FERMILAB-TM-2848
DAE/0023

Fermilab

FASTBUS Smart Crate Controller

Mark Bernett, Mark Bowden, Gustavo Cancelo, Rick Kwarciany, John Urish

Version PC 3
Manual

HN 96

October 22, 1990

This manuscript has been authored by Fermi Research Alliance, LLC under Contract N o. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

Table of Contents

1 General Information 3
1.1 Purpose 3
1.2 Standard Bus Connections (FASTBUS) 4
1.3 Packaging 4
1.3.1 Module Pinout (Backplane Connections) 4
1.3.2 Front Panel 4
1.4 Power Requirements 7
2 Theory of Operation and Operating Modes 7
2.1 Basic Operation 7
2.2 On-Board Processor 8
2.2.1 Control and Status Registers 9
2.2.2 Error Responses 9
2.2.3 Software 10
2.2.4 Interrupts 10
2.3 Communication Interfaces 11
2.3.1 Addressing Modes (FASTBUS) 11
2.3.1.1 Data Transfer Description and Transfer Rates 16
2.3.1.2 Internal Control and Status Registers 16
2.3.1.3 Error Responses 16
2.3.2 Ethernet 17
2.3.3 Description and Operation of OUTPUT PORT 19
2.3.3.1 OUTPUT PORT Controller Interface 29
2.3.3.2 OUTPUT PORT State Machine 30
2.3.3.3 Total Header and Event Counter Control System 31
2.3.3.4 OUTPUT PORT Auxiliary Connector Interface 33
2.3.4 Communication Protocols 35
3 System Software Description 36
3.1 System Software 36
3.2 Initialization 36
3.3 PROBE Initialization 36
3.4 PROBE Memory Map 37
4 System and Module Diagnostics 38
4.1 Initial Hardware Inspection 38
4.2 Diagnostic Tests 38
4.3 Diagnostic Operating Instructions 39
Appendix A - FPORT Controller Instruction Set 40
Appendix B - FSCC Parts List 73
Appendix C - PLD Equations 75
Appendix D - FSCC Documentation 206
Appendix E - FSCC Front Panel 208
Appendix F - FSCC EPROMs 210

1 General Information

1.1 Purpose

The FSCC is designed as a simple readout controller for low occupancy front-end modules. It performs most basic FASTBUS operations but is not intended to be a "general-purpose" FASTBUS master.

A Motorola 68020 processor is used to control operation of the module and any features which are not time-critical have been allocated to software.

The design goals are as follows;

a) a typical readout time of 1 µsec for a single slave module with a few words of data, including primary address and address release,

b) ability to execute most standard FASTBUS Master operations, and

c) design simplicity such that a working prototype module can be assembled in 6-9 months.

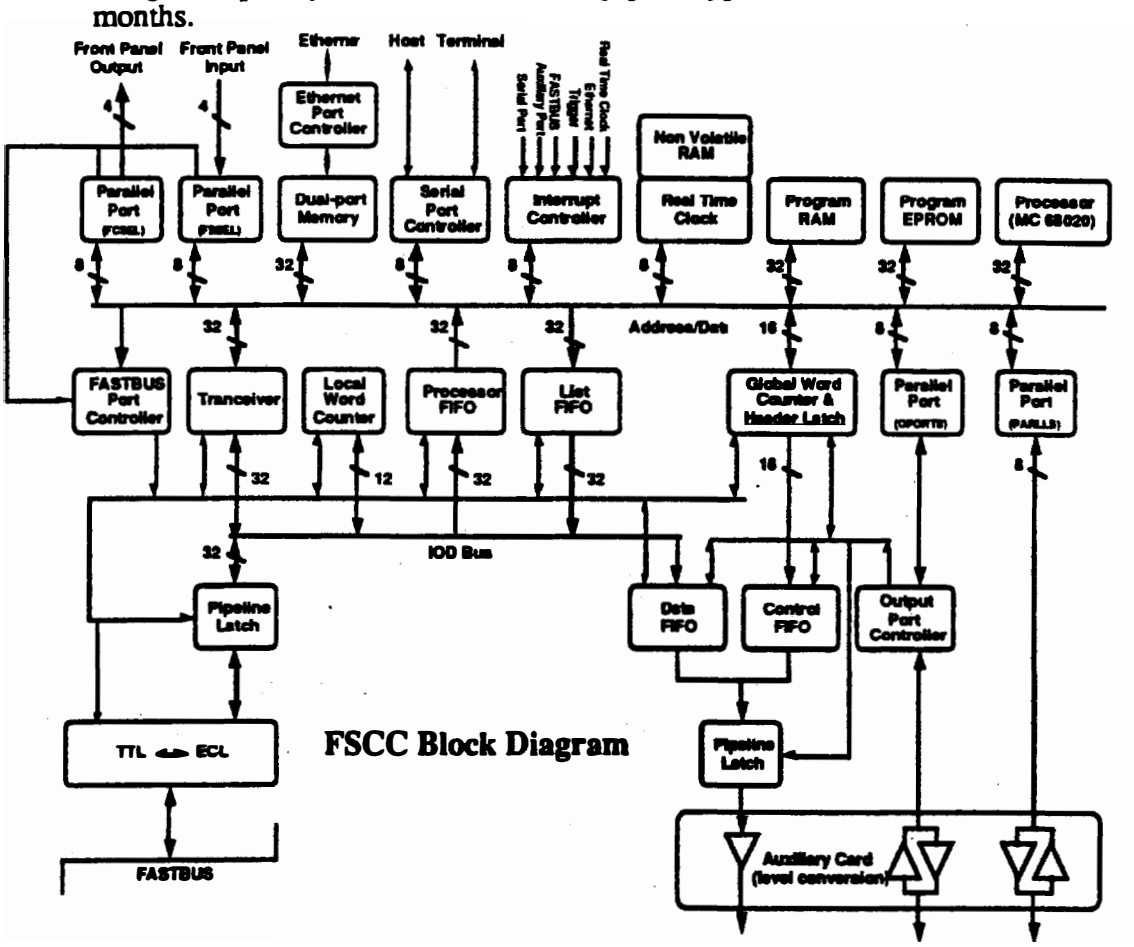


Figure 1

1.2 Standard Bus Connections (FASTBUS)

The FSCC operates as a master on both the FASTBUS crate segment port and the (non-FASTBUS) auxiliary port. It also supports FASTBUS slave operations on the crate segment, but at a very reduced rate.

1.3 Packaging

The FSCC is a single-width FASTBUS module containing approximately 100 integrated circuits. Physical dimensions of the module are per the FASTBUS specification.

1.3.1 Module Pinout (Backplane Connections)

a) FASTBUS Crate Segment (130 pin FASTBUS standard connector) Refer to the FASTBUS specification.

b) Auxiliary Port (195 pin FASTBUS standard 3 row connector), see Figure 2

1.3.2 Front Panel

FASTBUS Master	Green LED
FASTBUS Slave	Yellow LED
Reset Pushbutton	Hard processor reset.
Remote Reset	Hard processor reset. This is an active low TTL input. Shorting the connetor or applying a TTL low will cause a reset. This input may be Daisy-Chained.
Trigger Input	The Trigger Input consists of five differential RS 485 pairs. A 4 bit Trigger ID is latched on the leading edge of the Trigger Input Strobe. This value can be read by the processor through a parallel port. The Trigger Input Strobe generates a processor interrupt if enabled. The processor should clear this interrupt to allow further trigger recognition. State of the Trigger Strobe signal is also available to the FASTBUS Port Controller for fast triggering. The Trigger ID should be valid for at least 60 nsec prior to the leading edge of the Trigger Input Strobe and remain valid for at least 20 nsec after the leading edge of the strobe. The Trigger Strobe pulse width should be at least 150 ns to ensure recognition by the FASTBUS Port Controller.

Trigger Input (20 pin IDC)

Pin#	
1-10	Reserved
11	Trigger Strobe +
12	Trigger Strobe -
13	Trigger ID0 +
14	Trigger ID0 -
15	Trigger ID1 +
16	Trigger ID1 -
17	Trigger ID2 +
18	Trigger ID2 -
19	Trigger ID3 +
20	Trigger ID3 -

Front Panel Output: The Front Panel Output (20 pin IDC) consists of four latched differential RS485 pairs, which are driven by the processor via a parallel port. These are user defined.

Pin#	
1	Front Panel 0 +
2	Front Panel 0 -
3	Front Panel 1 +
4	Front Panel 1 -
5	Front Panel 2 +
6	Front Panel 2 -
7	Front Panel 3 +
8	Front Panel 3 -
9-20	Reserved

2 Serial Ports: RS232 signal levels (4 pin LEMO connectors) one for Host connection and one for Terminal connection.

Terminal port	Pin #	
	1	TXD (transmit data)
	2	(no connection)
	3	RXD (receive data)
	4	Ground
Host port (Null Modem)	Pin #	
	1	RXD (receive data)
	2	CTS (+10V Reference)
	3	TXD (transmit data)RXD (receive data)
	4	Ground

Ethernet Port: Cheapernet signal levels, Isolated BNC Connector.

Permit_In/Permit_Out: Serial "daisy-chain" signals for FSCC auxiliary port bussing, LEMO connectors.

Auxilliary Port Connector

Pin	Signal	Function	
B65	VTT	-2.0 Volts	
B64	GND	Power Return	
B63	AC15	NC	
B62	AC14	NC	
B61	AC13	Auxiliary Control Bit	13
B60	AC12	Auxiliary Control Bit	12
B59	AC11	Auxiliary Control Bit	11
B58	AC10	Auxiliary Control Bit	10
B47	D31	Output Data Bus Bit	31
B46	D30	Output Data Bus Bit	30
B45	D29	Output Data Bus Bit	29
B44	D28	Output Data Bus Bit	28
B43	D27	Output Data Bus Bit	27
B42	D26	Output Data Bus Bit	26
B41	D25	Output Data Bus Bit	25
B40	D24	Output Data Bus Bit	24
B39	D23	Output Data Bus Bit	23
B38	D22	Output Data Bus Bit	22
B37	D21	Output Data Bus Bit	21
B36	D20	Output Data Bus Bit	20
B35	D19	Output Data Bus Bit	19
B34	D18	Output Data Bus Bit	18
B33	D17	Output Data Bus Bit	17
B32	D16	Output Data Bus Bit	16
B31	D15	Output Data Bus Bit	15
B30	D14	Output Data Bus Bit	14
B29	D13	Output Data Bus Bit	13
B28	D12	Output Data Bus Bit	12
B27	D11	Output Data Bus Bit	11
B26	D10	Output Data Bus Bit	10
B25	D09	Output Data Bus Bit	9
B24	D08	Output Data Bus Bit	8
B23	D07	Output Data Bus Bit	7
B22	D06	Output Data Bus Bit	6
B21	D05	Output Data Bus Bit	5
B20	D04	Output Data Bus Bit	4
B19	D03	Output Data Bus Bit	3
B18	D02	Output Data Bus Bit	2
B17	D01	Output Data Bus Bit	1
B16	D00	Output Data Bus Bit	0
B15	AC09	Auxiliary Control Bit	9
B14	AC08	Auxiliary Control Bit	8
B13	AC07	Auxiliary Control Bit	7
B12	AC06	Auxiliary Control Bit	6
B11	AC05	Auxiliary Control Bit	5
B8	AC04	Auxiliary Control Bit	4
B7	AC03	Auxiliary Control Bit	3
B6	AC02	Auxiliary Control Bit	2
B5	AC01	Auxiliary Control Bit	1
B4	AC00	Auxiliary Control Bit	0
A12,A32,C63	VEE	-5.2 Volts	
A43,C53,C12	VCC	+5.2 Volts	
A20,A22,A53,A63,C22,C32,C43,C44		Power Return	

Figure 2

1.4 Power Requirements

This module dissipates approximately 40 watts.

+5.0 volts	6 amps
-5.2 volts	2 amp
-2.0 volts	0.2 amps

2 Theory of Operation and Operating Modes

Figure 1 shows a general block diagram of the FSCC. The individual blocks are explained in detail below.

2.1 Basic Operation

All module operations are controlled by memory-mapped instructions from the **68020 Processor**. Processor operation is described in section 2.2.

Low level management of the various ports is handled by dedicated controllers. In the case of the **Serial** and **Ethernet Ports**, the controllers are commercial integrated circuits. Controllers for the **FASTBUS Port** (FPORT) and Auxiliary **Output Port** (OPORT) are implemented using PLD state machines.

To maximize throughput, high speed block transfer data from FASTBUS to the Auxiliary Output Port is routed through the **Data FIFO** which serves two functions; it 1) decouples the input and output data rates, and 2) provides buffering of one full crate-event of front-end data for insertion of leading word counts on output. The buffer is implemented using commonly available integrated circuit FIFO's which provide a depth of 2K, 32-bit words. This is expandable by substitution of pin-compatible higher density parts.

A 256X32-bit **Processor FIFO** connects the FASTBUS port to the Processor. Input to this FIFO can be enabled or disabled by the Processor. When enabled, any data which is written to the Data FIFO will simultaneously be written to the Processor FIFO. The Processor FIFO provides a high-speed path through which the processor can sample data in the event stream.

A 256X32-bit **List FIFO** also connects the Processor to the FASTBUS data port. This FIFO is written by the Processor and contains a data list for use by application-specific FPORT Controller microcode.

To further improve the readout speed of the FSCC, a **Local Word Counter** is implemented. In a typical application, this counter is loaded directly from the first word of a Block or Pipelined transfer, with a fixed word count position (bits 0-11). The counter can also be preset by the processor for use with slave modules which do not supply a leading word count. It is implemented in PLD's and can be adapted for inclusive or non-inclusive word counts.

For slave modules capable of producing leading word counts, the local counter eliminates the extra delay the FPORT Controller normally incurs in checking for the SS=2 "end-of-block" condition. It allows better pipelining of the data since the controller will not be required to read beyond end-of-block and then back-up it's internal counters and pipeline registers. The Local Word Counter can also be used to produce intermediate word counts, at user-selected boundaries for insertion in the main data stream. Maximum word count is 4095.

A **Global Word Counter and Header Latch** is also implemented. This is normally used to provide a total count of all data read from a group of modules. It is clocked by the FPORT Controller. The value of this counter is inserted into the output data stream along with 5 bits of header information on command of the OPORT Controller. The FPORT Controller provides both an immediate and delayed "end-of-event" signal for this purpose. The word count is limited to 12 bits. The 5 bit header field is loaded directly by the processor. Register definitions and operating modes for the counter/header latch and OPORT controller are outlined in section 2.3.3.

A small **Control FIFO** connects the word count / header output and Data FIFO outputs to allow overlapping of events in the Data FIFO. This is necessary for variations in event size and input-output transfer rates. The immediate end-of-event signal is normally used to transfer the current word count/header to the Control FIFO. The delayed end-of-event signal, which tracks the data through the Data FIFO, is used to insert the word count/header into the output data stream.

2.2 On-Board Processor

The processor is a Motorola 68020 running at 20 MHz. The 68020 was selected strictly on the basis of software compatibility with existing Fermilab modules.

The processor address map is decoded as follows;

Base	Address	Transfer Type	Name	Purpose
0000	0000	byte, word, long word	ROM1S*	Program EPROM (Bank 1, 256K)
0004	0000	byte, word, long word	ROM2S*	Program EPROM (Bank 2, 256K)
0020	0000	byte, word, long word	RAM1S*	Program RAM (Bank 1, 128K)
0022	0000	byte, word, long word	RAM2S*	Program RAM (Bank 2, 128K)
0030	0000	byte	NVDS*	Real-time clock / Non-volatile memory
0040	0000	byte, word, long word	ETHS*	Ethernet Dual-Port RAM
0044	0000	long word	CAS	Ethernet Channel Attention
0046	0000	byte	UARTS*	Serial Ports (68681)
0048	0000	byte	TMRS*	Interrupt Vector/Timer Logic (MC68901)
004A	0000	byte	OPORTS*	OPORT Controller (EPB1400)
004C	0000	byte	PARLLS*	Auxiliary Parallel Port
004E	0000	word	H&CSEL*	Header/Counter Registers
0050	0000	byte, word, lon ord	LPBKS	Ethernet loopback mode set
0052	0000	byte, word, lon ord	LPBKC	Ethernet loopback mode clear
0054	0000	byte, word, lon ord	ETHRES	Ethernet reset set
0056	0000	byte, word, lon ord	ETHRESC	Ethernet reset clear
0060	0000	long word	FB1S*	FPORT Controller (Fast cycle instructions)
0062	0000	long word	FB2S*	FPORT Controller (Slow cycle instructions)
0064	0000	byte	FCSEL*	FASTBUS Parallel Port 1 (MC68230)
0066	0000	byte	FSSEL*	FASTBUS Parallel Port 2 (MC68230)
0068	0000	long word	FIFO1S*	Processor FIFO
006A	0000	long word	FIFO2S*	List FIFO

* Active low signal.

The processor memory consists of 256 KBytes of 0-wait state static RAM and 512 KBytes of 1-wait state EPROM. Byte, word and long-word accesses are supported. Minimal configuration is 128 KBytes of RAM and 256 KBytes of EPROM.

A DS1286 real-time clock with interrupt capability and 50 bytes of RAM is provided for system use. Both functions are maintained by a built-in battery. The RAM holds module-specific information (e.g., Ethernet address).

Processor reset occurs at power-up, by pressing the front-panel Reset Pushbutton or through the front panel reset input Lemo. A reset can also be generated by a "watchdog" timer contained in the FASTBUS Parallel Port 1 controller (68230). If enabled, this timer must be reset by software periodically. The timer can be set for any period from 4 µsec up to approximately 50 seconds. A module reset can be forced by software which drives the parallel port watchdog timeout bit low.

A Processor bus response timer is contained in the other 68230 port controller. When enabled, this timer will generate a BUS ERROR exception if the Processor fails to complete a bus cycle within the specified time. The bus response timer also serves as the FASTBUS "long timer". In this application however, a FASTBUS timeout will only be detected on a subsequent Processor to FPORT Controller access. When the FPORT Controller is operating in a standalone mode for extended periods (as in a microcode readout loop), the Processor should occasionally access the FPORT Controller using the NULL instruction to determine if a timeout has occurred.

2.2.1 Control and Status Registers

The FSCC has no control or status registers which are independent of the associated controllers. Refer to the appropriate controller description for register definitions.

2.2.2 Error Responses

The FPORT Controller normally monitors the FASTBUS exception logic. When a FASTBUS error or FASTBUS Reset is detected, the controller will terminate its current operation and return to an idle state. If another FASTBUS instruction is pending (Processor pipelined mode), the FPORT Controller will return Processor DSACK immediately to clear the bus and allow interrupt processing. The instruction which was pending will not be executed. If the FPORT Controller fails to recognize an error and return control to the Processor, the Processor will eventually timeout with a BUS ERROR exception. This can happen, for example, if the address/data cycle timer is disabled and no acknowledge is received.

FASTBUS exceptions can be cleared by asserting the parallel port FASTBUS clear error (**FCLERR**) or sequencer clear error signals. The FASTBUS interrupt vector must be programmed and the interrupt enabled to allow recognition of FASTBUS exceptions.

Because FASTBUS operations can be queued, a FASTBUS error interrupt may not apply to the current processor data cycle. For example, a FASTBUS block transfer can be initiated and then followed by any number of non-FASTBUS processor operations while the block transfer takes place. A FASTBUS error during the block transfer generates an interrupt which may be unrelated to the current processor activity. For any FASTBUS interrupt, the parallel port status lines must be examined to determine the cause of the interrupt.

2.2.3 Software

The pSOS operating system and pROBE resident monitor, module diagnostics and control software for the Ethernet and Serial Ports is implemented in EPROM. User code may be placed in the second EPROM bank or in RAM.

2.2.4 Interrupts

A 68901 multifunction peripheral provides the vectored interrupt control. Refer to the Motorola 68901 manual for register definitions. Interrupts for the following inputs are provided;

Interrupt Line	Name	Function
7	TSTRB	Trigger Input Strobe
6	RTCREQ*	Real-time Clock
5	ETHREQ	Ethernet
4	SERREQ*	Serial Port
3	FBERR*	FASTBUS error
2	AUXREQ*	OPORT Controller
1	FBREQ*	FASTBUS request
0	FPCREQ*	FASTBUS Port Controller

* Active low signal.

Interrupt vectors are programmable and interrupts can be separately enabled, disabled or masked in the 68901 controller. The 68020 interrupt mask is hardwired to IPL2, which corresponds to a interrupt level of 4, and is not used except to enable or disable all interrupts. A typical FSCC readout program will have an inner loop of less than 100 µsec, so for many applications, polling of the serial ports, slave mode and trigger inputs may be more efficient than use of interrupts.

TSTRB-IRQ7: The trigger input interrupt is generated by the rising edge of the front panel trigger strobe. Four trigger ID bits are latched in parallel port 1.

RTCREQ*-IRQ6: The real-time clock (DS1286-U38) can be programmed to generate periodic interrupts at a rate of 10 msec to 100 seconds. It can also be programmed to interrupt on a specific date or/and time.

ETHREQ-IRQ5: This is the Ethernet message interrupt. Ethernet messages are buffered by the controller in Dual-Port memory so immediate interrupt response is not required.

SERREQ*-IRQ4: The serial ports can be programmed to generate interrupts when the receive buffer is loaded or the transmit buffer is empty.

FBERR*-IRQ3: FASTBUS error conditions result in a processor interrupt when enabled. The conditions are:

1)FASTBUS timeout-failure of a slave module to respond within 1500 ns on an address or data cycle. This error can be disabled through the short timer enable bit (STEN) in parallel port 2.

2)FASTBUS SS errors-SS responses of 1,2,3,4,5,6 or 7 on a Address cycle or 1,3,4,5,6, or 7 on a Data cycle will cause a FASTBUS error interrupt. The last non-zero value of SS is latched at parallel port 2 (FLSS0, FLSS1, FLSS2).

3)Data FIFO overflow.

AUXREQ*-IRQ2: OPORT request to processor, see section 2.3.3

FBREQ*-IRQ1: (External) FASTBUS request-This interrupt is generated when the FSCC is accessed as a slave or when a FASTBUS Service Request (SR) or FASTBUS Reset (RB) is issued. RB does not directly reset the processor. A processor reset can be generated by software in the RB interrupt handler. RB does not cause an interrupt if it is being driven by the FSCC itself.

FPCREQ*-IRQ0: The FPORT controller can assert an interrupt request at any time. In the standard FASTBUS instruction set this interrupt is generated at the End of Block in Block and Pipelined transfers.

2.3 Communication Interfaces

The Ethernet interface physical connection will be through a front-panel "cheapernet" BNC connector. Software for this interface must support connection to VAX systems. The 82586 coprocessor interfaces to the 68020 through a 2K by 32 bit dual-port memory. This configuration allows both processors to operate independently and is necessary to avoid buffer overrun if the 68020 is unable to process interrupts for extended periods. It also improves the speed of FASTBUS operations since there is no contention on the processor bus.

Two standard RS-232 serial ports (Signetics 68681) are provided for development and diagnostic use. One of these will be connected to a terminal or PC. The other will typically serve as a link to a host machine. Data rates to 9600 baud will be supported. Operation of these ports is controlled by on-board software. At module initialization, the ports are configured for 9600 baud using XON / XOFF protocol. Refer to the 68681 data sheet for register definitions. The Host port is a null-modem connection.

FASTBUS is accessed through the crate segment backplane. All FASTBUS operations are performed by a microsequencer as directed by the processor. A sequence of FASTBUS transfers may be performed to the Data FIFOs or to the Processor FIFOs without processor intervention.

The Output Port is controlled by a programmable microsequencer which is directed by the processor. A transfer may be initiated and proceed unattended. The Output Port transfers data through the Auxillary connector of the backplane. Personality cards, which plug into the auxillary connector from the rear of the FASTBUS crate are available for standard protocols.

2.3.1 Addressing Modes (FASTBUS)

Two 68230 parallel ports provide status communication between the processor and the FASTBUS control logic. Parallel port lines are defined in Figure 3 ("I" in the first column indicates an input signal, "O" indicates an output, "S" indicates a special function pin)

Port data direction and signal states are individually programmable. The software should avoid defining or driving INPUT pins as OUTPUTs, since multiple drives on the same signal line can cause circuit damage. If in doubt, define all unused signals as INPUTs or leave ports in the normal power-on reset configuration.

The FASTBUS Port Controller provides most of the low-level control of FASTBUS operations, based on simple memory-mapped instructions from the processor. It consists of three parallel EPS448 programmable sequencers plus assorted PLD's. The EPS448 is limited to 256 states, of which only 64 support conditional branching. Therefore, the sequencer will directly execute only FASTBUS primitives plus a few generally used compound operations.

Parallel Port 1

Input/Output	Bit	Signal	Function
I	A0	FRRD	FASTBUS received RD
I	A1	FRDS	FASTBUS received DS
I	A2	FRMS0	FASTBUS received MS0
I	A3	FRMS1	FASTBUS received MS1
I	A4	FRMS2	FASTBUS received MS2
I	A5	FRAK	FASTBUS received AK
I	A6	FRDY	FASTBUS master
I	A7	FSLV*	FASTBUS slave mode
I	B0	TRIG0	Trigger Vector 0 All Port B inputs are latched
I	B1	TRIG1	Trigger Vector 1 on the rising edge of the
I	B2	TRIG2	Trigger Vector 2 trigger input strobe.
I	B3	TRIG3	Trigger Vector 3
O	B4	FP0	Front Panel 0 Front Panel differential ECL
O	B5	FP1	Front Panel 1 Outputs
O	B6	FP2	Front Panel 2
O	B7	FP3	Front Panel 3
O	C0	PRS*	Processor FIFO reset
I	C1	PEF*	Processor FIFO empty
O	C2	COPYEN	Processor FIFO copy enable
O,S	C3	WDTO*	"Watchdog" timeout
O	C4	DRS*	Data FIFO reset
O	C5	DRT*	Data FIFO retransmit
O	C6	SRS*	Sequencer List FIFO reset
I	C7	DFF*	Data FIFO overflow

Parallel Port 2

Input/Output	Bit	Signal	Function
O	A0	FDGK	FASTBUS drive GK
O	A1	FDRB	FASTBUS drive RB
O	A2	FDSS0	FASTBUS drive SS0
O	A3	FDSS1	FASTBUS drive SS1
O	A4	FDSS2	FASTBUS drive SS2
	A5		
	A6		
O	A7	LCEN	Local Counter Enable
I	B0	SSTAT0	Sequencer status 0
I	B1	SSTAT1	Sequencer status 1
I	B2	SSTAT2	Sequencer status 2
I	B3	SSTAT3	Sequencer status 3
I	B4	FRSR	FASTBUS SR
	B5		
O	B6	FCLERR	FASTBUS Clear Errors
O	B7	SNRESET*	FPORT Controller Reset
O	C0	STEN	Short timer enable
I	C1	STO*	Short timeout
I,S	C2	LTEN	Long timer enable
O,S	C3	LTO*	Long timeout
I	C4	FLSS0	FASTBUS latched SS0
I	C5	FLSS1	FASTBUS latched SS1
I	C6	FLSS2	FASTBUS latched SS2
I	C7	FRESET*	FASTBUS reset

Figure 3

The EPS448 is EPROM programmable and the instruction set cannot be modified by the processor. FPORT Controller output signals are as follows:

	Bit	Name	Function
EPS448-3	F00	SMUX0	Select sequencer condition code set 0
	F01	SMUX1	Select sequencer condition code set 1
	F02	SMUX2	Select sequencer condition code set 2
	F03	FDSACK*	Processor data strobe acknowledge
	F04	SSTAT0	Sequencer status bit 0
	F05	SSTAT1	Sequencer status bit 1
	F06	SSTAT2	Sequencer status bit 2
	F07	SSTAT3	Sequencer status bit 3
	F08	CCLERR	Controller Clear Error
	F09	TIMER	Short timer enable
	F10	CEOE	Control "End-of-Event"
	F11	DEOE	Data "End-of-Event"
	F12	FCLK	Global word counter clock
	F13	STATCLK	Sequencer status clock
	F14	SR*	Sequencer List FIFO read
	F15	PFIFOEN	Processor FIFO input enable
EPS448-2	F00	FSAS	FASTBUS set AS
	F01	FCAS	FASTBUS clear AS
	F02	FSDS	FASTBUS set DS
	F03	FCDS	FASTBUS clear DS
	F04	FSDK	FASTBUS set DK
	F05	FCDK	FASTBUS clear DK
	F06	FDWT	FASTBUS drive WT
	F07	FDMS0	FASTBUS drive MS0
	F08	FDMS1	FASTBUS drive MS1
	F09	FDMS2	FASTBUS drive MS2
	F10	FDRD	FASTBUS drive RD
	F11	FDEG	FASTBUS drive EG
	F12		
	F13	FEOBA	FASTBUS EOB Acknowledge
	F14	FREQ	FASTBUS request bus
	F15	FREL	FASTBUS release bus
EPS448-1	F00	DFIFOEN	Data FIFO input enable
	F01	FPCREQ*	Sequencer Interrupt Request (TRAP)
	F02	FCOE*	FASTBUS control output enable
	F03	FDOE*	FASTBUS data output enable
	F04	SDG*	Data pipeline latch enable
	F05	DSBA	Data pipeline latch B->A mode control
	F06	DCPBA	Data pipeline latch B->A clock
	F07	DSAB	Data pipeline latch A->B mode control
	F08	SDCPAB	Data pipeline latch A->B clock
	F09	DDIR	Data pipeline latch direction
	F10	SPOE*	Processor tranceiver output enable
	F11	SDW	Data FIFO write
	F12	SRT*	Sequencer List FIFO retransmit
	F13	SLCOE	Local word counter output enable
	F14	LC0	Local word counter control 0
	F15	LC1	Local word counter control 1

The following FASTBUS master operations are permitted by the sequencer instruction set:

EQU Address(HEX)	FASTBUS Operation
0062 0300	BUS_ARBITRATE
0060 0004	BUS_RELEASE
0060 0304	ADDRESS_DATA_GEOGRAPHICAL
0060 0308	ADDRESS_CSR_GEOGRAPHICAL
0060 030C	ADDRESS_DATA_LOGICAL
0060 0310	ADDRESS_CSR_LOGICAL
0060 0314	ADDRESS_DATA_BROADCAST
0060 0318	ADDRESS_CSR_BROADCAST
0060 031C	ADDRESS_RELEASE
0062 0320	DATA_PROCESSOR_RANDOM_READ
0060 0324	DATA_PROCESSOR_RANDOM_WRITE
0062 0328	DATA_PROCESSOR_SEC_ADDRESS_READ
0060 032C	DATA_PROCESSOR_SEC_ADDRESS_WRITE
0062 0008	DATA_PROCESSOR_BLOCK_TRANSFER_READ
0060 000C	DATA_PROCESSOR_BLOCK_TRANSFER_WRITE
0060 0330	DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE
0060 0334	DATA_FIFO_BLOCK_TRANSFER_READ
0060 0338	DATA_FIFO_PIPELINED_READ_100
0060 033C	DATA_FIFO_PIPELINED_READ_200
0060 0340	DATA_FIFO_PIPELINED_READ_400
0060 0020	NULL
TBD	LIST
TBD	LIST_REPEAT
0062 0010	LOCAL_COUNTER_LOAD
0062 0014	LOCAL_COUNTER_READ
0062 0018	FIFO_WRITE_DATA
0062 0024	EOE

Sequencer instructions related to slave mode are;

0062 001C	SLAVE_DATA_INPUT
0060 0344	SLAVE_DATA_OUTPUT

Refer to Appendix A for detailed instruction definitions.

The FSCC supports standard and assured access arbitration. It does not support prioritized arbitration. Bits 0-5 of the processor data bus contain the arbitration vector. Bit 7 determines whether the assured access protocol is active. Bit 6 and 8-31 are ignored. These bit assignments correspond to CSR 8. A FASTBUS arbitration can be performed by a processor instruction of the form -

```
MOVE.L    CSR8 , BUS_ARBITRATE
```

If there is more than one master on the segment, the arbitration request should be followed by a check of the parallel port **FRDY**, **FRAK** and **FSLV*** bits to confirm that the arbitration was successful (or that the module was addressed as a slave while waiting for the bus). **FRDY** indicates that the FSCC has won the arbitration cycle and is the pending bus master. Before proceeding with an address cycle, **FRAK** should be checked to ensure that the previous master has released the bus. If **FSLV*** is low following an arbitration request, it indicates that the FSCC has been addressed as a slave by the current bus master. This should simultaneously cause a FASTBUS slave interrupt. The bus can also be acquired simply by asserting the GK line through the parallel port. Note that this is not a standard FASTBUS operation and is provided only for single master systems without ancilliary logic.

The Processor can issue a FASTBUS RB (Reset Bus) signal directly through the parallel port. In this case the parallel port GK signal should be asserted simultaneously. RB does not cause a processor interrupt when it is driven by the FSCC itself.

The FSCC supports a limited slave mode through processor emulation. Geographical address recognition logic is contained in hardware. All other slave functions are controlled by the processor. FASTBUS WT is asserted on each DS transition. The data cycle response time is dependent on software and is typically 10-15 µsec per FASTBUS word.

Accessing the FSCC as a slave causes a Processor interrupt.The processor must poll the DS, MS, RD and AK lines via parallel port inputs. The SS response is then placed on the bus via parallel port outputs and a slave mode input or output operation is executed by the sequencer.

These instructions cause a single word of data to be transferred between the processor and FASTBUS with an associated DK transition. WT is released prior to the DK transition and is reasserted on the next DS transition. Any FASTBUS CSR or DATA location can be defined by processor software.

The FSCC is limited in its ability to execute master and slave operations simultaneously, since the processor is involved in both cases. There is normally no need for the FSCC to address itself, with the possible exception of crate mapping. If the FSCC does address itself, the slave logic will attach (return AK) but no data cycles will be possible. A slave mode interrupt is generated and the processor software must then recognize the simultaneous master/slave condition by examining the parallel port **FSLV*** bit. The software should bypass any FASTBUS data cycles and retrieve the information directly from internal memory.

Application-specific FASTBUS operations may be included through use of the LIST instruction. These routines can use, but are not required to use, the LIST FIFO as a source of data bus operands. The Sequencer List FIFO provides a means of executing complex FASTBUS instruction streams without Processor intervention. The List FIFO contains only the data bus operands, instructions must be preprogrammed in the sequencer. For example, a complete crate of similar slave modules could be read out by a sequencer loop using a List of primary addresses. Small burst data transfers (e.g., FASTBUS messages) can also be supplied by the List FIFO. This FIFO eliminates the overhead associated with Processor-FPORT Controller communication. The LIST routine cannot contain FASTBUS operations which require Processor interaction (such as read cycles) unless the data is simply placed in the Processor FIFO for future access.

The LIST instruction causes the operation to be executed once. The LIST_REPEAT instruction causes the operation to repeat indefinitely. During LIST_REPEAT, the sequencer checks the Processor-FPORT Controller select line at the completion of each pass and exits the loop if

another instruction is pending. The only exception is the NULL instruction where the sequencer returns Processor DSACK and continues. NULL can be used to periodically check that the FPORT Controller (or FASTBUS) is not locked up.

The LIST instructions rely on programming of the FPORT Controller microcode which is EPROM based. Use should be limited to applications requiring very fast operation. In general, the standard primitive instructions provide reasonably fast (Processor speed) access to FASTBUS without changes to microcode.

2.3.1.1 Data Transfer Description and Transfer Rates

The FSCC supports Pipelined Transfer rates of 100, 200 and 400 nsec per word. At the 100 nsec per word rate, End-of-Block is not normally returned in time to avoid another DS transition. Therefore, Pipelined transfers at 100 nsec will read one word beyond the end of block and this word will be included in the output data stream. Other pipelined transfer rates, in any multiple of 50 nsec with a minimum of 100 nsec, can be programmed through minor microcode changes.

The FSCC also supports Block Transfers at a rate of ≥150 nsec per word. The actual transfer rate will be 100 nsec plus the slave DS-DK response time, rounded to the next highest 50 nsec increment.

Block and Pipelined Transfers directed to the Data FIFO are functional for Read mode only. Data is placed in the FIFO and is not accessible to the Processor. However, data can be simultaneously routed to the Processor FIFO by setting the COPYEN bit in the parallel port. A Block Transfer Read/Write instruction for Processor memory is available, but operates at the same speed as single word Processor read and write operations. It implements one step of the block transfer for each processor MOVE instruction executed. The block transfer must be terminated with a DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE instruction.

Although the Data FIFO is limited to blocks of 2K words or less, larger blocks can still be transferred provided that the destination does not require a leading word count or is capable of interpreting a word count of 2048 as a partial transfer. In the second case, for block transfers which are some exact multiple of 2048 words, a final block of length zero must be transferred. If leading word count insertion is not required and the output port runs at the same rate as the FASTBUS port, the FIFO depth is not a factor.

2.3.1.2 Internal Control and Status Registers

All Control and Status Registers related to FASTBUS operation are implemented in 68020 software. The number of registers is limited only by available processor memory.

2.3.1.3 Error Responses

The FPORT Controller latches FASTBUS error conditions but does not attempt any recovery or retry. A processor interrupt is generated and the processor has the option of attempting recovery or, more likely, skipping the entire readout sequence. Because of internal pipelining, errors may not be reported immediately with the bus cycle generating the error. Also, the amount of status information available to the processor is limited.

The standard FASTBUS instructions will abort on errors and return to the processor with an interrupt. The user can optionally modify the microcode to ignore errors and continue with the FASTBUS operation, although the results may be undefined. If user supplied microcode is capable of recovering from errors without processor intervention, the processor error interrupt can be disabled. An error flag can be set in the header word of the current output event to indicate that the FASTBUS event readout failed and the data should be discarded. The FPORT Controller does not have the ability to test status flags in slave modules to locate problems, so error recovery through microcode is generally limited to reset and continue type operations.

2.3.2 Ethernet

The INTEL 82586 LAN coprocessor performs message framing management in transmission and reception functions. It acts as a bus master, accesses memory by DMA, carries out message error checking, collision recovery functions, etc. The INTEL 82C501 Ethernet Serial Interface (ESI) implements Manchester encoding/decoding and clock recovery. ESI functionality may be checked by the processor using Loopback mode. The National DP8392 Ethernet Transceiver performs collision detection and interfaces to the coaxial cable.

Ethernet Interface

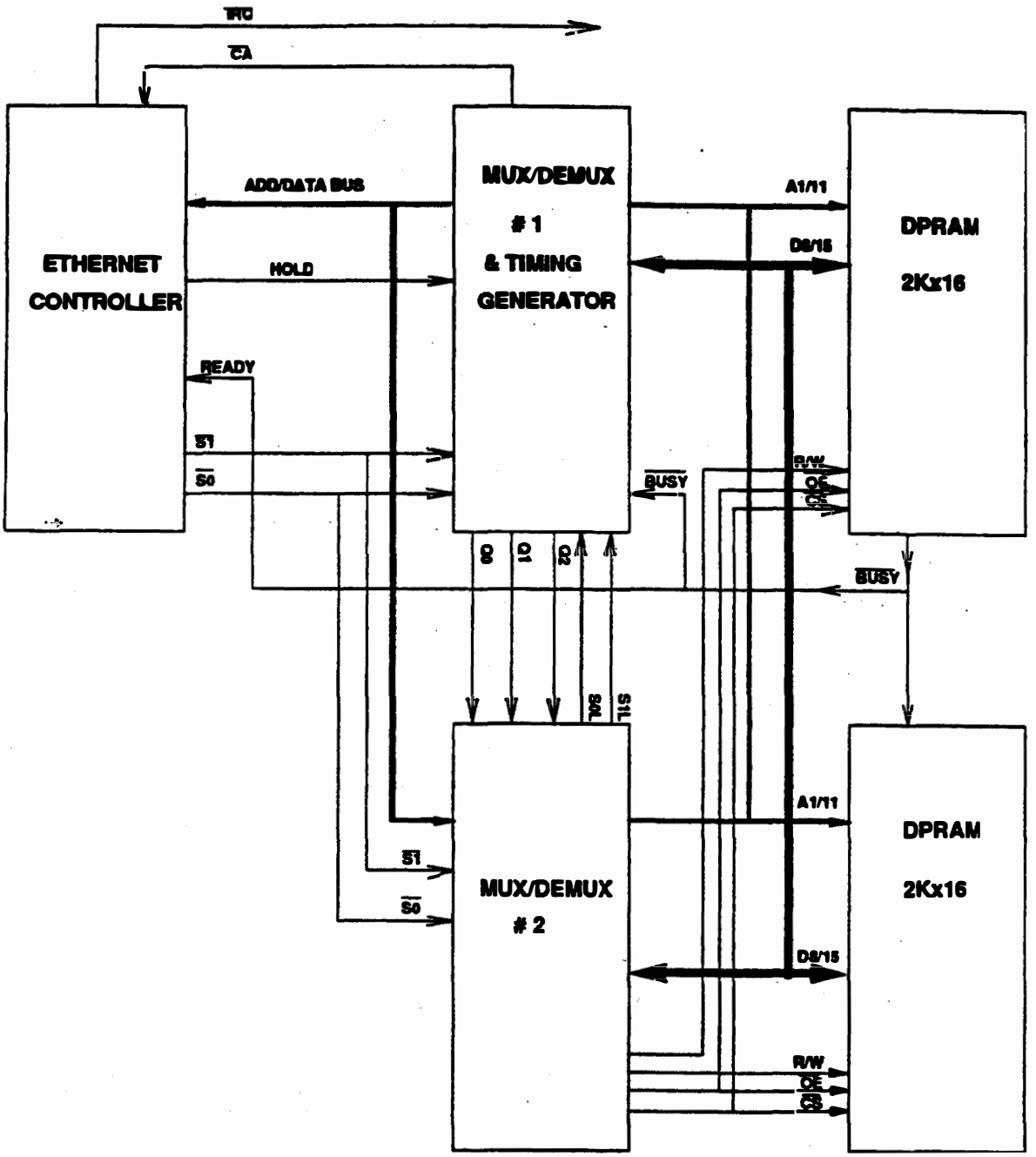
The INTEL 82586 coprocessor interfaces indirectly to the 68020 CPU system through a 4K by 16 bit dual ported memory. Figure 4 is a Block Diagram of the Ethernet Interface.

