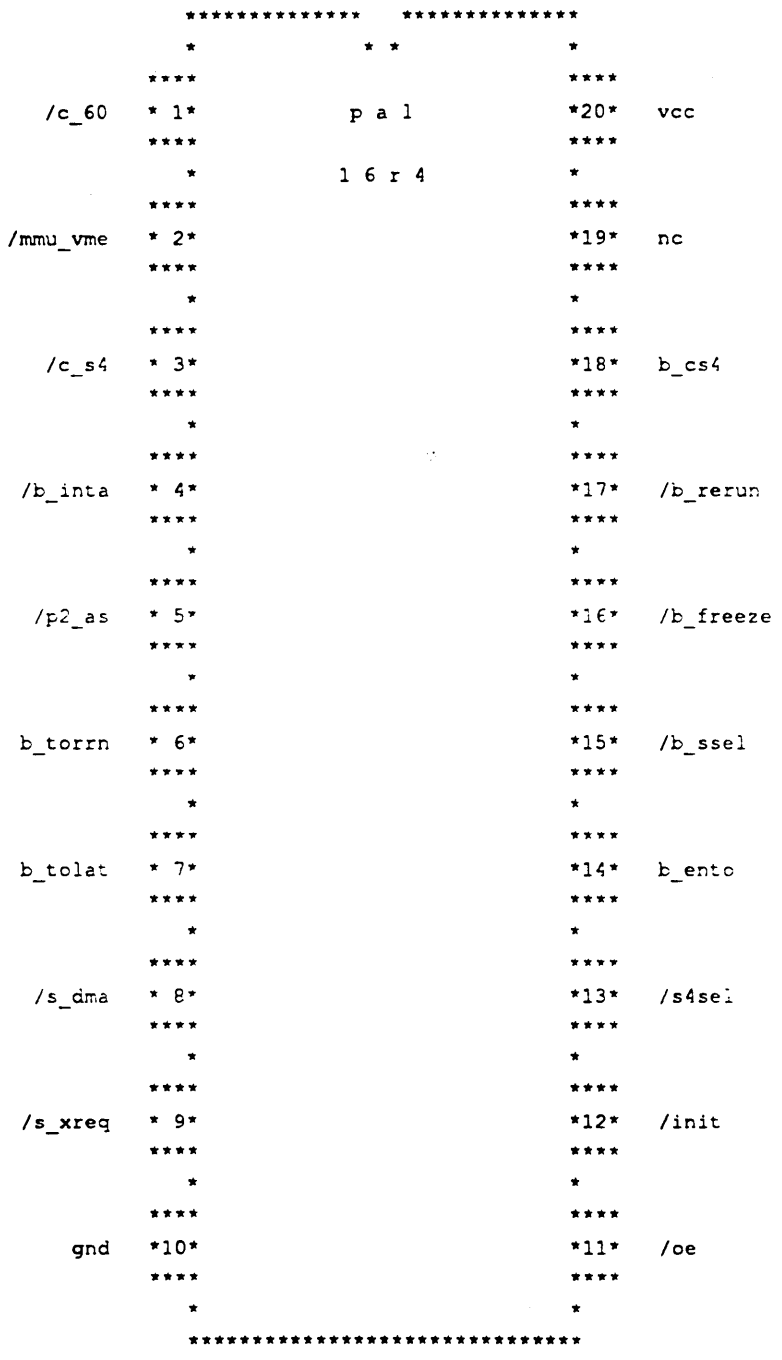
A CPU rerun occurs under two sets of circumstances:

1. when a VME device fails to respond within 2.88 microseconds, or
2. when the CPU attempts to access the VMEbus at the same time as an external VME master accesses the 2060.

.nout for the U2701 PAL

Pinout of the U2701 PAL is:

Figure 29-1 *U2701 Pinout*



Terminology for the VME Select and Freeze PAL

- B_CS4: VME Clock State 4. Latches CPU addresses into the VME master address latches at state 4 on every P2 Bus cycle except when the VME interface is frozen.
- B_ENTO: VME ENable Time Out. Tells the VME Rerun Timer to start counting the time from the start of a VME Master cycle. Resets the counter once a rerun has been requested.
- B_FREEZE: Tells the VME Master Controller PAL and the VME Data Buffer Control PAL to freeze their current state so that the CPU doesn't have to maintain all of its signals while it reruns a cycle.
- B_INTA: Signal from the Interrupt Acknowledge Generator PAL (U304) that indicates the CPU is trying to fetch an interrupt vector from the VMEbus.
- B_RERUN: Tells the Bus Error PAL (U202) to assert P_HALT and P_BERR simultaneously, causing the CPU to rerun its current cycle. This signal is combined in PAL U107 with the rerun request from the P2 Bus, SP2_RERUN, before going to the Bus Error PAL.
- B_SSEL: the key signal in all VME master cycles, indicating that the CPU is accessing the VMEbus either to write or read data or fetch an interrupt vector. Goes active at state 5.
- B_TOLAT: VME TimeOut LATch. Comes from the VME Rerun Timer (U2700) and signifies that it's time to close the latches that allow VME DTACKs onto the 2060 board.
- B_TORRN: VME TimeOut ReRuN. Comes from the VME Rerun Timer (U2700) and signifies that it's time to rerun the CPU because a VME Slave has taken too long to respond. This signal has meaning only when asserted in conjunction with B_TOLAT.
- LONG TIMEOUT: A period equal to 256 Short Timeouts, at which point we decide that the addressed VME device is never going to respond, and we abort the cycle.
- MMU_VME: Signal from the MMU Validation/Decode PAL (U612) that goes active when MMU_TYP[1] is active, indicating that the CPU is either reading or writing data to the VMEbus. Goes active at state 4.
- S4SEL: combinatorial signal that warns the VME arbiter that B_SSEL will be asserted on the next falling clock edge. This is required so that the arbiter won't decide to give up the bus just as the CPU starts to use it, which would create a condition where the VME Master Controller PAL could assert address and data strobes onto the VMEbus even though we had just granted control of the bus to another board. Goes active at state 4.
- SHORT TIMEOUT: A 2.88 microsecond period after which the CPU reruns a VME access if no response is received from the addressed VME device.

CPU Reruns on VME "Short Timeouts"

When a VME device being accessed by the 2060 has a response time of greater than 2.88 microseconds, the access will be broken up into one or more shorter cycles by VME "short timeouts." This is required because the VMEbus has no specified maximum response time, and the local bus must not be kept tied up for long periods of time or the 2060 will start missing Ethernet packets and dynamic RAM refresh cycles. The Xylogics disk controller board, for example, has response times as long as 80 microseconds because reads from certain registers are actually requests for the Xylogics microcontroller to perform a short program and report back the results.

In this case what happens is that the state of the VME Master Interface is frozen, incoming DTACKs from the VMEbus are disregarded, and a rerun of the current cycle is requested of the CPU via simultaneous assertion of P_BERR and P_HALT. The CPU indicates its acceptance of the rerun by negating address strobe (P_AS) and re-arbitrating for the local bus. If at that point a request for the local bus is pending (signified by P_BR being active), the CPU will grant local bus access to a DVMA device and a DVMA cycle will be performed before the CPU begins the instruction again. This would be the case if an Ethernet request (E_DMAREQ and S_EHOLD) or a refresh request (R_DMAREQ) were pending.

CPU Reruns on Deadlocks

The second type of CPU rerun is called a deadlock. Arbitration of deadlocks is required because the VMEbus makes no provision for backing off or rerunning a cycle. During deadlocks the B_RERUN signal is asserted but the B_FREEZE signal is not; since the CPU does not yet have control of the VMEbus there is no "VMEbus state" to preserve. The CPU will accept the rerun command, grant the local bus to the DVMA controller, the VME Slave cycle will complete, and then the CPU will perform its VME Master cycle.

29.2. VME Select and Freeze State Diagram

The state diagram covers four possible contingencies:

- normal operation
- resolution of deadlocks
- short timeouts, and
- long timeouts.

Each of these are explained below. Refer to the state diagram labelled "VME Select & Freeze State Machine" (in Appendix A) for the following explanations.

Normal Operation

Normal operation of the VME Select and Freeze PAL can be defined as VME accesses that don't involve reruns or deadlock conditions. Under normal operation, U2701 waits in the IDLE state, or state A of the figure, until receiving either the signal MMU_VME or B_INTA indicating a VME data access or interrupt vector fetch, respectively.

The equation for the IDLE state is

```
if !MMU_VME * !B_INTA then state = IDLE
```

When one of these signals becomes asserted, U2701 waits until CS4 is also asserted by the central timing generator, then asserts B_SSEL. Waiting for CS4 assures that B_SSEL won't go active until after state 5 because U2701 is clocked by /c60 (inverted 60 nsec clock). P2_AS is included in the equation with MMU_VME so that B_SSEL is negated properly at the end of the cycle. This takes us to state B, in which we stay until the next clock.

The equation for the state B is

```
In state IDLE;

if MMU_VME * P2_AS * CS4
   + B_INTA * CS4   then state = state B
```

In state B you can go either to state C (if S_XREQ is not asserted) or state E (if S_XREQ is asserted).

With the assertion of B_ENTO the VME short timeout counter starts counting to signal a rerun. If the VME device being accessed responds within 2.88 microseconds, the response (P1_DTACK) will get transmitted to the CPU, which will respond by negating P2_AS. This will cause U2701 to jump to state D by negating B_SSEL, then on the following clock it will return back to the IDLE state.

Deadlock Resolution

A deadlock occurs when the CPU accesses the VMEbus just as an external VME device (that already has control of the VMEbus) accesses the P2 bus. As in normal operation above, U2701 will proceed from state A to state B. Then S_XREQ will be received either before or after the transition to state C.

1. If S_XREQ is received during state B, the state machine goes directly to state E.
2. If S_XREQ is not received in state B, the state machine goes on to state C, whereupon assertion of S_XREQ moves the state machine to state E.

Once in state E, B_RERUN is sent to the Bus Error PAL, U202. On the next clock B_ENTO will go away (since we are already rerunning the CPU and don't want to be confused by short timeout signals). This will put us in state F where we will wait for the rerun to be accepted by the CPU, indicated by P2_AS going away. When P2_AS goes away we will jump to state G and negate B_SSEL, followed by a transition back to IDLE on the next clock.

At this point the CPU grants control of the P2 bus to the DVMA controller, which will perform a VME Slave cycle before returning control of the P2 bus back to the CPU. When the CPU regains control of the P2 bus, it will again attempt to perform the same cycle it attempted earlier.

VME Short Timeouts

When the CPU requests access of the VMEbus there may be some time wasted in acquiring control of the bus, which is followed by the selected device taking a further amount of time to respond. If the sum of these two time periods exceeds 2.88 microseconds, a VME Short Timeout is asserted. This results in the CPU rerunning the cycle (after the DVMA controller first checks to see if there are any pending requests from the refresh or Ethernet subsystems). In this case U2701 will proceed normally from state A to state B to state C, as above, and 2.88 microseconds later will receive the combination of signals that causes a Freeze/Rerun. That combination is the simultaneous assertion of B_TOLAT and B_RERUN.

The start of a Freeze/Rerun cycle will be signalled by U2701 asserting B_FREEZE and B_RERUN. B_FREEZE goes to the VME Master Interface and the VME Data Buffer Controller, causing them to freeze the state of the VMEbus interface until the CPU restarts the current access. B_RERUN goes to PAL U107 where it is combined with the synchronized P2 rerun signal, SP2_RERUN.†

The combined signal, RERUN-, then goes to the Bus Error PAL U202 where it causes P_BERR and P_HALT to be asserted.

On the next clock after B_FREEZE and B_RERUN are asserted, B_ENTO will be negated, causing the VME short timeout counter (U2700) to be reset. The state machine will then remain in state J until the CPU responds to the rerun request by negating P2_AS, at which point it will jump to state K by negating B_RERUN. On the next clock the state machine will proceed to state L with the assertion of B_ENTO to start the VME short timeout counter counting the period until the next rerun cycle. We will remain in state L until the state 4 of the next non-DMA cycle, indicated by S_DMA being negated and CS4 being asserted. This is guaranteed to be the CPU re-attempting its original transfer. When this occurs, we will jump back to state C by unfreezing the state of the VME interface (negating B_FREEZE).

VME Long Timeouts

The VME Rerun Timeout counter is an LS590 (U2805) that counts 256 Freeze cycles on a single CPU access before asserting B_TOUT. It starts counting when B_SSEL is asserted, which is accomplished by inverting the B_SSEL signal in inverter U2803 and running that into the clear input. When B_SSEL is asserted the clear condition is removed. If B_SSEL goes away before the counter reaches 256, the counter is cleared by B_SSEL.

The counter is clocked by the trailing edge of B_FREEZE, so we will be in state C when B_TOUT is asserted. This will cause TOUT to be asserted externally to the VME Select and Freeze State Machine. This will assert S_TOUT, which will cause the Bus Error PAL U107 to assert P_BERR, aborting the cycle. The VME Select and Freeze State Machine will see only the CPU's response to this, which will be the negation of P2_AS. This will cause it to jump to state D, followed by state A, the IDLE state.

†This PAL is used only because it had the required number of inputs and outputs to combine these signals, so can be thought of as an external AND gate (logical NOR).

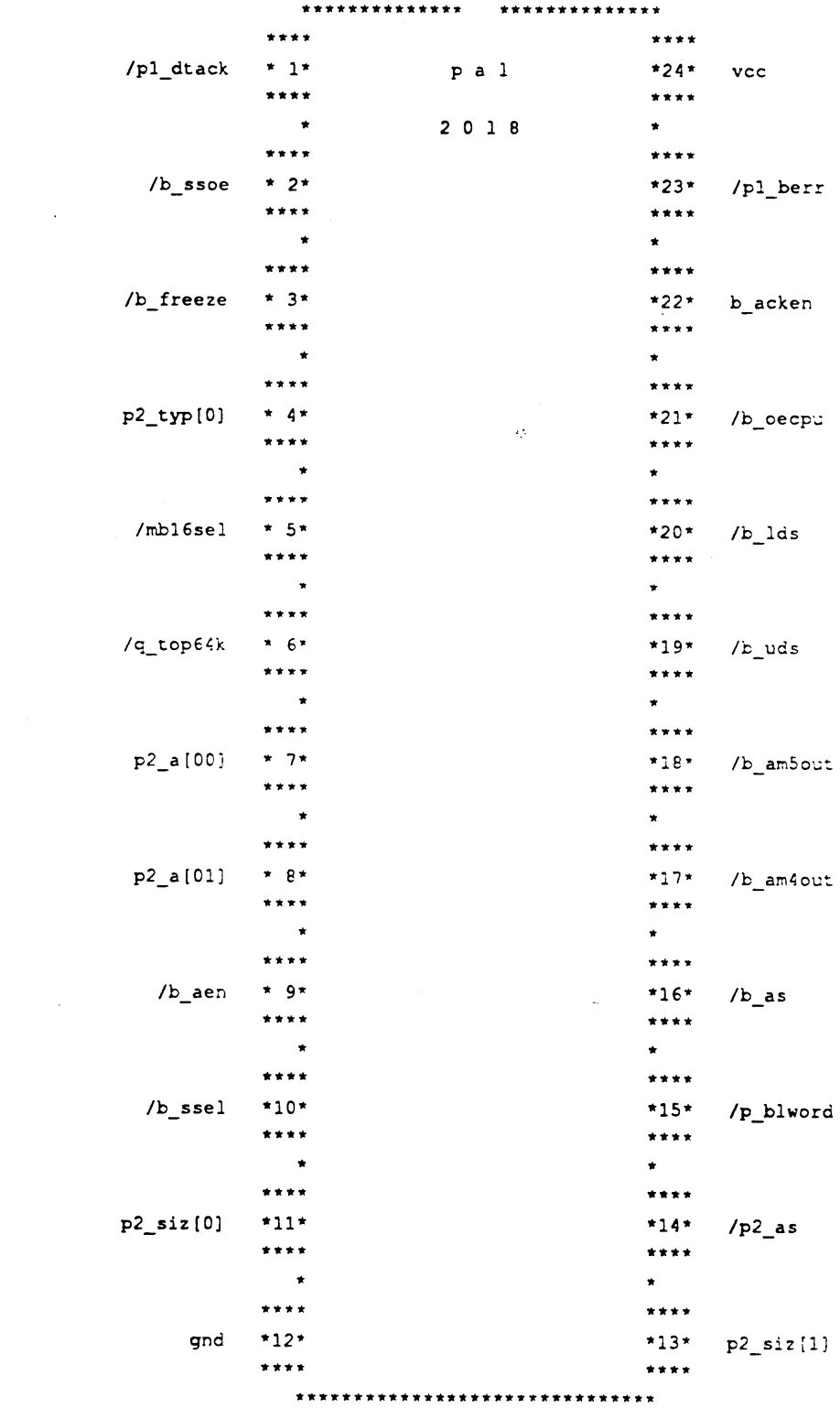
VME Long Timeouts are inhibited during the 200 millisecond VME reset period so that if the CPU attempts to access the VMEbus during this period it will rerun the cycle continuously until the reset period is over. This is done by inverting the B_RSTOUT signal in inverter U2803, and feeding that into the count enable input of the VME Rerun Timeout Counter.

29.3. VME Master Controller PAL U2806

The VME Master Controller PAL controls VME address strobe, data strobe, transfer size, output enable, acknowledge enable, and some address modifiers during VME Master cycles. It is an asynchronous state machine that will run as fast as the PAL is able. The PAL has one fairly complex state machine to handle output enable, address strobe, data strobes, and acknowledge enable; one fairly simple state machine to handle address modifiers; and one combinatorial output.

Pinout of the U2806 PAL is:

Figure 29-2 *U2806 Pinout*



Terminology for VME Master Controller

- B_ACKEN: VME ACKnowledge ENable. Allows acknowledges, P1_DTACK and P1_BERR, to flow to the CPU on VME cycles. Also clear them at the end of each cycle and holds them off during rerun cycles.
- B_AEN: Signal from the VME Arbiter/Requester PAL (U2704) indicating that the CPU has won control of the VMEbus.
- B_FREEZE: Signal from the VME Select and Freeze PAL (U2701) that causes the current state of the VME Master Interface to be frozen during CPU rerun cycles.
- B_OECPU: VME Output Enable CPU. Enables the Master Address Buffers onto the VMEbus and goes to the VME Data Buffer Control PAL (U3000) where it is combined with several other signals to enable the P2 Data bus onto the VMEbus under certain conditions.
- B_SSEL: Signal from the VME Select and Freeze PAL (U2701) indicating that the CPU is accessing the VMEbus.
- B_SSOE: VME Synchronized Synchronized Output Enable. B_OECPU from the VME Master Controller PAL (U2806) is run through two flip-flops to form B_SSOE, which indicates that addresses have been enabled onto the VMEbus long enough to satisfy the VME address setup requirement.
- MB16SEL: MegaByte 16 SELect; generated by address decoding logic in the Video section, this signal is active when P2_A[31:24] are all high. This function is shared with the Video section to save an IC, and chooses between 32- and 24-bit VME addressing modes on Master cycles.
- MMU_TYP[0]: The low-order type bit generated by the MMU. It is used here to distinguish between 16- and 32-bit VME accesses. When MMU_TYP[1] is high, MMU_TYP[0] low indicates 16-bit VME data space, and high indicates 32-bit VME data space.
- P_BLWORD: Processor to VME LongWORD. Indicates that the CPU is performing a legal VME 32-bit data transfer. Active only when performing a longword-aligned longword access to type 3 space. Latched externally in U2808.
- Q_TOP64K: Active when P2_A[23:16] are all high. It is used here to choose between 24- and 16-bit VME addressing modes on Master cycles when MB16SEL is active. The "Q" prefix has historical roots.

VME Master Controller State Machine

The following discussion refers to the state diagram titled "VME Master Controller State Machine," which is in Appendix A.

U2806 waits in the IDLE state (A) until the VME Arbiter signals that the CPU has been granted control of the VMEbus by asserting B_AEN. It then also checks to see that neither P1_DTACK nor P1_BERR is active before starting a cycle in order to satisfy the VME requirement that the acknowledges must be inactive before data strobes are asserted. If this is the case we jump to state B by asserting B_OECPU, which goes to the VME Master Address Buffers and enables the P2 addresses onto the VMEbus.

The equation for the IDLE state is

```
if B_AEN * !P1_DTACK * !P1_BERR then state = IDLE
```

The VMEbus specification requires addresses to be enabled onto the bus for a minimum of 35 nanoseconds before address strobe is asserted. In order to satisfy this requirement worst case, we run B_OECPU through two clocked flip- flops (routed twice through U2703) and wait for the result, B_SSOE, before asserting address strobe. This is shown on the state diagram as states C and D.

Once address strobe is asserted, we check for the size and alignment of the transfer and assert data strobes accordingly. If the transfer is word-aligned and its length is word, 3-byte, or longword, both data strobes are asserted and we jump to state F. If the transfer is word-aligned and byte size, upper data strobe is asserted and we find ourselves in state M. If the transfer is not word-aligned, only lower data strobe is asserted and we end up in state P.

Table 29-1 *U2806 Transfer Logic*

Transfer	Length	State	Equation
word aligned	word 3-byte longword	F	!P2_A00 * (!P2_SIZ[0] + P2_SIZ[1])
word-aligned	byte-size	M	!P2_A00 * (P2_SIZ[0] + !P2_SIZ[1])
not word-aligned		P	P2_A00

CPU Retains Control of VMEbus at End of Cycle

When the VME Slave currently being addressed responds with a P1_DTACK or P1_BERR, this will be sent to the CPU, which will negate P2_AS. If no other VME device has requested control of the VMEbus, the 2060 board will retain control by continuing to assert P1_BBSY and will indicate this to the VME Master Controller PAL by keeping B_AEN asserted.

If at the point when P2_AS becomes negated B_AEN is still asserted, we will leave state F, M, or P and move to state D to await the next Master cycle. The start of the next Master cycle will be indicated by the assertion of B_SSEL. The design could also have been implemented such that at the end of every Master cycle we return to IDLE state, but this would require us to waste the VME address setup time on every cycle. This implementation, on the other hand, takes advantage of the fact that the P2 addresses are valid at state 4 even though we don't know whether or not the current cycle involves the VMEbus until state 5. If it does, the VME address strobe can be asserted immediately.

If, instead of receiving another B_SSEL, the VME Arbiter/Requester PAL gives up the VMEbus and indicates this to the VME Master Controller by negating B_AEN while we're in state D, we simply negate B_OECPU by jumping to state

K.

CPU Relinquishes Control of VMEbus at End of Cycle

If B_AEN is not asserted at the point that the CPU negates P2_AS, this indicates that an external VME device has requested control of the VMEbus and a completely different set of actions occurs. When transferring the VMEbus from one master to another, a significant amount of time can be required to pass the bus grant down the bus grant daisy chain. For example, the 2060 on the average takes more than 90 nanoseconds to pass bus grant in to bus grant out when configured as a VMEbus requester only and not requesting the bus. Ten such boards would require over 900 nanoseconds to pass the bus grant to the lowest priority board on the VMEbus.

The VMEbus specification allows pipelining of this time with the time taken to perform the last transfer by the master that is presently relinquishing the bus. The 2060 board takes advantage of this by negating P1_BBSY and asserting P1_BG3OUT as soon as P1_AS has been asserted on the last transfer. This means that the only way we have of indicating that we still have control of the VMEbus is by continuing to assert P1_AS; therefore the VMEbus specification has a requirement that if this pipelining is implemented the address and data drivers must be tri-stated before P1_AS is negated at the end of the cycle.

States H and J in the VME Master Controller State Diagram allow this pipelining. The Controller negates VME data strobes and clears out the acknowledges as it moves from states F, M, or P to state H, then disables the drivers by negating B_OECPU as it moves from state H to state J. Now that the drivers are disabled we can remove the VME address strobe, which takes us to state K. States K and L don't perform any tasks.

CPU Freeze Cycles

When the VME Master Controller is waiting in states F, M, or P and B_FREEZE is asserted, we move to the corresponding state G, N, or Q by negating B_ACKEN. This occurs when 2.88 microseconds have passed since the CPU requested a VME cycle and no response has been received yet (see the discussion above in the section on the VME Select and Freeze PAL). It is necessary to block the acknowledges so that any DMA cycles that might occur before the CPU cycle actually reruns don't inadvertently get terminated by P1_DTACK from the VMEbus. We will remain in state G, N, or Q until B_FREEZE is negated, at which point we'll return to the corresponding state F, M, or P.

Address Modifiers and P_BLWORD

The VMEbus uses six signals in addition to the addresses to give more information about the type of cycle being performed. These are called the Address Modifiers. P1_AM2:0 correspond to function codes P_FC2:0 in the 68000 family in the sense that they select between Supervisor and User modes, and between Program and Data spaces.

- Address Modifier 3 is always high in in our implementation and all modes with it low are undefined.
- Address Modifiers 5 and 4 are generated in PAL U2806 based upon decodes of the top 16 P2 address bits. They select between 16-, 24-, and 32-bit addressing modes. The complete encoding of the address modifier bits as used by the 2060 board is shown in the following table:

Table 29-2 *Address Modifier Bits on the 2060 Board*

Address Modifier Codes		*Hex Code*
Address Modifier 5 4 3 2 1 0	*Function*	
L L H L L H	32 bit addressing, User Data Space	09
L L H L H L	32 bit addressing, User Program Space	0A
L L H H L H	32 bit addressing, Supervisor Data Space	0D
L L H H H L	32 bit addressing, Supervisor Program Space	0E
H L H L L H	16 bit addressing, User Data Space	29
H L H L H L	16 bit addressing, User Program Space	2A
H L H H L H	16 bit addressing, Supervisor Data Space	2D
H L H H H L	16 bit addressing, Supervisor Program Space	2E
H H H L L H	24 bit addressing, User Data Space	39
H H H L H L	24 bit addressing, User Program Space	3A
H H H H L H	24 bit addressing, Supervisor Data Space	3D
H H H H H L	24 bit addressing, Supervisor Program Space	3E

P_BLWORD is also generated in U2806 in accordance with the VMEbus Specification Revision B, which says that P1_LWORD can only be asserted on longword accesses that are longword aligned. We also check that MMU_TYP[0] is high, indicating an access to TYPE3 Space, which is where devices having 32-bit data buses are accessed. The address modifiers and type bits combine to generate the address map shown in the figure titled "Sun-3 Physical Address Mapping," which is in Appendix A.

Non-Aligned Master Cycles

The 68020 allows several types of transfers that are not allowed by the VMEbus Specification, including non-word-aligned word accesses, non-longword-aligned longword accesses, and three-byte transfers.

- A non-word-aligned word transfer will result in a byte being read or written.
- A longword transfer to an address misaligned by one or three bytes will also result in a byte being read or written.
- A longword transfer to an address misaligned by two bytes will result in a word being read or written.

The amount of data that the CPU assumes has been accepted or provided by the VME Slave being addressed is determined by which of P_DSACK[1:0] signals are asserted. This is controlled in turn by the DSACK PAL (U204), which looks at address bits 0 and 1, size bits 0 and 1, and type bit 0 to decide which DSACKs should be asserted.

NOTE *You may notice that this scheme doesn't quite fit in with the 68020's dynamic bus sizing, in that it is* **theoretically** *possible to transfer more data per transfer than this scheme allows, but the 2060 board was designed to meet the VMEbus Revision B Specification. Revision B requires that word transfers be word aligned and that longword transfers be longword aligned, a requirement that has since been loosened in the Revision C Specification.*

30

VME Slave Interface

VME Slave Interface **299**

30.1. VME Slave Address Latches, U2901-2 and U2911-13 299
30.2. VME Slave Address Decoder U2907 299
Terminology for the VME Slave Address Decoder, U2907 301
30.3. User DVMA Enable (U2905-6) and Context Registers (U509) 302
30.4. VME Slave Address Multiplexers (U2910:09) 302
30.5. VME Slave Request PAL (U2904) 302
Terminology for VME Slave Request PAL 305
VME Slave Request State Machine 306

30

VME Slave Interface

30.1. VME Slave Address Latches, U2901-2 and U2911-13

Five ICs, U2901-2 and U2911-13, make up the VME Slave Address Latches. Their purpose is to latch the addresses from the VMEbus at a time when they are guaranteed to be valid, and hold them until they are no longer needed. This decreases the sensitivity of the 2060 board to noise on the VMEbus, and it takes no extra ICs to implement latches instead of buffers. Address bits 20-31 are latched in transparent latches (F373s U2901 and U2902) on the leading edge of the buffered form of the VME address strobe (P1_AS buffered through U2803 to form B_ASIN-) in order to take advantage of the address-to-address setup time of 10 nanoseconds guaranteed to a VME Slave by the VMEbus specification. Since the flow-through time of an F373 is less than this setup time, as soon as P1_AS goes active we are guaranteed to have valid outputs from the address latches. This allows the use of a 25 nanosecond PAL in U2907 instead of a 15 nanosecond version, as will be explained later.

Address bits P1_A(19:04) are latched in D-type flip-flops (ALS374s U2913:12), also on the leading edge of P1_AS. They are enabled onto the processor address bus when the DVMA Controller grants control of the P2 bus to the VME Slave interface (X_DMAEN- is true). Address bits P1_A(03:01) and P1.WRITE are latched into U2911 on the rising edge of P1_DS.

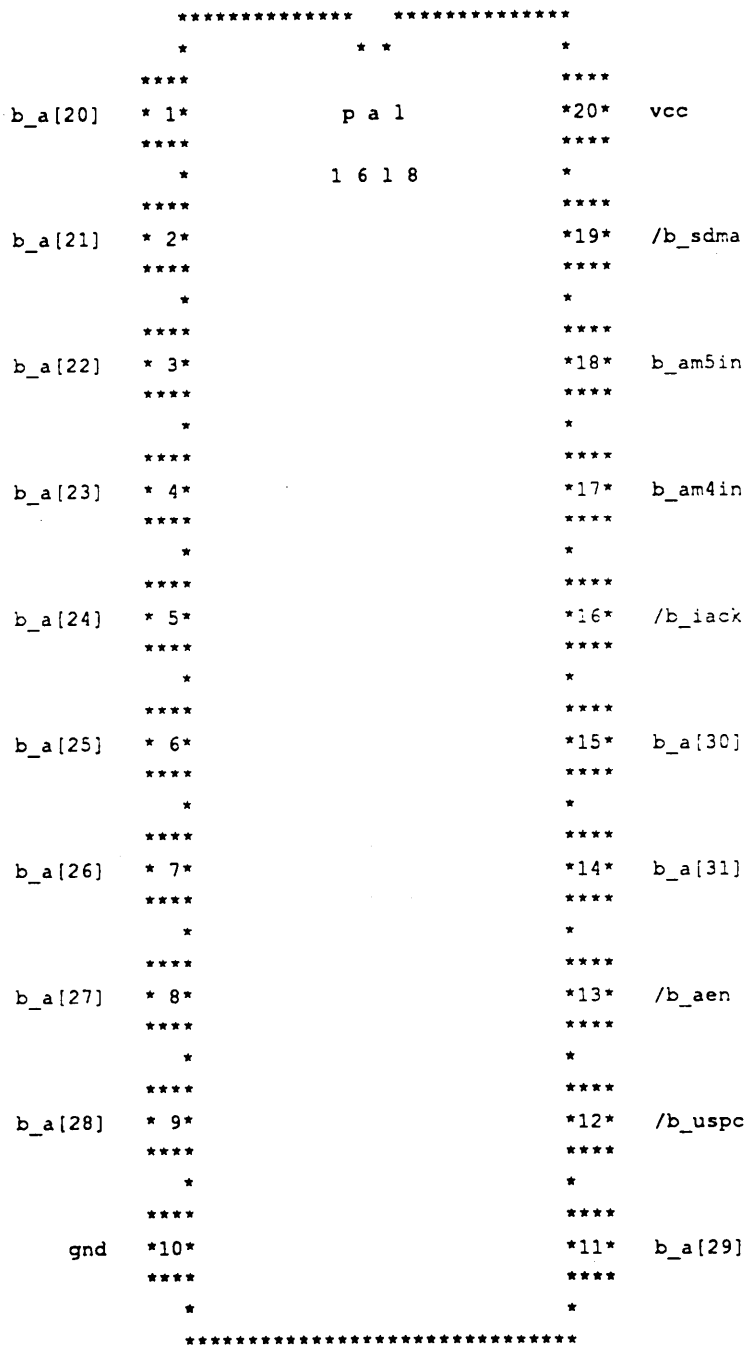
ALS technology was chosen here to minimize undershoot on the processor address bus as several devices sensitive to undershoot are located there, such as the 68020 and MMU RAMs.

30.2. VME Slave Address Decoder U2907

VMEbus addresses are latched on every VMEbus address strobe by the VME Slave Address Latches, and the VME Slave Address Decoder, PAL U2907, examines the contents of these latches continuously to see if the 2060 board is being referenced. It generates one of two signals, B_SDMA or B_USPC, if the address and number of the valid address bits (as indicated by the address modifiers) match any of the locations to which the 2060 board is permanently wired to respond. (See the section above on the VME Master Controller State Machine for a discussion of the VME Address Modifiers.) The figure titled "VMEbus Slave Address Mapping" (in Appendix A) gives a graphic representation of the addresses to which the 2060 board responds.

Pinout of the U2907 PAL is:

Figure 30-1 *U2907 Pinout*



Terminology for the VME Slave Address Decoder, U2907

- B_A[31:20]: these signals are versions of the VME address bus signals P1_A[31:20] latched on the falling edge of /P1_AS.
- B_AM5IN, B_AM4IN: correspond to VMEbus address modifiers P1_AM5 and P1_AM4, which are latched on the falling edge of /P1_AS. When both are high, the latched address has 24 bits valid. When both are low, the latched address has 32 bits valid.†
- B_SDMA stands for VME System DMA and indicates that the bottom megabyte of either the 24-bit or 32-bit VME address space is being accessed.

```
if(vcc) b_sdma = /b_a[31] * /b_a[30] * /b_a[29] *
                 /b_a[28] * /b_a[27] * /b_a[26] *
                 /b_a[25] * /b_a[24] * /b_a[23] *
                 /b_a[22] * /b_a[21] * /b_a[20] *
                 /b_am5in * /b_am4in * /b_iack * /b_ssoe
   lowest megabyte in 32-bit address space => system DVMA

               + /b_a[23] * /b_a[22] * /b_a[21] *
                 /b_a[20] * b_am5in * b_am4in *
                 /b_iack * /b_ssoe
   lowest megabyte in 24-bit address space => system DVMA
```

- B_USPC stands for VME User Space and is roughly analogous to the User form of B_SDMA except that it goes through a further level of qualification before becoming a valid decode, at which point it is called B_UDMA. B_USPC goes active if the address bits indicate an access to the top 2 gigabytes of the 32-bit VME address space.

```
if(vcc) b_uspc = b_a[31] * /b_am5in * /b_am4in *
                 /b_iack * /b_ssoe
   highest 2 gigabytes in 32-bit addr. space => user dvma
```

- B_SSOE is included in the decoding equations of U2907 in order to eliminate self-referential cycles over the VMEbus. That is, when the 2060 board has control of the VMEbus, B_OECPU is asserted to enable the 2060 address drivers onto the VMEbus, and two clocks later this becomes B_SSOE. When B_SSOE is active, all Slave mode decodes are disabled. This allows identical software to be plugged into two 2060 boards in the same backplane, and allows each board to reference the other at the same range of addresses.

†See the VMEbus specification for further details.

- B_IACK: since the 2060 board is not capable of requesting interrupts over the VMEbus, it stands to reason that its Slave interface should not be activated on any interrupt acknowledge cycles. For this reason, whenever B_IACK is active all Slave decodes are disabled.

30.3. User DVMA Enable (U2905-6) and Context Registers (U509)

The 2060 Slave interface divides the top half of the VME 32-bit address space into eight 256-megabyte "contexts" that can be individually enabled or disabled. These contexts correspond to the context bits stored in the Context Register, U509, and are controlled by VME address bits 30:28. On User DVMA cycles, these bits are stored in the Context Register instead of the value that is currently stored there.

The information to which User DVMA contexts are currently enabled is stored in the User DVMA Enable Register, U2905-6. It is an 8-bit read/write register located in Control Space at address 0x50000000. Bit 0 corresponds to context 0, bit 1 corresponds to context 1, and so on. The register is cleared on resets, and a one must be written to enable a context.

These context enable bits are compared in the User DVMA Context Selector, U2908, to the context that the VMEbus is attempting to access. If the latched versions of VME address bits P1_A[30:28] correspond to a context that is enabled, that is, a one is in the proper bit location in the User DVMA Enable Register, and if the VME Slave Address Decoder is asserting B_USPC as explained above, then output B_UDMA will be asserted.

30.4. VME Slave Address Multiplexers (U2910:09)

The VME Slave Address Multiplexers are two 4-bit tri-stable multiplexers that drive processor address lines P_A[27:20] during VME Slave cycles. On User DVMA cycles, they choose the latched VME address lines B_A[27:20]. On System DVMA cycles, they drive P_A[27:20] to all ones, causing the System DVMA to be relocated to a virtual address of 0xFFFX XXXX, where the Xs stand for the address bits actually on the VMEbus. Both 24-bit and 32-bit System DVMA are relocated to this address range, and it is impossible for the 2060 board to distinguish between the two addressing modes.

You will note that the upper four address lines appear to not be driven during VME Slave cycles, but this is done on page 8 by U813, an ALS240 that is enabled on S_DMA which goes active on VME Slave or Ethernet DVMA cycles. In this way we can share the driver between both types of cycles and save half an IC. ALS technology is used for all of the processor address bus drivers to minimize undershoot, as mentioned above in the section on the VME Slave Address Latches.

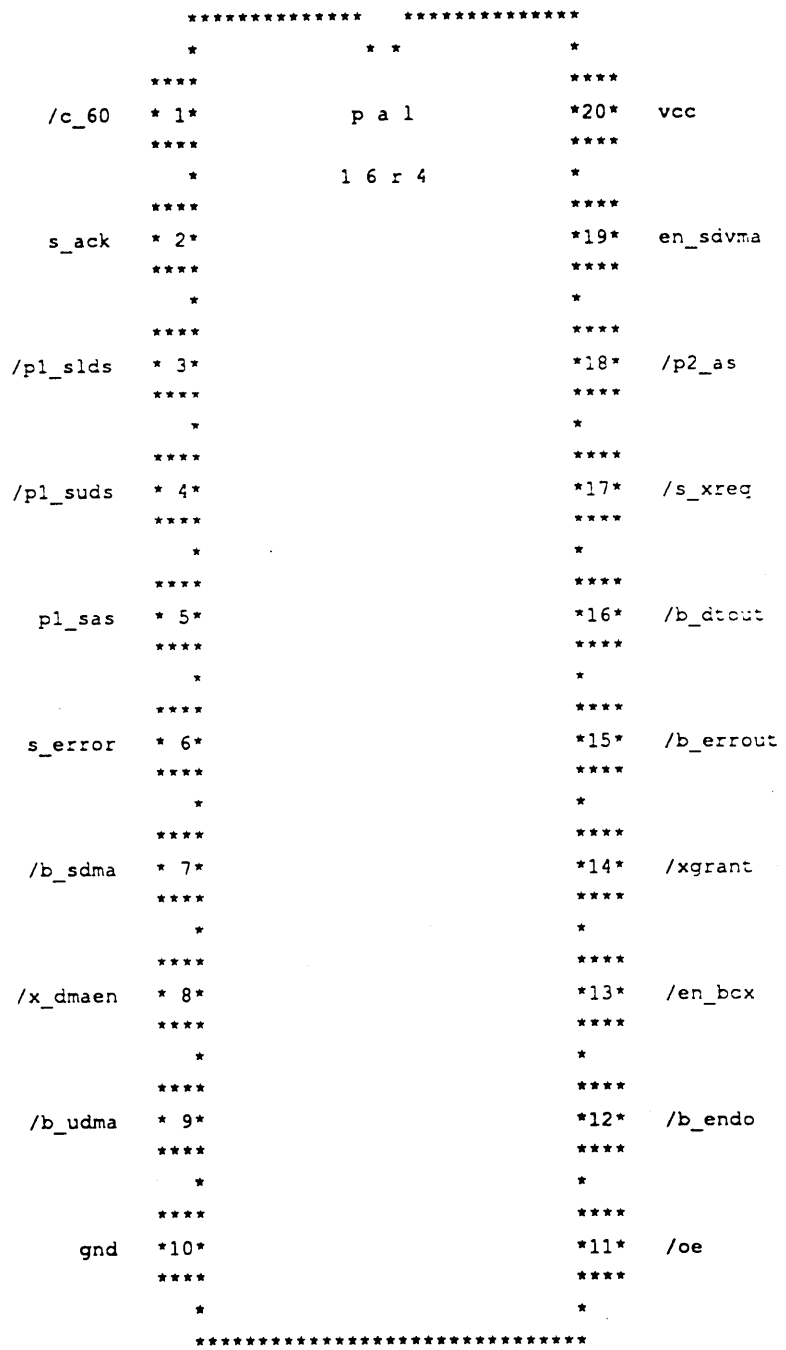
30.5. VME Slave Request PAL (U2904)

The VME Slave Request PAL implements an asynchronous state machine that generates requests for VME Slave cycles to the DVMA Controller, and responds to the completion of a Slave cycle by generating P1_DTACK back to the external master. An earlier implementation of this PAL on the 2060 board used a synchronous state machine, which required the synchronization of the incoming data strobes and increased the turnaround time between cycles to an unacceptably long period. The current implementation asynchronously removes DTACK at the end of the cycle as soon as the external master negates its data strobes. This

is important because the VME spec requires the next cycle not to begin until DTACK from the previous cycle goes away. The synchronous implementation locked out the next cycle for approximately 100 nanoseconds, while the synchronous implementation locks it out for only about 15 nanoseconds.

