Fermi National Accelerator Laboratory

FERMILAB-Pub-87/145

A VME to FASTBUS Interface Using a Finite State Machine Coprocessor*

Dr. Leif Gustafsson
Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

September 18, 1987

*Submitted to *Computer Standards and Interfaces*

A VME to FASTBUS Interface using a Finite State Machine Coprocessor. *

Dr. Leif Gustafsson
Fermi National Accelerator Laboratory **
P.O.BOX 500
Batavia, Illinois 60510

September 18, 1987
16:52:57

* This work has in part been done at CERN Geneva,Switzerland.

** Operated by Universities Research Association for the Department of Energy.

Abstract

A VME to FASTBUS master interface is described. This interface executes FASTBUS instructions set up by a master on the VME segment. The instructions are decoded using finite state machines that run the FASTBUS master protocol and drives a FASTBUS cable segment. All internal data and control paths are implemented using TTL level circuitry. Data is internaly buffered by a 64 Kb memory, and a transfer speed of 100 ns per 32 bit word is possible to achieve.

Keywords: Cable Segment, Coprocessor, FASTBUS, Finite State Machine, Interface, PILS, Programmable Logic Device, Protocol, VME.

1. Introduction

In order to test a FASTBUS master interface that can easily be driven from different bus systems as a coprocessor, we have developed two VME modules that house a VME slave interface, interrupter, registers, DMA, memories and drivers needed to run a FASTBUS cable segment. The main blocks are shown in figure 1 and 2.

A high level FASTBUS operation can be handled directly as an instruction with parameters. This operation is loaded from VME into dedicated registers [1]. On a start signal asserted in the VME control register, a set of finite state machines (FSM) decode the instruction. These FSMs are implemented using programmable logic devices (PLD), for example the 16L8 and 20L8 PAL circuits.

The FSM handling the data transfers is running the handshaking protocol defined in the FASTBUS specification [2]. A pipeline register,between an internal 64 Kb memory and the FASTBUS segment, makes possible a block transfer speed of 100 ns per 32 bit word. On the completion of a FASTBUS operation, either polling or a vectored interrupt can be used to signal the VME master.

2. The VME interface

All registers in the interface are seen as 32 bit words, separated in two sets by a buffer that is disabled when a start bit (GO) is asserted in the control register, see table 1. A VME master can always read and write to the control register. The registers that are locked from the VME side on a start signal comprises FASTBUS status, DMA control, address and word count, parameters, instruction and arbitration level. The internal formats of the VME control and FASTBUS status registers are seen in table 2.

The VME port can transfer either 16 or 32 bit words, handled by a finite state machine as seen in figure 3. On a valid internal address and VME data strobe (IDS), the port is timed by two signals, READY1 and READY2. These signals show that data has been clocked into and out of a port register. The VME master using the interface is acknowledged through the signal DTACK~ [3]. A new cycle on the port can only start if the valid address and data strobe are momentarily disasserted by the VME master.

2.1. The DMA

The DMA comprises two Am2940 circuits giving a 16 bit address register and a 16 bit word counter, which makes it possible to use up to 256 Kb of memory as an internal data buffer. An external address register has been added in order to get an address pipeline. The address and data lines (AD0-AD31) to the FASTBUS segment are also pipelined through a register, making it possible to cover some of the transfer time during a block transfer. The DMA is presently using a 64 Kb buffer memory.

2.2. The parameter registers

The parameter registers are implemented as a fast register file using four AS870 circuits. In the present version of the interface, only two registers are used. These registers are for the FASTBUS primary and secondary addresses. The remaining eight registers can be used if a VME master or a FASTBUS instruction pipeline is implemented in the future.

2.3. The interrupt logic

The interrupter part of the interrupt logic is shown in figure 4. If the POLL bit in the VME control register is asserted or the FASTBUS master coprocessor is not in the status interrupt state, activating a master release signal (MR1~), the interrupt acknowledge out signal (IACKOUT~) is passed on. If the POLL bit is disasserted when MR1~ is active and no interrupt acknowledge in signal (IACKIN~) is coming from the interrupt handler, an interrupt level (IRQX~) is asserted on VME. If a match is found with the interrupt vector address sent by the interrupt handler, the valid interrupt register is opened by the enable vector signal (ENVECT~). As the address strobe (AS~) is disactivated by the interrupt handler a service request end signal (SREND~) is issued. This signal takes the FASTBUS master coprocessor to its idle state. If no match is found, IACKOUT~ is passed on. When the present interrupt cycle has ended, a new cycle is directly started until a match is found [4]. The present implementation of the interrupt logic uses two PLDs as shown in figure 5.

2.4. The Instruction and Arbitration level registers

The instruction and arbitration level registers are located on the second board. The function of the bits in the instruction register is seen in table 3. In the arbitration level register (FASTBUS CSR#8) the priority is set for the interface when participating in a FASTBUS arbitration cycle.

3. The FASTBUS interface

When the start signal is asserted in the control register the interface is running autonomously using seven state machines, see figure 6. The state machines can be categorized as arbitration supervisor, arbitration machine, primary address supervisor, primary address machine, secondary address and data cycle supervisor, data cycle machine and data transfer machine.

3.1. The arbitration supervisor

The arbitration supervisor is started by asserting the GO bit in the VME control register, see figure 7. The INAR bit in the FASTBUS instruction register indicates whether an arbitration is required. If arbitration is not required and we are already the current FASTBUS master, control passes directly onto the primary address supervisor by asserting the signal START1. If arbitration is required and there is no internal GK signal (IGK) set, a SETAR signal is asserted which starts the arbitration machine and an arbitration timer. When FASTBUS mastership is granted, the primary address supervisor is started.

Should an error occur during the arbitration process, the error is encoded in a status encoder PLD and clocked into the FASTBUS status register by the state asserting the master release 1 signal (MR1). Before returning to the idle state, either polling or a vectored interrupt is used to signal the VME master that the operation has finished. When the vectored interrupt mechanism is used, the return to the idle state is governed by the SREND~ signal generated by the interrupter logic. If no vectored interrupts are required, the POLL bit should be asserted in the VME control register.