The dual port RAM speeds up both systems as the processor does not release the bus each time a DMA occurs, also controller bus latency is reduced to zero.

Direct interface is provided with CA and IRQ lines. The 68020 CPU drives CA to get the attention of the controller, indicating that new commands were included in the command list to be processed.

The 82586 Ethernet controller uses IRQ to interrupt the processor when a command is over or upon message reception.

The shared DPRAM structure is composed of four parts: The Initialization Root, the System Control Block (SCB), the Command List and the Receive Frame Area (RFA). The Initialization Root is fixed in memory. The Ethernet controller addresses that variable as **$FFFFF6**, but in fact, due to the partial decoding, its physical location is **$1FF6**. The origin of the DPRAM is **$40000** from the CPU side and **$0000** from the Ethernet controller. Transmission and reception messages are split into small buffers to better use the available memory. The buffers, when necessary, are chained in frames. Transmission and reception buffer descriptors are accessible through the SCB table pointers.



Ethernet Interface Block Diagram

Figure 4

Ethernet Controller Interface

The interface between the Ethernet controller and the DPRAM consists of two ALTERA EPB1400s. The Ethernet controller, address and data buses are multiplexed, so the EPB1400s demultiplex/remultiplex those buses and generate the necessary timing for the memory signals: CS*, R/W*, OE*.

Logic in the EPB1400s is used for as a state machine (Figure 5) to control timing. This logic monitors the state of the BUSY line from the DPORT RAM. To indicate a memory access contention in read and write cycles, BUSY* is connected to the READY line of the controller and inserts wait states.

In both, read and write cycles, the state machine's counter is kept at states 3 and 4 when BUSY* is activated.

Timing Diagrams

The Ethernet controller was designed to share an external bus, it asserts a HOLD line to keep the bus when performing burst memory cycles. The controller has a private bus to the DPRAM, so hold acknowledge HLDA is returned instantaneously, and the state machine goes directly to IDLE to wait for the beginning of a cycle.

S1 and S0, from the Ethernet controller indicate the type of cycle: S1=0, S0=1 is a read cycle and S1=1, S0=0 is a write cycle. Those lines are driven only during T1 and T2, and are latched in S1L and S2L.

Figures 6a and 6b show timing diagrams for read and write cycles. Table one is a truth table.

2.3.3 Description and Operation of OUTPUT PORT

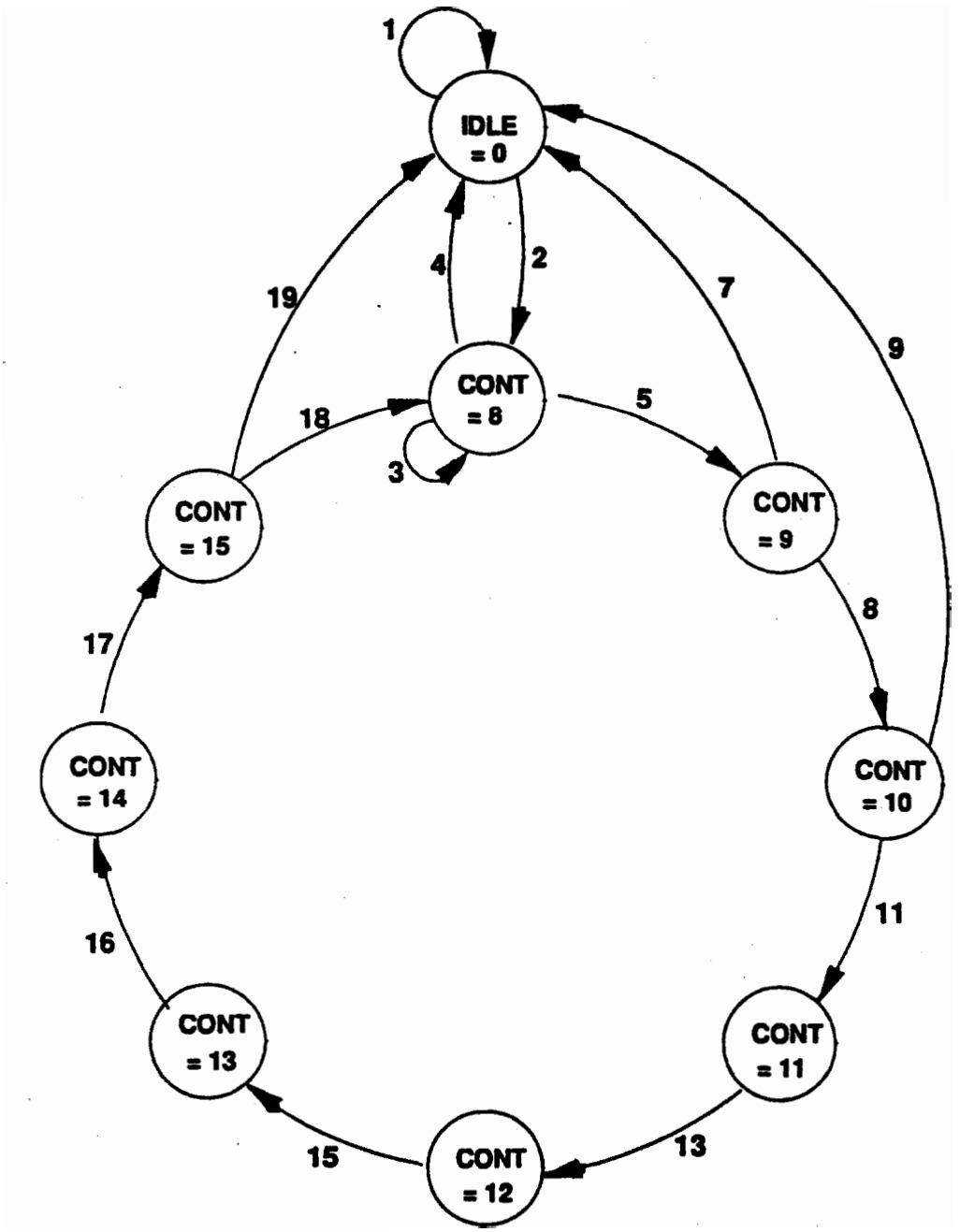
The basic function of the Output Port Controller (OPORT) is to provide a continuous stream of data with strobes for transmission via RS-485, ECLine or fiber-optic auxiliary cards. A Block diagram of the Output Port is shown in Figure 7. For this application, it need only multiplex the FIFO outputs and generate appropriate timing. Connection to more complicated external bus formats will require reprogramming of the OPORT Controller or addition of a more sophisticated sequencer on the auxiliary card. The OPORT can also be disabled to allow an external controller mounted on the FASTBUS auxiliary card to take over data transmission. This allows for various protocols to be implemented by fitting different auxiliary cards to the controller.

All levels at the output port are single ended TTL. Level adapters to different protocols will be mounted on the FASTBUS auxiliary card. The data path is 32 bits wide. Two control lines are provided to regulate data flow: the WAIT input pauses the OPORT and STROBE is a synchronous data strobe output.

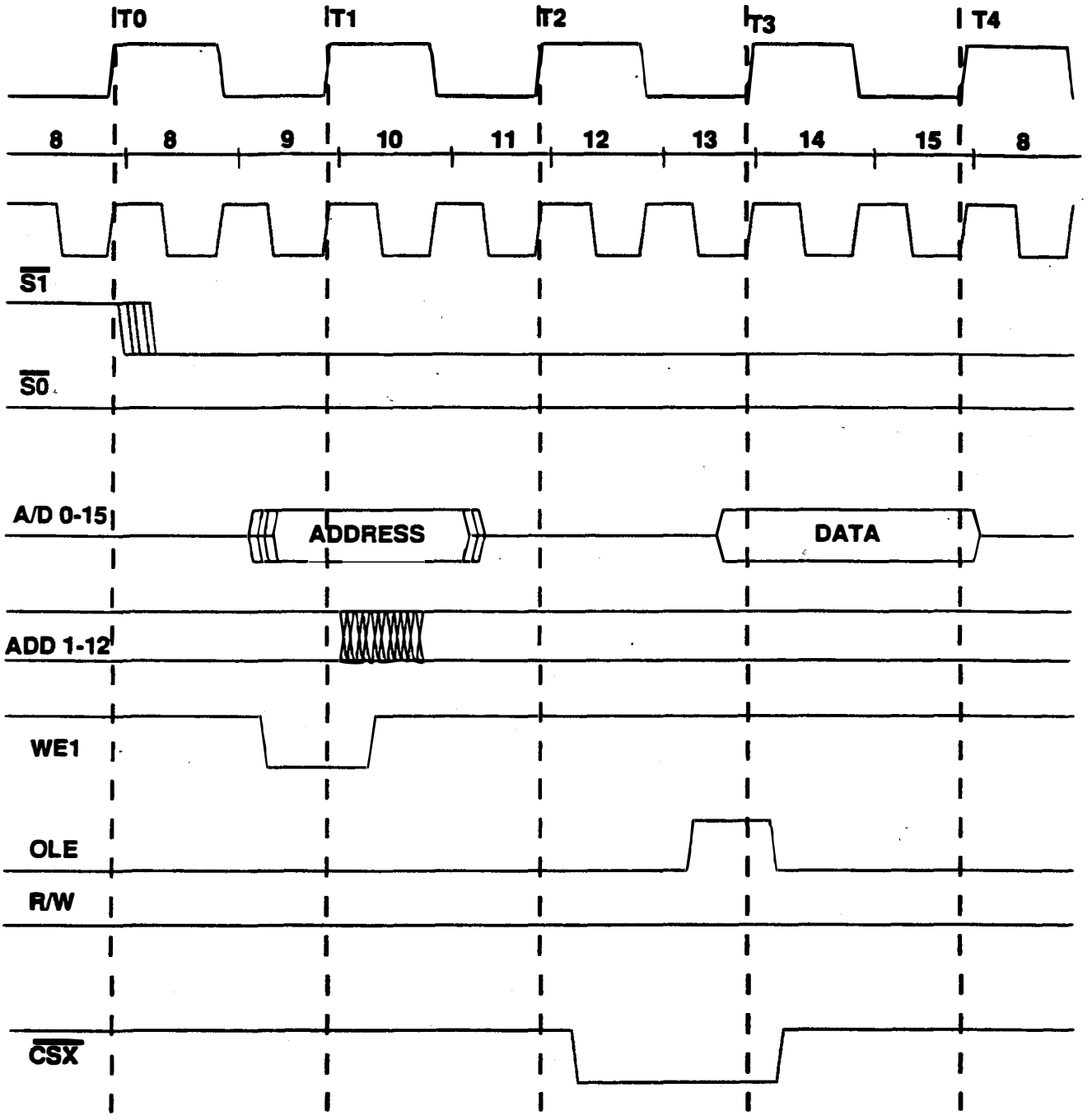
Ethernet Controller Truth Table

	HOLD	S1	S0	BUSY*	COUNT		COUNT
	0	X	X	X	0	|	0
	1	X	X	X	0	|	8
	1	1	1	X	8	|	8
	0	X	X	X	8	|	0
READ CYCLE							
	1	0	1	X	8	|	9
	0	X	X	X	9	|	0
	1	1	1	X	9	|	0
	1	0	1	X	9	|	10
	0	X	X	X	10	|	0
	1	1	1	X	10	|	0
	1	0	1	X	10	|	11
	X	X	X	X	11	|	12
	X	X	X	X	12	|	13
	X	X	X	H	13	|	14
	X	X	X	L	13	|	13
	X	X	X	X	14	|	15
	1	X	X	X	15	|	8
	0	X	X	X	15	|	0
WRITE CYCLE							
	1	1	0	X	8	|	9
	0	X	X	X	9	|	0
	1	1	1	X	9	|	0
	1	1	0	X	9	|	10
	0	X	X	X	10	|	0
	1	1	1	X	10	|	0
	1	1	0	X	10	|	11
	X	X	X	X	11	|	12
	X	X	X	X	12	|	13
	X	X	X	H	13	|	14
	X	X	X	L	13	|	13
	X	X	X	X	14	|	15
	1	X	X	X	15	|	8
	0	X	X	X	15	|	0

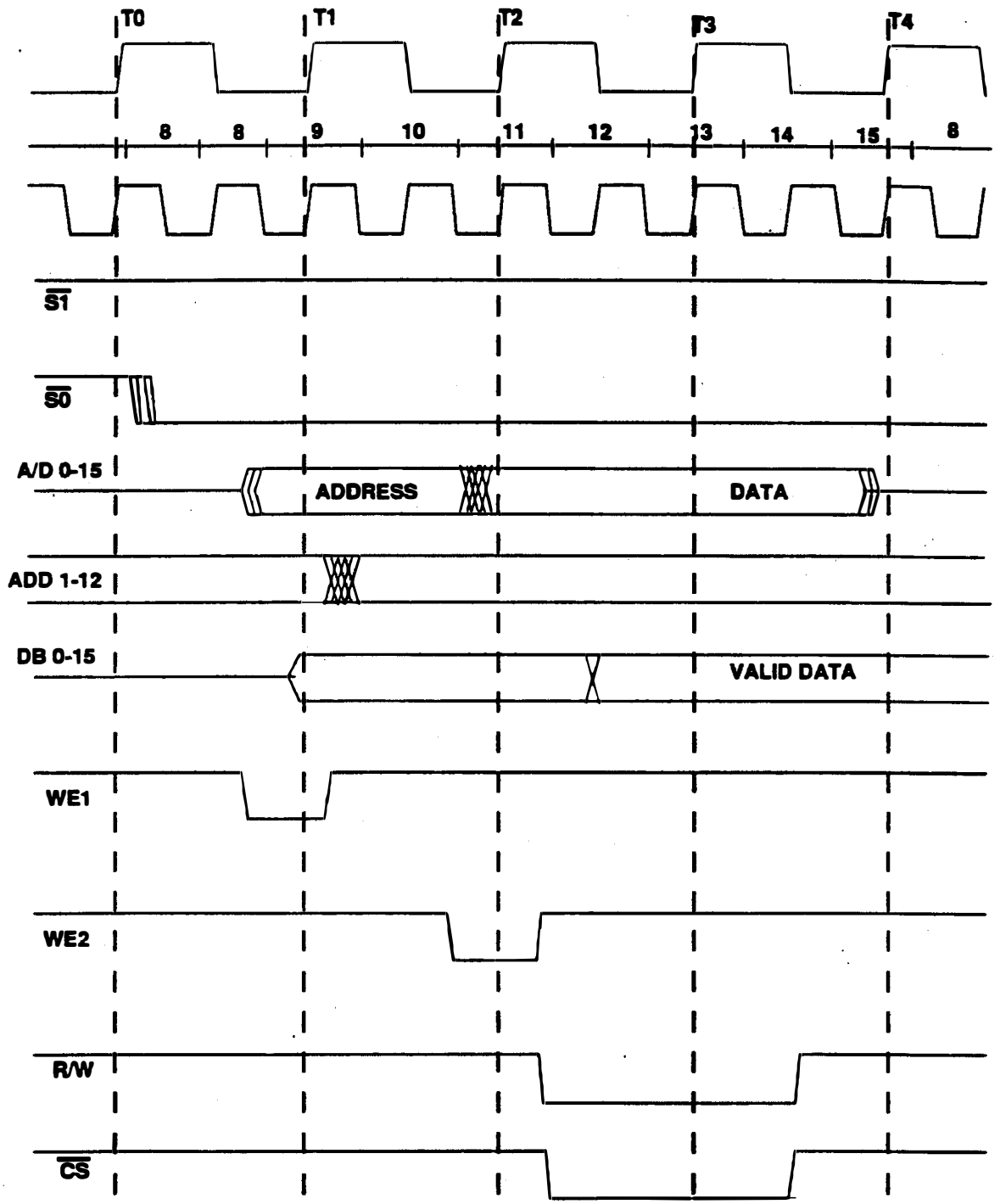
Table 1



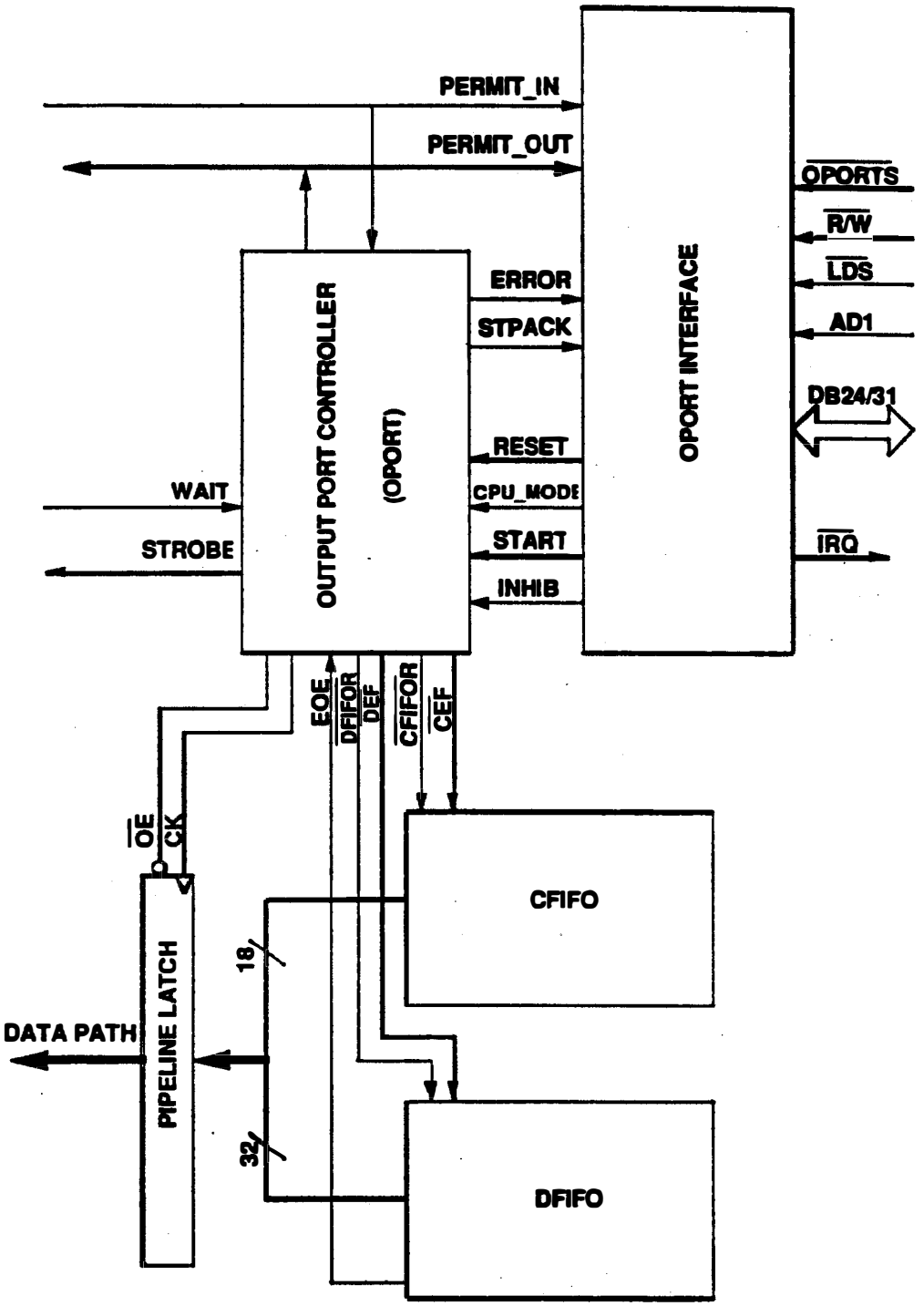
Ethernet Controller State Machine Diagram
Figure 5



Ethernet Controller Read Cycle
Figure 6a



Ethernet Controller Write Cycle
Figure 6b



OPORT Block Diagram

Figure 7

In some applications the auxiliary bus will be common to several FSCC modules. The PERMIT_IN and PERMIT_OUT signals provide a "token passing" mechanism for enabling and disabling data output. The external WAIT input, located on the auxiliary card, can also be used to regulate the data output rate.

The OPORT Controller performs arbitration and control for the Data and Word Count FIFO output ports. It has two modes of operation; *Event Mode*, where a word of Control information is transmitted from the Control FIFO followed by synchronous data from the Data FIFO at a rate of 10 MHz, and *Processor Controlled Mode*, where data is transmitted from the Control FIFO only. The operating mode is selected by the processor.

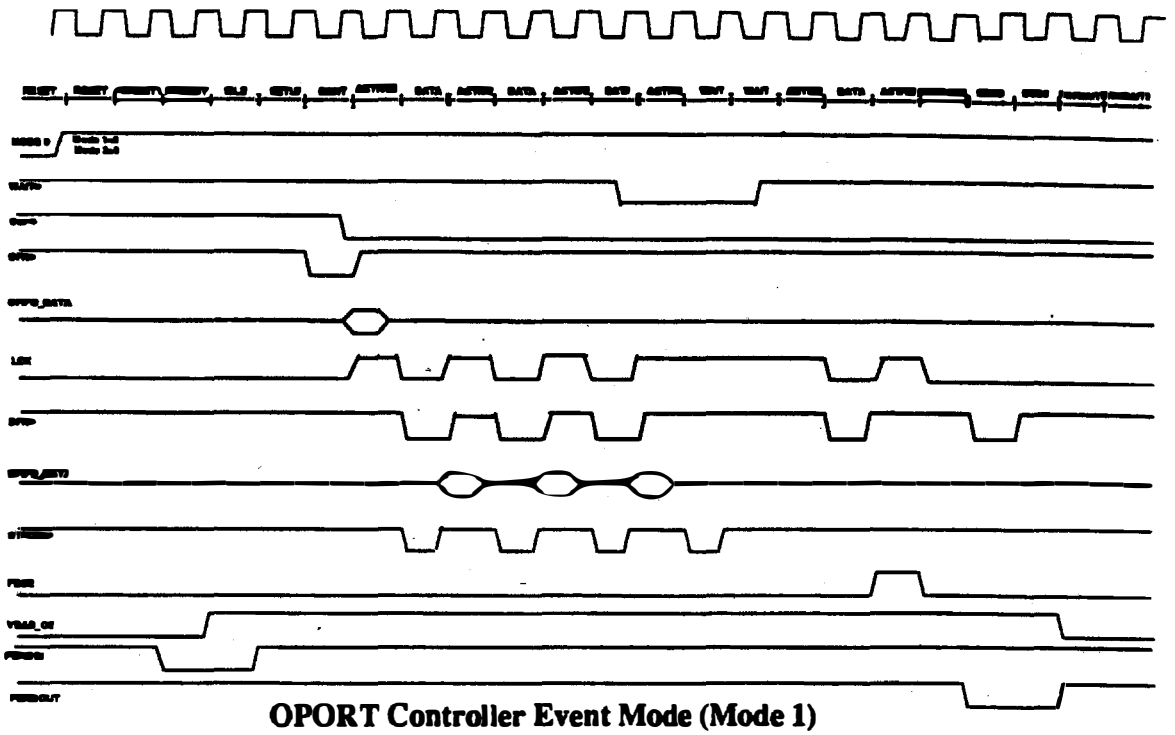
Event Mode : This is the normal data taking mode and data is transmitted to the output with synchronous, no handshake, protocol at 10MHz or 40 Mbyte/sec. The data contains a non-inclusive leading word count,(the lower order 12 bits). That header word from the control FIFO (CFIFO) is transmitted first, followed by the specified number of words from the data FIFO. Bits 12-16 contain a processor programmable header, usually the Trigger ID, and bit 17 is an error flag. Bits 18-31 of the control word are undefined and transmitted as zeros. A pipeline latch between the data FIFO (DFIFO) outputs and the output port stabilizes the data field. A pipeline register holds the data on the Output Port for a period of approximately 50 nsec preceding and 50 nsec following the trailing edge of the output data strobe (STROBE*). Wait states can be inserted by the auxiliary port in increments of 100 nsec. Refer to figure 8a for an example of Output Port timing in normal frame mode.

Processor Controlled Mode: In this mode, data is written into the Control FIFO one word at a time by the processor using the H&C transparent mode (refer to 2.3.3.3.3). Then, the MODE signal is driven high and the OPORT Controller extracts data from the Control FIFO for transmission. The processor can read the Step Acknowledge (STPACK) status bit to confirm transmission or can be interrupted. Data width is limited to 17 bits. Transmission of data in processor controlled mode is also subject to external WAIT. Refer to figure 8b for an example of timing in the processor controlled mode.

Refer to figure 9 for a state transition diagram of the OPORT Controller. At power up or RESET, the state machine is in POWER ON RESET. Receipt of a FORCE_EVENT command takes the controller directly to IDLE, while an EVENT, STEP or EMPO command puts the controller into PINWAIT. A PERMIM will then take the controller from PINWAIT to Idle. As soon as the control FIFO has a word in it, the controller will move from IDLE to SETUP state. A STEP command will take it to STEP state, any other mode will allow the controller to continue on to CONT state. However, if the WAIT signal is high before any data has been transmitted, the controller assumes that the memory buffers are not connected and enters the ERROR state asserting the ERROR signal. A RESET is required to return the controller to IDLE.

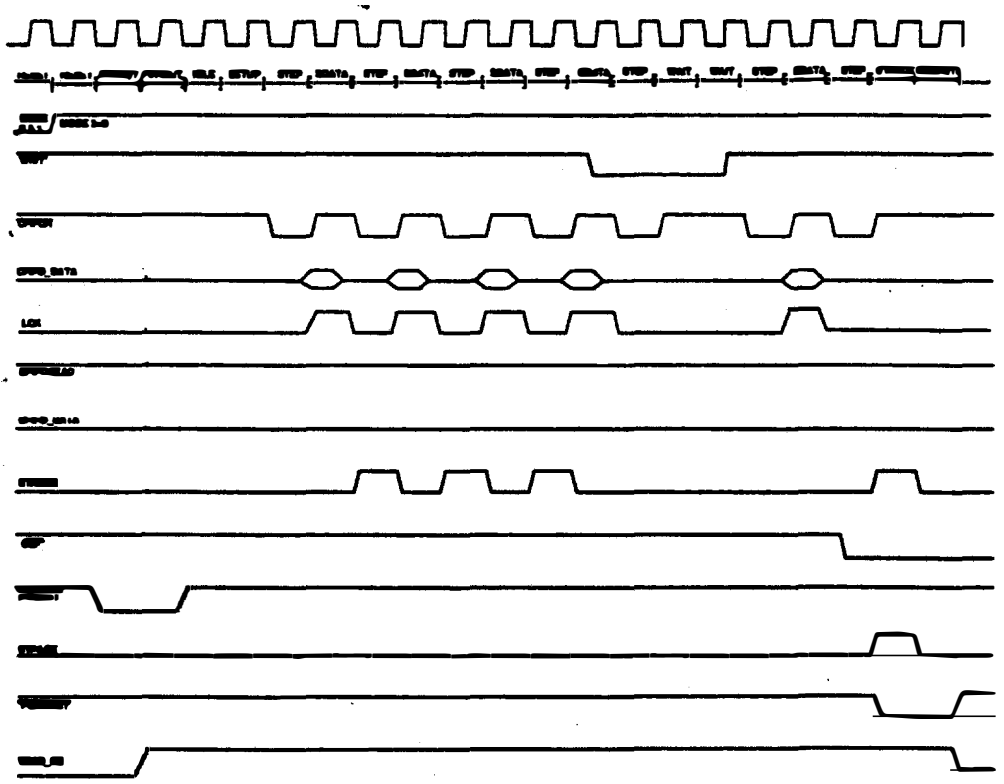
From the CONTINUE state (Event Mode) it proceeds to the ACTIVE2 state and transmits one word from the Control FIFO. It then checks for data in the Data FIFO. As long as the WAIT is low, the controller continues to the DATA state, then loops between ACTIVE and DATA states, transmitting one word of data each cycle. The block transfer is terminated by the EOE signal from the Data FIFO.

From the STEP state (Processor Controlled mode), the controller transmits data from the Control FIFO until FIFO empty, and then asserts the STPACK signal. The processor must initiate the next cycle.



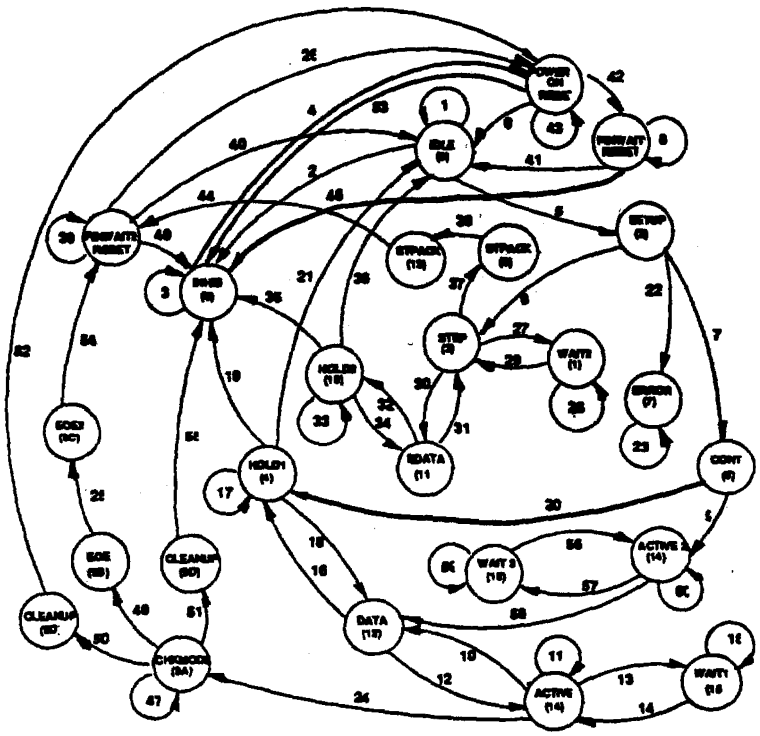
OPORT Controller Event Mode (Mode 1)

Figure 8a



OPORT Controller. CPU Controlled Mode

Figure 8b



OPORT Controller State Machine Diagram

Transition	Branch	Data	Condition
0	2	[out 00]	If Mode=2
1	3	[out 00]	Idle loop
2	1	[out 08]	If Mode =4
3	1	[out 08]	Mode=4
4	2	[out 00]	
5	2	[out 02]	If Mode=1 or 2 or 3 or 5
			/CEF
6	2	[out 03]	If Mode 3
7	3	[out 06]	
8	3	[out 00]	Idle loop
9		[out 14]	
10	3	[out 12]	If /DEF*
11	4	[out 14]	Idle loop
12	2	[out 14]	
13	1	[out 15]	If WAIT*
14	1	[out 14]	If /WAIT*
15	2	[out 15]	Idle loop
16	1	[out 04]	If Mode =0
17	4	[out 04]	Idle loop
18	3	[out 12]	If Mode=1 or 2 or 5
19	1	[out 08]	If Mode=4
20	1	[out 04]	If Mode=0
21	2	[out 00]	If Mode=3
22	1	[out 07]	If WAIT
23		[out 07]	Idle loop
24	2	[out 9A]	If EOE
25		[out 9C]	
26	2	[out 00]	If Mode=0
27	1	[out 01]	If WAIT
28	2	[out 01]	Idle loop
29	1	[out 03]	If /WAIT

Transition	Branch	Data	Condition
30	3	[out 11]	
31	1	[out 3x]	If Mode=3
32	2	[out 10]	
33	4	[out 10]	
34	3	[out 11]	If Mode=3
35	1	[out 08]	If Mode=4
36	2	[out 00]	If Mode=1 or 2 or 5
37	2	[out 05]	If /CEF *
38		[out 13]	
39	4	[out 00]	Idle loop
40	1	[out 00]	If PERMIN
41	1	[out 00]	If PERMIN + Mode=2
42	1	[out 00]	If Mode=1 or 3 or 5
43	4	(RESET)	
44		[out 00]	
46	2	[out 08]	If Mode=4
47	4	[out 9A]	
48	1	[out 9B]	If Mode=6 or 1 or 2
49	3	[out 08]	If Mode=4
50	2	[out 14D]	If Mode=0
51	3	[out 9D]	If Mode=4
52		[RESET]	
53	3	[out 08]	If Mode=4
54		(RESET)	
55		[out 08]	
56	1	[out 15]	If WAIT*
57	1	[out 14]	If /WAIT*
58	2	[out 12]	If /DEF*
59	2	[out 15]	If WAIT*
60		[out 14]	

Figure 9

OPORT Controller State Machine Outputs

State	Outputs
RESET	LCLR*
0	LCLR*,LOE*,PARLL(11)(VDAS_OE)
1	CFR*,LOE*,PARLL(11)(VDAS_OE)
2	LCLR*,LOE*,PARLL(11)(VDAS_OE)
3	CFR*,LOE*,PARLL(11)(VDAS_OE)
4	LCK,DFR*,LOE*,PARLL(11)(VDAS_OE)
5	STPACK,LOE*,PARLL(11)(VDAS_OE),PERMOUT*
6.	CFR*,LOE*,PARLL(11)(VDAS_OE)
7	ERROR,LCLR*
8	no output
9A	LCLR*,LOE*,PARLL(11)(VDAS_OE)
9B	DFR*,PERMOUT*,LOE*,PARLL(11)(VDAS_OE)
9C	LCLR*,LOE*,PERMOUT*,PARLL(11)(VDAS_OE)
9D	DFR*,LCLR*
10	LCK,LOE*,PARLL(11)(VDAS_OE)
11	LCK,LOE*,PARLL(11)(VDAS_OE)
12	STROBE,DFR*,LOE*,PARLL(11)(VDAS_OE)
13	STPACK,LOE*,PERMOUT*,PARLL(11)(VDAS_OE)
14	LCK,LOE*,PARLL(11)(VDAS_OE)
15	LCK,LOE*,PARLL(11)(VDAS_OE)

OPORT input/output signals

The processor can control and monitor the OPORT through an EPB1400 parallel port interface. The following signals are used to configure and monitor the OPORT.

RESET: This asynchronous line forces the OPORT into an inactive state if an external controller is to be used.
MODE 2-0: Selects one of seven possible OPORT operating modes.

Command	Hex-Code	Function
Hold	$00	Send a HOLD request to the OPORT. Normal execution can be resumed any time.
Event_Mode	$01	Sets the OPORT in Event Mode execution.
Force_Event_Mode	$02	Sets OPORT in Event Mode with no PERMIN Required.
CPU_Mode	$03	Sets the OPORT in Processor Cont. Mode execution
Inhib_Mode	$04	Puts the OPORT off-line. The output lines are floated.
EMPO	$05	Event_With_Manual_Permit_Out_Mode: Functionally same as Event_Mode except that upon completion of the event data transfer, the controller waits with it's outputs enabled and does not generate a Permit Out until a PO command is received.
the		
PO	$06	Permit_Out_Mode: Used in conjunction with the EMPO command, it tells the controller to generate Permit_Out, disable it's outputs and return to PINWAIT state until a new token arrives.
Opo_Reset	$08	Send a reset signal to the OPORT.

The OPORT controller drives output lines to indicate status conditions to the processor:

STPACK: Indicates the CPU controlled cycle is over..
ERROR: Indicates that WAIT line has been detected active before the beginning of a cycle. In that case the OPORT assumes the receiver is unconnected and halts itself indicating the error condition with ERROR line.
SM(0-3): Return the OPORT state machine status.

External interface:

PRMOUT: Enable next device onto the token passing logical ring.
PERMIN: Indicates the token has been received.
STROBE: Signals active data on the pipeline latches, the rising edge is used to strobe that data into the personality card.
WAIT: This input stops the output data stream either in Event or Processor Controlled Modes.

The OPORT is controlled from the CPU by programming the 8 bit control register in the OPORT interface. Refer to output port controller interface for programming codes and status conditions.

2.3.3.1 OUTPUT PORT Controller Interface

The output port controller interface (OPO_INTF) allows the CPU to control and monitor the output port microsequencer (OPORT). The CPU sets up the OPORT in one of the 5 defined modes and receives OPORT status information. The OPO_INTF also drives an interrupt to the CPU system under some OPORT conditions.

Configuration:

The OPO_INTF is a byte wide port, placed at address **$4A0000** in the memory map. Its internal architecture consists of one control and two status registers. The control register sets up the OPORT modes allowing it to individually mask the three defined interrupts. The OPORT defined modes are: Inhibit, Event_Mode, Force_Event_Mode, CPU_Mode, Hold, EMPO, PO, OPO_Reset. They are controlled by individual bit settings.

Address	Write Function	Read Function
$4A0000	None	Status register 1 **b3-b0**: State Machine(Table XX) **b4**: Not Implemented **b5**: Permit_in line flag **b6**: STPACK flag **b7**: ERROR flag
$4A0001	Control register **b0-b2**: Mode **b3**: OPO_Res **b4**: Undefined **b5**: CPU_Md_Mask **b6**: Not Implemented **b7**: Error_Mask	Status register 2 **b0-b3**: 4 LSB's of the control register **b4**: Permit_out flag **b5** : STPACK mask status **b6**: Not Implemented **b7**: ERROR mask

The software reset has the same effect over the OPORT that the hardware reset does. The OPORT goes to the power on reset state, and the control FIFOS are cleared. The INTF output lines.are cleared during a reset operation until a command word is written to the INTF command register.

Three interrupt conditions are recognized by the OPO_INTF:
STPACK: indicates the end of a CPU_mode cycle.
INCOMP: signals that a FASTBUS error has occured during the readout of an event.
ERROR: indicates that there is no receiver connected to the FSCC.

The interrupt conditions are also reflected in the status registers, along with the OPORT state machine status. A read from this register clears the interrupt flags.

2.3.3.2 OUTPUT PORT State Machine

OPORT States:

HOLD.....0
WAIT2 1
SETUP..... 2
STEP 3
HOLD1.....4
STPACK1 5
CONT..... 6
ERROR 7
INHIB.....8
EOE 9
HOLD.....10
SDATA..... 11
DATA.....12
STPACK2 ...13
ACTIVE..... 14
WAIT 15

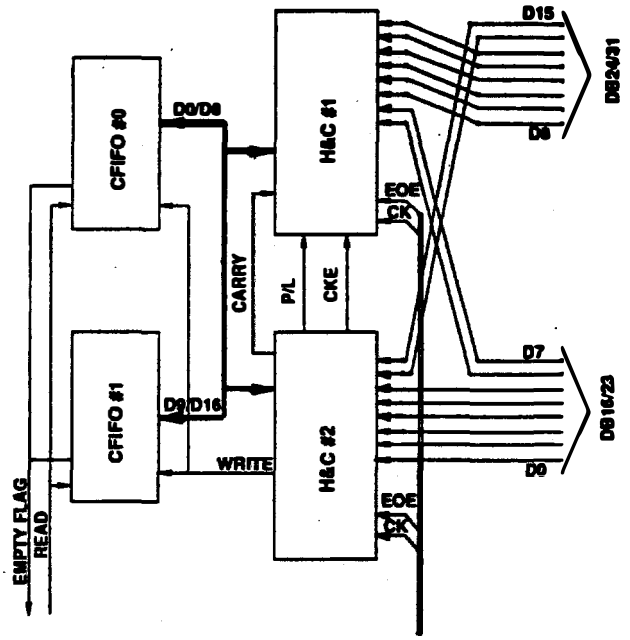
Timing

The OPORT reads the FIFOs and generates the pipeline latch clock, the output data strobe, and the output latch tri-state enable. The timing diagrams are shown in figures 8a and 8b.

2.3.3.3 Total Header and Event Counter Control System

Header and Counter (H&C)

The H&C system keeps a crate wide total count of words read from FASTBUS and transmits this count to a FIFO upon receipt of an EOE (End of Event) signal. The header is a 5 bit word which is directly loaded by the processor into the H&C system. The 12 bit word counter is incremented by a clock signal (CK) coming from the FASTBUS controller. A block diagram is shown in Figure 10.



H&C Block Diagram
Figure 10

Functional Blocks

The H&C system uses two EPB1400 devices. One of them (HEADB) configures the header and the counter's 6 high order bits; the other (HEADA) has the counter's 6 low order bits and the H&C state machine controller. The CPU sees the H&C system as a 16/8 bit peripheral. Four registers can be accessed from the processor bus.

a) Command Register (Write only):
- **RESET**: H&C content are cleared and the state machine is forced to HOLD state.
- **LOAD**: The counter is loaded with the preset register's contents.
- **HOLD**: The count value is kept constant but not cleared.
- **INCREMENT**: Counter increments are enabled and clocked by CK signal.

b) Two counter preset registers.
c) A header register
d) A STATUS register (Read only) which echoes the command register bits and the status of the control state machine.

System Interface

The 12 bit counter has been split in two 6 bit counters. The least significant counter (LSC) generates a CARRY signal to the MSC each time it reaches its maximum value. From the CPU bus, the counters and other H&C registers are accessible in bytes or words.

The internal registers may be selected through select and control lines: H&CSEL*, R/W*, UDS*, LDS* and A1.

A1	UDS	LDS	R/W	Reg. Selected
0	0	0	X	12 bit counter
0	0	1	X	Lower 8 bits of the Counter
0	1	0	X	Upper 4 bits of the counter
1	0	1	X	Header
1	1	0	0	Command
1	1	0	1	Status

The 12 bit counter is placed at **$4E0000** or in consecutive bytes from that address. The lower 12 bits of a 16 bit transfer are loaded into the Preload register during a write cycle.