Pinout of the U2904 PAL is:

Figure 30-2 *U2904 Pinout*



Terminology for VME Slave Request PAL

- B_DTOUT: VME Data Transfer acknowledge OUT. This signal is buffered to form P1_DTACK, and indicates to an external master that the current cycle has been completed.
- B_ERROUT: VME ERRor OUT. This signal is buffered to form P1_BERR, and indicates to an external master that the current cycle has terminated abnormally with some sort of error condition.
- B_SDMA: VME System DMA. Active when the addresses on the VMEbus correspond to the System DVMA address space on the 2060 board. See the section above on the VME Slave Address Decoder.
- B_SXDMA: VME Synchronized eXternal DMA enable. X_DMAEN is fed through a flip-flop clocked on the falling edge of the system clock so that it is delayed one cycle. This allows us to detect the condition at the end of an external DMA cycle when X_DMAEN has just gone away.
- B_UDMA: VME User DMA. Active when the addresses on the VMEbus correspond to the User DVMA address space on the 2060 board. See the section above on the VME Slave Address Decoder.
- EN_BCX: ENable VME ConteXt. Goes to the Context Register, U509, where it selects between the bits currently stored there and the upper address bits coming in from the VMEbus on User DVMA cycles.
- EN_SDVMA: ENable System DVMA. Comes from the System Enable Register, U1406, and indicates that DVMA from the VMEbus has been enabled for System Mode (User Mode is enabled by the User DVMA Enable Register, U2905.)
- P1_DS: VME Data Strobe. Not actually a signal on the VMEbus, this signal is active when either or both P1_DS0 or P1_DS1 are active. It exists to minimize inputs to PALs that require only the presence of any VME Data Strobe (U2904 and U3000).
- P1_SAS: VME Synchronized Address Strobe. Also not actually a VMEbus signal, P1_SAS is formed by running P1_AS- through an inverter (U2803) and then through a synchronizer (U2903). It is used to guarantee enough time for the addresses to ripple through the Slave Decoder section.
- S_ACK: Synchronized ACKnowledge. P_DSACK1 and P_BERR are fed into a NAND gate (U2406) and then into a synchronizer (U2408) to form this signal. It indicates that we are at state 5 of a P2 bus cycle.
- S_ERROR: Synchronous ERROR. Indicates that a protection error or a parity error has been detected on the current cycle. It is synchronous in that errors are reported in the cycle during which they occur, as opposed to one cycle later as happens with parity errors encountered by the CPU.
- S_XREQ: Synchronized eXternal REQuest. XREQ is fed through a synchronizing flip-flop (U2407) before going to the DVMA Controller (U2409) so that we don't confuse the controller, which is a synchronous state machine.

- X_DMAEN: eXternal DMA ENable. Indicates that the CPU is off the local bus and control has been granted to the VME Slave Interface.
- XGRANT: eXternal GRANT. A state variable used only in the VME Slave Request PAL. It goes active when the currently-requested external DVMA cycle has been started on the P2 bus, as indicated by both X_DMAEN and P2_AS being asserted. It stays active until the VMEbus data strobes have been negated by the external master, indicating that it has ended its cycle.
- XREQ: eXternal REQuest. An asynchronous signal that indicates a valid request for use of the P2 bus has been received from an external VMEbus master.

VME Slave Request State Machine

The following discussion refers to the figure titled, "VME Slave Requester State Machine," which is in Appendix A.

The VME Slave Request State Machine (U2904) waits in the IDLE state until it receives a valid address decode in combination with the synchronized VME address strobe (P1_SAS) and at least one VME Data Strobe (P1_DS).

The equation for the IDLE state is

```
In state IDLE;

if !P1_SAS + !P1_DS + B_DTOUT + B_ERROUT then state = IDLE
```

The equation for state A is

```
In state IDLE;

if P1_SAS * P1_DS + * !B_DTOUT *
         ADDRESS_DECODE * !B_ERROUT then state = A
```

This address decode can be either of B_SDMA or B_UDMA, indicating an access to the system address space or user address space, respectively. At that point it checks to see if the previous cycle has terminated, as indicated by both P1_DTACK and P1_BERR being inactive. If this is the case, it asserts XREQ and jumps to state A. XGRANT in that equation (state B) is the end condition, not a start condition, and will be explained later.

The XREQ signal is synchronized externally before it is fed into the DVMA Controller, U2409, where it will initiate the Bus Request/Bus Grant/Bus Grant Acknowledge sequence. When it has gained control of the P2 bus, the DVMA controller will respond with X_DMAEN to enable the VME Slave Addresses onto the P2 bus, then one clock later P2_AS will be asserted to begin the cycle.

The equation for the state B is

```
In state A;

if X_DMAEN + P2_AS then state = B
```

When U2904 sees this it will assert XGRANT, which will negate XREQ, jumping us to state B then C. XGRANT provides a way to remember that we have already requested the current cycle and been granted it, so that we don't assert XREQ for a second time on the same cycle.

We will wait in state C until the end of the cycle, which will be indicated by X_DMAEN going away but the delayed form of it, B_SXDMA, still being present. At this point, if S_ERROR is not asserted, we will jump to state D and send B_DTOUT to the VMEbus (buffered to form P1_DTACK).

The equation for the state D is

```
In state C;

if B_SXDMA * !X_DMAEN * !S_ERROR then state = D
```

If S_ERROR is asserted, we will jump to state E instead, and assert B_ERROUT to signal a termination with error.

The equation for the state E is

```
In state D;

if B_SXDMA * !X_DMAEN * S_ERROR then state = E
```

Either state D or E will signal to the external master to end its cycle, which will cause it to respond by negating the VME data strobes. We will see this as a negation of P1_DS (state D). When this happens, we will jump back to the IDLE state.

The equation for the IDLE state is

```
In state D or E;

if !P1_DS then state = IDLE
```

Note the presence of B_DTOUT in the equation to assert B_ERROUT and vice-versa. This is to ensure that we get one and only one of the two possible responses to end a cycle, no matter what X_DMAEN does. X_DMAEN can do some irregular things when lock mode is engaged and disengaged.

One tricky aspect of this state machine is that we will not jump back to the IDLE state until the entry condition to states D and E has also gone away, that is, until B_SXDMA goes away. This is how we allow for an external master that negate.

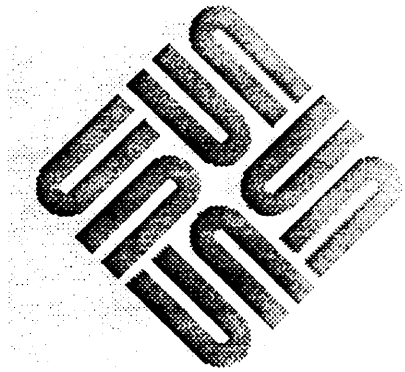
address strobe, then reasserts it before P1_SAS can go high. Since we must also wait for B_SXDMA to go away, we are guaranteed enough time to decode the new address before we reassert XREQ. It is important to remember that, worst case, the clock on the 2060 board is 90 nanoseconds long and so it is conceivable that an external master could be fast enough to do this.

31

VME Data Buffers, U3000 to U3006

VME Data Buffers, U3000 to U3006 ..... **311**

31.1. 16-Bit Operation 311
31.2. 32-Bit Operation 312
31.3. CPU Cycles 312
31.4. DVMA Cycles 312



31

VME Data Buffers, U3000 to U3006

The VME Data Buffer section is shown on page 30 of the schematics, and is composed of one Data Buffer Control PAL, U3000, and six bidirectional octal latch/transceivers, U3001-6. Along with the obvious buffering function, this subsystem implements a data multiplexer that shuffles data among both halves of both the VMEbus and the P2 bus according to the size and address alignment of the cycle. The basic problem is that when performing a 16-bit data transfer, the VMEbus uses data lines 0-15 while the 68020 uses data lines 16-31. During a 32-bit data transfer, however, all 32 bits of the VMEbus align with the like-numbered bits on the P2 bus.

The same set of buffers are used for CPU cycles and DVMA cycles, but the direction of data flow is reversed. That is, data travels the same direction for a CPU read as it does for a DVMA write. Likewise, it travels the same direction for a CPU write as it does for a DVMA read.

31.1. 16-Bit Operation

The figure labelled "Carrera VME Diagnostic PROMs Data Paths," (in Appendix A) sections A-F, show the different operating modes used in TYPE2 space, the VME 16-bit data space.

- When the CPU writes or reads, the top half of the CPU data bus is connected to the bottom half of the VME data bus (sections A and B of the diagram) via the "cross buffers."
- When an external VME master writes to the P2 bus, the bottom half of the VME data bus is enabled onto both halves of the P2 data bus (sections C and D). This is done because the information required to decide which half of the P2 bus is being accessed is not available until later in the cycle.

On DVMA reads, however, we must decide which half of the P2 data bus is being read in order to avoid data buffer conflicts.

- If address bits P1_A[01:00] are 00, this means the top half of the P2 data bus is being accessed and the "cross buffers" are used.
- If address bits P1_A[01:00] are 10, the bottom half of the P2 data bus is being accessed, and the "longword buffers" are used.†

† Although this enables the top half of the P2 data bus onto the top half of the VME data bus, it is not relevant and doesn't hurt anything.

31.2. 32-Bit Operation

32-bit operation is much simpler. The 32-bit P2 data bus always lines up with the 32-bit VME data bus, and the longword buffers are always used. Data flows out from the 2060 board on CPU writes and VME reads, and flows into the 2060 board on CPU reads and VME writes.

31.3. CPU Cycles

The various enables for the data buffers are little state machines that start at state 4 on writes and state 1 on reads. Once the signals are asserted they are held until the end of the cycle because the type bits coming out of the MMU can glitch, which can cause P_BLWORD to glitch, which could cause certain signals to be illegally active at the same time. On writes, the enables are also held during Freeze cycles. See the section on the VME Select and Freeze PAL for more information on Freeze cycles.

On writes, the transparent latches open at the beginning of the cycle and remain open for the duration of the cycle. On reads, the latches open at state 6 and likewise remain open for the entire cycle.

31.4. DVMA Cycles

The enables for data out on reads are state machines during DVMA cycles. They start at state 4 and hold the data valid until the external master ends the cycle as indicated by negating the VME data strobes. Note that this can be significantly after the end of the P2 bus cycle, and that the CPU can have gone on its way much earlier because the data is latched at state 7.

On writes, the enables are purely combinatorial. The latches are closed at state 4.

32

Direct Virtual Memory Access

Direct Virtual Memory Access **315**

32.1. A Generic DVMA Cycle 315
32.2. Optimizations to the DVMA Cycle 316
Back-to-Back DVMA 316
Ethernet Hold 316
VMEbus Lock 316
32.3. Refresh as a Special Case 316
32.4. The DVMA Strobe PAL (U2410) 317
Input and Output Signals 319

32

Direct Virtual Memory Access

Direct Virtual Memory Access, or DVMA, is Sun's method of allowing devices other than the CPU to access the system's main memory, and allowing them to use virtual addresses rather than physical ones when doing so. This means the addresses provided by a DVMA device are translated by the Memory Management Unit, or MMU, just like those provided by the CPU. This simplifies the software by not requiring DVMA devices to perform their own address translation. The simplest way of approaching DVMA is to think of it as literally replacing the CPU during DVMA cycles, generating all the identical signals with identical or better timing.

The DVMA circuitry consists of the DVMA Controller PAL (U2409), an input synchronizer (U2408), the bus lock flip-flop (U2407), and the DVMA Strobe PAL (U2410).

The DVMA Controller is a 20R8 PAL that receives

- requests for the use of the P2 bus,
- requests the bus from the CPU,
- grants the bus to the DVMA devices, and
- handles assertion and negation of the address strobe on DVMA cycles.

Requests can be generated by the Refresh subsystem, the Ethernet Controller, and the VME Slave Interface and are prioritized in that order, which is to say that if two requests are pending at the point when the DVMA Controller gets the bus from the CPU, the one with higher priority gets serviced first.

32.1. A Generic DVMA Cycle

The DVMA Controller State Diagram is shown in the figure titled, "DVMA Controller State Machine, Fig. 2.1," in Appendix A. Rather than attempt to describe every transition here, we will present a simplified version of a generic DVMA cycle. Details will become clearer when we go through the individual cycles in later sections.

A DVMA cycle begins when a device requests the P2 bus via one of the signals R_DMAREQ, E_DMAREQ, or XREQ (which particular one it is doesn't concern us at the moment). These signals are run through synchronizers and into the DVMA Controller, which will assert P_BR to request the bus from the CPU. Approximately three clocks later the CPU will respond with Processor Bus Grar (P_BG). The DVMA Controller then waits until the current CPU cycle has

completed by waiting for CS3, the delayed version of the processor address strobe, to go away. At this point it will decide which device should be granted the bus based on the fixed priority mentioned above, and will assert one of the DMA enables R_DMAEN, E_DMAEN, or X_DMAEN.

On the next clock the DVMA controller will issue the appropriate address strobe (D_AS for Ethernet and VME, REFR for Refresh), and will start waiting for the end of the cycle. On Ethernet & VME cycles the end of the cycle is indicated by the assertion of S_ACK, while on Refresh cycles it is indicated by R_SSAS. When the response is received the DVMA Controller will negate the address strobe, negate Bus Grant Acknowledge, negate the DMA enable, and then return to the IDLE state.

32.2. Optimizations to the DVMA Cycle

There are several optimizations not mentioned above in the simplified description.

Back-to-Back DVMA

First, the DVMA Controller need not go through the IDLE state between each cycle of DVMA. If a request from another DVMA device is present at the point where the DVMA Controller negates address strobe and P_BACK, address strobe will be negated but P_BACK will be held so that the CPU can't get back on the bus. The next DVMA cycle will then be performed before the next CPU cycle. This avoids the overhead associated with transferring control of the P2 bus back and forth between the CPU and the DVMA Controller, which amounts to four clocks for each round trip.

Ethernet Hold

Secondly, it is possible for devices to keep control of the P2 bus even when they aren't using it. The Ethernet interface can assert S_EHOLD to retain control of the bus. While S_EHOLD is asserted the DVMA Controller will continue to assert P_BACK, keeping the CPU off the P2 bus. Refresh cycles will still get access to the bus, however, as will VME cycles as long as the Ethernet isn't attempting to perform a cycle immediately. The Ethernet chip will assert EHOLD for as long as it thinks it might be needing the bus.

VMEbus Lock

The VME Slave Interface can also keep control of the P2 bus from reverting back to the CPU by engaging lock mode. This happens when an external VME master is "fast" in turning around between cycles. "Fast" is defined here as less than 200 nanoseconds between generation of P1_DTACK and start of the external master's next cycle, as indicated by re-assertion of P1_AS and either one or both of P1_DS0 and P1_DS1. Bus lock cycles will be covered in more detail later.

32.3. Refresh as a Special Case

Refresh cycles are analogous to Ethernet and VME Slave cycles, but have a few differences.

1. The first difference is that instead of asserting the DVMA address strobe, D_AS, the DVMA Controller asserts REFR, which is buffered to form P2_REFR. This acts like an address strobe, but insures that no decodes or enables will be generated anywhere on the board. Since there is no real address strobe there will be no DSACKs generated, therefore we have to know when to end the cycle independent of DSACK. This function is performed by R_SSAS (Refresh Sync'd Sync'd Address Strobe), the REFR

signal fed through two flip-flops so that it is delayed by two system clock periods. (REFR used to be called R_AS, or Refresh Address Strobe, at which point these names made more sense.)

2. A second difference between Refresh and the other DVMAs is that P_BACK is released one clock before the "address strobe" (REFR) is released. This information is provided by R_SAS. On Ethernet and VME cycles P_BACK is released at the same time as address strobe (D_AS). This is made possible by the fact that we know exactly how long each Refresh cycle is, so we can pipeline the end of the cycle. On Ethernet and VME cycles we don't know exactly when the end of the cycle will be since it is legal to DMA into Video memory, which has an indeterminate response time.

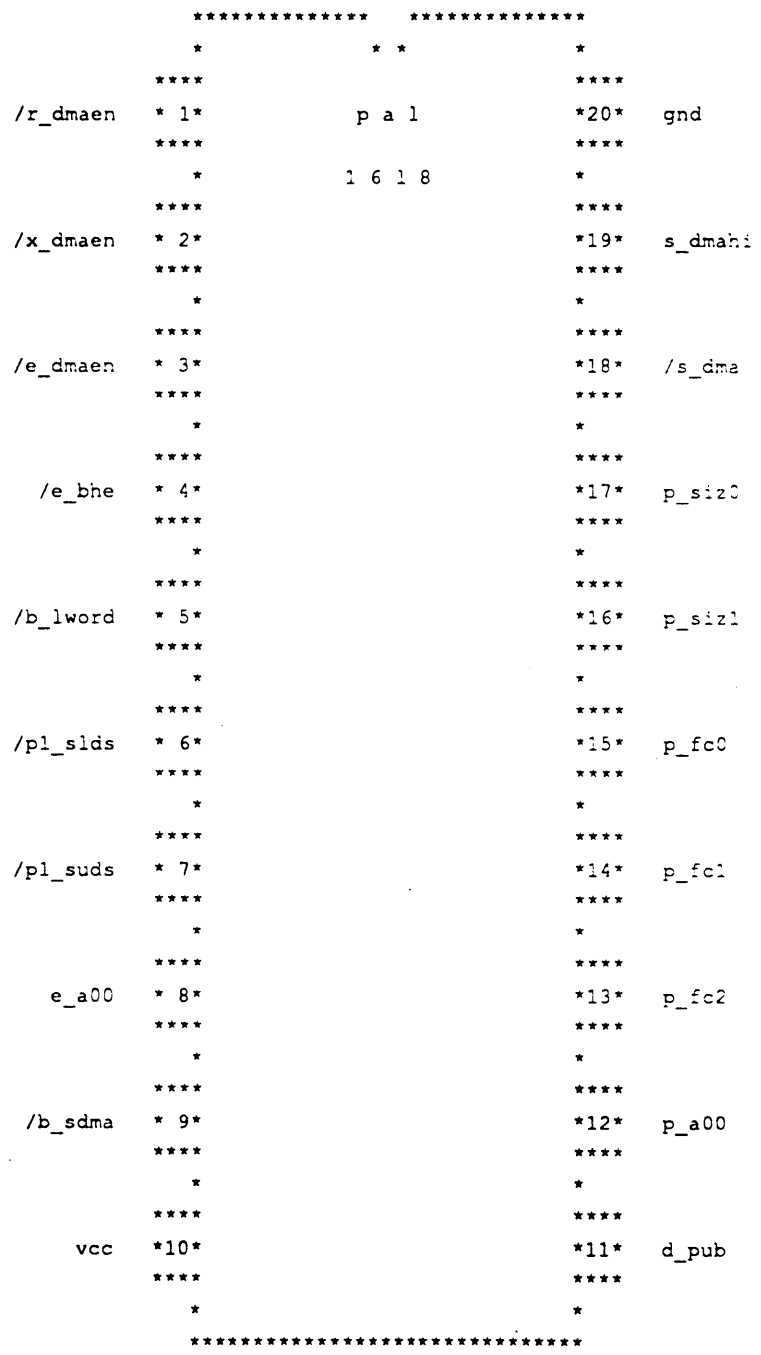
3. The third difference between Refresh and other forms of DVMA is that EHOLD and B_LOCK don't work after a refresh cycle. By that we mean that the bus is always given back to the CPU after a refresh cycle as a sort of fail-safe feature in case one of those subsystems breaks down. If this happens the CPU will still be able to limp along (albeit extremely slowly).

32.4. The DVMA Strobe PAL (U2410)

The DVMA Strobe PAL generates the P2 bus control signals (with the exception of address strobe, which is generated by the DVMA Controller) that are required during DVMA cycles. These signals include the processor function codes P_FC[2:0], processor address bit zero P_A[00], and the processor size bits P_SIZ[1:0]. In addition, both polarities of the S_DMA are generated, a signal that indicates that the current cycle is an Ethernet or VME Slave cycle. Since DVMA can be thought of as replacing the CPU during DVMA cycles, it is essential that all the necessary control signals that would otherwise be generated by the CPU be generated somewhere in the DVMA section. Note that processor data strobe P_DS is not used anywhere in the 2060 design, so it is not generated here. The encodings of the size and function code bits match those of the 68020.

Pinout of the U2410 PAL is:

Figure 32-1 *U2410 Pinout*



Input and Output Signals

Inputs to the U2410 PAL are:

```
r_dmaen  - active during refresh DVMA cycles

x_dmaen  - active during external DVMA cycles, when
           the 2060 board is in VME slave mode.

e_dmaen  - active during ethernet DVMA cycles.

b_lword  - signal from an external DVMA master that is
           asserted during a longword transfer.

p1_slds,
p1_suds  - buffered versions of the VMEbus upper
           and lower data strobes, asserted to
           indicate activity on the corresponding
           byte of the VME data bus.

e_a00  - ethernet address line zero

b_sdma  - VME system direct memory access.  Asserted during
          external DVMA cycles that are occurring in the
          system DVMA part of the address space, as defined
          in the Sun-3 architecture manual.

d_pub  - used only during board test to check the operation
         of the p_fc0 output.
```

Outputs from the U2410 PAL are:

```
p_fc0,
p_fc1,
p_fc2  - processor function codes, used by the 68000
         family to generate 8 separate address spaces.

p_siz0,
p_siz1  - processor size codes, used by the 68020 for
          dynamic bus sizing, indicating the number of
          bytes of the data bus that are involved in
          the current transfer.

p_a00  - processor address line zero.

s_dma  - this signal is asserted whenever some type of DMA is in
         progress.
```

The signals generated are encoded according to the MC68020 User's Manual as follows (0=> zero volts, 1=> five volts):

Table 32-1 *MC68020 Data Size Output Encodings*

siz1	*siz0*	*size*
0	1	byte
1	0	word
1	1	3-byte
0	0	longword

Table 32-2 *MC68020 Function Code Output Encodings*

p_fc2	*p_fc1*	*p_fc0*	*cycle type*
0	0	0	undefined reserved
0	0	1	user data space
0	1	0	user program space
0	1	1	undefined
1	0	0	undefined reserved
1	0	1	supervisor data space
1	1	0	supervisor program space
1	1	1	cpu space

33

Sample Cycles

Sample Cycles **323**

33.1. VME Master Cycles 323
CPU Access of Idle VMEbus 323
CPU Access of a Busy VMEbus 324
CPU Access of VMEbus, Currently Bus Master 325
CPU Rerun During VME Access 325
Relinquishing the VMEbus 326
33.2. VME Slave Cycles 327
VME Device Accesses P2 Bus 327
Lock Mode Cycles 328
VME Device Initiates P2 Bus Lock 328
VME Device Ends P2 Bus Lock 329
VME Device Not Fast Enough to Initiate P2 Bus Lock 329
33.3. Ethernet Cycles 329
33.4. Refresh Cycles 330

33

Sample Cycles

In this section we will present timing diagrams and discuss how the various circuits previously described interact to produce this behavior.

33.1. VME Master Cycles

The VME Master Interface operates in several different ways depending on:

- whether or not the 2060 board currently has control of the VMEbus,
- the activity currently on the VMEbus, and
- the response time of the VME device being addressed.

CPU Access of Idle VMEbus

In order to use the VMEbus the CPU must first obtain control of it. The simplest case is when the VMEbus is currently idle, which means that P1_BBSY, P1_AS, P1_DS[1:0], P1_DTACK, and P1_BERR are all inactive. This situation is shown in the timing diagram labelled "CPU Access of Idle VMEbus," in Appendix A.

1. The CPU starts a cycle by asserting P2_AS at state 1.
2. By state 4 the MMU has finished its address translation and indicates that this is a VME Master cycle by asserting MMU_VME.
3. MMU_VME goes into the VME Select and Freeze PAL U2701, causing it to assert B_SSEL at state 5.
4. B_SSEL goes into the VME Arbiter/Requester PAL U2704, which observes that the VMEbus is idle by checking that P1_BBSY and P1_AS are inactive.
5. On the next clock U2704 asserts P1_BBSY to take control of the VMEbus and B_AEN to indicate to the VME Master Interface that we have control of the VMEbus.
6. The VME Master Controller PAL U2806 receives B_AEN, checks also that the VMEbus is idle by making sure that P1_DTACK and P1_BERR are inactive, and asserts B_OECPU immediately to enable the VME Master Address Latches onto the VMEbus.
7. The VME Data Buffer Control PAL also receives B_OECPU and enables the VME Data Buffers onto the VMEbus in the case of a write, or onto the P2 bus in the case of a read.

8. At this point we wait two clocks to satisfy the VMEbus address-to-address strobe setup requirement of 35 nanoseconds by running B_OECPU through two flip-flops clocked by the system clock and wait for the result.
9. When this delayed signal, B_SSOE, is received by the VME Master Controller PAL U2806, it will immediately assert P1_AS, followed one PAL delay later by the appropriate VME data strobes P1_DS[1:0] and B_ACKEN, which allows the acknowledges (P1_DTACK & P1_BERR) to flow to the CPU.
10. At this point, state 11, the 2060 board simply waits for a response from the addressed VME slave. In the diagram, this response is P1_DTACK received just before state 15. (If the VME slave takes a long time to respond — more than 2.88 microseconds — the 2060 board will perform a rerun cycle, which will be examined later.)
11. The VME Slave's response (P1_DTACK in our scenario) is synchronized to the falling edge of the system clock in flip-flop U2804, then fed to the DSACK PAL U204.
12. DSACK PAL U204 generates P_DSACK1 for 16-bit data transfers or P_DSACK1 and P_DSACK0 for 32-bit data transfers.
13. The CPU receives this on the next falling edge, and on the following falling edge negates P2_AS.
14. Negation of P2_AS causes the VME Master Controller PAL U2806 to immediately negate the VME address and data strobes and clear B_DTACK from flip-flop U2804 by negating B_ACKEN.
15. The VME slave responds by negating P1_DTACK at some arbitrary time in the future.
16. At state 1 of the next cycle, B_SSEL will go away and the cycle will be finished.

CPU Access of a Busy VMEbus

This situation — an external VME master is using the VMEbus to access an external VME slave just as the CPU attempts to use the VMEbus — is shown in the figure labelled "CPU Access of Busy VMEbus," in Appendix A. As you can see, this cycle takes significantly longer and appears much more complex, but all of the differences occur before the start of the CPU's VME cycle.

1. In this case, as before, the CPU starts a VME Master cycle and B_SSEL is asserted at state 5.
2. The VME Arbiter/Requester PAL U2704, however, sees that P1_BBSY is active when B_SSEL becomes asserted, so instead of jumping to the MASTER state it jumps to the BUSREQ state, where it asserts B_BROUT (which is buffered to form P1_BR3).
3. The external VME master that is using the bus will eventually respond to this bus request by releasing P1_BBSY, although there is no VME specification for the maximum time allowable for this.

4. When the VME Arbiter/Requester sees this P1_BBSY released it will assert B_BBOUT (which gets buffered externally to form P1_BBSY).
5. The VME Arbiter/Requester then jumps to the WAITREQ state.
6. On the next clock the VME Arbiter/Requester jumps to the MASTER state if B_SAS has gone away by then (indicating P1_AS went away earlier).
7. If B_SAS is still present at this point, the Arbiter/Requester will jump to the WAIT state to wait for it to go away.

From here on the cycle is the same as that for the idle VMEbus as discussed above.

CPU Access of VMEbus, Currently Bus Master

Since the 2060 board uses a Release-On-Request Arbiter, it will retain control of the VMEbus until an external master requests it. The advantage of this is made clear in the figure labelled "CPU Access of VMEbus (Currently Bus Master)," in Appendix A, where we see that the cycle is roughly three clocks shorter. This is because we already have the addresses and data enabled onto the VMEbus during this time; when we discover at state 5 that the CPU is accessing the VMEbus, we can assert the VME address and data strobes immediately. The VME address-to-address strobe setup and data-to-data strobe setup times will already have been met. Other than that, the cycle is identical to those described earlier.

CPU Rerun During VME Access

When the VME slave currently being addressed has a response time of greater than 2.88 microseconds, or when it takes the CPU a while to gain control of the VMEbus, the CPU will be instructed to rerun its cycle. This will cause it to re-arbitrate for the P2 bus and allow other pending DVMA cycles to complete before it re-attempts the cycle. This situation is shown in the figure labelled "CPU Rerun During VME Access (Currently Bus Master)," in Appendix A.

1. B_ENTO is asserted on the next falling edge after B_SSEL, causing the VME Rerun Timer U2700 to start counting system clocks.
2. At state 71 U2700 will assert B_TOLAT.
3. This forces the outputs of the OR gates in the VME DTACK and BERR Input Latch (U2802) to go high, which means that P1_DTACK and P1_BERR arriving after that point will have no effect.
4. Since the flip-flops at U2804 are self latching, we need to allow them some time to settle, as an acknowledge might have been received that didn't meet the setup requirement. Then we need to allow enough time to see if the acknowledge gets through to the CPU, which will respond by negating its address strobe, P_AS. This settle-and-wait time period is generated by waiting for bit 4 of the VME Rerun Timer (B_TORRN) to go high, which will occur at state 103 if no acknowledge has reached the CPU.
5. B_TORRN causes the VME Freeze and Select PAL U2701 to assert B_FREEZE and B_RERUN on the next falling edge of the system clock.
6. B_FREEZE goes to the VME Master Interface to make sure that all signals on the VMEbus remain in their current states.

7. B_RERUN goes to PAL U107, where it is combined with the rerun signal from the Floating Point Accelerator before it proceeds to the Bus Error PAL, U202.
8. B_RERUN also causes the VME Select and Freeze PAL to negate B_ENTO.
9. Negation of B_ENTO clears the VME Rerun Timer in anticipation of the next rerun cycle.
10. The Bus Error PAL asserts P_BERR and P_HALT on the next falling edge, which is the indication to the CPU that it should rerun the cycle.
11. The CPU sees these signals on the next falling edge.
12. On the falling edge after that the CPU negates P_AS.
13. On the falling edge after that, the VME Select and Freeze PAL will respond to the negation of P_AS by negating B_RERUN.
14. On the next falling edge the Bus Error PAL responds to the negation of B_RERUN by negating P_BERR and P_HALT.
15. In addition, B_ENTO will be asserted again so that the VME Rerun Timer can begin timing for the next rerun.
16. At this point the CPU will either rerun its cycle or, if a DVMA request is pending, a DVMA cycle will be performed.

Relinquishing the VMEbus

When an external VME device requests the VMEbus while the 2060 board is performing a VME cycle, we will end the cycle in a different way as shown in the timing diagram labelled "CPU Relinquishes VMEbus (Currently Bus Master)," in Appendix A, release P1_BBSY as soon as we detect both P1_BR3 and P1_AS, so that the VMEbus grant will be able to make its way down the daisy chain while the last cycle is in progress, overlapping the two processes. The cycle goes like this:

1. In the timing diagram we see the external device assert P1_BR3, which gets synchronized to the system clock in U2703 to form B_SBR.
2. The VME Arbiter/Requester PAL U2704 will respond on the next falling edge by jumping to MASTER_NG state and negating P1_BBSY and B_AEN.
3. When B_SBBIN, the synchronized input of our own output, goes away, the VME Arbiter/Requester will assert P1_BG3OUT, which will trickle down the daisy chain until it reaches the board that is currently requesting a cycle.
4. In the meantime we see the CPU's cycle completing with the receipt of P1_DTACK, which is synchronized to form B_DTACK.
5. Three clocks later the CPU responds by negating its address strobe, P2_AS, but here is where the differences start. The VMEbus Specification requires that if P1_BBSY is negated before the end of the last cycle in order to realize the pipelining mentioned above, then the master's address and data drivers must be disabled before the master negates its address strobe, P1_AS. To accomplish this the VME Master Controller PAL U2806 senses

that the bus has been given up by noting that B_AEN is no longer asserted, even though it is still holding its address strobe, B_AS, asserted. Due to this it continues to hold B_AS while first the data strobes are negated and then B_OECPU is negated.

Other than that, the cycle is identical to other cycles.

33.2. VME Slave Cycles

VME Slave cycles occur when an external VME device gets control of the VMEbus and accesses the main memory or video memory on the 2060 P2 bus. In this section we will cover a typical VME Slave cycle and discuss Lock Mode, the high-speed access mode automatically engaged by a fast VME master.

VME Device Accesses P2 Bus

The timing diagram labelled "VME Device Acquires VMEbus and Accesses P2 Bus," in Appendix A, shows a typical VME Slave cycle.

1. The cycle starts with the external VME device requesting control of the VMEbus by asserting P1_BR3.
2. P1_BR3 is synchronized to the system clock to form B_SBR, which feeds into the VME Arbiter/Requester PAL U2704.
3. U2704 asserts B_BGOUT (which is the same signal as P1_BG3OUT) since the VMEbus is currently IDLE.
4. The external master responds to the bus grant by asserting P1_BBSY, indicating that is has taken control of the VMEbus.
5. This is received by the VME Arbiter/Requester as B_SBBIN, causing it to release the bus grant signal.
6. After the external master takes control of the VMEbus it will enable its addresses and data onto the VMEbus and assert its address and data strobes P1_AS and P1_DS[1:0].
7. The addresses will be latched in the VME Slave Address Latches U2901-2 and U2911-3, and be decoded by the VME Slave Address Decoder U2907.
8. B_USPC enables the User DVMA Context Selector U2908, which indicates that an enabled context has been chosen by asserting B_UDMA. (In this example we show a User DVMA cycle, as indicated by the assertion of B_USPC by the decoder.)
9. B_UDMA in association with P1_SAS and P1_DS causes the VME Slave Request PAL U2904 to assert XREQ.
10. XREQ gets synchronized to the system clock to form S_XREQ.
11. Assertion of S_XREQ causes the DVMA Controller PAL U2409 to assert Processor Bus Request P_BR on the next falling edge.
12. About three clocks later the CPU will respond by asserting Processor Bus Grant P_BG.
13. The DVMA Controller will then wait until the delayed version of P_AS, CS3, goes away and then asserts Processor Bus Grant Acknowledge P_BACK to take control of the P2 bus, and X_DMAEN to enable the VME

Slave Interface onto the P2 bus.

14. On the next falling edge it asserts the DVMA Address Strobe D_AS, which is buffered to form P_AS.
15. On the falling edge after that P_BR will be released, leaving only P_BACK to hold onto the P2 bus.
16. Sometime later the CPU will respond to the negation of P_BR by negating P_BG.
17. The point at which D_AS is asserted becomes state 1 of the DVMA cycle, which will proceed identically to a CPU cycle. In our example an access of main memory is shown.
18. On main memory accesses P_DSACK1 is asserted at state 4, which will cause S_ACK to go active at state 5.
19. The DVMA Controller will respond to this by negating D_AS and P_BACK at state 7, ending the cycle and turning the P2 bus back over to the CPU.
20. In the case of a VME read cycle, the data will be latched into the VME Data buffers at state 7.
21. These data buffers will remain enabled onto the VMEbus until the external master negates the data strobes P1_DS[1:0].
22. At state 9 P1_DTACK will be asserted to indicate to the external VME master that the transfer has been completed, and X_DMAEN will be negated. P1_DTACK occurs at state 9 instead of state 7 for two reasons: to satisfy the VME data-to-DTACK setup time and to allow time for parity checking. The VMEbus doesn't allow simultaneous assertion of DTACK and BERR, so the data must be checked before asserting DTACK.
23. The external master will respond to P1_DTACK by negating its VME address and data strobes.
24. This will cause P1_DS to go away.
25. When P1_DS is deasserted, the VME Slave Request PAL negates P1_DTACK.

Lock Mode Cycles

VME Device Initiates P2 Bus Lock

In the timing diagram labelled "VME Device Initiates P2 Bus Lock," in Appendix A, we see the circumstances surrounding the initiation of P2 bus lock mode, comprised of the tail end of a VME Slave cycle, a CPU cycle, and two more VME Slave cycles. The end of the first VME Slave cycle is identical to that shown in the previous section, but a new signal relevant to lock mode operation is shown: B_5SXDMA. This is simply the VME Slave Address Enable signal, X_DMAEN, delayed by passing through 5 flip-flops clocked on the falling edge of the system clock. It can be thought of as timing a period of 300 nanoseconds from the end of the Slave cycle, or 240 nanoseconds from the generation of P1_DTACK. If, within this period, a new DMA request is received from the external VME Master in the form of a valid XREQ, we will enter Lock Mode.

This is accomplished by clocking the state of XREQ into a D flip-flop (U2407) on the trailing edge of B_5SXDMA.

The output of U2407 flip-flop is B_LOCK, the indication of Lock Mode. We see B_LOCK being asserted in the timing diagram, but it occurs too late to lock out the current CPU cycle, which is already in progress. Instead, the DVMA Controller goes through the normal process of requesting and being granted the local bus, and performing the first Slave cycle.

At this point, in the absence of B_LOCK, the local bus would be returned to the CPU. Since B_LOCK is asserted in this case, however, the bus is kept by the DVMA controller and prepared for the start of the next cycle by re-asserting X_DMAEN. This allows the next Slave cycle to start on the next falling edge of the system clock if XREQ is present. From now on, as long as a new XREQ is present at the trailing edge of B_5SXDMA, we will remain in Lock Mode.

A fail-safe mechanism is provided which returns the local bus to the CPU for one cycle after every refresh cycle, in case Lock Mode gets stuck in the "on" state. Note that Lock Mode does not lock out refresh or Ethernet cycles, both of which still retain their higher priority.

VME Device Ends P2 Bus Lock

When the external VME Master stops initiating cycles within the window timed by B_5SXDMA, B_LOCK will be negated and control of the P2 bus will be returned to the CPU. This situation is shown in the timing digram labelled "VME Device Ends P2 Bus Lock," in Appendix A, where we see the last VME Slave cycle. (A certain amount of time is wasted at the end of Lock Mode because the DVMA controller holds onto the P2 bus for approximately six clocks before deciding that it is no longer required.)

VME Device Not Fast Enough to Initiate P2 Bus Lock

If the external VME Master is not fast enough to initiate bus lock, the CPU will perform a bus cycle in between most Slave cycles. Hits on the internal 68020 cache will lower the bus utilization so that this doesn't always happen, but about half of the time it will. The situation where the CPU performs a cycle in between each Slave cycle is shown in the timing diagram labelled "VME Device Not Fast Enough to Initiate P2 Bus Lock," in Appendix A.

33.3. Ethernet Cycles

The Ethernet interface runs off of an 8 megahertz clock that is asynchronous to the CPU clock, so Ethernet cycles take a variable amount of time depending on the relationship of the two clocks at the time the cycle starts. The timing diagram labelled "Ethernet Cycle (Fastest Case)," in Appendix A, shows the fastest case, which occurs when the clocks have the optimal relationship. The timing diagram labelled "Ethernet Cycle (Slowest Case)," in Appendix A, shows the slowest case.

1. An Ethernet cycle starts when the 82586 asserts either E_RD or E_WR, which signify either a read or a write.
2. These signals are synchronized to the 8 megahertz Ethernet clock, E_C125, in flip-flop U2500† to form E_SRD and E_SWR.

†Due to BITS, or Bizarre Intel Timing Specs.

3. On the next rising edge of E_C125 the Ethernet Control PAL, U2501, will assert E_AS, or Ethernet Address Strobe.
4. E_AS goes to the Ethernet Request Flip-Flop, U2407, where it clocks in the Ethernet DMA Request (E_DMAREQ) as long as there have been no Ethernet errors.
5. E_AS also clocks the addresses into the Ethernet Address Latches, U2512-15.
6. E_DMAREQ gets synchronized to the CPU clock in U2408 to form S_EREQ.
7. S_EREQ goes into the DVMA Controller (U2409), causing it to generate the Ethernet Address Enable signal, E_DMAEN, as soon as the CPU is not using the P2 bus.
8. This enables the Ethernet Address Latches onto the P2 bus, and will be followed on the next system clock by the DVMA Address Strobe, D_AS.
9. As soon as E_DMAEN is asserted, E_CLR will clear E_DMAREQ, preparing it for the next cycle.
10. A normal memory cycle will be performed and D_AS will be negated at state 7, causing flip-flop U2504 to assert E_READY to the 82586.
11. The 82586 will recognize this and end its cycle two E_C125 clocks later, after an internal synchronization delay.

The 82586 multiplexes addresses with the data on all 16 data lines, so some contortions are required in order to keep from having contention on these lines. On reads, signal E_RDT1 indicates the T1 period in Intel parlance. On the next rising edge of E_C125, one of E_RDU or E_RDL goes active, depending on which half of the P2 bus is being accessed. These signals enable the Ethernet Data Buffers U2508-11 onto the local Ethernet address/data bus at a time guaranteed to be *after* the 82586 stops driving these lines with address information. The read data is latched into these latches at state 7, with the negation of D_AS.

33.4. Refresh Cycles

A typical refresh cycle is illustrated by the timing diagram labelled "Refresh Cycle," in Appendix A.

Refresh cycles are requested by free running counter U2400 approximately every 15 microseconds — every time bit 7 goes high. The counter is reset when it reaches binary count 10011001, counting off a 100 nanosecond clock.

1. Bit 7 of the Refresh Period Counter U2400 clocks the Refresh Request Flip-Flop U2402, which generates R_DMAREQ.
2. This is synchronized to the system clock by U2408, then fed into the DVMA Controller U2409 in the form of S_RREQ.
3. Referring to the timing diagram, you will see that the DVMA Controller goes through its normal bus request/bus grant/bus grant acknowledge routine and asserts R_DMAEN as soon as it gets control of the P2 bus, enabling the refresh addresses onto the P2 bus (U2403-5).

4. On the next clock, REFR, a signal analogous to an address strobe, is asserted.
5. This signal goes to the RAS PALs U3100 and U3102 after being buffered in U704.
6. RAS PALs U3100 and U3102 assert RAS to all banks of RAM simultaneously.
7. REFR is delayed by two flip-flops to form R_SAS and R_SSAS.
8. R_SAS causes the DVMA controller to release P_BACK, so that the CPU can start the process of retaking the P2 bus.
9. R_SSAS ends the cycle by telling the DVMA Controller to negate REFR and R_DMAEN.

34

RAS Decode PALs — U3100 and U3102

RAS Decode PALs — U3100 and U3102 ..... **335**

34.1. U3100 and U3102 Pinouts 335
34.2. U3100 and U3102 Input Signals 336
34.3. U3100 and U3102 Outputs 336
U3100 Output Signals 337
U3102 Output Signals 339

34

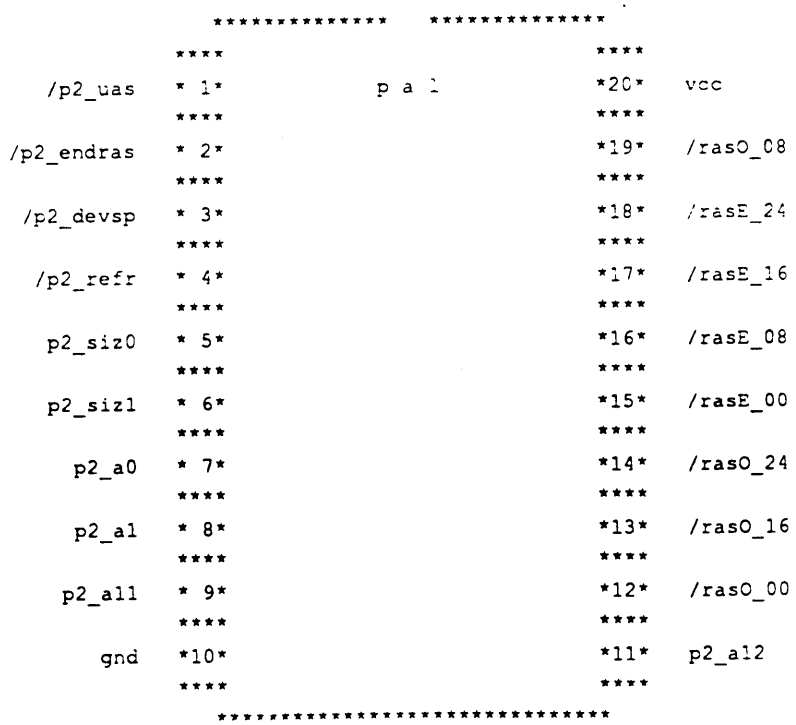
RAS Decode PALs — U3100 and U3102

34.1. U3100 and U3102 Pinouts

Pinouts for the two decoder PALs are generically the same; ras0 signals from U3100 are derived from the same equations as ras2 in U3102; ras1 in u3100 are derived from the same equations as ras3 in U3102. These signals are lumped under the general terms EVEN (rasE) and ODD (ras0). Thus, rasO is connected to the two odd megabytes in the four megabyte space — megabytes 1 and 3. The rasE signals go to the two even megabytes — 0 and 2. In other words,

- RAS to megabyte 0 comes from U3100
- RAS to megabyte 1 comes from U3100
- RAS to megabyte 2 comes from U3102
- RAS to megabyte 3 comes from U3102.