3.2. The arbitration machine

The arbitration is implemented following the assured access protocol as described in the FASTBUS specification [2]. The state machine version is shown in figure 8. When the SETAR signal is asserted and the arbitration inhibit signal (AI) from the ancillary logic is not asserted, an internal arbitration request (IAR) is generated. Arbitration is started when an arbitration grant signal (AG) is asserted by the FASTBUS arbitration and timing controller (ATC). During the arbitration process a continuous comparison of arbitration levels proceeds until AG is disasserted. If the master containing the arbitration machine is the winner of the arbitration cycle the arbitration logic asserts ISMINE and the arbitration machine asserts IGK. The GK signal is only disasserted if a release GK signal (RELGK) in the instruction register and the MR1 signal are set. Figure 9 details the actual hardware implementation. It is important that the vector comparison is done within the time limits set up by the FASTBUS specification. If TTL PLDs are used, they must have a delay not exceeding 15 ns [5].

3.3. The primary address supervisor

Before starting the primary address machine, it is necessary to verify if certain criteria are fulfilled. Figure 10 shows the different criteria used by the primary address supervisor. If the primary address cycle signal (PRIM) is asserted in the instruction register, a check is done for the address strobe (AS) and address acknowledge (AK) signals. This condition is called an AS/AK lock. If no lock exists, PRIAC is generated which enables the primary address register and starts the primary address machine. Two status conditions may be signalled by the primary address machine, AK timeout (AKTO) and a slave status (SS) greater then zero.

If the AS/AK lock is established without errors, the supervisor machine for the secondary address is started. If the user has specified that no

primary address cycle should occur, the operation will only proceed if an AS/AK lock already exists.

3.4. The primary address machine

The primary address machine is started on the assertion of a delayed PRIAC signal (DPRIAC) from the primary address supervisor, see figure 11. The primary address from the primary address register is clocked (CLKR1) in to the pipeline register, described above. The MS code bits from the instruction register are enabled directly out onto the FASTBUS segment. After a delay time compensating for the pipeline register internal delay, an internal address strobe signal (IAS) is generated and an AK timer is started. On AK timeout, a status signal ERR5 is asserted. When AK is returned by the FASTBUS slave the SS lines are checked. If SS is equal to zero, the primary address transfer machine retains AS until a release AS (RELAS) and a master ready signal (MR1) are asserted.

3.5. The secondary address and data cycle supervisor

The supervisor machinery starts on a signal START2 from the primary address supervisor, see figure 12. The supervisor will abort if either the data strobe (DS) or data acknowledge (DK) signals are already asserted. If the user has requested a secondary address cycle (SEC), the content of the secondary address register is enabled and the data cycle machine is started. A secondary address cycle is a data cycle with a special master status code (MS=2). The secondary address cycle is terminated if there is a DK timeout (DKTO) or SS is greater then zero.

Following a successfully completed secondary address operation and if the user has requested data cycles (DATC), the data transfer machinery is started (START3) and the FASTBUS operation proceeds as in the secondary address cycle. The secondary address and data cycle supervisor then "loops" waiting for the data cycle to complete. Upon completion, it returns to the idle state which allows the primary address supervisor to return in turn to it's idle state. Finally, the arbitration supervisor becomes idle.

3.6. The data cycle machinery

Compared with other bus protocols, the FASTBUS protocol is unique in its treatment of block transfers. A data word can be transferred on each edge of DS. In order to be able to use our FSM design in other bus systems, we have decided to split the data protocol into two state machines, the data cycle machine and data transfer machine, where the latter handles the special FASTBUS protocol [2].

The data cycle machine is shown in figure 13. The machine can run in both reading and writing mode using the FASTBUS read signal RD as steering signal for input and output functions. In the FASTBUS read mode, the machine works as follows.

When the START3 signal is asserted by the supervisory machine, the data cycle machine ensures that the transfer ready fastbus signal (TRFB) and a ready signal (READY4) from the memory control logic are present. The actual data transfer on FASTBUS is managed by the data transfer machine.

This machine is started by a start fastbus signal (STFB). When handshaking takes place between the two machines, the pipeline register is clocked (CLKR2). Depending on the type of transfer, random or block, a signal called word count zero (WCZ) is generated by the memory transfer logic indicating whether the machine should fetch another word or should end the transfer. If another transfer is required, the memory control logic calculates the next address by clocking the address register (CLKADR1) and the DMA logic signal CDMA. When terminating the transfer, only a CLKADR1 signal is generated.

In the FASTBUS write mode, the data cycle machine operates in a similar way as when in the read mode. Write mode, however, reverses the order of the different test operations and output signals.

3.7. The data transfer machine

As mentioned above, the actual data transfer on FASTBUS is managed by the data transfer machine, shown in figure 14. This machine does not support parity check, but it would be easy to implement. When the STFB signal is asserted, DS is toggled by the TOGDS signal. The TOGDS signal is also used to start a DK timer. Using handshaking, the transfer machine waits for DK to come to the same level as DS. If DK is not back before the DK timeout signal is asserted, an error is indicated. When DK is back without DK timeout, a check is done on the SS response.

4. The cable segment

To get the FASTBUS signals into a cable segment, a differential driver and receiver pair are used as outlined in figure 15 [2]. The cable segment can be connected to a FASTBUS segment interconnect module (SI) or a cable to crate segment converter module in order to reach FASTBUS slave modules [6], [7].

5. Performance

Tests of the interface have been done in a VME based system called VALET-plus [8]. By using a programming language called PILS in this system, programs have been written implementing most of the FASTBUS standard routines, enabling us to run random and block transfers to different slave modules [1]. The PILS code used in a block transfer write is shown in figure 16.

In all the tests, a 10 m cable segment was connected to a cable to crate segment converter [7]. It was possible in all tests to confirm that an internal response time of 100 ns was possible to cover through the pipeline register used in the interface, figure 17. In order to test arbitration, another master was used, alternating the mastership of the FASTBUS segment.

6. Conclusions

The interface has been shown to fulfil the handshaking part of the FASTBUS master protocol, making it possible to collect data from FASTBUS slave modules and process it in a VME system.

There are some features that have been left out in this first version, i.e wait (WT), parity (PA,PE), reset bus (RB), bus halt (BH), and service request (SR). These functions may be implemented by slight changes and extensions.

As long as no conclusive decision has been made on how the syncronous mode of the FASTBUS data transfer protocol should be handled on the systems level, we have not atempted any implementation of it. But by including a speed register and a new data tranfer machinery, it should be no problem to support this mode.

A future extension of the present internal configuration can also include a pointer unit, pointing to different portions of a memory holding FASTBUS instructions and parameters. This would make the programming model of the interface very simple.

By changing the FASTBUS part of the design to ECL circuitry, it should be possible to at least double the transfer speed into the internal buffer memory.