Address	Write Function	Read Function
$4E0000	Preload register **b0-b11:** Preload count **b12-b15:** Unassigned	Counter Register **b0-b11:** Counter contents **b12-b15:** Unassigned
$4E0002	Header register **b0-b4:** H0-4 Header preload **b5-b7:** Unassigned	Header register **b0-b4:** H0-4 Header contents **b5-b7:** Unassigned
$4E0003	Control register **b0-b1:** S0,S1 Command select field **b2:** RF Software reset bit **b3-b7:** Unassigned	Status register **b0-b1:** S0,S1 Command select field **b2-b5:** Q0-Q3 H&C state status **b6-b7:** Read as zeros

The command modes can be set using the following values:

RESET:	$04
PRELOAD:	$01
HOLD:	$00
TRANSPARENT:	$03
INCRM:	$02

The interface between the H&C controller and the FASTBUS sequencer is through FCLK (counter clock) and CEOE (End of Event) lines. Both are asynchronous to the 20 MHz clock of the state machine. FCLK increments H&C counter when it is in Increment mode CEOE indicates the count is over and the H&C value should be transfered to the CFIFO.

STATE MACHINE

The following states (Figure 11) have been defined in the H&C state machine controller:

OFF LINE: Is the power up state. At this time the control register content is unknown.
HOLD: This is the idle state. The counter value is kept constant.
PRELOAD: The counter's state is initialized with the value found in the counter input
INCREMENT: Enable the FASTBUS clock to increment the counter.
WRITE: Performs a Write memory cycle to the CFIFOs.
CLEAR: Clears the counter after an EOE cycle.
PASS: Enable data written into the counter's input register to be loaded directly into the FIFO memory.
WDAT: Manages PASS mode detecting DS* (data strobe) line active.
DSHIG: Similar to WDAT's function but detects DS* line low-to-high transition.

Interface between HEADA and HEADB

Only one of the two EPB1400s (i.e. HEADA) has the state machine which controls the H&C functioning. It generates signals for controling the other device.

PL: This signal indicates to HEADB that it has to perform a load operation from the input registers.
CKE: Enables HEADB counter increment on CK clock signal.

Internal Signals:

RES and CLR signals are generated for internal use:
RES is a logical OR of the external RESET line, the command register RF bit and the CLEAR state. The RES line is generated in both devices and feeds the counter's macrocells. CLS line is only generated in HEAD2 and clears the MASK bit when the RF bit is set.

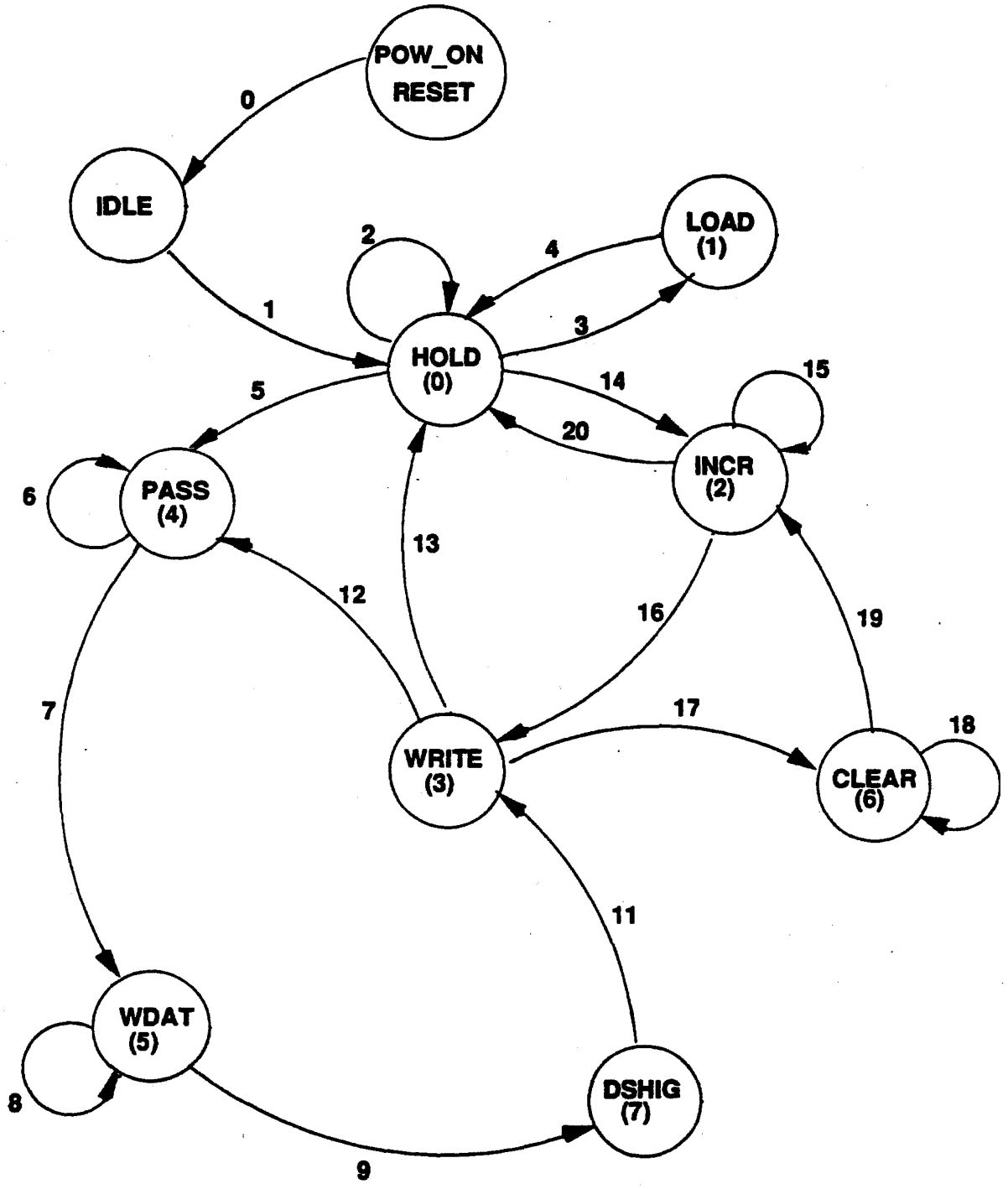
The WRITE signal also functions as a tristate enable for the counter and header outputs during FIFO memory write cycles.

2.3.3.4 OUTPUT PORT Auxiliary Connector Interface

All levels at the output port are single ended TTL. Level adapters to different protocols will be mounted on the FASTBUS auxiliary card. The data path is 32 bits wide. Three control lines are provided to regulate data flow:

WAIT pauses the OPORT
STROBE is a synchronous data strobe

The OPORT generates the pipeline latch clock, the output data strobe, and the output latch tri-state enable. Timing diagrams are shown in figures 8a and 8b.



Header/Counter State Machine diagram
Figure 11

2.3.4 Communication Protocols

The following control line assignments apply to the RS-485 VDAS interface;

AC15-12	not used
AC11	Output Enable
AC10	WAIT
AC09	not used
AC08	STROBE*
AC07-00	not used (Processor controlled I/O)

The following control line assignments apply to the RS-485 RBUF interface;

AC15-13	not used
AC12	Word Select
AC11	Output Enable
AC10	WAIT
AC09	SSTROBE*
AC08	DSTROBE*
AC07-00	not used (Processor controlled I/O)

The following control line assignments apply to the ECLine interface;

AC15-12	not used
AC11	Output Enable*
AC10	WAIT
AC09	End-of-event
AC08	STROBE*
AC07-00	not used (Processor controlled I/O)

The following control line assignments apply to the Fiber-optic interface;

(TBD)

3 System Software Description

3.1 System Software

The operating system is a commercial product of 'Software Components Group, Inc.' which is composed of several related products.

pSOS -68K; A real-time multi-tasking operating system kernel.
pROBE-68K; System debug analyzer.

These combined products form the operating system of the FSCC. The pSOS-pROBE-pRISM operating system is configured at Fermilab for the FSCC hardware implementation. The operating system and diagnostic software are contained in on-board EPROM. A second bank of EPROM is included to contain user programs which may run independently of use services (eg. serial port driver) provided by the operating system. Please refer to "Diagnostics for the FASTBUS Smart Crate Controller", see appendix D.

3.2 Initialization

A reset is generated at power on or from the front panel reset, the FSCC responds by performing initializing all ports & controllers to a passive configuration and sending the following prompt to the serial port:

```
PROBE V3.14 (68020)
COPYRIGHT 1986, SOFTWARE COMPONENTS GROUP INC.
ALL RIGHTS RESERVED

pROBE>
```

The following items 1-4, will need to be set by the user on the first power on cycle. As the values are contained in NVRAM this operation need only be done once unless a change is required.

1. Set the real time clock.
2. Set Ethernet Address
3. Set FASTBUS ID
4. Terminal and Host ports are configurable by writing to the NVRAM at address 30000E as follows:

Byte Hex	bit 6	bit 5	
00	0	0	Host and terminal functions use TERM port.
20	0	1	Host function at HOST port and terminal function at TERM port
40	1	0	Host function uses the TERM port and terminal function uses the HOST port.
60	1	1	Host and terminal functions use the HOST port.

These bits should be selective-set as other bits in the byte may be used for other purposes. These bits will normally be set to 20 Hex when testing is completed.

3.3 PROBE Initialization

Because PROBE is designed to be independent of the hardware implementation it doesn't directly access any hardware devices (other than the microprocessor itself). The only device it accesses indirectly is the serial port. The boot code in the EPROM provides the routines to pROBE that it uses to access the serial port. This boot code also initializes whatever other hardware on the board is not put into a known or usable state by RESET. Once pROBE is called, it loops on polling for input from the serial port. If there is a problem with the serial port hardware, pROBE may not be able to communicate at all.

The boot code was originally debugged using XRAY, emulating the serial port.

3.4 PROBE Memory Map

System ROM for the FSCC nominally occupies 64K bytes of EPROM bank 1. It consists of four sections at offsets that are fixed, i.e., hardwired into the boot code source file and intended to remain invariant for all future versions. These four sections and their offsets (from the start of ROM) are:

Section 1	Offset	Function
	0	boot code
Section 2	1000	pSOS code
	2700	dummy pSOS configuration table
	2762	dummy pSOS I/O jump table
Section 3	2800	pROBE code
Section 4	CA00	pRISM code (not used on the FSCC)

The first 4K bytes are allocated for boot code, configuration tables, the pROBE serial port driver, etc. The boot code must be specific to each hardware platform. Following this are: the code for pSOS, together with a default configuration table and I/O jump table (about 6 KB); the code for pROBE (about 40 KB); and the code for pRISM (about 13 KB). Within the boot code for the FSCC, the layout is as follows:

Offset	
0	reset vector (initial SP and PC)
8	boot data (flags, identification text in ASCII)
3C	default boot configuration table
AC	boot code proper (initial PC points here)
2A4	dummy pSOS ROOT process
2A8	pROBE serial port driver
40A	prototype system configuration table
---	prototype pRISM configuration table
---	prototype pRISM initialization table
49A	prototype pROBE configuration table

At boot time (when the board is RESET), the prototype configuration tables are used to construct corresponding tables in RAM, after which pROBE is started up. The application program that is downloaded contains a pSOS configuration table, which is specific to that application; the location of this table is patched into the system configuration table as part of the download. Then, when pSOS is started (by the pROBE 'gs' command), it finds the application and runs it. The dummy pSOS tables and ROOT process allow pSOS to be started without an application downloaded; it just traps to pROBE.

4 System and Module Diagnostics

4.1 Initial Hardware Inspection

The incoming inspection will be as described in "Diagnostics for the FASTBUS Smart Crate Controller".

4.2 Diagnostic Tests

- Memory Test
  - RAM
  - ROM
- Reset Test
  - Front Panel Switch
  - Front Panel Connector
  - Power On
  - FASTBUS
- Serial Port
- Interrupt Controller
- Ethernet Interface
  - Loopback
  - Dual-Port RAM
  - Channel Atten.
- Output Port
  - Output Port Control Registers
  - Header Counter Registers
  - Control Fifo
  - Data Out to Aux. Interrupt.
  - Auxiliary Parallel Port
  - Data Transfer
    - Normal Frame Mode
    - Processor Controlled Mode
  - Permit in/Permit out
- Front Panel Output Port
- Front Panel Trigger Port
- FASTBUS functions.
  - Parallel ports
    - Port 1 Registers
    - Port 2 Registers
    - Check Status & Control bits
  - FPORT Controller Instructions
  - All FASTBUS operations
  - Short and Long Timers
  - FASTBUS Error Handling
  - List Fifo
  - Data Transfers
- Local Word Counter
- Data Fifo
- Processor Fifo
- Real Time Clock
  - Set and Read Clock
  - Set and Read Ethernet Address
- General Hardware exerciser runs all of the Diagnostics

4.3 Diagnostic Operating Instructions

The following sequence is executed to start the diagnostic program. This loads the diagnostic from EPROM into the RAM space of the FSCC. Once in the diagnostic menu most commands are self explanatory. For procedures and detailed descriptions of the tests refer to "Diagnostics for the FASTBUS Smart Crate Controller".

```
pROBE>go 10000

Exception. Level=7 F/O=00A4 (TRAP #9) Running: **** NO pSOS ****
-----------------------------------------------------------------
SR=2700-tfSm.111...xnzvc USP=00000000 MSP=00200800 ISP=00200800
VBR=00200000 SFC=0 DFC=0 CACR=00 CAAR=00000000
DR=00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000
AR=00000000 00000000 00200E80 002007F8  00002D54 00200DE0 00007F56
PC=00010036-00010036  FFFF          DC.W   $FFFF

pROBE>gs

pSOS Initialized. Running: 'SPRS' -#002262E4
-----------------------------------------------------------------
SR=1000-tfsM.000...xnzvc USP=0023F262 MSP=0023F0D2 ISP=00225C44
VBR=00200000 SFC=0 DFC=0 CACR=00 CAAR=00000000
DR=FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
AR=FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF  00210AB2 0021E308 00000000
PC=002129CC-002129CC  4EBACF46        JSR    $0020F914(PC)

pROBE>go

      FSCC
Diagnostic Menu    key: -  not available
      ====================        .. not complete
       1) Memory test
       2) Reset test
       3) Serial Port
       4)- Interrupt Controller
       5) Ethernet Interface
       6) Output Port /Data FIFO
       7) Trigger Port
       8) FASTBUS
       9) Local Word Counter
      10) Processor FIFO
      66) Setup Default test Slave  PAD (hex: 12)
      77) Loop on a menu item switch (OFF)
      88) General Hardware Exerciser Test
      99) Exit to pROBE
Enter Command:
```

Appendix A - FPORT Controller Instruction Set

BUS_ARBITRATE	$0062 0300

Description: Arbitrate for FASTBUS using the low byte of the data operand. Bits 0-5 supply the arbitration vector. Bit 7 enables assured access mode. Bit 6 (prioritized access mode) is ignored. Note that the data operand is a long word and is normally identical to the value of CSR 8.

Example Syntax: MOVE.L CSR_8, BUS_ARBITRATE

Operation:

```
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    {goto INTS;}
    elseif  IGK(FRDY)  return processor acknowledge A; FDSACK*⇓
    else                   {request bus; FREQ⇑
                           goto C4;}
C5: return processor acknowledge B; FDSACK*⇓
C6: Delay Cycle; /*FPORT deselect*/
C7: Delay Cycle;
C8: Delay Cycle;
```

Note: In a multi-master system, the processor should examine the parallel port FSLV*, FRDY and FRAK inputs to confirm that the FSCC has either acquired the bus or been addressed as a slave while attempting to acquire the bus. 0

BUS_RELEASE $060 0004

Description: Release FASTBUS.

Example Syntax: MOVE.L XX, BUS_RELEASE

Operation:

C1: FPORT select;
return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: {DS=0; FCDS⇑
DK=0; FCDK⇑
AS=0; FCAS⇑
release bus;} FREL⇑
C5: Delay Cycle; /*FPORT deselect*/
C6: Delay Cycle;
C7: Delay Cycle;

ADDRESS_DATA_GEOGRAPHICAL $0060 0304

Description: Perform a FASTBUS geographical primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR, ADDRESS_DATA_GEOGRAPHICAL

Operation:

```
C1: FPORT select;
    latch FB_ADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  AK(FRAK)  {reset short timer; TIMER⇓
                      MS=0;
                      exit;}
    else              {AS=1; FSAS⇑
                      RD=0;
                      MS=0;
                      EG=1; FDEG⇑
                      enable short timer;
                      goto C4;} TIMER⇑
```

ADDRESS_CSR_GEOGRAPHICAL $0060 0308

Description: Perform a FASTBUS geographical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_CSR_GEOGRAPHICAL

Operation:

```
C1: FPORT select;
    latch FB_ADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  AK(FRAK)  {reset short timer; TIMER⇓
                      MS=0; FDMS0⇑
                      exit;}
    else              {AS=1; FSAS⇑
                      RD=0;
                      MS=1; FDMS0⇑
                      EG=1; FDEG⇑
                      enable short timer; TIMER⇑
                      goto C4;}
```

ADDRESS_DATA_LOGICAL $0060 030C

Description: Perform a FASTBUS logical primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_DATA_LOGICAL

Operation:

```
C1: FPORT select;
    latch FB_ADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  AK(FRAK)  {reset short timer; TIMER⇓
                      MS=0;
                      exit;}
    else              {AS=1; FSAS⇑
                      RD=0;
                      enable short timer; TIMER⇑
                      goto C4;}
```

ADDRESS_CSR_LOGICAL $0060 0310

Description: Perform a FASTBUS logical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_CSR_LOGICAL

Operation:

```
C1: FPORT select;
    latch FB_ADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  AK(FRAK)  {reset short timer; TIMER⇓
                      MS=0; FDMS0⇓
                      exit;}
    else              {AS=1; FSAS⇑
                      RD=0;
                      MS=1; FDMS0⇑
                      enable short timer; TIMER⇑
                      goto C4;}
```

ADDRESS_DATA_BROADCAST $0060 0314

Description: Perform a FASTBUS broadcast primary address cycle to DATA Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_DATA_BROADCAST

Operation:

```
C1: FPORT select;
    latch FB_ADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  AK(FRAK)  {reset short timer; TIMER⇓
                      MS=0; FDMS1⇓
                      exit;}
    else              {AS=1; FSAS⇑
                      RD=0;
                      MS=2; FDMS1⇑
                      enable short timer;} TIMER⇑
```

ADDRESS_CSR_BROADCAST $0060 0318

Description: Perform a FASTBUS geographical primary address cycle to CSR Space.

Example Syntax: MOVE.L FB_ADDR,ADDRESS_CSR_BROADCAST

Operation:

```
C1: FPORT select;
    latch FB_ADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  AK(FRAK)  {reset short timer; TIMER⇓
                      MS=0; FDMS0⇓, FDMS1⇓
                      exit;}
    else              {AS=1; FSAS⇑
                      RD=0;
                      MS=3; FDMS0⇑, FDMS1⇑
                      enable short timer; TIMER⇑
                      goto C4;}
```

ADDRESS_RELEASE $0060 031C

Description: Release address lock.

Example Syntax: MOVE.L XX,ADDRESS_RELEASE

Operation:

```
C1: FPORT select;
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    else              {DS=0; FCDS⇑
                      DK=0; FCDK⇑
                      MS=0; FDMS0⇓, FDMS1⇓,FDMS2⇓
                      AS=0; FSAS⇑
                      enable short timer;} TIMER⇑

C5: if      IRQ(SEQINT) goto INTF;
    elseif  AK(FRAK)    goto C5;
    else                exit;
```

DATA_PROCESSOR_RANDOM_READ $0062 0320

Description: Perform a FASTBUS single word read data cycle.

Example Syntax: MOVE.L DATA_PROCESSOR_RANDOM_READ,DATA

Operation:

```
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTS;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  DK(FRDK)  return processor acknowledge A; FDSACK*⇓
    else              {
                      enable short timer; TIMER⇑
                      RD=1; FDRD⇑
                      MS=0;
                      DS=1; FSDS⇑
                      goto C4;}
C5: reset short timer; TIMER⇓
    return processor acknowledge B; FDSACK*⇓
C6: if      IRQ(SEQINT)    INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C6;}
    elseif  !DK(FRDK*)     reset short timer; TIMER⇓
    else              {enable short timer; TIMER⇑
                      RD=0; FDRD⇓
                      MS=0;
                      DS=0; FCDS⇑
                      goto C6;}
C7: delay cycle; /* processor deselect */
C8: delay cycle;
C9: delay cycle;
```

DATA_PROCESSOR_RANDOM_WRITE $0060 0324

Description: Perform a FASTBUS single word write data cycle.

Example Syntax: MOVE.L DATA,DATA_PROCESSOR_RANDOM_WRITE

Operation:

```
C1: FPORT select;
    latch DATA; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  DK(FRDK)  reset short timer; TIMER⇓
    else              {enable short timer; TIMER⇑
                      RD=0;
                      MS=0;
                      DS=1; FSDS⇑
                      goto C4;}
C5: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C5;}
    elseif  !DK(FRDK*)exit;
    else              {enable short timer; TIMER⇑
                      RD=0;
                      MS=0;
                      DS=0; FCDS⇑
                      goto C5;}
```

DATA_PROCESSOR_SEC_ADDRESS_READ $0062 0328

Description: Perform a FASTBUS secondary address read cycle.

Example Syntax: MOVE.L DATA_PROCESSOR_SEC_ADDRESS_READ,SADDR

Operation:

```
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTS;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  DK(FRDK)  return processor acknowledge A; FDSACK*⇓
    else              {enable short timer; TIMER⇑
                      RD=1; FDRD⇑
                      MS=2; FDMS1⇑
                      DS=1; FSDS⇑
                      goto C4;}
C5: reset short timer; TIMER⇓
    return processor acknowledge B; FDSACK*⇓
C6: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C6;}
    elseif  !DK(FRDK*)     {reset short timer; TIMER⇓
    else              {enable short timer; TIMER⇑
                      RD=0; FDRD⇓
                      MS=2; FDMS1⇑
                      DS=0; FCDS⇑
                      goto C6;}
C7: delay cycle;      /* processor deselect */
C8: delay cycle;
C9: delay cycle;
```

DATA_PROCESSOR_SEC_ADDRESS_WRITE $0060 032C

Description: Perform a FASTBUS secondary address write cycle.

Example Syntax:

```
MOVE.L
    SADDR,DATA_PROCESSOR_SEC_ADDRESS_WRITE
```

Operation:

```
C1: FPORT select;
    latch SADDR; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  DK(FRDK)  reset short timer; TIMER⇓
    else              {enable short timer; TIMER⇑
                      RD=0;
                      MS=2; FDMS1⇑
                      DS=1; FSDS⇑
                      goto C4;}
C5: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C5;}
    elseif  !DK(FRDK*)      exit;
    else              {enable short timer; TIMER⇑
                      RD=0;
                      MS=0; FDMS1⇓
                      DS=0; FCDS⇑
                      goto C5;}
```

DATA_PROCESSOR_BLOCK_TRANSFER_READ $0062 0008

Description: Perform one step of a FASTBUS block transfer read cycle.

Example Syntax:

```
MOVE.L
DATA_PROCESSOR_BLOCK_TRANSFER_READ,DATA
```

Operation:

```
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTS;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    elseif  ! DS(FRDS*)     {enable short timer; TIMER⇑
                      RD=1; FDRD⇑
                      MS=1; FDMS1⇑
                      DS=1; FSDS⇑
                      goto C6;}
    else              {enable short timer; TIMER⇑
                      RD=1; FDRD⇑
                      MS=1; FDMS0⇑
                      DS=0;} FCDS⇑
C5: if      IRQ(SEQINT)     goto INTS;
    elseif  ! DK(FRDK*)     {return processor acknowledge A; FDSACK*⇓
                      reset short timer; TIMER⇓
                      goto C7;}
    else              goto C5;
C6: if      IRQ(SEQINT)     goto INTS;
    elseif  DK(FRDK)  {return processor acknowledge A; FDSACK*⇓
                      reset short timer;} TIMER⇓

    else              goto C6;
C7: return processor acknowledge B; FDSACK*⇓
C8: delay cycle;        /* processor deselect */
C9: delay cycle;
C10: delay cycle;
C11: delay cycle;
```

DATA_PROCESSOR_BLOCK_TRANSFER_WRITE $0060 000C

Description: Perform one step of a FASTBUS block transfer write cycle.

Example Syntax:

```
MOVE.L
DATA,DATA_PROCESSOR_BLOCK_TRANSFER_WRITE
```

Operation:

```
C1: FPORT select;
    latch DATA; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      RD=0;
                      MS=1; FDMS0⇑
                      goto C4;}
    elseif  ! DS(FRDS*)     {enable short timer; TIMER⇑
                      RD=0;
                      MS=1; FDMS0⇑
                      DS=1; FSDS⇑
                      goto C6;}
    else              {enable short timer; TIMER⇑
                      RD=0;
                      MS=1; FDMS0⇑
                      DS=0;} FCDS⇑
C5: if      IRQ(SEQINT)     goto INTF;
    elseif  ! DK(FRDK*)     {reset short timer; TIMER⇓
                      RD=0;
                      MS=0; FDMS0⇓
                      exit;}
    else              goto C5;
C6: if      (IRQ)     goto INTF;
    elseif  (DK)      {reset short timer; TIMER⇓
                      RD=0;
                      MS=0; FDMS0⇓
                      exit;}
    else              goto C6;
```

DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE $0060 0330

Description: Perform termination step of a FASTBUS block transfer.

Example Syntax:

MOVE.Lxx,DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE

Operation:

```
C1: FPORT select;
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  goto C4;
    else    {MS=0; FDMS0⇓, FDMS1⇓, FDMS2⇓
            RD=0; FDRD⇓
            DS=0;} FCDS⇑
C5: delay cycle;
C6: delay cycle;
C7: delay cycle;
```

DATA_FIFO_BLOCK_TRANSFER_READ $0060 0334

Description: Perform a FASTBUS block transfer read to the Data FIFO.

Example Syntax: MOVE.L xx,DATA_FIFO_BLOCK_TRANSFER_READ

Operation:

```
C1: FPORT select;
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  {reset short timer; TIMER⇓
                      goto C4;}
    else              {enable FIFO; DFIFOEN⇑,PFIFOEN⇑
                      RD=1; FDRD⇑
                      MS=1; FDMS0⇑
                      DS=1; FSDS⇑
                      local counter mode = decrement; LC0⇑, LC1⇑
                      clock global word counter;} FCLK⇑
C5: {local counter mode = hold; LC0⇓, LC1⇓
    enable short timer;} TIMER⇑
C6: continue;                          /* delay cycle */
C7: if      ((FRDK*FRSS1)#SEQINT #FEOB)
    goto C11;
    elseif  WT(FRWT)  {reset short timer;  LC0⇑, LC1⇑
                      goto C7;}
    elseif  DK(FRDK)  {DS=0; FCDS⇑
                      local counter mode = decrement; LC0⇑, LC1⇑
                      reset short timer; TIMER⇓
                      clock global word counter;} FCLK⇑
    else              {enable short timer; TIMER⇑
                      goto C7;}
C8: {local counter mode = hold; LC0⇓, LC1⇓     /* delay cycle */
    enable short timer;} TIMER⇑
C9: delay cycle;
C10: if     ((!FRDK*FRSS1)#SEQINT#FEOB)
    goto C11;
    elseif  WT(FRWT)  {reset short timer;
                      goto C9;}
```

```
    elseif    !DK(!FRDK)      {DS=1; FSDS⇑
                              local counter mode=decrement; LC0⇑, LC1⇑
                              reset short timer; TIMER⇓
                              clock global word counter; FCLK⇑
                              goto C5;}
    else                      goto C9;
C11: if       SS1(FRSS1)  clock global word counter; FCLK⇑
/* block transfer termination */
C12: if       IRQ(SEQINT)     goto INTF;
     elseif   DS(FRDS)    continue;
     else                 goto C22;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
/* data written to FIFO on DK down is dummy word with EOE flag*/
C13:if        IRQ(SEQINT)     goto INTF;
    elseif    DK(FRDK)   {reset short timer; TIMER⇓
                              MS=0;} FDMS0⇓
    else                      {enable short timer; TIMER⇑
                              goto C13;}
C14:continue;                           /* delay cycle */
C15:continue;                           /* delay cycle */
C16:continue;                           /* delay cycle */
C17:if        IRQ(SEQINT)     goto INTF;
    elseif    WT(FRWT)   {reset short timer; TIMER⇓
                              goto C17;}
    else                      {enable short timer; TIMER⇑
                              clock global word counter; FCLK⇑
                              DS=0;} FCDS⇑
C18:if        IRQ(SEQINT)     goto INTF;
    elseif    ! DK(! FRDK)    {RD=0; FDRD⇓
                              reset short timer;} TIMER⇓
    else                      {enable short timer; TIMER⇑
                              goto C18;}
C19:continue;                           /* delay cycle */
C20:continue;                           /* delay cycle */
C21:exit;
/* termination routine for even word count transfer */
C22:if        IRQ(SEQINT)     goto INTF;   /* check that DK is low */
    elseif    ! DK(! FRDK)    {RD=0; FDRD⇓
                              reset short timer;} TIMER⇓
    else                      {enable short timer; TIMER⇑
                              goto C20;}
C23:continue;                           /* delay cycle */
C24:continue;                           /* delay cycle */
```

```
C25:continue;                    /* delay cycle */
C25:exit;
```

DATA_FIFO_PIPELINED_READ_100 $0060 0338

Description: Perform a FASTBUS pipelined read to the Data FIFO at 100 nsec/word.

Example Syntax: MOVE.L xx,DATA_FIFO_PIPELINED_READ_100

Operation:

```
C1: FPORT select;
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  goto C4;
    elseif  EOB(FEOB) goto PIPELINE_TERMINATE;
    else            {enable FIFO; PFFEN⇑, DFFEN⇑
                    RD=1; FDRD⇑
                    MS=3; FDMS0⇑,FDMS1⇑
                    DS=1; FSDS⇑
                    local counter mode = decrement; LC0⇑,LC1⇑
                    clock global word counter;} FCLK⇑
C5: local counter mode = hold; LC0⇓,LC1⇓
C6: if      IRQ(SEQINT)    goto INTF;
    elseif  WT(FRWT)  goto C6;
    elseif  EOB(FEOB) goto PIPELINE_TERMINATE;
    else            {DS=0; FCDS⇑
                    local counter mode = decrement; LC0⇑,LC1⇑
                    clock global word counter;} FCLK⇑
C7: {local counter mode = hold; LC0⇓,LC1⇓
    goto C4;}
```

DATA_FIFO_PIPELINED_READ_200 $0060 033C

Description: Perform a FASTBUS pipelined read to the Data FIFO at 200 nsec/word.

Example Syntax: MOVE.L xx,DATA_FIFO_PIPELINED_READ_200

Operation:

```
C1: FPORT select;
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  goto C4;
    elseif  EOB(FEOB) goto PIPELINE_TERMINATE;
    else            {enable FIFO; PFFEN⇑, DFFEN⇑
                    RD=1; FDRD⇑
                    MS=3; FDMS0⇑,FDMS1⇑
                    DS=1; FSDS⇑
                    local counter mode = decrement; LC0⇑,LC1⇑
                    clock global word counter;} FCLK⇑
C5: local counter mode = hold; LC0⇓,LC1⇓
C6: continue;                   /* delay cycle */
C7: continue;                   /* delay cycle */
C8: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  goto PIPELINE_TERMINATE;
    elseif  EOB(FEOB) goto C12;
    else            {DS=0; FCDS⇑
                    local counter mode = decrement; LC0⇑,LC1⇑
                    clock global word counter;} FCLK⇑
C9: local counter mode = hold; LC0⇓,LC1⇓
C10:continue;                   /* delay cycle */
C11:goto C4;                    /* delay cycle */
```

DATA_FIFO_PIPELINED_READ_400 $0060 0340

Description: Perform a FASTBUS pipelined read to the Data FIFO at 400 nsec/word.

Example Syntax: MOVE.L xx,DATA_FIFO_PIPELINED_READ_400

Operation:

```
C1: FPORT select;
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  goto C4;
    elseif  EOB(FEOB) goto PIPELINE_TERMINATE;
    else            {enable FIFO; PFFEN⇑, DFFEN⇑
                    RD=1; FDRD⇑
                    MS=3; FDMS0⇑,FDMS1⇑
                    DS=1; FSDS⇑
                    local counter mode = decrement; LC0⇑,LC1⇑
                    clock global word counter;} FCLK⇑
C5: local counter mode = hold; LC0⇓,LC1⇓
C6: continue;                   /* delay cycle */
C7: continue;                   /* delay cycle */
C8: continue;                   /* delay cycle */
C9: continue;                   /* delay cycle */
C10:continue;                   /* delay cycle */
C11:continue;                   /* delay cycle */
C12:if      IRQ(SEQINT)     goto INTF;
    elseif  WT(FRWT)  goto C12;
    elseif  EOB(FEOB) goto PIPELINE_TERMINATE;
    else            {DS=0; FCDS⇑
                    local counter mode = decrement; LC0⇑,LC1⇑
                    clock global word counter;} FCLK⇑
C13:local counter mode = hold; LC0⇓,LC1⇓
C14:continue;                   /* delay cycle */
C15:continue;                   /* delay cycle */
C16:continue;                   /* delay cycle */
C17:continue;                   /* delay cycle */
C18:continue;                   /* delay cycle */
C19:goto C4;                    /* delay cycle */
```

NULL	$0060 0020

Description: Access the FPORT Controller without performing any operation. Confirms that the FPORT Controller is active. If the FPORT Controller is stalled, NULL will cause a Processor BUS ERROR interrupt.

Example Syntax: MOVE.L xx,NULL

Operation:

C1: FPORT select;
return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: exit;

Note: If FASTBUS WT is asserted, the FPORT Controller will wait indefinitely for a slave response. A WT timeout will only be generated (after the long timeout period) if the processor attempts another FASTBUS instruction while the first operation is pending. In cases where the processor does not access the FPORT Controller for extended periods of time (e.g., a standalone microcode readout loop) an occasional NULL instruction will detect a FASTBUS lockup condition. NULL will also guarantee that all pending FASTBUS operations have been completed by clearing the instruction pipeline.

LOCAL_COUNTER_LOAD $0062 0010

Description: Initialize Local Word Counter for block transfer.

Example Syntax: MOVE.L COUNT,LOCAL_COUNTER_LOAD

Operation:

C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return processor acknowledge A; FDSACK*⇓
local counter mode = load;} LC1⇑
C5: {return processor acknowledge B; FDSACK*⇓
local counter mode = hold;} LC1⇓
C6: delay cycle; /* processor deselect */
C7: delay cycle;
C8: delay cycle

Note: The counter is loaded from the low order 12 bits of COUNT.

LOCAL_COUNTER_READ $0062 0014

Description: Read current value of Local Word Counter.

Example Syntax: MOVE.L LOCAL_COUNTER_READ,COUNT

Operation:

C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return processor acknowledge A; FDSACK*⇓
local counter mode = read;} SLCOE⇑
C5: return processor acknowledge B; FDSACK*⇓
C6: delay cycle; /* processor deselect */
C7: delay cycle;
C8: delay cycle;

Note: The counter is returned in the low order 12 bits of COUNT.

FIFO_WRITE	$0062 0018

Description: Write a single word from the processor to the Data FIFO.

Example Syntax: MOVE.L DATA,FIFO_WRITE

Operation:

C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {return processor acknowledge A; FDSACK*⇓
clock FIFO input;} SDW⇑
C5: {return processor acknowledge B; FDSACK*⇓
clock global word counter;} FCLK⇑
C6: delay cycle; /* processor deselect */
C7: delay cycle;
C8: delay cycle;

EOE	$0060 0024

Description: Write dummy word to the data FIFO with the End-Of-Event bit set.

Example Syntax: MOVE.L XX,EOE

Operation:

C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: {set EOE flag to output controller; CEOE⇑
set EOE flag to data FIFO;} DEOE⇑
C5: delay cycle;
C6: delay cycle;
C7: delay cycle;

SLAVE_DATA_INPUT $0062 001C

Description: Transfer one data word from FASTBUS to the processor in slave mode.

Example Syntax: MOVE.L SLAVE_DATA_INPUT,DATA

Operation:

```
C1: FPORT select;
C2: FPORT instruction fetch;
C3: instruction dispatch;
C4: if      IRQ(SEQINT)  goto INTS;
    elseif  WT(FRWT)     goto C4;
    elseif  DS(FRDS)     {return processor acknowledge A; FDSACK*⇓
                         clear WT;} FCWT⇑
    else                 {return processor acknowledge A; FDSACK*⇓
                         clear WT; FCWT⇑
                         goto C6;}
C5: {return processor acknowledge B; FDSACK*⇓
    DK=1; FSDK⇑
    goto C7;}
C6: {return processor acknowledge B; FDSACK*⇓
    DK=0;} FCDK⇑
C7: delay cycle;
C8: delay cycle;
C9: delay cycle;
```

SLAVE_DATA_OUTPUT $0060 0344

Description: Transfer one data word from the processor to FASTBUS in slave mode.

Example Syntax: MOVE.L DATA,SLAVE_DATA_OUTPUT

Operation:

```
C1: FPORT select;
    latch DATA; DCPBA⇑
    return processor acknowledge A; FDSACK*⇓
C2: FPORT instruction fetch;
    return processor acknowledge B; FDSACK*⇓
C3: instruction dispatch;
C4: if      IRQ(SEQINT)   goto INTF;
    elseif  WT(FRWT)      got0 C4;
    elseif  DS(FRDS)      {clear WT; FCWT⇑
                          DK=1; FSDK⇑
                          exit;}
    else                  {clear WT; FCWT⇑
                          DK=0; FCDK⇑
                          exit;}
```

PIPELINE_TERMINATE Internal Subroutine

Description: Internal routine to terminate pipelined transfers.

Example Syntax:

Operation:

```
C1: if        SS1(FRSS1)  clock global work coutner; FCLK⇑
C2: if        IRQ(SEQINT)     goto INTF;
    elseif    DS(FRDS)    continue; TIMER⇑
    else                  goto C8;
/* termination routine for odd word count transfer */
/* check that DK is high */
/* set DS low and wait for DK low */
C3: if        IRQ(SEQINT)     goto INTF;
    else      ! DK(! FRDK)    goto C3; TIMER⇑
C4: continue;                     /* delay cycle */
C5: continue;                     /* delay cycle */
C6: if        IRQ(SEQINT)     goto INTF;
    elseif    WT(FRWT)  goto C6;
    else              {clock global word counter; FCLK⇑
                      DS=0;} FCDS⇑
C7: if        IRQ(SEQINT)     goto INTF;
    elseif    ! DK(! FRDK)    {RD=0; FDRD⇓
                              exit;
    else              goto C7; TIMER⇑
/* termination routine for even word count transfer */
/* check that DK is low */
C8:if         IRQ(SEQINT)     goto INTF;
    elseif    ! DK(! FRDK)    reset short timer; TIMER⇑
    else              goto C8;
C9:continue;
C10:continue;
C11:exit;
```

INTS	Internal Subroutine

Description: Internal routine to abort instruction on error interrupt.
INTS assumes that processor acknowledge has not yet been returned.

Operation:

C1: return processor acknowledge A; FDSACK*⇓

C2: return processor acknowledge B; FDSACK*⇓
goto INTF;

INTF	Internal Subroutine

Description: Internal routine to abort instruction on error interrupt.
INTF assumes that processor acknowledge has already been returned.
Exit the current FASTBUS operation by returning all signals to inactive state.