Figure 34-1 *U3100 and U3102 Pinouts*



34.2. U3100 and U3102 Input Signals

Inputs to U3100 and U3102 decoders are:

```
p2_uas-       =  row address is stable - generate RAS

p2_endras-    =  terminate RAS to meet precharge spec

p2_devsp-     =  device space cycle

p2_refr-      =  RAS all DRAM for refresh

p2_siz<1:0>   =  size bits from bus master (to
                   select byte, word, 3-byte or longword)

p2_a<12:11>   =  physical address from MMU (to
                   select 1 of 4 Megs)

p2_a<01:00>   =  physical address from bus master
                   (to select bytes)
```

34.3. U3100 and U3102 Outputs

Outputs from U3100 and U3102 are:

```
                  U3100          U3102
rasE_24-    =     Meg 0          Meg  2 - byte 24

rasE_16-    =     Meg 0          Meg  2 - byte 16

rasE_08-    =     Meg 0          Meg  2 - byte 08

rasE_00-    =     Meg 0          Meg  2 - byte 00

rasO_24-    =     Meg 1          Meg  3 - byte 24

rasO_16-    =     Meg 1          Meg  3 - byte 16

rasO_08-    =     Meg 1          Meg  3 - byte 08

rasO_00-    =     Meg 1          Meg  3 - byte 00
```

Derivations of the individual RAS signals are given below. First, though, a couple of macros need to be defined for convenience.

The "do a RAS cycle" macro is defined as a device space access during a cs2, when RAS is stable, until p2_endras is deasserted.

```
#define DORAS   p2_devsp*p2_uas*/p2_endras
```

The next four definitions define one of four megabytes in the on-board 4 Mbyte physical memory, by decoding address bits p2_a11 and p2_a12.

```
#define MEG0    /p2_a12*/p2_a11     first megabyte
#define MEG1    /p2_a12* p2_a11     second megabyte
#define MEG2     p2_a12*/p2_a11     third megabyte
#define MEG3     p2_a12* p2_a11     fourth megabyte
```

Now we can list the PAL equations for the eight RAS signals issued from U3100 and the eight RAS signals from U3102. Remember that:

- U3100 issues RAS to megabytes 0 and 1;
- U3102 issues RAS to megabytes 2 and 3.

U3100 Output Signals

The following eight signals are issued from U3100.

The RAS strobe for upper data byte (data bits [31:24]) in 0 megabyte space is decoded:

```
rasE_24 =   DORAS*MEG0*/p2_a1*/p2_a0 +
            valid RAS to first megabyte on the board, and 0 byte offset

                p2_refr
            RAS always on refresh
```

RAS issued to byte [23:16] in the 0 megabyte space is:

```
rasE_16 =       DORAS*MEG0*/p2_a1* p2_a0 +
          valid RAS to megabyte 0, data with +1 byte offset

                DORAS*MEG0* p2_siz1*/p2_a1 +
          3 byte or word-sized data, 0 or +1 byte offset

                DORAS*MEG0*/p2_siz0*/p2_a1 +
          1 byte or longword-sized data, 0 or +1 byte offset

                p2_refr
          RAS always on refresh
```

The remaining RAS signals are decoded similarly. RAS to byte [15:08] in megabyte 0 is:

```
rasE_08 =       DORAS*MEG0* p2_a1*/p2_a0 +
                DORAS*MEG0*/p2_siz1*/p2_siz0*/p2_a1 +
                DORAS*MEG0* p2_siz1* p2_siz0*/p2_a1 +
                DORAS*MEG0* p2_siz1*/p2_a1* p2_a0 +
                p2_refr
```

RAS to byte [07:00] in megabyte 0 is decoded:

```
rasE_00 =       DORAS*MEG0* p2_a1* p2_a0 +
                DORAS*MEG0*/p2_siz1*/p2_siz0 +
                DORAS*MEG0* p2_siz1* p2_a1 +
                DORAS*MEG0* p2_siz1* p2_siz0* p2_a0 +
                p2_refr
```

RAS to byte [31:24] in megabyte 1 is decoded:

```
rasO_24 =       DORAS*MEG1*/p2_a1*/p2_a0 +
                p2_refr
```

RAS to byte [23:16] in megabyte 1 is decoded:

```
ras0_16 =       DORAS*MEG1*/p2_a1* p2_a0 +
                DORAS*MEG1* p2_siz1*/p2_a1 +
                DORAS*MEG1*/p2_siz0*/p2_a1 +
                p2_refr
```

RAS to byte [15:08] in megabyte 1 is decoded:

```
ras0_08 =       DORAS*MEG1* p2_a1*/p2_a0 +
                DORAS*MEG1*/p2_siz1*/p2_siz0*/p2_a1 +
                DORAS*MEG1* p2_siz1* p2_siz0*/p2_a1 +
                DORAS*MEG1* p2_siz1*/p2_a1* p2_a0 +
                p2_refr
```

RAS to byte [07:00] in megabyte 1 is decoded:

```
ras0_00 =       DORAS*MEG1* p2_a1* p2_a0 +
                DORAS*MEG1*/p2_siz1*/p2_siz0 +
                DORAS*MEG1* p2_siz1* p2_a1 +
                DORAS*MEG1* p2_siz1* p2_siz0* p2_a0 +
                p2_refr
```

U3102 Output Signals

The following eight signals are issued from U3102 PAL. Remember that DORAS and MEG(3:0) are macros defined as:

```
#define DORAS   p2_devsp*p2_uas*/p2_endras
```

```
#define MEG0    /p2_a12*/p2_a11     first megabyte
#define MEG1    /p2_a12* p2_a11     second megabyte
#define MEG2     p2_a12*/p2_a11     third megabyte
#define MEG3     p2_a12* p2_a11     fourth megabyte
```

RAS to byte [31:24] in megabyte 2 is decoded:

```
rasE_24 =       DORAS*MEG2*/p2_a1*/p2_a0 +
                p2_refr
```

RAS to byte [23:16] in megabyte 2 is decoded:

```
rasE_16 =       DORAS*MEG2*/p2_a1* p2_a0 +
                DORAS*MEG2* p2_siz1*/p2_a1 +
                DORAS*MEG2*/p2_siz0*/p2_a1 +
                p2_refr
```

RAS to byte [15:08] in megabyte 2 is decoded:

```
rasE_08 =       DORAS*MEG2* p2_a1*/p2_a0 +
                DORAS*MEG2*/p2_siz1*/p2_siz0*/p2_a1 +
                DORAS*MEG2* p2_siz1* p2_siz0*/p2_a1 +
                DORAS*MEG2* p2_siz1*/p2_a1* p2_a0 +
                p2_refr
```

RAS to byte [07:00] in megabyte 2 is decoded:

```
rasE_00 =       DORAS*MEG2* p2_a1* p2_a0 +
                DORAS*MEG2*/p2_siz1*/p2_siz0 +
                DORAS*MEG2* p2_siz1* p2_a1 +
                DORAS*MEG2* p2_siz1* p2_siz0* p2_a0 +
                p2_refr
```

RAS to byte [31:24] in megabyte 3 is decoded:

```
rasO_24 =       DORAS*MEG3*/p2_a1*/p2_a0 +
                p2_refr
```

RAS to byte [23:16] in megabyte 3 is decoded:

```
ras0_16 =       DORAS*MEG3*/p2_a1* p2_a0 +
                DORAS*MEG3* p2_siz1*/p2_a1 +
                DORAS*MEG3*/p2_siz0*/p2_a1 +
                p2_refr
```

RAS to byte [15:08] in megabyte 3 is decoded:

```
ras0_08 =       DORAS*MEG3* p2_a1*/p2_a0 +
                DORAS*MEG3*/p2_siz1*/p2_siz0*/p2_a1 +
                DORAS*MEG3* p2_siz1* p2_siz0*/p2_a1 +
                DORAS*MEG3* p2_siz1*/p2_a1* p2_a0 +
                p2_refr
```

RAS to byte [07:00] in megabyte 3 is decoded:

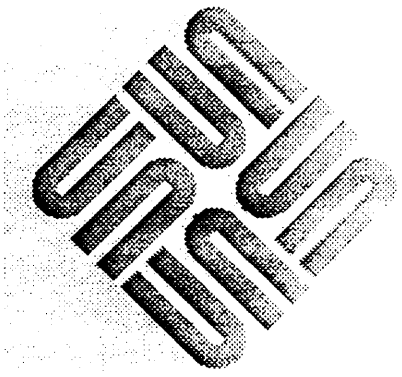
```
ras0_00 =       DORAS*MEG3* p2_a1* p2_a0 +
                DORAS*MEG3*/p2_siz1*/p2_siz0 +
                DORAS*MEG3* p2_siz1* p2_a1 +
                DORAS*MEG3* p2_siz1* p2_siz0* p2_a0 +
                p2_refr
```

35

CAS Decode PAL — U3104

CAS Decode PAL — U3104 ..... **345**

35.1. U3104 Pinout 345
35.2. U3104 Input Signals 346
35.3. U3104 Output Signals 346



35

CAS Decode PAL — U3104

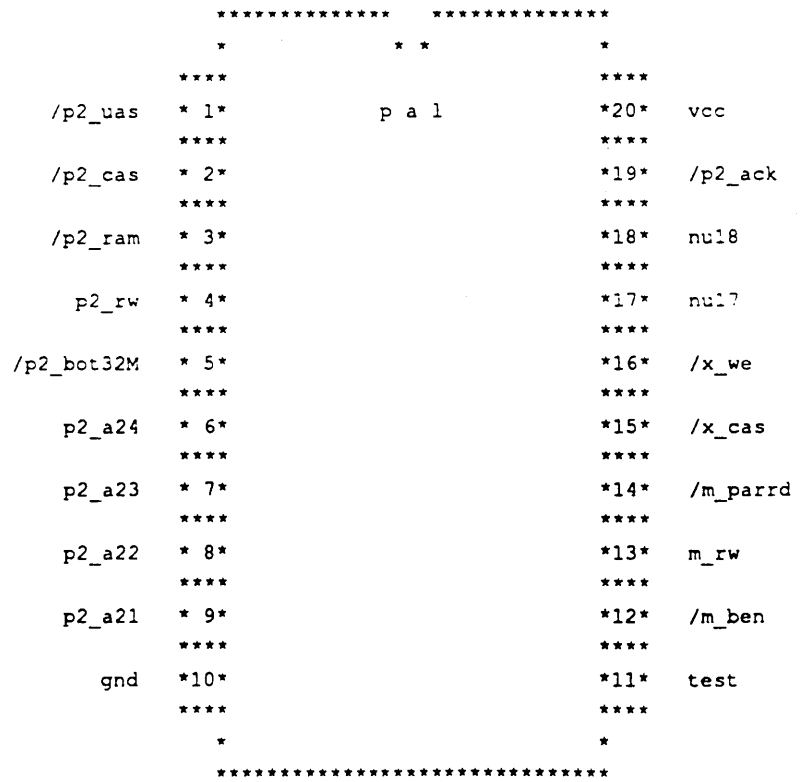
The CAS decode PAL generates eight column address strobe signals — one to each bank of eighteen chips (two bytes and two parity bits). Two of its output signals, x_we- and x_cas-, are buffered by U3105 and U3115 control buffers.

- x_we- is used to generate write enables for the eight banks of memory;
- x_cas- generates the column address strobes for these same eight banks of memory.

35.1. U3104 Pinout

Pinout for U3104 CAS decoder PAL is:

Figure 35-1 *U3104 Pinout*



35.2. U3104 Input Signals

Inputs to the CAS decoder PAL are:

```
p2_uas-      =   "delayed" address strobe (actually cs2-)

p2_cas-      =   output of MMU is stable - generate CAS

p2_ram-      =   validated type 0 cycle

p2_rw        =   read/write-

p2_bot32M-   =   output of upper address comparator
                   (selects bottom 32 Mbytes)

p2_a[24:21]  =   select 2 or 4 Mbytes in a 32 Mbyte
                   address range

test         =   only used when applying test vectors;
                   used to break loops
```

35.3. U3104 Output Signals

Outputs from the CAS decoder PAL are:

```
x_we-        =   modified version of R/W signal, generates
                   write enables for all memory

x_cas-       =   CAS for all memory

m_parrd-     =   gate parity read data to
                   p2_par<24,16,08,00>

m_rw         =   read/write control for the memory
                   data buffers

p2_ack-      =   acknowledge

m_ben-       =   memory data buffer enable
```

Before providing the individual PAL equations for each output signal, there are several macros that must be defined. First, read and write are defined as levels of the p2_rw signal: read is a high and write is a low.

```
#define READ    p2_rw
#define WRITE   /p2_rw
```

Next, two and four megabyte systems must be defined.

```
#ifdef TWOMEG

#define MEG2    /p2_a24*/p2_a23*/p2_a22*/p2_a21
#               2 Mbyte system
#else
#define MEG4    /p2_a24*/p2_a23*/p2_a22
#               4 Mbyte system
#endif
```

Finally, memory on the CPU board is defined as occupying a portion of the bottom 32 Mbytes of address space. M_SEL, or "memory select," defines this space.

```
#define M_SEL   p2_uas*p2_ram*p2_bot32M*ADDR
```

Write enables are generated by a slightly modified version of the R/W signal, and must not change until CAS has been deasserted. Otherwise reads may look like late writes, with R/W changes and CAS still asserted.

```
x_we    = WRITE * p2_uas
          no writes until previous read is complete
```

CAS to the memories is asserted when p2_cas is asserted (cs4) and is deasserted at the end of the cycle (p2_uas invalid).

```
x_cas   = M_SEL*p2_cas +        set CAS
          x_cas*p2_uas*/test    hold until end of cycle
```

This signal gates parity read data to p2_par[24:16:08:00] when selected and doing a read.

```
m_parrd = x_cas * READ * p2_uas
```

The p2_rw signal is passed straight through.

```
/m_rw   = /p2_rw
```

Generate p2_ack as soon as the board is selected. By ANDing with p2_uas, we drive m_ack high before shutting off, since p2_uas- is deasserted before p2_cas or m_sel.

```
if ( x_cas ) p2_ack = p2_uas
```

Turn on the P2 data buffers when selected (doing either a read or a write).

```
m_ben   = x_cas * p2_uas * READ +   selected
          p2_uas * WRITE
```

36

Control Buffers — U3105 and U3115

Control Buffers — U3105 and U3115 ..... **351**

36

Control Buffers — U3105 and U3115

The two control buffers drive cross-coupled write enable and CAS signals to on-board memory. U3105 drives four of the eight write enable and four of the eight CAS signals to on-board RAM; U3115 drives the other four write enables and CAS signals to on-board RAM.

These buffer/drivers are permanently gated ON by the connection of pulldowns to both output enables.

37

Row and Column Address Multiplexers — U3110:07

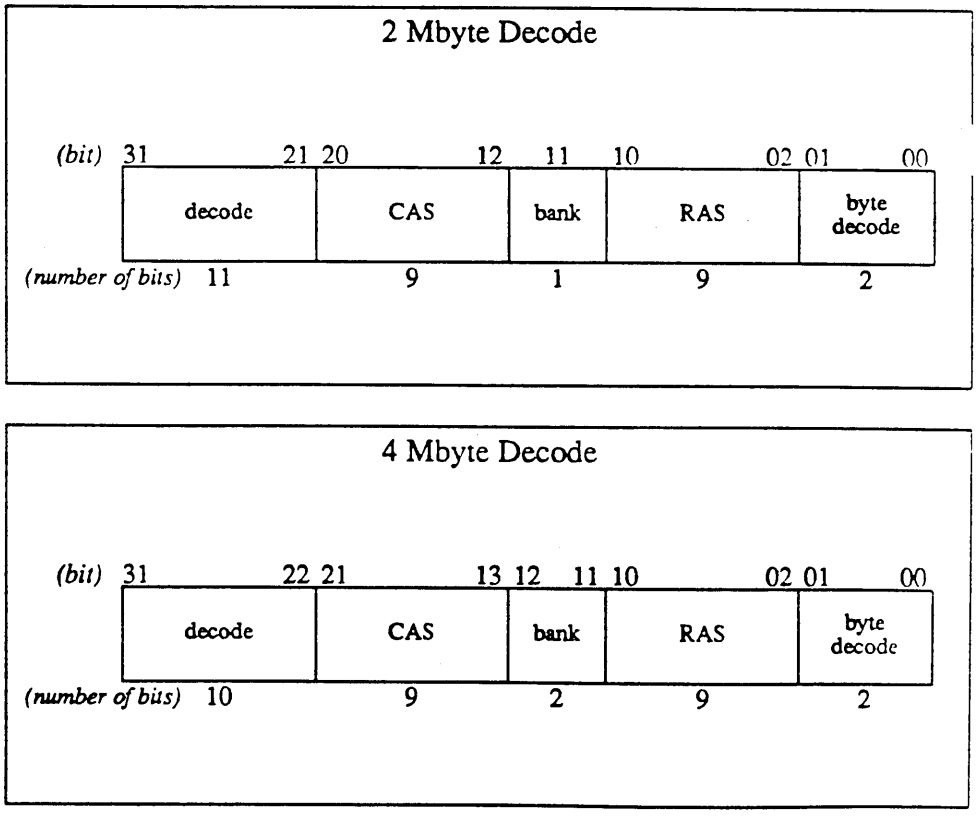
Row and Column Address Multiplexers — U3110:07 ..... **355**

37

Row and Column Address Multiplexers — U3110:07

U3107 through U3110 multiplex the 18 bits of row and column address needed to decode a bit from each of the 256K-by-1 bit memory RAM. However these same address bits are used for an entire 18 chip/bit bank (a 16-bit word with 2 bits of parity) so that taken as a whole, these address bits access a **word** (plus 2 bits of parity) from the entire 2/4 Mbyte memory space — which is why the LSB of the RAS/CAS decode is p2_a(02).

Figure 37-1 *RAS/CAS Decode Bit Assignments for 2 and 4 Mbyte Systems*



During the assertion of p2_mux- (aliased cs3), the row address bits p2a(09:02) are gated onto the m_a(7:0) memory address bus. The ninth bit, p2_a(10) is supplied through U3110. RAS is valid during cs3, and is asserted to the memory

RAM.

With the deassertion of p2_mux- (indicating you are no longer in cs3), column address bits p2_a(20:13) are gated onto the m_a(7:0) memory address bus. The ninth address bit is supplied through U3109 mux and U3110 buffer. CAS is now valid, and is asserted to the memory RAM.

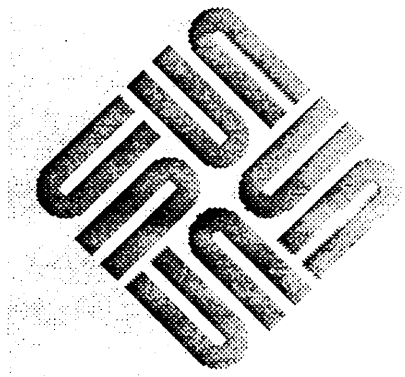
Both row and column address bits are latched into row and column decoder registers inside each RAM chip with the assertion of their respective RAS and CAS signals.

J3101 selects p2_a12 (for a 2 Mbyte system) or p2_a21 (for a 4 Mbyte system).

A

Figures and Timing Diagrams

Figures and Timing Diagrams 363



A

Figures and Timing Diagrams

Index

2

2060 Block Diagram, 15
2060 data paths, 16
2060 mechanical specifications, 19
  board form factor, 19
  connectors, 19
  switches, 19
2060 VME Implementation
  environmental characteristics, 264
  master capabilities, 263
  power characteristics, 265
  slave capabilities, 264
  system controller capabilities, 264
  VME Arbiter and Requester, 269

6

68020, 23
68881
  coprocessor selection, 25
68881 Floating Point Coprocessor, 23

A

aliases, 148

B

bootcy- signal, 27
Bus Transfer Size, 60
Byte Selection in the Page Map RAM, 210

C

cache disable jumper, J100, 23
clkinh signal, 29
clock generation, 101
  U400-U405, 101
control space, 24
control space device addresses, 4
control space devices
  addresses", 4
CPU Access of Idle VMEbus
  state diagram, 323
CPU space, 24
CPU Space PALs, U106 and U107, 24
ctlspc- signal, 28

D

data alignment, 61
device space, 24
devspc- signal, 28
DVMA, 315
  Optimizations to the DVMA Cycle, 316
  Refresh as a Special Case, 316
  Sample Cycles, 323
  U2407, 315
  U2408, 315
  U2409, 315
  U2409 DVMA Controller, 315
  U2410, 315
  U2410 DVMA Strobe PAL, 317
DVMA—Direct Virtual Memory Access, 315
dynamic bus sizing, 62

E

ECL Circuitry, 256
  ECL Clock, 256
ECL Clock
  J2301, 256
  U2303:01, 257
  U2305, 256
  U2306, 257
  U2308 ECL oscillator, 256
  U2312, 256
EEPROM and EPROM, 188
Ethernet Control Register
  U1405 Ethernet Control Write Buffer, 223
  U1407 Ethernet Control Read Buffer, 224

F

fpa_bei- signal, 29
fpp_berr signal, 32
fpp_cs signal, 33
Frame Buffer RAM, 255
  data paths, 255
function codes, 24
  control space, 24
  CPU space, 24
  device space, 24

H

Horizontal State Machine
  U2202, 257

I

I/O devices
  addresses, 5
interrupt acknowledge cycle, 74
interrupt circuitry
  priority, 67
Interrupt circuitry
  U301:U300, J300, 67
interrupt request cycle, 73
interrupts
  spurious, 93

J

J100, 23
J1001, 187
J1100, 187
J2301, 256
J300, 69
  interrupt disable and enable, 69
J300 Interrupt circuitry, 67

L

lberr signal, 54

M

Memory Management Unit
  Page Map RAM, 127
  virtual addressing, 111
Memory Management Unit (MMU), 111
Memory RAM, 359
  Memory Read, 359
  Memory Write, 359
MMU, 111
  protection bits, 141
MOS bus devices
  Boot PROM (EPROM), 165
  EEPROM and EPROM, 188
  J1001, 187
  J1100, 187
  Keyboard, 165
  MOS Decoders, 166
  MOS Read and Write Cycles, 185
  Mouse, 165
  Real Time Clock (RTC), 165
  Serial Ports (Port A and Port B), 165
  Serial Ports A and B, 187
  Time of Day (TOD) Clock, 190
  TYPE1 Space, 155
  U1000 Mouse and Keyboard SCC, 186
  U1100 Serial Ports A and B, 187
  U405 and U2207 Baud Rate Clock, 187
  U900 MOS Enables PAL, 166
  U901 MOS SACK State Machine, 169
  U902 MOS Write Buffer, 184
  U903 MOS Read Buffer, 184
  U904 MOS Read/Write Strobe Decoder, 179
  User-accessible EAROM (EEPROM), 165
MOS Read and Write Cycles, 185

O

Offset Bits, 60
Optimizations to the DVMA Cycle, 316
  Back-to-Back DVMA, 316
  Ethernet Hold, 316
overview of Sun-3 architecture, 3
  68881, 3
  control space devices, 4
  CPU, 3
  DVMA controller, 4
  EEPROM, 5
  Encryption processor, 5
  EPROM, 5
  FPA, 3
  I/O devices, 5
  interrupts, 6
  main memory, 5
  Memory Management Unit, 4
  memory space, 5
  MMU, 4
  Parity Error registers, 5
  refresh, 4
  system DVMA, 3
  TYPE0 space, 5
  TYPE2 Space, 6
  TYPE3 Space, 6
  video memory, 5
  wait states, 3

P

p_berr signal, 53
p_bufen signal, 32
p_halt signal, 53
p_inta- signal, 32
P2 Bus Control and Address Buffers
  U700 comparator, 147
  U703:01 P2 address buffers, 147
  U704 Control signal buffer, 147, 148
P2 size bits, 60
p2_fpa signal, 28
Page Map RAM, 127
  byte selection, 128
  Page Map Entries (format), 127
  Type-Bit Decode, 127
Page Map RAM control signals
  mmu_gt16, 128
  mmu_we24, 128
PAL equation
  bootcy- signal, 27
  c60 clock, 103
  c60- clock, 103
  c60k clock, 103
  clkinh signal, 29
  ctlspc- signal, 28
  ctxt(2:0) signals, 117
  devspc- signal, 28
  diagcy signal, 169
  fpa_bei- signal, 29
  fpp_berr signal, 32
  fpp_cs signal, 33
  hp_ack0, 85
  hp_ack1, 85

PAL equation, *continued*
hp_eo, 87
ipc[2:0], 82, 87
ipl(2:0), 91
lp_ack01, 80
lp_ack1, 80
lp_eo, 82
ltyp0 signal, 140
m_ben signal, 348
m_parrd signal, 348
m_rw signal, 348
mmu_gt00 signal, 215
mmu_gt08 signal, 215
mmu_gt16 signal, 214
mmu_gt24 signal, 213
mmu_gtseg signal, 217
mmu_io signal, 143
mmu_perr signal, 142
mmu_ram, 144
mmu_val signal, 141, 142
mmu_verr signal, 142
mmu_vme signal, 143
mmu_we00 signal, 217
mmu_we08 signal, 216
mmu_we16 signal, 216
mmu_we24 signal, 216
mmu_weseg signal, 217
mos_a0, 183
mosrden signal, 168
moswren signal, 167
output enable OE, 134
p_berr signal, 53
p_bufen signal, 32
p_halt signal, 53
p_inta- signal, 32
p2_ack signal, 348
p2_fpa signal, 28
p2rden0 signal, 238
p2rden1 signal, 238
pad24 signal, 209
par_as signal, 161
par_err(00) signal, 154
par_err(08) signal, 154
par_err(16) signal, 153
par_err(24) signal, 153
par_irq signal, 161
rasE_00 signal, 338, 340
rasE_08 signal, 338, 340
rasE_16 signal, 338, 340
rasE_24 signal, 337, 340
rasO_00 signal, 339, 341
rasO_08 signal, 339, 341
rasO_16 signal, 339, 341
rasO_24 signal, 338, 340
rd_berr signal, 221, 223
rd_ctxt signal, 221
rd_eeprom signal, 181, 189
rd_eprom signal, 181, 188
rd_ether signal, 207
rd_id signal, 221
rd_int signal, 208
rd_keybdm signal, 182
rd_pad00 signal, 210
rd_pad08 signal, 209
PAL equation, *continued*
rd_pad16 signal, 209
rd_pad24 signal, 157
rd_par signal, 208
rd_serial, 182
rd_sysen signal, 222
rd_tod signal, 183
rd_usren signal, 222
rden0 signal, 238
rden1 signal, 239
rerun signal, 33
s_error signal, 155
sample- signal, 160
ti_d[3:0] signals, 117
tsack, 197
ttlbfen signal, 196
vcopydet signal, 239
vidrd signal, 240
vidwr signal, 240
wr_ctxt signal, 222
wr_diag signal, 223
wr_eeprom signal, 181, 189
wr_ether signal, 207
wr_int signal, 208
wr_keybdm signal, 182
wr_par signal, 208
wr_serial signal, 183
wr_sysen signal, 222
wr_tod signal, 183
wr_usren signal, 223
wren00 signal, 251
wren08 signal, 251
wren16 signal, 251
wren24 signal, 251
wren32 signal, 251
wren40 signal, 252
wren48 signal, 252
wren56 signal, 252
write strobe stwe, 134
x_cas signal, 348
x_we signal, 347
PAL equations
copy16sel signal, 236
lberr signal, 54
mb16sel signal, 235
vmeberr signal, 55
Parity Circuitry, 151
U3112 Parity Data Buffer, 162
U801 Memory Error Register, 161
U802 and U812 Parity Control PALs, 152
U802 Parity Control PAL, 156
U803 Multiplexer, 152
U807:04 Parity Generator/Checkers, 151
U811:08 Parity Address Latch, 151
U813 Byte Select Buffer, 162
power-on circuit, 37
dual comparator LM393 (U200), 37
por- signal, 37
Processor Data Buffers, U105:2, 23

R

Refresh as a Special Case, 316
rerun signal, 33
response synchronizer U206, 41

S

Sample Cycles, 323
  CPU Access of a Busy VMEbus, 324
  CPU Access of VMEbus, Currently Bus Master, 325
  CPU Rerun During VME Access, 325
  Ethernet Cycles, 329
  Relinquishing the VMEbus, 326
  VME Master Cycles, 323
  VME Slave Cycles, 327
Sample Interrupt Cycle, 92
Segment Map RAM control signals
  mmu_gtseg signal, 123
  mmu_weseg signal, 123
Serial Ports A and B, 187
signal aliases, 148
spurious interrupts, 93
Sun-3 Function Code Address Space, 24

T

Time of Day (TOD) Clock, 190
  7170 TOD clock chip, 191
  U1209, 190
TTL Bus, 195
  Ethernet Control Register, 223
  read cycle, 195
  U1400 TTL Bus Sack State Machine, 195
  U1401 TTL Bus Device Decoder, 204
  U1402 MMU Decoder, 210
  U1403 CPU Signal TTL Bus Decoder, 218
  U1404 P2-to-TTL Data Buffer, 225
  U1406 System Enable Register, 224
  U1408 System Enable Register, 224
  U1409 ID PROM, 225
  U1410 Diagnostics Register, 225
  U203 Bus Error Register, 226
  U2905 User DVMA Enable Register, 226
  U2906 User DVMA Enable Register, 226
  U509 Context Register, 226
  write cycle, 195

U

U1000 Mouse and Keyboard SCC, 186
U105:2, 23
  data flow, 23
U106, 24
  input signals, 26
  output signals, 27
  pinout, 26
U107, 24, 30
  input signals, 31
  output signals, 31
  pinout, 30
U1100, 187
U1100 Serial Ports
  Receive Data Path, 188
  Transmit Data Path, 188
U1400 TTL Bus Sack State Machine, 195
  cycle timing, 203
  outputs, 196
  pinout, 195
  read cycle, 203
  state diagram, 198
  write cycle, 203
U1401 TTL Bus Device Decoder, 204
  inputs, 205
  outputs, 206
  pinout, 205
U1402 MMU Decoder, 210
  inputs, 212
  outputs, 213
  pinout, 212
U1403 CPU Signal TTL Bus Decoder, 218
  inputs, 219
  outputs, 220
  pinout, 219
U1404 P2-to-TTL Data Buffer, 225
  Data Flow, 225
U1405 Ethernet Control Write Buffer, 223
U1406 and U1408 System Enable Register, 224
U1406 System Enable Write Register, 224
U1408 System Enable Read Register, 225
U1409 ID PROM, 225
U1500 Buffer, 253
U1501 Byte Decode PAL, 250
  outputs, 251
  pinout, 250
U1502 Video Control Decoder, 236
  inputs, 237
  outputs, 237
  pinout, 237
U1503 P2 Interface State Machine, 241
  inputs, 241
  outputs, 242
  pinout, 241
U1504 Video Select Decoder
  copy-mode, 234
  inputs, 235
  outputs, 235
  pinout, 235
U1505 DIP, 253
U1605/07, 241
U1608-U1603 Video Controller, 253
U1700-01 Video RAS/CAS Latches, 254
U201 Reset PAL, 45
  cslow signal, 46
  input signals, 46
  output signals, 47
  p_reset signal, 47
  pinout, 46
  what it does, 46
U202, 51
  input signals, 51
  output signals, 53
  pinout, 52
U203, 51
U203 Bus Error Register, 56, 226
U204 DSACK PAL, 59

U204 DSACK PAL, *continued*
Bus Transfer Size, 60
input signals, 60
Offset Bits, 60
output signals, 62
pinout, 59
U205 User Reset Switch, 45
what it does, 45
U2206, 258
U2207, 258
U2207 Baud Rate Clock, 187
U2303:01, 257
U2305, 256
U2306, 257
U2308 ECL oscillator, 256
U2312, 256
U2409 DVMA Controller, 315
Sample DVMA Cycle, 315
U2410 DVMA Strobe PAL, 317
inputs, 319
outputs, 319
pinout, 318
U2701 VME Select and Freeze PAL, 285
inputs, 287
outputs, 287
pinout, 286
state diagram, 288
VME Long Timeouts, 290
VME Short Timeouts, 290
U2704 VME Arbiter and Requester
input signals, 269
J2700, 269
J2701, 269
output signals, 269
pinout, 271
signals, 269
state machine, 272
U2901-2 Slave Address Latches, 299
U2904 Slave Request PAL, 302
inputs, 305
outputs, 305
pinout, 304
state machine, 306
U2905 User DVMA Enable Register, 226
U2905-6 User DVMA Enable, 302
U2906 User DVMA Enable Register, 226
U2907 VME Slave Address Decoder, 299
inputs, 301
outputs, 301
pinout, 300
U2910:09 Slave Address Multiplexers, 302
U2911-13 Slave Address Latches, 299
U300 Interrupt circuitry, 67
U3006:00 VME Data Buffers, 311
16-Bit Operation, 311
32-Bit Operation, 312
CPU Cycles, 312
DVMA Cycles, 312
U301 Interrupt circuitry, 67
U301:0 Interrupt Enable Registers, 68
U302 Lower-Priority Encoder, 78
U302 Lower-Priority Encoder, *continued*
input signals, 79
output signals, 80
pinout, 79
U302-U304 Interrupt circuitry, 73
U303 Higher-Priority Encoder, 84
input signals, 84
output signals, 85
pinout, 84
U304 Second-Level Interrupt Priority Encoder, 89
input signals, 90
output signals, 90
pinout, 89
U305, 94
U305 Interrupt circuitry, 73
U3100, 335
U3100 and U3102 RAS Decode PALs, 335
inputs, 336
outputs, 336
pinout, 335
U3100 Output Signals, 337
U3102 Output Signals, 339
U3100 Output Signals, 337
U3100 RAS Decode PAL, 335
U3102, 335
U3102 Output Signals, 339
U3102 RAS Decode PAL, 335
U3104 CAS Decode PAL, 345
inputs, 346
outputs, 346
pinout, 345
U3105 Control Buffer, 351
U3110:07 Row and Column Address Multiplexers, 355
U3115 Control Buffer, 351
U400 Clock PAL
input signals, 102
output signals, 103
pinout, 101
U401 Flip-Flops, 104
U401 PAL, 107
U402 Flip-Flops, 104
U405 and U2207 Baud Rate Clock, 187
U407 ATE pulldowns, 97
U408 PAL
pinout, 107
U508:00 Segment Map, 121
RAM control signals, 123
Read Cycle, 122
U507:00 Segment Map RAM, 121
U508 buffer, 123
Write Cycle, 123
U509 context register, 115, 226
input signals, 116
U509 Context Register
inputs, 227
U509 context register
output signals, 117
U509 Context Register
outputs, 227
U509 context register
pinout, 116, 227

U509 Context Registers, 302
U611 Statistics Control PAL, 133
  input signals, 133
  output signals, 134
U612 MMU Validation and Decode PAL, 139
  input signals, 139
  output signals, 140
U700 comparator, 147
U704 Control Signal Buffer, signals buffered are
  b_c62, 148
  mmu_a(24), 148
  p_as, 148
  p_fc(2), 148
  p_rw, 148
  p_siz(1:0), 148
  refr, 148
U801 Memory Error Register, 161
U802 Parity Control PAL, 156
  input signals, 156
  output signals, 157
  pinout, 159
U803 Multiplexer, 152
U807:04 Parity Generator/Checkers, 151
U811:08 Parity Address Latch, 151
  control signals, 151
U812 Parity Check PAL, 153
  input signals, 153
  output signals, 153
U813 Byte Select Buffer, 162
U900 MOS Enables PAL, 166
  outputs, 166
  pinout, 167
U901 MOS SACK State Machine, 169
  pinout, 170
U902 MOS Write and U903 MOS Read Buffers, 184
U902 MOS Write Buffer, 184
U903 MOS Read Buffer, 184
U904 MOS Read/Write Strobe Decoder, 179
  input signals, 180
  output signals, 180
  pinout, 179
user space, 25

V

Vertical State Machine
  U2206, 258
  U2207, 258
U1608-U1603 Video Controller
  Circuitry", 253
Video Circuitry, 233
  ECL Circuitry, 256
  ECL Clock, 256
  Frame Buffer RAM, 255
  Horizontal State Machine, 257
  U1500 Buffer, 253
  U1501 Byte Decode PAL, 250
  U1502 Video Control Decoder, 236
  U1503 P2 Interface State Machine, 241
  U1504 Video Select Decoder, 234
  U1505 DIP, 253
  U1605/07, 241
Video Circuitry, *continued*
  U1607, 233
  U1700-01 Video RAS/CAS Latches, 254
  VARB and Video Side State Machines, 242
  Vertical State Machine, 257
  Video Cycle Timing, 233
  Video Read, 243
  Video Write, 247
  Video Write Timing Diagrams, 249
video memory
  address, 5
virtual address, 25
virtual address space, 25
VME Arbiter and Requester, 269
  U2704 Arbiter state machine, 269
VME Compliance
  options, 11
  Performance Parameters, 11
VME Master Controller PAL U2806, 291
  inputs, 293
  outputs, 293
  pinout, 292
  state machine, 293
VME Master Cycles
  CPU Access of Idle VMEbus, 323
VME Master Interface
  U2701 VME Select and Freeze PAL, 285
VME Slave Cycles, 327
  Lock Mode Cycles, 328
  VME Device Accesses P2 Bus, 327
VME Slave Interface, 299
  U2901-2 Slave Address Latches, 299
  U2904 Slave Request PAL, 302
  U2905-6 User DVMA Enable, 302
  U2907 VME Slave Address Decoder, 299
  U2910:09 Slave Address Multiplexers, 302
  U2911-13 Slave Address Latches, 299
  U509 Context Registers, 302
vmeberr signal, 55
VMEbus
  2060 VME Implementation, 263
  performance, 263
  U3006:00 VME Data Buffers, 311
  VME Master Controller PAL U2806, 291
  VME Master Interface, 285
  VME Slave Interface, 299

W

wait states, 3

2060 CPU SPECIFICATION

1.0 Introduction

This document provides implementation information for the 2060 CPU and Expansion boards. It provides specific information concerning SUN-3 architecture and VME compliancy, general implementation information, and mechanical specifications. Detailed hardware descriptions (will) be found in other documents.

2.0 SUN-3 architecture compliancy

This section is divided into five parts. The first part describes the CPU and DVMA devices. The next two parts describe the Control Space (68020 extensions and MMU) and all devices which must be accessed through the MMU. The last two sections describe interrupts, resets, and timeouts. References to the SUN-3 architecture manual are in brackets [].

2.1 CPU/DVMA devices: Everything in front of the MMU

2.1.1 CPU

The processor will be a 68020 running at 16.67 MHz. All bus cycles will incur a minimum of 1.5 wait states. S4 will be stretched by 30 nsec to cause the half wait state.

There will be an optional 68881 Floating Point Processor. The FPP can be run on an independent clock.

All CPU space cycles will be implemented as in [3.1]. Disabled (System Enable register D6=0) FPP coprocessor cycles will be terminated with an immediate bus error. All other coprocessor addresses and accesses to an enabled but uninstalled FPP will result in a Timeout bus error. Interrupt Acknowledge cycles and installed and enabled FPP cycles will terminate normally with DSACK or be aborted with a synchronous bus error.

2.1.2 System DVMA

The two system DVMA devices are the Ethernet Interface (Intel 82586) [5.11] and the VME Slave System DVMA [5.13.1]. Both use supervisor data function codes and are implemented as in the SUN-3 architecture manual.

The Ethernet Interface has one feature other DVMA devices do not implement. Fifo operation of the 82586 requires that the Ethernet Interface be able to retain bus mastership. Therefore it can issue a HOLD signal along with bus request.

2.1.3 VME Slave User DVMA

This will be implemented as in the SUN-3 architecture manual [5.13.2]. User DVMA is performed in user data function codes. There will be no response to an access to a disabled context.

2.1.4 Refresh

The refresh timer will request the bus via the DVMA controller like

other DVMA devices. Once the bus has been obtained for a refresh operation, the controller will not execute a DVMA cycle but instead execute a refresh cycle. A REFResh strobe will be issued instead of AS- so that the refresh cycle will not conflict with any other address space cycle.

2.1.5 DVMA Controller

DVMA/CPU device priority is as follows:
1) Refresh- nothing can stop a refresh cycle
2) Ethernet- can issue bus hold to lock out 3 and 4
3) VME System/User DMA- dynamic bus hold feature to lock out 4
4) 68020/68881

2.2 Control space: Everything in FC3

2.2.1 Control Space Devices

The following Control Space devices will be implemented [4.1]: All devices are byte read and write except for the bus error register which is byte read only. The ID PROM, Page Map, and the Segment Map are implemented as an array of bytes. This will allow word and longword accesses via the 68020 dynamic bus sizing capability.

ADDRESS		DEVICE
[0x00000000]	+ Virtual	ID PROM
[0x10000000]	+ Virtual	Page Map
[0x20000000]	+ Virtual	Segment Map
[0x30000000]		Context Register
[0x40000000]		System Enable Register
[0x50000000]		User Enable Register
[0x60000000]		Bus Error Register
[0x70000000]		Diagnostic Register
[0x80000000]	to	Non-responding addresses which
[0xE0000000]		will cause a timeout bus error
[0xF0000000]		MMU bypass access to Serial Port for diagnostics

2.2.2 Memory Management Unit

The MMU will be implemented as in the SUN-3 Architecture Manual [4.3] with the following exception: the cache bits will read back as zero since they are not implemented.

2.3 Device space: Everything after the MMU

2.3.1 ECC Memory: not implemented

2.3.2 Memory Space: TYPE0 space

2.3.2.1 Parity Main Memory

Main memory will be implemented as in section [5.1]. A positive acknowledge scheme will be used so that non-existing memory locations will result in a timeout bus error.