If a FASTBUS slave part is developed on a separate VME module, the present FASTBUS port can be used with it. This would require changing the PAL expressions for the signals stearing the segment drivers and receivers.

References

[1] U.S. NIM Committee, DOE/ER-0325. FASTBUS Standard Routines, March 1987.

[2] FASTBUS specification, ANSI/IEEE Std 960-1986.

[3] VME specification, Rev. C. IEEE P1014/D1.0, Feb. 1985.

[4] Gustafsson,L. and Gallno,P. VME bus protocol chipset using programmable logic. Proc. of VMEbus conference, CERN, 1985.

[5] Fremont,G. and Sanches,E. A general purpose PAL based master and slave FASTBUS coupler. CERN/EP-F6803, March 1985.

[6] Downing,R. and Haney,M. The FASTBUS Segment Interconnect. IEEE Trans. on Nucl. Sci. NS-29, No. 1, 94 (1982).

[7] Swoboda C. and Moore G. Cable to Crate Segment Converter. FNAL internal report.

[8] Berners-Lee,T. et al. The VALET-PLUS, a VMEbus Microcomputer for Physics Applications. Fifth conference on Real Time Computer Applications in Nuclear, Particle and Plasma Physics- San Francisco, May 1987.

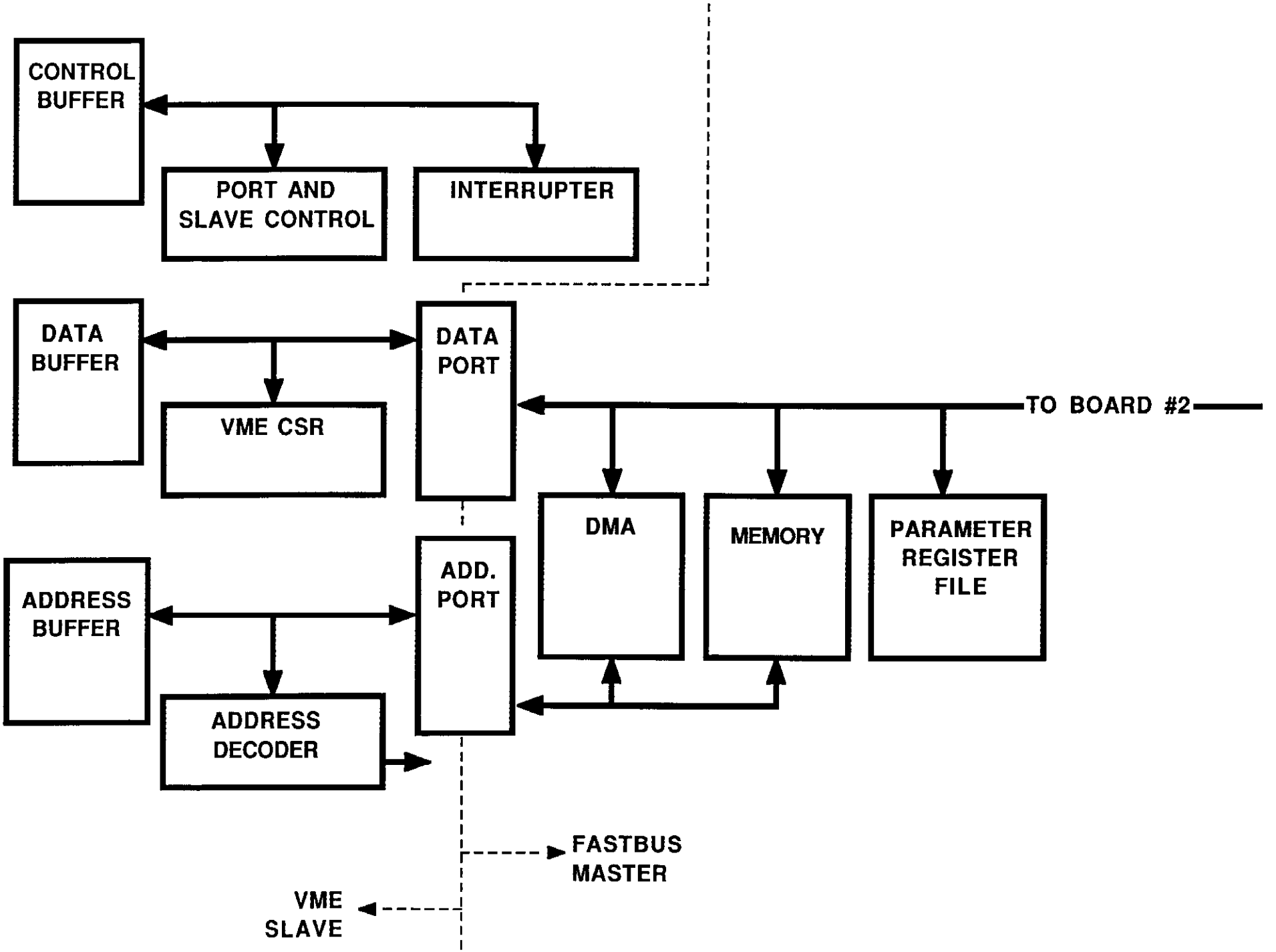
CONTROL BUFFER
PORT AND SLAVE CONTROL
INTERRUPTER
DATA BUFFER
VME CSR
DATA PORT
TO BOARD #2
DMA
MEMORY
PARAMETER REGISTER FILE
ADDRESS BUFFER
ADD. PORT
ADDRESS DECODER
FASTBUS MASTER
VME SLAVE