Operation:

```
C1: {disable tranceivers;
     AS=0; FCAS⇑
     DS=0; FCDS⇑
     DK=0;} FCDK⇑
C2: delay cycle;        /* processor deselect */
C3: exit;
```

Appendix B - FSCC Parts List

Description	Quan	Manf.	Manf. #		
3.6864 MHz Crystal	1	CTS	MP037-3.6864 MHz		
.01uF Non-Polarized Capacitor	3				
.057" FUSE Sockets	20	AUGAT	8134-HC-6P2		
.1uF 25V Non-Polarized Capacitor	5				
.1uF Ceramic,Dip Cap	118				
Coaxial LEMO Connector, K-Loc	3	Kings	1077-3		
10 Pin Sip PAK, 100 Ohms	3	Bourns	4310R-101-101		
10 Pin Sip Socket	2	Samtec	SL-110-G-19		
10 Pin Sip, 100 Ohms, 5 Individual Res.	2	Bourns	4310R-102-101		
10/32 Shoulder Screws	2	R.A.F. Electt.	7039-SS MOD E=1/2		
100pF Non-Polarized Capacitor	1				
10KH ECL PAL	2	TI	PAL1016P8JC		
16 Pin Dip Socket, 300 mils wide	2	Samtec	ICO-316-SGG		
1K Ohm, 10-Pin Sip PAK	2	Bourns	4310R-101-102		
1N914 Diode	1	Motorola	1N914		
20 Pin Dip Socket, 300 mils wide	1	Samtec	ICO-320-SGG		
20.000 MHz Crystal	1	CTS	MP200-20MHz		
20pF Non-Polarized Capacitor	2				
22V10 Reprogramable AND-OR Array	14	Lattice	GAL22V10-15LP		
24 Pin Dip Socket, 300 mils wide	16	Samtec	ICO-324-SGG		
28 Pin Dip Socket, 300 mils wide	7	Preci-Dip	110-99-328-41-001		
28 Pin Dip Socket, 600 mils wide	30	Samtec	ICO-628-SGG		
2K X 16-Bit, Dual-Port RAM (Master)	1	IDT	IDT7133S70G		
2K X 16-Bit, Dual-Port RAM (Slave)	1	IDT	IDT7143S70G		
32-Bit Microprocessor (20 MHz)	1	Motorola	MC68020RC20		
32K X 8-Bit CMOS 70ns Static RAM	8	Sony	CXK58258P-45		
4-Pin LEMO Connector	2	LEMO	RA0.304NYL		
4/40 X 1/4" Bind Head Screw	4				
40 MHz Oscillator	1	CTS	MX055GA-2C-40 Mhz		
40 Pin Dip Socket, 600 mils wide	6	Samtec	ICO-640-SGG		
48 Pin Dip Socket, 600 mils wide	1	Samtec	ICA-648-SGG		
5-Bit Magnitude Comparator	1	Motorola	MC10H166		
5pF Non-Polarized Capacitor	2				
5V-IN,9V-OUT,DC-DC Converter	1	Reliability	2VP5U9		
6.8uF Polarized Capacitor	3				
64K X 8 Bit CMOS EPROM	8	AMD	AM27C512-120		
68 Pin Grid Array	2	Samtec	MVAS-68-ZSGG-11		
75uH, Pulse Transformer	1	Datatronics	PT10312		
9-Bit D-Type Edge-Triggered Flip	1	Signetics	N74F823N		
Dialight Green LED with Integral RES	1	Dialight	558-0202-003		
Dialight Yellow LED with Integral RES	1	Dialight	558-0302-003		
Dual D-Type Master-Slave Flip-Flop	2	Motorola	MC10H131		

Duart	1	Signetics	SCN68681C1N40		
ECL/TTL Octal Transceiver/Translator	9	Brooktree	BT501KC		
Ethernet Serial Interface	1	Intel	P82C501-10MHz		
Ethernet Transceiver Chip	1	National	DP8392A		
FASTBUS Front Panel Mounting Bracket	2	FNAL Drawing	0882-MB-199070		
FASTBUS Module Auxilliary Connector	1	AMP	534974-9		
FASTBUS Module Segment Connector	1	AMP	1-102585-3		
FSCC Front Panel	1				
FSCC P.C. Board	1				
HEX 2-Input or Drivers	1	TI	SN74AS832BN		
Local Area Network Coprocessor	1	Intel	P82586-10MHz		
Male 20-Pin 100M X 100M Dip Header	2	3M	3428-5302		
Multi-Function Peripheral	1	Motorola	MC68901		
NE555 Precision Timer	1	RCA	LM555CN		
Octal Bidirectional Transceiver, 3-state	4	Signetics	N74F545N		
Octal Buffer and Driver, 3-State	3	Signetics	N74F1244		
Octal Bus Transceiver, 3-State	4	Signetics	N74F646N		
Octal D-Type Edge-Triggered Flip-Flops	3	Signetics	N74F825N		
Octal D-Type Flip-Flop w/3-State Outputs	2	Signetics	N74F574N		
Isolated BNC Solder Jack Assembly	1	AMP	227726-1		
Parallel 2048 X 9-Bit FIFO	4	Sharp	LH5498-35		
Parallel 256 X 9-Bit FIFO	10	Sharp	LH5496-35		
Parallel Interface/Timer	2	Motorola	MC68230P10		
Subminiature Fuse	10	Pico	251.005		
Quad 2-Input or Gate	1	Motorola	MC10H103		
Quadruple Differential Line Driver	1	Fairchild	UA96174		
Quadruple Differential Line Receiver	2	Fairchild	UA96175		
Resistor 1.5K 1/8W 5%	4				
Resistor 100K 1/8W 5%	1				
Resistor 10K 1/8W 5%	1				
Resistor 150 1/8W 5%	1				
Resistor 1K 1/8W 5%	16				
Resistor 1K 1/8W 1%	1				
Resistor 1M 1/8W 5%	4				
Resistor 20 1/4W 5%	10				
Resistor 220 1/8W 5%	5				
Resistor 240 1/4W 5%	2				
Resistor 39 1/8W 5%	6				
RS232 Driver/Receiver	1	Maxim	MAX233C		
SPDT Push Button Switch	1	C&K	8121-S-D-A6-G-E		
Stand-Alone Microsequencer	4	Altera	EPS488DC-25		
User-Configurable uP Peripheral	6	Altera	EPB1400DC		
Watchdog Timekeeper, R/T Clock	1	Dallas Semi.	DS1286		
114 Pin Grid Array	1	Samtec	CPAS-114-ZSGG-13A		

Appendix C - PLD Equations

ADD_REC.ABL-8/25/89

```
MODULE ADDRESS_RECOGNITION
TITLE 'ADDRESS_RECOGNITION    VERSION 25-AUG-89 0930 hrs'

ADD_REC  DEVICE  'EC16P8N';


AS              PIN     1;
AD7             PIN     2;
AD6             PIN     3;
AD5             PIN     9;
MS2             PIN     10;
MS1             PIN     11;
AK              PIN     13;
EG              PIN     14;
AI              PIN     15;
IREQ            PIN     16;
IREL            PIN     21;
EAI             PIN     22;
IGK             PIN     23;

D               PIN     7;
CLK             PIN     18;
R               PIN     20;
!AS_DELAY       PIN     4;
!GA_MATCH       PIN     8;
IAR             PIN     17;


EQUATIONS

"       Address Recognition

D               =       !AK & !MS1 & !MS2 & !AD7 & !AD6 & !AD5 & GA_MATCH;

R               =       !AS;

CLK             =       EG & AS_DELAY;

AS_DELAY        =       AS;

"       Arbitration Latch

IAR             =       (IAR & !IREL & !IGK) # (IREQ & (!EAI # !AI));

END ADDRESS_RECOGNITION
```

BERR.ABL-2/15/90

```
module bus_err
title 'bus error generator
      Gustavo Cancelo,  Fermilab,  4-18-89'
"       Changed polarity of LTO 1-SEP-89 Rick Kwarciany
"       Added EXTRES to Reset equations  15-Feb-90  RK

      berr    device    'p22v10';

" bus error controller inputs/outputs
CLK,AS                    pin  1,2;
ADDERR,!LTO               pin  3,5;
BERR                      pin  23;
Q2,Q1,Q0                  pin  17,18,19;

" reset generator i/o
!EXTRES                         pin 8;
WDTO                      pin 9;
PUSHB,PWRUP               pin 10,11;
RESETI                    pin 13;
TRIG                      pin 14;
RESET                     pin 16;
RESETN                    pin 15;

L,H,C,X = 0,1,.C.,.X.;

Q2,Q1,Q0 ISTYPE 'reg_d,feed_reg';

"buser  =   [Q2,Q1,Q0];

"IDLE=0;  BER1=1;   BER3=3;  BER4=4;   BER5=5;

equations

Q2 := !ADDERR & !LTO & !AS & !Q2 & !Q1 & !Q0
   #  Q2 & !Q1 & !Q0
   #  Q2 & !Q1 & Q0 & !AS;

Q1 :=  !Q2 & !Q1 & Q0
   #   !Q2 & Q1 & Q0 & LTO;

Q0 :=  !Q2 & !Q1 & !Q0 & !AS & LTO
   #   !Q2 & !Q1 & Q0
   #   !Q2 &  Q1 & Q0 & LTO
   #   Q2 & !Q1 & !Q0
   #   Q2 & !Q1 & Q0 & !AS;

!BERR := !Q2 & !Q1 & !Q0 & !AS & LTO
      #  !Q2 & !Q1 & Q0
      #  !Q2 & !Q1 & !Q0 & !LTO & !AS & !ADDERR
      #  Q2 & !Q1 & !Q0;

!TRIG = PUSHB # !PWRUP # !WDTO;

RESET = RESETI + EXTRES;
!RESETN = RESETI + EXTRES;
```

```
test_vectors

 ([CLK,AS,ADDERR,LTO,Q2,Q1,Q0] -> [Q2,Q1,Q0,BERR])
  [ C, H,   L,    L, L, L, L ] -> [L, L, L,  H  ];
  [ C, H,   X,    X, L, L, L ] -> [L, L, L,  H  ];
  [ C, L,   X,    H, L, L, L ] -> [L, L, H,  L  ];
  [ C, X,   X,    X, L, L, H ] -> [L, H, H,  L  ];
  [ C, X,   X,    H, L, H, H ] -> [L, H, H,  H  ];
  [ C, X,   X,    L, L, H, H ] -> [L, L, L,  H  ];
  [ C, L,   L,    L, L, L, L ] -> [H, L, L,  L  ];
  [ C, X,   X,    X, H, L, L ] -> [H, L, H,  L  ];
  [ C, L,   X,    X, H, L, H ] -> [H, L, H,  H  ];
  [ C, H,   X,    X, H, L, H ] -> [L, L, L,  H  ];

test_vectors

 ([PUSHB,PWRUP,WDTO] -> [TRIG])
  [  L,    H,   H  ] -> [ H  ];
  [  H,    H,   H  ] -> [ L  ];
  [  L,    L,   H  ] -> [ L  ];
  [  L,    H,   L  ] -> [ L  ];

end bus_err_
```

CLOCKS.ABL-11/11/89

```
module clock
title 'FSCC PC2 VERSION 11-NOV-89 0800'
clocks device 'P22V10';

OSC,OSC2        pin   1,2;
CLK1            pin   22;
CLK2A           pin   23;
CLK2B           pin   21;
CLK2C           pin   16;
CLK3            pin   20;
CLK4            pin   17;
CLK5           pin   19;
CLK2N           pin   18;

CLK = [CLK5,CLK4,CLK3,CLK2A];

equations

CLK1 = OSC2;

CLK := CLK - 1;

CLK2B := !CLK2A;

CLK2C := !CLK2A;

CLK2N := CLK2A;

END
```

DECO1.ABL-3/24/90

```
module decoder_1
title 'Fastbus Readout Controller (FSCC), decoder map
Gustavo Cancelo,  Fermilab, 4/3/89'
"Changed ADDERR to allow access to 500000-57FFFF(LPBK,and ETHRES) 2-11-89 RK
"Changed ADDERR to not allow access to 420000-43FFFF(Former LOOPS) 2-11-89 RK
"Changed ADDERR to allow access to 6C0000-6DFFFF now FBUSRS.  24-3-90  RK

          deco1    device    'p22V10';

AS,RW,FC1,FC0                                pin     2,4,5,6;
A22,A21,A20,A19,A18,A17                      pin     7,8,9,10,11,13;
NVDS,LTEN                                    pin     17,15;
ADDERR                                       pin     18;
IACK,TMRS                                    pin     19,14;
ROM2S,ROM1S                                  pin     20,21;
RAM2S,RAM1S                                  pin     22,23;

L,H,X,Z = 0,1,.X.,.Z.;
ADDRESS = [X,A22,A21,A20,A19,A18,A17,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X];
ROM1LOW =  ^h000000;
ROM1HIGH = ^h03FFFF;
ROM2LOW =  ^h040000;
ROM2HIGH = ^h07FFFF;

RAM1LOW =  ^h200000;
RAM1HIGH = ^h21FFFF;
RAM2LOW =  ^h220000;
RAM2HIGH = ^h23FFFF;

TMRSLOW = ^h480000;
"TMRSHIGH = ^h49FFFF;
NVDLOW = ^h300000;
NVDHIGH = ^h31FFFF;

RES1LOW = ^h080000;
RES1HIGH = ^h0FFFFF;
RES2LOW = ^h100000;
RES2HIGH = ^h17FFFF;
RES3LOW = ^h180000;
RES3HIGH = ^h1FFFFF;

RES4LOW = ^h240000;
RES4HIGH = ^h2FFFFF;

RES5LOW = ^h320000;
RES5HIGH = ^h3FFFFF;

RES6LOW = ^h420000;
RES6HIGH = ^h43FFFF;

RES7LOW = ^h580000;
RES7HIGH = ^h5FFFFF;

RES8LOW = ^h6E0000;
RES8HIGH = ^h6FFFFF;
RES9LOW = ^h700000;
```

```
RES9HIGH = ^hFFFFFF;

ADCODE = [A19,A18,A17];
INTERRUPT = 7;
FCODE = [FC1,FC0];
CPU_SPACE = 3;

equations
!RAM1S=(FCODE != CPU_SPACE)&(ADDRESS >= RAM1LOW)&(ADDRESS <= RAM1HIGH)&!AS;

!RAM2S=(FCODE != CPU_SPACE)&(ADDRESS >= RAM2LOW)&(ADDRESS <= RAM2HIGH)&!AS;

!ROM1S=(FCODE != CPU_SPACE)&(ADDRESS >= ROM1LOW)&(ADDRESS <=
ROM1HIGH)&!AS&RW;

!ROM2S=(FCODE != CPU_SPACE)&(ADDRESS >= ROM2LOW)&(ADDRESS <=
ROM2HIGH)&!AS&RW;

!TMRS =(FCODE != CPU_SPACE)&(ADDRESS == TMRSLOW) & !AS;

!IACK= (FCODE == CPU_SPACE) & (ADCODE == INTERRUPT) & !AS & A22;

"          (ADDRESS >= ROM2LOW)&(ADDRESS <= ROM2HIGH) & !AS
"        # (ADDRESS >= RAM2LOW)&(ADDRESS <= RAM2HIGH) & !AS
!ADDERR = (ADDRESS >= RES1LOW)&(ADDRESS <= RES3HIGH) & !AS
        # (ADDRESS >= RES4LOW)&(ADDRESS <= RES4HIGH) & !AS
        # (ADDRESS >= RES5LOW)&(ADDRESS <= RES5HIGH) & !AS
        # (ADDRESS >= RES6LOW)&(ADDRESS <= RES6HIGH) & !AS
        # (FCODE!= CPU_SPACE)&(ADDRESS >= RES7LOW)&(ADDRESS<= RES7HIGH) & !AS
        # (FCODE!= CPU_SPACE)&(ADDRESS >= RES8LOW)&(ADDRESS<=RES9HIGH) & !AS;

!NVDS = (FCODE != CPU_SPACE)&(ADDRESS >= NVDLOW)&(ADDRESS <= NVDHIGH) & !AS;

LTEN = !AS;

test_vectors
([RW,FC1,FC0,ADDRESS,AS]->[RAM1S,RAM2S,ROM1S,ROM2S,TMRS,IACK,ADDERR,NVDS,LTEN])
[X, H,  L ,^h000000,H ]->[ H,    H,    H,    H,    H,    H,    H,    H,    L ];
[L, H,  L ,^h000000,L ]->[ H,    H,    H,    H,    H,    H,    H,    H,    H ];
[H, H,  L ,^h000000,L ]->[ H,    H,    L,    H,    H,    H,    H,    H,    H ];
[H, H,  L ,^h01FFFF,L ]->[ H,    H,    L,    H,    H,    H,    H,    H,    H ];
[H, L,  H ,^h040000,L ]->[ H,    H,    H,    L,    H,    H,    H,    H,    H ];
[X, L,  H ,^h200400,L ]->[ L,    H,    H,    H,    H,    H,    H,    H,    H ];
[X, L,  H ,^h220400,L ]->[ H,    L,    H,    H,    H,    H,    H,    H,    H ];
[X, L,  H ,^h22FFFF,L ]->[ H,    L,    H,    H,    H,    H,    H,    H,    H ];
[X, H,  L ,^h480000,L ]->[ H,    H,    H,    H,    L,    H,    H,    H,    H ];
[X, H,  L ,^h080000,L ]->[ H,    H,    H,    H,    H,    H,    L,    H,    H ];
[X, H,  L ,^h1FFFFF,L ]->[ H,    H,    H,    H,    H,    H,    L,    H,    H ];
[X, L,  H ,^h300000,L ]->[ H,    H,    H,    H,    H,    H,    H,    L,    H ];
[X, L,  H ,^h500000,L ]->[ H,    H,    H,    H,    H,    H,    H,    H,    H ];
[X, L,  H ,^h700000,L ]->[ H,    H,    H,    H,    H,    H,    L,    H,    H ];
[X, H,  H ,^hFF0000,L ]->[ H,    H,    H,    H,    H,    L,    H,    H,    H ];

END
```

DECO2.ABL-11/1/89

```
module decoder_2 flag '-tl'
title 'Fastbus Readout Controller (FSCC), decoder map
Gustavo Cancelo,  Fermilab, 4/3/89'
"Changed polarity of CAS (positive true)  1-NOV-89  RK
"Added software reset of Ethernet coprocessor  1-NOV-89  RK
"Changed LOOPS to LPBK        1-NOV-89  RK

        deco2    device    'p22V10';

CLK                                        pin     1;
AS,FC1,FC0                                 pin     2,5,6;
A22,A21,A20,A19,A18,A17                    pin     7,8,9,10,11,13;
UARTS                                      pin     19;
CAS,LPBK                                   pin     20,14;
ETHS                                       pin     22;
ETHRES                                  pin     16;
RESET                                   pin     15;
ETHRES1                                         pin
        17;

L,H,X,Z,C = 0,1,.X.,.Z.,.C.;
ADDRESS = [X,A22,A21,A20,A19,A18,A17,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X];

ETHLOW =  ^h400000;
ETHHIGH = ^h41FFFF;

LPBKC = ^h500000;
LPBKS = ^h520000;

ETHRESS = ^h540000;
ETHRESC = ^h560000;

CASLOW = ^h440000;
UARTSLOW = ^h460000;

ADCODE = [A19,A18,A17];
INTERRUPT = 7;
FCODE = [FC1,FC0];
CPU_SPACE = 3;

LPBK    ISTYPE 'REG_D,FEED_REG';
ETHRES1 ISTYPE 'REG_D,FEED_REG';

equations

!ETHS=  (FCODE != CPU_SPACE) & (ADDRESS >= ETHLOW) & (ADDRESS <=
ETHHIGH)&!AS;

CAS = (FCODE != CPU_SPACE) & (ADDRESS == CASLOW) & !AS;

!UARTS = (FCODE != CPU_SPACE) & (ADDRESS == UARTSLOW) & !AS;

ETHRES1 := (ADDRESS == ETHRESS) & !ETHRES1
        #  (ADDRESS != ETHRESC) & ETHRES1;

ETHRES = ETHRES1 # RESET;
```

```
LPBK := (ADDRESS == LPBKC) & !LPBK
     # (ADDRESS != LPBKS) & LPBK;


END
```

DECO3.ABL-3/24/90

```
module decoder_3
title 'Fastbus Readout Controller (FSCC), decoder map
Gustavo Cancelo,  Fermilab, 4/3/89
FB2S added by RK 4-AUG-89
FBUSRS added at address 6C0000 by RK  24-May-90'

          deco3    device    'p22V10';

CLK                                        pin     1;
AS,DS,RW,FC1,FC0                           pin     2,3,4,5,6;
A22,A21,A20,A19,A18,A17                    pin     7,8,9,10,11,13;
HCSEL,PARLLS,OPORTS,HCSELI                 pin     14,15,16,18;
FIFO1S,FIFO2S                              pin     20,19;
FSSEL,FCSEL,FB1S,FB2S                      pin     21,22,23,17;

L,H,X,Z,C = 0,1,.X.,.Z.,.C.;
ADDRESS = [X,A22,A21,A20,A19,A18,A17,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X];

FB1LOW =  ^h600000;
FB1HIGH = ^h61FFFF;
FB2LOW =  ^h620000;
FB2HIGH = ^h63FFFF;
FCLOW =   ^h640000;
FCHIGH =  ^h65FFFF;
FSLOW =   ^h660000;
FSHIGH =  ^h67FFFF;
FBUSRS =  ^h6C0000;

OPORTLW = ^h4A0000;
"OPORTHG = ^h4BFFFF;

PARLLOW = ^h4C0000;
HCLOW = ^h4E0000;

FIFO1LOW = ^h680000;
FIFO1HG = ^h69FFFF;
FIFO2LOW = ^h6A0000;
FIFO2HG = ^h6BFFFF;

ADCODE = [A19,A18,A17];
INTERRUPT = 7;
FCODE = [FC1,FC0];
CPU_SPACE = 3;

HCSELI istype 'reg_d,feed_reg';
HCSEL istype 'reg_d';

equations

!FB1S= (FCODE != CPU_SPACE) & ((ADDRESS == FB1LOW) # (ADDRESS == FBUSRS)) &
!AS;

!FB2S= (FCODE != CPU_SPACE) & ((ADDRESS == FB2LOW) # (ADDRESS == FBUSRS)) &
!AS;
```

```
!FCSEL= (FCODE != CPU_SPACE) & (ADDRESS >= FCLOW) & (ADDRESS <= FCHIGH) & !AS
& !DS;

!FSSEL= (FCODE != CPU_SPACE) & (ADDRESS >= FSLOW) & (ADDRESS <= FSHIGH) & !AS
& !DS;

!OPORTS= (FCODE != CPU_SPACE) & (ADDRESS == OPORTLW)&!AS;

!PARLLS = (FCODE != CPU_SPACE)&(ADDRESS == PARLLOW) & !AS;

HCSELI := (FCODE != CPU_SPACE)&(ADDRESS == HCLOW) & !AS;

!HCSEL := HCSELI & !AS;

!FIFO1S = (FCODE != CPU_SPACE)&(ADDRESS >= FIFO1LOW)&(ADDRESS <= FIFO1HG) &
!AS & RW;

!FIFO2S = (FCODE != CPU_SPACE)&(ADDRESS >= FIFO2LOW)&(ADDRESS <= FIFO2HG) &
!AS & !RW;

test_vectors

([CLK,RW,DS,FC1,FC0,ADDRESS,AS] ->
[FB1S,FCSEL,FSSEL,FIFO1S,FIFO2S,OPORTS,PARLLS,HCSELI,HCSEL])
 [ C, X, X,  H,  L, ^h600000, H ] -> [ H,     H,    H,    H,     H,     H,
H,    L,      H  ];
 [ C, X, X,  H,  L, ^h600000, L ] -> [ L,     H,    H,    H,     H,     H,
H,    L,      H  ];
 [ C, X, X,  L,  H, ^h640000, L ] -> [ H,     L,    H,    H,     H,     H,
H,    L,      H  ];
 [ C, X, X,  L,  H, ^h660000, L ] -> [ H,     H,    L,    H,     H,     H,
H,    L,      H  ];
 [ C, X, X,  L,  H, ^h4A0000, L ] -> [ H,     H,    H,    H,     H,     L,
H,    L,      H  ];
 [ C, X, X,  L,  H, ^h4C0000, L ] -> [ H,     H,    H,    H,     H,     H,
L,    L,      H  ];
 [ C, H, X,  L,  H, ^h680000, L ] -> [ H,     H,    H,    L,     H,     H,
H,    L,      H  ];
 [ C, L, X,  L,  H, ^h680000, L ] -> [ H,     H,    H,    H,     H,     H,
H,    L,      H  ];
 [ C, H, X,  L,  H, ^h6A0000, L ] -> [ H,     H,    H,    H,     H,     H,
H,    L,      H  ];
 [ C, L, X,  L,  H, ^h6A0000, L ] -> [ H,     H,    H,    H,     L,     H,
H,    L,      H  ];
 [ C, X, X,  L,  H, ^h4E0000, L ] -> [ H,     H,    H,    H,     H,     H,
H,    H,      H  ];
 [ C, X, X,  L,  H, ^h4E0000, L ] -> [ H,     H,    H,    H,     H,     H,
H,    H,      L  ];

END
```

DSGEN.ABL-12/11/89

```
module dsgen
title 'Fastbus Readout Controller (FSCC), data strobe generator
Gustavo Cancelo,  Fermilab, 6/14/89'
"Changed FASTBUS addresses 7-AUG-89, RK
"Changed H&C Select from 0 wait states to 1 wait state. 9-Oct-89 RK.
"Added LPBKS, and ETHRES  1-NOV-89  RK
"Added ROM2, and RAM2, to DSACK1 equation   11-DEC-89   RK.

          dsgen    device    'p22V10';

CLK                                        pin     1;
AS,DS,RW,FC1,FC0                           pin     2,3,4,5,6;
A22,A21,A20,A19,A18,A17                    pin     7,8,9,10,11,13;
BUSYL                                      pin     14;
FBDSACK                                    pin     23;
FDSYNC                                     pin     16;
DSACK0,DSACK1                              pin     18,19;
TMRACK                                     pin     15;
WS0,WS1,WS2                                pin     20,21,22;

L,H,X,Z,C = 0,1,.X.,.Z.,.C.;
WS0,WS1,WS2 istype 'reg_d,feed_reg';
FDSYNC istype 'reg_d';

ADDRESS = [X,A22,A21,A20,A19,A18,A17,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X];
ROM1LOW =  ^h000000;          "32 bits - 1WS
ROM1HIGH = ^h03FFFF;
ROM2LOW =  ^h040000;          "32 bits - 1WS
ROM2HIGH = ^h07FFFF;

RAM1LOW =  ^h200000;          "32 bits - 0WS
RAM1HIGH = ^h21FFFF;
RAM2LOW =  ^h220000;          "32 bits - 0WS
RAM2HIGH = ^h23FFFF;

NVDLOW = ^h300000;            "8bits - 2WS
NVDHIGH = ^h310000;

TMRSLOW = ^h480000;           "8 bits - ASYNCH
"TMRSHIGH = ^h49FFFF;

ETHLOW =  ^h400000;           "32 bits - 1WS
ETHHIGH = ^h41FFFF;

LPBKS = ^h500000;          "16 bits - 0WS
LPBKC = ^h520000;

ETHRESS = ^h540000;         "16 bits - 0WS
ETHRESC = ^h560000;

CASLOW = ^h440000;            "32 bits - 1WS
UARTSLOW = ^h460000;          "8 bits - 3WS

FBLOW =   ^h600000;           "32 bits - 1WS on Writes/0WS on Reads
FBHIGH =  ^h63FFFF;
FCLOW =   ^h640000;           "8 bits - 1WS
```

```
FCHIGH =  ^h65FFFF;
FSLOW =   ^h660000;          "8 bits - 1WS
FSHIGH =  ^h67FFFF;

OPORTLW = ^h4A0000;          "8 bits - 0WS
"OPORTHG = ^h4BFFFF;

PARLLOW = ^h4C0000;          "8 bits - 0WS
HCLOW = ^h4E0000;           "16 bits - 1WS

FIFO1LOW = ^h680000;         "32 bits - 0WS
FIFO1HG = ^h69FFFF;
FIFO2LOW = ^h6A0000;         "32 bits - 0WS
FIFO2HG = ^h6BFFFF;

ADCODE = [A19,A18,A17];
INTERRUPT = 7;
FCODE = [FC1,FC0];
CPU_SPACE = 3;

equations
"         WS0.RE = AS;
"         WS1.RE = AS;
"         WS2.RE = AS;
"         WS3.RE = AS;

         WS0 := !AS;

         WS1 := WS0 & !AS;

         WS2 := WS1 & !AS;

FDSYNC := (!FBDSACK # FDSYNC) * !AS;

!DSACK0 = (ADDRESS >= RAM1LOW) & (ADDRESS <= RAM1HIGH) & !AS
        # (ADDRESS >= ROM1LOW) & (ADDRESS <= ROM1HIGH) & WS1 & !AS
        # (ADDRESS >= RAM2LOW) & (ADDRESS <= RAM2HIGH) & !AS
        # (ADDRESS >= ROM2LOW) & (ADDRESS <= ROM2HIGH) & WS1 & !AS
        # (ADDRESS >= ETHLOW) & (ADDRESS <= ETHHIGH) & !AS & BUSYL & WS1
        # (ADDRESS == CASLOW) & WS1 & !AS
        # (ADDRESS >= FIFO1LOW) & (ADDRESS <= FIFO1HG) & !AS
        # (ADDRESS >= FIFO2LOW) & (ADDRESS <= FIFO2HG) & !AS
        # (ADDRESS == OPORTLW) & !AS
        # (ADDRESS == PARLLOW) & !AS
        # FDSYNC
        # (ADDRESS >= NVDLOW) & (ADDRESS <= NVDHIGH) & !AS & WS2
        # !TMRACK;

!DSACK1 = (ADDRESS >= RAM1LOW)&(ADDRESS <= RAM1HIGH) & !AS
        #  (ADDRESS >= RAM2LOW)&(ADDRESS <= RAM2HIGH) & !AS
        #  (ADDRESS >= ROM1LOW)&(ADDRESS <= ROM1HIGH) & WS1 & !AS
        #  (ADDRESS >= ROM2LOW)&(ADDRESS <= ROM2HIGH) & WS1 & !AS
        #  (ADDRESS >= ETHLOW) & (ADDRESS <= ETHHIGH) & !AS & BUSYL & WS1
        #  (ADDRESS == LPBKS) & !AS
        #  (ADDRESS == LPBKC) & !AS
        #  (ADDRESS == ETHRESS) & !AS
        #  (ADDRESS == ETHRESC) & !AS
```

```
        #   (ADDRESS == CASLOW) & WS1 & !AS
        #   (ADDRESS >= FIFO1LOW)&(ADDRESS <= FIFO1HG) & !AS
        #   (ADDRESS >= FIFO2LOW)&(ADDRESS <= FIFO2HG) & !AS
        #   (ADDRESS == HCLOW) & WS1 & !AS
        # FDSYNC;

test_vectors
([CLK, ADDRESS,AS,BUSYL,TMRACK,FBDSACK]->[WS0,WS1,WS2,DSACK1,DSACK0,FDSYNC])
 [ C, ^h000000, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];
 [ C, ^h000000, L,  H,    H,      H   ]->[ H,  L,  L,   H,     H,     L   ];
 [ C, ^h000000, L,  H,    H,      H   ]->[ H,  H,  L,   L,     L,     L   ];
 [ C, ^h000000, L,  H,    H,      H   ]->[ H,  H,  H,   L,     L,     L   ];
 [ C, ^h000000, L,  H,    H,      H   ]->[ H,  H,  H,   L,     L,     L   ];
 [ C, ^h000000, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];
 [ C, ^h200400, L,  H,    H,      H   ]->[ H,  L,  L,   L,     L,     L   ];
 [ C, ^h200400, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];
 [ C, ^h081400, L,  H,    H,      H   ]->[ H,  L,  L,   H,     H,     L   ];
 [ C, ^h081400, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];

[  C, ^h300000, L,  H,    H,      H   ]->[ H,  L,  L,   H,     H,     L   ];
[  C, ^h300000, L,  H,    H,      H   ]->[ H,  H,  L,   H,     H,     L   ];
[  C, ^h300000, L,  H,    H,      H   ]->[ H,  H,  H,   H,     L,     L   ];
[  C, ^h300000, L,  H,    H,      H   ]->[ H,  H,  H,   H,     L,     L   ];
[  C, ^h300000, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];

"[CLK,ADDRESS,AS,BUSYL,TMRACK,FBDSACK] -> [WS0,WS1,WS2,DSACK1,DSACK0,FDSYNC])
[  C, ^h480000, L,  H,    H,      H   ] -> [ H,  L,  L,   H,     H,     L   ];
[  C, ^h480000, L,  H,    L,      H   ] -> [ H,  H,  L,   H,     L,     L   ];
[  C, ^h480000, H,  H,    H,      H   ] -> [ L,  L,  L,   H,     H,     L   ];
[  C, ^h440000, L,  H,    H,      H   ] -> [ H,  L,  L,   H,     H,     L   ];
[  C, ^h440000, L,  H,    H,      H   ] -> [ H,  H,  L,   L,     L,     L   ];
[  C, ^h440000, L,  H,    H,      H   ] -> [ H,  H,  H,   L,     L,     L   ];
[  C, ^h440000, H,  H,    H,      H   ] -> [ L,  L,  L,   H,     H,     L   ];
[  C, ^h600000, L,  H,    H,      H   ] -> [ H,  L,  L,   H,     H,     L   ];
[  C, ^h600000, L,  H,    H,      H   ] -> [ H,  H,  L,   H,     H,     L   ];
[  C, ^h600000, H,  H,    H,      H   ] -> [ L,  L,  L,   H,     H,     L   ];
[  C, ^h600000, L,  H,    H,      L   ] -> [ H,  L,  L,   L,     L,     H   ];
[  C, ^h600000, H,  H,    H,      H   ] -> [ L,  L,  L,   H,     H,     L   ];
[  C, ^h680000, L,  H,    H,      H   ] -> [ H,  L,  L,   L,     L,     L   ];
[  C, ^h680000, H,  H,    H,      H   ] -> [ L,  L,  L,   H,     H,     L   ];
[  C, ^h4A0000, L,  H,    H,      H   ] -> [ H,  L,  L,   H,     L,     L   ];
[  C, ^h4A0000, H,  H,    H,      H   ] -> [ L,  L,  L,   H,     H,     L   ];

"[CLK, ADDRESS,AS,BUSYL,TMRACK,FBDSACK]->[WS0,WS1,WS2,DSACK1,DSACK0,FDSYNC])
[  C, ^h640000, L,  H,    H,      H   ]->[ H,  L,  L,   H,     H,     L   ];
[  C, ^h640000, L,  H,    H,      H   ]->[ H,  H,  L,   H,     H,     L   ];
[  C, ^h640000, L,  H,    H,      H   ]->[ H,  H,  H,   H,     H,     L   ];
[  C, ^h640000, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];
[  C, ^h660000, L,  H,    H,      H   ]->[ H,  L,  L,   H,     H,     L   ];
[  C, ^h660000, L,  H,    H,      H   ]->[ H,  H,  L,   H,     H,     L   ];
[  C, ^h660000, L,  H,    H,      H   ]->[ H,  H,  H,   H,     H,     L   ];
[  C, ^h660000, H,  H,    H,      H   ]->[ L,  L,  L,   H,     H,     L   ];

END
```

EPB.ADF-11/7/89

```
module epb
Gustavo Cancelo
Fermilab
10/9/89
title mux_demux and timing generator (2)

PART:EPB1400

INPUTS: S1@8,S0@36,Q2@13,Q1@12,Q0@25,AD1@9,CLK@7,CLK2@14

OUTPUTS:
ADT8@26,ADT9@27,ADT10@28,ADT11@29,ADT12@32,ADT13@33,ADT14@34,ADT15@35
         AD8@24,AD9@23,AD10@22,AD11@21,AD12@20,CS2@19,RW1@18,RW2@17
         DB8@4,DB9@3,DB10@2,DB11@1,DB12@40,DB13@39,DB14@38,DB15@37
         OME1@5,OME2@6,S1L@16,S0L@15

NETWORK:

S1=INP(S1)
S0=INP(S0)
Q2=INP(Q2) %Q2,Q1,Q0: epb control lines. refer to epbtemp %
Q1=INP(Q1)
Q0=INP(Q0)
AD1=INP(AD1)
CLK=INP(CLK)
CLK2=INP(CLK2)

RW1=CONF(RW1c,)  %R/!W to dual port memory%
RW2=CONF(RW2c,)  %R/!W to dual port memory%
OME1=CONF(OME1c,)    %!OE to dual port memory%
OME2=CONF(OME2c,)    %!OE to dual port memory%
CS2=CONF(CS2c,)      %!CS to dual port memory #2 %
S0L,S0L=RORF(S0Ld,CLK2,,,) %latchs 82586 S0 line%
S1L,S1L=RORF(S1Ld,CLK2,,,) %latchs 82586 S1 line%

%high add/data byte tranceiver w/tri_state enable%
ADT8,ADT9,ADT10,ADT11,ADT12,ADT13,ADT14,ADT15 = BUSX (IBUS,,OE)

%high address byte input register w/tri_state enable%
OR10,OR11,OR12,OR13,OR14,OR15,OR16,OR17 = RBUSI (IBUS,,WE1)

%high data byte input register w/tri_state enable 2 %
OR20,OR21,OR22,OR23,OR24,OR25,OR26,OR27 = RBUSI (IBUS,,WE2)

%high data byte output register 2 %
IBUS = LBUSO (DB8,DB9,DB10,DB11,DB12,DB13,DB14,DB15,,OLE,RE)

%high address byte external output w/tri_state enable%
AD8 =  CONF (AD8c,)
AD9 =  CONF (AD9c,)
AD10 = CONF (AD10c,)
AD11 = CONF (AD11c,)
AD12 = CONF (AD12c,)

%high data byte external output w/tri_state enable%
DB8,DB8   = COIF (DB8c,HAB2)
```

```
DB9,DB9   = COIF (DB9c,HAB2)
DB10,DB10 = COIF (DB10c,HAB2)
DB11,DB11 = COIF (DB11c,HAB2)
DB12,DB12 = COIF (DB12c,HAB2)
DB13,DB13 = COIF (DB13c,HAB2)
DB14,DB14 = COIF (DB14c,HAB2)
DB15,DB15 = COIF (DB15c,HAB2)


EQUATIONS:

S1Ld = !Q2 * S1
     +  Q2 * S1L;

S0Ld = !Q2 * S0
     +  Q2 * S0L;

OE = S0L * !S1L * (Q2 * !Q1 * Q0
                + Q2 * Q1 * !Q0
                + Q2 * Q1 * Q0);  %tranceiver enable%

WE1 = !(!S1 * S0 * !Q2 * !Q1 * Q0
    + S1 * !S0 * !Q2 * !Q1 * Q0);   %latchs add. at rising edge%

WE2 = !(S1 * !S0 * !Q2 * Q1 * Q0);             %latchs data at rising edge%


HAB2 = S1L * !S0L;   %tri_state data out. enable%

OLE = !S1L * S0L * Q2 * !Q1 * Q0;   %latchs data when high%

RE = !S1L * S0L * (Q2 * !Q1 * Q0
                 + Q2 * Q1 * !Q0
                 + Q2 * Q1 * Q0); %out. enable in read cycles%


%dual port RAM selection%

CS2c = !(!S1L * S0L * AD1 *  Q2 * !Q1 * !Q0
        +!S1L * S0L * AD1 *  Q2 * !Q1 * Q0
        + S1L * !S0L * AD1 * Q2 * !Q1 * !Q0
        + S1L * !S0L * AD1 * Q2 * !Q1 * Q0);

%dual port RAM read/write cycle%
RW1c = !(S1L * !S0L * !AD1 * Q2 * !Q1 * !Q0
       + S1L * !S0L * !AD1 * Q2 * !Q1 * Q0);

RW2c = !(S1L * !S0L * AD1 * Q2 * !Q1 * !Q0
       + S1L * !S0L * AD1 * Q2 * !Q1 * Q0);

%dual port RAM output enable%
OME1c = !(!S1L * S0L * !AD1 * Q2 * !Q1 * !Q0
        + !S1L * S0L * !AD1 * Q2 * !Q1 * Q0);

OME2c = !(!S1L * S0L * AD1 * Q2 * !Q1 * !Q0
        + !S1L * S0L * AD1 * Q2 * !Q1 * Q0);
```

```
AD8c = OR10;
AD9c = OR11;
AD10c = OR12;
AD11c = OR13;
AD12c = OR14;

DB8c = OR20;
DB9c = OR21;
DB10c = OR22;
DB11c = OR23;
DB12c = OR24;
DB13c = OR25;
DB14c = OR26;
DB15c = OR27;


END$_
```

EPBTEMP.ADF-7/27/90

```
module prueba
Gustavo Cancelo
Fermilab
7/19/89
title mux_demux and timing generator
%Modified, 7-27-90 to correct state machine operation when busy%
%is true.  Added BUSY wait loop to state 13. RK%

PART:EPB1400

INPUTS: CLK@7,BUSY@13,S1L@12,S0L@25,S1@9,S0@8,CLK2@14,HOLD@36

OUTPUTS: Q0@5,Q1@6,Q2@15,Q3@16
         ADT0@26,ADT1@27,ADT2@28,ADT3@29,ADT4@32,ADT5@33,ADT6@34,ADT7@35
         CS1@24,AD1@23,AD2@22,AD3@21,AD4@20,AD5@19,AD6@18,AD7@17
         DB0@4,DB1@3,DB2@2,DB3@1,DB4@40,DB5@39,DB6@38,DB7@37

NETWORK:

CLK=INP(CLK)
CLK2=INP(CLK2)
BUSY=INP(BUSY) %from dual port RAM BUSY contention line%
S1=INP(S1)     %S1,S0: read/write cycle indicators%
S0=INP(S0)
S0L=INP(S0L)
S1L=INP(S1L)
HOLD=INP(HOLD)

Q3,Q3 = TOTF(Q3t,CLK2,,,)
Q2,Q2 = TOTF(Q2t,CLK2,,,)   %Q2,Q1,Q0: internal counter for timing generator%
Q1,Q1 = TOTF(Q1t,CLK,,,)
Q0,Q0 = TOTF(Q0t,CLK,,,)

%82586 add/data bus interface%
ADT0,ADT1,ADT2,ADT3,ADT4,ADT5,ADT6,ADT7 = BUSX (IBUS,,OE)

%low address byte input latch. RBUSI_A(1)%
OR10,OR11,OR12,OR13,OR14,OR15,OR16,OR17 = RBUSI (IBUS,,WE1)

%low data byte input latch. RBUSI_A(2)%
OR20,OR21,OR22,OR23,OR24,OR25,OR26,OR27 = RBUSI (IBUS,,WE2)

%low data byte output latch. LBUSO(2)%
IBUS = LBUSO (DB0,DB1,DB2,DB3,DB4,DB5,DB6,DB7,,OLE,RE1)

% este registro no se usa%
%IBUS = LBUSO (AD0,AD1,AD2,AD3,AD4,AD5,AD6,AD7,,OLE2,RE2)%

%low address byte at external pins w/tri_state habilitation line%
AD1,AD1 = COIF (AD1c,)
AD2,AD2 = COIF (AD2c,)
AD3,AD3 = COIF (AD3c,)
AD4,AD4 = COIF (AD4c,)
AD5,AD5 = COIF (AD5c,)
AD6,AD6 = COIF (AD6c,)
AD7,AD7 = COIF (AD7c,)
```

```
%low data byte at external pins w/tri_state habilitation line%
DB0,DB0 = COIF (DB0c,HAB2)
DB1,DB1 = COIF (DB1c,HAB2)
DB2,DB2 = COIF (DB2c,HAB2)
DB3,DB3 = COIF (DB3c,HAB2)
DB4,DB4 = COIF (DB4c,HAB2)
DB5,DB5 = COIF (DB5c,HAB2)
DB6,DB6 = COIF (DB6c,HAB2)
DB7,DB7 = COIF (DB7c,HAB2)

CS1 = CONF(CS1c,)

EQUATIONS:
%Read mode: wait_state insertion if BUSY active & count == 13 %
%Write mode: wait_state insertion if BUSY active & count == 13 %

          Q0t = !(!HOLD * !Q3 * !Q2 * !Q1 * !Q0
               +   HOLD * !Q3 * !Q2 * !Q1 * !Q0  %S1 Y S0 DON'T CARE%
               +   HOLD *  Q3 * !Q2 * !Q1 * !Q0 * S1 * S0
               +  !HOLD *  Q3 * !Q2 * !Q1 * !Q0
               +  !HOLD *  Q3 * !Q2 *  Q1 * !Q0
               +   HOLD *  Q3 * !Q2 *  Q1 * !Q0 * S1 * S0
               +  !BUSY *  Q3 *  Q2 * !Q1 *  Q0);
        %added busy to state 13%

          Q1t =  Q3 * !Q2 *  !Q1 * Q0 * !S0 * S1 * HOLD
               + Q3 * !Q2 *  !Q1 * Q0 * !S1 * S0 * HOLD
               + Q3 * !Q2 *  Q1 * !Q0 * !HOLD
               + Q3 * !Q2 *  Q1 * !Q0 * HOLD * S1 * S0
               + Q3 * !Q2 *  Q1 *  Q0 %* BUSY%
               + Q3 *  Q2 * !Q1 *  Q0 * BUSY
               + Q3 *  Q2 *  Q1 *  Q0;

          Q2t = BUSY * Q3 * !Q2 * Q1 * Q0
               + Q0 * Q1 * Q2 * Q3;

          Q3t =   HOLD * !Q3 * !Q2 * !Q1 * !Q0 *  S1 *  S0
               + !HOLD *  Q3 * !Q2 * !Q1 * !Q0
               + !HOLD *  Q3 * !Q2 * !Q1 *  Q0
               +  HOLD *  Q3 * !Q2 * !Q1 *  Q0 *  S1 *  S0
               +  HOLD *  Q3 * !Q2 * !Q1 *  Q0 * !S1 * !S0
               + !HOLD *  Q3 * !Q2 *  Q1 * !Q0
               +  HOLD *  Q3 * !Q2 *  Q1 * !Q0 *  S1 *  S0
               + !HOLD *  Q3 *  Q2 *  Q1 *  Q0;

OE = S0L * !S1L * (Q2 * !Q1 * Q0 + Q2 * Q1 * !Q0
                + Q2 * Q1 * Q0);  %transceiver abilitation%

WE1 = !(!S1 * S0 * !Q2 * !Q1 * Q0
    + S1 * !S0 * !Q2 * !Q1 * Q0);   %latchs add. at rising edge%

WE2 = !(S1 * !S0 * !Q2 * Q1 * Q0);              %latchs data at rising edge%


HAB2 = S1L * !S0L;  %tri_state data out. enable%
```

```
OLE = !S1L * S0L * Q2 * !Q1 * Q0;   %latchs data when high%

RE1 = !S1L * S0L * (Q2 * !Q1 * Q0
                   + Q2 * Q1 * !Q0
                   + Q2 * Q1 * Q0); %out. enable in read cycles%

RE2 = !S1 * S1;
OLE2 = !S1 * S1;

CS1c= !(!S1L * S0L * !AD1 * (Q2 * !Q1 * !Q0        % address 1 %
                           + Q2 * !Q1 * Q0)
       + S1L * !S0L * !AD1 * (Q2 * !Q1 * !Q0
                           + Q2 * !Q1 * Q0));


AD1c = OR11;
AD2c = OR12;
AD3c = OR13;
AD4c = OR14;
AD5c = OR15;
AD6c = OR16;
AD7c = OR17;

DB0c = OR20;
DB1c = OR21;
DB2c = OR22;
DB3c = OR23;
DB4c = OR24;
DB5c = OR25;
DB6c = OR26;
DB7c = OR27;

END$
```