256K X 1, 120 nsec DRAMs will be used to implement the parity memory. Accesses to the memory will incur 1.5 wait states on reads and writes. There will be a minimum of two megabytes of memory on the CPU board and additional memory on the Expansion boards.

2.3.2.2 Video Memory

Video memory is a 128K byte block of memory starting at location 0xFF000000. Copy Mode, if enabled, will cause any write operation to a 128K byte block of memory starting at location 0x00100000 to also be written into the video memory.

The Video display format will be two types. The first is a 1152 X 900 format and the second is a 1024 X 1024 format. The Vertical rate will be 67 Hz and the pixel rate will be 10 nsec per pixel.

The outputs to the video monitor will be as follows:

1) Serial Video- differential ECL
2) Horizontal Sync- positive TTL pulse, sync on rising edge
3) Vertical Sync- positive TTL pulse, sync on rising edge

2.3.3 I/O Devices: TYPE1 space

The following devices will be implemented in TYPE1, 21 bit address space as per the Sun 3 Architecture Manual [5.2, 5.4 to 5.11]:

ADDRESS		DEVICE
[0x00000000]		Keybcard/Mouse interface
[0x00020000]		Serial I/O ports
[0x00040000]		EEPROM
[0x00060000]		Time of Day Clock
[0x00080000]		Parity Error registers
[0x000A0000]		Interrupt register
[0x000C0000]		Ethernet Control register
[0x000E0000]		Non-responding address which will cause a timeout bus error
[0x00100000]		EPROM
[0x00120000]	TO	Non-responding addresses which
[0x001A0000]		will cause a timeout bus error
[0x001C0000]		Encryption processor
[0x001E0000]		Non-responding addresses which will cause a timeout bus error

The Parity Error registers, EPROM, and the EEPROM appear as an array n processor
of bytes. This will allow usage of the 68020 dynamic bus sizing capability in accessing these devices.

The Encryption processor is an option. To comply with U.S. Customs law, both the 9518 DCP and support PAL will reside in sockets. The absence of the PAL will cause a time out error.

The Time of Day clock will provide the level 7 non-maskable interrupt. The same interrupt can also provide a level 5 interrupt [5.4: Int. reg].

The EEPROM has a 10 msec per byte write overhead. It will be a software responsibility not write to the EEPROM faster than 10 msec/byte.

2.3.4 VME Master: TYPE2:3 space

CPU accesses to the VME bus will be through TYPE2 space for 16 bit data transfers and TYPE3 space for 32 bit data transfers. The 32 bit address will be decoded to supply the VME Address Modifier bits and define the VME address size.

TYPE2 32-bit Address	VME bus with 16-bit data	AM5:3 (H) Address Modifiers
[0x00000000]	VME 32-bit address space	(L L H)
[0xFF000000]	VME 24-bit address space	(H H H)
[0xFFFF0000]	VME 16-bit address space	(H L H) I/O access only

TYPE3 32-bit Address	VME bus with 32-bit data	AM5:3 (H) Address Modifiers
[0x00000000]	VME 32-bit address space	(L L H)
[0xFF000000]	VME 24-bit address space	(H H H)
[0xFFFF0000]	VME 16-bit address space	(H L H) I/O access only

2.4 Interrupts

2.4.1 On-Board Interrupts

On-board interrupts will be autovectored on all levels except for level 6 where the 8530's will provide a vector.

LEVEL	DEVICES
7	NMI- Real Time Clock and Parity Error
6	All Serial Controllers (8530A's)
5	Real Time Clock
4	Video vertical interrupt
3	Ethernet, System enable register 3
2	System enable register 2
1	System enable register 1

2.4.2 VME Vectored Interrupts

VME interrupts will be vectored and lower priority than on-board interrupts.

2.5 CPU Resets and Timeout

1) Power On Reset: see [6.0].

2) Watchdog Reset: see [6.0]. A user accessible panic button will also cause a watchdog reset.

3) CPU Reset: see [6.0]. In addition, access to the VME bus will be inhibited for the 200 msec min. SYSRESET- period.

4) CPU Board Timeout: Minimum of one refresh period, maximum of two. All non-responding addresses and devices will result in a timeout bus error.

3.0 VME Compliancy

This section concerns VME compliancy and performance.

3.1 Options

1) Multiple Arbiters: A jumper will be provided so that if installed will give arbitration control to another VME device.

2) Arbiter Option: ONE, Only BR3- will be monitored.

3) Requester Option: ROR, Release on request

4) Timeouts: Two VME Master timeouts are provided. The first is a "retry" period of 3.06 usec (60 nsec clock/ 3clks + 240 nsec *12) at which time the VME interface "freezes" and other DVMA devices (Refresh, Ethernet) can obtain the local bus. After 128 retries, a timeout error will occur. This will provide a timeout when the CPU board is master. No timeout will be provided for VME Slave or User mode since it is the responsibility of each master to provide it's own timeout.

5) Backoff Mechanism: If the CPU starts an access to the VME at the same time a VME devices accesses the P2, the CPU cycle will be re-run.

6) Non-implemented features: Since multiprocessing will not be allowed on our systems, READ-MODIFY-WRITE will not be implemented.
The ACFAIL- timing during power down will not meet spec nor will it even be close. Power up is similar. We need to use an ACFAIL signal from the power supply, use massive power supply caps, or have battery backup if we want to implement power down timing. Power up timing has a chance since we need to implement the 200 msec VME reset lockout of the CPU anyway.

3.2 Performance Parameters

The following performance parameters assume a 60 nsec clock and 1.5 wait states on read and write cycles.

1) CPU to VME latency (assume ideal VME device)

not currently bus master	600-660 nsec
currently VME bus master	420-480 nsec

2) CPU to VME bandwidth (assume ideal VME device)

burst rate	8.3-9.5 MBytes/sec
	480-420 nsec/longword

3) VME to P2 latency (not currently P2 bus master, assume idle P2 bus)

AS to DTACK	570-630 nsec

4) VME to P2 bandwidth (assume P2 bus locked)

bandwidth	6.3-8.9 MBytes/sec
	635-450 nsec/longword

5) VME to VME transfer

time to acquire VME bus	70-155 nsec
bandwidth	limited by VME spec and VME devices

4.0 Block diagram discussions

4.1 High level of everything

Figure 1 shows a simplified block diagram of the SUN 2060 system. Devices from section 2.1 are on the left side. They supply a virtual address to the MMU and arbitrate for control via the DVMA controller. Devices from section 2.2 are located in the center. These are the CPU extensions and are accessed in FC3 space. The MMU translates the virtual address into a physical address that is used by the devices described in section 2.3. This Device Space is divided into four types, type0 for main and video memory, type1 for I/O and Control devices, and type2:3 for the VME Master interface.

4.2 Data paths

Figure 2 provides details about data bus connections. There are two bus sizes, a 32 bit and an 8 bit. The 32 bit bus provides a high-bandwidth path between the CPU and DVMA devices and main memory. An 8 bit bus size is used to reduce significantly board routing problems. This works well since most of the Control and Device Space devices are 8 bits. The Parity Latch and Page Map interface bandwidth will be less than possible due to the 8 bit bus restriction but accesses to these devices will be infrequent and the loss of bandwidth not noticeable. The dynamic bus sizing capability of the 68020 is used so that longword moves can be done to these two devices.

There are two 8 bit busses to segregate the MOS and TTL devices. This is due to the different data bus interfacing capabilities of the two technologies. MOS devices have weak bus drivers and are sensitive to undershoot while TTL devices have the opposite characteristics. The separation of the two technologies will improve system reliability.

5.0 Mechanical Specifications

5.1 Board Form Factor

The CPU and Expansion boards will conform to the triple height Eurocard specification. This will allow either board to plug into a 50 or 160 chassis.

Height 366.67 mm
Width 400.00 mm

5.2 Connectors

There are eight connectors on the CPU board. Three are the P1:3, 96 pin, VME bus connectors. The other five connectors are the 9 pin video output, 15 pin Ethernet, two 25 pin serial ports, and a 15 pin long distance keyboard and mouse connector.

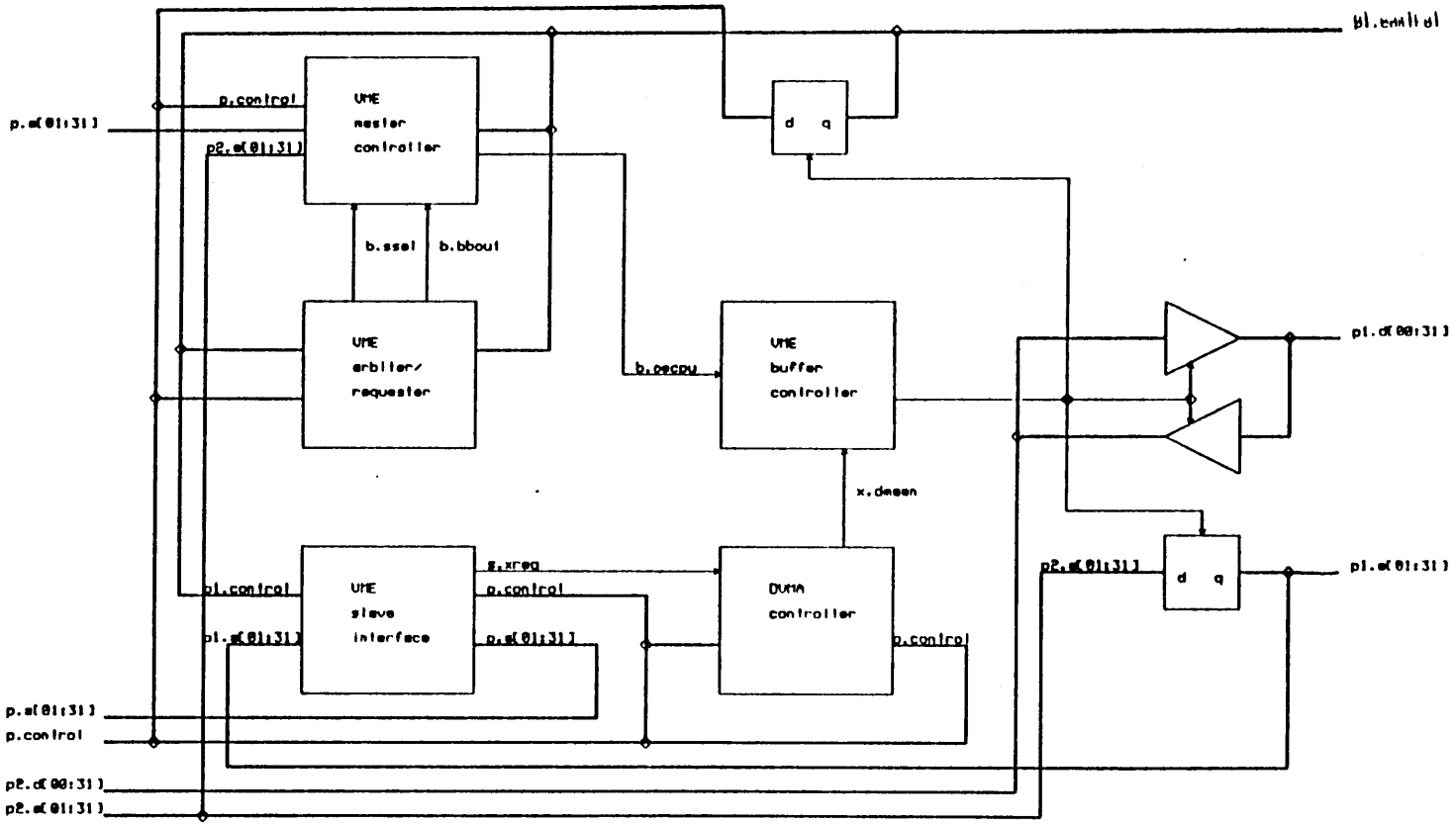
The basic Expansion board has the three P1:3 VME bus connectors. Additional connectors will depend on what other functions besides memory will be on the board.

5.3 Switches

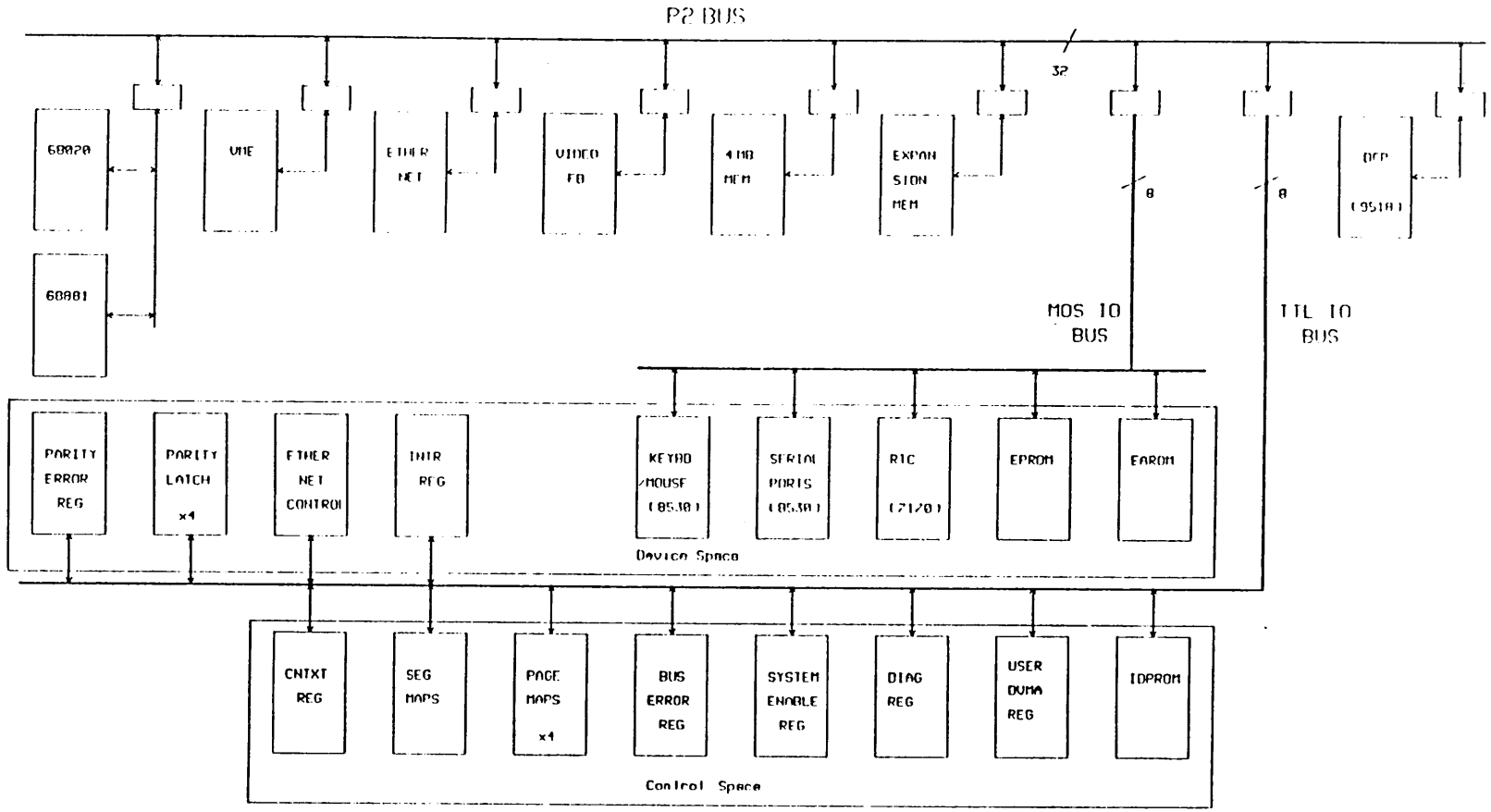
There are two user accessible switches. One is the diagnostic switch which is used to enter and exit diagnostic mode. The other is the user reset switch which will cause a watchdog reset.

5.4 Backplanes

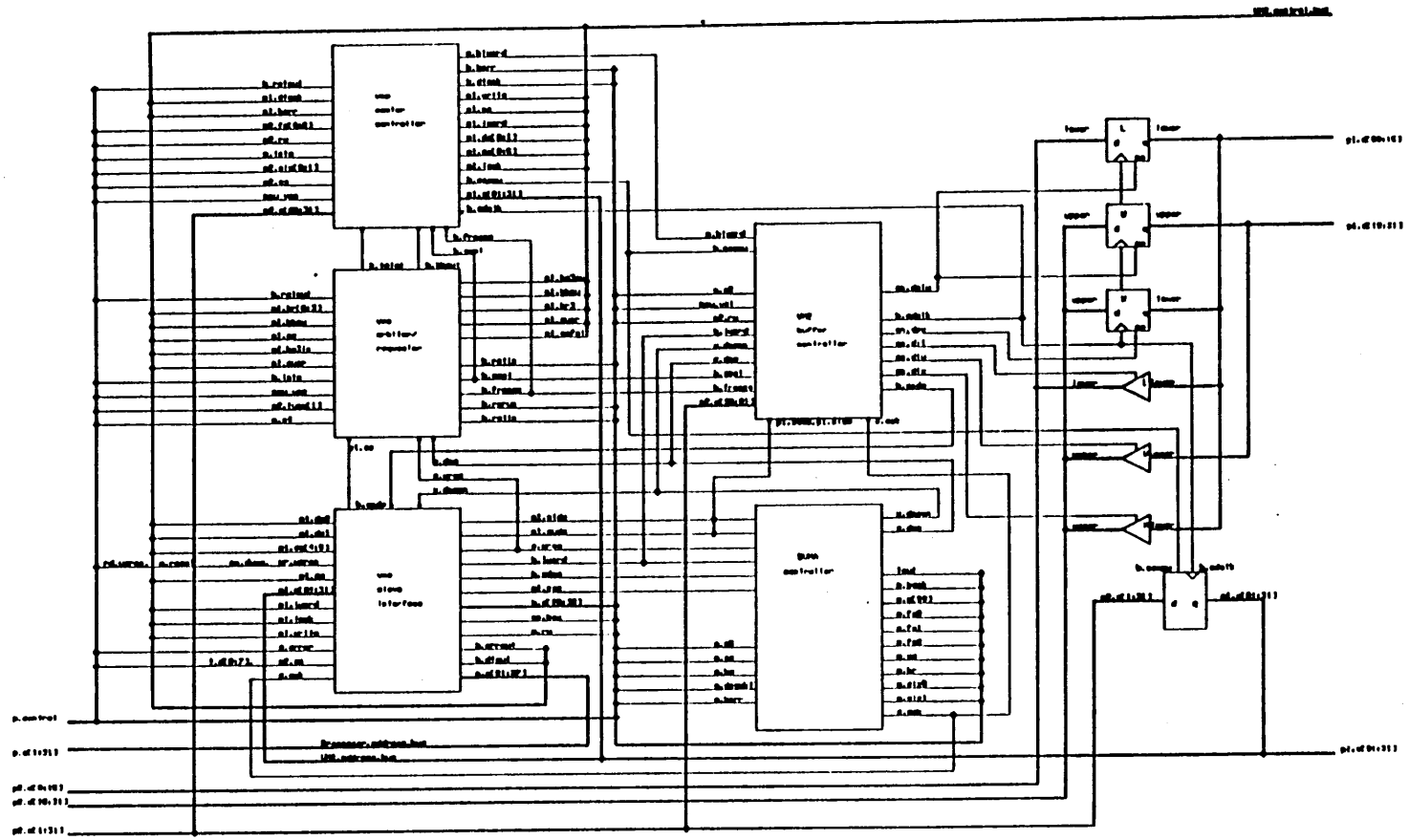
2050, 2060, Sirius compatibility will be examined.



Simplified_VME_Interface_Block_Diagram



2060 DATA BUSSING JOE MURPHY 1/31/85 REV 1.4



Detailed_VME_Interface_Block_Diagram

2060 P2 BUS SIGNAL DEFINITION

Joe Murphy

The following are the signals that I am calling collectively the "P2 Bus" for the 2060. These signals will go off board to the P2/P3 connector of the VME connectors, as well as interfacing to the memory on the 2060 cpu board. The principle purpose of the P2 bus is to provide a high bandwidth path to main memory, and as result it is very heavily optimized for dynamic memory timing. I have buffered most of the signals from the processor, and from the page map rams, to prevent problems associated with mos output stages driving long low impedence lines. For all processor values assume no derating until we know more accurately what the loading will be (only exception to this is p_as which *must* be derated by 3 ns).

SIGNAL DEFINITIONS

p2_a[31:00]	This is a buffered version of physical address. p2_a[31:13] is the output of the MMU, and p2_a[12:00] is the output of the bus master (68020 or VME). For p2_a[31:13] timing is (mmu_a + f244) where (mmu_a == 100) worst case (p_a + seg_ram + page_ram). For p2_a[12:00] timing is (p_a + f244).
p2_d[31:00]	Bidirectional data bus. For processor writes is (p_dw + als245). Writes to memory are early writes.
p2_siz[1:0]	Buffered version of the bus master's size bits. They are used in tandem with p2_a[01:00], to determine which bytes need to be selected for a given cycle. Timing is (p_siz + f244).
p2_rw	Buffered version of the bus master's r/w signal. Timing is (p_rw + f244).
m_ack-	Memory acknowledge. An addressed device on the P2 bus must respond with a positive acknowledge to complete the cycle. The bus master will add wait states until this happens, or until the cycle time-outs via the time-out bus error sequence. There is no minimum spec for how soon m_ack- can be asserted after the start of the cycle. Responding devices can assert m_ack- as soon as p2_cas- is asserted. Maximum spec is determined by system level timeouts. Data can become valid after m_ack- is asserted, as long as setup time for all bus masters is met.

p2_devsp- Device space cycle. This signal is used to qualify the generation of ras for memory cycles. It is the alternative to using the function codes directly. We do not generate ras for cycles that are not in device space. We do this to conserve power, and to decouple minimum cycle time for devices that can respond faster than main memory (such as the 68881). This signal is asserted when p_fc[2:0] = (1,2,5,6) for non boot state cycles, and p_fc[2:0] = (1,2,5) for boot state cycles. Since the physical address spaces qualify with p2_ram-, and p2_ram- is only generated if we are doing a devicespace access, this signal is only neccesary to qualify ras - and then only if we have cycles that are shorter than the normal cycle length. Timing is (p_fc + bpal).

p2_ras Row Address Stobe. Timing strobe to determine when to assert ras for main memory. Must be qualified with p2_devsp- for a device space cycle, p2_a[12:11] for ras bank decoding (to save power), size[1:0] and p2_a[01:00] for byte decode, and p2_endras- not asserted (termination of ras). This signal is asserted on (s2 + f74), and deasserted on (cycle+f74). Equivalent to CS2.

p2_mux Timing strobe to determine when to switch the address presented to the rams from row address to column address. When low we are selecting the row address, and when high the column address. Is equivalent to CS3-.

p2_ram- Signal to qualify p2 cycles. Indicates that the cycle is legitimate memory cycle because: we doing a device space cycle, the page is valid, the protection checks are ok, and the type bits == TYPE0. Must be qualified by p2_cas- if one wants an edge to indicate when this signal is stable.

p2_cas- Column Address Strobe. Timing strobe to determine when to assert cas for main memory. Must be qualified with p2_a[31:22] for "board" select, and p2_ram- to designate as a memory cycle. Equivalent to CS4-

p2_par[24,16,08,00] Bidirectional parity data bus. Writes to memory for parity are late writes.

p2_parwrt- Parity write strobe. Used to strobe in parity information on writes to memory. This signal is needed to do late writes for parity data. Parity write data will not be valid till after cas is asserted at the rams. Equivalent to CS5-.

p2_endras- End ras. Timing strobe to determine when to deassert ras. Is asserted at cs6, and deasserted at cycle+s(1). Equivalent to CS6-.

p2_refr-	Refresh all the ram chips. p_as should not be asserted on refresh cycles, to keep from triggering all the other timing chains. Refresh to memory should not look like normal memory cycles. Obtain the bus like a full blown bus master would do, but then just gate the refresh address onto at least p2_a[11:02] (10 bits assumes that the largest rams for product life time will be 1 Mbit), and one clock later assert p2_refr- for 3 clocks (Tras == 120ns -> 2 clocks + skew_slop).
c2_io-	Type 1 access
p2_c60m	System clock.

2060 BOM

	DESCRIPTION	QTY
	LM393	1
	74LS273	4
3.	74LS652 or ALS	12
	74F244	27
5.	74ALS245	13
	256K DRAM	144
7.	16R4	3
	16R4A	2
	16R4B	3
	16R6	1
	16R6A	1
12.	16R8B	1
	16L8	3
14.	16L8A	4
	16L8 B	8
16.	20L8A	3
17	20L10	4
	20R8A	2
19.	22V10	1
	74F158	1
	74F374	12
	74F280	4
23.	74LS244	3
	2168-35	7
	1403-35	8
27.	74LS158	1
28	25LS2518	1
29.	74F74A	9
	68020	1
31.	68881	1
	5516A	1
33.	27256K or 512K	1
34	7170	1
	74LS245	2
36.	9517	1
	74LS374	4
38.	PSK 74S288	2
39.		2
	74LS573	4
41.		1

	DESCRIPTION	QTY
41.	74F00	1
42.	74LS163	1
43.	74ALS244	2
44.	82586	1
45.	74ALS374	7
46.	74ALS04	1
47.	74F521	2
48.	74F32	1
49.	29821	10
50.	74F373	2
51.	74ALS273	1
52.	74ALS257A	2
53.	74F151	1
54.	74F08	1
55.	74F109	1
56.	74LS393	5
57.	74F534	8
58.	4416	16
59.	74LS74A	1
60.	74LS112A	1
61.	10H105	1
62.	10H124	2
63.	10H125	1
64.	10H131	1
65.	10H136	1
66.	10H141	2
67.	26LS29	2
68	26LS30	1
69.	26LS32	4
70.	8530	2
71.	74LS04	2
72.	74F157	3
73.	33Ω 16pin DIP	18
74.	33.33MHz OSC	1
75.	19.6608MHz OSC	1
76.	? MHz OSC	1
77.	18pf	1
78.	30pf VAR	1
79.	BATT, LITHIUM 3V	1
80.	BATT. HOLDER	1

2060 BOM CONT'D

	DESCRIPTION	QTY
	37268 Hz CRYSTAL	1
	16 MHz OSC.	1
83.	SWITCH	1
1.	QUAD LED	2
85.	33pf	2
	0.022μF	1
87.	100pf	2
	100MHz OSC	1
	2 pin header	IIII
90.	4 pin header	III
	8 pin header	II
92.	16 pin header	I
	Diode IN4148	
94.	LM 395 1.2v REGUL	1
	15KΩ 1%	1
96.	100 KΩ 1%	1
97	51.1 KΩ 1%	1
	23.7 KΩ 1%	1
99.	10 KΩ 1%	1
	1KΩ 5%	2
101	1MΩ 5%	2
	33μF	1
105.	10KΩ 10pin SIP	3
	2.2KΩ	6
	47pf	8
106.	50pf	2
	15 pin D conn	2
109	100Ω 5%	2
	9 pin D conn	1
112.	Diode IN4448	4
	20MHz crystal	1
	120Ω 5%	2
113	390Ω 5%	2
	51Ω 10pin SIP	1
115.	75Ω 5%	1
	120Ω 5%	1
117.	1μF	2
	LM 301T	1
	1KΩ 8pin DIP	1
120.	4.7KΩ 10pin SIP	2

	DESCRIPTION	QTY
122.	470Ω 16pin DIP	1

2062 SECOND SOURCE LISTING

* SINGLE SOURCED

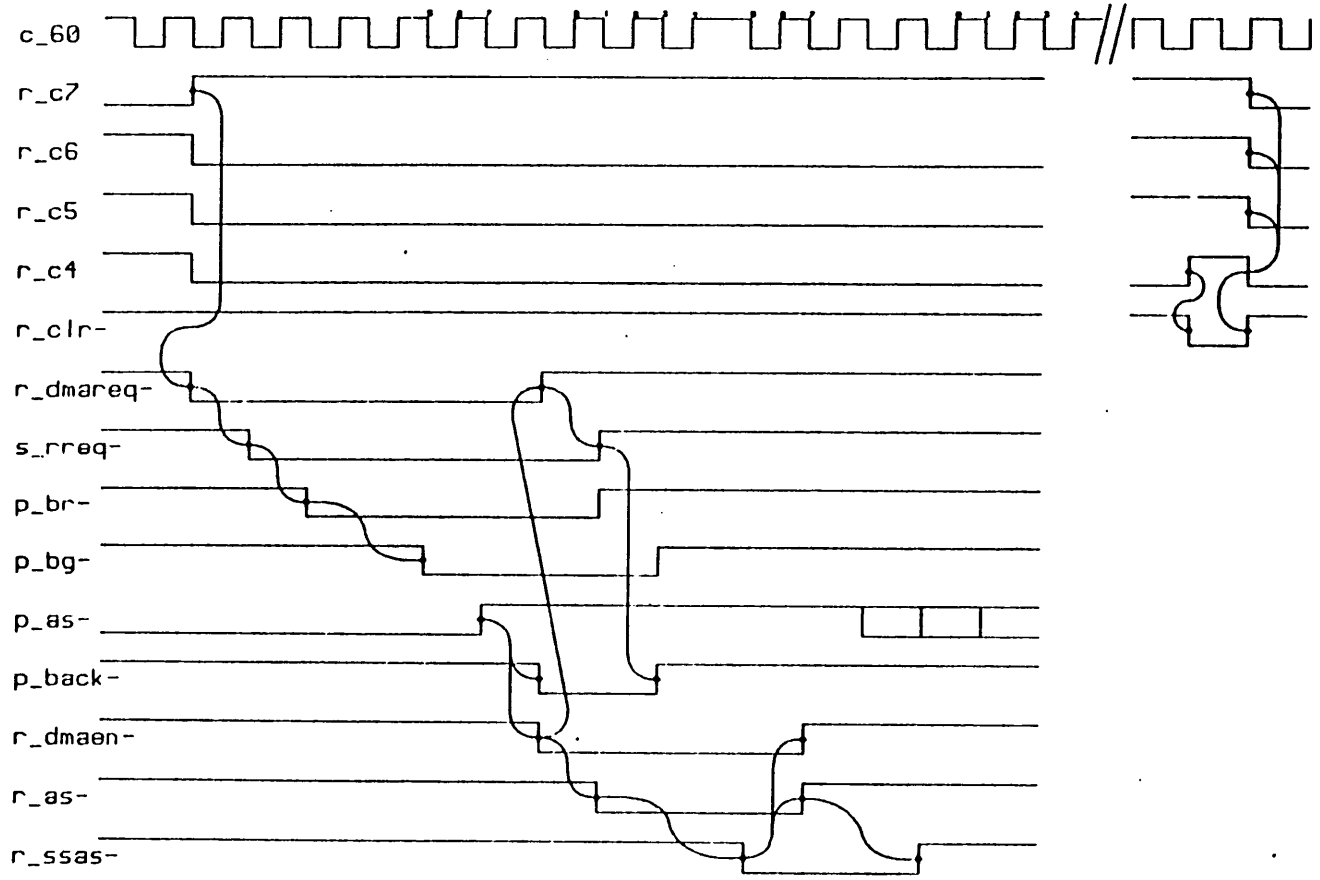
	Description	Manufacturer	Manufacturer
1.	LM393	National	TI
2.	74LS273	TI	Signetics
3.	74LS652 or ALS	TI	Motorola
4.	74F244	Fairchild	Signetics
5.	74ALS245	TI	Motorola
6.	74F241	Fairchild	Signetics
7.	74F158	Fairchild	Signetics
8.	74F374	Fairchild	Signetics
9.	74F280	Fairchild	Signetics
10.	74LS244	TI	National
11.	74LS158	TI	National
12.	74F74A	Fairchild	Signetics
13.	74LS245	TI	National
14.	74LS374	TI	National
15.	74LS590	TI	
16.	74F20	Fairchild	Signetics
17.	74F00	Fairchild	Signetics
18.	74LS162	TI	National
19.	74ALS244	TI	Motorola
20.	74ALS374	TI	Motorola
21.	74ALS04	TI	Motorola
22.	74F04	Fairchild	Signetics
23.	74ALS641-1	TI	Motorola
24.	74F521	Fairchild	Signetics
25.	74F32	Fairchild	Signetics
26.	74F373	Fairchild	Signetics
27.	74ALS273	TI	Motorola
28.	74ALS257A	TI	Motorola
29.	74F151	Fairchild	Signetics
30.	74F08	Fairchild	Signetics
31.	74F109	Fairchild	Signetics
32.	74LS393	TI	National
33.	74F534	Fairchild	Signetics
34.	74LS74A	TI	Motorola
35.	16R4A	AMD	MMI
36.	16R4	AMD	MMI
37.	16L8B	TI	National
38.	16L8A	AMD	MMI

	Description	Manufacturer	Manufacturer
39.	16R3B	TI	National
40.	22V10	AMD	
	20L10	AMD	MMI
42.	16L8	AMD	MMI
43.	16R6	AMD	MMI
44.	20R8A	AMD	MMI
45.	16R4B	TI	National
46.	20L8A	AMD	MMI
47.	16L8A	AMD	MMI
48.	16R4A	AMD	MMI
49.	25S2518	AMD	
50.	PSX8 74S288	National	63S081 MMI
51.	PROM 512X4	AMD	Signetics
52.	256K DRAM	Hitachi	Toshiba & Fujitsu
53.	2168-35 RAM	AMD	
54.	1403 RAM	INMOS	
55.	4416 RAM	TI	
56.	68320	Motorola	
57.	68281	Motorola	
	27256K EPROM 32K	AMD	Intel
59.	5516A EEPROM	SEEQ	
60.	7170	Intersil	
61.	9513	AMD	Zilog
62.	82586	Intel	
63.	29821	AMD	
64.	10H105	Motorola	MMI
65.	10H124	Motorola	MMI
66.	10H125	Motorola	MMI
67.	10H131	Motorola	MMI
68.	10H134	Motorola	MMI
69.	10H141	Motorola	MMI
70.	8530A	AMD	Zilog
71.	26LS32	AMD	Motorola
72.	26LS27	AMD	Motorola
73.	26LS30	AMD	Motorola
74.	74LS114A	Motorola	Mitsubishi
75.			

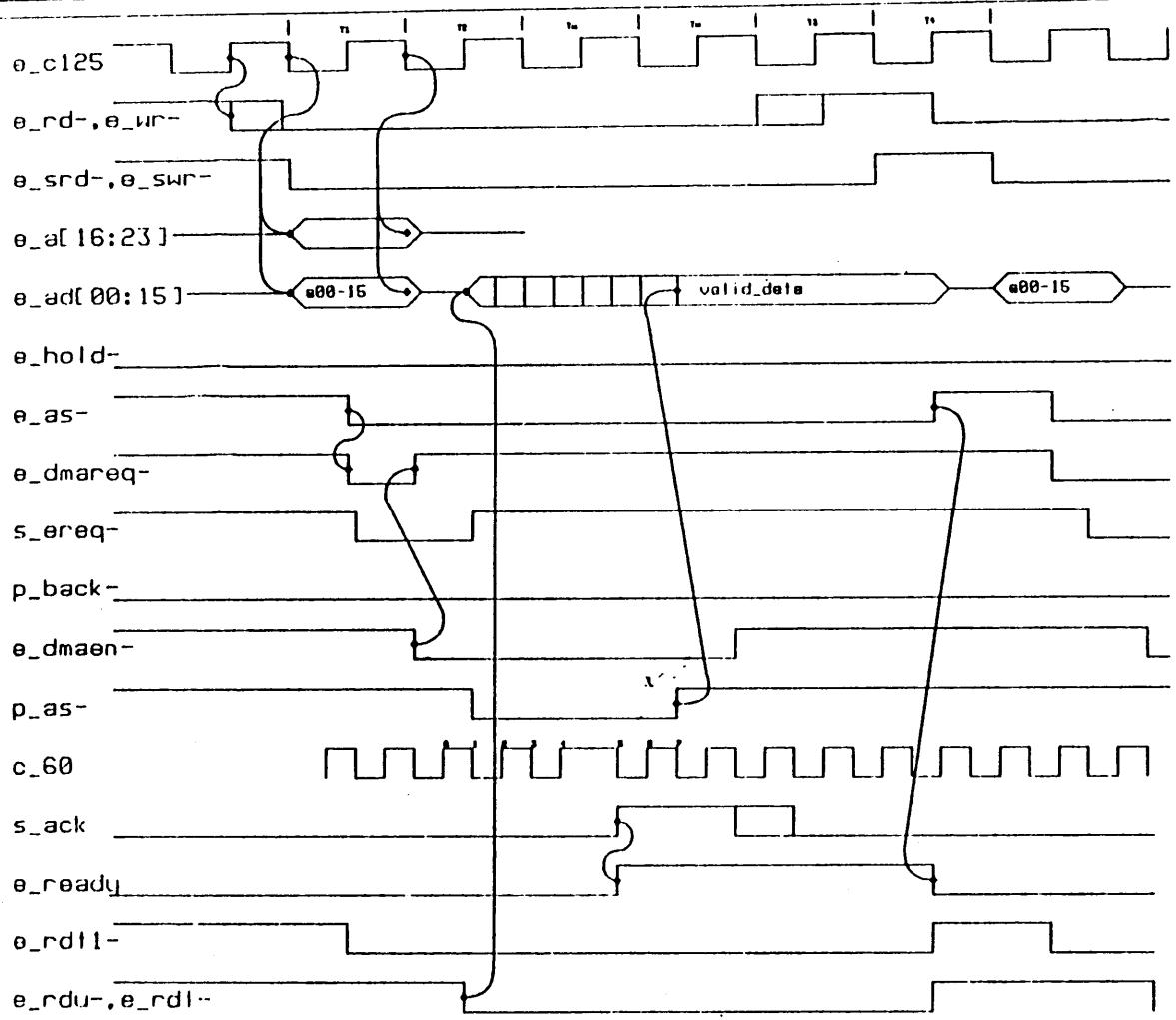
```
/*
        Header file for the various decode pals of the 2060
*/

/*
        Virtual Address Spaces
         Leave as "fc".  Can be either p2_fc or p_fc as appropriate.
*/
#define FC_0            /fc2*/fc1*/fc0          /* FC0 (illegal) */
#define USR_DATA        /fc2*/fc1* fc0          /* FC1 */
#define USR_PRGM        /fc2* fc1*/fc0          /* FC2 */
#define CNTLSPACE       /fc2* fc1* fc0          /* FC3 */
#define FC_4             fc2*/fc1*/fc0          /* FC4 (illegal) */
#define SUPV_DATA        fc2*/fc1* fc0          /* FC5 */
#define SUPV_PRGM        fc2* fc1*/fc0          /* FC6 */
#define CPUSPACE         fc2* fc1* fc0          /* FC7 */

#define DEVSPACE_DATA       /fc1* fc0           /* DATA 1/2 of Device Space */
#define DEVSPACE_PRGM        fc1*/fc0           /* PRGM 1/2 of Device Space */

/*
        Byte decode strobes
         Leave as "a".  Can be either p2_a or p_a as appropriate.
*/
#define BYTE24          /a1*/a0
#define BYTE16          /a1* a0
#define BYTE08           a1*/a0
#define BYTE00           a1* a0

/*
        Map for Control Space elements
*/
#define IDPROM          /p_a31*/p_a30*/p_a29*/p_a28     /* 0 */
#define PAGEMAP         /p_a31*/p_a30*/p_a29* p_a28     /* 1 */
#define SEGMAP          /p_a31*/p_a30* p_a29*/p_a28     /* 2 */
#define CONTEXT         /p_a31*/p_a30* p_a29* p_a28     /* 3 */
#define SYSENABLE       /p_a31* p_a30*/p_a29*/p_a28     /* 4 */
#define USERENABLE      /p_a31* p_a30*/p_a29* p_a28     /* 5 */
#define BUSERROR        /p_a31* p_a30* p_a29*/p_a28     /* 6 */
#define DIAG            /p_a31* p_a30* p_a29* p_a28     /* 7 */
```

```
#define dvma_stb_pal_off        bpal_off


/*      ethernet miscellaneous                                  */
#define ethernet_palr           apalr           /* 16R4 */
#define ethernet_pal            apal
#define ethernet_pal_ts_i       apal_ts_i
#define ethernet_pal_th_i       apal_th_i


/*      vme freeze controller                                   */
#define vme_freeze_palr         apalr           /* 16R4 */
#define vme_freeze_pal          apal
#define vme_freeze_pal_ts_i     apal_ts_i
#define vme_freeze_pal_th_i     apal_th_i


/*      vme arbiter                                             */
#define vme_arb_palr            apalr           /* 16R4 */
#define vme_arb_pal_ts_i        apal_ts_i
#define vme_arb_pal_th_i        apal_th_i


/*      vme master                                              */
#define vme_master_pal          apal            /* 20L8 */
#define vme_master_pal_skew     apal_skew


/*      vme slave space address decoder                         */
#define vme_slvspc_pal  apal                    /*16L8  */


/*      vme slave request generator                             */
#define vme_slvreq_palr         apalr           /*16L8  */
#define vme_slvreq_pal          apal
#define vme_slvreq_pal_ts_i     apal_ts_i
#define vme_slvreq_pal_th_i     apal_th_i


/*      vme buffer controller                                   */
#define vme_buffer_pal  apal                    /* 20L8 */
```