Figure 1

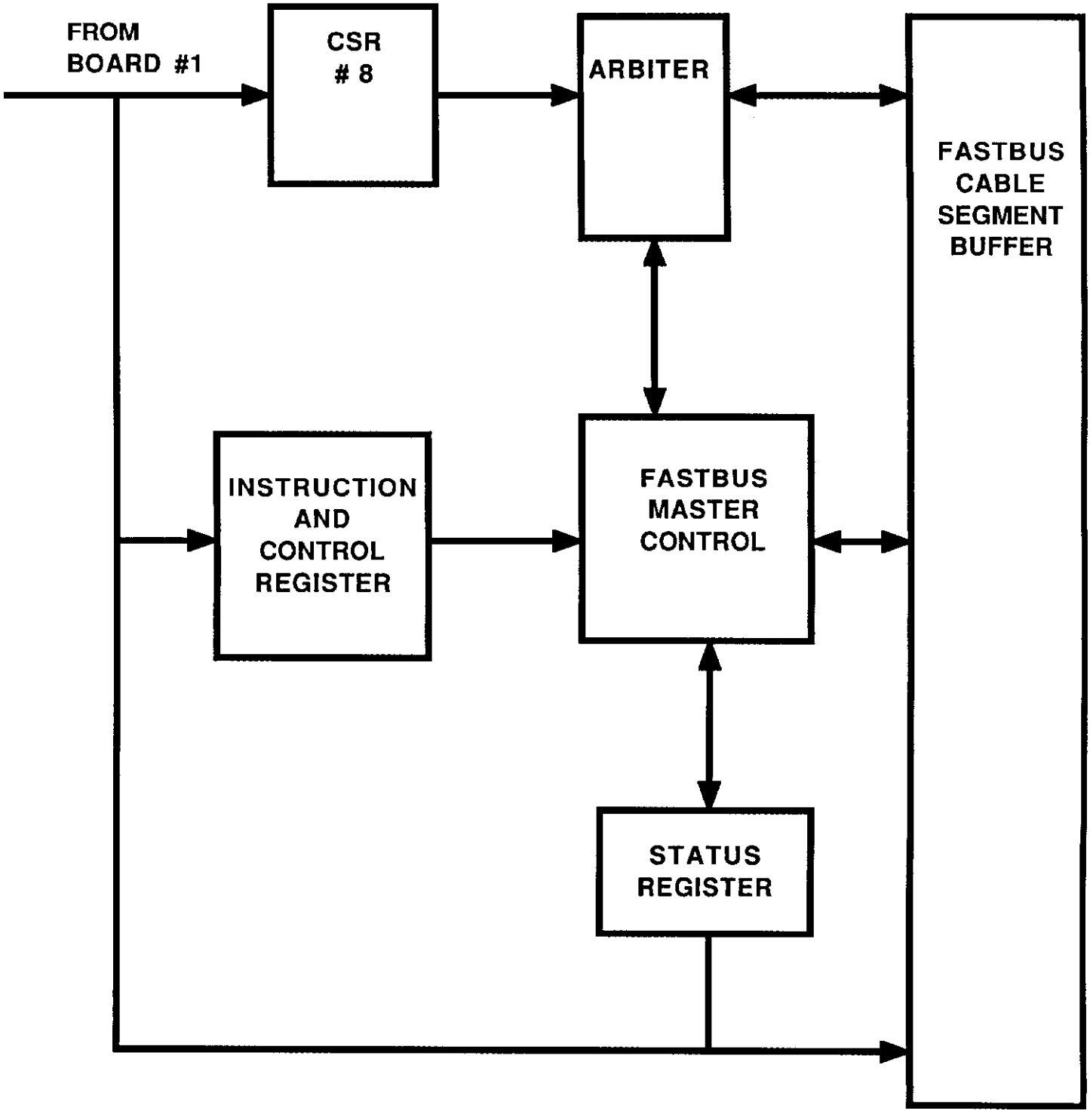
FROM
BOARD #1
CSR
8
ARBITER
FASTBUS
CABLE
SEGMENT
BUFFER
INSTRUCTION
AND
CONTROL
REGISTER
FASTBUS
MASTER
CONTROL
STATUS
REGISTER

Figure 2

BASE +	$00	VME CONTROL REGISTER
	$02	NO ERROR INTERRUPT REGISTER
	$03	ERROR INTERRUPT REGISTER
	$04	FASTBUS STATUS REGISTER
	$05	VME ADDRESS PIPELINE REGISTER
	$06	DMA CONTROL REGISTER
	$07	DMA ADDRESS COUNTER
	$08	DMA WORD COUNTER
	$09	DMA REINITIALIZATION
	$0A	EXTERNAL DMA CLOCKING
	.	
	.	
	.	
	$10	PRIMARY ADDRESS REGISTER
	$11	SECONDARY ADDRESS REGISTER
	.	
	.	
	.	
	$21	FASTBUS INSTRUCTION REGISTER
	.	
	.	
	.	
	$25	CSR #8
$180000 +	.	MEMORY
	.	
	.	
	.	
	.	
	.	
	.	

Table 1

D5	ERROR	FASTBUS ERROR BIT
D4	MREADY	MASTER READY
D3		
D2	RESET~	GENERAL RESET
D1	POLL	POLLING BIT
D0	GO	START BIT

(a)

$81	ERR1•/INAR	NO INAR AND NO GK
$82	ERR1•INAR	INAR AND GK
$90	ART0	ARBITRATION TIME OUT
$91	ERR3	NO PRIM AND NO AS/AK LOCK
$92	ERR4	AS/AK LOCK ALREADY EXIST
$A1	ERR5•/SS2•/SS1•SS0	SS=1)
$A2	ERR5•/SS2•SS1•/SS0	SS=2)
$A3	ERR5•/SS2•SS1•SS0	SS=3)
$A4	ERR5•SS2•/SS1•/SS0	SS=4 > PRIMARY ADDRESS CYCLE
$A5	ERR5•SS2•/SS1•SS0	SS=5)
$A6	ERR5•SS2•SS1•/SS0	SS=6)
$A7	ERR5•SS2•SS1•SS0	SS=7)
$B0	ERR5•AKTO	AK TIMEOUT
$C0	ERR6	DS OR DK (BEFORE DATA CYCLE)
$D1	ERR7•/SS2•/SS1•SS0	SS=1)
$D2	ERR7•/SS2•SS1•/SS0	SS=2)
$D3	ERR7•/SS2•SS1•SS0	SS=3)
$D4	ERR7•SS2•/SS1•SS0	SS=4 > DATA CYCLE
$D5	ERR7•SS2•/SS1•SS0	SS=5)
$D6	ERR7•SS2•SS1•/SS0	SS=6)
$D7	ERR7•SS2•SS1•SS0	SS=7)
$E0	ERR7•DKTO	DK TIMEOUT

(b)