ERR_DETE.ABL-11/1/89

```
MODULE ERROR_DETECT
TITLE 'ERROR_DETECT        VERSION Wednesday 1-NOV-89 0900 hrs'
ERR_DETECT DEVICE 'P22V10';

CLK     PIN     1;
FRDK    PIN     2;
FRAK    PIN     3;
FRSS0   PIN     4;
FRSS1   PIN     5;
FRSS2   PIN     6;
!FSLV   PIN     7;
FEOBA   PIN     8;
!LTC0   PIN     9;
!LTC1   PIN     10;
LCEN    PIN     11;
FCLERR  PIN     13;
CCLERR  PIN     16;

SSERR   PIN     14;
LTC0Z   PIN     15;
FEOB    PIN     17;
FLSS2   PIN     18;
FLSS1   PIN     19;
FLSS0   PIN     20;
ADDR    PIN     21;
FRAKZ   PIN     22;
FRDKZ   PIN     23;

LATCHED_SS      =       [FLSS2..FLSS0];
SS              =       [FRSS2..FRSS0];

ZERO            =       [0,0,0];
TWO             =       [0,1,0];

EQUATIONS

FRAKZ           :=      FRAK;
FRDKZ           :=      FRDK;
LTC0Z           :=      LTC0;

"LATCH THE CURRENT VALUE OF SS IF AK UP OR DK UP/DOWN WITH NON-ZERO SS,
"OTHERWISE KEEP LAST VALUE OF LATCHED SS

"LATCHED_SS HAS LAST NON-ZERO SS RESPONSE

"LATCHED_SS IS ZERO AFTER CLEAR

LATCHED_SS      :=      ((((FRAK & !FRAKZ) # (FRDK != FRDKZ))
                                & (SS != ZERO) & SS)
                        # ((((!(FRAK & !FRAKZ) & (FRDK == FRDKZ))
                                # (SS == ZERO)) & LATCHED_SS))
                        & !FCLERR & !CCLERR & !FSLV;

"ADDR INDICATES WHETHER VALUE OF LATCHED_SS OCCURRED ON ADDRESS
"OR DATA CYCLE
```

```
ADDR            :=      ((FRAK & !FRAKZ) & (SS != ZERO)
                      # ((FRDK == FRDKZ) # (SS == ZERO)) & ADDR)
                      & !FCLERR & !CCLERR & !FSLV;

"SSERR IS SET FOR LATCHED_SS NOT EQUAL ZERO ON ADDRESS CYCLE
"OR LATCHED_SS NOT EQUAL ZERO/TWO ON DATA CYCLE

SSERR           :=      ((LATCHED_SS != ZERO) & ADDR
                      #  (LATCHED_SS != ZERO) & (LATCHED_SS != TWO) & !ADDR);

"FEOB IS SET IF COUNTER REACHES ZERO AND COUNTER IS ENABLED OR
"IF SS=2 ON DATA CYCLE

"IT IS RESET BY SEQUENCER FEOB ACKNOWLEDGE


FEOB            :=      ((LTC0 & !LTC0Z) & LTC1 & LCEN
                      # (FRDK != FRDKZ) & (SS == TWO)
                      # FEOB & !FEOBA)
                      & !FCLERR & !CCLERR & !FSLV;


END ERROR_DETECT
```

FB_ARB.ABL-8/25/89

```
MODULE FASTBUS_ARBITRATION
TITLE 'FASTBUS_ARBITRATION    VERSION 23-AUG-89 1000 hrs'
FB_ARB DEVICE 'EC16P8N';


OAL5            PIN     18;
OAL4            PIN     20;
OAL3            PIN     4;
OAL2            PIN     8;
OAL1            PIN     17;
OAL0            PIN     21;

IAL5            PIN     3;
IAL4            PIN     9;
IAL3            PIN     10;
IAL2            PIN     11;
IAL1            PIN     13;
IAL0            PIN     14;

FAL5            PIN     1;
FAL4            PIN     2;

ISMINE          PIN     5;
GATEALS                 PIN     15;


EQUATIONS

!OAL5   =       !IAL5 # !GATEALS;

!OAL4   =       !IAL4 # (!IAL5 & FAL5) # !GATEALS;

!OAL3   =       !IAL3 # (!IAL4 & FAL4) # (!IAL5 & FAL5) # !GATEALS;

!OAL2   =       !IAL2 # (!IAL3 & OAL3) # (!IAL4 & FAL4) # (!IAL5 & FAL5)
                # !GATEALS;

!OAL1   =       !IAL1 # (!IAL2 & OAL2) # (!IAL3 & OAL3) # (!IAL4 & FAL4)
                # (!IAL5 & FAL5) # !GATEALS;

!OAL0   =       !IAL0 # (!IAL1 & OAL1) # (!IAL2 & OAL2) # (!IAL3 & OAL3)
                # (!IAL4 & FAL4) # (!IAL5 & FAL5) # !GATEALS;

!ISMINE                 =       (!IAL0 & OAL0) # (!IAL1 & OAL1) # (!IAL2 &
OAL2)
                # (!IAL3 & OAL3) # (!IAL4 & FAL4) # (!IAL5 * FAL5)
                # !GATEALS;

END FASTBUS_ARBITRATION
```

FB_REQ.ABL-7/25/90

```
MODULE FB_REQUEST
TITLE 'FB_REQUEST        VERSION 25-JUL-90 1500 hrs'
FB_REQ DEVICE 'P22V10';
"INPUTS
CLK     PIN     1;
!FRESET         PIN     2;
SSERR   PIN     3;
!BUSYR  PIN     4;
!STO    PIN     5;
!DFF    PIN     6;
FRSR    PIN     7;
!FSLV   PIN     8;
FCWT    PIN     9;
FRDS    PIN     10;
CCLERR  PIN     11;
FCLERR  PIN     13;
"OUTPUTS
FRDSZ   PIN     14;
FDWT    PIN     15;
!SBUSYR         PIN     17;     "synchronous BUSYR
ECLK6M  PIN     18;
EC0     PIN     19;     "CLK13M
EC1     PIN     20;
!SEQINT         PIN     21;
!FBERR  PIN     22;
!FBREQ  PIN     23;
ECLK    =       [EC1..EC0];
ZERO    =       [0,0];
TWO     =       [1,0];
EQUATIONS
"Ethernet clock generator
"       divide 40MHz by 3 to get 13.333MHz
"       then divide 13.3 by 2 to get 6.667MHz
WHEN (ECLK == ZERO) THEN ECLK := TWO;
        ELSE ECLK := ECLK - 1;
WHEN (ECLK == TWO) THEN ECLK6M := !ECLK6M;
        ELSE ECLK6M := ECLK6M;
"Synchrnize BUSYR* for Ethernet interface to ensure proper setup, and hold
"       time for the EPB1400's.
WHEN (ECLK == TWO) THEN SBUSYR := BUSYR;
        ELSE SBUSYR := SBUSYR;
"Interrupt equations:
"FBREQ = We're a slave, or We've received SR, or RB
"FBERR = Any SS error except SS=2, or Short Timout, or Data FIFO full.
"SEQINT = FBERR or RB.
FBERR           =       (SSERR # STO # DFF # FBERR)
                        & !FCLERR
                        & !CCLERR;
FBREQ           =       FSLV # FRSR # FRESET;
SEQINT          =       FBERR # FRESET;
"WT generation:
"FASTBUS WT is set high whenever the fscc is a slave, and there is a DS
"transition.  WT is cleared by FCWT from the FASTBUS sequencer.
FRDSZ           :=      FRDS;
FDWT            =       (FSLV * (FRDS != FRDSZ)) # (FDWT * !FCWT);
END FB_REQUEST
```