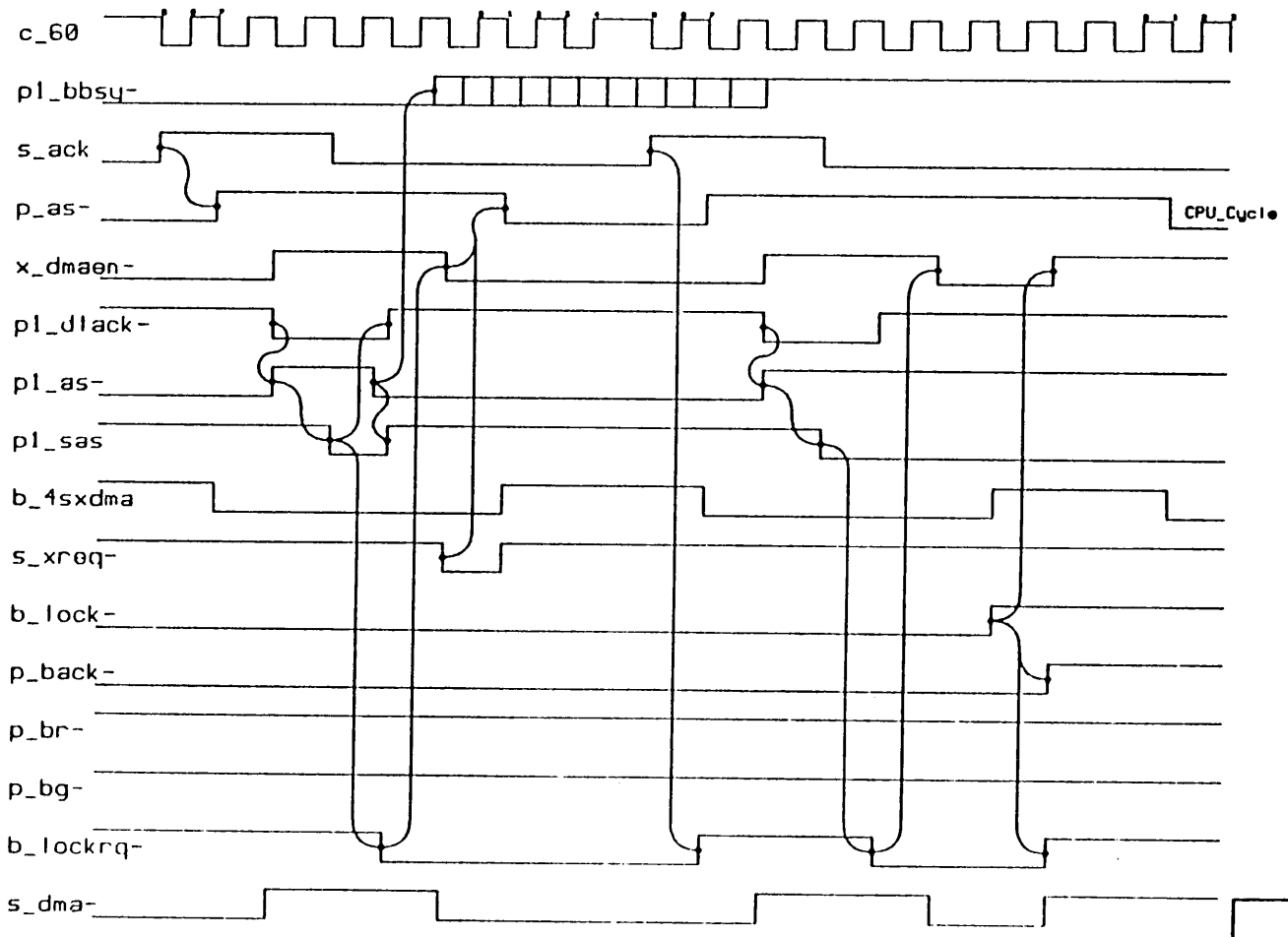
```
#include "lib/pals.h"
/*
        Assign speed versions for pals used in 2060 design.

        I've tried to group the pals into the functional block that
        is most appropriate for each pal.  I've also included just
        the parameters that are relavent for each pal.  Besides
        keeping this file from becoming tooooo cluttered, it collects
        in one central location the kind of pal we are using for each
        section, and how it is being used.
*/

/*      clock           */
#define clock_pal       bpal            /* 16r8b */
#define clock_palr      bpalr
#define clock_palskew   bpalskew
#define clock_pal_ts_i  bpal_ts_i
#define clock_pal_th_i  bpal_th_i


/*      uP              */
#define mmuvalid_pal    bpal            /* 1618b */


/*      mmu             */
#define cpuspace_pal    bpal            /* 1618b */


/*      mem             */
#define ras_pal         bpal            /* 1618b */

#define cas_pal         bpal            /* 1618b */

#define mem_pal         apal            /* 1618a */
#define mem_pal_on      apal_on
#define mem_pal_off     apal_off


/*      io decoder      */
#define iobus_pal       pal
#define mosbus_pal      pal20110        /* 20110 */
#define ttlbus_pal      pal20110        /* 20110 */
#define ctlspc_mmu_pal  pal20110        /* 20110 */
#define ctlspc_sys_pal  pal20110        /* 20110 */


/*      dvma controller                                 */
#define dvma_ctlr_palr          apalr           /* 20R8 */
#define dvma_ctlr_pal           apal
#define dvma_ctlr_pal_ts_i      apal_ts_i
#define dvma_ctlr_pal_th_i      apal_th_i


/*      dvma strobe generator                           */
#define dvma_stb_pal            bpal            /* 16L8 */
#define dvma_stb_pal_on         bpal_on
```



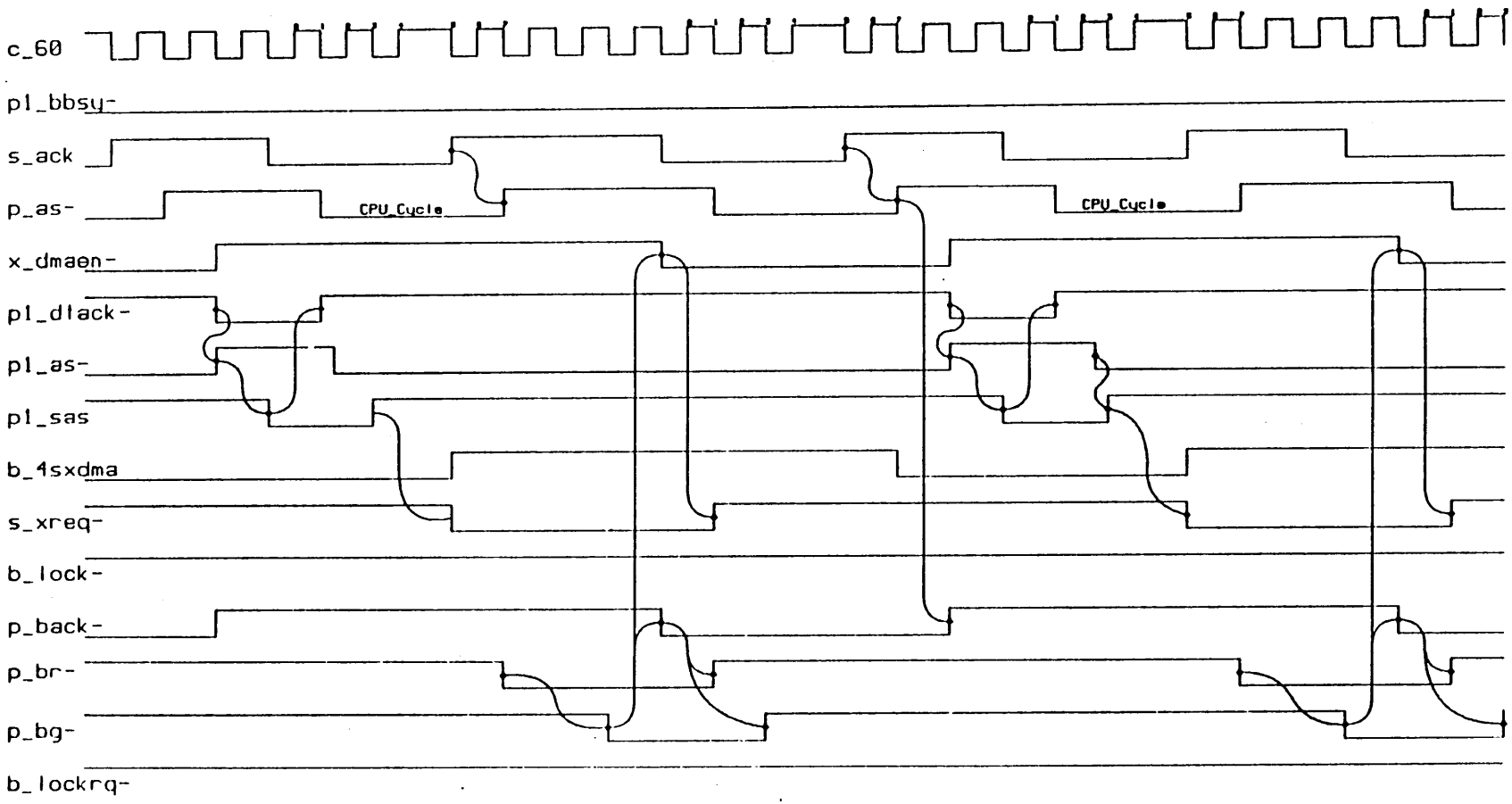
SUN MICROSYSTEMS, INC.		
Refresh_Cycle		
B	NO.	REV
DATE 2/1	SCALE	PAGE OF



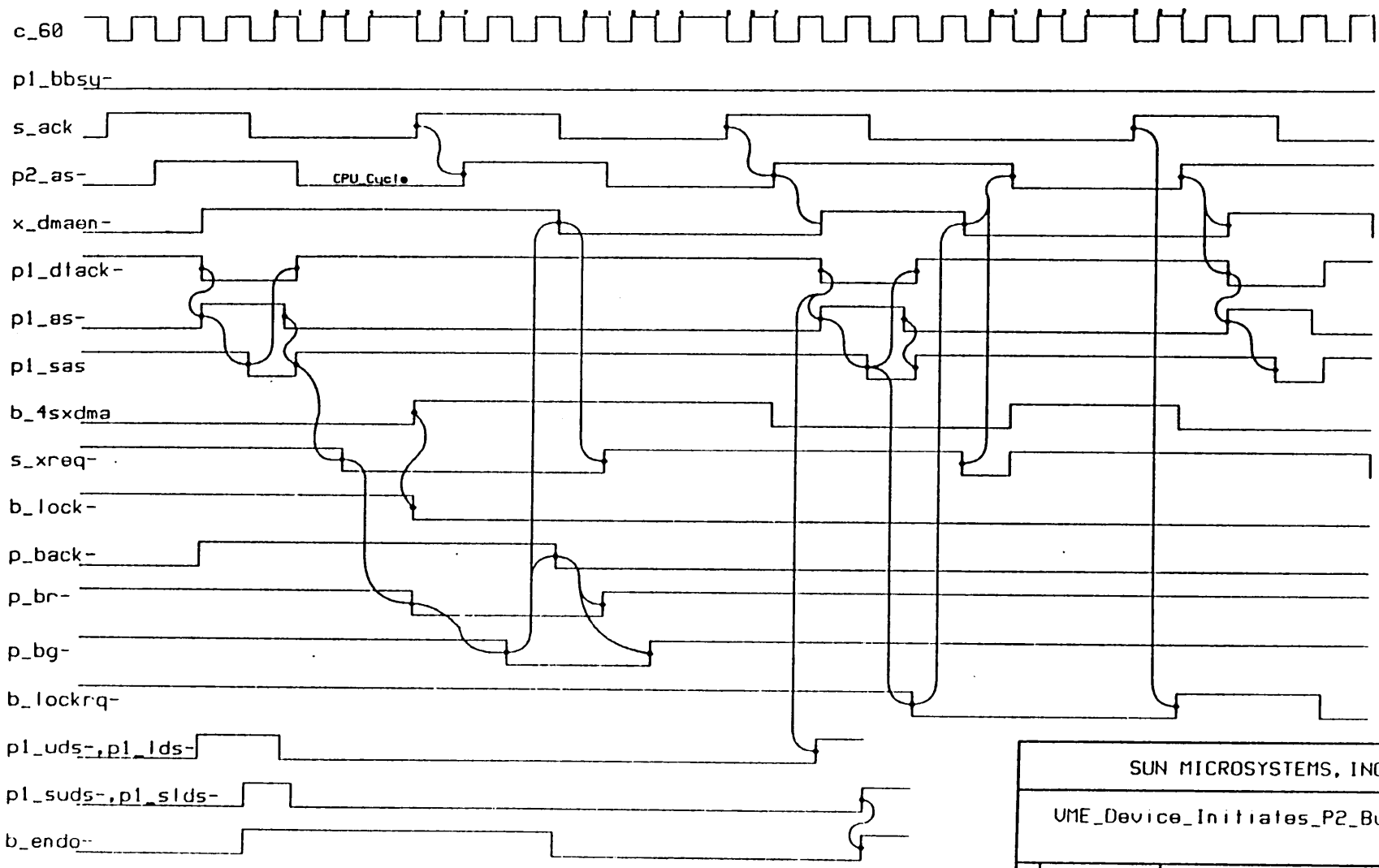
Ethernet_Cycle_(Fastest_Case)



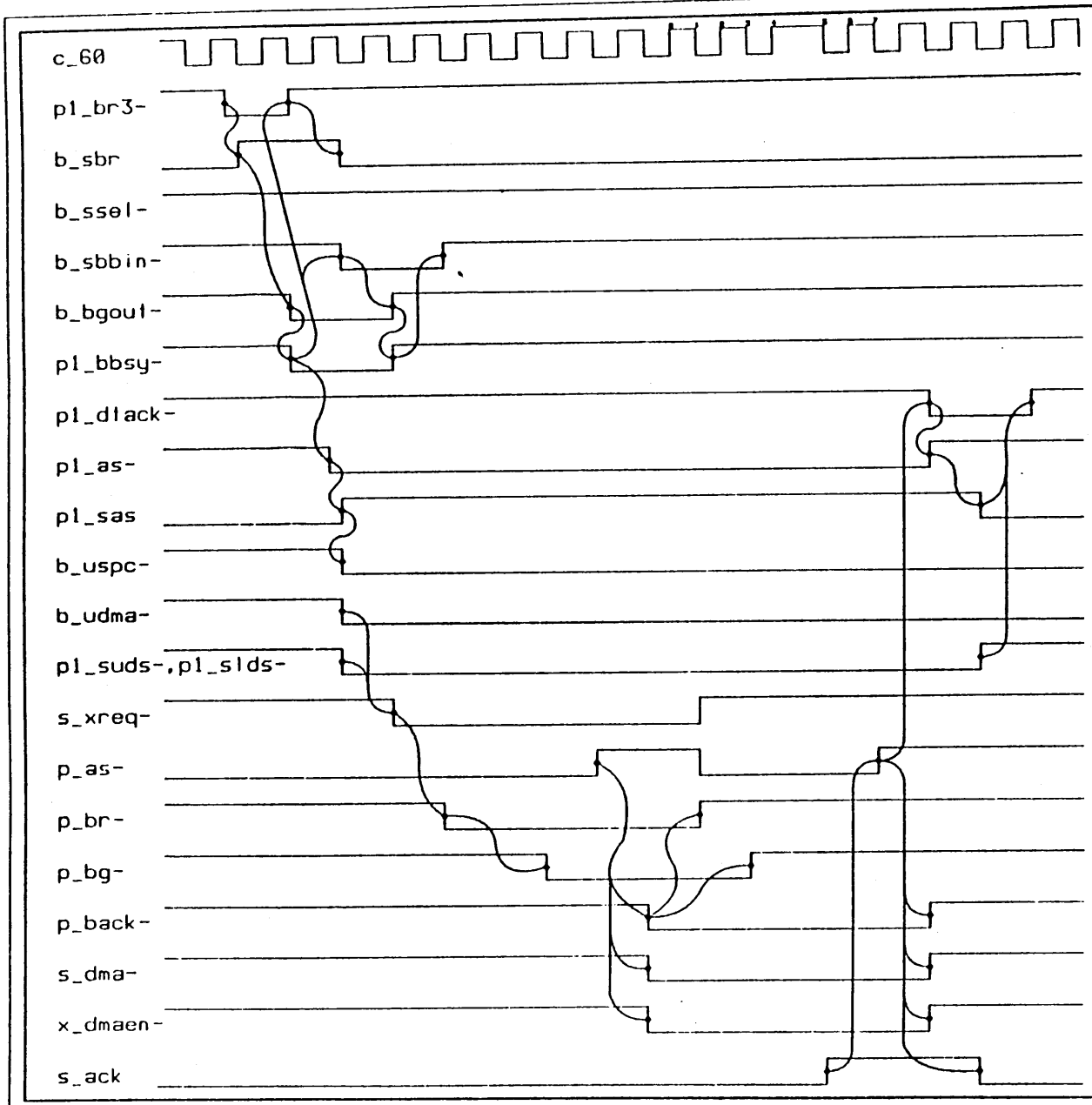
SUN MICROSYSTEMS, INC.			
VME_Device_Ends_P2_Bus_Lock			
B		NO.	REV
DATE 2/1		SCALE	P/ OF



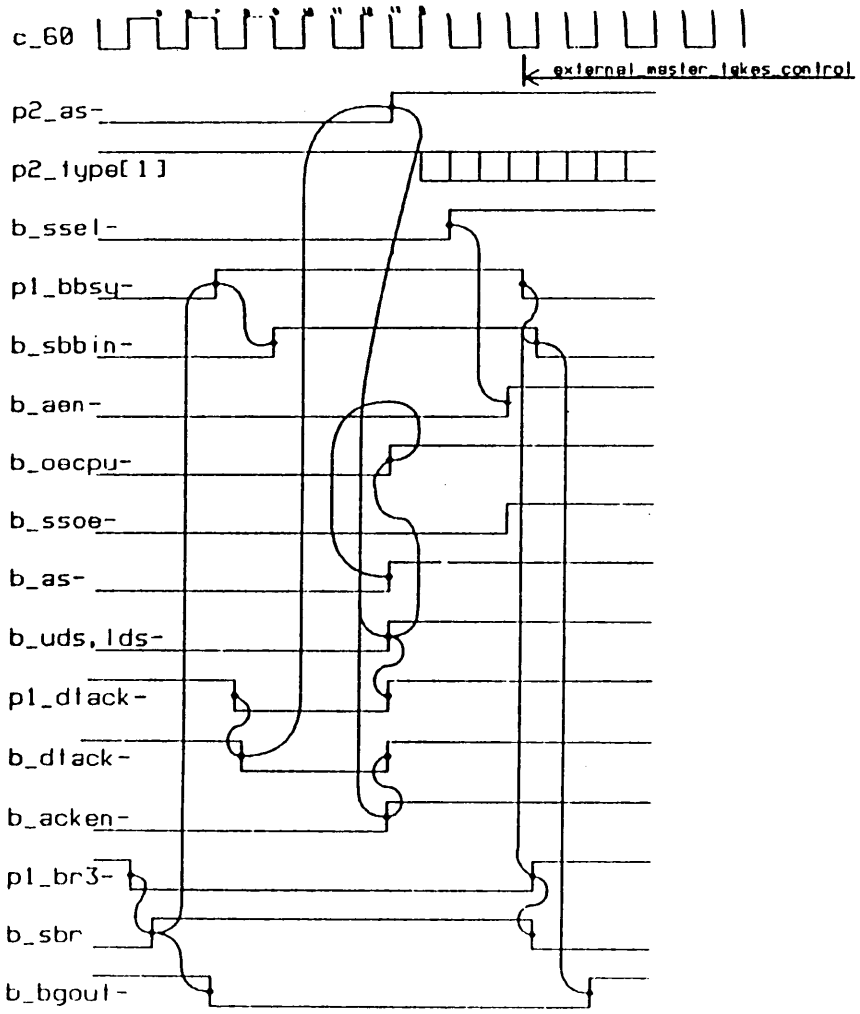
SUN MICROSYSTEMS, INC.			
VME_Device_Not_Fast_Enough_to Initiate_P2_Bus_Lock			
B		NO.	REV
DATE 2/1		SCALE	PAGE OF



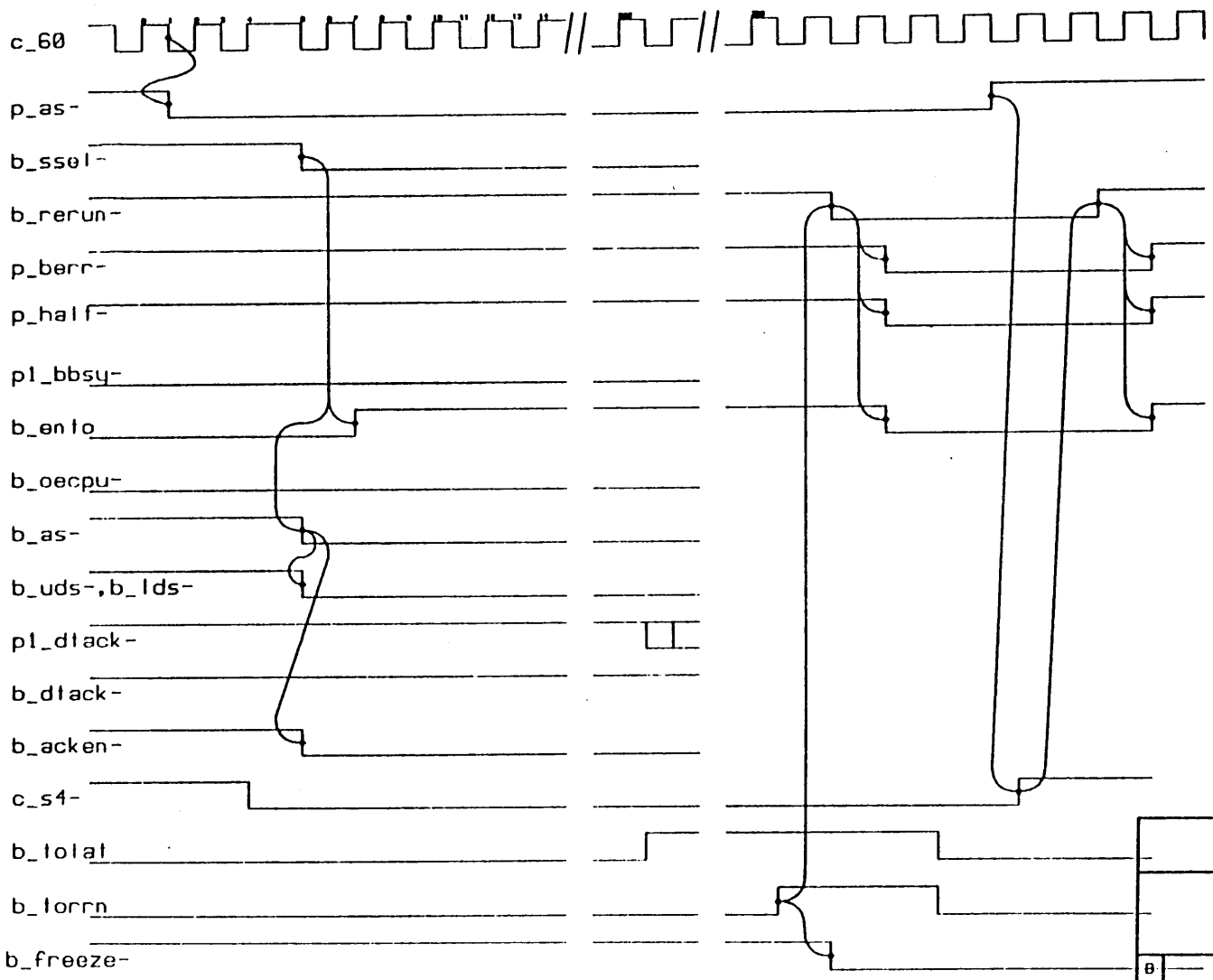
SUN MICROSYSTEMS, INC.			
UME_Device_Initiates_P2_Bus_Lock			
B		NO.	REV
DATE 2/1		SCALE	PAGE OF



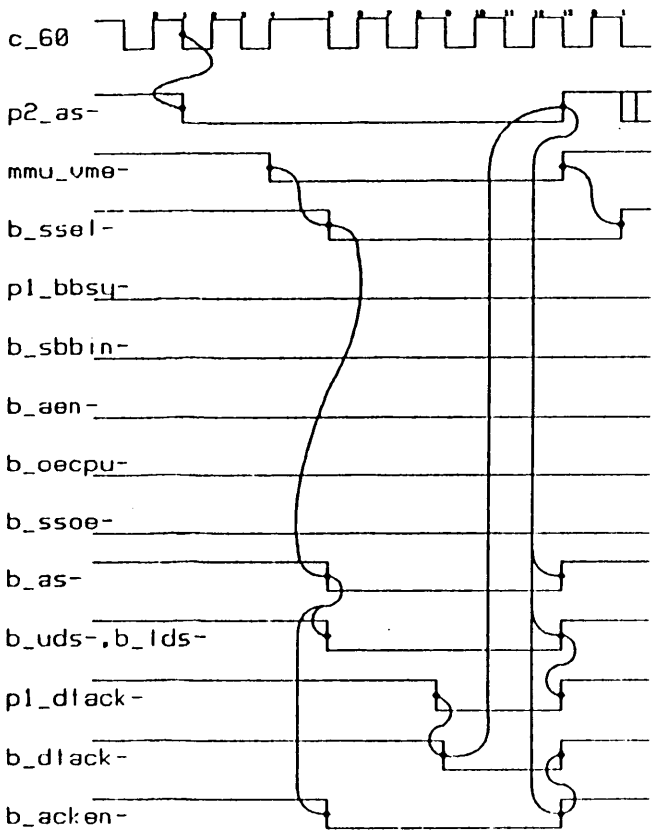
SUN MICROSYSTEMS, INC.			
VME_Device_Acquires_VMEbus and_Accesses_P2_Bus			
B		NO.	REV
DATE 2/1	SCALE	PAGE	OF



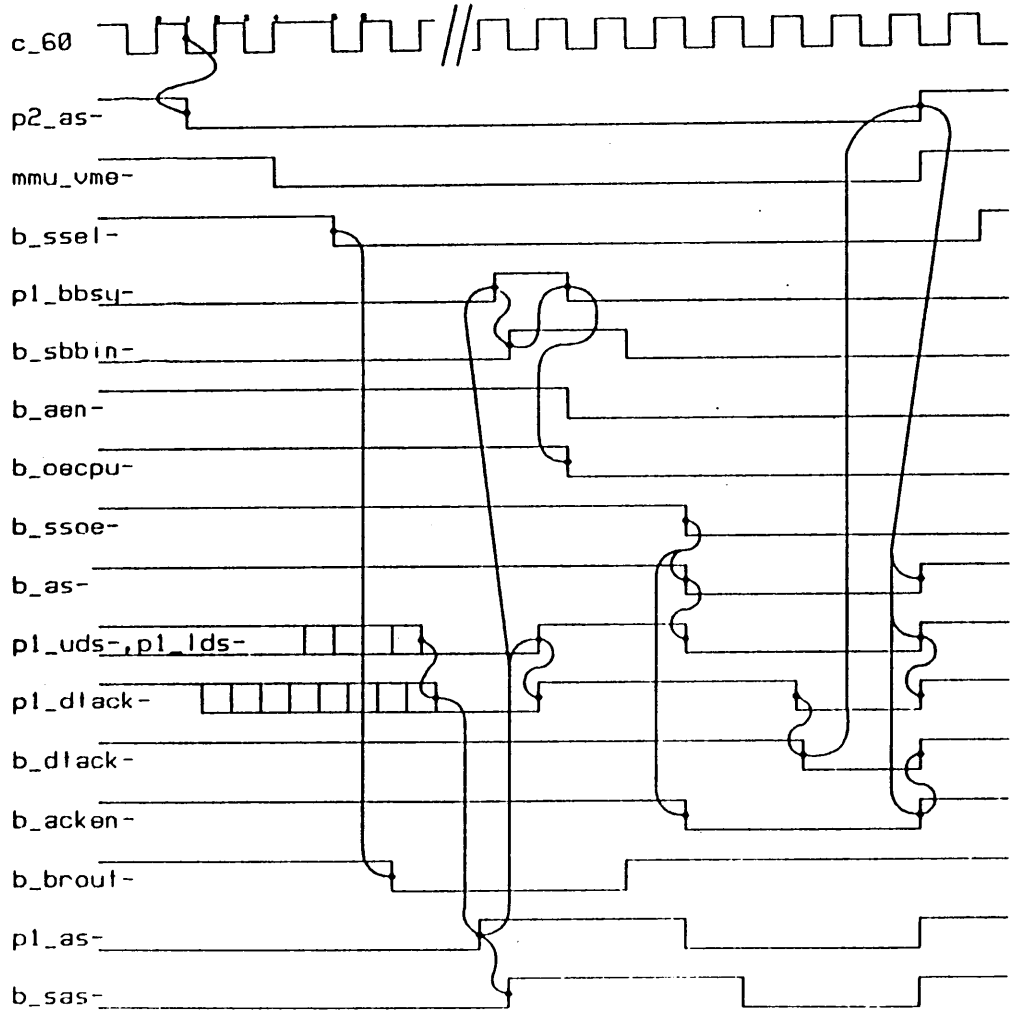
SUN MICROSYSTEMS, INC.			
CPU_Relinquishes_VMEbus (Currently_Bus_Master)			
B		NO.	REV
DATE 2/1		SCALE	PAGE OF



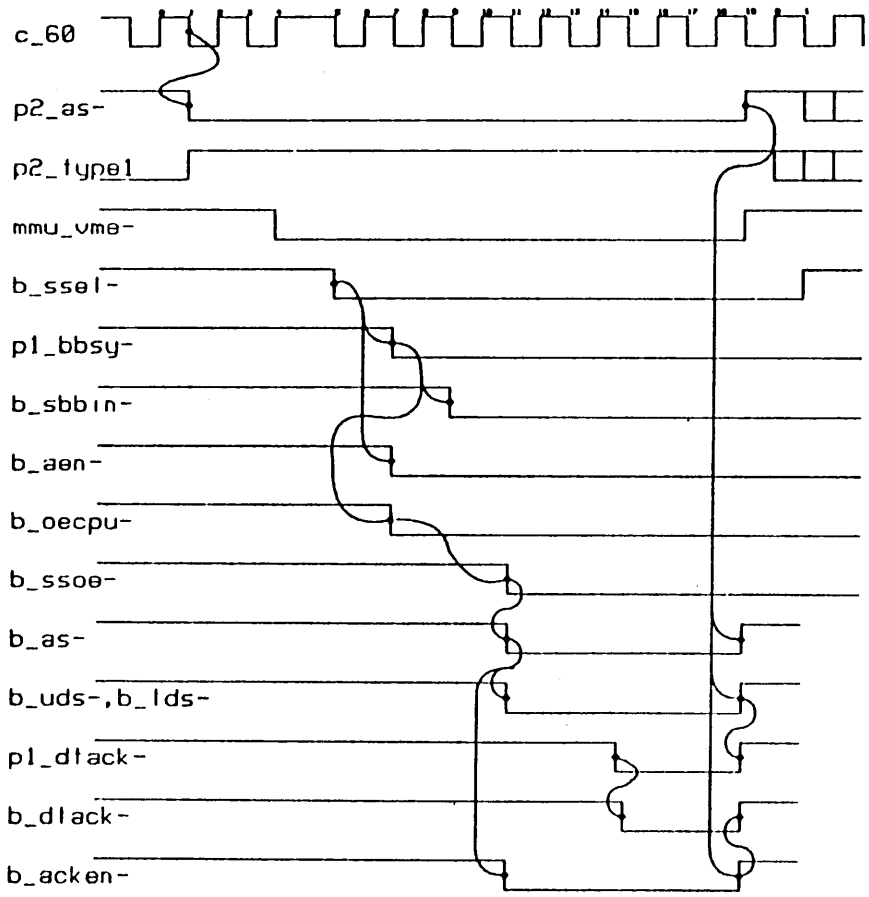
SUN MICROSYSTEMS, INC.		
CPU_Rerun_During_VME_Access (Currently_Bus_Master)		
B	NO.	REV
DATE 2/1	SCALE	PAGE OF



SUN MICROSYSTEMS, INC.			
CPU_Access_of_VMEbus (Currently_Bus_Master)			
B		NO.	REV
DATE 2/1		SCALE	PAGE OF

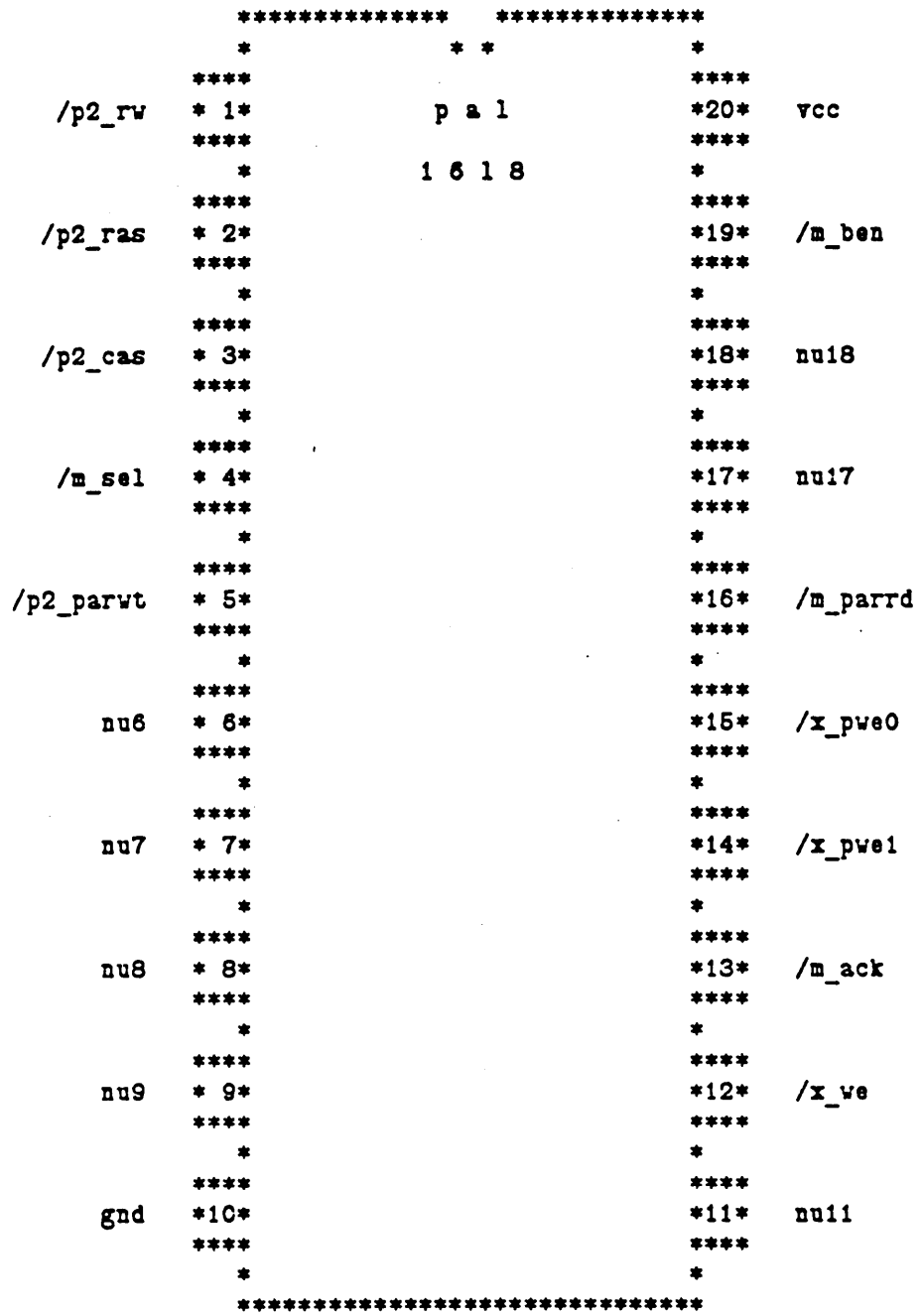


SUN MICROSYSTEMS, INC.			
CPU_Access_of_Busy_VMEbus			
B		NO.	REV
DATE 2/1		SCALE	PAGE OF



SUN MICROSYSTEMS, INC.			
CPU_Access_of_Idle_VMEbus (Not_Currently_Bus_Master)			
B		NO.	REV
DATE 1/1		SCALE	PAGE OF