Table 2

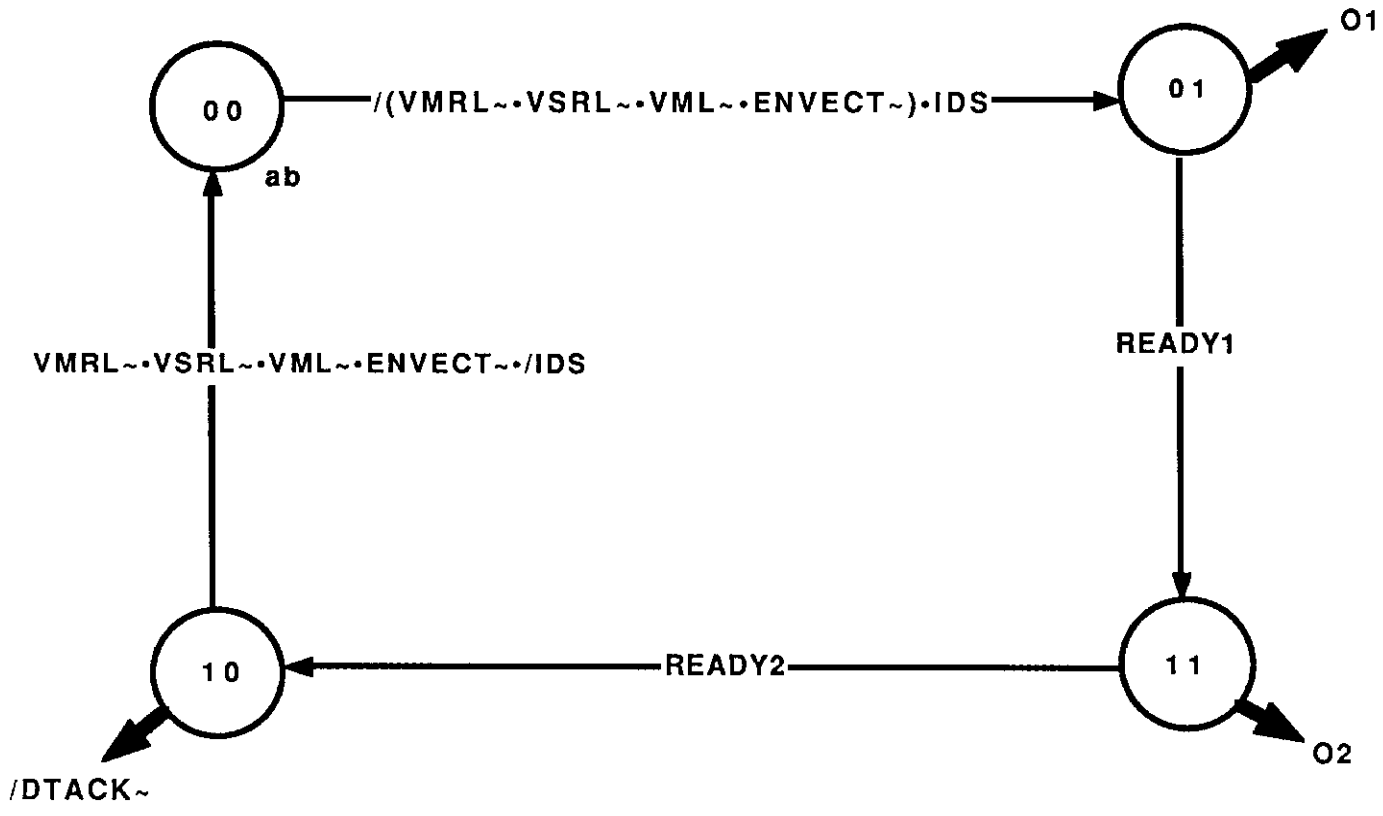
00
ab
/(VMRL~•VSRL~•VML~•ENVECT~)•IDS
01
O1
READY1
11
O2
READY2
10
/DTACK~
VMRL~•VSRL~•VML~•ENVECT~•/IDS

Figure 3

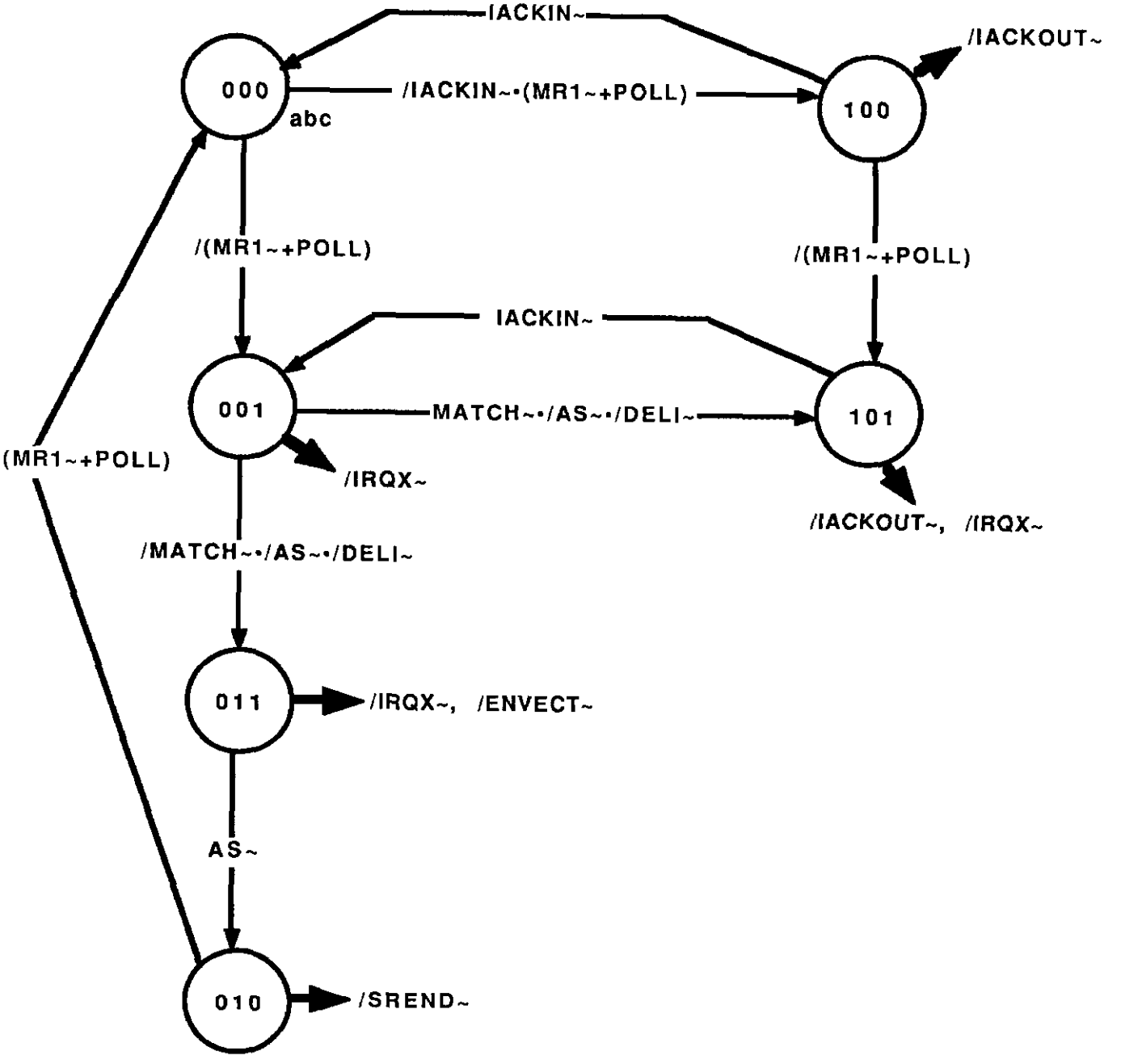
IACKIN~
/IACKOUT~
000
/IACKIN~•(MR1~+POLL)
100
abc
/(MR1~+POLL)
/(MR1~+POLL)
IACKIN~
001
MATCH~•/AS~•/DELI~
101
(MR1~+POLL)
/IRQX~
/IACKOUT~, /IRQX~
/MATCH~•/AS~•/DELI~
011
/IRQX~, /ENVECT~
AS~
010
/SREND~