FBSEQV2.ASM-9/24/90

```
Richard Kwarciany
Fermilab
Saturday, 24-Mar-90  1500hrs
FSCC Version: 2
Revision: 2
EPS448
U106,U110,U108

PART:   EPS448,EPS448,EPS448

INPUTS:         I0@12
        I1@11
        I2@10
        I3@9
        I4@5
        I5@4
        I6@3
        I7@2
OUTPUTS:LC1@1           %       1       %
        LC0@28          %       2       %
        LCOE@27                 %       3       %
        /SRT@26                 %       4       %
        FW@25           %       5       %
        /POE@24                 %       6       %
        DDIR@23                 %       7       %
        CPAB@22                 %       8       %
        SAB@20          %       9       %
        CPBA@19                 %       10      %
        SBA@18          %       11      %
        /DG@17          %       12      %
        /FDOE@16        %       13      %
        /FCOE@15        %       14      %
        FPCREQ@14       %       15      %
        DFIFOEN@13      %       16      %
OUTPUTS:REL@1           %       17      %
        REQ@28          %       18      %
        FEOBA@27        %       19      %
        EG@25           %       20      %
        RD@24           %       21      %
        MS2@23          %       22      %
        MS1@22          %       23      %
        MS0@20          %       24      %
        CWT@19          %       25      %
        CDK@18          %       26      %
        SDK@17          %       27      %
        CDS@16          %       28      %
        SDS@15          %       29      %
        CAS@14          %       30      %
        SAS@13          %       31      %
OUTPUTS:PFIFOEN@1       %       32      %
        /SR@28          %       33      %
        SCLK@27                 %       34      %
        FCLK@26                 %       35      %
        DEOE@25                 %       36      %
        CEOE@24                 %       37      %
        TMR@23          %       38      %
```

```
        CCLEAR@22       %       39      %
        STAT3@20        %       40      %
        STAT2@19        %       41      %
        STAT1@18        %       42      %
        STAT0@17        %       43      %
        /DSACK@16       %       44      %
        /MUX2@15        %       45      %
        /MUX1@14        %       46      %
        /MUX0@13        %       47      %

%DEFAULT = all bits false except: DDIR=1, FCOE*=0, MUX1*=0%
%123456789012345678901234567890123456789012345678901234567%
DEFAULT:[00010110000110100000000000000000010000000000001101B]

%       Macros for Chip 1       %

MACROS:         CHOLD           ="000"          %LC1,LC0,LCOE%
        CLOAD           ="100"
        COUNT           ="110"
        CREAD           ="001"
        CDEF            ="000"
        SRT             ="0"            %SRT*%
        NSRT            ="1"
        DW              ="1"            %SDW%
        NDW             ="0"
        DATADEF                 ="1000001"
        %POE*,DDIR,CPAB,SAB,CPBA,SBA,DG*%
        DATAOUT                 ="0000000"
        DATAOUTLATCH    ="0000100"
        DATAOUTHOLD     ="1000010"
        DATAINPROC      ="0100000"
        DATAINLATCH     ="0110000"
        DATAINHOLD      ="0101000"
        DATAINFIFO      ="1101000"
        COUNTIN                 ="0000001"
        FDOE            ="0"            %FDOE*%
        NFDOE           ="1"
        FCOE            ="0"            %FCOE*%
        NFCOE           ="1"
        IRQ             ="0"            %FPCREQ*%
        NIRQ            ="1"
        DFFEN           ="1"            %DFIFOEN%
        DFFDIS          ="0"
        CTRLDEF                 ="0001011000011010"
        %DEFAULTS FOR U42X40%

        Macros for Chip 2       %

        REQ             ="010000000000000"      %REL,REQ%
        REL             ="101000000101010"      %(EOEA, Clear DS,DK,AS,GK)%
        HOLDB           ="00"
        NREQ            ="00"
        EOBA            ="1"            %EOBA%
        NEOBA           ="0"
        EG              ="1"            %EG%
        NEG             ="0"
        RD              ="1"            %RD%
```

```
NRD             ="0"
MS0             ="000"          %MS2,MS1,MS0%
MS1             ="001"
MS2             ="010"
MS3             ="011"
MS4             ="100"
MS5             ="101"
MS6             ="110"
MS7             ="111"
CWT             ="1"            %CWT%
XWT             ="0"
SDK             ="01"           %CDK,SDK%
CDK             ="10"
XDK             ="00"
SDS             ="01"           %CDS,SDS%
CDS             ="10"
XDS             ="00"
SAS             ="01"           %CAS,SAS%
CAS             ="10"
XAS             ="00"
FBDEF           ="000000000000000"     %DEFAULTS FOR U47X40%

Macros for Chip 3

PFFEN           ="1"            %PFIFOEN%
PFFDIS          ="0"
SR              ="0"            %SR*%
NSR             ="1"
SCLK            ="1"            %SCLK%
NSCLK           ="0"
FCLK            ="1"            %FCLK%
NFCLK           ="0"
DEOE            ="1"            %DEOE%
NDEOE           ="0"
CEOE            ="1"            %CEOE%
NCEOE           ="0"
TMR             ="1"            %TMR%
NTMR            ="0"
CCLR            ="1"            %CCLEAR%
NCCLR           ="0"
STAT0           ="0000"                %STATE3,STATE2,STATE1,STATE0%
STAT1           ="0001"
STAT2           ="0010"
STAT3           ="0011"
STAT4           ="0100"
STAT5           ="0101"
STAT6           ="0110"
STAT7           ="0111"
STAT8           ="1000"
STAT9           ="1001"
STAT10          ="1010"
STAT11          ="1011"
STAT12          ="1100"
STAT13          ="1101"
STAT14          ="1110"
STAT15          ="1111"
DSAK            ="0"            %DSACK*%
```

```
        NDSAK           ="1"
        MUX2            ="011"          %MUX2,MUX1,MUX0%
        MUX1            ="101"
        MUX0            ="110"
%       RESETO = All bits false except: FREL FCWT FCDK FCDS FCAS MUX1%
%                         12345678901234567890123456789012345678901234567%
        RESETO          ="00010110000111101000000011010100100000000001101"

%-----------------------------------------------------------------------%
%Here the 448's set all outputs to their default state, then           %
%wait for either the 68020 to call, or somebody on FASTBUS             %
%to want something.  When FBSEL is true, that means the 68020          %
%is calling, and we need to get the vector from MUX2 and go to         %
%the routine at that address.  If, on the other hand, FSLV is          %
%true, we go to the slave handler.                                     %
%-----------------------------------------------------------------------%

PROGRAM:

RESET:  [RESETO] JUMP SELECT;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Routine entry points.                                           %
%       Entry points are the first instruction of each routine.        %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine ARBITRATE                                               %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
ARB:    IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I2       %FRDY%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT1 DSAK MUX1]
                JUMP ARBX;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD NFDOE FCOE NIRQ DFFDIS
                REQ
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT1 NDSAK
MUX0]
                JUMP ARB;

%-----------------------------------------------------------------------%
%       Routine BUS_RELEASE                                             %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       This routine sets FREL true, clears DS, DK, and AS, then goes   %
%       home.  Note that since no data is needed, the control lines     %
%       latching the 68020 data       are returned to their default states,  %
```

```
%       discarding the data.                                            %
%-----------------------------------------------------------------------%

BUSREL:                 [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT2 NDSAK
MUX1]
                JUMP BUSRELX;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Primary Address Cycles                                          %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine ADDRESS_DATA_GEOGRAPHICAL                               %
%                                                                       %
%       Geographical primary address cycle to data space.               %
%       (we assert EG)                                                  %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ Release bus, TMR=0, JUMP WAYHOME.                      %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate EG,   %
%               MS, and go home.                                        %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDDG:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDDG;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS0 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDDG;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_CSR_GEOGRAPHICAL                                %
%                                                                       %
%       Geographical primary address cycle to CSR space.                %
%       (We assert EG)                                                  %
```

```
%       --FAST-- --WRITE--                                                  %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ THEN Release bus, JUMP WAYHOME.                        %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate EG,   %
%               MS, and go home.                                        %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDCG:  IF /I6          %FBREQ%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDCG;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS1 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDCG;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_DATA_LOGICAL                                    %
%                                                                       %
%       This routine is identical to ADDRESS_DATA_GEOGRAPHICAL          %
%       except that EG is not driven.                                   %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ THEN Release bus, TMR=0, JUMP WAYHOME.                 %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate MS,   %
%               and go home.                                            %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDDL:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK XDS XAS
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDDL;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDDL;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_CSR_LOGICAL                                     %
%-----------------------------------------------------------------------%
%       This routine is identical to ADDRESS_CSR_GEOGRAPHICAL except    %
%       that EG is not driven.                                          %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ THEN Release bus, TMR=0, JUMP WAYHOME.                 %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate MS,   %
%       and go home.                                                    %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDCL:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX0]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDCL;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDCL;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_DATA_BROADCAST                                  %
%                                                                       %
%       This routine does a broadcast primary address cycle to          %
%       DATA space.                                                     %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
```

```
%       IF FBREQ THEN Release bus, TMR=0, jump WAYHOME.                 %
%       ELSEIF WT=1, then loop here.                                    %
%       ELSEIF AK=1, Then turn off FASTBUS AD line drivers, negate MS,  %
%               and go home.                                            %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDDB:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDDB;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDDB;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_CSR_BROADCAST                                   %
%                                                                       %
%       This routine does a broadcast primary address cycle to          %
%       CSR space.                                                      %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN Release bus, TMR=0, JUMP WAYHOME.                 %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate MS,   %
%               and go home.                                            %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDCB:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDCB;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
```

```
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDCB;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_RELEASE                                         %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%               EG=0 RD=0 MS=0 WT=0 Clear DK,DS,AS                      %
ADDREL:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT4 NDSAK
MUX1]
                JUMP ADDREL;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT CDK CDS CAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT4 NDSAK
MUX0]
                JUMP ADDRELX;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Data Cycle Routines                                             %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_READ                              %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
DATPRR:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRR;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX0]
```

```
                JUMP DATPRRX;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DATPRR;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_WRITE                             %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
DATPRW:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRW;
        ELSEIF I4               %FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRWX;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DATPRW;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_READ                         %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
DATPSR:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSR;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 DSAK MUX0]
                JUMP DATPSRX;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
```

```
                JUMP DATPSR;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_WRITE                        %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
DATPSW:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSW;
        ELSEIF I4               %FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSWX;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
                JUMP DATPSW;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_READ                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
DATPBR:                 [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBRX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_WRITE                     %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
DATPBW:                 [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBWX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE                 %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       This routine does a graceful termination of a block transfer    %
%       where DS and DK are brought to their idle states without        %
%       effecting the slave.                                            %
%               MS=0 RD=0                                               %
```

```
%               then
%               DS=0
DATPBT:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX1]
                JUMP DATPBT;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_LOAD                                      %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
LC_LOAD:        [CLOAD NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                JUMP LC_LOADX;

%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_READ                                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
LC_READ:        [CREAD NSRT NDW COUNTIN NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                JUMP LC_READX;

%-----------------------------------------------------------------------%
%       Routine FIFO_WRITE                                              %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
FIFO_WR:        [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT10 DSAK MUX1]
                JUMP FIFO_WRX;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_BLOCK_TRANSFER_READ                           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
DATFBR:         IF /I6                  %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
```

```
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP DATFBR;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP DATFBRX;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_1  (100ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
DATFP1:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP1;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP1X;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_2  (200ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
DATFP2:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP2;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP2X;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_4  (400ns transfers)           %
%       --FAST-- --READ or WRITE--                                      %
%-----------------------------------------------------------------------%
DATFP4:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP4;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP4X;

%-----------------------------------------------------------------------%
%       SLAVE_DATA_INPUT                                                %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
SLVDIN:                 [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 CWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX0]
                JUMP SLVDINX;

%-----------------------------------------------------------------------%
%       SLAVE_DATA_OUTPUT                                               %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       (WT has been cleared back at LETSGO).                           %
%       IF FBREQ* THEN release bus, jump WAYHOME.                       %
%       ELSEIF WT=1 THEN wait here.                                     %
%       ELSEIF DS=1 THEN set DK jump HOME.                              %
%       ELSE clear DK jump HOME.                                        %

SLVDOUT:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP WAYHOME;
        ELSEIF I7       %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX0]
                JUMP SLVDOUT;
        ELSEIF I1       %FRDS%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT SDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX1]
                JUMP WAYHOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT CDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX1]
                JUMP WAYHOME;
```

```
%-----------------------------------------------------------------------%
%       Null                                                            %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
NULL:           [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT12 NDSAK
MUX1]
                JUMP NULLX;

%-----------------------------------------------------------------------%
%       End Of Event                                                    %
%       --SLOW-- --WRITE --                                             %
%-----------------------------------------------------------------------%
EOE:            [CHOLD NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK DEOE CEOE NTMR NCCLR STAT14 DSAK MUX1]
                JUMP EOEX;

%-----------------------------------------------------------------------%
%       Bulb Test                                                       %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
BULBTST:        [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG RD MS7 XWT SDK SDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT11 NDSAK
MUX1]
                JUMP BULBTST;

%-----------------------------------------------------------------------%
%       Sequencer Select                                                %
%       Wait here to be selected by the 68020                           %
%               FB1S* + FB2S means that the FASTBUS standard sequencer  %
%                       has been selected.                              %
%               FB1S* * FB2S* means that the USER sequencer has been    %
%                       selected.                                       %
%                                                                       %
%       Whichever sequencer has not been selected, puts it's outputs    %
%               into high Z, and is not accessable unless it is reset.  %
%-----------------------------------------------------------------------%
SELECT:         IF /I1 * /I5            %FB1S* * FB2S*%
        THEN [Z] CONTINUE;
        ELSEIF (/I1 * I5) + (I1 * /I5)          %FB2S* + FB1S*%
        THEN [RESETO] JUMP HOME;
        ELSE [RESETO] JUMP SELECT;

TURNOFF:[Z] JUMP TURNOFF;
%-----------------------------------------------------------------------%
%       If, in the cycle executed just previous to returning here,      %
%       DSACK was asserted, then the select line (FBxS*)                %
%       will still be asserted (it takes the 68020 and it's PAL's a     %
%       certain amount of time to negate a select line after it sees    %
%       DSACK).  HOME would then see FBxS* true and think that it is    %
%       being selected again, when, in fact, FBxS* is just lingering    %
%       from the previous cycle.  As you might imagine, if this were    %
%       to happen, the sequencer would jump to what ever address        %
%       just happened to be on the 68020's address bus at that time     %
```

```
%       and anarchy would reign.  So, to keep order, if we                %
%       want to come HOME right after DSACK, then we instead return       %
%       here where three NOP type instructions are executed.  This        %
%       allows the 68020 enough time to negate FBxS* from any             %
%       preveously executed cycle.                                        %
%-----------------------------------------------------------------------%
WAYHOME:        [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                CONTINUE;

%       Since the IOD lines could be latched on the next cycle (if there%
%       is a pending instruction), the Processor transceivers must be    %
%       set to DATAOUT before coming HOME.  Any instructions leading to %
%       HOME must have the processor transceivers set in this way or it %
%       will be        bad.                                              %
%       When HOME, all outputs are default
HOME:   IF /I1          %FB1S* (Fast cycle, return DSACK now.)
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 DSAK MUX2]
                JUMP LETSGOF;
        ELSEIF /I5      %FB2S* (Slow cycle, return DSACK in routine.%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX2]
                JUMP LETSGOS;
        ELSE    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%       These are the fast and slow jumps to the appropriate FASTBUS    %
%       primative routines as determined by the vector received from    %
%       the '020 at MUX2.                                               %
%-----------------------------------------------------------------------%

LETSGOF:        [CDEF NSRT NDW DATAOUTLATCH NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 DSAK MUX2]
```

```
                PUSHI;

%       Clear WT here is for Slave write routine, dosn't effect anything
else.%
                [CDEF NSRT NDW DATAOUTHOLD NFDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 CWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX0]
                RETURN;

LETSGOS:        [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX2]
                PUSHI;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX0]
                RETURN;

%-----------------------------------------------------------------------%
%       Routine ARBITRATE                                               %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       Starting with something simple, this routine checks to make     %
%       sure that FBREQ is false, then sets FREQ true, waits for        %
%       FRDY (means that we have the bus), then returns DSACK and goes  %
%       home.                                                           %
%-----------------------------------------------------------------------%

ARBX:           [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT1 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Bus Release                                                     %
%-----------------------------------------------------------------------%
%       Release Bus, Status=2                                           %
%                                                                       %
%       Status=2, Status clock = 1                                      %
BUSRELX:        [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT2 NDSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_RELEASE                                         %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%               Wait for AK to go low, then go HOME.                    %
ADDRELX:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I5               %FRAK%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT CDK CDS CAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT4 NDSAK MUX0]
                JUMP ADDRELX;
        ELSE    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT4 NDSAK MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Data Cycle Routines                                             %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_READ                              %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO INTS.         %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers        %
%               (FDOE*=0,DDIR=1,CPBA=0, DSAB=0,DSBA=0,DG*=0), enable    %
%               processor transceivers (POE*=0), Assert RD=1, MS=0 and  %
%               loop here.                                              %
%       ELSEIF DK=1 THEN return DSACK, and continue.                    %
%       ELSE Enable FASTBUS AD line drivers, enable proccessor xceivers %
%               Assert RD=1, MS=0, DS(u), TMR, and loop here.           %
%                                                                       %
%       Hold DSACK true for one more cycle, and continue.               %
%                                                                       %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, NDSACK, JUMP TO       %
%               WAYHOME.                                                %
%       ELSEIF WT=1 THEN negate DSACK, disable processor and FASTBUS    %
%               transceivers, continue to assert RD=1, MS=0, and DS=1,  %
%               and loop here.                                          %
%       ELSEIF DK=0 THEN negate DSACK, disable processor and FASTBUS    %
%               transceivers, deassert TMR, RD, and MS, and go WAYHOME. %
%       ELSE negate DSACK, disable processor and FASTBUS transceivers,  %
%               deassert RD, MS, and DS, assert TMR, and loop here.     %
%                                                                       %
%-----------------------------------------------------------------------%
DATPRRX:        [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX0]
                CONTINUE;

WAIT1:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
```

```
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP WAIT1;
        ELSEIF /I4              %/FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP WAYHOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP WAIT1;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_WRITE                             %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO WAYHOME.      %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers        %
%               (FDOE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=1, DG*=0),       %
%               Assert RD=0, MS=0 and loop here.                        %
%       ELSEIF DK=1 THEN disable FASTBUS transceivers and drivers, and  %
%               continue.                                               %
%       ELSE Assert DS(u), TMR, keep FASTBUS lines asserted, and loop   %
%               here.                                                   %
%       be bop a lula she's my baby                                     %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO HOME.         %
%       ELSEIF WT=1 THEN continue to assert DS=1, and loop here.        %
%       ELSEIF DK=0 THEN go HOME.                                       %
%       ELSE deassert DS, assert TMR, and loop here.                    %
%-----------------------------------------------------------------------%
DATPRWX:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRWX;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DATPRWX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_READ                         %
```

```
%       --SLOW-- --READ--                                                %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO INTS.          %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers         %
%               (FDOE*=0, DDIR=1,CPBA=0, DSAB=0,DSBA=0,DG*=0), enable    %
%               processor transceivers (POE*=0), Assert RD=1, MS=2 and   %
%               loop here.                                               %
%       ELSEIF DK=1 THEN return DSACK, and continue.                     %
%       ELSE Assert DS(u), TMR, and loop here.                           %
%                                                                        %
%       Hold DSACK true for one more cycle, and continue.                %
%                                                                        %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, DEassert DSACK, jump   %
%               to WAYHOME.                                              %
%       ELSEIF WT=1 THEN negate DSACK, disable processor and FASTBUS     %
%               transceivers, continue to assert RD=1, MS=2, and DS=1,   %
%               and loop here.                                           %
%       ELSEIF DK=0 THEN negate DSACK, disable processor and FASTBUS     %
%               transceivers, deassert RD, and MS, and go WAYHOME.       %
%       ELSE negate DSACK, disable processor and FASTBUS transceivers,   %
%               deassert RD, MS, and DS, assert TMR, and loop here.      %
%                                                                        %
%-----------------------------------------------------------------------%
DATPSRX:        [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 DSAK MUX0]
                CONTINUE;

WAIT2:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP WAIT2;
        ELSEIF /I4              %/FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK MUX1]
                JUMP WAYHOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS2 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
                JUMP WAIT2;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_WRITE                         %
%       --FAST-- --WRITE--                                               %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, jump to WAYHOME.       %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers         %
%               (FDOE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=1, DG*=0),        %
```

```
%               Assert RD=0, MS=2 and loop here.                        %
%       ELSEIF DK=1 THEN disable FASTBUS transceivers and drivers, and  %
%               continue.                                               %
%       ELSE Assert DS(u), keep FASTBUS lines asserted, assert TMR,     %
%               and loop here.                                          %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, jump to HOME.         %
%       ELSEIF WT=1 THEN continue to assert DS=1, and loop here.        %
%       ELSEIF DK=0 THEN go HOME.                                       %
%       ELSE deassert DS, assert TMR, and loop here.                    %
%-----------------------------------------------------------------------%
DATPSWX:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSWX;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
                JUMP DATPSWX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_READ                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       This routine is going to be a little bit different than any of  %
%               the others, in that, DSACK will have to be returned     %
%               after each DK(t) to allow the 68020 to read the data.   %
%       Here goes:                                                      %
%                                                                       %
%       Turn on FASTBUS AD line receivers, Assert RD=1, MS=1,switch     %
%               to MUX1 and continue.                                   %
%       IF FBREQ THEN Release bus, JUMP TO INTS.                        %
%       ELSEIF WT=1 THEN Assert RD=1, MS=1, on FASTBUS  Turn on FASTBUS %
%               xceivers and enable processor xceivers                  %
%               (POE*=0, DDIR=1, CPBA=0, DSAB=0, DSBA=0, DG*=0, FDOE*=0)%
%               and loop here.                                          %
%       ELSEIF DS=0 THEN Assert RD=1, MS=1, DS(u), enable FASTBUS and   %
%               processor transceivers (POE*=0, DDIR=1, CPBA=0,         %
%               DSAB=0, DSBA=0, DG*=0, FDOE*=0)and assert TMR, and JUMP %
%               to DKUPR.                                               %
%       ELSE Assert RD=1, MS=1, DS(d), enable FASTBUS and processor     %
%               transceivers (POE*=0, DDIR=1, CPBA=0, DSAB=0,           %
%               DSBA=0, DG*=0, FDOE*=0) assert TMR, and JUMP to DKDWNR. %
%DKDWNR                                                                 %
%       IF FBREQ THEN RD=1, MS=1, Clear DS, jump to INTS.               %
```

```
%       ELSEIF DK=0 THEN Assert DSACK, continue to drive FASTBUS lines, %
%               and processor and FASTBUS transceivers, deassert TMR,    %
%               and JUMP to next1.                                       %
%       ELSE Continue to drive FASTBUS lines, and transceivers, and      %
%               TMR, and loop here.                                      %
%DKUPR                                                                   %
%       IF FBREQ THEN RD=1, MS=1, Clear DS, assert DSACK, jump to        %
%               NEXT2.                                                   %
%       ELSEIF DK=1 THEN Assert DSACK, continue to drive FASTBUS lines, %
%               and processor and FASTBUS transceivers, deassert TMR,    %
%               and JUMP to next1.                                       %
%       ELSE Continue to drive FASTBUS lines, and transceivers,          %
%               assert TMR, and loop here.                               %
%NEXT1                                                                   %
%       Continue to drive FASTBUS lines, DSACK, and transceivers, and    %
%               Continue.                                                %
%       Continue to drive FASTBUS lines, and transceivers negate DSACK   %
%               and JUMP WAYHOME.                                        %
%                                                                        %
%NEXT2  Clear FASTBUS lines, release bus, keep DSACK asserted, jump       %
%               to WAYHOME.                                              %
%------------------------------------------------------------------------%
DATPBRX:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBRX;
        ELSEIF /I1              %/FRDS%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPR;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNR;

DKDWNR:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX1]
                JUMP NEXT1;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNR;

DKUPR:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 DSAK
MUX1]
                JUMP NEXT2;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX1]
                JUMP NEXT1;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPR;

NEXT1:          [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP WAYHOME;

NEXT2:          [CTRLDEF
                REL
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_WRITE                     %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       As with DATPBR, DSACK will have to be returned after each DK(t) %
%               to allow the 68020 to output new data.                  %
%                                                                       %
%       Keep FASTBUS AD line drivers enabled, assert MS=1, set MUX1,    %
%               and continue.                                           %
%       IF FBREQ THEN Release bus, JUMP TO WAYHOME.                     %
%       ELSEIF WT=1 THEN Assert MS=1, keep FASTBUS transceivers enabled,%
%               and enable FASTBUS drivers (POE*=1, DDIR=0,             %
%               CPBA=0, DSAB=0, DSBA=1, DG*=0, FDOE*=0), and loop here. %
%       ELSEIF DS=0 THEN Assert MS=1, DS(u), TMR, keep FASTBUS          %
%               transceivers enabled and enable FASTBUS drivers (POE*=1,%
%               DDIR=0, CPBA=0, DSAB=0, DSBA=1, DG*=0, FDOE*=0), and    %
%               JUMP to DKUPW.                                          %
%       ELSE Assert MS=1, DS(d), TMR, keep FASTBUS transceivers enabled,%
%               and enable FASTBUS drivers (POE*=1, DDIR=0,             %
%               CPBA=0, DSAB=0, DSBA=1, DG*=0, FDOE*=0), and JUMP to    %
%               DKDWNW.                                                 %
%DKDWNW                                                                 %
%       IF FBREQ THEN Release bus, JUMP TO HOME.                        %
```

```
%       ELSEIF DK=0 THEN deassert MS, TMR, disable FASTBUS transceivers %
%               and drivers (POE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=0,     %
%               DG*=1, FDOE*=1), and JUMP home.                          %
%       ELSE continue to assert MS, TMR, and keep FASTBUS transceivers   %
%               and drivers enabled, and loop here.                      %
%DKUPW                                                                  %
%       IF FBREQ THEN Release bus, JUMP TO HOME.                         %
%       ELSEIF DK=1 THEN deassert MS, TMR, disable FASTBUS transceivers %
%               and drivers (POE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=0,     %
%               DG*=1, FDOE*=1), and JUMP home.                          %
%       ELSE continue to assert MS, TMR, and keep FASTBUS transceivers   %
%               and drivers enabled, and loop here.                      %
%-----------------------------------------------------------------------%
DATPBWX:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBWX;
        ELSEIF /I1              %/FRDS%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPW;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNW;

DKDWNW:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNW;

DKUPW:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
```

```
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPW;

%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_LOAD                                      %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       This routine loads the Local Word Counter with data from        %
%       the processor.  Only the lower 12 bits of the 32 bit word       %
%       are used.                                                       %
%                                                                       %
%       Set POE* true, put Local Word Counter into load mode,           %
%               assert DSACK and continue.                              %
%       Negate POE*, put Local Word Counter into hold, keep DSACK       %
%               asserted, and continue.                                 %
%       Negate DSACK (hold DSACK false for two cycles to prevent        %
%               sequencer from seeing FBSEL from this cycle when        %
%               returning home).                                        %
%                                                                       %
LC_LOADX:       [CHOLD NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 NDSAK MUX1]
                JUMP WAYHOME;


%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_READ                                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       Routine to allow the processor to read the value of the Local   %
%               Word Counter.                                           %
%                                                                       %
%       Set POE* true, put Local Word Counter into Read mode, assert    %
%               DSACK, and continue.                                    %
%       Keep all outputs the same for one more cycle.                   %
%       Negate DSACK (hold DSACK false for two cycles to prevent        %
%               sequencer from seeing FBSEL from this cycle when        %
%               returning home).                                        %
%                                                                       %
LC_READX:       [CREAD NSRT NDW COUNTIN NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
```

```
%       Routine FIFO_WRITE                                              %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       Routine to write a word from the 68020 into the processor       %
%       and data FIFO's.                                                %
%                                                                       %
%       Enable processor transceivers (POE*=0), set FIFO Write true     %
%               (DW=1), enable FIFO's, set DSACK true.                  %
%       Set FCLK true.                                                  %
%       Negate DSACK, DW, FCLK, and POE*, JUMP TO WAYHOME.              %
%                                                                       %
FIFO_WRX:       [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFEN NSR SCLK FCLK NDEOE NCEOE NTMR NCCLR STAT10 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_BLOCK_TRANSFER_READ                           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
%       This routine does a FASTBUS handshake block transfer read from  %
%       the FASTBUS crate segment to the FSCC's Output Port Data FIFO,  %
%       and also the Processor Data FIFO.                               %
%-----------------------------------------------------------------------%
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFBR:IF FBREQ                                                        %
%       THEN release bus jump to INTF                                   %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers, enable processor and Data FIFO's  %
%               MS=1, RD=1, Place local word counter into hold, and     %
%               loop here.                                              %
%       ELSE Turn on FASTBUS xcievers, enable processor and Data FIFO's %
%               MS=1, RD=1, Place local word counter into count, set    %
%               DS=1, NO FCLK!, and jump to DATFBRX.                    %
%                                                                       %
%BXFERLP:IF (/FRDK * FRSS1) + EOB + FBREQ                               %
%       THEN Enable FASTBUS xceivers, enable processor and Data FIFO's  %
%               MS=1, RD=1, Place local word counter into hold mode,    %
%               and JUMP to EOBIRQ.                                     %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers, enable processor and Data FIFO's  %
%               MS=1, RD=1, Place local word counter into hold, and     %
%               loop here.                                              %
%       ELSEIF DK=0                                                     %
%       THEN Turn on FASTBUS xcievers, enable processor and Data FIFO's %
%               MS=1, RD=1, Place local word counter into count, set    %
%               DS=1, set FCLK, and continue.                           %
%       ELSE Place local word counter into hold, and loop here.         %
%                                                                       %
%DATFBRX:Place counter into hold mode, negate FCLK, set TMR, and        %
%               continue.  This        cycle ensures that the FSCC's minimum %
%               FASTBUS cycle time (100ns) is not violated.  If some    %
%               slave returns DK too quickly, the FSCC could            %
%               try to do another DS before the FIFO's were ready for   %
%               the next word.  This cycle keeps the sequencer from     %
```

```
%               looking for DK before it should.                        %
%                                                                       %
%       Keep TMR true, and do another NOP for the same reason as above. %
%                                                                       %
%       IF (FRDK* FRSS1) + EOB + FBREQ                                  %
%       THEN counter=hold, NTMR, keep FIFO's enabled and jump to EOBIRQ.%
%       ELSEIF WT=1                                                     %
%       THEN counter=hold, NTMR, keep both FIFO's enabled and loop here.%
%       ELSEIF DK=1                                                     %
%       THEN counter=decr, set DS=0, set FCLK, keep FIFO's enabled,     %
%               Clear TMR, and continue.                                %
%       ELSE counter=hold, TMR, keep FIFO's enabled, and loop here.     %
%                                                                       %
%       Place local word counter in hold, negate FCLK, keep processor   %
%               and oport FIFO's enabled, set TMR, and JUMP to DATFBR.  %
%               This cycle ensures that the FSCC's minimum cycle time   %
%               is not violated in the same way as above.               %
%       Keep TMR set, do another NOP for the same reason as above, and  %
%               jump to BXFERLP.                                        %
%                                                                       %
%EOBIRQ:IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF DS=1                                                     %
%       THEN counter=hold, TMR, continue.                               %
%       ELSE counter=hold, TMR, JUMP DSLOWH.                            %
%                                                                       %
%       --case DS=1--                                                   %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=1 THEN Clear TMR, continue.                           %
%       ELSE keep TMR true, and wait here.                              %
%                                                                       %
%       Wait two cycles to allow for FIFO write.                        %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF WT=1                                                     %
%       THEN RD=1, MS=0, counter=hold, FIFO's disabled loop here.       %
%       ELSE DS=0, RD=0, MS=0, FIFO's Disabled continue.                %
%                                                                       %
%       IF FBREQ THEN Release bus, jump HOME.                           %
%       ELSEIF DK=0 THEN NTMR, EOBA, IRQ, jump HOME.                    %
%       ELSE TMR, wait here.                                            %
%                                                                       %
%DSLOWH:IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF DK=0 THEN clear TMR, continue.                           %
%       ELSE TMR, wait here.                                            %
%                                                                       %
%       wait three cycles to ensure FIFO write, then IRQ, EOBA, jump    %
%       home.                                                           %
%                                                                       %
BXFERLP:IF (/I4 * I0) + I3 + /I6      %(/FRDK * FRSS1) + FEOB + FBREQ*%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP EOBIRQ;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
```

```
                JUMP BXFERLP;
        ELSEIF /I4              %/FRDK%
        THEN    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP BXFERLP;

DATFBRX:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

WAIT3:  IF (I4 * I0) + I3 + /I6                %(FRDK * FRSS1) + FEOB + FBREQ*%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP EOBIRQ;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP WAIT3;
        ELSEIF I4               %FRDK%
        THEN    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP WAIT3;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP BXFERLP;

EOBIRQ:         IF /I0          %/FRSS1%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
```

```
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

        IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX0]
                JUMP INTF;
        ELSEIF I1               %FRDS%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP DSLOWH;

        Case--DS=1
DSHIGHH:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX0]
                JUMP INTF;
        ELSEIF I4               %FRDK%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP DSHIGHH;

                [CHOLD NSRT NDW DATAINFIFO NFDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO NFDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

WAIT5:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK
MUX0]
                JUMP WAIT5;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

WAIT6:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %FRDK=0%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE IRQ DFFDIS
                HOLDB EOBA NEG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP WAIT6;

%       Case--DS=0
DSLOWH:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %FRDK=0%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP DSLOWH;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE IRQ DFFEN
                HOLDB EOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
```

```
%       Routine DATA_FIFO_PIPELINED_READ_1  (100ns transfers)            %
%       --FAST-- --WRITE or READ--                                       %
%-----------------------------------------------------------------------%
%       FASTBUS pipeline block transfer read from the FASTBUS crate      %
%       segment to the FSCC's Output Port Data FIFO, and also the        %
%       Processor Data FIFO if enabled through the parrallel port.       %
%                                                                        %
%       Set FASTBUS xcievers for input (DDIR=1, SAB=1, FDOE*=0),         %
%               enable data and processor FIFO's (DFIFOEN=1, PFIFOEN=1),%
%               MS=3, RD=1, and continue.                                %
%                                                                        %
%       Note: DDIR, and SAB must be held high throughout the block       %
%               transfer operation.                                      %
%                                                                        %
%DATFP1:IF FBREQ THEN Release bus, jump to INTS.                          %
%       ELSEIF WT=1                                                      %
%       THEN Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into hold, and loop here.                   %
%       ELSE Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into count, set DS=1, and JUMP to DATFP1X:  %
%                                                                        %
%PXFR1LP:IF FBREQ THEN Release bus, jump to INTS.                         %
%       ELSEIF WT=1                                                      %
%       THEN Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into hold, and loop here.                   %
%       ELSEIF EOB                                                       %
%       THEN Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into hold mode, and JUMP to EOBPIPE.        %
%       ELSE Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into count, set DS=1, set FCLK, and         %
%               continue.                                                %
%                                                                        %
%DATFP1X:Place counter into hold mode, negate FCLK, and                  %
%               continue.  This        cycle provides the FSCC's         %
%               FASTBUS cycle time (100ns).                              %
%                                                                        %
%       IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF WT=1                                                      %
%       THEN counter=hold, keep both FIFO's enabled and loop here.       %
%       ELSEIF EOB                                                       %
%       THEN counter=hold, keep FIFO's enabled and jump to EOBPIPE.      %
%       ELSE counter=decr, set DS=0, set FCLK, keep FIFO's enabled,      %
%               and continue.                                            %
%                                                                        %
%       Place local word counter in hold, negate FCLK, keep processor    %
%               and oport FIFO's enabled, and JUMP to PXFR1LP.           %
%               This cycle provides the FSCC's cycle time.               %
%                                                                        %
%EOBPIPE:IF SS <> 2 (Local counter terminated the block transfer)        %
%       THEN FCLK continue.                                              %
%       ELSE (SS=2 terminated the block transfer) continue.              %
%                                                                        %
%       IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF DS=1                                                      %
%       THEN counter=hold, TMR, continue.                                %
%       ELSE counter=hold, TMR, JUMP DSLOWP.                             %
%                                                                        %
```

```
%       --case  DS=1--                                                  %
%DSHIGHP:IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=1 THEN Clear TMR, continue.                           %
%       ELSE keep TMR true, and wait here.                              %
%                                                                       %
%       Wait two cycles to allow for FIFO write.                        %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF WT=1 THEN RD=1, counter=hold, disable FIFO's, loop here. %
%       ELSE RD=0, MS=0, DS=0, disable FIFO', continue.                 %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=0 THEN clear TMR, EOBA, IRQ, continue.                %
%       ELSE TMR, wait here.                                            %
%                                                                       %
%       --case  DS=0--                                                  %
%DSLOWP:IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=0 THEN clear TMR, CONTINUE.                           %
%       ELSE TMR, wait here.                                            %
%                                                                       %
%       Wait three cycles to ensure FIFO write, then IRQ, EOBA, jump    %
%       home.                                                           %
%                                                                       %
PXFR1LP:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR1LP;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

DATFP1X:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT7:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
```

```
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT7;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR1LP;

EOBPIPE:IF /I0          %/FRSS1%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

        IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I1               %FRDS%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                JUMP DSLOWP;

%       Case  DS=1                                                      %
DSHIGHP:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I4               %FRDK%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
```

```
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                JUMP DSHIGHP;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT9:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK
MUX0]
                JUMP WAIT9;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT10:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4      %FRDK=0%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE IRQ DFFDIS
                HOLDB EOBA NEG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT10;

%       Case  DS=0                                                      %
DSLOWP:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %FRDK=0%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
```

```
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO NFDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DSLOWP;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE IRQ DFFEN
                HOLDB EOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_2  (200ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
%       This routine does a FASTBUS pipeline block transfer read from   %
%       the FASTBUS crate segment to the FSCC's Output Port Data FIFO,  %
%       and also the Processor Data FIFO.                               %
%                                                                       %
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFP2:IF FBREQ THEN Release bus, JUMP WAYHOME.                        %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, Enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold, and loop here.%
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold mode, and      %
%               JUMP to EOBPIPE.                                        %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into count, set DS=1,    %
%               and JUMP to DATFP2X.                                    %
%                                                                       %
%PXFR2LP:IF FBREQ THEN Release bus, JUMP WAYHOME.                       %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, Enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold, and loop here.%
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold mode, and      %
%               JUMP to EOBPIPE.                                        %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into count, set DS=1,    %
%               set FCLK, and continue.                                 %
%                                                                       %
```

```
%DATFP2X:Place counter into hold mode, negate FCLK, and                %
%               continue.  These cycles provide the FSCC's             %
%               FASTBUS cycle time (200ns).                            %
%                                                                      %
%       Continue (add 50ns to cycle time).                             %
%                                                                      %
%       Continue (add 50ns to cycle time).                             %
%                                                                      %
%       IF FBREQ THEN Release bus, jump to HOME.                       %
%       ELSEIF WT=1                                                    %
%       THEN counter=hold, keep both FIFO's enabled and loop here.     %
%       ELSEIF EOB                                                     %
%       THEN counter=hold, keep FIFO's enabled and jump to EOBPIPE.    %
%       ELSE counter=decr, set DS=0, set FCLK, keep FIFO's enabled,    %
%               and continue.                                          %
%                                                                      %
%       Place local word counter in hold, negate FCLK, keep processor  %
%               and oport FIFO's enabled, and Continue.                %
%               These cycles provide the FSCC's cycle time.            %
%                                                                      %
%       Continue (add 50ns to cycle time).                             %
%                                                                      %
%       Jump to PXFR2LP. (add 50ns to cycle time).                     %
%                                                                      %
PXFR2LP:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR2LP;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

DATFP2X:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
```

```
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT11:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT11;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR2LP;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_4  (400ns transfers)           %
%       --FAST-- --READ or WRITE--                                      %
%-----------------------------------------------------------------------%
%       This routine does a FASTBUS pipeline block transfer read from   %
%       the FASTBUS crate segment to the FSCC's Output Port Data FIFO,  %
%       and also the Processor Data FIFO.                               %
%                                                                       %
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFP4:IF FBREQ THEN Release bus, jump to WAYHOME.                      %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, enable data and         %
%               processor FIFO's, MS=3, RD=1, Place local word counter  %
%               into hold, and loop here.                               %
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, Place    %
```

```
%               local word counter into hold mode, MS=3, RD=1, and JUMP %
%               to EOBPIPE.                                             %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, Place    %
%               local word counter into count, MS=3, RD=1, set DS=1,    %
%               and JUMP to DATFP4X.                                    %
%                                                                       %
%PXFR4LP:IF FBREQ THEN Release bus, jump to WAYHOME.                     %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, enable data and         %
%               processor FIFO's, MS=3, RD=1, Place local word counter  %
%               into hold, and loop here.                               %
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, Place    %
%               local word counter into hold mode, MS=3, RD=1, and JUMP %
%               to EOBPIPE.                                             %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, Place    %
%               local word counter into count, MS=3, RD=1, set DS=1,    %
%               set FCLK, and continue.                                 %
%                                                                       %
%       Place counter into hold mode, negate FCLK, and                  %
%               continue.  These cycles provide the FSCC's              %
%               FASTBUS cycle time (400ns).                             %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF WT=1                                                     %
%       THEN counter=hold, keep both FIFO's enabled and loop here.      %
%       ELSEIF EOB                                                      %
%       THEN counter=hold, keep FIFO's enabled and jump to EOBPIPE.     %
%       ELSE counter=decr, set DS=0, set FCLK, keep FIFO's enabled,     %
%               and continue.                                           %
%                                                                       %
%       Place local word counter in hold, negate FCLK, keep processor   %
%               and oport FIFO's enabled, and Continue.                 %
%               These cycles provide the FSCC's cycle time.             %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
```

```
%       Jump to PXFR4LP.(add 50ns to cycle time).
%
PXFR4LP:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR4LP;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

DATFP4X:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;


                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
```

```
                CONTINUE;

WAIT12:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT12;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR4LP;
```

```
%-----------------------------------------------------------------------%
%       SLAVE_DATA_INPUT                                                %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       Enable FASTBUS AD line receivers, Clear WT, jump to SLVDINX.    %
%                                                                       %
%       IF FBREQ then release bus, jump to INTS.                        %
%       ELSEIF WT=1, THEN wait here.                                    %
%       ELSEIF DS=1 THEN latch FASTBUS AD lines, set DK, DSACK,         %
%               continue.                                               %
%       ELSE (DS=0) THEN latch FASTBUS AD lines, clear DK, DSACK,       %
%               continue.                                               %
%                                                                       %
%       DSACK, Hold latched FASTBUS data, jump WAYHOME.                 %

SLVDINX:IF /I6          %FBREQ%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7       %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX0]
                JUMP SLVDINX;
        ELSEIF I1       %FRDS%
        THEN    [CDEF NSRT NDW DATAINLATCH FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT SDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 DSAK
MUX0]
                CONTINUE;
        ELSE    [CDEF NSRT NDW DATAINLATCH FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT CDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 DSAK
MUX0]
                CONTINUE;

                [CDEF NSRT NDW DATAINHOLD NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Null                                                            %
%       --FAST-- WRITE or READ--                                        %
%-----------------------------------------------------------------------%
NULLX:          [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT12 NDSAK
MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       End Of Event                                                    %
```

```
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       --Two dummy words--                                             %
%       DW (Data FIFO Write), DEOE, CEOE, DFFEN, Set FASTBUS interface  %
%               transceivers for output to FIFO (Data and Processor).   %
%               DSACK, continue.                                        %
%                                                                       %
%       Hold all lines for one more cycle (make sure that the write     %
%               pulse to the FIFO's is long enough).                    %
%                                                                       %
%       Hold DEOE true for one more cycle while DW is false.  This      %
%               provides the "hold" time for the fifo's.                %
%                                                                       %
%       Set DW, for two cycles to insert another dummy word with DEOE   %
%               false.                                                  %
%                                                                       %
%       Hold DEOE true for one more cycle while DW is false, to         %
%               ensure that the fifo data hold time spec is met.        %
%                                                                       %
EOEX:           [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK DEOE CEOE NTMR NCCLR STAT14 DSAK MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK DEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%       Interrupt handler                                               %
%-----------------------------------------------------------------------%
%       Being interrupted means that there was a FASTBUS error, or      %
%       the RB (Reset Bus) line has been driven true.  In either case   %
%       the sequencer's assigned task is simply to abort any operation  %
%       it is currently doing, and return to idle state.                %
%                                                                       %
%       Returning DSACK to the 68020, is a prcisely timed function      %
```

```
%       which the sequencer must perform.  DSACK is asserted true        %
%       for exactly two sequencer cycles.  Then it is deasserted, and    %
%       the FBxS* lines may not be sampled for three sequencer cycles.   %
%       What this means is that when a routine wants to return DSACK     %
%       and return to idle, it must do three nop type cycles before      %
%       testing either of the two FBxS* lines.                           %
%                                                                        %
%       The interrupt handler is a strange routine since the IRQ line    %
%       is tested at almost every "IF" instruction there is.  If IRQ is  %
Richard Kwarciany
Fermilab
Monday, 24-Sep-90  1300hrs
FSCC Version: 2
Revision: 4
EPS448
U106,U110,U108


PART:   EPS448,EPS448,EPS448

INPUTS:         I0@12
        I1@11
        I2@10
        I3@9
        I4@5
        I5@4
        I6@3
        I7@2


OUTPUTS:LC1@1           %       1       %
        LC0@28          %       2       %
        LCOE@27                 %       3       %
        /SRT@26                 %       4       %
        FW@25           %       5       %
        /POE@24                 %       6       %
        DDIR@23                 %       7       %
        CPAB@22                 %       8       %
        SAB@20          %       9       %
        CPBA@19                 %       10      %
        SBA@18          %       11      %
        /DG@17          %       12      %
        /FDOE@16        %       13      %
        /FCOE@15        %       14      %
        FPCREQ@14       %       15      %
        DFIFOEN@13      %       16      %

OUTPUTS:REL@1           %       17      %
        REQ@28          %       18      %
        FEOBA@27        %       19      %
        EG@25           %       20      %
        RD@24           %       21      %
        MS2@23          %       22      %
        MS1@22          %       23      %
        MS0@20          %       24      %
        CWT@19          %       25      %
```

```
        CDK@18          %       26      %
        SDK@17          %       27      %
        CDS@16          %       28      %
        SDS@15          %       29      %
        CAS@14          %       30      %
        SAS@13          %       31      %

OUTPUTS:PFIFOEN@1       %       32      %
        /SR@28          %       33      %
        SCLK@27                 %       34
        FCLK@26                 %       35
        DEOE@25                 %       36
        CEOE@24                 %       37
        TMR@23          %       38      %
        CCLEAR@22       %       39      %
        STAT3@20        %       40      %
        STAT2@19        %       41      %
        STAT1@18        %       42      %
        STAT0@17        %       43      %
        /DSACK@16       %       44      %
        /MUX2@15        %       45      %
        /MUX1@14        %       46      %
        /MUX0@13        %       47      %

%       DEFAULT = all bits false except: DDIR=1, FCOE*=0, MUX1*=0
%        123456789012345678901234567890123456789012345678901234567       %
DEFAULT:[0001011000011010000000000000000010000000000001101B]

%       Macros for Chip 1       %

MACROS:         CHOLD           ="000"          %LC1,LC0,LCOE%
        CLOAD           ="100"
        COUNT           ="110"
        CREAD           ="001"
        CDEF            ="000"

        SRT             ="0"            %SRT*%
        NSRT            ="1"

        DW              ="1"            %SDW%
        NDW             ="0"

        DATADEF                 ="1000001"
        %POE*,DDIR,CPAB,SAB,CPBA,SBA,DG*%
        DATAOUT                 ="0000000"
        DATAOUTLATCH    ="0000100"
        DATAOUTHOLD     ="1000010"
        DATAINPROC      ="0100000"
        DATAINLATCH     ="0110000"
        DATAINHOLD      ="0101000"
        DATAINFIFO      ="1101000"
        COUNTIN                 ="0000001"

        FDOE            ="0"            %FDOE*%
        NFDOE           ="1"

        FCOE            ="0"            %FCOE*%
```

```
        NFCOE           ="1"

        IRQ             ="0"            %FPCREQ*%
        NIRQ            ="1"

        DFFEN           ="1"            %DFIFOEN%
        DFFDIS          ="0"

        CTRLDEF                 ="0001011000011010"    %DEFAULTS FOR U42X40%

        Macros for Chip 2       %

        REQ             ="010000000000000"     %REL,REQ%
        REL             ="101000000101010"     %(EOEA, Clear DS,DK,AS,GK)%
        HOLDB           ="00"
        NREQ            ="00"

        EOBA            ="1"            %EOBA%
        NEOBA           ="0"

        EG              ="1"            %EG%
        NEG             ="0"

        RD              ="1"            %RD%
        NRD             ="0"

        MS0             ="000"          %MS2,MS1,MS0%
        MS1             ="001"
        MS2             ="010"
        MS3             ="011"
        MS4             ="100"
        MS5             ="101"
        MS6             ="110"
        MS7             ="111"

        CWT             ="1"            %CWT%
        XWT             ="0"

        SDK             ="01"           %CDK,SDK%
        CDK             ="10"
        XDK             ="00"

        SDS             ="01"           %CDS,SDS%
        CDS             ="10"
        XDS             ="00"

        SAS             ="01"           %CAS,SAS%
        CAS             ="10"
        XAS             ="00"

        FBDEF           ="000000000000000"     %DEFAULTS FOR U47X40%

%       Macros for Chip 3       %

        PFFEN           ="1"            %PFIFOEN%
        PFFDIS          ="0"
```

```
        SR              ="0"            %SR*%
        NSR             ="1"

        SCLK            ="1"            %SCLK%
        NSCLK           ="0"

        FCLK            ="1"            %FCLK%
        NFCLK           ="0"

        DEOE            ="1"            %DEOE%
        NDEOE           ="0"

        CEOE            ="1"            %CEOE%
        NCEOE           ="0"

        TMR             ="1"            %TMR%
        NTMR            ="0"

        CCLR            ="1"            %CCLEAR%
        NCCLR           ="0"

        STAT0           ="0000"                 %STATE3,STATE2,STATE1,STATE0%
        STAT1           ="0001"
        STAT2           ="0010"
        STAT3           ="0011"
        STAT4           ="0100"
        STAT5           ="0101"
        STAT6           ="0110"
        STAT7           ="0111"
        STAT8           ="1000"
        STAT9           ="1001"
        STAT10          ="1010"
        STAT11          ="1011"
        STAT12          ="1100"
        STAT13          ="1101"
        STAT14          ="1110"
        STAT15          ="1111"

        DSAK            ="0"            %DSACK*%
        NDSAK           ="1"

        MUX2            ="011"          %MUX2,MUX1,MUX0%
        MUX1            ="101"
        MUX0            ="110"

%       RESETO = All bits false except: FCOE* FREL FCWT FCDK FCDS FCAS MUX1%
%                        123456789012345678901234567890123456789012345678901234567%
        RESETO          ="0001011000011010100000001101010010000000000001101"

%       RESETO2 = All bits false except: FREL FCWT FCDK FCDS FCAS MUX1%
%                        123456789012345678901234567890123456789012345678901234567%
        RESETO2
        ="0001011000011110100000001101010010000000000001101"

%-----------------------------------------------------------------------------%
%       Here the 448's set all outputs to their default state, then         %
```

```
%       wait for either the 68020 to call, or somebody on FASTBUS        %
%       to want something.  When FBSEL is true, that means the 68020     %
%       is calling, and we need to get the vector from MUX2 and go to    %
%       the routine at that address.  If, on the other hand, FSLV is     %
%       true, we go to the slave handler.                                %
%-----------------------------------------------------------------------%


PROGRAM:

RESET:  [RESETO2] JUMP SELECT;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Routine entry points.                                            %
%       Entry points are the first instruction of each routine.          %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine ARBITRATE                                                %
%       --SLOW-- --WRITE--                                               %
%-----------------------------------------------------------------------%
ARB:    IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I2       %FRDY%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT1 DSAK MUX1]
                JUMP ARBX;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD NFDOE FCOE NIRQ DFFDIS
                REQ
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT1 NDSAK
MUX0]
                JUMP ARB;


%-----------------------------------------------------------------------%
%       Routine BUS_RELEASE                                              %
%       --FAST-- --WRITE--                                               %
%-----------------------------------------------------------------------%
%                                                                        %
%       This routine sets FREL true, clears DS, DK, and AS, then goes    %
%       home.  Note that since no data is needed, the control lines      %
%       latching the 68020 data       are returned to their default states, %
%       discarding the data.                                             %
%-----------------------------------------------------------------------%


BUSREL:                 [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT2 NDSAK
MUX1]
                JUMP BUSRELX;


%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Primary Address Cycles
        %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine ADDRESS_DATA_GEOGRAPHICAL                               %
%                                                                       %
%       Geographical primary address cycle to data space.               %
%       (we assert EG)                                                  %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ Release bus, TMR=0, JUMP WAYHOME.                      %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate EG,   %
%               MS, and go home.                                        %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDDG:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDDG;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS0 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDDG;


%-----------------------------------------------------------------------%
%       Routine ADDRESS_CSR_GEOGRAPHICAL                                %
%                                                                       %
%       Geographical primary address cycle to CSR space.                %
%       (We assert EG)                                                  %
%       --FAST-- --WRITE--                                              %
```

```
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ THEN Release bus, JUMP WAYHOME.                        %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate EG,   %
%               MS, and go home.                                        %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDCG:  IF /I6          %FBREQ%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDCG;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA EG NRD MS1 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDCG;


%-----------------------------------------------------------------------%
%       Routine ADDRESS_DATA_LOGICAL                                    %
%                                                                       %
%       This routine is identical to ADDRESS_DATA_GEOGRAPHICAL          %
%       except that EG is not driven.                                   %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ THEN Release bus, TMR=0, JUMP WAYHOME.                 %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate MS,   %
%               and go home.                                            %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDDL:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK XDS XAS
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDDL;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDDL;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_CSR_LOGICAL                                     %
%-----------------------------------------------------------------------%
%       This routine is identical to ADDRESS_CSR_GEOGRAPHICAL except    %
%       that EG is not driven.                                          %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       IF FBREQ THEN Release bus, TMR=0, JUMP WAYHOME.                 %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate MS,   %
%       and go home.                                                    %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDCL:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX0]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDCL;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDCL;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_DATA_BROADCAST                                  %
%                                                                       %
%       This routine does a broadcast primary address cycle to          %
%       DATA space.                                                     %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
```

```
%       IF FBREQ THEN Release bus, TMR=0, jump WAYHOME.                 %
%       ELSEIF WT=1, then loop here.                                    %
%       ELSEIF AK=1, Then turn off FASTBUS AD line drivers, negate MS,  %
%               and go home.                                            %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDDB:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDDB;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDDB;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_CSR_BROADCAST                                   %
%                                                                       %
%       This routine does a broadcast primary address cycle to          %
%       CSR space.                                                      %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN Release bus, TMR=0, JUMP WAYHOME.                 %
%       ELSEIF WT=1, loop here.                                         %
%       ELSEIF AK=1 THEN turn off FASTBUS AD line drivers, negate MS,   %
%               and go home.                                            %
%       ELSE set AS, TMR (turn on short timer), and loop here.          %
%                                                                       %
%-----------------------------------------------------------------------%
ADDCB:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK
MUX0]
                JUMP ADDCB;
        ELSEIF I5               %FRAK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
```

```
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT3 NDSAK MUX1]
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT3 NDSAK MUX0]
                JUMP ADDCB;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_RELEASE                                         %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%               EG=0 RD=0 MS=0 WT=0 Clear DK,DS,AS
ADDREL:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT4 NDSAK
MUX1]
                JUMP ADDREL;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT CDK CDS CAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT4 NDSAK
MUX0]
                JUMP ADDRELX;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Data Cycle Routines                                             %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_READ                              %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
DATPRR:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRR;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX0]
```

```
                JUMP DATPRRX;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DATPRR;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_WRITE                             %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
DATPRW:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRW;
        ELSEIF I4               %FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRWX;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS0 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DATPRW;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_READ                         %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
DATPSR:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSR;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 DSAK MUX0]
                JUMP DATPSRX;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
```

```
                JUMP DATPSR;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_WRITE                        %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
DATPSW:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSW;
        ELSEIF I4               %FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSWX;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS2 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
                JUMP DATPSW;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_READ                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
DATPBR:                 [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBRX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_WRITE                     %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
DATPBW:                 [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBWX;


%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_TERMINATE                 %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       This routine does a graceful termination of a block transfer    %
%       where DS and DK are brought to their idle states without        %
%       effecting the slave.                                            %
```

```
%               MS=0 RD=0                                                %
%               then                                                     %
%               DS=0                                                     %
DATPBT:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX1]
                JUMP DATPBT;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_LOAD                                      %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
LC_LOAD:        [CLOAD NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                JUMP LC_LOADX;


%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_READ                                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
LC_READ:        [CREAD NSRT NDW COUNTIN NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                JUMP LC_READX;

%-----------------------------------------------------------------------%
%       Routine FIFO_WRITE                                              %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
FIFO_WR:        [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT10 DSAK MUX1]
                JUMP FIFO_WRX;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_BLOCK_TRANSFER_READ                           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
DATFBR:         IF /I6                  %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
```

```
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP DATFBR;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP DATFBRX;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_1  (100ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
DATFP1:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP1;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP1X;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_2  (200ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
DATFP2:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP2;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP2X;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_4  (400ns transfers)           %
%       --FAST-- --READ or WRITE--                                      %
%-----------------------------------------------------------------------%
DATFP4:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
```

```
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP4;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DATFP4X;

%-----------------------------------------------------------------------%
%       SLAVE_DATA_INPUT                                                %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
SLVDIN:                 [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 CWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX0]
                JUMP SLVDINX;

%-----------------------------------------------------------------------%
%       SLAVE_DATA_OUTPUT                                               %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       (WT has been cleared back at LETSGO).                           %
%       IF FBREQ* THEN release bus, jump WAYHOME.                       %
%       ELSEIF WT=1 THEN wait here.                                     %
%       ELSEIF DS=1 THEN set DK jump HOME.                              %
%       ELSE clear DK jump HOME.                                        %

SLVDOUT:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP WAYHOME;
        ELSEIF I7       %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX0]
                JUMP SLVDOUT;
        ELSEIF I1       %FRDS%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT SDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX1]
                JUMP WAYHOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT CDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX1]
                JUMP WAYHOME;
```

```
%-----------------------------------------------------------------------%
%       Null                                                            %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
NULL:           [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT12 NDSAK
MUX1]
                JUMP NULLX;


%-----------------------------------------------------------------------%
%       End Of Event                                                    %
%       --SLOW-- --WRITE --                                             %
%-----------------------------------------------------------------------%
EOE:            [CHOLD NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK DEOE CEOE NTMR NCCLR STAT14 DSAK MUX1]
                JUMP EOEX;


%-----------------------------------------------------------------------%
%       Bulb Test                                                       %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
BULBTST:        [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE IRQ DFFDIS
                HOLDB NEOBA EG RD MS7 XWT SDK SDS SAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT11 NDSAK
MUX1]
                JUMP BULBTST;


%-----------------------------------------------------------------------%
%       Sequencer Select                                                %
%       Wait here to be selected by the 68020                           %
%               FB1S* + FB2S means that the FASTBUS standard sequencer  %
%                       has been selected.                              %
%               FB1S* * FB2S* means that the USER sequencer has been    %
%                       selected.                                       %
%                                                                       %
%       Whichever sequencer has not been selected, puts it's outputs    %
%               into high Z, and is not accessable unless it is reset.  %
%-----------------------------------------------------------------------%
SELECT:         IF /I1 * /I5            %FB1S* * FB2S*%
        THEN [Z] CONTINUE;
        ELSEIF (/I1 * I5) + (I1 * /I5)          %FB2S* + FB1S*%
        THEN [RESETO2] JUMP HOME;
        ELSE [RESETO2] JUMP SELECT;

TURNOFF:[Z] JUMP TURNOFF;
%-----------------------------------------------------------------------%
%       If, in the cycle executed just previous to returning here,      %
%       DSACK was asserted, then the select line (FBxS*)                %
%       will still be asserted (it takes the 68020 and it's PAL's a     %
%       certain amount of time to negate a select line after it sees    %
%       DSACK).  HOME would then see FBxS* true and think that it is    %
```

```
%       being selected again, when, in fact, FBxS* is just lingering     %
%       from the previous cycle.  As you might imagine, if this were     %
%       to happen, the sequencer would jump to what ever address         %
%       just happened to be on the 68020's address bus at that time      %
%       and anarchy would reign.  So, to keep order, if we               %
%       want to come HOME right after DSACK, then we instead return      %
%       here where three NOP type instructions are executed.  This       %
%       allows the 68020 enough time to negate FBxS* from any            %
%       preveously executed cycle.                                       %
%-----------------------------------------------------------------------%
WAYHOME:        [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                CONTINUE;

%       Since the IOD lines could be latched on the next cycle (if there%
%       is a pending instruction), the Processor transceivers must be    %
%       set to DATAOUT before coming HOME.  Any instructions leading to %
%       HOME must have the processor transceivers set in this way or it %
%       will be        bad.                                              %

%       When HOME, all outputs are default                              %
HOME:   IF /I1          %FB1S* (Fast cycle, return DSACK now.)%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 DSAK MUX2]
                JUMP LETSGOF;
        ELSEIF /I5      %FB2S* (Slow cycle, return DSACK in routine.%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX2]
                JUMP LETSGOS;
        ELSE    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%       These are the fast and slow jumps to the appropriate FASTBUS     %
%       primative routines as determined by the vector received from     %
%       the '020 at MUX2.                                                %
%-----------------------------------------------------------------------%
```

```
LETSGOF:        [CDEF NSRT NDW DATAOUTLATCH NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 DSAK MUX2]
                PUSHI;

%       Clear WT here is for Slave write routine, dosn't effect anything
else.%
                [CDEF NSRT NDW DATAOUTHOLD NFDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 CWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX0]
                RETURN;

LETSGOS:        [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX2]
                PUSHI;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT0 NDSAK
MUX0]
                RETURN;

%-----------------------------------------------------------------------%
%       Routine ARBITRATE                                               %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%                                                                       %
%       Starting with something simple, this routine checks to make     %
%       sure that FBREQ is false, then sets FREQ true, waits for        %
%       FRDY (means that we have the bus), then returns DSACK and goes  %
%       home.                                                           %
%-----------------------------------------------------------------------%
ARBX:           [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT1 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Bus Release                                                     %
%-----------------------------------------------------------------------%
%       Release Bus, Status=2                                           %
%                                                                       %
%       Status=2, Status clock = 1                                      %
BUSRELX:        [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT2 NDSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine ADDRESS_RELEASE                                         %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%               Wait for AK to go low, then go HOME.                    %
```

```
ADDRELX:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I5               %FRAK%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT CDK CDS CAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT4 NDSAK MUX0]
                JUMP ADDRELX;
        ELSE    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT4 NDSAK MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%
%       Data Cycle Routines                                             %
%-----------------------------------------------------------------------%
%-----------------------------------------------------------------------%

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_READ                              %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO INTS.         %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers        %
%               (FDOE*=0,DDIR=1,CPBA=0, DSAB=0,DSBA=0,DG*=0), enable    %
%               processor transceivers (POE*=0), Assert RD=1, MS=0 and  %
%               loop here.                                              %
%       ELSEIF DK=1 THEN return DSACK, and continue.                    %
%       ELSE Enable FASTBUS AD line drivers, enable proccessor xceivers %
%               Assert RD=1, MS=0, DS(u), TMR, and loop here.           %
%                                                                       %
%       Hold DSACK true for one more cycle, and continue.               %
%                                                                       %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, NDSACK, JUMP TO       %
%               WAYHOME.                                                %
%       ELSEIF WT=1 THEN negate DSACK, disable processor and FASTBUS    %
%               transceivers, continue to assert RD=1, MS=0, and DS=1,  %
%               and loop here.                                          %
%       ELSEIF DK=0 THEN negate DSACK, disable processor and FASTBUS    %
%               transceivers, deassert TMR, RD, and MS, and go WAYHOME. %
%       ELSE negate DSACK, disable processor and FASTBUS transceivers,  %
%               deassert RD, MS, and DS, assert TMR, and loop here.     %
%                                                                       %
%-----------------------------------------------------------------------%
DATPRRX:        [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX0]
                CONTINUE;

WAIT1:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP WAIT1;
        ELSEIF /I4              %/FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP WAYHOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP WAIT1;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_RANDOM_WRITE                             %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO WAYHOME.      %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers        %
%               (FDOE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=1, DG*=0),       %
%               Assert RD=0, MS=0 and loop here.                        %
%       ELSEIF DK=1 THEN disable FASTBUS transceivers and drivers, and  %
%               continue.                                               %
%       ELSE Assert DS(u), TMR, keep FASTBUS lines asserted, and loop   %
%               here.                                                   %
%       be bop a lula she's my baby                                     %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO HOME.         %
%       ELSEIF WT=1 THEN continue to assert DS=1, and loop here.        %
%       ELSEIF DK=0 THEN go HOME.                                       %
%       ELSE deassert DS, assert TMR, and loop here.                    %
%-----------------------------------------------------------------------%
DATPRWX:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPRWX;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
```

```
                JUMP DATPRWX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_READ                         %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, JUMP TO INTS.         %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers        %
%               (FDOE*=0, DDIR=1,CPBA=0, DSAB=0,DSBA=0,DG*=0), enable   %
%               processor transceivers (POE*=0), Assert RD=1, MS=2 and  %
%               loop here.                                              %
%       ELSEIF DK=1 THEN return DSACK, and continue.                    %
%       ELSE Assert DS(u), TMR, and loop here.                          %
%                                                                       %
%       Hold DSACK true for one more cycle, and continue.               %
%                                                                       %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, DEassert DSACK, jump  %
%               to WAYHOME.                                             %
%       ELSEIF WT=1 THEN negate DSACK, disable processor and FASTBUS    %
%               transceivers, continue to assert RD=1, MS=2, and DS=1,  %
%               and loop here.                                          %
%       ELSEIF DK=0 THEN negate DSACK, disable processor and FASTBUS    %
%               transceivers, deassert RD, and MS, and go WAYHOME.      %
%       ELSE negate DSACK, disable processor and FASTBUS transceivers,  %
%               deassert RD, MS, and DS, assert TMR, and loop here.     %
%-----------------------------------------------------------------------%
DATPSRX:        [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 DSAK MUX0]
                CONTINUE;

WAIT2:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS2 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP WAIT2;
        ELSEIF /I4              %/FRDK%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK MUX1]
                JUMP WAYHOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS2 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
                JUMP WAIT2;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_SEC_ADDRESS_WRITE                        %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
```

```
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, jump to WAYHOME.        %
%       ELSEIF WT=1 THEN Enable FASTBUS transceivers and drivers          %
%               (FDOE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=1, DG*=0),         %
%               Assert RD=0, MS=2 and loop here.                          %
%       ELSEIF DK=1 THEN disable FASTBUS transceivers and drivers, and    %
%               continue.                                                 %
%       ELSE Assert DS(u), keep FASTBUS lines asserted, assert TMR,       %
%               and loop here.                                            %
%       IF FBREQ THEN clear FASTBUS lines, TMR=0, jump to HOME.           %
%       ELSEIF WT=1 THEN continue to assert DS=1, and loop here.          %
%       ELSEIF DK=0 THEN go HOME.                                         %
%       ELSE deassert DS, assert TMR, and loop here.                      %
%-----------------------------------------------------------------------%
DATPSWX:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK
MUX0]
                JUMP DATPSWX;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT6 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT6 NDSAK MUX0]
                JUMP DATPSWX;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_READ                        %
%       --SLOW-- --READ--                                                 %
%-----------------------------------------------------------------------%
%       This routine is going to be a little bit different than any of    %
%               the others, in that, DSACK will have to be returned       %
%               after each DK(t) to allow the 68020 to read the data.     %
%       Here goes:                                                        %
%                                                                         %
%       Turn on FASTBUS AD line receivers, Assert RD=1, MS=1,switch       %
%               to MUX1 and continue.                                     %
%       IF FBREQ THEN Release bus, JUMP TO INTS.                          %
%       ELSEIF WT=1 THEN Assert RD=1, MS=1, on FASTBUS  Turn on FASTBUS %
%               xceivers and enable processor xceivers                    %
%               (POE*=0, DDIR=1, CPBA=0, DSAB=0, DSBA=0, DG*=0, FDOE*=0)%
%               and loop here.                                            %
%       ELSEIF DS=0 THEN Assert RD=1, MS=1, DS(u), enable FASTBUS and     %
%               processor transceivers (POE*=0, DDIR=1, CPBA=0,           %
%               DSAB=0, DSBA=0, DG*=0, FDOE*=0)and assert TMR, and JUMP %
%               to DKUPR.                                                 %
%       ELSE Assert RD=1, MS=1, DS(d), enable FASTBUS and processor       %
%               transceivers (POE*=0, DDIR=1, CPBA=0, DSAB=0,             %
```

```
%               DSBA=0, DG*=0, FDOE*=0) assert TMR, and JUMP to DKDWNR.  %
%DKDWNR                                                                  %
%       IF FBREQ THEN RD=1, MS=1, Clear DS, jump to INTS.                %
%       ELSEIF DK=0 THEN Assert DSACK, continue to drive FASTBUS lines, %
%               and processor and FASTBUS transceivers, deassert TMR,    %
%               and JUMP to next1.                                       %
%       ELSE Continue to drive FASTBUS lines, and transceivers, and      %
%               TMR, and loop here.                                      %
%DKUPR                                                                   %
%       IF FBREQ THEN RD=1, MS=1, Clear DS, assert DSACK, jump to        %
%               NEXT2.                                                   %
%       ELSEIF DK=1 THEN Assert DSACK, continue to drive FASTBUS lines, %
%               and processor and FASTBUS transceivers, deassert TMR,    %
%               and JUMP to next1.                                       %
%       ELSE Continue to drive FASTBUS lines, and transceivers,          %
%               assert TMR, and loop here.                               %
%NEXT1                                                                   %
%       Continue to drive FASTBUS lines, DSACK, and transceivers, and    %
%               Continue.                                                %
%       Continue to drive FASTBUS lines, and transceivers negate DSACK   %
%               and JUMP WAYHOME.                                        %
%                                                                        %
%NEXT2  Clear FASTBUS lines, release bus, keep DSACK asserted, jump       %
%               to WAYHOME.                                              %
%------------------------------------------------------------------------%
DATPBRX:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBRX;
        ELSEIF /I1              %/FRDS%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPR;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNR;

DKDWNR:         IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX1]
```

```
                JUMP NEXT1;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNR;

DKUPR:  IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 DSAK
MUX1]
                JUMP NEXT2;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX1]
                JUMP NEXT1;
        ELSE    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPR;

NEXT1:          [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 DSAK MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP WAYHOME;

NEXT2:          [CTRLDEF
                REL
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine DATA_PROCESSOR_BLOCK_TRANSFER_WRITE                     %
%       --FAST-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       As with DATPBR, DSACK will have to be returned after each DK(t) %
%               to allow the 68020 to output new data.                  %
%                                                                       %
%       Keep FASTBUS AD line drivers enabled, assert MS=1, set MUX1,    %
%               and continue.                                           %
%       IF FBREQ THEN Release bus, JUMP TO WAYHOME.                     %
%       ELSEIF WT=1 THEN Assert MS=1, keep FASTBUS transceivers enabled,%
%               and enable FASTBUS drivers (POE*=1, DDIR=0,             %
%               CPBA=0, DSAB=0, DSBA=1, DG*=0, FDOE*=0), and loop here. %
%       ELSEIF DS=0 THEN Assert MS=1, DS(u), TMR, keep FASTBUS          %
%               transceivers enabled and enable FASTBUS drivers (POE*=1,%
%               DDIR=0, CPBA=0, DSAB=0, DSBA=1, DG*=0, FDOE*=0), and    %
%               JUMP to DKUPW.                                          %
%       ELSE Assert MS=1, DS(d), TMR, keep FASTBUS transceivers enabled,%
%               and enable FASTBUS drivers (POE*=1, DDIR=0,             %
%               CPBA=0, DSAB=0, DSBA=1, DG*=0, FDOE*=0), and JUMP to    %
```

```
%               DKDWNW.                                                 %
%DKDWNW                                                                 %
%       IF FBREQ THEN Release bus, JUMP TO HOME.                        %
%       ELSEIF DK=0 THEN deassert MS, TMR, disable FASTBUS transceivers %
%               and drivers (POE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=0,    %
%               DG*=1, FDOE*=1), and JUMP home.                         %
%       ELSE continue to assert MS, TMR, and keep FASTBUS transceivers  %
%               and drivers enabled, and loop here.                     %
%DKUPW                                                                  %
%       IF FBREQ THEN Release bus, JUMP TO HOME.                        %
%       ELSEIF DK=1 THEN deassert MS, TMR, disable FASTBUS transceivers %
%               and drivers (POE*=0, DDIR=0, CPBA=0, DSAB=0, DSBA=0,    %
%               DG*=1, FDOE*=1), and JUMP home.                         %
%       ELSE continue to assert MS, TMR, and keep FASTBUS transceivers  %
%               and drivers enabled, and loop here.                     %
%-----------------------------------------------------------------------%
DATPBWX:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK
MUX0]
                JUMP DATPBWX;
        ELSEIF /I1              %/FRDS%
        THEN    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK SDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPW;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNW;

DKDWNW:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %/FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKDWNW;

DKUPW:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
```

```
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I4               %FRDK%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT5 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CDEF NSRT NDW DATAOUTHOLD FDOE FCOE NIRQ DFFDIS
                HOLDB NEOBA NEG NRD MS1 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT5 NDSAK MUX0]
                JUMP DKUPW;

%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_LOAD                                      %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       This routine loads the Local Word Counter with data from        %
%       the processor.  Only the lower 12 bits of the 32 bit word       %
%       are used.                                                       %
%                                                                       %
%       Set POE* true, put Local Word Counter into load mode,           %
%               assert DSACK and continue.                              %
%       Negate POE*, put Local Word Counter into hold, keep DSACK       %
%               asserted, and continue.                                 %
%       Negate DSACK (hold DSACK false for two cycles to prevent        %
%               sequencer from seeing FBSEL from this cycle when        %
%               returning home).                                        %
%                                                                       %
LC_LOADX:       [CHOLD NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 NDSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine LOCAL_COUNTER_READ                                      %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       Routine to allow the processor to read the value of the Local   %
%               Word Counter.                                           %
%                                                                       %
%       Set POE* true, put Local Word Counter into Read mode, assert    %
%               DSACK, and continue.                                    %
%       Keep all outputs the same for one more cycle.                   %
%       Negate DSACK (hold DSACK false for two cycles to prevent        %
%               sequencer from seeing FBSEL from this cycle when        %
%               returning home).                                        %
%                                                                       %
LC_READX:       [CREAD NSRT NDW COUNTIN NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT9 DSAK MUX1]
                JUMP WAYHOME;
```

```
%-----------------------------------------------------------------------%
%       Routine FIFO_WRITE                                              %
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       Routine to write a word from the 68020 into the processor       %
%       and data FIFO's.                                                %
%                                                                       %
%       Enable processor transceivers (POE*=0), set FIFO Write true     %
%               (DW=1), enable FIFO's, set DSACK true.                  %
%       Set FCLK true.                                                  %
%       Negate DSACK, DW, FCLK, and POE*, JUMP TO WAYHOME.              %
%                                                                       %
FIFO_WRX:       [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFEN NSR SCLK FCLK NDEOE NCEOE NTMR NCCLR STAT10 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_BLOCK_TRANSFER_READ                           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
%       This routine does a FASTBUS handshake block transfer read from  %
%       the FASTBUS crate segment to the FSCC's Output Port Data FIFO,  %
%       and also the Processor Data FIFO.                               %
%-----------------------------------------------------------------------%
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFBR:IF FBREQ                                                        %
%       THEN release bus jump to INTF                                   %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers, enable processor and Data FIFO's  %
%               MS=1, RD=1, Place local word counter into hold, and     %
%               loop here.                                              %
%       ELSE Turn on FASTBUS xcievers, enable processor and Data FIFO's %
%               MS=1, RD=1, Place local word counter into count, set    %
%               DS=1, NO FCLK!, and jump to DATFBRX.                    %
%                                                                       %
%BXFERLP:IF (/FRDK * FRSS1) + EOB + FBREQ                               %
%       THEN Enable FASTBUS xceivers, enable processor and Data FIFO's  %
%               MS=1, RD=1, Place local word counter into hold mode,    %
%               and JUMP to EOBIRQ.                                     %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers, enable processor and Data FIFO's  %
%               MS=1, RD=1, Place local word counter into hold, and     %
%               loop here.                                              %
%       ELSEIF DK=0                                                     %
%       THEN Turn on FASTBUS xcievers, enable processor and Data FIFO's %
%               MS=1, RD=1, Place local word counter into count, set    %
%               DS=1, set FCLK, and continue.                           %
%       ELSE Place local word counter into hold, and loop here.         %
%                                                                       %
%DATFBRX:Place counter into hold mode, negate FCLK, set TMR, and        %
%               continue.  This       cycle ensures that the FSCC's minimum %
%               FASTBUS cycle time (100ns) is not violated.  If some    %
%               slave returns DK too quickly, the FSCC could            %
```

```
%               try to do another DS before the FIFO's were ready for    %
%               the next word.  This cycle keeps the sequencer from      %
%               looking for DK before it should.                         %
%                                                                        %
%       Keep TMR true, and do another NOP for the same reason as above.  %
%                                                                        %
%       IF (FRDK* FRSS1) + EOB + FBREQ                                   %
%       THEN counter=hold, NTMR, keep FIFO's enabled and jump to EOBIRQ.%
%       ELSEIF WT=1                                                      %
%       THEN counter=hold, NTMR, keep both FIFO's enabled and loop here.%
%       ELSEIF DK=1                                                      %
%       THEN counter=decr, set DS=0, set FCLK, keep FIFO's enabled,      %
%               Clear TMR, and continue.                                 %
%       ELSE counter=hold, TMR, keep FIFO's enabled, and loop here.      %
%                                                                        %
%       Place local word counter in hold, negate FCLK, keep processor    %
%               and oport FIFO's enabled, set TMR, and JUMP to DATFBR.   %
%               This cycle ensures that the FSCC's minimum cycle time    %
%               is not violated in the same way as above.                %
%       Keep TMR set, do another NOP for the same reason as above, and   %
%               jump to BXFERLP.                                         %
%                                                                        %
%EOBIRQ:IF FBREQ THEN Release bus, jump to HOME.                          %
%       ELSEIF DS=1                                                      %
%       THEN counter=hold, TMR, continue.                                %
%       ELSE counter=hold, TMR, JUMP DSLOWH.                             %
%                                                                        %
%       --case DS=1--                                                    %
%       IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF DK=1 THEN Clear TMR, continue.                            %
%       ELSE keep TMR true, and wait here.                               %
%                                                                        %
%       Wait two cycles to allow for FIFO write.                         %
%                                                                        %
%       IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF WT=1                                                      %
%       THEN RD=1, MS=0, counter=hold, FIFO's disabled loop here.        %
%       ELSE DS=0, RD=0, MS=0, FIFO's Disabled continue.                 %
%                                                                        %
%       IF FBREQ THEN Release bus, jump HOME.                            %
%       ELSEIF DK=0 THEN NTMR, EOBA, IRQ, jump HOME.                     %
%       ELSE TMR, wait here.                                             %
%                                                                        %
%DSLOWH:IF FBREQ THEN Release bus, jump to HOME.                          %
%       ELSEIF DK=0 THEN clear TMR, continue.                            %
%       ELSE TMR, wait here.                                             %
%                                                                        %
%       wait three cycles to ensure FIFO write, then IRQ, EOBA, jump     %
%       home.                                                            %
%                                                                        %
BXFERLP:IF (/I4 * I0) + I3 + /I6      %(/FRDK * FRSS1) + FEOB + FBREQ*%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP EOBIRQ;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
```

```
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP BXFERLP;
        ELSEIF /I4              %/FRDK%
        THEN    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP BXFERLP;

DATFBRX:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

WAIT3:  IF (I4 * I0) + I3 + /I6                %(FRDK * FRSS1) + FEOB + FBREQ*%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP EOBIRQ;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                JUMP WAIT3;
        ELSEIF I4               %FRDK%
        THEN    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP WAIT3;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP BXFERLP;

EOBIRQ:         IF /I0          %/FRSS1%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
```

```
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

        IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX0]
                JUMP INTF;
        ELSEIF I1               %FRDS%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP DSLOWH;

%       Case--DS=1
DSHIGHH:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX0]
                JUMP INTF;
        ELSEIF I4               %FRDK%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP DSHIGHH;

                [CHOLD NSRT NDW DATAINFIFO NFDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO NFDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

WAIT5:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
```

```
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK
MUX0]
                JUMP WAIT5;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG NRD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                CONTINUE;

WAIT6:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %FRDK=0%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE IRQ DFFDIS
                HOLDB EOBA NEG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP WAIT6;

%       Case--DS=0
DSLOWH:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %FRDK=0%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT7 NDSAK MUX0]
                JUMP DSLOWH;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE IRQ DFFEN
                HOLDB EOBA NEG RD MS1 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT7 NDSAK MUX1]
                JUMP HOME;
```

```
%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_1  (100ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
%       FASTBUS pipeline block transfer read from the FASTBUS crate     %
%       segment to the FSCC's Output Port Data FIFO, and also the       %
%       Processor Data FIFO if enabled through the parrallel port.      %
%                                                                       %
%       Set FASTBUS xcievers for input (DDIR=1, SAB=1, FDOE*=0),        %
%               enable data and processor FIFO's (DFIFOEN=1, PFIFOEN=1),%
%               MS=3, RD=1, and continue.                               %
%                                                                       %
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFP1:IF FBREQ THEN Release bus, jump to INTS.                         %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into hold, and loop here.                  %
%       ELSE Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into count, set DS=1, and JUMP to DATFP1X:  %
%                                                                       %
%PXFR1LP:IF FBREQ THEN Release bus, jump to INTS.                        %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into hold, and loop here.                  %
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into hold mode, and JUMP to EOBPIPE.       %
%       ELSE Enable FASTBUS xceivers for input, MS=3, RD=1, Place local %
%               word counter into count, set DS=1, set FCLK, and        %
%               continue.                                               %
%                                                                       %
%DATFP1X:Place counter into hold mode, negate FCLK, and                 %
%               continue.  This       cycle provides the FSCC's         %
%               FASTBUS cycle time (100ns).                             %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF WT=1                                                     %
%       THEN counter=hold, keep both FIFO's enabled and loop here.      %
%       ELSEIF EOB                                                      %
%       THEN counter=hold, keep FIFO's enabled and jump to EOBPIPE.     %
%       ELSE counter=decr, set DS=0, set FCLK, keep FIFO's enabled,     %
%               and continue.                                           %
%                                                                       %
%       Place local word counter in hold, negate FCLK, keep processor   %
%               and oport FIFO's enabled, and JUMP to PXFR1LP.          %
%               This cycle provides the FSCC's cycle time.              %
%                                                                       %
%EOBPIPE:IF SS <> 2 (Local counter terminated the block transfer)       %
%       THEN FCLK continue.                                             %
%       ELSE (SS=2 terminated the block transfer) continue.             %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DS=1                                                     %
%       THEN counter=hold, TMR, continue.                               %
%       ELSE counter=hold, TMR, JUMP DSLOWP.                            %
```

```
%                                                                       %
%       --case  DS=1--                                                  %
%DSHIGHP:IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=1 THEN Clear TMR, continue.                           %
%       ELSE keep TMR true, and wait here.                              %
%                                                                       %
%       Wait two cycles to allow for FIFO write.                        %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF WT=1 THEN RD=1, counter=hold, disable FIFO's, loop here. %
%       ELSE RD=0, MS=0, DS=0, disable FIFO', continue.                 %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=0 THEN clear TMR, EOBA, IRQ, continue.                %
%       ELSE TMR, wait here.                                            %
%                                                                       %
%       --case  DS=0--                                                  %
%DSLOWP:IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF DK=0 THEN clear TMR, CONTINUE.                           %
%       ELSE TMR, wait here.                                            %
%                                                                       %
%       Wait three cycles to ensure FIFO write, then IRQ, EOBA, jump    %
%       home.                                                           %
%                                                                       %
PXFR1LP:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR1LP;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

DATFP1X:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT7:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
```

```
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT7;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR1LP;

EOBPIPE:IF /I0          %/FRSS1%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

        IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I1               %FRDS%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                JUMP DSLOWP;

%       Case  DS=1                                                      %
DSHIGHP:IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I4               %FRDK%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;
```

```
        ELSE    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                JUMP DSHIGHP;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT9:  IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK
MUX0]
                JUMP WAIT9;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK CDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT10:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4      %FRDK=0%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE IRQ DFFDIS
                HOLDB EOBA NEG NRD MS0 XWT XDK XDS XAS
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR          SAK MUX1]
                JUMP HOME;
        ELSE    [CTRLDEF
                HOLDB NEOBA NEG RD MS0 XWT XDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT10;

%       Case  DS=0                                                      %
DSLOWP:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF /I4              %FRDK=0%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
```

```
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                CONTINUE;
        ELSE    [CHOLD NSRT NDW DATAINFIFO NFDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP DSLOWP;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE IRQ DFFEN
                HOLDB EOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE TMR NCCLR STAT8 NDSAK MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_2  (200ns transfers)           %
%       --FAST-- --WRITE or READ--                                      %
%-----------------------------------------------------------------------%
%       This routine does a FASTBUS pipeline block transfer read from   %
%       the FASTBUS crate segment to the FSCC's Output Port Data FIFO,  %
%       and also the Processor Data FIFO.                               %
%                                                                       %
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFP2:IF FBREQ THEN Release bus, JUMP WAYHOME.                        %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, Enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold, and loop here.%
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold mode, and      %
%               JUMP to EOBPIPE.                                        %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into count, set DS=1,    %
%               and JUMP to DATFP2X.                                    %
%                                                                       %
%PXFR2LP:IF FBREQ THEN Release bus, JUMP WAYHOME.                       %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, Enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold, and loop here.%
%       ELSEIF EOB                                                      %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into hold mode, and      %
%               JUMP to EOBPIPE.                                        %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, MS=3,    %
%               RD=1, Place local word counter into count, set DS=1,    %
%               set FCLK, and continue.                                 %
```

```
%                                                                       %
%DATFP2X:Place counter into hold mode, negate FCLK, and                 %
%               continue.  These cycles provide the FSCC's              %
%               FASTBUS cycle time (200ns).                             %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       IF FBREQ THEN Release bus, jump to HOME.                        %
%       ELSEIF WT=1                                                     %
%       THEN counter=hold, keep both FIFO's enabled and loop here.      %
%       ELSEIF EOB                                                      %
%       THEN counter=hold, keep FIFO's enabled and jump to EOBPIPE.     %
%       ELSE counter=decr, set DS=0, set FCLK, keep FIFO's enabled,     %
%               and continue.                                           %
%                                                                       %
%       Place local word counter in hold, negate FCLK, keep processor   %
%               and oport FIFO's enabled, and Continue.                 %
%               These cycles provide the FSCC's cycle time.             %
%                                                                       %
%       Continue (add 50ns to cycle time).                              %
%                                                                       %
%       Jump to PXFR2LP. (add 50ns to cycle time).                      %
%                                                                       %
PXFR2LP:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR2LP;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

DATFP2X:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
```

```
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT11:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT11;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR2LP;

%-----------------------------------------------------------------------%
%       Routine DATA_FIFO_PIPELINED_READ_4  (400ns transfers)           %
%       --FAST-- --READ or WRITE--                                      %
%-----------------------------------------------------------------------%
%       This routine does a FASTBUS pipeline block transfer read from   %
%       the FASTBUS crate segment to the FSCC's Output Port Data FIFO,  %
%       and also the Processor Data FIFO.                               %
%                                                                       %
%       Note: DDIR, and SAB must be held high throughout the block      %
%               transfer operation.                                     %
%                                                                       %
%DATFP4:IF FBREQ THEN Release bus, jump to WAYHOME.                      %
%       ELSEIF WT=1                                                     %
%       THEN Enable FASTBUS xceivers for input, enable data and         %
%               processor FIFO's, MS=3, RD=1, Place local word counter  %
%               into hold, and loop here.                               %
%       ELSEIF EOB                                                      %
```

```
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, Place     %
%               local word counter into hold mode, MS=3, RD=1, and JUMP %
%               to EOBPIPE.                                              %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, Place     %
%               local word counter into count, MS=3, RD=1, set DS=1,     %
%               and JUMP to DATFP4X.                                     %
%                                                                        %
%PXFR4LP:IF FBREQ THEN Release bus, jump to WAYHOME.                      %
%       ELSEIF WT=1                                                      %
%       THEN Enable FASTBUS xceivers for input, enable data and          %
%               processor FIFO's, MS=3, RD=1, Place local word counter   %
%               into hold, and loop here.                                %
%       ELSEIF EOB                                                       %
%       THEN Enable FASTBUS xceivers for input, enable FIFO's, Place     %
%               local word counter into hold mode, MS=3, RD=1, and JUMP %
%               to EOBPIPE.                                              %
%       ELSE Enable FASTBUS xceivers for input, enable FIFO's, Place     %
%               local word counter into count, MS=3, RD=1, set DS=1,     %
%               set FCLK, and continue.                                  %
%                                                                        %
%       Place counter into hold mode, negate FCLK, and                   %
%               continue.  These cycles provide the FSCC's               %
%               FASTBUS cycle time (400ns).                              %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       IF FBREQ THEN Release bus, jump to HOME.                         %
%       ELSEIF WT=1                                                      %
%       THEN counter=hold, keep both FIFO's enabled and loop here.       %
%       ELSEIF EOB                                                       %
%       THEN counter=hold, keep FIFO's enabled and jump to EOBPIPE.      %
%       ELSE counter=decr, set DS=0, set FCLK, keep FIFO's enabled,      %
%               and continue.                                            %
%                                                                        %
%       Place local word counter in hold, negate FCLK, keep processor    %
%               and oport FIFO's enabled, and Continue.                  %
%               These cycles provide the FSCC's cycle time.              %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
%                                                                        %
%       Continue (add 50ns to cycle time).                               %
```

```
%                                                                        %
%       Jump to PXFR4LP.(add 50ns to cycle time).                        %
%                                                                        %
PXFR4LP:IF /I6          %FBREQ*%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP PXFR4LP;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK SDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

DATFP4X:        [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;


                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
```

```
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

WAIT12:         IF /I6          %FBREQ*%
        THEN    [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFDIS
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTF;
        ELSEIF I7               %FRWT%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP WAIT12;
        ELSEIF I3               %FEOB%
        THEN    [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                JUMP EOBPIPE;
        ELSE    [COUNT NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK CDS XAS
                PFFEN NSR NSCLK FCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
                CONTINUE;

                [CHOLD NSRT NDW DATAINFIFO FDOE FCOE NIRQ DFFEN
                HOLDB NEOBA NEG RD MS3 XWT XDK XDS XAS
                PFFEN NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT8 NDSAK MUX0]
```

```
                JUMP PXFR4LP;

%-----------------------------------------------------------------------%
%       SLAVE_DATA_INPUT                                                %
%       --SLOW-- --READ--                                               %
%-----------------------------------------------------------------------%
%       Enable FASTBUS AD line receivers, Clear WT, jump to SLVDINX.    %
%                                                                       %
%       IF FBREQ then release bus, jump to INTS.                        %
%       ELSEIF WT=1, THEN wait here.                                    %
%       ELSEIF DS=1 THEN latch FASTBUS AD lines, set DK, DSACK,         %
%               continue.                                               %
%       ELSE (DS=0) THEN latch FASTBUS AD lines, clear DK, DSACK,       %
%               continue.                                               %
%                                                                       %
%       DSACK, Hold latched FASTBUS data, jump WAYHOME.                 %
SLVDINX:IF /I6          %FBREQ%
        THEN    [CTRLDEF
                REL
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP INTS;
        ELSEIF I7       %FRWT%
        THEN    [CDEF NSRT NDW DATAINPROC FDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 NDSAK
MUX0]
                JUMP SLVDINX;
        ELSEIF I1       %FRDS%
        THEN    [CDEF NSRT NDW DATAINLATCH FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT SDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 DSAK
MUX0]
                CONTINUE;
        ELSE    [CDEF NSRT NDW DATAINLATCH FDOE FCOE NIRQ DFFDIS
                NREQ NEOBA NEG NRD MS0 XWT CDK XDS XAS
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 DSAK
MUX0]
                CONTINUE;

                [CDEF NSRT NDW DATAINHOLD NFDOE FCOE NIRQ DFFDIS
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT13 DSAK MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       Null                                                            %
%       --FAST-- WRITE or READ--                                        %
%-----------------------------------------------------------------------%
NULLX:          [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT12 NDSAK
MUX1]
                JUMP WAYHOME;

%-----------------------------------------------------------------------%
%       End Of Event                                                    %
```

```
%       --SLOW-- --WRITE--                                              %
%-----------------------------------------------------------------------%
%       --Two dummy words--                                             %
%       DW (Data FIFO Write), DEOE, CEOE, DFFEN, Set FASTBUS interface  %
%               transceivers for output to FIFO (Data and Processor).   %
%               DSACK, continue.                                        %
%                                                                       %
%       Hold all lines for one more cycle (make sure that the write     %
%               pulse to the FIFO's is long enough).                    %
%                                                                       %
%       Hold DEOE true for one more cycle while DW is false.  This      %
%               provides the "hold" time for the fifo's.                %
%                                                                       %
%       Set DW, for two cycles to insert another dummy word with DEOE   %
%               false.                                                  %
%                                                                       %
%       Hold DEOE true for one more cycle while DW is false, to         %
%               ensure that the fifo data hold time spec is met.        %
%                                                                       %
EOEX:           [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK DEOE CEOE NTMR NCCLR STAT14 DSAK MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK DEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT DW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                CONTINUE;

                [CDEF NSRT NDW DATAOUT NFDOE FCOE NIRQ DFFEN
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT14 NDSAK
MUX1]
                JUMP HOME;

%-----------------------------------------------------------------------%
%       Interrupt handler                                               %
%-----------------------------------------------------------------------%
%       Being interrupted means that there was a FASTBUS error, or      %
%       the RB (Reset Bus) line has been driven true.  In either case   %
%       the sequencer's assigned task is simply to abort any operation  %
%       it is currently doing, and return to idle state.                %
%                                                                       %
%       Returning DSACK to the 68020, is a prcisely timed function      %
```

```
%       which the sequencer must perform.  DSACK is asserted true        %
%       for exactly two sequencer cycles.  Then it is deasserted, and    %
%       the FBxS* lines may not be sampled for three sequencer cycles.   %
%       What this means is that when a routine wants to return DSACK     %
%       and return to idle, it must do three nop type cycles before      %
%       testing either of the two FBxS* lines.                           %
%                                                                        %
%       The interrupt handler is a strange routine since the IRQ line    %
%       is tested at almost every "IF" instruction there is.  If IRQ is  %
%       true at any of these places, the IH must make sure that DSACK    %
%       is returned with the proper timing in mind.                      %
%                                                                        %
INTS:           [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 DSAK MUX1]
                CONTINUE;

                [CTRLDEF
                FBDEF
                PFFDIS NSR NSCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 DSAK
MUX1]
                JUMP WAYHOME;

INTF:           [CTRLDEF
                FBDEF
                PFFDIS NSR SCLK NFCLK NDEOE NCEOE NTMR NCCLR STAT15 NDSAK
MUX1]
                JUMP WAYHOME;
```