U3114 2060's memory acknowledge/buffer control pal



VME ARBITER & REQUESTOR, - ARBITER MODE

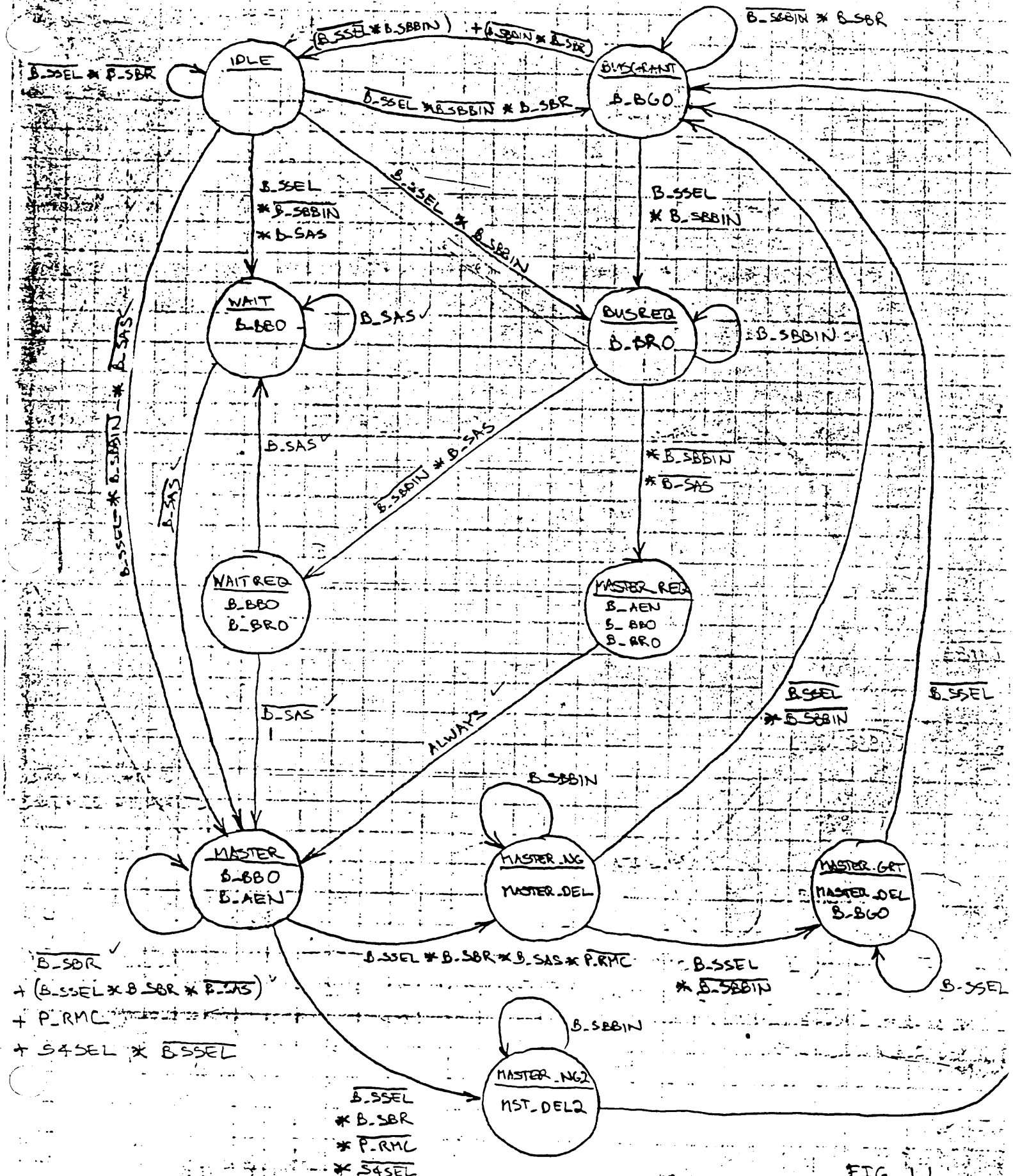


FIG. 1.1

VME ARBITER/REQUESTOR, REQUESTOR-ONLY MODE

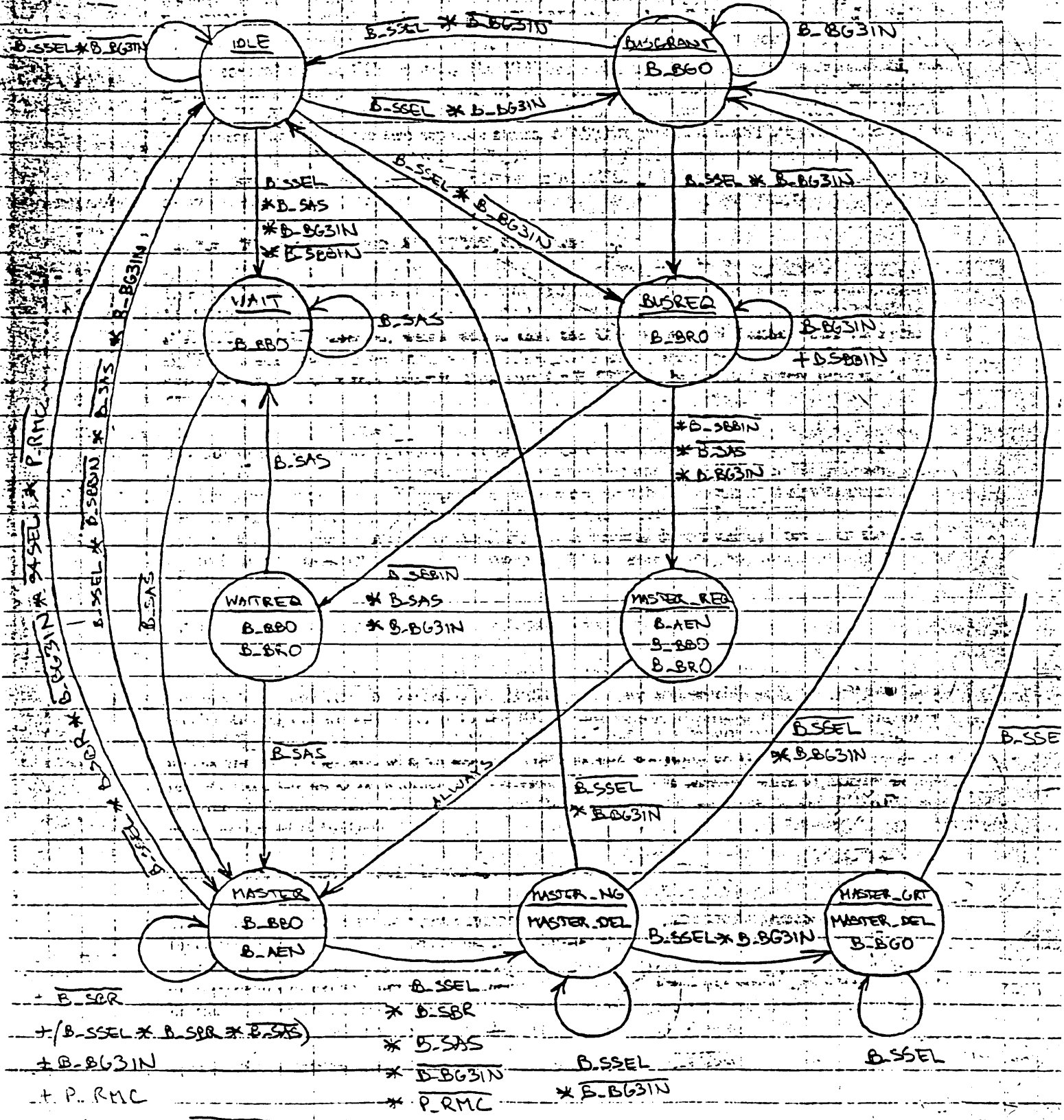


Fig. 1.2

VME SELECT & FREEZE STATE MACHINE

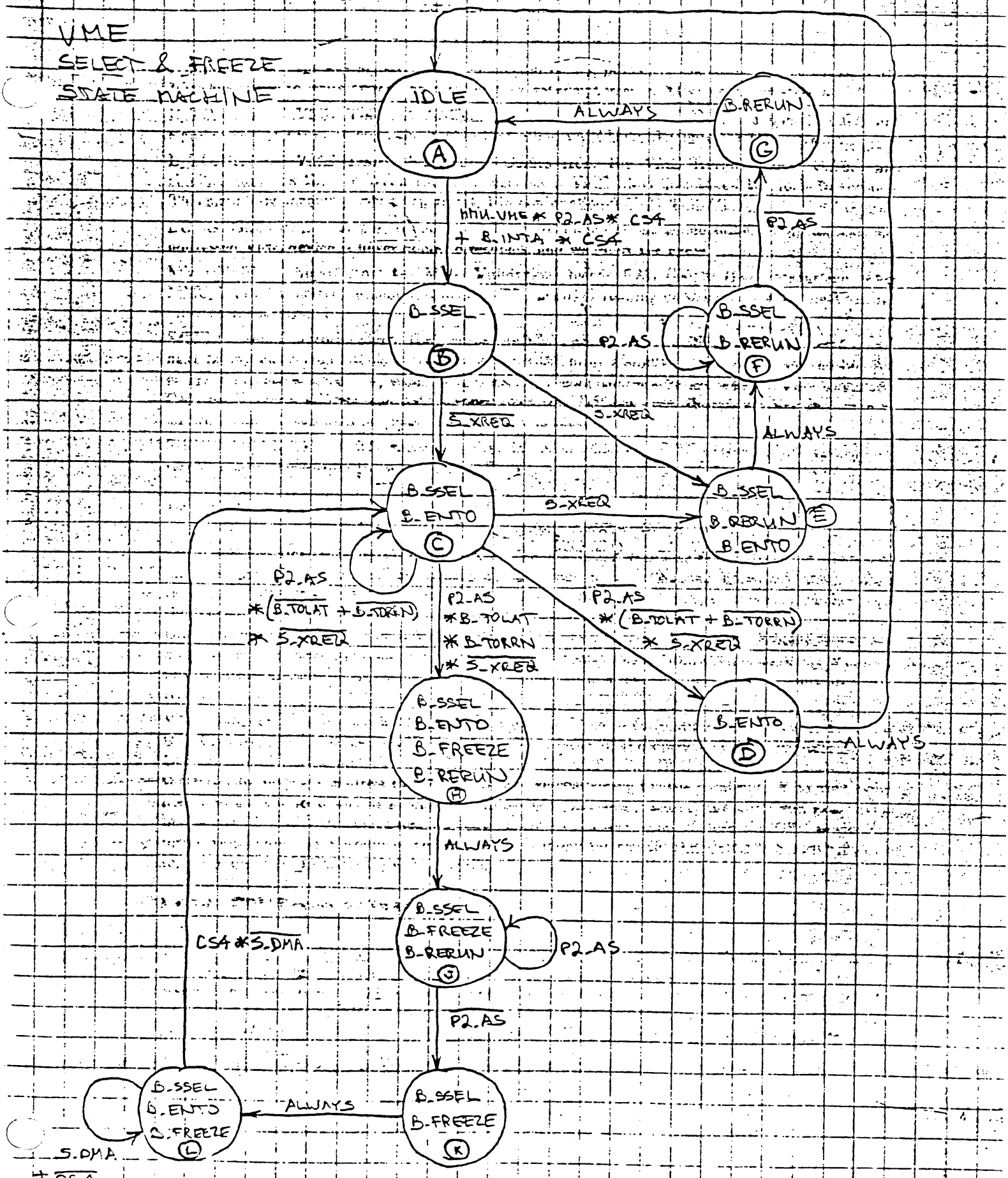


Fig. 1.3

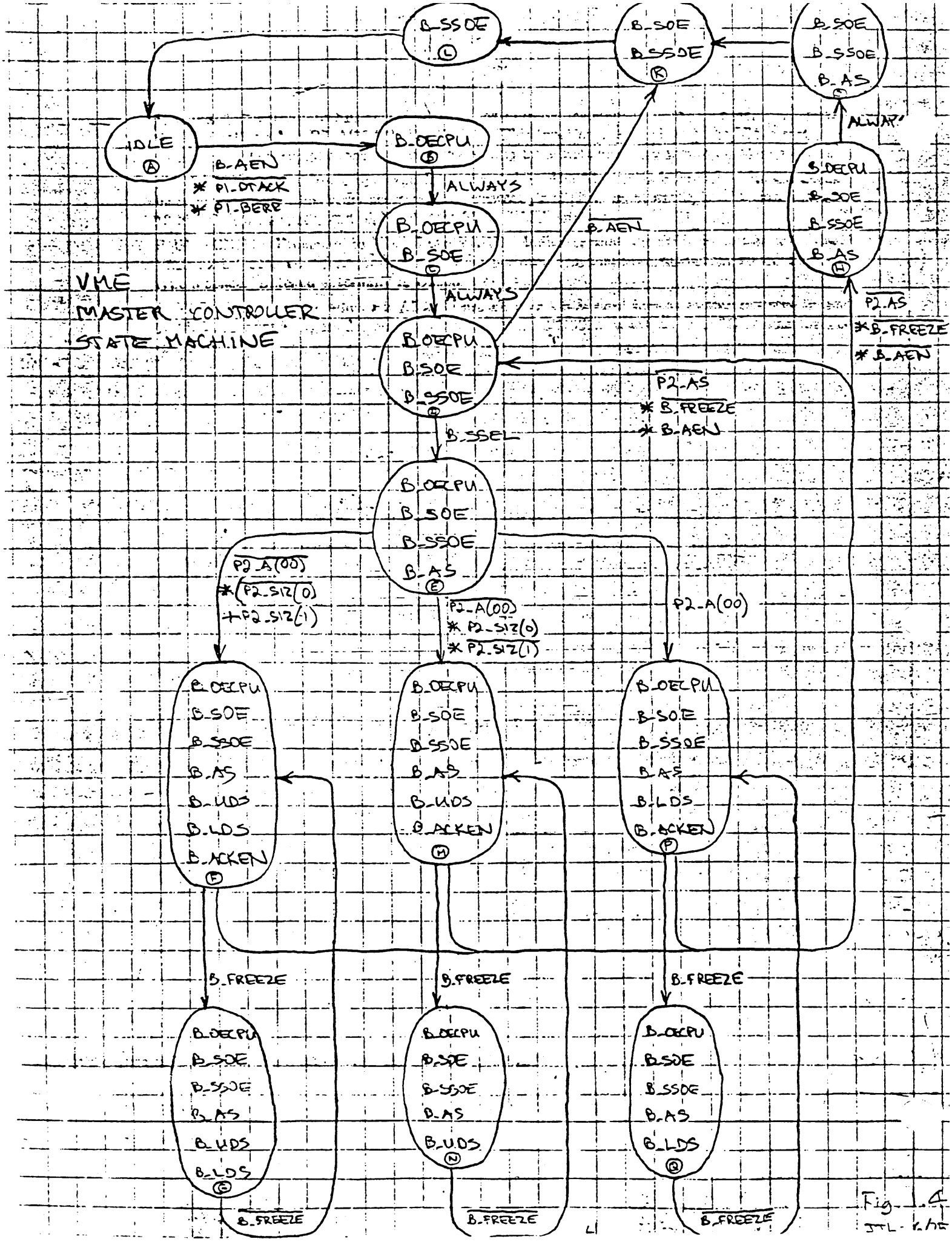
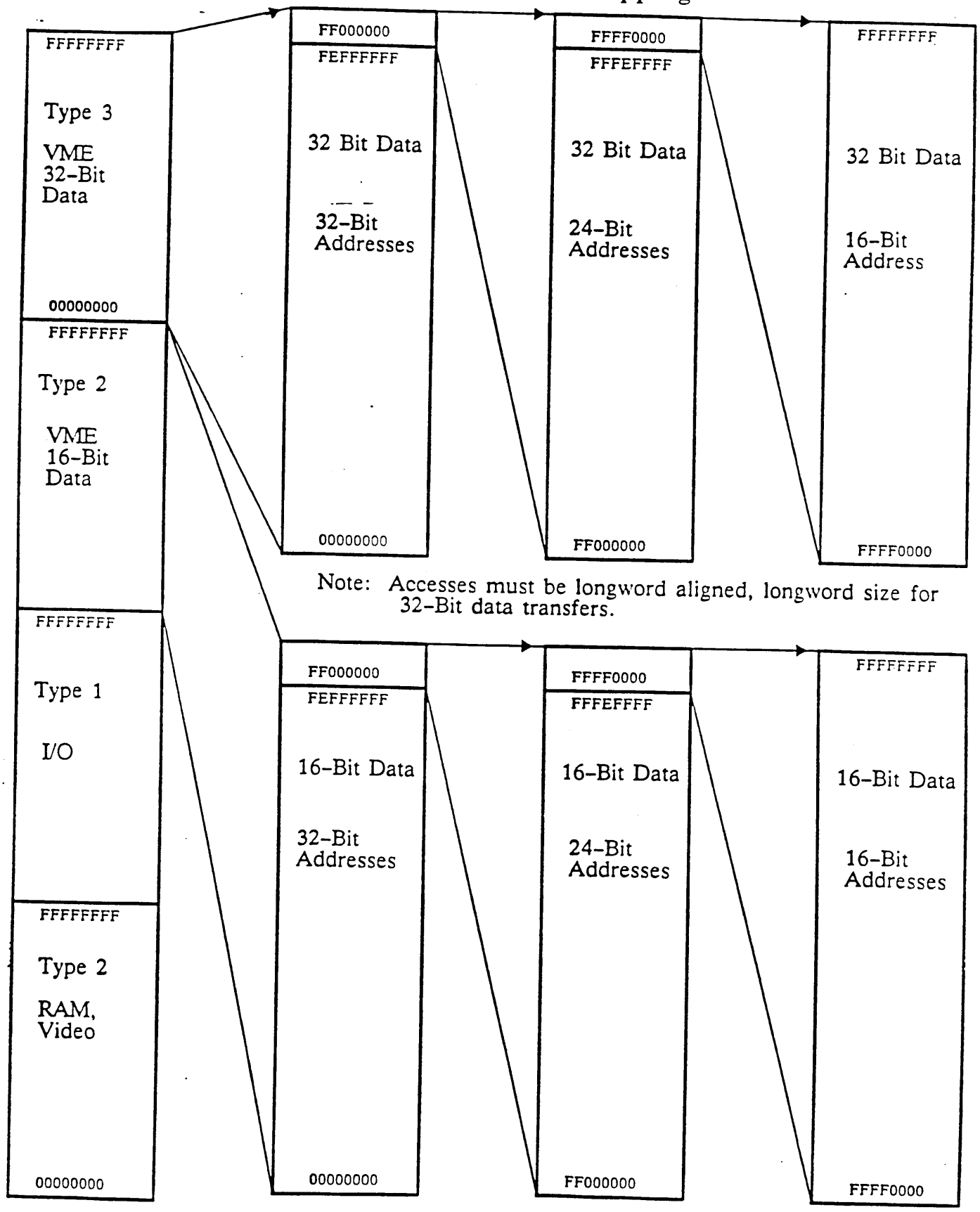

VME
MASTER CONTROLLER
STATE MACHINE
IDLE
A
B_AEN
* P1_DTACK
* P1_BERR
B_DECPU
B
ALWAYS
B_DECPU
B_SOE
C
ALWAYS
B_DECPU
B_SOE
B_SSOE
D
B_SSEL
B_DECPU
B_SOE
B_SSOE
B_AS
E
P2_A(00)
* (P2_SIZ(0)
+P2_SIZ(1)
P2_A(00)
* P2_SIZ(0)
* P2_SIZ(1)
P2_A(00)
B_DECPU
B_SOE
B_SSOE
B_AS
B_UDS
B_LDS
B_ACKEN
F
B_DECPU
B_SOE
B_SSOE
B_AS
B_UDS
B_ACKEN
M
B_DECPU
B_SOE
B_SSOE
B_AS
B_LDS
B_ACKEN
P
B_FREEZE
B_FREEZE
B_FREEZE
B_DECPU
B_SOE
B_SSOE
B_AS
B_UDS
B_LDS
G
B_DECPU
B_SOE
B_SSOE
B_AS
B_UDS
N
B_DECPU
B_SOE
B_SSOE
B_AS
B_LDS
Q
B_FREEZE
B_FREEZE
B_FREEZE
B_SSOE
L
B_SOE
B_SSOE
K
B_SOE
B_SSOE
B_AS
J
B_DECPU
B_SOE
B_SSOE
B_AS
H
ALWAYS
B_AEN
P2_AS
* B_FREEZE
* B_AEN
P2_AS
* B_FREEZE
* B_AEN

Fig. 4

Sun-3 Physical Address Mapping



Note: Accesses must be longword aligned, longword size for 32-Bit data transfers.

Fig. 1.3

VMEbus Slave Address Mapping

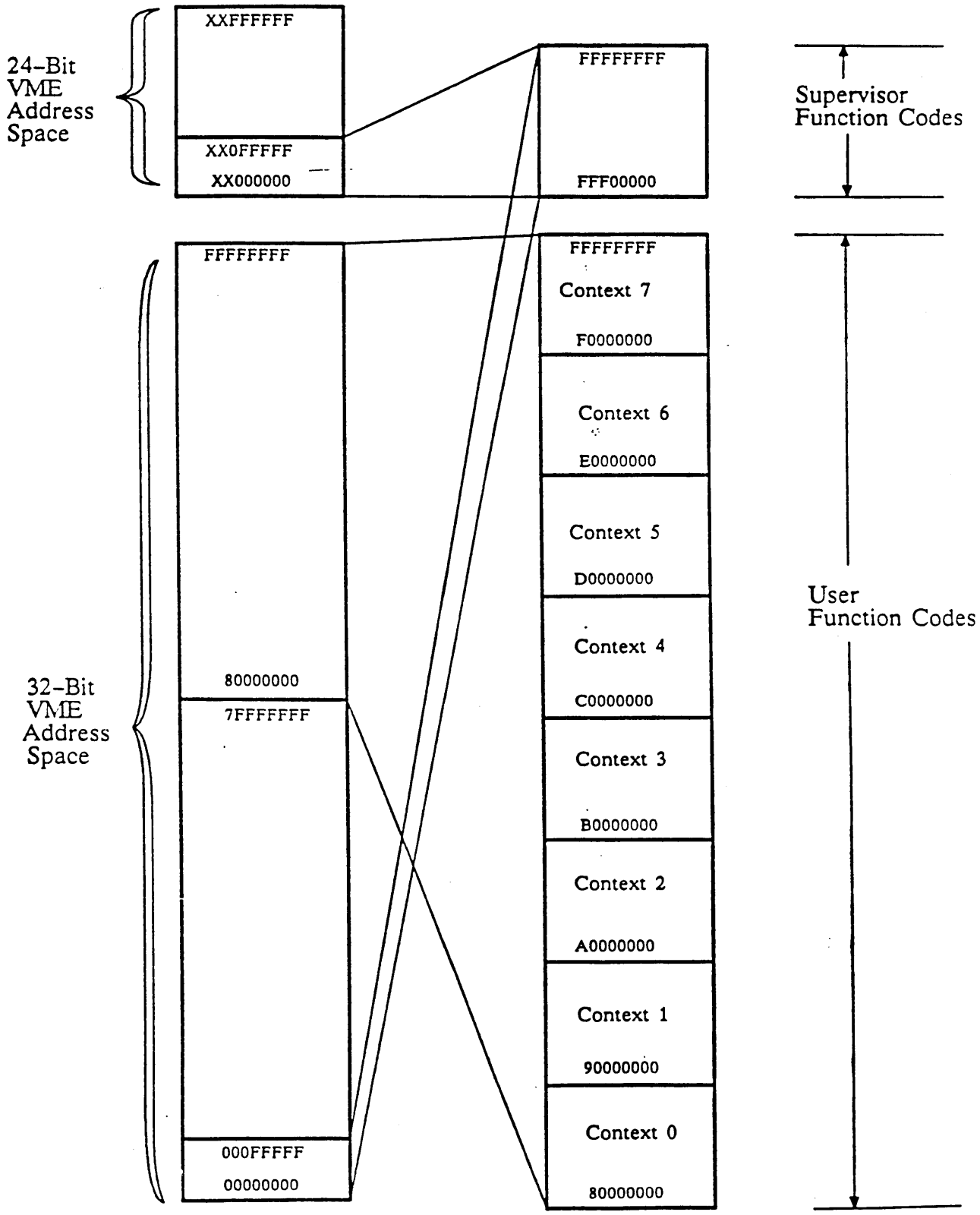


Fig. 1.6

VME SLAVE REQUESTOR STATE MACHINE

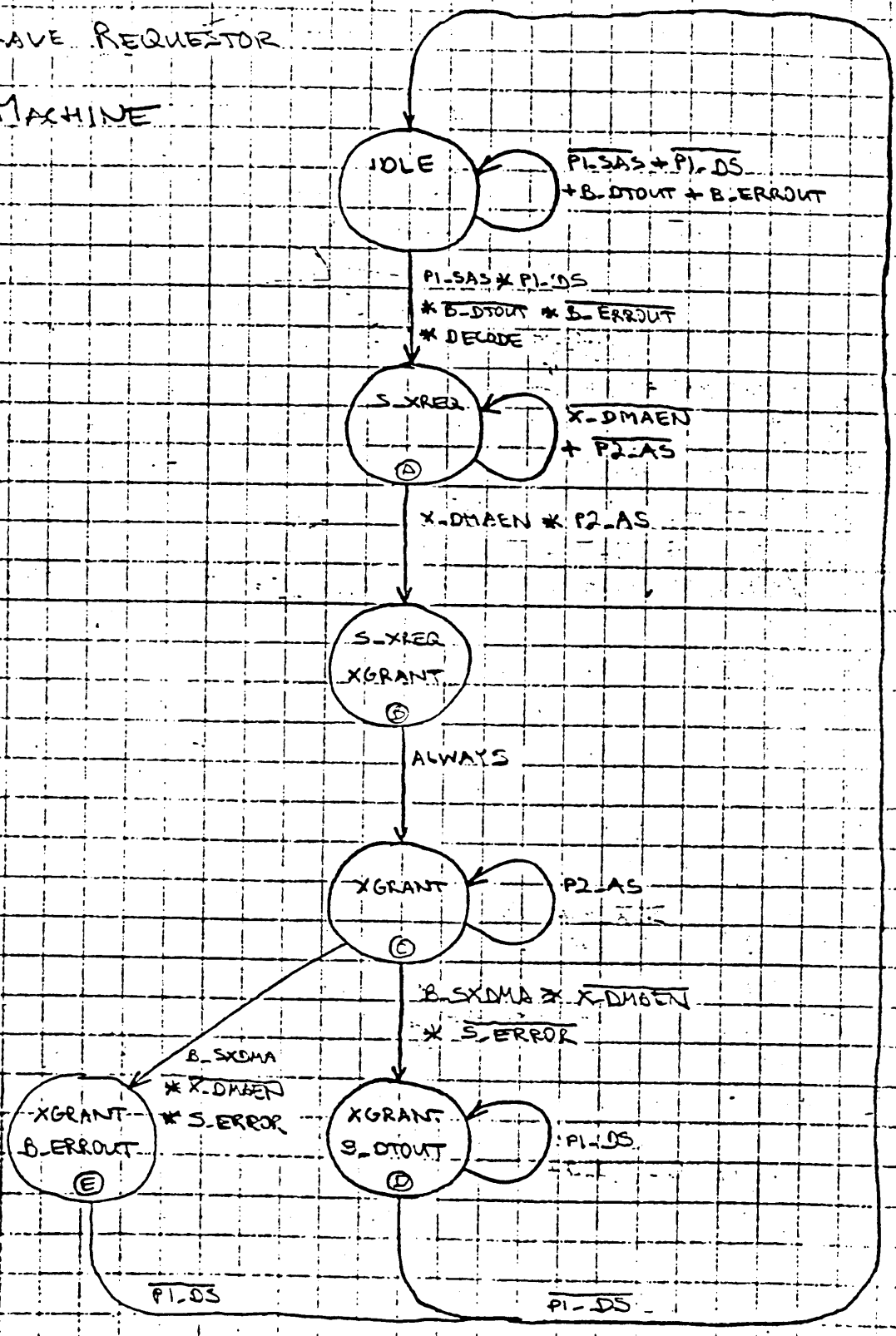
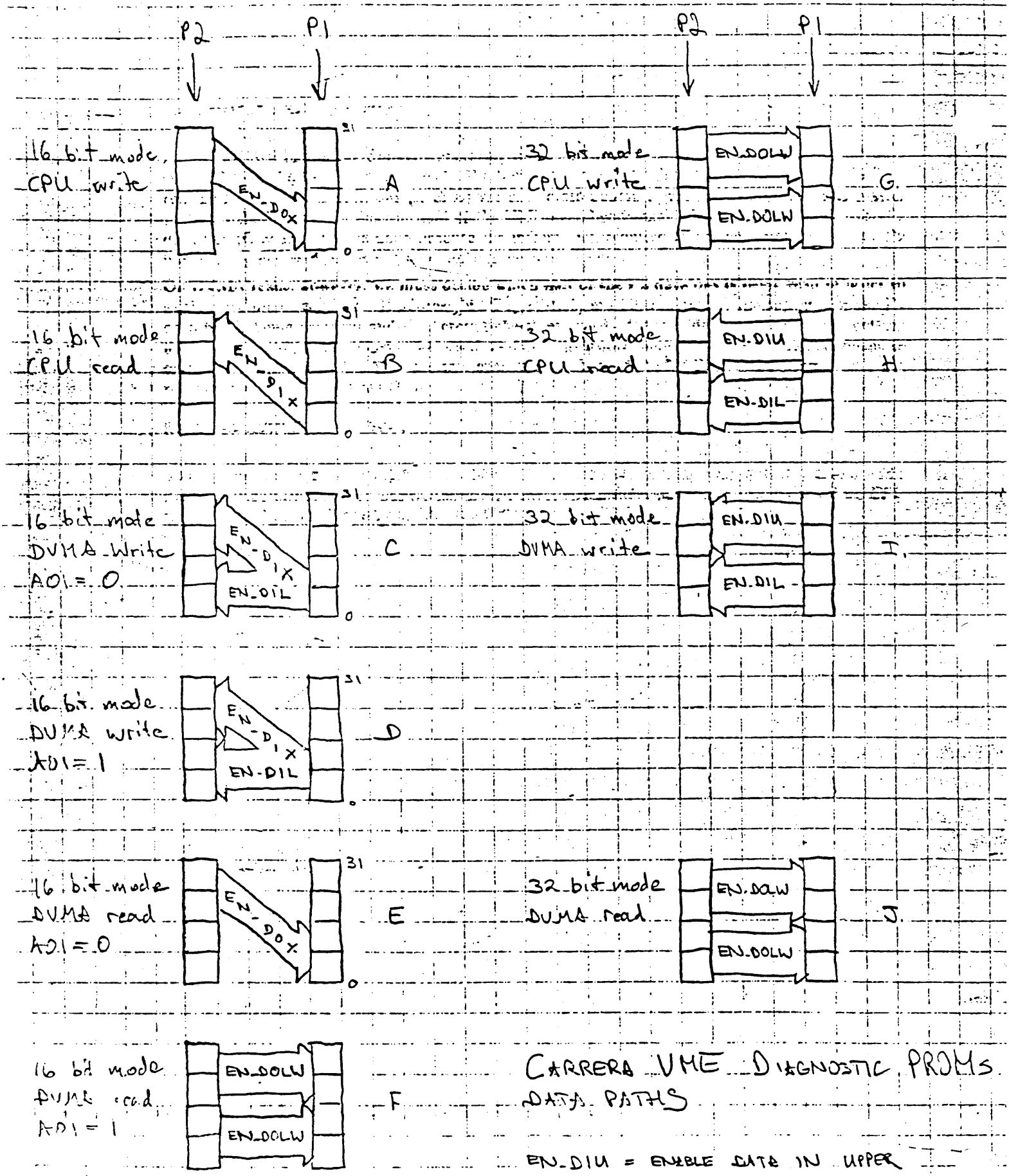


Fig. 1.7



CARRERA VME DIAGNOSTIC PROMS

DATA PATHS

EN_DIU = ENABLE DATA IN UPPER

EN_DIL = ENABLE DATA IN LOWER

EN_DIX = ENABLE DATA IN CROSS

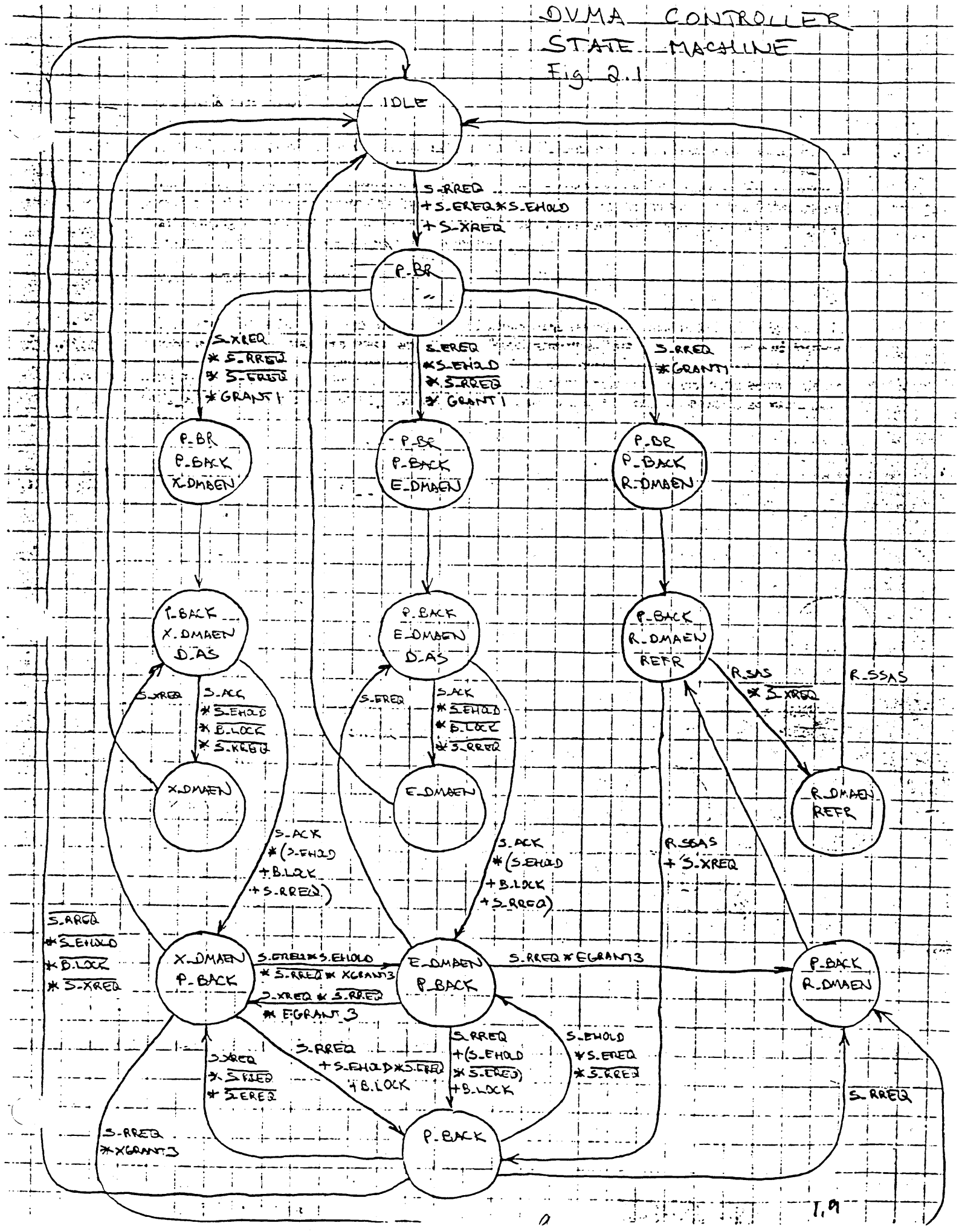
EN_DOLW = ENABLE DATA OUT LONGWORD

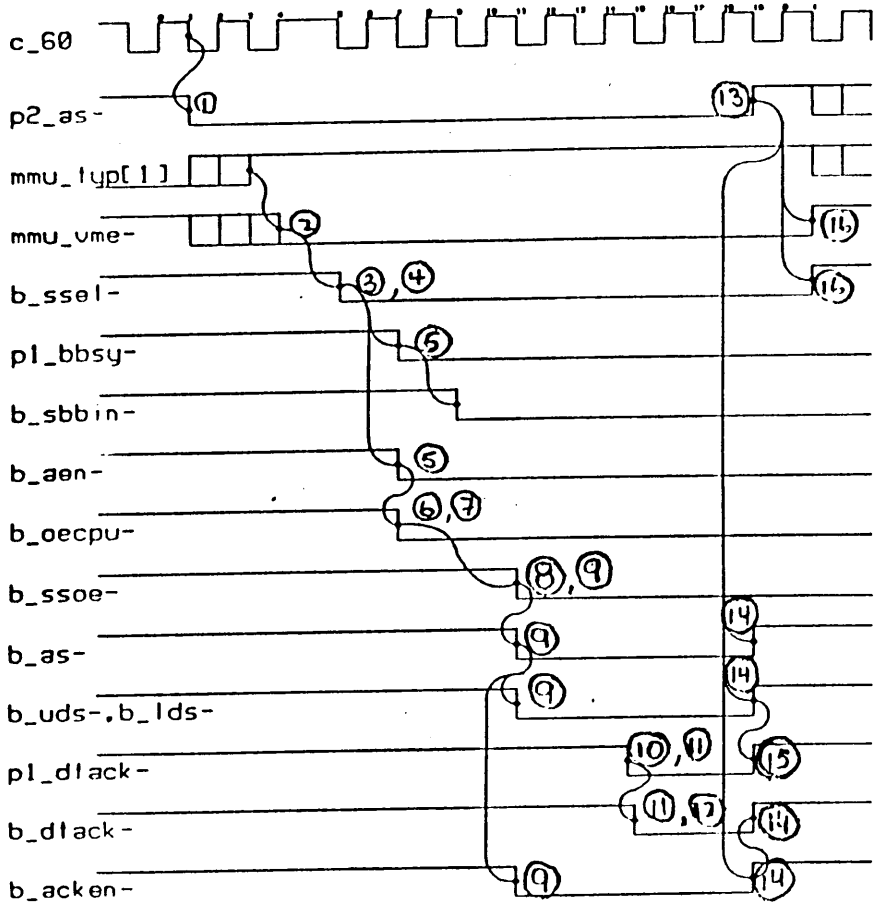
EN_DOX = ENABLE DATA OUT CROSS

Fig. 1.8

DVMA CONTROLLER STATE MACHINE

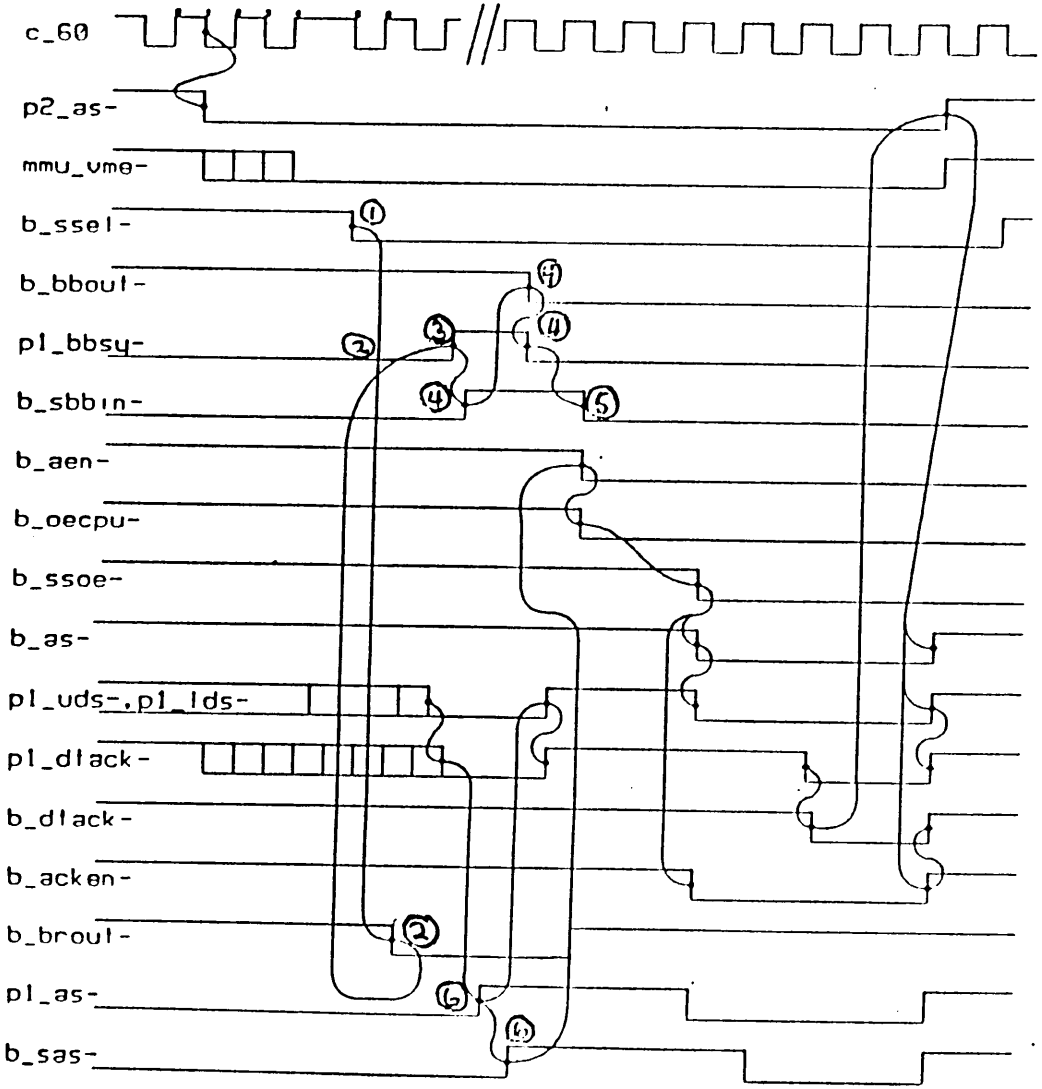
Fig. 2.1



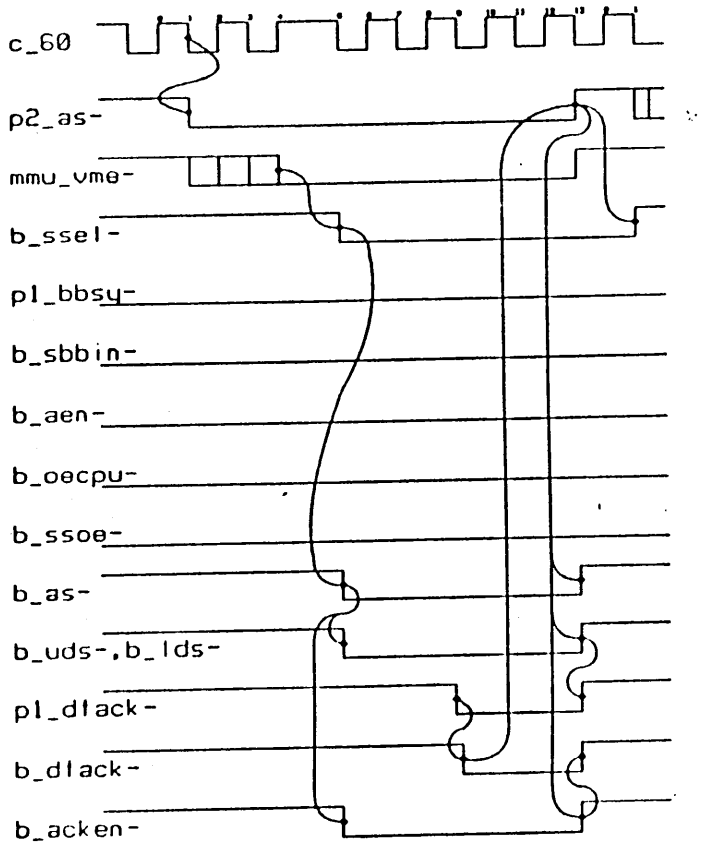


CPU_Access_of_Idle_VMEbus

(Not_Currently_Bus_M

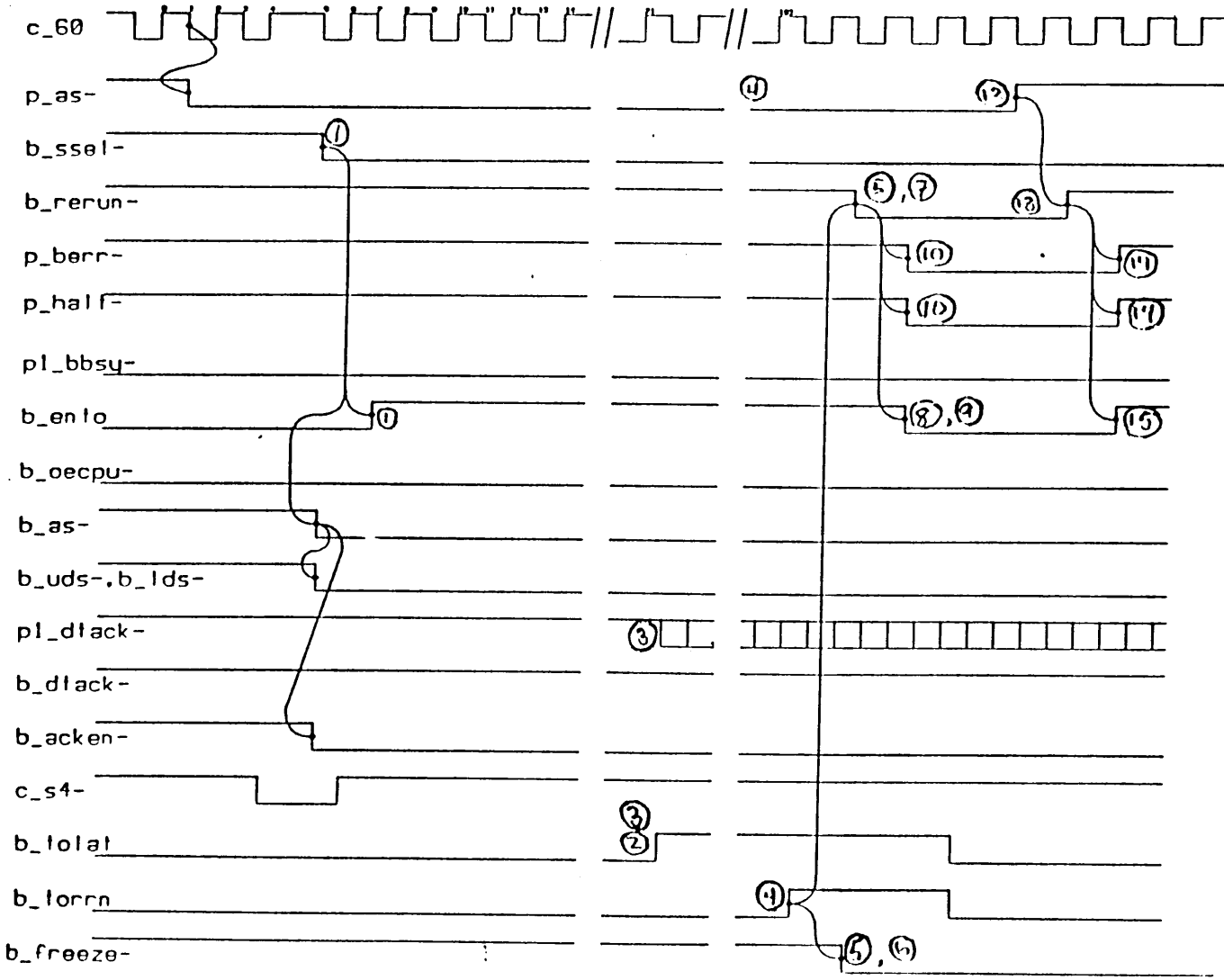


CPU_Access_of_Busy_VMEbus



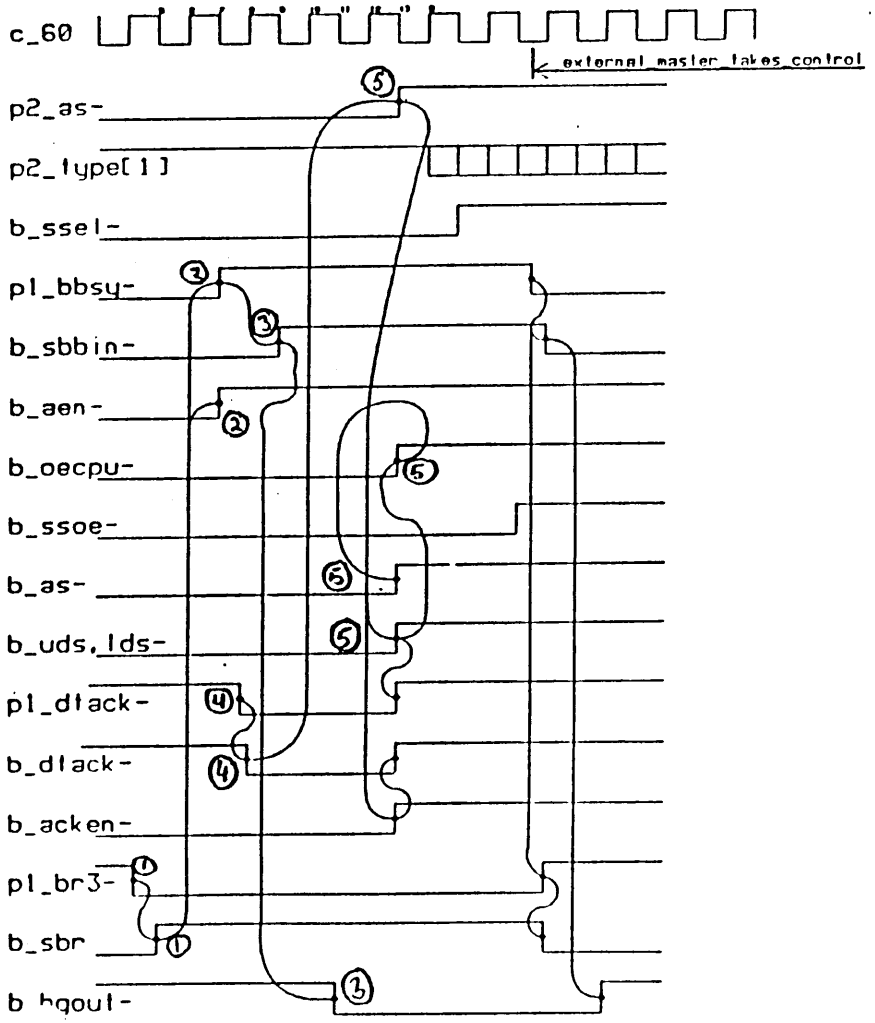
CPU_Access_of_VMEbus

(Currently_Bus_Master)