Figure 4

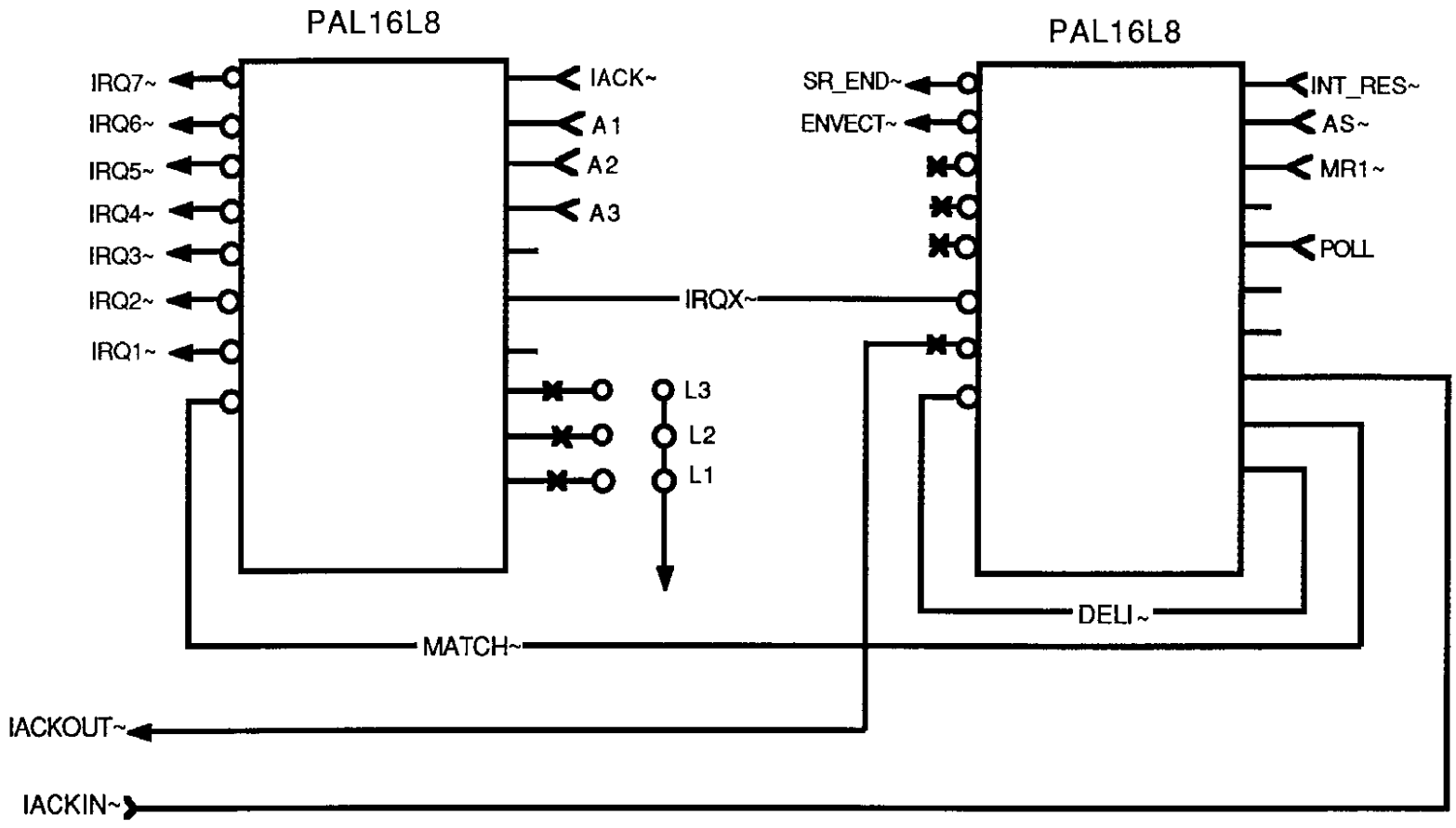
PAL16L8
PAL16L8
IRQ7~
IRQ6~
IRQ5~
IRQ4~
IRQ3~
IRQ2~
IRQ1~
IACK~
A1
A2
A3
IRQX~
L3
L2
L1
SR_END~
ENVECT~
INT_RES~
AS~
MR1~
POLL
DELI~
MATCH~
IACKOUT~
IACKIN~

Figure 5

AD23		
AD22		
AD21		
AD20		
AD19		
AD18	BLOCK	BLOCK TRANSFER
AD17	IEG	INTERNAL EG
AD16	IRD	INTERNAL RD
AD15		
AD14	MSDA2)	
AD13	MSDA1 >	DATA CYCLE MS CODE
AD12	MSDA0)	
AD11		
AD10	MSPA2)	
AD9	MSPA1 >	PRIMARY ADDRESS CYCLE MS CODE
AD8	MSPA0)	
AD7		
AD6	RELAS	RELEASE AS
AD5	RELGK	RELEASE GK
AD4	DATC	DATA CYCLE
AD3	SEC	SECONDARY ADDRESS CYCLE
AD2	PRIM	PRIMARY ADDRESS CYCLE
AD1	INAR	INTERNAL ARBITRATION REQUEST
AD0	STCL	STATUS CLOCK