FF_CTRL.ABL-2/28/90

```
MODULE FB_CTRL_FF
TITLE 'FB_CTRL_FF        VERSION 14-AUG-89 1000 hrs'
FF_CTRL DEVICE 'P22V10';

FSAS    PIN     1;
FCAS    PIN     2;
FSDS    PIN     3;
FCDS    PIN     4;
FSDK    PIN     5;
FCDK    PIN     6;
!SPOE   PIN     7;
SLCOE   PIN     8;
!SDG    PIN     9;
!SR     PIN     10;
DDIR    PIN     11;
!FCOE   PIN     13;
!DG     PIN     14;
!LCOE   PIN     15;
!POE    PIN     16;
!FB2S   PIN     18;
!FB1S   PIN     19;
!PWRITE         PIN     20;
FDDK    PIN     21;
FDDS    PIN     22;
FDAS    PIN     23;

EQUATIONS

FDAS    =       (FSAS # (FDAS & !FCAS)) & FCOE;

FDDS    =       (FSDS # (FDDS & !FCDS)) & FCOE;

FDDK    =       (FSDK # (FDDK & !FCDK)) & FCOE;

"       Equations to prevent bus conflicts on the IOD bus, by preventing
"       any two enables from being true at one time.
"       POE* is only true during a processor write when selected (SPOE &
"       PWRITE & (FB1S # FB2S)), when the 646's (!(SDG & DDIR)), the Data
"       FIFO (!SR), and the Local Counter (!SLCOE), are not driving the IOD
"       bus.  POE* is also true during processor reads (# (SPOE & !PWRITE).
POE     = ((SPOE & PWRITE) & !(SDG & DDIR) & !SR & !SLCOE # (SPOE & !PWRITE))
& (FB1S # FB2S);

"       In similar fashion to POE*, LCOE is only true during processor reads
"       (!(SPOE & PWRITE)), when the 646's !(SDG & DDIR), and the Data FIFO
"       (!SR), are not enabled.
LCOE    =       SLCOE & !(SDG & DDIR) & !SR & !(SPOE & PWRITE);

"       Again, as above, DG* (646's) can only be true during processor reads
"       (!(SPOE & PWRITE)), when the Data FIFO (!SR), and the Local Counter
"       (!SLCOE), are not enabled.  Also, DG* can be true when driving the
"       FASTBUS AD lines (# (SDG & !DDIR)).
DG      = (SDG & DDIR) & !SR & !SLCOE & !(SPOE & PWRITE) # (SDG & !DDIR);

END FB_CTRL_FF
```

FIFO_CTR.ABL-2/28/90

```
MODULE FIFO_CTRL
TITLE 'FIFO_CTRL        VERSION 28-FEB-90 1500 hrs'
FIFO_CTR DEVICE 'P22V10';

CLK     PIN     1;
DFIFOEN         PIN     2;
PFIFOEN PIN     3;
COPYEN  PIN     4;
FRDK    PIN     5;
FW      PIN     6;
SDCPAB  PIN     7;
FRSS1   PIN     8;

DKT     PIN     15;
FRDKZ   PIN     16;
SA      PIN     17;
SB      PIN     18;
SC      PIN     19;
SD      PIN     20;
DCPAB   PIN     21;

!DW     PIN     23;
!PW     PIN     22;

NSTATE  =       [SA,SB]
PSTATE  =       [SC,SD]
S0      =       [0,0];
S1      =       [0,1];
S2      =       [1,0];
S3      =       [1,1];

EQUATIONS

FRDKZ   :=      FRDK;

DKT     :=      ((!FRSS1 & (FRDK != FRDKZ)) # DKT) & !(NSTATE == 0);

WHEN (NSTATE == S0) & ((!FRSS1 & (FRDK != FRDKZ)) # DKT) THEN NSTATE := S1;

WHEN (NSTATE == S1) THEN NSTATE := S2;

WHEN (NSTATE == S2) THEN NSTATE := S3;

WHEN (NSTATE == S3) THEN NSTATE := S0;

DW      :=      ((((NSTATE == S0) & ((!FRSS1 & (FRDK != FRDKZ)) # DKT))
                # (NSTATE == S1)
                # (NSTATE == S2))
                & (DFIFOEN == 1))
                # (FW & DFIFOEN);

WHEN ((PSTATE == S0) & (!COPYEN)) # ((PSTATE == S3) & (!PFIFOEN))
THEN PSTATE := S0;

WHEN ((PSTATE == S1) & (PFIFOEN)) # ((PSTATE == S0) & (COPYEN))
THEN PSTATE := S1;
```

```
WHEN ((PSTATE == S2) & (!PFIFOEN)) # ((PSTATE == S1) & (!PFIFOEN))
THEN PSTATE := S2;

WHEN ((PSTATE == S3) & (PFIFOEN)) # ((PSTATE == S2) & (PFIFOEN))
THEN PSTATE := S3;

PW      :=      ((((NSTATE == S0) & ((!FRSS1 & (FRDK != FRDKZ)) # DKT))
                # (NSTATE == S1)
                # (NSTATE == S2))
                & (PSTATE == S3))
                # (FW & PFIFOEN);

DCPAB   :=      ((NSTATE == S0) & ((DFIFOEN == 1) # (PFIFOEN == 1))
                & ((!FRSS1 & (FRDK != FRDKZ)) # DKT))
                # SDCPAB;

END FIFO_CTRL
```

HEADAV2.ADF-7/11/90

```
module counter
Gustavo I. Cancelo
Fermilab
10/9/89
title header latch & 12 bit event total counter
%       Changed PL equation from registered output to combinatorial.  RK %
%       11-dec-89  1400hrs%
%       --Added state 7 (CLEAR2) to the state machine to lengthen the clear%
%       pulse to the counter.  11-JUL-90  RK%

PART: EPB1400

INPUTS: DS@25, CS@12, A1@13, RW@36, RESET@9, EOE@8, CK@14, CLK@7

OUTPUTS: CARRY@17, CKE@6, WRITE@37, RC@38
         P0@24, P1@23, P2@22, P3@21, P4@20, P5@19, P6@18, %counter%
         DAT0@26, DAT1@27, DAT2@28, DAT3@29, DAT4@32, DAT5@33, DAT6@34,
DAT7@35
         Q3@39, Q2@40, Q1@1, Q0@2, PL@5

NETWORK:
        DS=INP(DS)
        CS=INP(CS)
        A1=INP(A1)
        RW=INP(RW)
        RESET=INP(RESET)
        EOE=INP(EOE)
        CK=INP(CK)
        CLK=INP(CLK)

% data tranceiver bus %
DAT0,DAT1,DAT2,DAT3,DAT4,DAT5,DAT6,DAT7 = BUSX (IBUS,,XOE)

% low order counter. input register #1 LBUSI %
B0,B1,B2,B3,B4,B5,B6,B7 = LBUSI (IBUS,DS,WE1)

% control register. input register #2 LBUSI %
SE0,SE1,RF,WRT,QE2,QE1,QE0,X0 = LBUSI (IBUS,DS,WE2)

% low order counter.  output register #1 %
IBUS = LBUSO(P0,P1,P2,P3,P4,P5,P6,CARRY,,OE1,RE1)

% status register. output register #2 %
IBUS = LBUSO (S0,S1,Q0,Q1,Q2,Q3,RC,WRITE,,OE2,RE2)

%macrocell connection %
S0 = NOCF(S0c)
S1 = NOCF(S1c)

RC,RC = RORF(RCd,CLK,,,)          %software reset macrocell%

Q3,Q3 = RORF(Q3d,CLK,RESET,,)

Q2,Q2 = RORF(Q2d,CLK,RESET,,)
Q1,Q1 = RORF(Q1d,CLK,RESET,,)
Q0,Q0 = RORF(Q0d,CLK,RESET,,)
```

```
%PL,PL = RORF(PLd,CLK,RES,,)        %%enables Preload/Transparent modes for
HEAD1%
PL,PL = COCF(PLd,)     %Changed from registered to combinatorial rk 11-dec-89%

WRITE,WRITE = RORF(WRITEd,CLK,RESET,,WOE)  %control FIFO's write pulse%

CKE = RONF(CKEd,CLK,RES,,)              %Fastbus clock enable for HEAD1%

% counter macrocells %
P0,P0 = TOTF(P0t,RELO,RES,,)
P1,P1 = TOTF(P1t,RELO,RES,,)
P2,P2 = TOTF(P2t,RELO,RES,,)
P3,P3 = TOTF(P3t,RELO,RES,,)
P4,P4 = TOTF(P4t,RELO,RES,,)
P5,P5 = TOTF(P5t,RELO,RES,,)
P6,P6 = TOTF(P6t,RELO,RES,,)

CARRY,CARRY = COCF(CARRYc,)

EQUATIONS:
          RE1 = /DS * /CS * RW * /A1;       % low_counter read enable %
          RE2 = /DS * /CS * RW *  A1;       % status register read enable %
          % RE = RE1 + RE2  read function is activated for either RE1 or RE2%

          WE1 = /RW * /CS * /A1;            % low_counter write enable %
          WE2 = /RW * /CS * A1;             % control register write enable %

          OE1 = /A1 + A1;          % output registers tri state always enabled

          OE2 = /A1 + A1;

          XOE = /CS * RW;

% counter equations %
% INCR mode == S1 * !s0 %
% if (mode=INCR) RELO=ck   else RELO=clk   %

RELO =  Q3 * /Q2 * Q1 * /Q0 * CK + /Q1 *  CLK;

RES = RC;

P0t = Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0% counter increments if (mode==INCR)  %
 + Q3 * /Q2 * /Q1 *  Q0 *  B0 * /P0   % preload mode !S1 * S0, state mach=9 %
 + Q3 * /Q2 * /Q1 *  Q0 * /B0 *  P0   % toggle if (mode==LOAD & Bi != Pi)   %
 + Q3 *  Q2 * /Q1 *  Q0 *  B0 * /P0   % transparent mode S1 * S0, SM=13     %
 + Q3 *  Q2 * /Q1 *  Q0 * /B0 *  P0;  % toggle if (mode==WRDT & Bi != Pi)   %
P1t = Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  P0
    + Q3 * /Q2 * /Q1 *  Q0 *  B1 * /P1 + Q3 * /Q2 * /Q1 *  Q0 * /B1 *  P1
    + Q3 *  Q2 * /Q1 *  Q0 *  B1 * /P1 + Q3 *  Q2 * /Q1 *  Q0 * /B1 *  P1;
P2t = Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  P0 *  P1
    + Q3 * /Q2 * /Q1 *  Q0 *  B2 * /P2 + Q3 * /Q2 * /Q1 *  Q0 * /B2 *  P2
    + Q3 *  Q2 * /Q1 *  Q0 *  B2 * /P2 + Q3 *  Q2 * /Q1 *  Q0 * /B2 *  P2;
P3t = Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  P2 *  P1 *  P0
    + Q3 * /Q2 * /Q1 *  Q0 *  B3 * /P3 + Q3 * /Q2 * /Q1 *  Q0 * /B3 *  P3
    + Q3 *  Q2 * /Q1 *  Q0 *  B3 * /P3 + Q3 *  Q2 * /Q1 *  Q0 * /B3 *  P3;
P4t = Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  P3 * P2 * P1 * P0
```

```
    + Q3 * /Q2 * /Q1 *  Q0 *  B4 * /P4 + Q3 * /Q2 * /Q1 *  Q0 * /B4 *  P4
    + Q3 *  Q2 * /Q1 *  Q0 *  B4 * /P4 + Q3 *  Q2 * /Q1 *  Q0 * /B4 *  P4;
P5t = Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  P4 *  P3 *  P2 *  P1 *  P0
    + Q3 * /Q2 * /Q1 *  Q0 *  B5 * /P5 + Q3 * /Q2 * /Q1 *  Q0 * /B5 *  P5
    + Q3 *  Q2 * /Q1 *  Q0 *  B5 * /P5 + Q3 *  Q2 * /Q1 *  Q0 * /B5 *  P5;
P6t = P6 * /P6;

S0c = SE0;
S1c = SE1;

% state machine, controls the counter's operation  %
% states: Off_line=0, Idle=8, Preload=9, Incr=10, Write=11, %
%         Pass=12, pass_1=13, Clear=14, DS_High=15 %
Q0d =  Q3 * /Q2 * /Q1 * /Q0 * /S1 *  S0 +  Q3 * /Q2 * /Q1 *  Q0 * /S1 *  S0
    +  Q3 *  Q2 * /Q1 * /Q0 * /DS * /A1 * /RW * /CS +  Q3 *  Q2 * /Q1 *  Q0
    +  Q3 *  Q2 *  Q1 *  Q0 +  Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  EOE
    +  Q3 *  Q2 *  Q1 * /Q0 * /EOE;
Q1d =  Q3 *  Q2 * /Q1 *  Q0 *  DS +  Q3 *  Q2 *  Q1 *  Q0
    +  Q3 * /Q2 * /Q1 * /Q0 *  S1 * /S0 +  Q3 * /Q2 *  Q1 *  Q0 *  EOE
    +  Q3 *  Q2 *  Q1 * /Q0 +  /Q2 *  Q1 * /Q0 *  S1 * /S0
    + /Q3 *  Q2 *  Q1 *  Q0;
Q2d =  Q3 * /Q2 * /Q1 * /Q0 *  S1 *  S0 +  Q3 *  Q2 * /Q1 * /Q0 *  S1 *  S0
    +  Q3 *  Q2 * /Q1 * /Q0 * /DS +  Q3 *  Q2 * /Q1 *  Q0
    +  Q3 * /Q2 *  Q1 *  Q0 *  S1 *  S0 * /EOE +  Q3 * /Q2 *  Q1 *  Q0 *  EOE
    +  Q3 *  Q2 *  Q1 * /Q0;
Q3d = /CS * /RW * /DS * A1
    +  Q3 * /Q2 * /Q1 * /Q0          %Q=8%
    +  Q3 * /Q2 * /Q1 *  Q0          %Q=9%
    +  Q3 * /Q2 *  Q1 * /Q0          %Q=A%
    +  Q3 * /Q2 *  Q1 *  Q0          %Q=B%
    +  Q3 *  Q2 * /Q1 * /Q0          %Q=C%
    +  Q3 *  Q2 * /Q1 *  Q0          %Q=D%
    +  Q3 *  Q2 *  Q1 * /Q0 *  EOE   %Q=E * EOE=1%
    +  Q3 *  Q2 *  Q1 *  Q0          %Q=F%
    + /Q3 *  Q2 *  Q1 *  Q0;         %Q=7%

PLd =  Q3 * /Q2 * /Q1 * /Q0 * /S1 *  S0 +  Q3 * /Q2 * /Q1 * /Q0 *  S1 *  S0
    +  Q3 *  Q2 * /Q1 * /Q0 *  S1 *  S0 +  Q3 *  Q2 * /Q1 *  Q0
    +  Q3 *  Q2 *  Q1 *  Q0 +  Q3 * /Q2 *  Q1 *  Q0 *  S1 *  S0 * /EOE;

WOE =  Q3 + /Q3 *  Q2 *  Q1 *  Q0;

WRITEd = /(Q3 *  Q2 *  Q1 *  Q0 +   Q3 *  Q2 * /Q1 *  Q0 *  DS
       +   Q3 * /Q2 *  Q1 * /Q0 *  S1 * /S0 *  EOE
       +   Q3 * /Q2 *  Q1 *  Q0 *  S1 * /S0);    %CAMBIE EOE POR S1 * !S0%

CLR =  Q3 * /Q2 * /Q1 * Q0;

CARRYc = P5 * P4 * P3 * P2 * P1 * P0;

CKEd =  Q3 * /Q2 * /Q1 * /Q0 *  S1 * /S0 +  Q3 * /Q2 *  Q1 * /Q0 *  S1 * !S0
* /EOE + Q3 * Q2 *  Q1 * /Q0 *  S1 * !S0 + /Q3 *  Q2 *  Q1 *  Q0 *  S1 * /S0;

RCd =   Q3 *  Q2 *  Q1 * /Q0 + /Q3 *  Q2 *  Q1 *  Q0 + RESET + RF;

ZEROc =  A1 * /A1;
END$
```

HEADBV1.ADF-12/9/89

```
module counter
Gustavo I. Cancelo
Fermilab
10/9/89
 Added counter reset (RC) input.     Richard Kwarciany  17-NOV-89
 Removed HAB1 from Header pin equations,  Header outputs are now always
enabled.
  Reversed WE2, with WE1, RE2, with RE1, and OE2 with OE1, in input and
output
  register equations.   9-dec-89  rk.
title header latch & 12 bit event total counter

PART: EPB1400

INPUTS: DS@25, CS@12, A1@13, RW@36, RESET@9, CK@14, CKE@8, PL@5, CARRY@6,
CLK@7,
        RC@15

OUTPUTS: H0@2, H1@1, H2@40, H3@39, H4@38  %header%
         P6@24, P7@23, P8@22, P9@21, P10@20, P11@19, %counter%
         DAT0@26, DAT1@27, DAT2@28, DAT3@29, DAT4@32, DAT5@33, DAT6@34,
DAT7@35

NETWORK:
        DS=INP(DS)
        CS=INP(CS)
        A1=INP(A1)
        RW=INP(RW)
        RESET=INP(RESET)
        CKE=INP(CKE)
        PL=INP(PL)
        CARRY=INP(CARRY)
        CK=INP(CK)
        CLK=INP(CLK)
        RC=INP(RC)      %added 17-NOV-89  RK%

% data tranceiver bus %
DAT0,DAT1,DAT2,DAT3,DAT4,DAT5,DAT6,DAT7 = BUSX (IBUS,,XOE)

% header. input register #2 LBUSI %
XE1,XE2,HE0,HE1,HE2,HE3,HE4,XE3 = LBUSI (IBUS,DS,WE2)   %diferente nombre%

% counter's 6 highest bits. input register #1 LBUSI %
B0,B1,B2,B3,B4,B5,B6,B7 = LBUSI (IBUS,DS,WE1)

% output register #2 %
IBUS = LBUSO (X1,X2,H0,H1,H2,H3,H4,X3,,OE2,RE2)

% output register #1 %
IBUS = LBUSO (P6,P7,P8,P9,P10,P11,N1,N2,,OE1,RE1)

% header outputs%      %Removed HAB1 from OE   9-dec-89  RK%
H0,H0 = COCF (HE0,)
H1,H1 = COCF (HE1,)
H2,H2 = COCF (HE2,)
H3,H3 = COCF (HE3,)
```

```
H4,H4 = COCF (HE4,)

% junk. just to give a name to unused macrocells %
X1 = NOCF(X1c)
X2 = NOCF(X2c)
X3 = NOCF(X3c)

% counter outputs %            %changed RESET to CRESET in counter eqs.%
                               %17-nov-89  RK%
P6,P6 = TOTF(P6t,RELO,CRESET,,)
P7,P7 = TOTF(P7t,RELO,CRESET,,)
P8,P8 = TOTF(P8t,RELO,CRESET,,)
P9,P9 = TOTF(P9t,RELO,CRESET,,)
P10,P10 = TOTF(P10t,RELO,CRESET,,)
P11,P11 = TOTF(P11t,RELO,CRESET,,)

N1 = NOCF(N2c)
N2 = NOCF(N2c)

RELO = NOCF(RELOc)

CRESET = NOCF(CRESETc)

EQUATIONS:
          RE1 = !DS * !CS * RW * !A1;
          RE2 = !DS * !CS * RW *  A1;
% RE = RE1 + RE2 %

          WE1 = !RW * !CS * !A1;
          WE2 = !RW * !CS * A1;

          OE1 = !DS * !CS * RW * !A1;
          OE2 = !DS * !CS * RW * A1;

          XOE = !CS * RW;

           HAB1 = !CS;                 9-DEC-89 RK%

          CRESETc = RESET + RC;                %added 17-NOV-89  RK%

% counter equations %

RELOc = CLK * !CKE
      + CK  *  CKE;

P6t = !PL * CKE * CARRY
     + B0 * !P6 * PL
     + !B0 * P6 * PL;

P7t = P6 * !PL * CKE * CARRY
    + B1 * !P7 * PL
    + !B1 * P7 * PL;

P8t = P6 * P7 * !PL * CKE * CARRY
    + B2 * !P8 * PL
    + P8 * !B2 * PL;
```

```
P9t = P6 * P7 * P8 * !PL * CKE * CARRY
    + B3 * !P9 * PL
    + P9 * !B3 * PL;

P10t = P9 * P8 * P7 * P6 * !PL * CKE * CARRY
     + B4 * !P10 * PL
     + P10 * !B4 * PL;

P11t = P10 * P9 * P8 * P7 * P6 * !PL * CKE * CARRY
     + B5 * !P11 * PL
     + P11 * !B5 * PL;

X2c = PL * !PL;

X3c = PL * !PL;

N1c = PL * !PL;

N2c = PL * !PL;

X1c = PL * !PL;

END$_
```