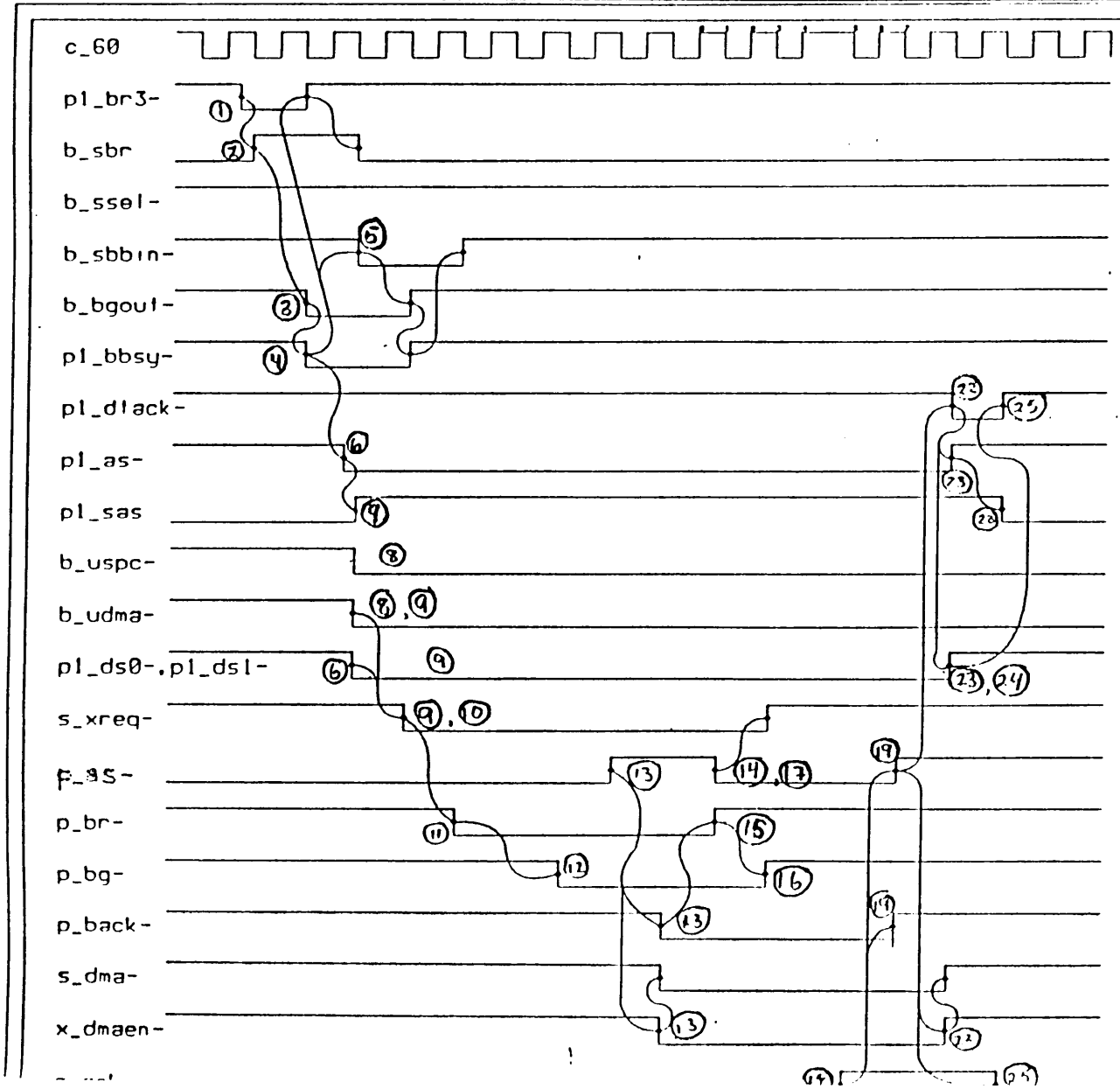
CPU_Rerun_During_VME_Access

(Currently_Bus_Master)

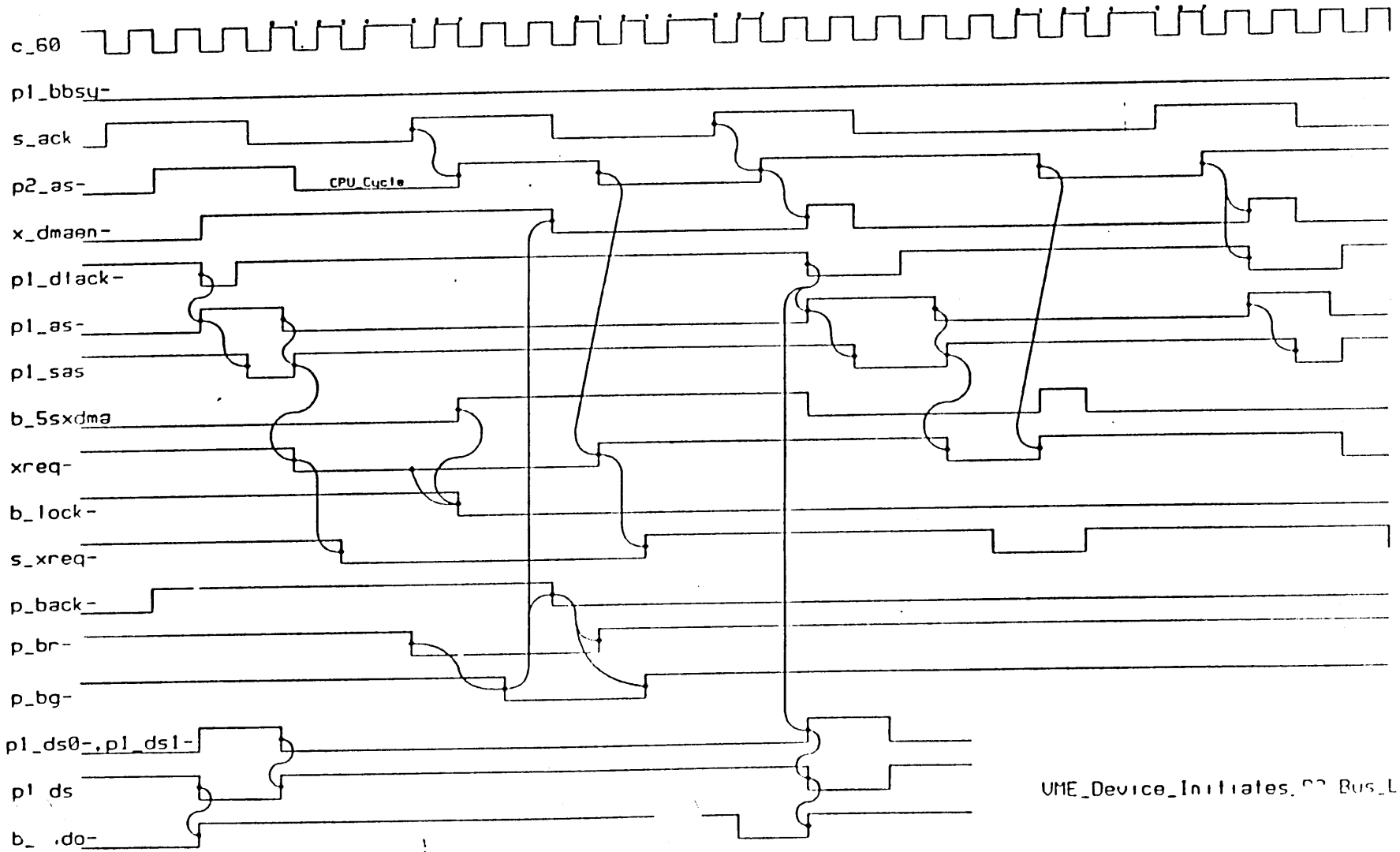


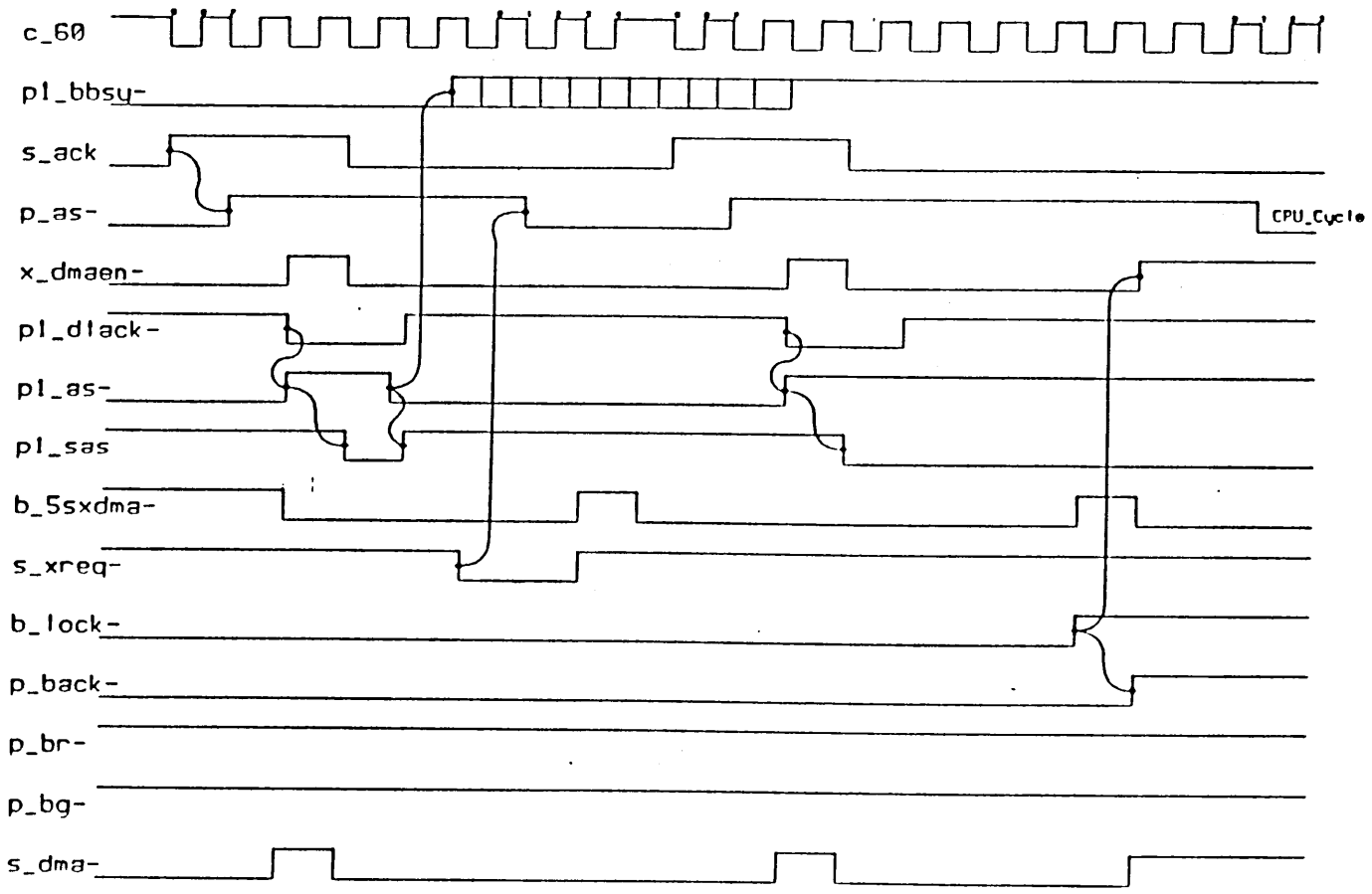
CPU_Relinquishes_VMEbus

(Currently_Bus_Maste

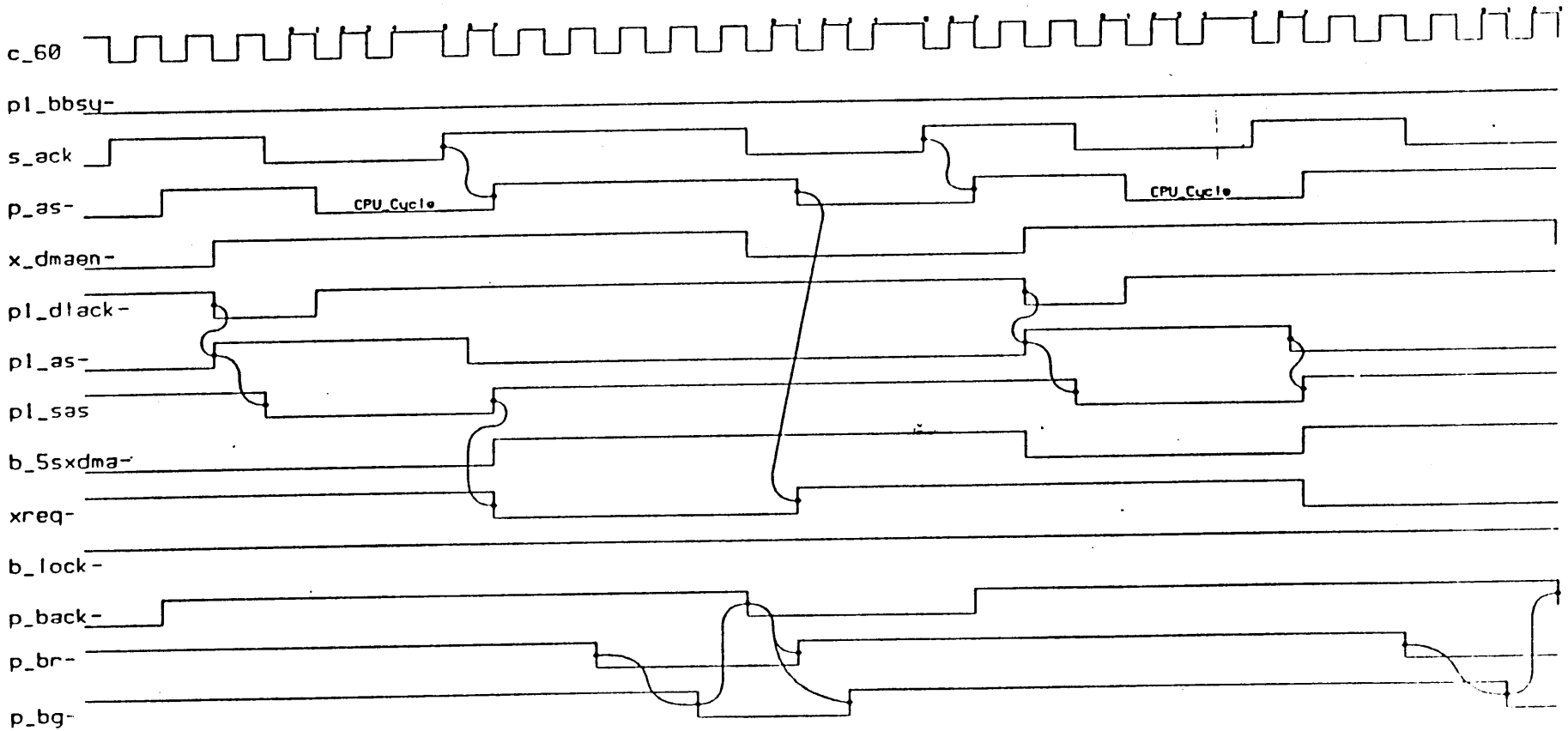


VME_Device_Acquires_VMEbus
and_Accesses_P2_Bus

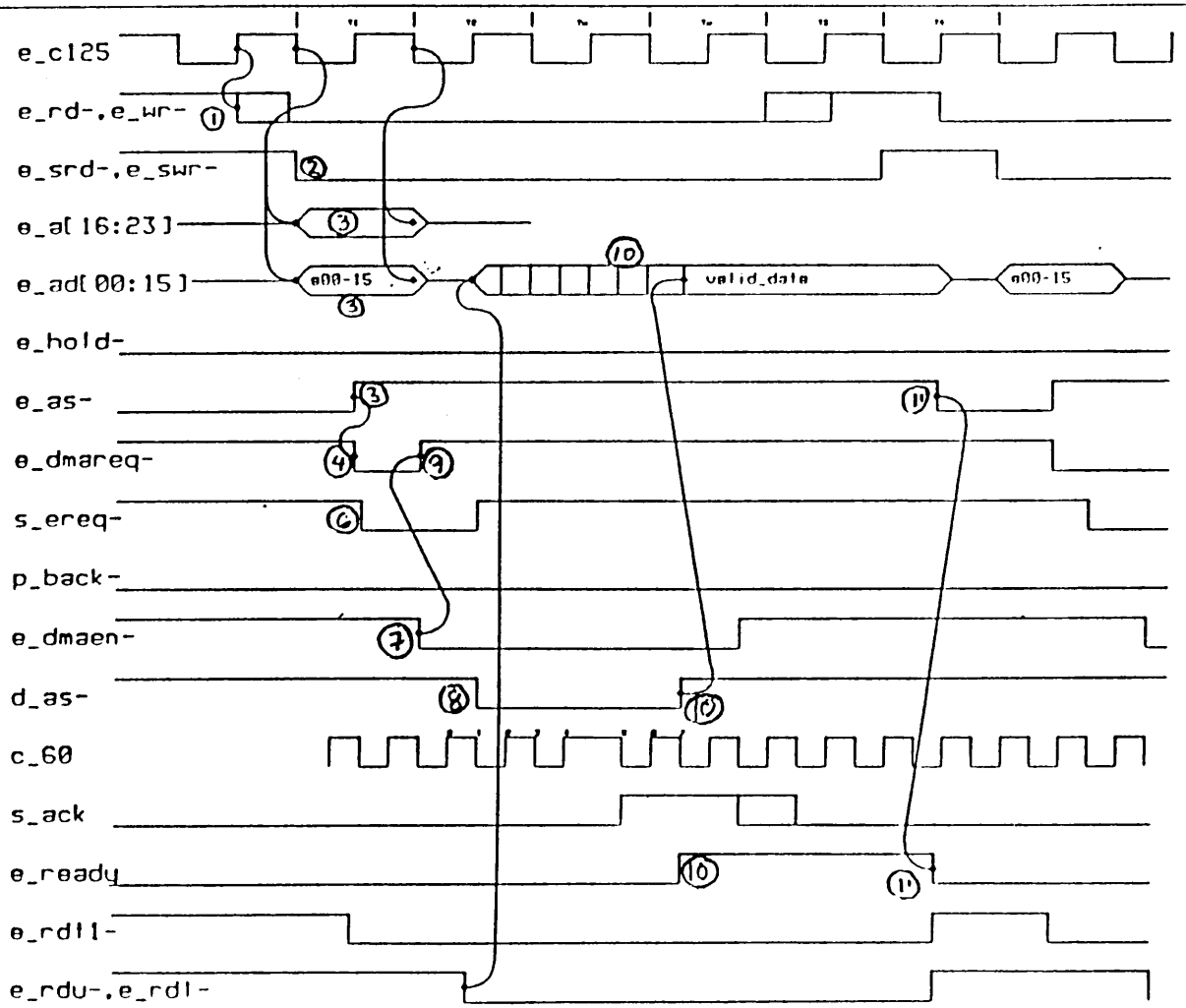

c_60
p1_bbsy-
s_ack
p2_as-
CPU_Cycle
x_dmaen-
p1_dlack-
p1_as-
p1_sas
b_5sxdma
xreq-
b_lock-
s_xreq-
p_back-
p_br-
p_bg-
p1_ds0-,p1_ds1-
p1 ds
b_ ,do-
VME_Device_Initiates_P2_Bus_Lock



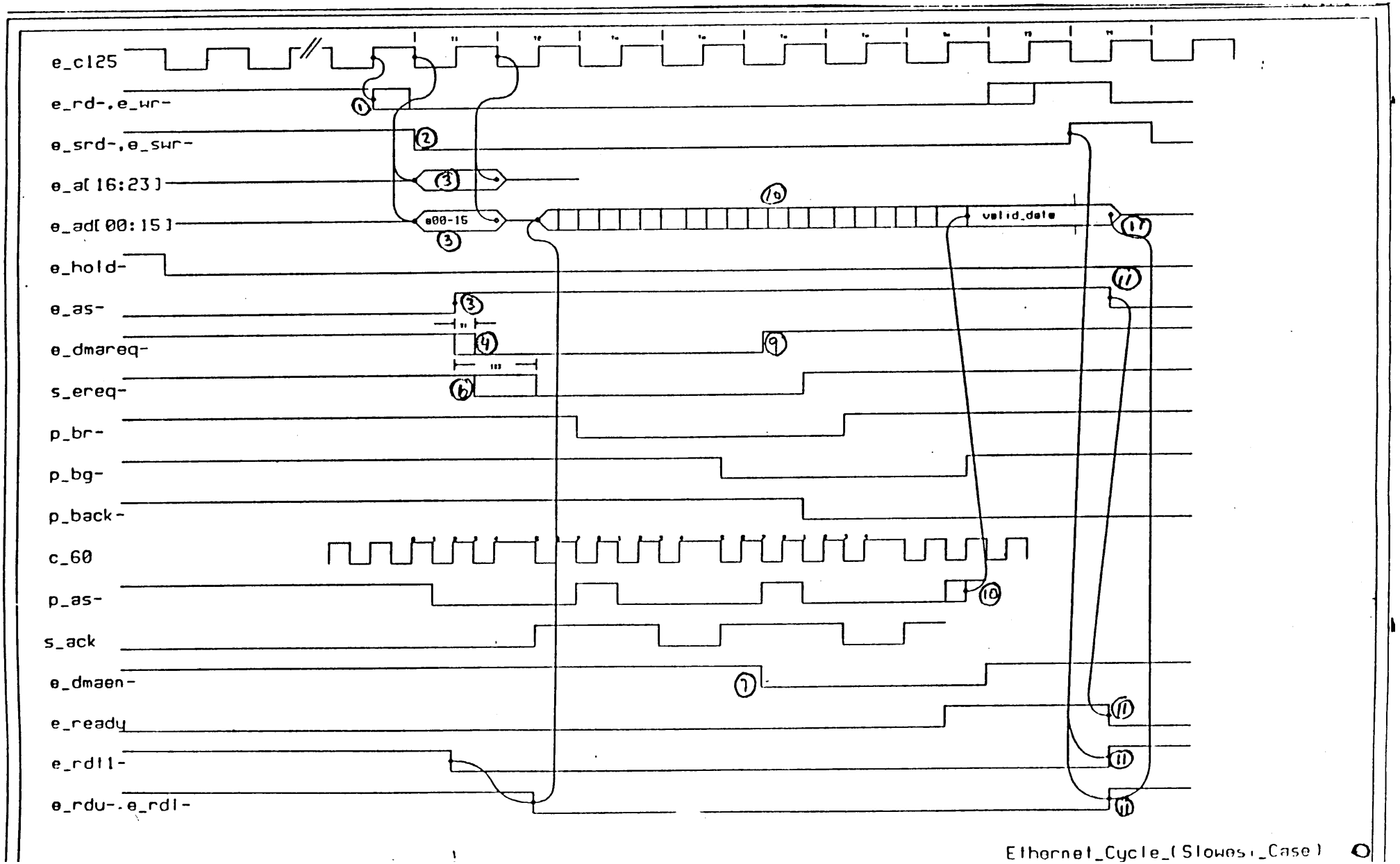
VME_Device_Ends_P2_Bus_Lock

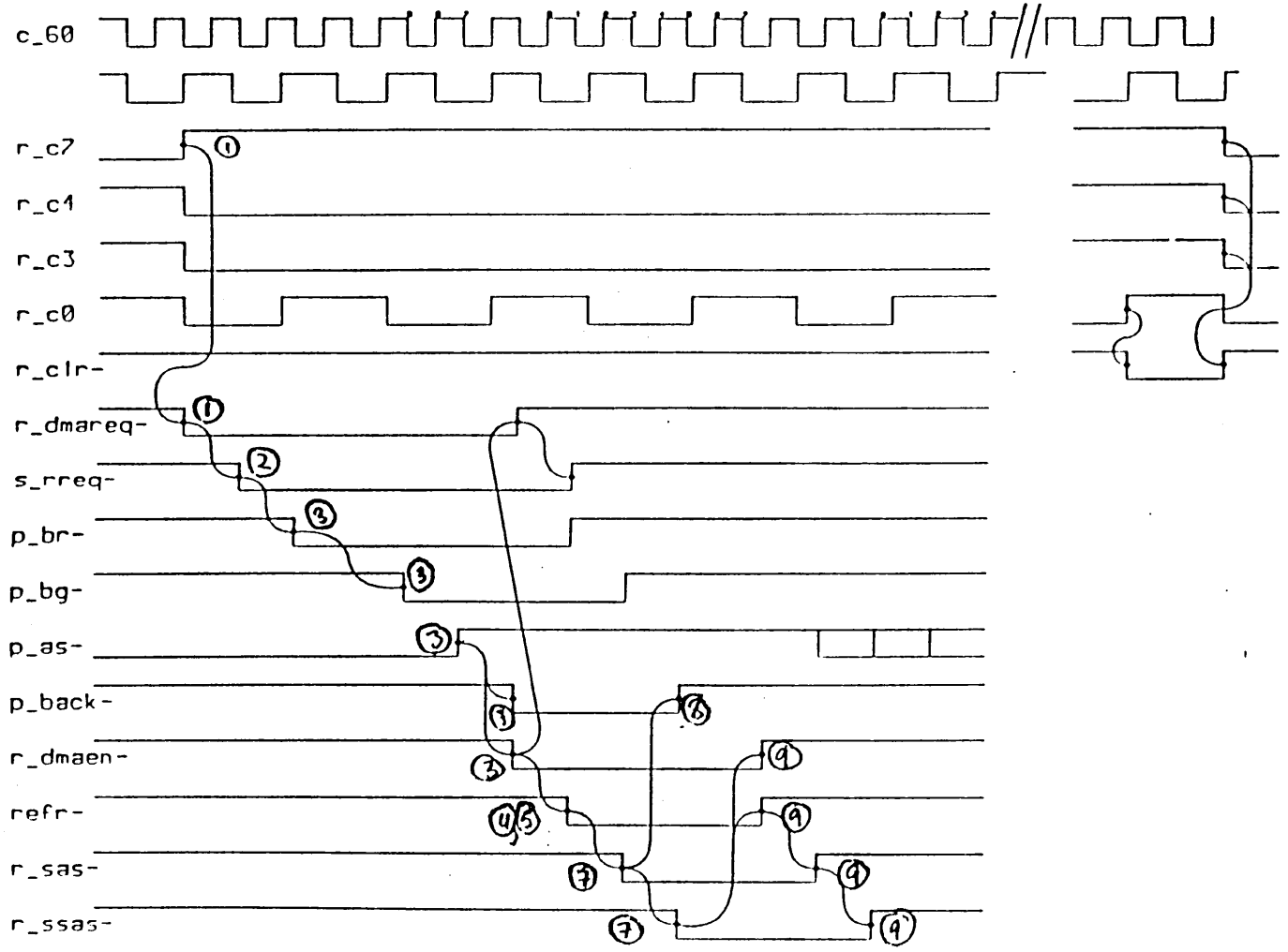


VME_Device_Not_Fast_Enough_to
Initiate_P2_Bus

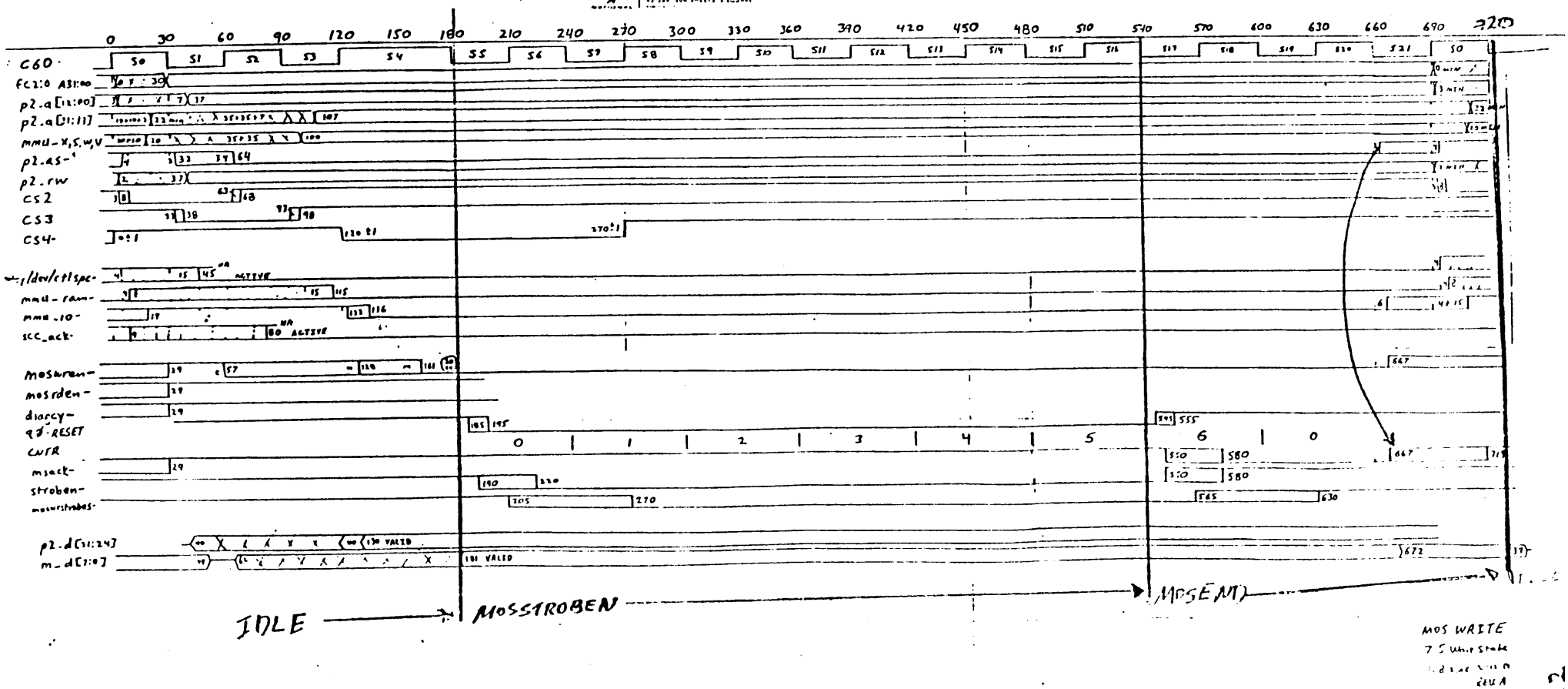


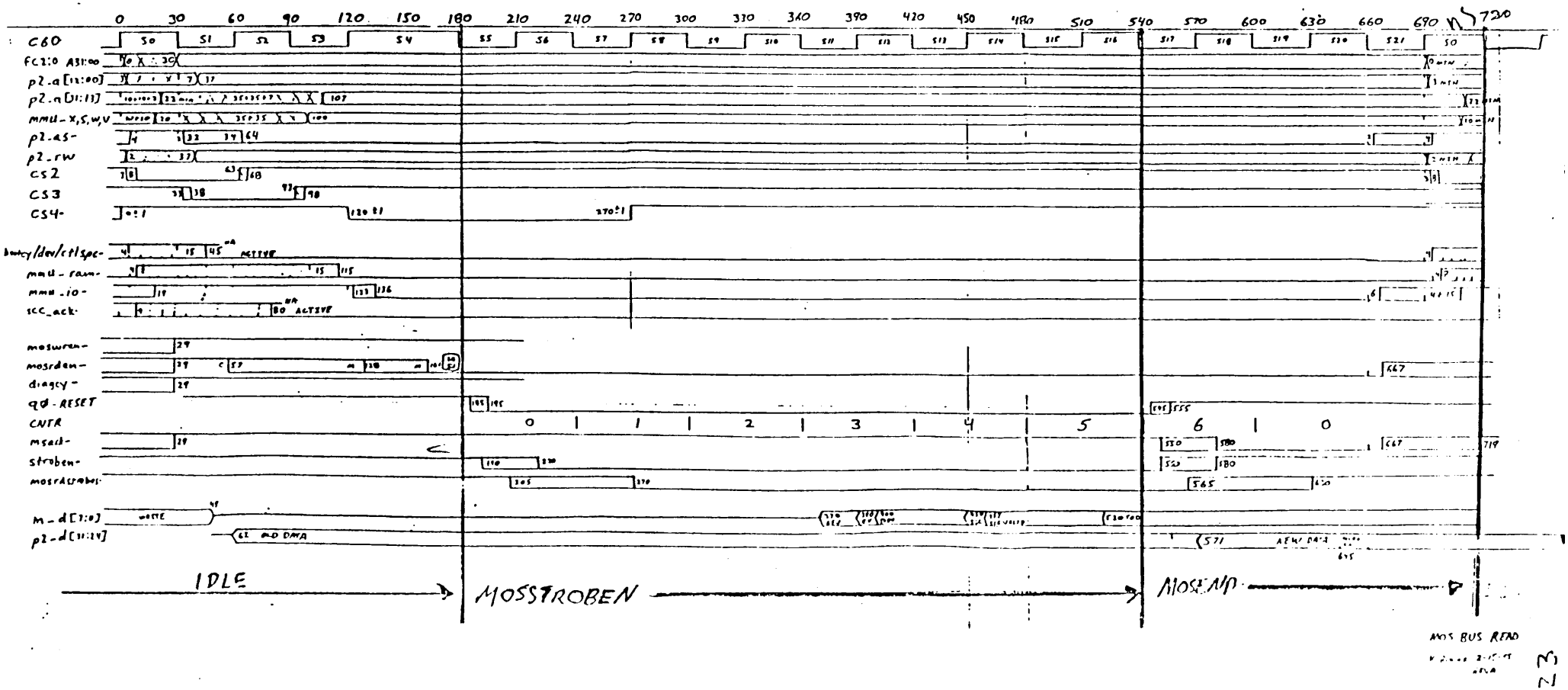
Ethernet_Cycle_(Fastest_Case)

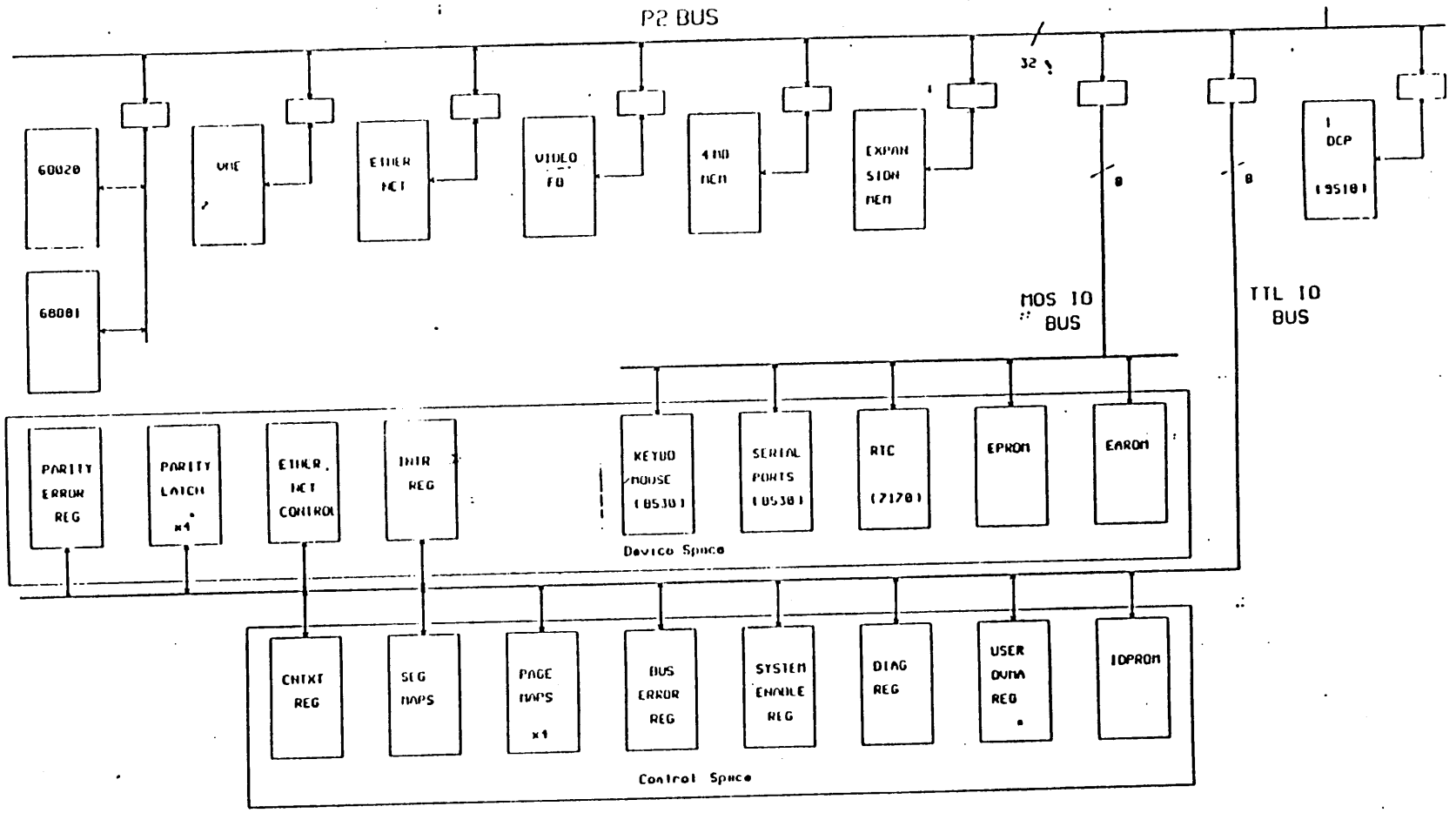

e_c125
e_rd-,e_wr-
e_srd-,e_swr-
e_a[16:23]
e_ad[00:15]
e_hold-
e_as-
e_dmareq-
s_ereq-
p_br-
p_bg-
p_back-
c_60
p_as-
s_ack
e_dmaen-
e_ready
e_rdll-
e_rdu-,e_rdl-
a00-15
valid_data
Ethernet_Cycle_(Slowest_Case)



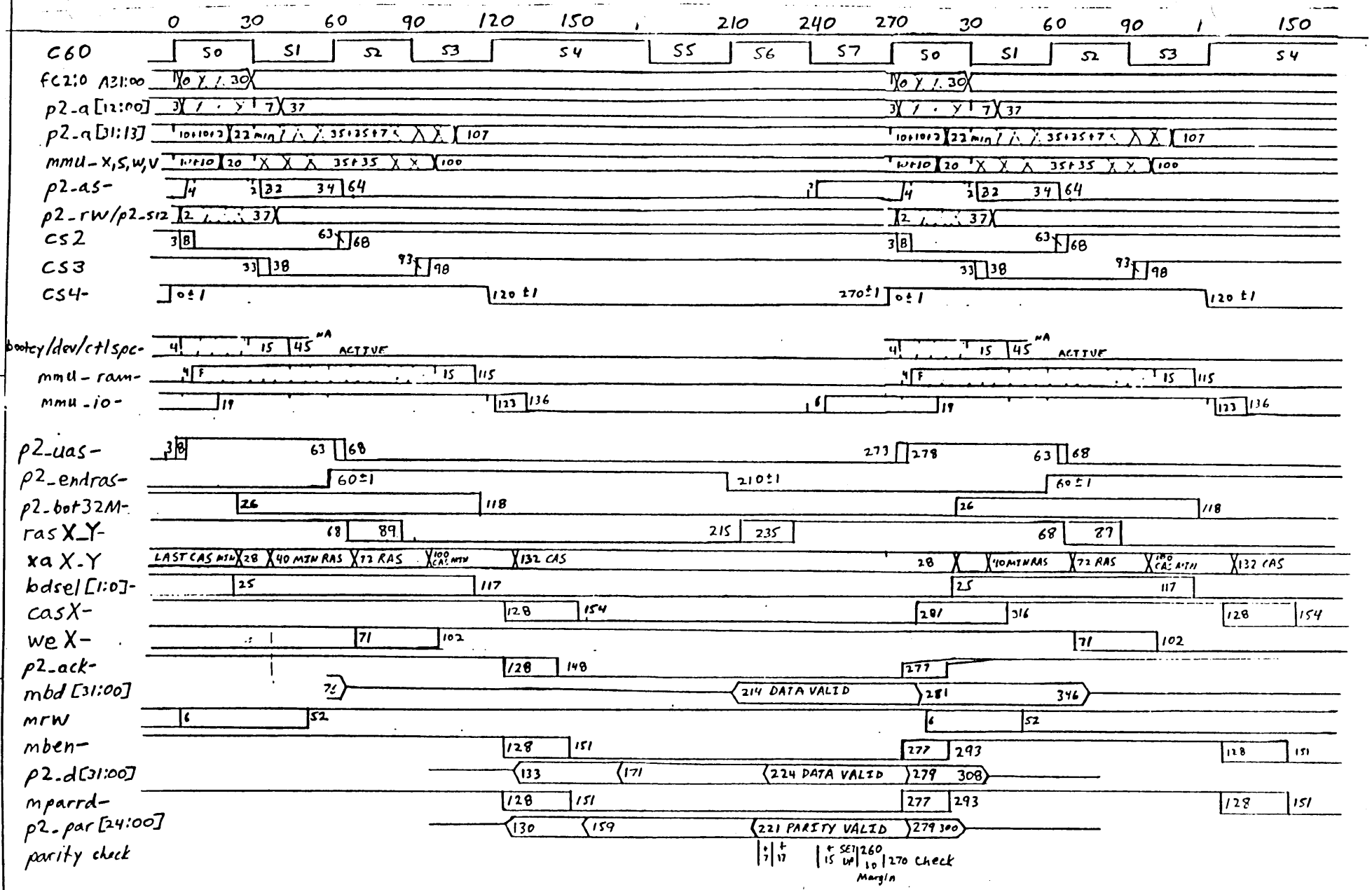
Refresh_Cycle


0 30 60 90 120 150 180 210 240 270 300 330 360 390 420 450 480 510 540 570 600 630 660 690
C60
FC2:0 A31:00
p2.a[15:00]
p2.a[31:11]
mmu-X,S,W,V
p2.as-
p2.rw
cs2
cs3
cs4-
mmu_ram-
mmu_io-
scc_ack-
mosaren-
mosrden-
diaccy-
RESET
CWTR
msack-
stroben-
mosmstrobes-
p2.d[31:24]
m_d[7:0]
IDLE
MOSSTROBEN
MOSEM1
MOS WRITE