Table 3

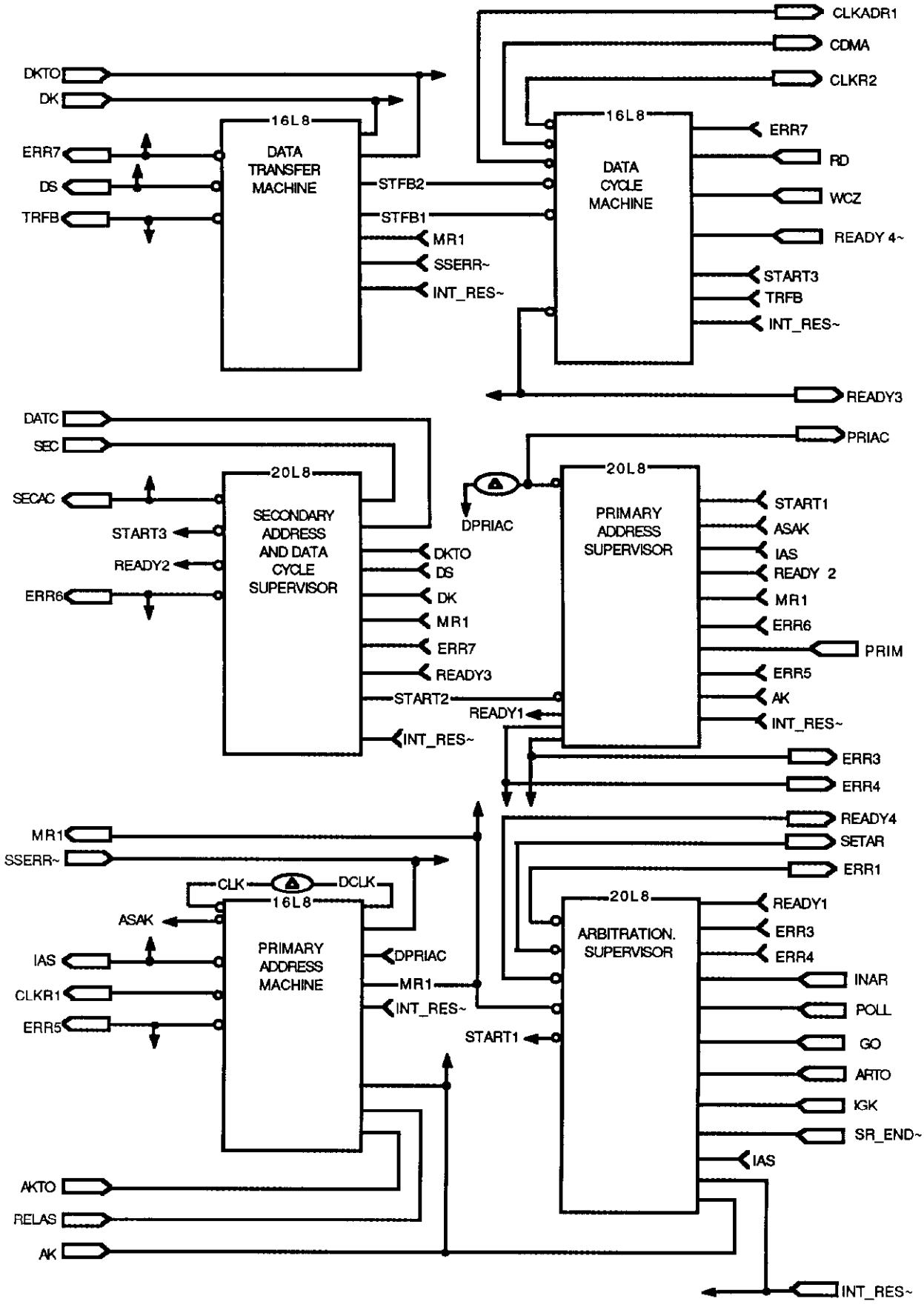
CLKADR1
CDMA
CLKR2
DKTO
DK
16L8
DATA TRANSFER MACHINE
ERR7
DS
TRFB
STFB2
STFB1
MR1
SSERR~
INT_RES~
16L8
DATA CYCLE MACHINE
ERR7
RD
WCZ
READY 4~
START3
TRFB
INT_RES~
READY3
DATC
SEC
SECAC
START3
READY2
ERR6
20L8
SECONDARY ADDRESS AND DATA CYCLE SUPERVISOR
DKTO
DS
DK
MR1
ERR7
READY3
START2
INT_RES~
PRIAC
DPRIAC
20L8
PRIMARY ADDRESS SUPERVISOR
READY1
START1
ASAK
IAS
READY 2
MR1
ERR6
PRIM
ERR5
AK
INT_RES~
ERR3
ERR4
MR1
SSERR~
CLK
DCLK
16L8
PRIMARY ADDRESS MACHINE
ASAK
IAS
CLKR1
ERR5
DPRIAC
MR1
INT_RES~
START1
READY4
SETAR
ERR1
20L8
ARBITRATION. SUPERVISOR
READY1
ERR3
ERR4
INAR
POLL
GO
ARTO
IGK
SR_END~
IAS
AKTO
RELAS
AK
INT_RES~