INTF3.ADF-9/06/90

```
module oport_interface
Gustavo Cancelo
Fermilab
10/5/89                INTF3
-Changed RES equation.   Rick Kwarciany  16-NOV-89
-Changed clock on START, INHIB, MODE, RES, pins, from RELO (CLK2C), to CS;
-Changed polarity of WE2.  Kwar  16-DEC-89
-Changes from INTF2.... added RPERMIN, and RWAIT to register PERMIN, and
       WAIT signals.  Kwar  4-20-90
-Changed interrupt, and interrupt mask equations.  kwar  9-4-90
-Changed INCOMP interrupt to PERMIN interrupt.  kwar 9-4-90
-Added clear to IRQ line.  kwar 9-6-90

title interface the Oport controller to the CPU system

PART:EPB1400

INPUTS: CS@12,RW@36,DS@25,A0@13,CLK@14,CLK2@7,RESET@9
        PERMIN@8,PRMOUT@40,STPACK@18,ERROR@17,SM3@21,SM2@22,SM1@23,SM0@24
        WAIT@5

OUTPUTS: DB0@26,DB1@27,DB2@28,DB3@29,DB4@32,DB5@33,DB6@34,DB7@35
         MODE2@2,MODE0@4,MODE1@3,RES@1,STPMK@39,PERMINMK@38,ERRMK@37
         IRQ@16,RPERMIN@19,RWAIT@6,IRQS@20

NETWORK:

A0=INP(A0)
CLK=INP(CLK)
CLK2=INP(CLK2)
CS=INP(CS)
RW=INP(RW)
DS=INP(DS)
RESET=INP(RESET)

SM3=INP(SM3) %SM3,SM2,SM1,SM0: OPORT's state machine status%
SM2=INP(SM2)
SM1=INP(SM1)
SM0=INP(SM0)
PERMIN=INP(PERMIN)
PRMOUT=INP(PRMOUT)
STPACK=INP(STPACK)
ERROR=INP(ERROR)
WAIT=INP(WAIT)

MODE0,MODE0=RORF(MODE0d,RELO,RESET,,)
MODE1,MODE1=RORF(MODE1d,RELO,RESET,,)
MODE2,MODE2=RORF(MODE2d,RELO,RESET,,)
RES,RES=RORF(RESd,CLK2,,,)

IRQ,IRQ=RORF(IRQd,CLK,,,)
STPMK,STPMK=RORF(STPMKc,RELO,RESET,,)
PERMINMK,PERMINMK=RORF(PERMINMKc,RELO,RESET,,)
ERRMK,ERRMK=RORF(ERRMKc,RELO,RESET,,)
IRQS,IRQS=COCF(IRQSd,)
```

```
RPERMIN,RPERMIN=RORF(PERMINd,CLK,RESET,,)
RWAIT,RWAIT=RORF(WAITd,CLK2,RESET,,)

%high add/data byte tranceiver w/tri_state enable%
DB0,DB1,DB2,DB3,DB4,DB5,DB6,DB7 = BUSX (IBUS,,XOE)

%high address byte input register w/tri_state enable 2%
MODE01,MODE11,MODE21,RES1,OR10,STPMK1,PERMINMK1,ERRMK1 = LBUSI (IBUS,,WE2)

%high data byte output register 1 %
IBUS = LBUSO (SM0,SM1,SM2,SM3,IRQS,RPERMIN,STPACK,ERROR,,OLE1,RE1)

%high data byte output register 2 %
IBUS = LBUSO (MODE0,MODE1,MODE2,RES,PRMOUT,STPMK,PERMINMK,ERRMK,,OLE2,RE2)


EQUATIONS:

          RELO = !DS * !CS * !RW * A0 * CLK2;

          RE1 = !DS * !CS * RW * !A0;
          RE2 = !DS * !CS * RW *  A0;
          % RE = RE1 + RE2 %

          WE2 = (!RW * !CS * A0);     %Changed polarity 16-dec-89 rk%

           OLE1 = CLK;
           OLE2 = CLK2;

          XOE = !CS * RW;


MODE0d = MODE01;
MODE1d = MODE11;
MODE2d = MODE21;

!RESd = RESET + RES1;
!IRQd = (STPACK * STPMK)
       + (!PERMIN * PERMINMK)
       + (ERROR * ERRMK)
       + (!IRQ * RES * !(!DS * !CS * RW * !A0));

STPMKc = STPMK1;
PERMINMKc = PERMINMK1;
ERRMKc = ERRMK1;
IRQSd = IRQ;

WAITd = WAIT;
PERMINd = PERMIN;

END$
```

LC_HIGH.ABL-2/5/90

```
MODULE LOCAL_COUNTER_HIGH
TITLE 'LOCAL_COUNTER_HIGH  PC2 VERSION 5-FEB-90 0900 hrs'
LC_HIGH DEVICE 'P22V10';

CLK     PIN     1;
LC0     PIN     2;
LC1     PIN     3;

!CIN    PIN     22;
!TC     PIN     23;

!OE     PIN     13;

I0      PIN     4;
I1      PIN     5;
I2      PIN     6;
I3      PIN     7;
I4      PIN     8;
I5      PIN     9;

Q0      PIN     16;
Q1      PIN     17;
Q2      PIN     18;
Q3      PIN     19;
Q4      PIN     20;
Q5      PIN     21;

DATA    =       [Q5..Q0];
INPUT   =       [I5..I0];
MODE    =       [LC1,LC0];
HOLD    =       [0,.X.];
WRITE   =       [1,0];
COUNT   =       [1,1];
MATCH   =       0;

EQUATIONS


ENABLE DATA     =       OE;

WHEN (MODE == COUNT) & (CIN == 1) THEN DATA:=DATA-1;

WHEN (MODE == WRITE) THEN DATA:=INPUT;

WHEN (MODE == HOLD) # ((MODE == COUNT) & (CIN == 0)) THEN DATA:=DATA;

TC      =       (DATA == MATCH);

ENABLE CIN=0;

END LOCAL_COUNTER_HIGH
```

LC_LOW.ABL-2/5/90

```
MODULE LOCAL_COUNTER_LOW
TITLE 'LOCAL_COUNTER_LOW   PC2 VERSION 5-FEB-90  0900hrs'
LC_LOW DEVICE 'P22V10';

CLK     PIN     1;
LC0     PIN     2;
LC1     PIN     3;

!COUT   PIN     22;
!TC     PIN     23;

!OE     PIN     13;

I0      PIN     4;
I1      PIN     5;
I2      PIN     6;
I3      PIN     7;
I4      PIN     8;
I5      PIN     9;

Q0      PIN     16;
Q1      PIN     17;
Q2      PIN     18;
Q3      PIN     19;
Q4      PIN     20;
Q5      PIN     21;

DATA    =       [Q5..Q0];
INPUT   =       [I5..I0];
MODE    =       [LC1,LC0];
HOLD    =       [0,.X.];
WRITE   =       [1,0];
COUNT   =       [1,1];
MATCH   =       0;

EQUATIONS


ENABLE DATA     =       OE;

WHEN (MODE == COUNT) THEN DATA:=DATA-1;

WHEN (MODE == WRITE) THEN DATA:=INPUT;

WHEN (MODE == HOLD) THEN DATA:=DATA;

TC      =       (DATA == MATCH);

COUT    =       (MODE == COUNT) & (DATA == 0);

END LOCAL_COUNTER_LOW
```

OPOV7.02.ASM-6/4/90

```
OPORT State Machine SAM format file
VDAS controller
V7.02  Monday 4-JUN-90        1530hrs
Change from V3: Keep oe true during reset to prevent 485 drivers from
                oscillating on cable.
Change from V4: Change Start, and Step inputs to Mode0, and Mode1.
                Four possible modes are now:
                M1      M0      Mode
                0       0       Hold
                0       1       Start
                1       0       Force_Start
                1       1       CPU
Change from V5:Reverse polarity of Strobe output, and remove Strobe, and
                OE from RESET state.  Now a negative going ~25 ns pulse
                will be generated by a one shot on the AUX card from the
                rising edge of the the VDAS controller Strobe output.
                Change made 21-Mar-90  by Richard Kwarciany.
Change from V6:Inverted PERMIN/PERMOUT signals as per E771's request, and
                corrected error in macros for vdas_oe line.  24-Mar-90  RK
Change from V6.01:Change polarity of WAIT input to match VDAS's /WAIT signal.
                10-APR-90, RK.
Change from V6.02: Change INHIB line to MODE2.  Encode the three mode lines
                to allow for two new modes.  The new modes are; EMPO, and PO.
                (Event With Manual Permit Out, and Permit Out).  The Eight
                possible modes are now:
                Mode            M2      M1      M0
                Hold            0       0       0
                Event           0       0       1
                Force_Event     0       1       0
                CPU (Step)      0       1       1
                Inhibit         0       1       0
                EMPO            1       0       1
                PO              1       1       0
                Reserved        1       1       1
Change from V7.00: Added two new states; Active2, and Wait3. Active2 is the
                same as Active, except that EOE is not tested. This state will
                only be used on the first data cycle, where EOE is not valid.
                Wait3 is the same as Wait, except that it returns to Active2
                instead of Active.
Change from V7.01: Made changes to fix bugs in Step mode operation.

PART: EPS448

INPUTS: wait@2, eoe@3, def@4, cef@5, permin@9, mode2@10, mode1@11, mode0@12

OUTPUTS: Q3@25, Q2@24, Q1@23, Q0@22,
        lck@13, strobe@20, /cfifor@17, /dfifor@18,
        /loe@14, error@26, stpack@27, /prmout@28,
        /lclr@16, vdas_oe@1

MACROS:  % outNo   state    outputs %
          RESET = "0000    0011100100"      %LCLR*%
           OUT0  = "0000    0011000101"      %LCLR*,LOE*,VDAS_OE%
           OUT1  = "0001    0001000111"      %CFR*,LOE*,VDAS_OE%
           OUT2  = "0010    0011000101"      %LCLR*,LOE*,VDAS_OE%
           OUT3  = "0011    0001000111"      %CFR*,LOE*,VDAS_OE%
```

```
        OUT4  = "0100    1010000111"      %LCK,DFR*,LOE*,VDAS_OE%
        OUT5  = "0101    0011001011"      %STPACK,LOE*,VDAS_OE,PERMOUT*%
        OUT6  = "0110    0001000111"      %CFR*,LOE*,VDAS_OE%
        OUT7  = "0111    0011110100"      %ERROR,LCLR*%
        OUT8  = "1000    0011100110"      %no outputs%
        OUT9A = "1001    0011000101"      %LCLR*,LOE*,VDAS_OE%
        OUT9B = "1001    0010000001"      %DFR*,PERMOUT*,LOE*,VDAS_OE%
        OUT9C = "1001    0011000001"      %PERMOUT*,LCLR*,LOE*,VDAS_OE%
        OUT9D = "1001    0010100100"      %DFR*,LCLR*%
        OUT10 = "1010    1011000111"      %LCK,LOE*,VDAS_OE%
        OUT11 = "1011    1011000111"      %LCK,LOE*,VDAS_OE%
        OUT12 = "1100    0110000111"      %STROBE,DFR*,LOE*,VDAS_OE%
        OUT13 = "1101    0011001011"      %STPACK,LOE*,VDAS_OE,PERMOUT*%
        OUT14 = "1110    1011000111"      %LCK,LOE*,VDAS_OE%
        OUT15 = "1111    1011000111"      %LCK,LOE*,VDAS_OE%

EQUATIONS:
        MODE_0  =       /mode2  *  /mode1  *  /mode0;
        %hold%
        MODE_1  =       /mode2  *  /mode1  *   mode0;
        %Event%
        MODE_2  =       /mode2  *   mode1  *  /mode0;
        %Force_Event%
        MODE_3  =       /mode2  *   mode1  *   mode0;
        %CPU%
        MODE_4  =        mode2  *  /mode1  *  /mode0;
        %Inhibit%
        MODE_5  =        mode2  *  /mode1  *   mode0;
        %EMPO%
        MODE_6  =        mode2  *   mode1  *  /mode0;
        %PO%
        MODE_7  =        mode2  *   mode1  *   mode0;
        %Reserved%

PROGRAM:
%*************************************************************************%
%    State = IDLE, CK=L, strobe=H, cfifor=H, dfifor=H, oe=H, error=0,     %
%    stpack=L,  prmout=L                                                  %
%*************************************************************************%
        [RESET] CONTINUE;

RES:    IF (MODE_1 + MODE_3 + MODE_5)          %Event + CPU + EMPO%
        THEN [RESET] CONTINUE;
        ELSEIF MODE_2 THEN [OUT0] JUMP IDLES;
        %Force_Event%
        ELSEIF MODE_4 THEN [OUT8] JUMP INHIBS;
        %inhibit%
        ELSE [RESET] JUMP RES;

PINWAIT1:IF /permin + MODE_2                   %permin* + Force_Event%
        THEN [OUT0] JUMP IDLES;
        ELSEIF MODE_4 THEN [OUT8] JUMP INHIBS;
        %inhibit%
        ELSE [RESET] JUMP PINWAIT1;
```

```
%*************************************************************************%
%    State = IDLE, CK=L, strobe=H, cfifor=H, dfifor=H, oe=H, error=0,      %
%    stpack=L,  prmout=L                                                  %
%    IF inhib THEN State=INHIB (8)                                        %
%    ELSEIF ((Event + Force_Event + CPU + EMPO) * cef) State=SETUP (2)    %
%    ELSE state=IDLES (0)                                                 %
%*************************************************************************%
IDLES:  IF MODE_4                                  %Inhibit%
        THEN [OUT8] JUMP INHIBS;
        ELSEIF (MODE_1 + MODE_2 + MODE_3 + MODE_5) * cef
        THEN [OUT2] JUMP SETUPS;
        ELSE [OUT0] JUMP IDLES;

%*************************************************************************%
%    State = INHIB, CK=L, strobe=H, cfifor=H, dfifor=H, oe=H, error=0,    %
%    stpack=L,  prmout=L                                                  %
%*************************************************************************%
INHIBS: IF MODE_4 THEN [OUT8] JUMP INHIBS;   %Inhibit%
        ELSE [OUT0] JUMP RES;

%*************************************************************************%
%    State = SETUP, CK=L, strobe=H, cfifor=H, dfifor=H, oe=H, error=0,    %
%    stpack=L,  prmout=L                                                  %
%    IF /wait State=ERROR (7), error go to H                              %
%    ELSE State=CONTS (6), cfifor=L, oe=L                                 %
%*************************************************************************%
SETUPS: IF (/wait) THEN [OUT7] JUMP ERRORS;
        ELSEIF MODE_3 THEN [OUT3] JUMP STEPS;
        %Step%
        ELSE [OUT6] JUMP CONTS;

%*************************************************************************%
%    State = CONTS, CK=L, strobe=H, cfifor=L, dfifor=H, oe=L, error=0,    %
%    stpack=L,  prmout=L                                                  %
%    IF cef State=ACTIVE (14), ck go to H, strobe=L, cfifor=H             %
%*************************************************************************%
CONTS: IF MODE_0 THEN [OUT4] JUMP HOLD1S;    %hold%
       ELSE [OUT14] JUMP ACTIVE2S;

%*************************************************************************%
%    State = ACTIVES, CK=H, strobe=L, cfifor=H, dfifor=H, oe=L, error=0,  %
%    stpack=L,  prmout=L                                                  %
%    IF /WAIT THEN WAITS                                                  %
%    IF eoe State=EOES (9), ck go to H, oe=H,                             %
%    ELSEIF def THEN State=DATA (12), ck=L, strobe=H, dfifor=H,           %
%    ELSE State = ACTIVES                                                 %
%*************************************************************************%
ACTIVES: IF (/wait) THEN [OUT15] JUMP WAITS;
         ELSEIF (eoe) THEN [OUT9A] JUMP CHKMODE;
         ELSEIF (def) THEN [OUT12] JUMP DATAS;
         ELSE [OUT14] JUMP ACTIVES;

ACTIVE2S:IF (/wait) THEN [OUT15] JUMP WAIT3S;
         ELSEIF (def) THEN [OUT12] JUMP DATAS;
         ELSE [OUT14] JUMP ACTIVE2S;
```

```
%*************************************************************************%
%    State = DATAS, CK=L, strobe=H, cfifor=H, dfifor=H, oe=L, error=0,     %
%    stpack=L,  prmout=L                                                  %
%    IF /start THEN HOLD1S.                                               %
%    ELSE State=ACTIVE (14), ck go to H, strobe=L, cfifor=H               %
%*************************************************************************%
DATAS: IF MODE_0 THEN [OUT4] JUMP HOLD1S;    %hold%
       ELSE [OUT14] JUMP ACTIVES;

%*************************************************************************%
%    State = WAITS, CK=H, strobe=L, cfifor=H, dfifor=H, oe=L, error=0,     %
%    stpack=L,  prmout=L                                                  %
%    IF cef State=ACTIVE (14), same outputs                               %
%*************************************************************************%
WAITS: IF (wait) THEN [OUT14] JUMP ACTIVES;  %/wait*%
       ELSE [OUT15] JUMP WAITS;

WAIT3S:IF (wait) THEN [OUT14] JUMP ACTIVE2S; %/wait*%
       ELSE [OUT15] JUMP WAIT3S;

%*************************************************************************%
%    State = HOLD1S, CK=H, strobe=L, cfifor=H, dfifor=H, oe=L, error=0,    %
%    stpack=L,  prmout=L                                                  %
%    IF inhib State=INHIBS (8), ck go to L, strobe=H, oe=H                %
%    ELSEIF (step * !start) State=IDLE, ck goto L, strobe=H, oe=H         %
%    ELSEIF (start) State=ACTIVE, same outputs                            %
%*************************************************************************%
HOLD1S: IF MODE_4 THEN [OUT8] JUMP INHIBS;
        ELSEIF MODE_3 THEN [OUT0] JUMP IDLES;
        %Step%
        ELSEIF MODE_1 + MODE_2 + MODE_5
        %Event + Force_Event + EMPO%
        THEN [OUT12] JUMP DATAS;
        ELSE [OUT4] JUMP HOLD1S;

%*************************************************************************%
%    State = STEPS, CK=L, strobe=H, cfifor=L, dfifor=H, oe=L, error=0,     %
%    stpack=L,  prmout=L                                                  %
%    IF /wait State=WAIT2S (1), same outputs                              %
%    ELSEIF (!cef) State=STPACK1 cfifor=H, oe=H, stpack=H                 %
%    ELSE State=IDLE, stpack goto H, cfifor=H, oe=H                       %
%*************************************************************************%
STEPS: IF (/wait) THEN [OUT1] JUMP WAIT2S;
       ELSEIF (!cef) THEN [OUT5] JUMP STPACK1S;
       ELSE [OUT11] JUMP SDATAS;

%*************************************************************************%
%    State = SDATAS, CK=H, strobe=L, cfifor=H, dfifor=H, oe=L, error=0,    %
%    stpack=L,  prmout=L                                                  %
%    IF (!step) State=HOLD2S, same outputs                                %
%    ELSEIF (cef) State=STEPS (3), ck go to L, strobe=H, cfifor=L         %
%*************************************************************************%
SDATAS: IF MODE_3 THEN [OUT3] JUMP STEPS;    %step%
        ELSE [OUT10] JUMP HOLD2S;
```

```
%*****************************************************************************%
%    State = HOLD2S, CK=H, strobe=L, cfifor=H, dfifor=H, oe=L, error=0,       %
%    stpack=L,  prmout=L                                                      %
%    IF inhib State=INHIBS (8), ck go to L, strobe=H, oe=H                    %
%    ELSEIF (/step * /start) State=IDLES, ck goto L, strobe=H, oe=H           %
%    ELSEIF (start * step) State=SDATAS, same outputs                         %
%*****************************************************************************%
HOLD2S: IF MODE_4 THEN [OUT8] JUMP INHIBS;   %Inhibit%
        ELSEIF MODE_1 + MODE_2 + MODE_5
        %Event + Force_Event + EMPO%
        THEN [OUT0] JUMP IDLES;
        ELSEIF MODE_3 THEN [OUT11] JUMP SDATAS;
        %step%
        ELSE [OUT10] JUMP HOLD2S;

%*****************************************************************************%
%    State = WAIT2S, CK=L, strobe=H, cfifor=L, dfifor=H, oe=L, error=0,       %
%    stpack=L,  prmout=L                                                      %
%    IF cef State=STEP (3), same outputs                                      %
%*****************************************************************************%
WAIT2S: IF (wait) THEN [OUT3] JUMP STEPS;
        ELSE [OUT1] JUMP WAIT2S;

%*****************************************************************************%
%     EOE goto IDLE                                                           %
%*****************************************************************************%
EOES:   [OUT9C] CONTINUE;

PINWAIT2:IF (/permin) THEN [OUT0] JUMP IDLES;
        %/permin*%
        ELSEIF MODE_0 THEN [RESET] JUMP RES;  %hold%
        ELSEIF MODE_4 THEN [OUT8] JUMP INHIBS;
        %inhibit%
        ELSE [RESET] JUMP PINWAIT2;

%*****************************************************************************%
%     STPACK1S goto STPACK2                                                   %
%*****************************************************************************%
STPACK1S:  [OUT13] JUMP STPACK2S;

%*****************************************************************************%
%     STPACK2S goto PINWAIT2                                                  %
%*****************************************************************************%
STPACK2S:   [OUT0] JUMP PINWAIT2;
```

```
%*************************************************************************%
%     State = ERRORS, CK=L, strobe=H, cfifor=H, dfifor=H, oe=H, error=1,   %
%     stpack=L,  prmout=L                                                 %
%     stay in ERRORS until hardware Reset                                 %
%*************************************************************************%
ERRORS: [OUT7] JUMP ERRORS;

CHKMODE:IF MODE_6 + MODE_1 + MODE_2          %PO + Event + Force_Event%
        THEN [OUT9B] JUMP EOES;
        ELSEIF MODE_0 THEN [OUT9D] JUMP CLEANUP;
        %hold%
        ELSEIF MODE_4 THEN [OUT9D] JUMP INHIBS;
                %Inhibit%
        ELSE [OUT9A] JUMP CHKMODE;

CLEANUP:[RESET] JUMP RES;

END$
```

STIMER.ABL-10/13/89

```
MODULE SHORT_TIMER
TITLE 'SHORT_TIMER        VERSION 21-AUG-89 1430 hrs'
STIMER DEVICE 'P22V10';

CLK     PIN     1;
FRBH    PIN     3;
FRRB    PIN     4;
FDRB    PIN     5;
STEN    PIN     8;
TIMER   PIN     9;
FRWT    PIN     10;
FCLERR  PIN     13;

T0      PIN     14;
T1      PIN     15;
T2      PIN     16;
T3      PIN     17;
T4      PIN     18;
!STO    PIN     20;
R0      PIN     21;
R1      PIN     22;
!FRESET         PIN     23;

COUNT   =       [T4..T0];
RCOUNT  =       [R1..R0];
RINIT   =       [1,0];
CINIT   =       [1,1,1,1,1];

EQUATIONS

WHEN (FCLERR == 1) # (FRBH == 1) # (FRRB == 0) # (FDRB == 1) THEN
RCOUNT:=RINIT;
                                                ELSE RCOUNT:=RCOUNT-1;

WHEN (TIMER == 0) # (FCLERR == 1) # (STEN == 0) # (FRWT == 1) THEN
COUNT:=CINIT;
                                                ELSE COUNT:=COUNT-1;

STO     :=      ((COUNT == 0) # (STO == 1)) & (FCLERR == 0);

FRESET  :=      ((RCOUNT == 0) # (FRESET ==1)) & (FCLERR == 0);

END SHORT_TIMER
```

STROBE.ABL-9/28/89

```
module ds_generator
title 'Fastbus Readout Controller, data strobe generator
Gustavo Cancelo, Fermilab, 4/3/89'

"Added DS to LDS and UDS 19-SEP-89 RK
"Changed LDS, and UDS from pins 16 and 17 to pins 15 and 16.  27-SEP-89 RK

        strobe   device    'P22V10';

AD0,AD1,SIZ0,SIZ1,RW,DS       pin     8,7,6,5,4,3;
LDS,UDS                       pin     15,16;
WLDS,WUDS                     pin     22,23;
WUUDS,WUMDS,WLMDS,WLLDS       pin     21,20,19,18;
RDS                           pin     14;

L,H,X,Z = 0,1,.X.,.Z.;

TRANF_TYPE = [SIZ1,SIZ0];
LONG = ^b00;
TRI_BY = ^b11;
WORD = ^b10;
BYTE = ^b01;

AD1_0 = [AD1,AD0];

equations
         !UDS = !AD0 * !DS;
         !LDS =  AD0 * !DS # !SIZ0 * !DS # SIZ1 * !DS;

         !WUDS = !AD0 & !RW;
         !WLDS =  (AD0 # !SIZ0 # SIZ1) & !RW;

         !RDS = RW;


         !WUUDS = !AD0 & !AD1 & !RW;

         !WUMDS = (!AD1 & !SIZ0
                #  AD0 & !AD1
                # !AD1 & SIZ1) & !RW;

         !WLMDS = (!AD0 & AD1
               # !AD1 & !SIZ0 & !SIZ1
               # !AD1 & SIZ0 & SIZ1
               #  AD0 & !AD1 & !SIZ0) & !RW;

         !WLLDS = (AD0 & SIZ0 & SIZ1
               # !SIZ0 & !SIZ1
               #  AD0 & AD1
               #  AD1 & SIZ1) & !RW;

test_vectors

([TRANF_TYPE,AD1,AD0, RW] -> [WUUDS,WUMDS,WLMDS,WLLDS])
 [   LONG,      0, 0,  L ] -> [  L,     L,     L,     L  ];
 [   TRI_BY,    0, 0,  L ] -> [  L,     L,     L,     H  ];
```

```
  [   WORD,      0, 0,  L ] -> [  L,    L,    H,    H  ];
  [   BYTE,      0, 0,  L ] -> [  L,    H,    H,    H  ];
  [   LONG,      0, 1,  L ] -> [  H,    L,    L,    L  ];
  [   TRI_BY,    0, 1,  L ] -> [  H,    L,    L,    L  ];
  [   WORD,      0, 1,  L ] -> [  H,    L,    L,    H  ];
  [   BYTE,      0, 1,  L ] -> [  H,    L,    H,    H  ];

  [   LONG,      1, 1,  H ] -> [  H,    H,    H,    H  ];
  [   TRI_BY,    1, 1,  H ] -> [  H,    H,    H,    H  ];
  [   WORD,      1, 1,  H ] -> [  H,    H,    H,    H  ];
  [   BYTE,      1, 1,  H ] -> [  H,    H,    H,    H  ];
  [   LONG,      1, 0,  H ] -> [  H,    H,    H,    H  ];
  [   TRI_BY,    1, 0,  H ] -> [  H,    H,    H,    H  ];
  [   WORD,      1, 0,  H ] -> [  H,    H,    H,    H  ];
  [   BYTE,      1, 0,  H ] -> [  H,    H,    H,    H  ];


test_vectors
([TRANF_TYPE, DS,AD1,AD0, RW] -> [WUDS,WLDS,UDS,LDS])
  [  LONG,      0, 0,  0,  L ] -> [ L,   L,   L,  L];
  [  WORD,      0, 0,  1,  H ] -> [ H,   H,   H,  L];
  [  BYTE,      0, 1,  0,  H ] -> [ H,   H,   L,  H];
  [  BYTE,      0, 1,  1,  H ] -> [ H,   H,   H,  L];
  [  BYTE,      0, 0,  0,  H ] -> [ H,   H,   L,  H];
  [  BYTE,      0, 0,  1,  H ] -> [ H,   H,   H,  L];


END_
```

Appendix D - FSCC Documentation

Fermilab Drawing Numbers

FNAL #	Title	Description
0882-MB-199070	Mounting Bracket	FASTBUS Module Front Panel Mount
0880.000-ED-215714	FSCC/VDAS Interface	E771 FSCC/VDAS interface schematic
0880.000-ED-269065	FSCC/VDAS Interface	E791 FSCC/VDAS interface schematic
0880.000-AC-269129	FSCC Layer 1	Trace Layer 1
0880.000-AC-269130	FSCC Layer 2	-2.0V & -5.2V Layer 2
0880.000-AC-269131	FSCC Layer 3	Ground Layer 3
0880.000-AC-269132	FSCC Layer 4	+5.0V Layer 4
0880.000-AC-269133	FSCC Layer 5	Trace Layer 5
0880.000-AC-269134	FSCC Top Silk	Top Silkscreen Photo
0880.000-AC-269135	FSCC Bot Silk	Bottom Silkscreen Photo
0880.000-AC-269136	FSCC Solder Mask	Top & Bot. Solder Mask
0880.000-AC-269137	FSCC Front Panel	Front Panel Silkscreen Photo
0880.000-MD-269138	FSCC Assembly	Assembly drawing
0880.000-MD-269139	FSCC Front Panel	Front Panel Mechanical Drawing
0880.000-MD-269140	FSCC Mechanical	Board Dimensions & Pads
0880.000-MD-269141	FSCC OPORT Test Board	OPORT Test Board Schematic
0880.000-MD-269142	FSCC FPP Test Board	Front Panel Port Test Board Schematic
	FSCC FPP Top Layer	Front Panel Port Test Board Top Layer
	FSCC FPP Bottom Layer	Front Panel Port Test Board Bottom Layer
0880.000-MD-269143	FSCC Schematic	11 Page Schematic and Block Diagram

Fermilab Documents

FASTBUS Smart Crate Controller - PC3, Design Specification
Fermilab Computing Division
Mark Bernett - Online and Data Acquisition Software Groups
Mark Bowden, Rick Kwarciany, John Urish - Data Acquisition Electronics Group
Fermilab Physics Department
Gustavo Cancelo

Diagnostics for the FASTBUS Smart Crate Controller - PN417
Fermilab Computing Division
Mark Bernett, Dave Slimmer - Online and Data Acquisition Software Groups
Fermilab Computing Division
Rick Kwarciany, John Urish - Data Acquisition Electronics Group

Release Notes for SCG68K V2.3
PN 376
David M. Berg, Bryan MacKinnon - Fermilab Computing Division
Online Systems Software Group

SCG68K User's Guide and Reference
PN369
Peter Heinicke, David Berg, Bryan MacKinnon, Tom Nicinski, Gene Oleynik -
Fermilab Computing Division, Online Systems and Data Acquisition Software Groups

Serial Port Driver for the PAN-DA pSOS Environment
PN379.2

Bryan MacKinnon - Fermilab Computing Division, Data Acquisition Software Group

FSCC Parts List, Front Panel Port Test Board Parts List, OPORT Parts List and FSCC/VDAS interface Parts List

The parts lists are implemented in a Microsoft EXCEL spreadsheet program to allow easy updating and sorting. They are available from John Urish at Fermilab as a printed copy or on a Macintosh floppy disk.

Non-Fermilab Documents

pSOS-68K, pROBE-68K, pRISM-68K

Software Components Group, Inc.
4655 Old Ironsides Drive
Santa Clara, CA 95054

IEEE Standard FASTBUS
IEEE 960

Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, New York - 10017

68020 32-Bit Microprocessor Manual

Motorola Literature Distribution
P.O. Box 20912
Phoenix, AZ 85036

FASTBUS Standard Routines
DOE/ER 0325

National Technical Information Service, U.S. Dept. of Commerce
Springfield, Virginia 22161

FSCC Software

FSCC System Software

Software Components Group, Inc.
4655 Old Ironsides Drive
Santa Clara, CA 95054
(Refer to "SCG68K User's Guide and Reference" or contact David Berg at Fermilab)

FSCC Diagnostic Software

The diagnostic software is supplied by Fermilab and described in "Diagnostics for the FASTBUS Smart Crate Controller". The source code or compiled code may be obtained by contacting Dave Slimmer at Fermilab.

FSCC Microcode

PAL/PLD Equations for FSCC

The PAL/PLD sources are listed in Appendix C of this document. The source listings and JEDEC files may be obtained on a Macintosh or IBM format floppy disk from John Urish at Fermilab.

FPORT Equations for FSCC

The EPS448 sources are listed in Appendix C of this document. The source listings and JEDEC files may be obtained on a Macintosh or IBM format floppy disk from John Urish at Fermilab.

PAL/PLD Equations for OPORT Test Board

The source listings and JEDEC files may be obtained on a Macintosh or IBM format floppy disk from John Urish at Fermilab.

Appendix E - FSCC Front Panel

FASTBUS Slave: Yellow LED

FASTBUS Master: Green LED

Front Panel Output (20 pin IDC):

Pin#	RS422 Funciton
1	Front Panel 0 +
2	Front Panel 0 -
3	Front Panel 1 +
4	Front Panel 1 -
5	Front Panel 2 +
6	Front Panel 2 -
7	Front Panel 3 +
8	Front Panel 3 -
9-20	Reserved

Trigger Input (20 pin IDC):

Pin#	RS422 Funciton
1-10	Reserved
11	Trigger Strobe +
12	Trigger Strobe -
13	Trigger ID0 +
14	Trigger ID0 -
15	Trigger ID1 +
16	Trigger ID1 -
17	Trigger ID2 +
18	Trigger ID2 -
19	Trigger ID3 +
20	Trigger ID3 -

Reset Pushbutton: Hard processor reset.

Remote Reset:: Hard processor reset. This is an active low TTL input. Shorting the connetor or applying a TTL low will cause a reset. This input may be Daisy-Chained.

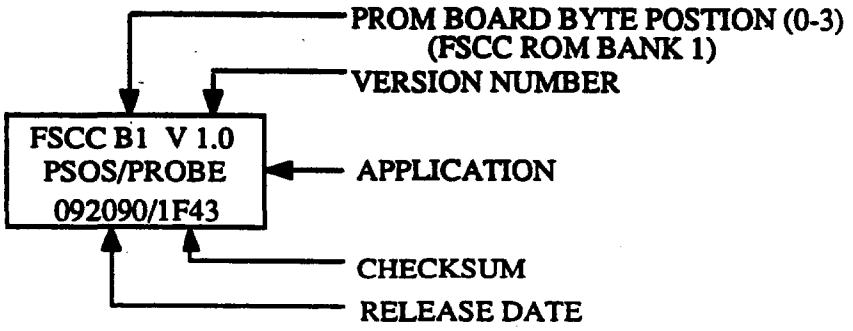
Perm_In/Perm_Out: Serial "daisy-chain" signals for FSCC auxiliary port bussing, LEMO connectors.

2 Serial Ports: RS232 signal levels (4 pin LEMO connectors) one for Host connection and one for Terminal connection.
Terminal port
Host port (Null Modem)

Ethernet Port: Cheapernet signal levels, Isolated BNC Connector.

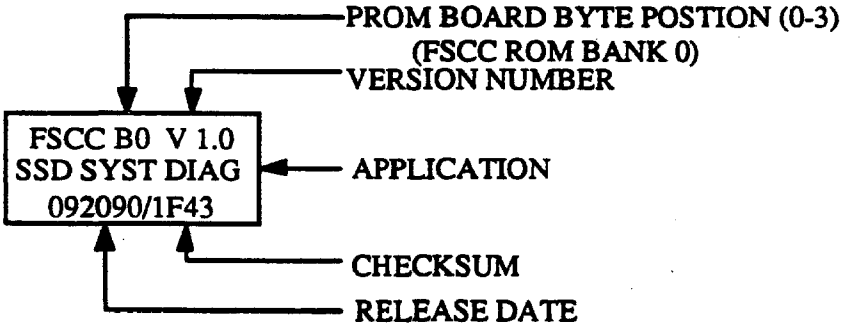
Appendix F - FSCC EPROMs

ROM BANK 1: OPERATING SYSTEM AND FSCC DIAGNOSTICS

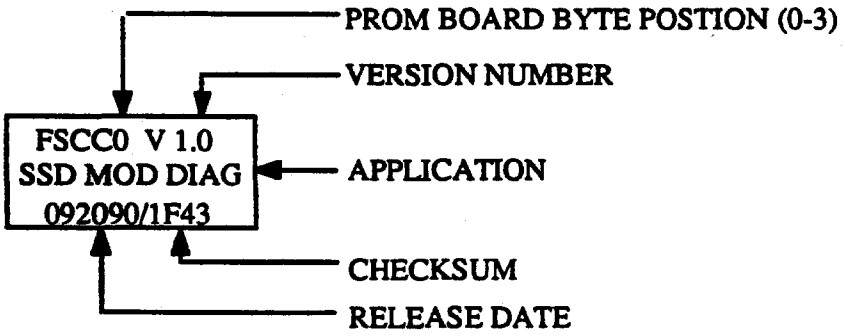


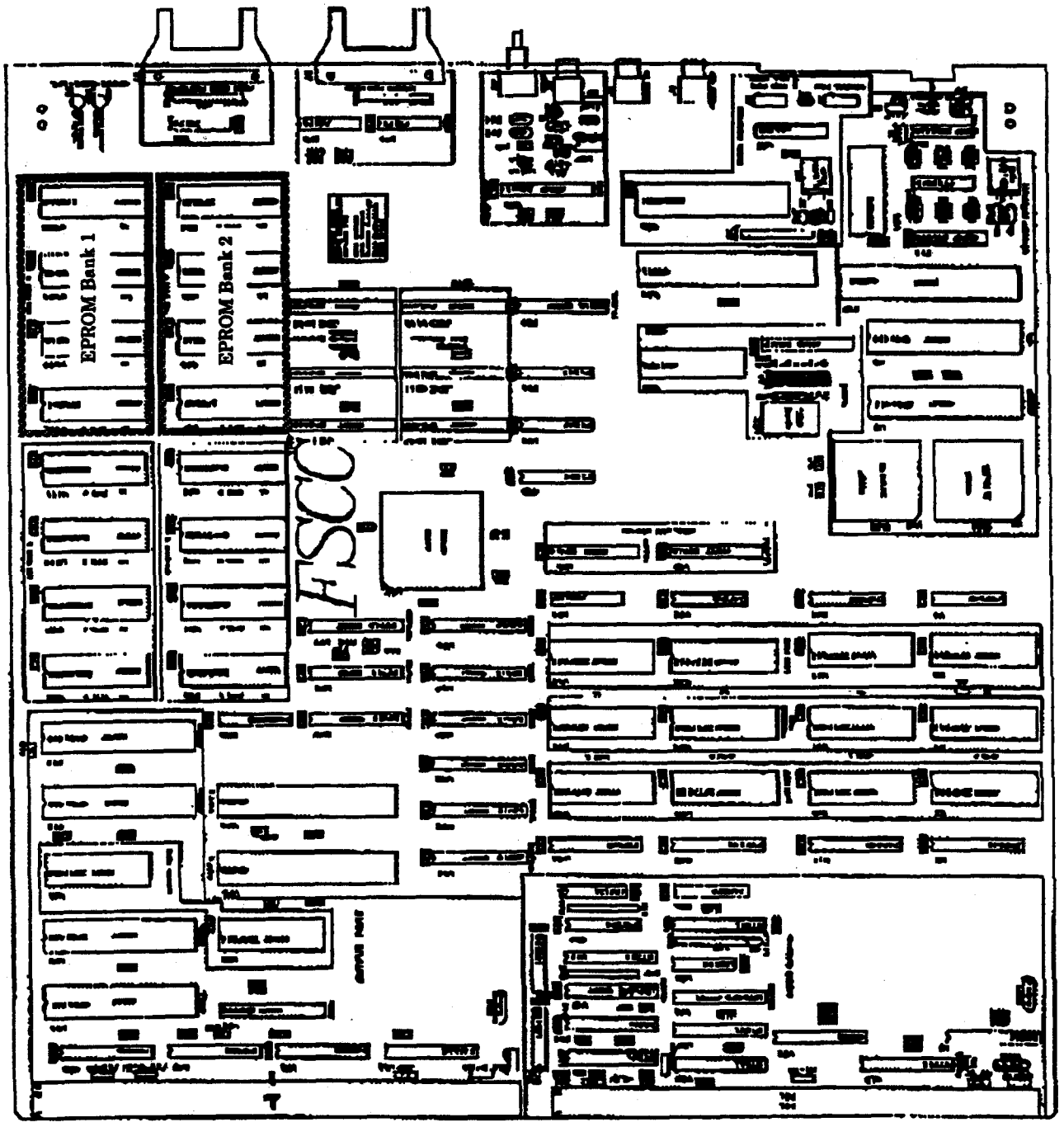
ROM BANK 2: SYSTEM OR INDIVIDUAL MODULE DIAGNOSTICS

OPTION 1; System Diagnostic Tests



OPTION 2; Individual Module Tests





FASTBUS SMART CRATE CONTROLLER

VERSION PC3

EPROM Assembly Drawing