C60
FC2:0 A31:00
p2_a [31:00]
p2_d [31:13]
mmu_x,s,w,v
p2_as-
p2_rw
cs2
cs3
cs4-
busCy/dev/ctlspec-
mmu_ram-
mmu_io-
scc_ack-
mosWren-
mosRden-
diagCy-
q0_RESET
CNTR
msack-
strobeN-
mosRAstrobe-
m_d [7:0]
p2_d [31:24]
IDLE
MOSSTROBEN
MOSEAP
MOS BUS READ

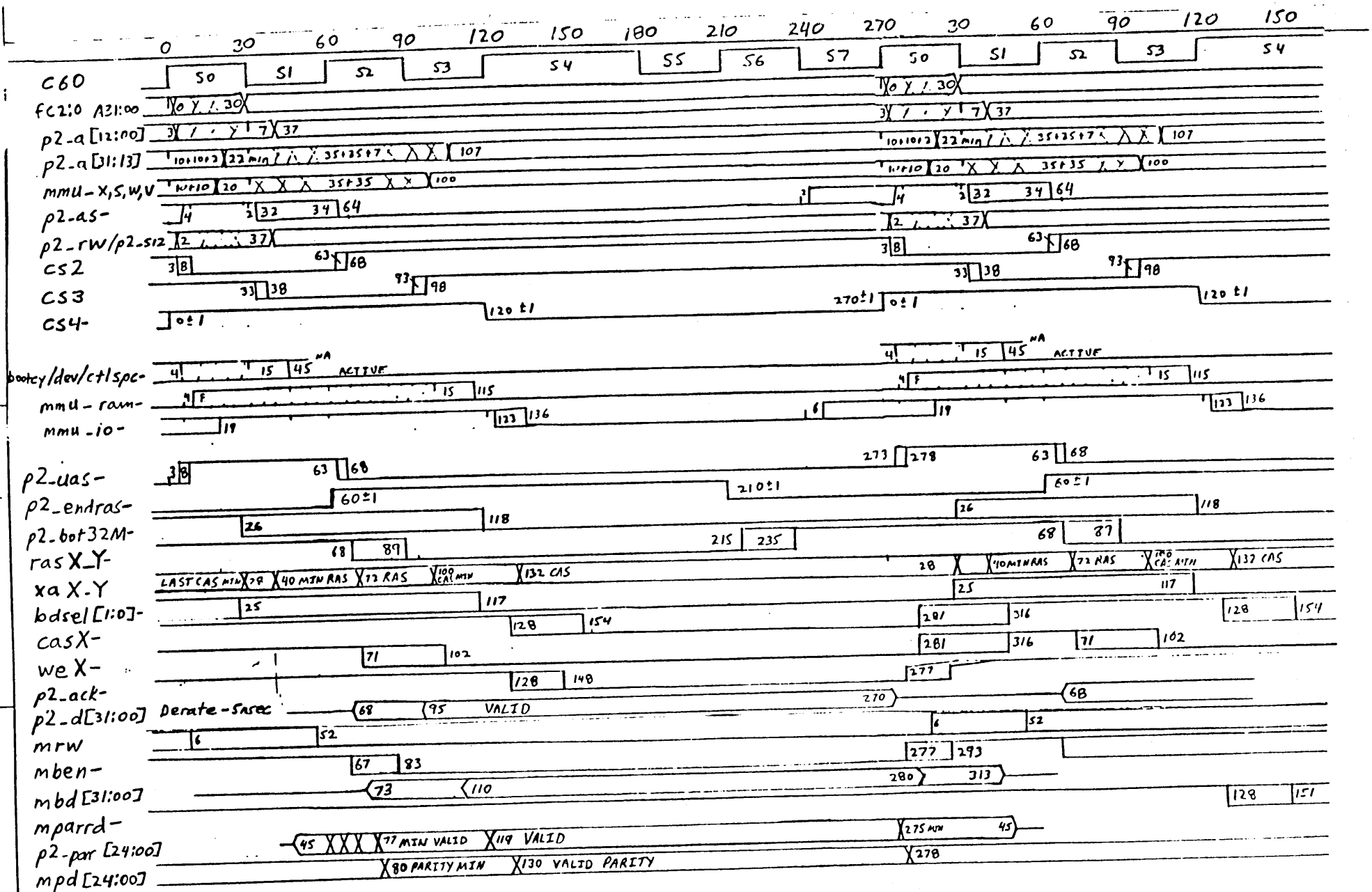


2060 DATA BUSSING



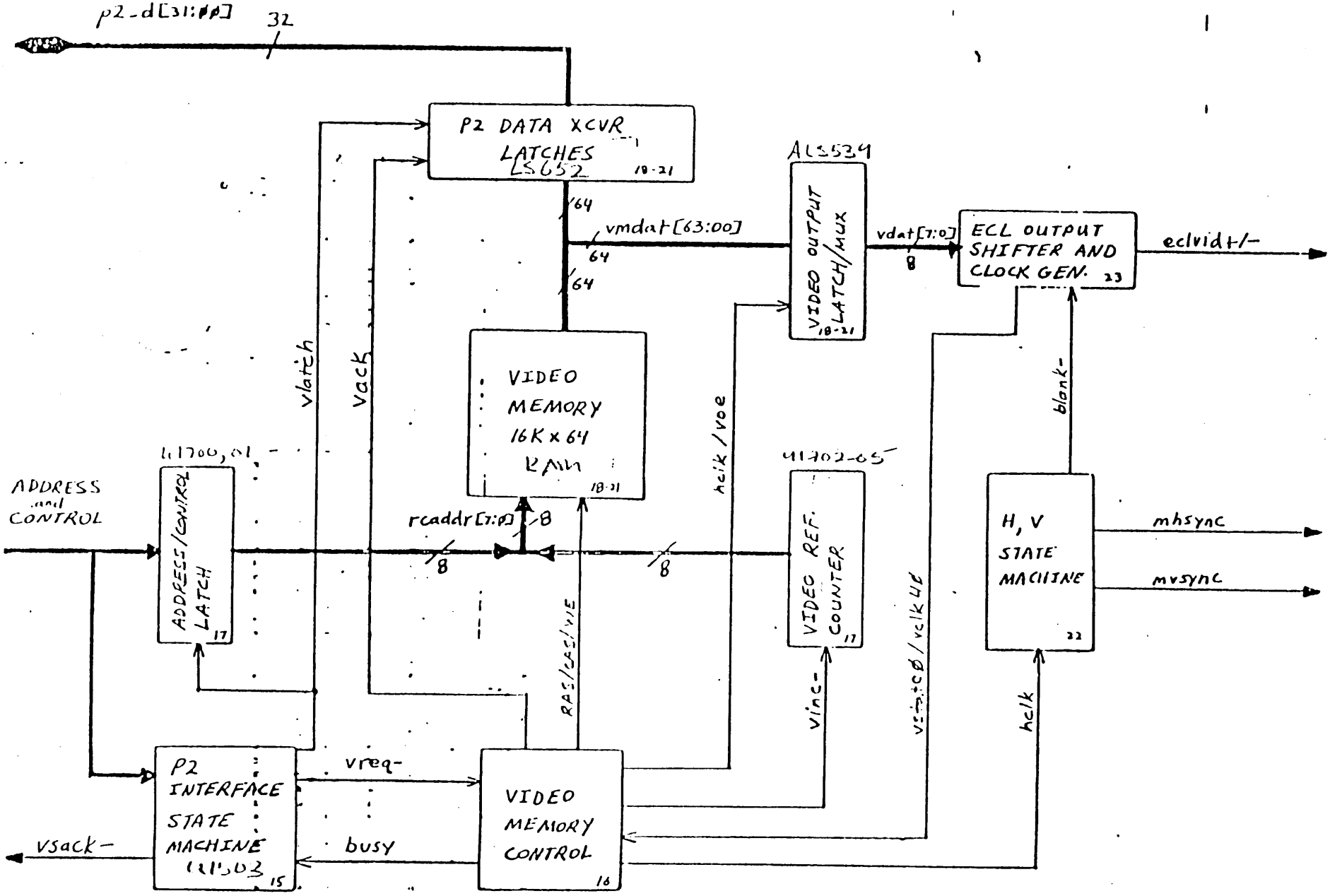
2060 MEMORY READ

9-11-85 K.BIZJAK



2060 MEMORY WRITE

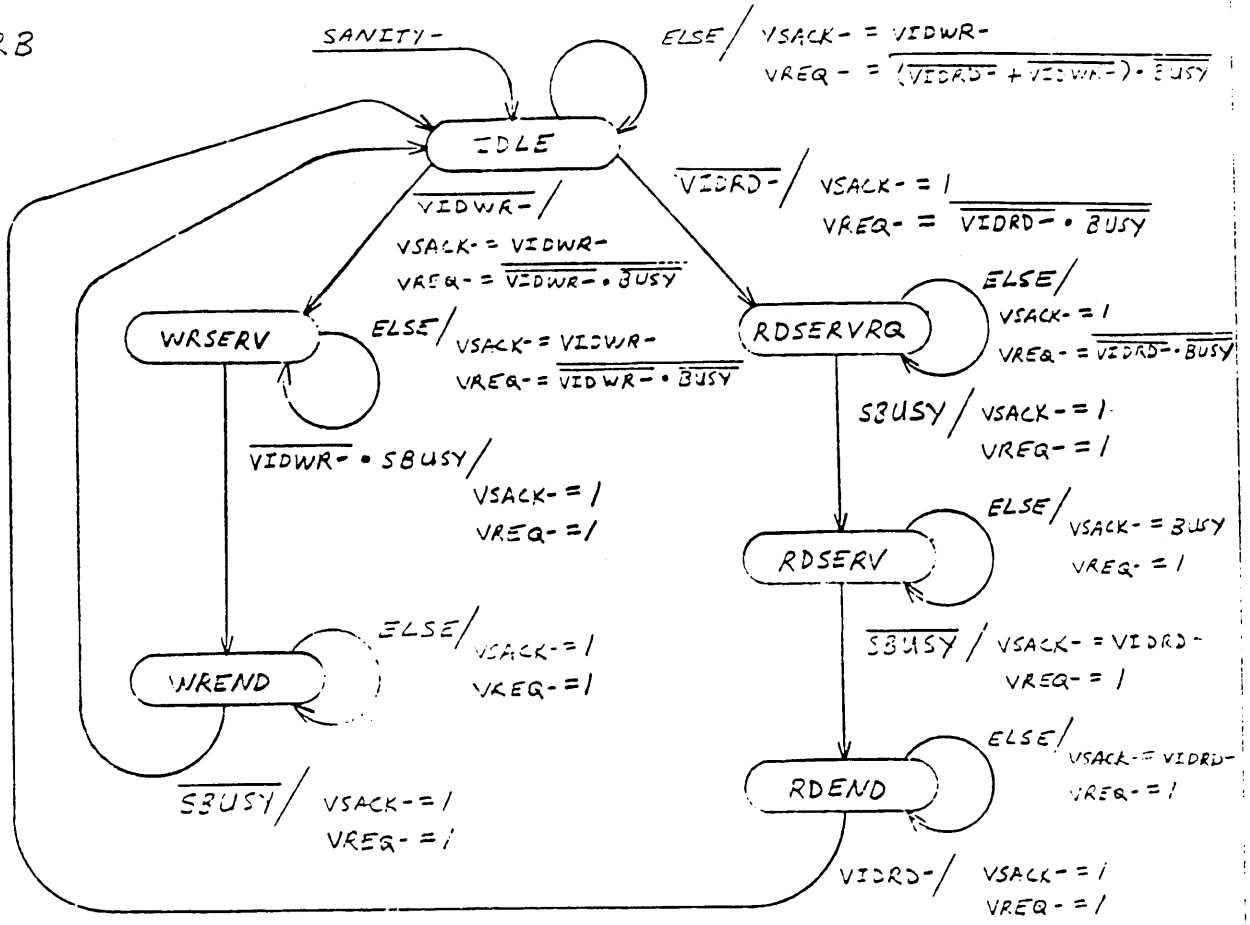
9-11-85 K. BIZJAK



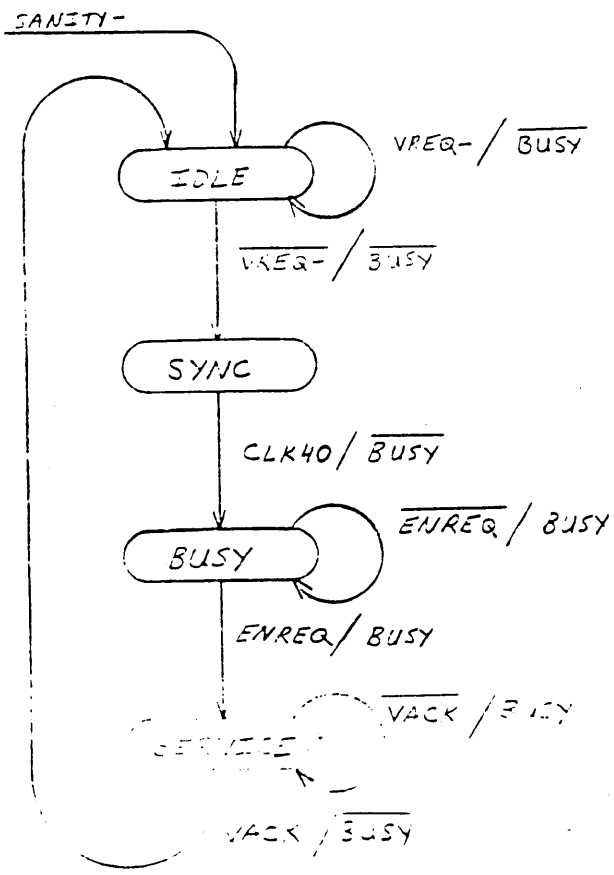
VIDEO BLOCK DIAGRAM

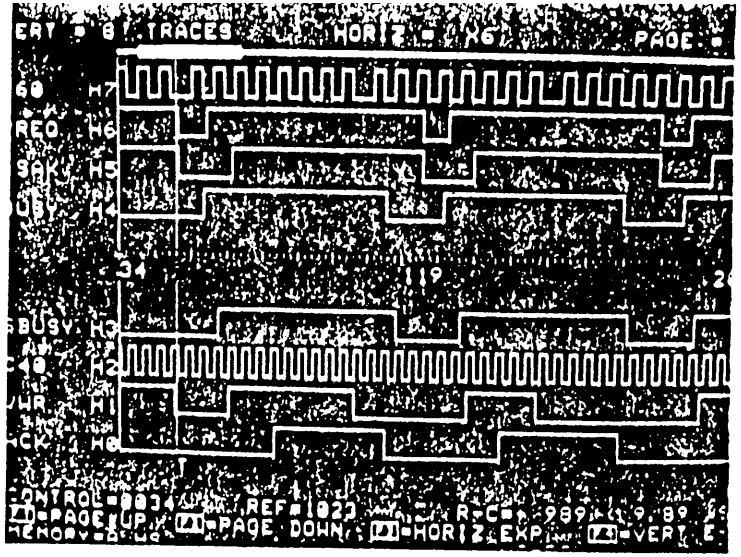
K.BI 7 JAN 3-7-85

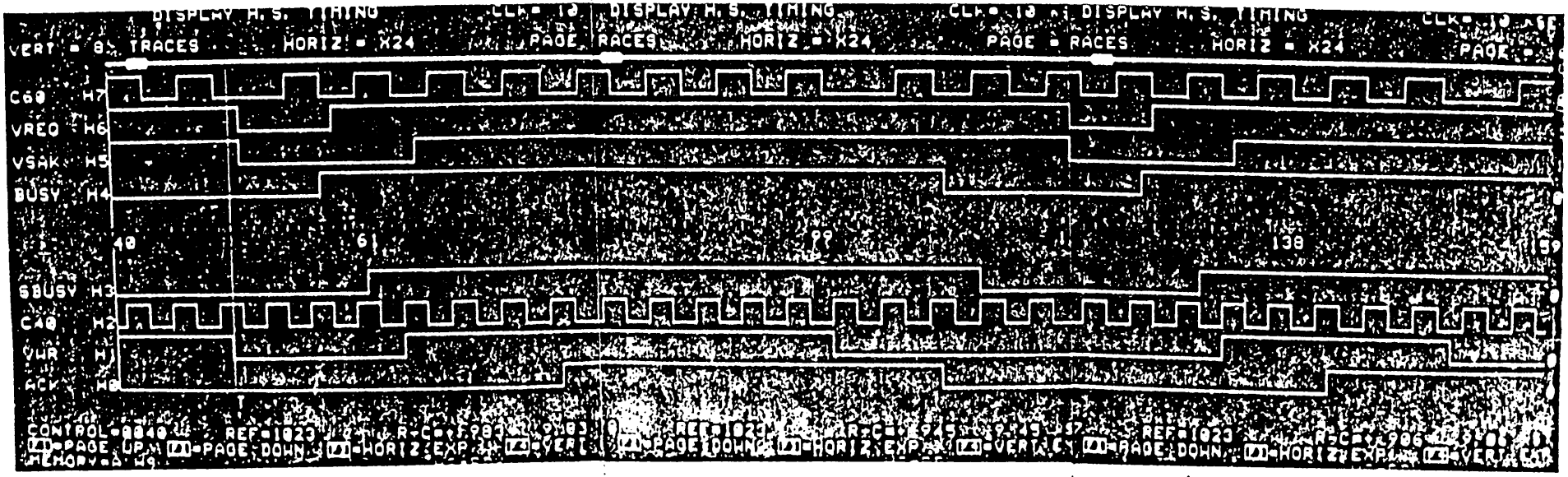
VARB



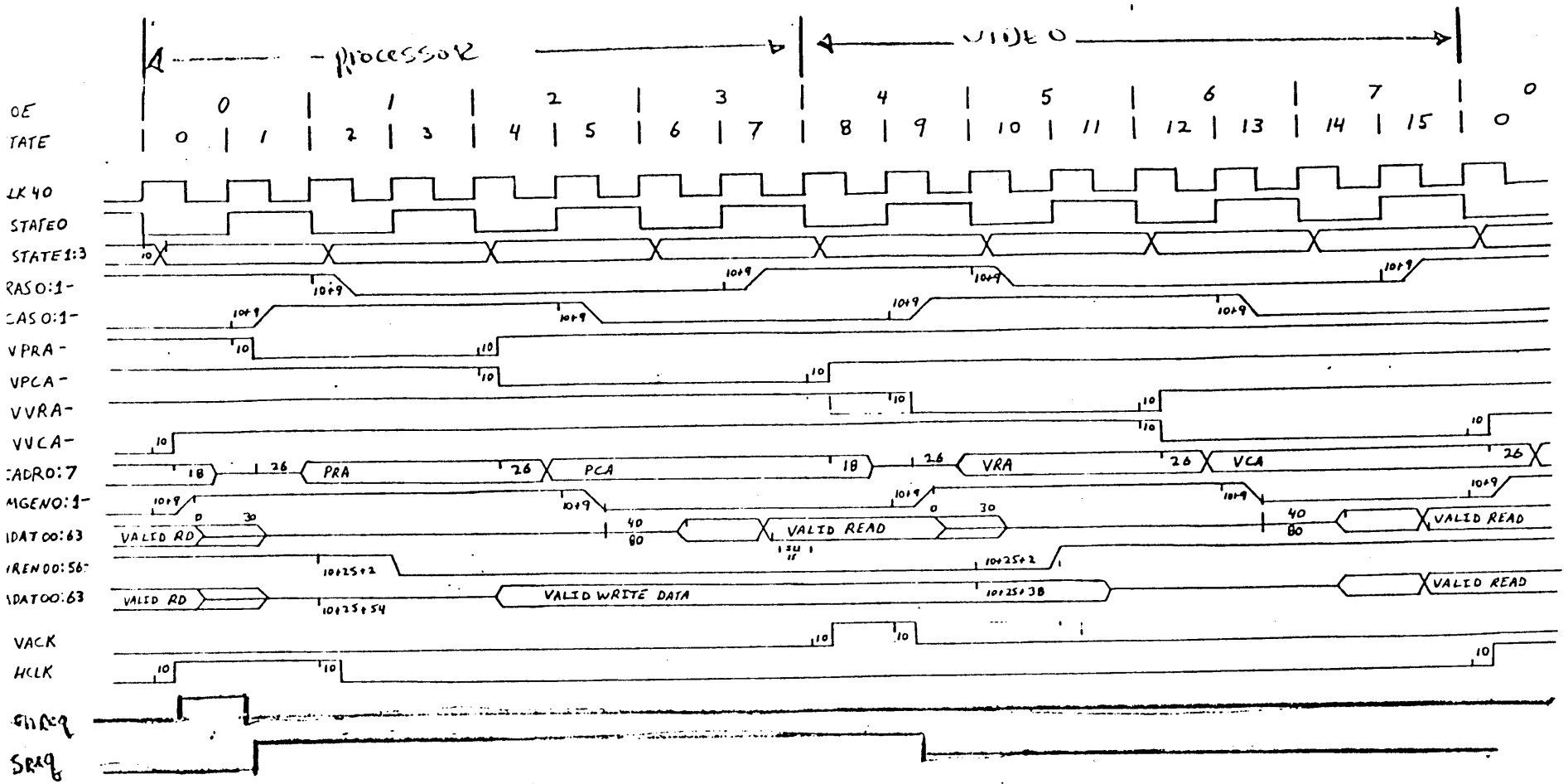
VIDEO SIDE





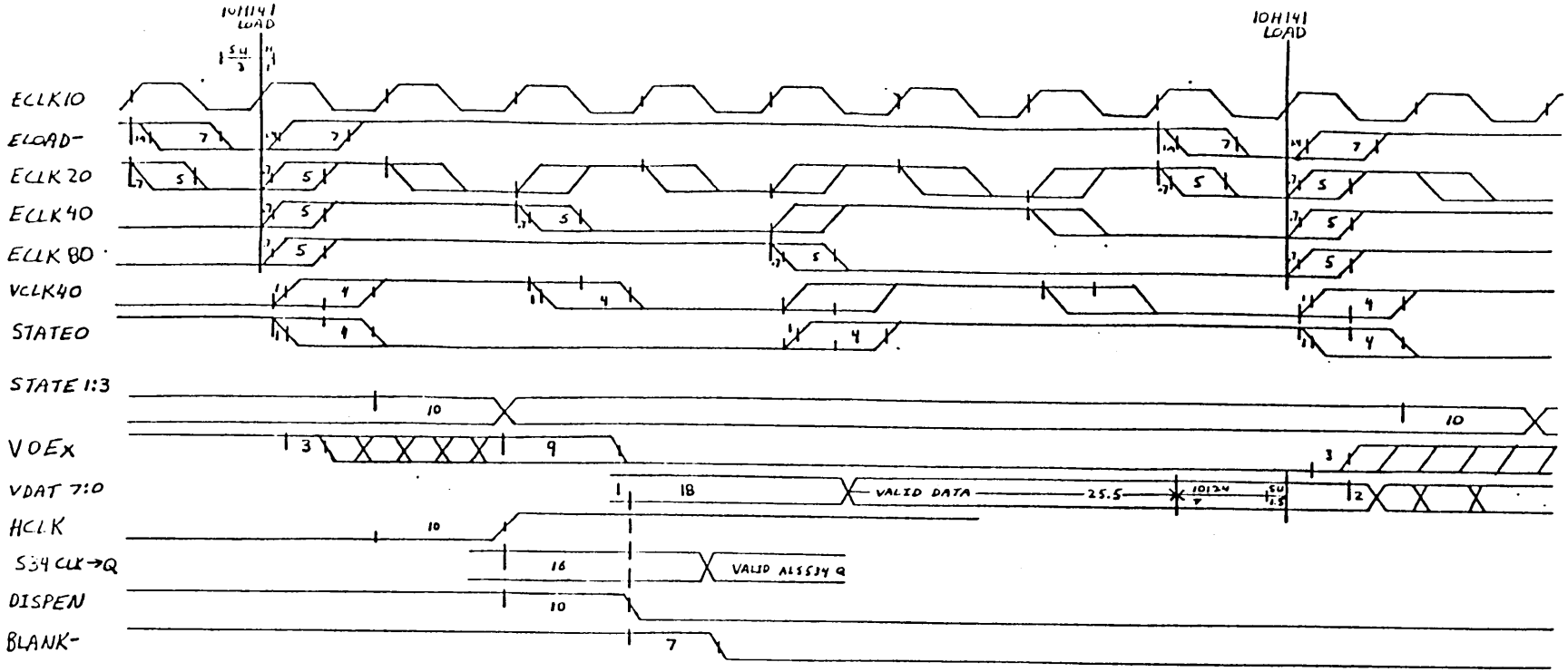


VIDEO WRITE TIMING
(FROM ANALYZER)



2060 FRAME BUFFER CONTROL

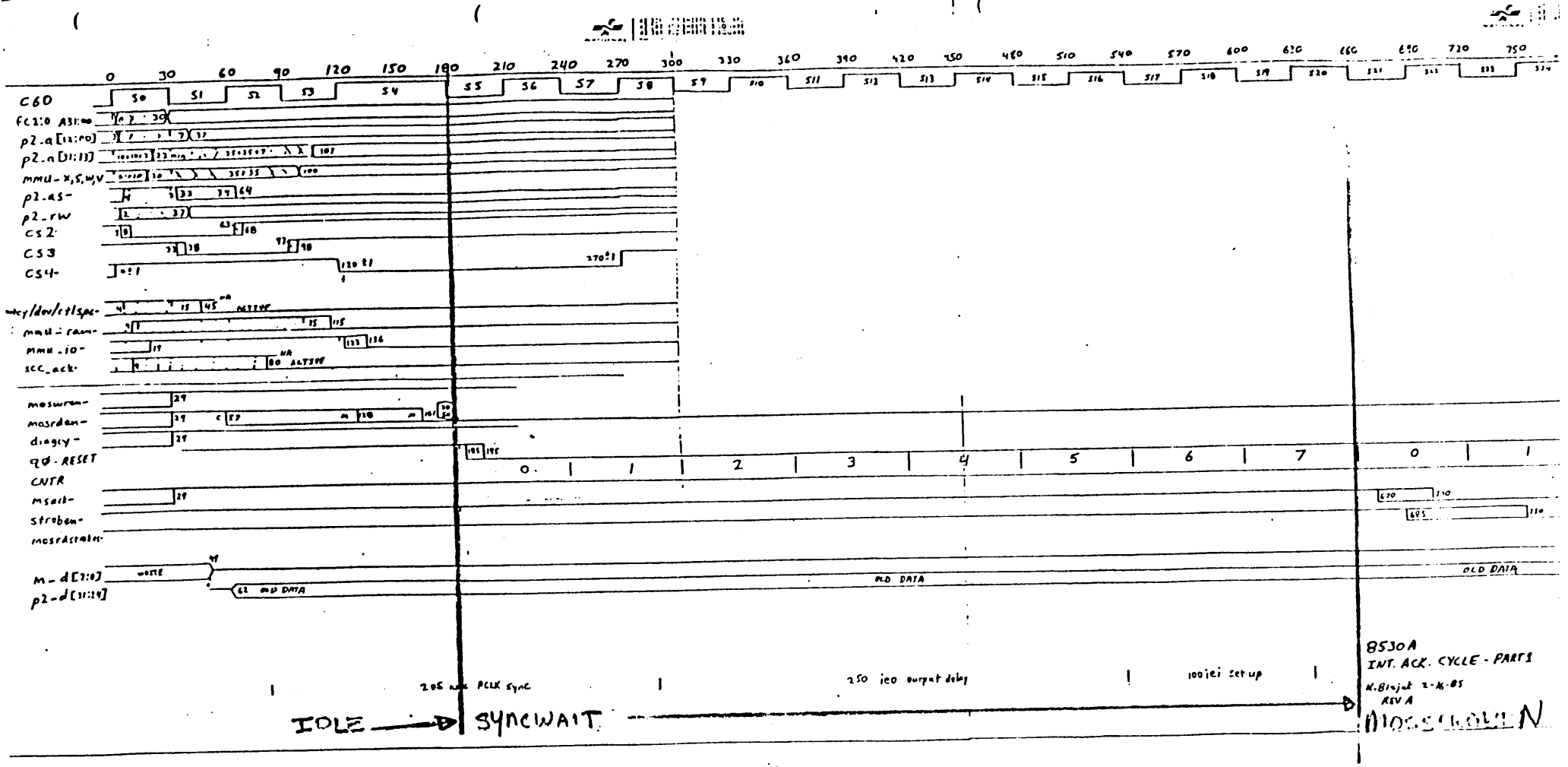
A:3123AK
11-13-54
REV A

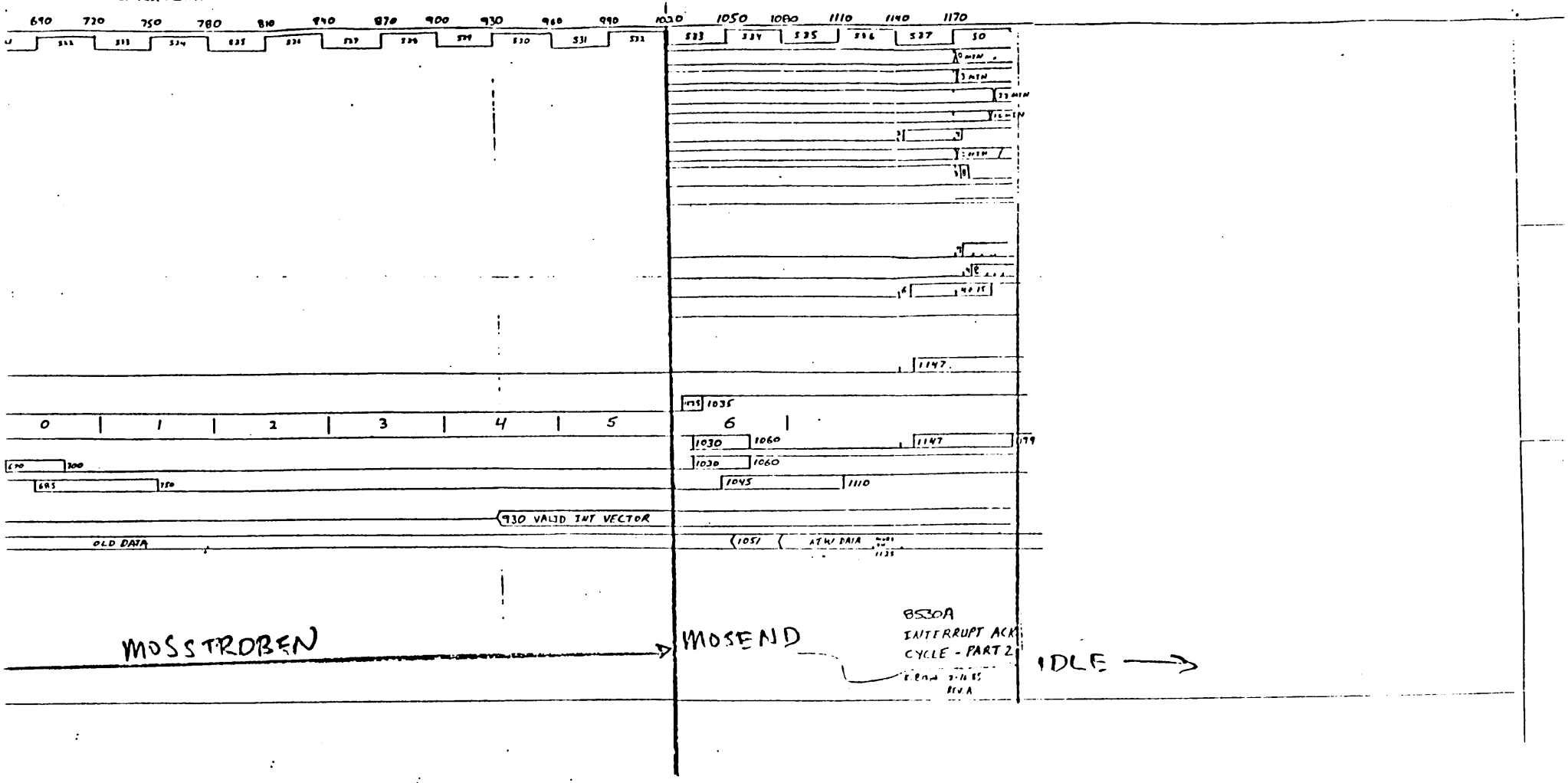


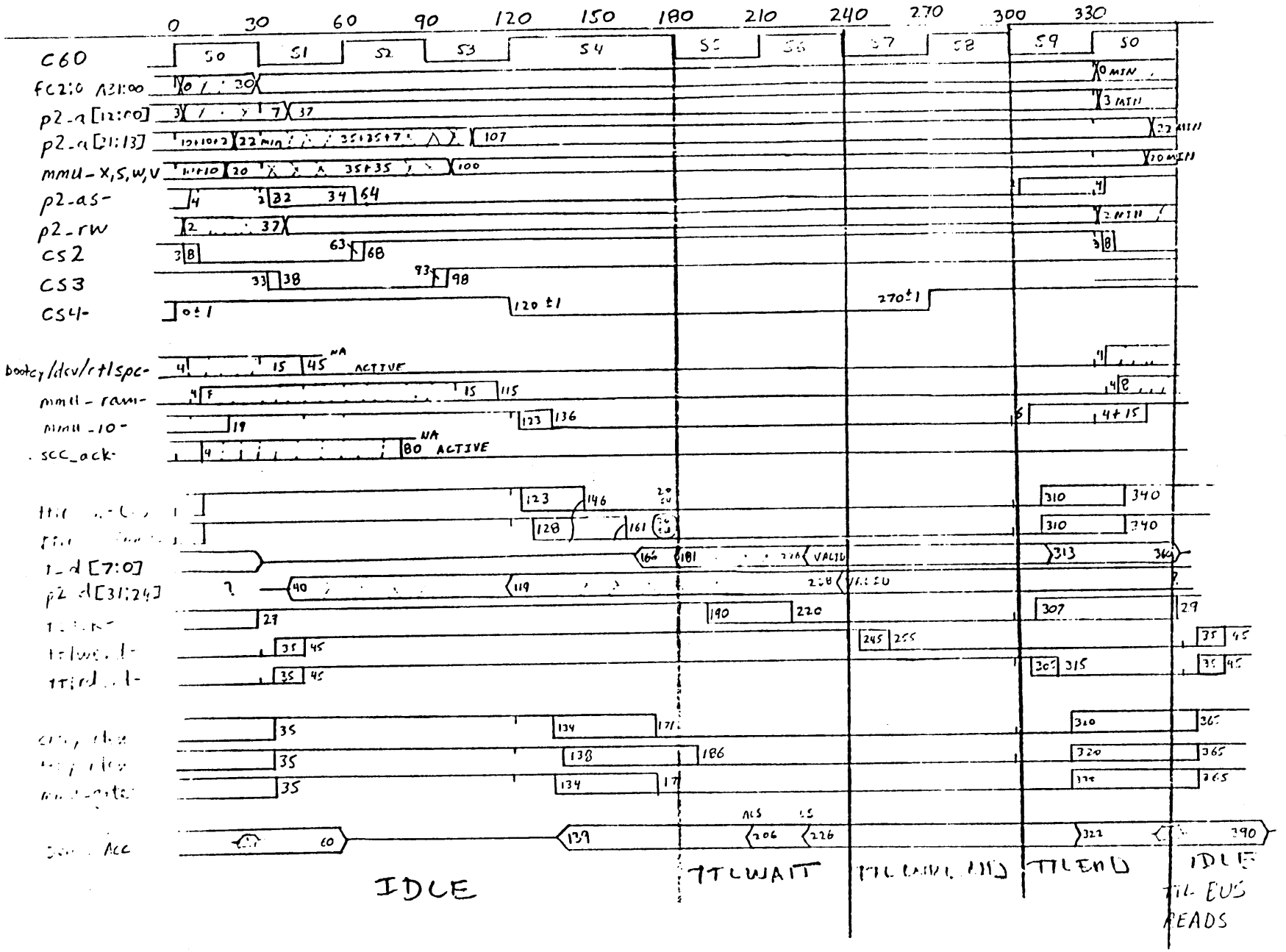
NOTE: ECLK MIN DELAYS BASED ON 10H136
MAX DELAYS BASED ON 10136

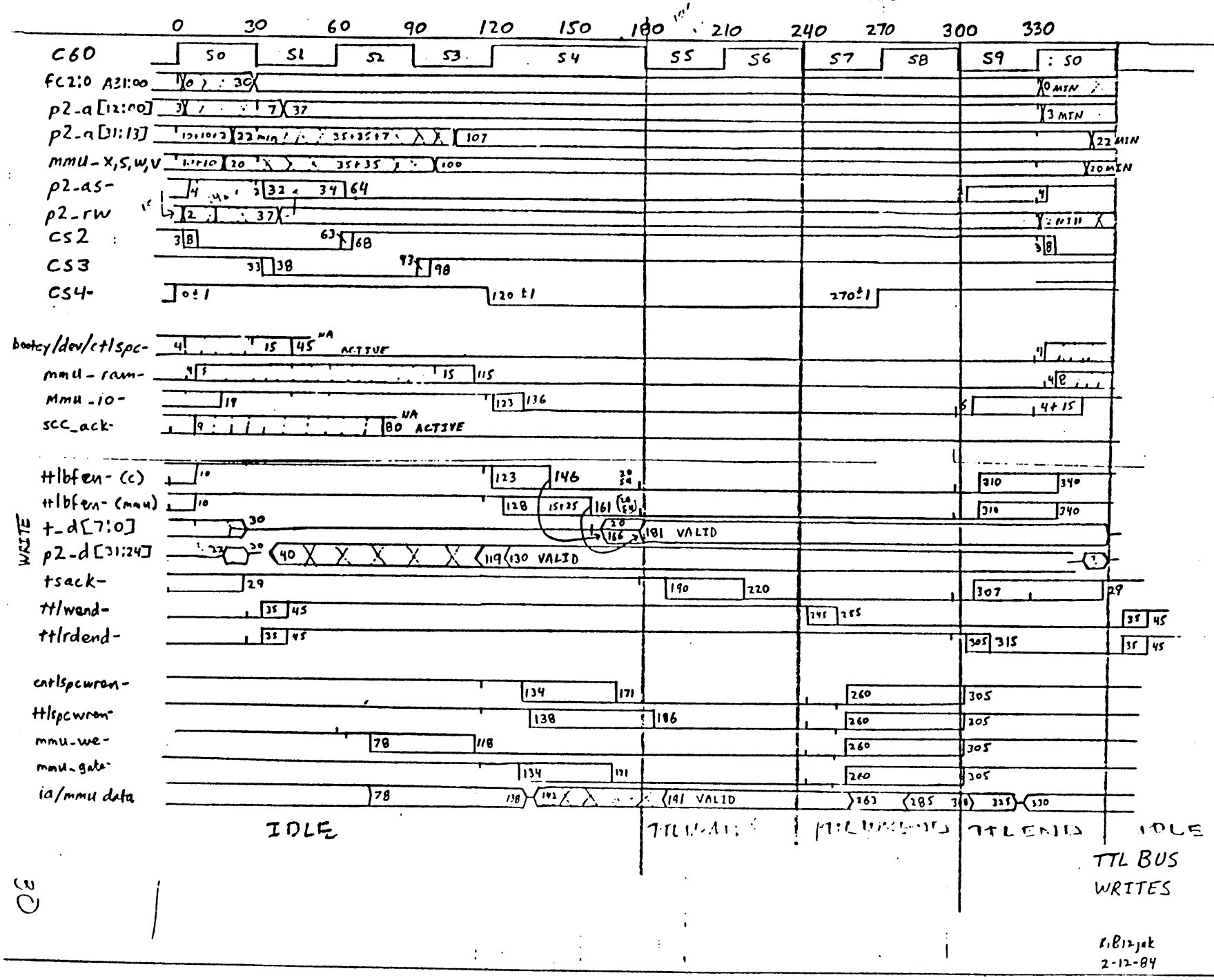
2080 VIDEO OUTPUT SHIFTER,
CLOCK GENERATOR, AND
BLANKING

K.BIZJAK


C60
fc2:0 asi:
p2_a[31:00]
p2_d[31:00]
mmu_x,s,w,v
p2_as-
p2_rw
cs2-
cs3
cs4-
mcy/dev/ctlspc-
mau_ram-
mmu_io-
scc_ack-
mosworn-
mosrden-
diagcy-
Q0 - RESET
CNTR
msack-
strobem-
mosrAstrobe-
m_d[7:0]
p2_d[31:24]
WRITE
OLD DATA
OLD DATA
IDLE
SYNCWAIT
8530A
INT. ACK. CYCLE - PART1
REV A


690 720 750 780 810 840 870 900 930 960 990 1020 1050 1080 1110 1140 1170
0 1 2 3 4 5 6
930 VALID INT VECTOR
OLD DATA
MOSSTROBEN
MOSEND
8530A
INTERRUPT ACK
CYCLE - PART 2
IDLE


0
30
60
90
120
150
180
210
240
270
300
330
C60
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S0
FC2:0 A31:00
p2_a[12:00]
p2_a[31:13]
mmu_x,s,w,v
p2_as-
p2_rw
cs2
cs3
cs4-
0±1
120±1
270±1
boot_cy/dcv/rtlspc-
mmu_ram-
mmu_io-
scc_ack-
ACTIVE
IDLE
TTLWAIT
TTLEND
IDLE
TIL EOS
READS


0 30 60 90 120 150 180 210 240 270 300 330
C60
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 : S0
fc2:0 A31:00
p2_a[12:00]
p2_a[31:13]
mmu_x,s,w,v
p2_as-
p2_rw
cs2
cs3
cs4-
bootcy/dev/ctlspc-
mmu_ram-
mmu_io-
scc_ack-
ttlbfen- (c)
ttlbfen- (mmu)
t_d[7:0]
p2_d[31:24]
tsack-
ttlwend-
ttlrdend-
ctlspcwren-
ttlspcwren-
mmu_we-
mmu_gate-
ia/mmu data
WRITE
IDLE
TTL BUS WRITES
IDLE
R.Bizjak
2-12-84

Revision History

Revision	Date	Comments
50	26 February 1986	First version of this Hardware Engineering Manual
A-10	25 July 1986	Released version of this Hardware Engineering Manual
1-11	1 of 20 October 1986	Changed revision number to comply with Doc Control's new numbering scheme.
1-12	20 January 1987	Corrected revision level to reflect new numbering scheme. This is the only change in the manual.
1-13	10 May 1987	Added changes from the latest ECO; PALs 210, 202, 408; 25 MHz clock removed; TOD circuit revised.