Figure 6

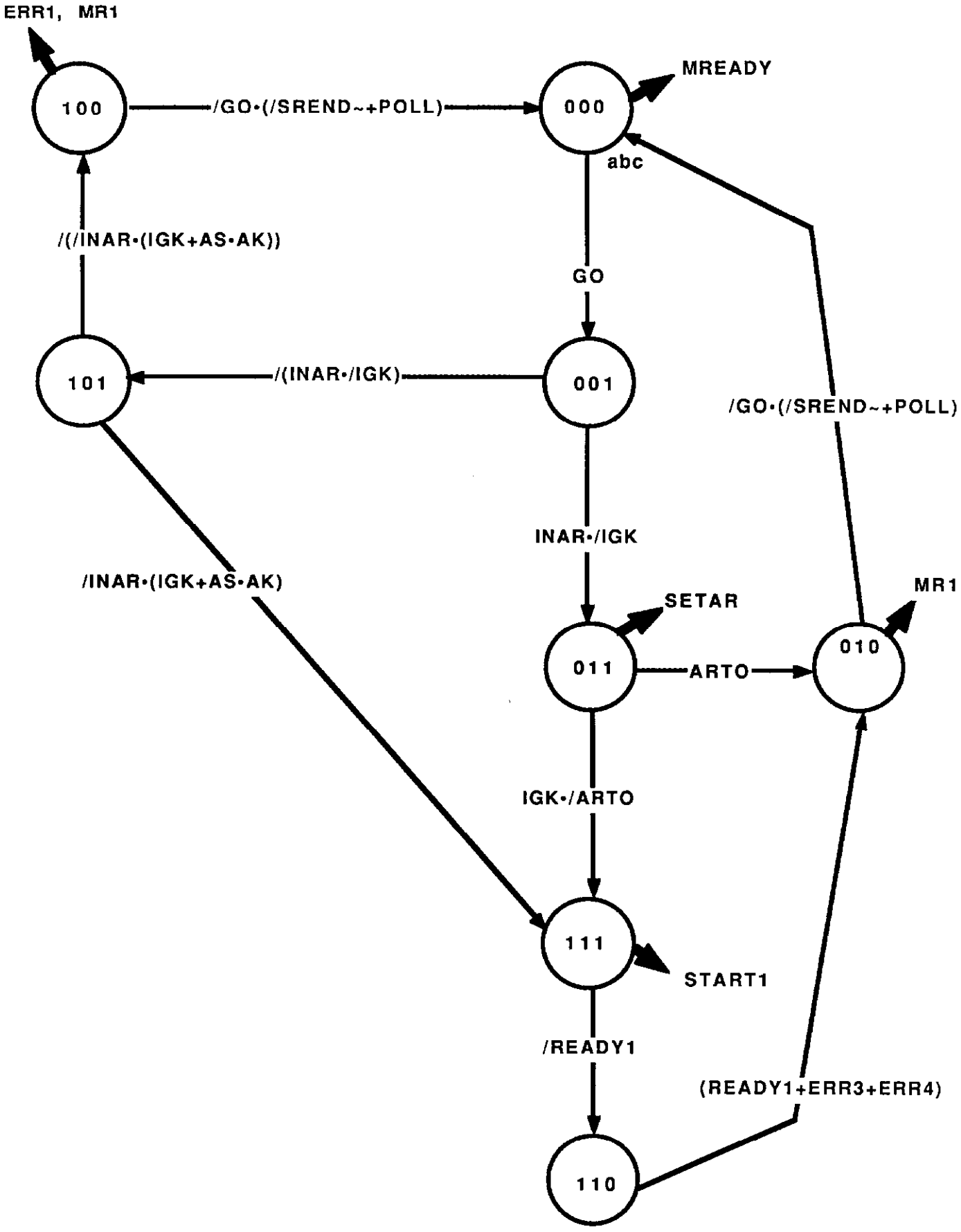
ERR1, MR1
100
/GO•(/SREND~+POLL)
000
MREADY
abc
/(/INAR•(IGK+AS•AK))
GO
101
/(INAR•/IGK)
001
/GO•(/SREND~+POLL)
INAR•/IGK
/INAR•(IGK+AS•AK)
SETAR
MR1
011
ARTO
010
IGK•/ARTO
111
START1
/READY1
(READY1+ERR3+ERR4)
110

Figure 7

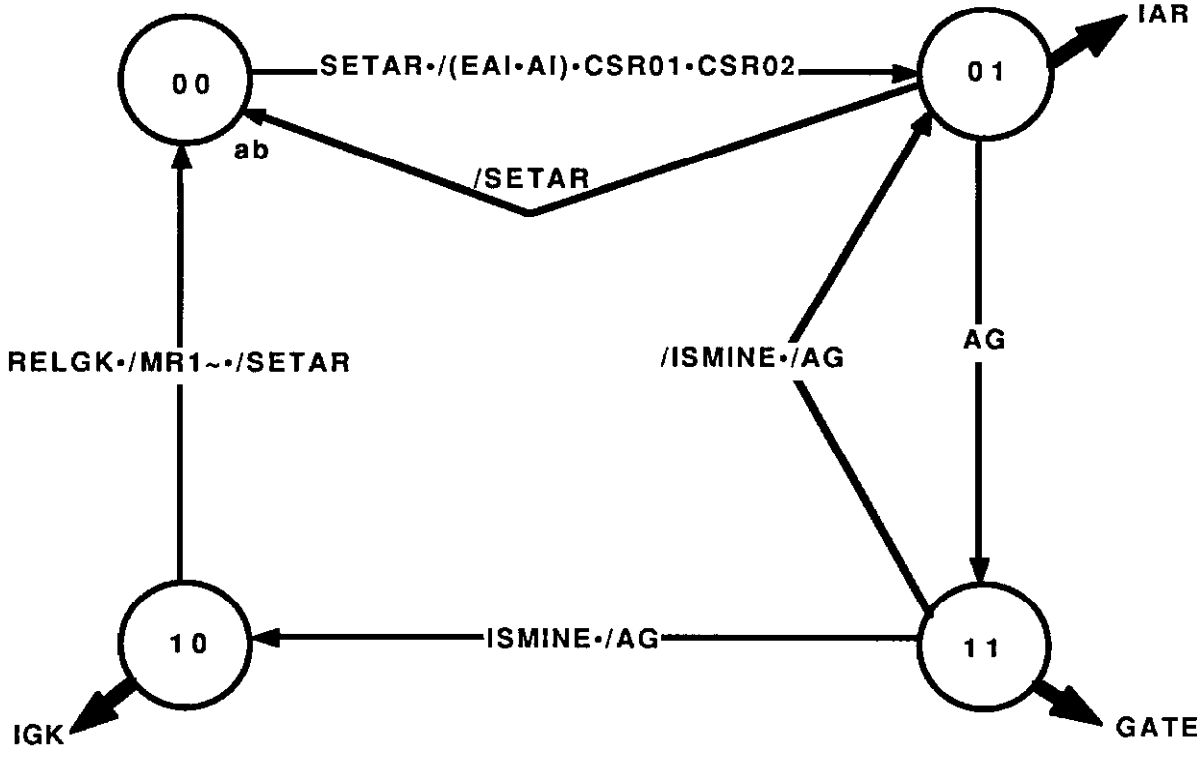
00
01
10
11
SETAR•/(EAI•AI)•CSR01•CSR02
ab
/SETAR
IAR
AG
/ISMINE•/AG
ISMINE•/AG
RELGK•/MR1~•/SETAR
IGK
GATE

Figure 8

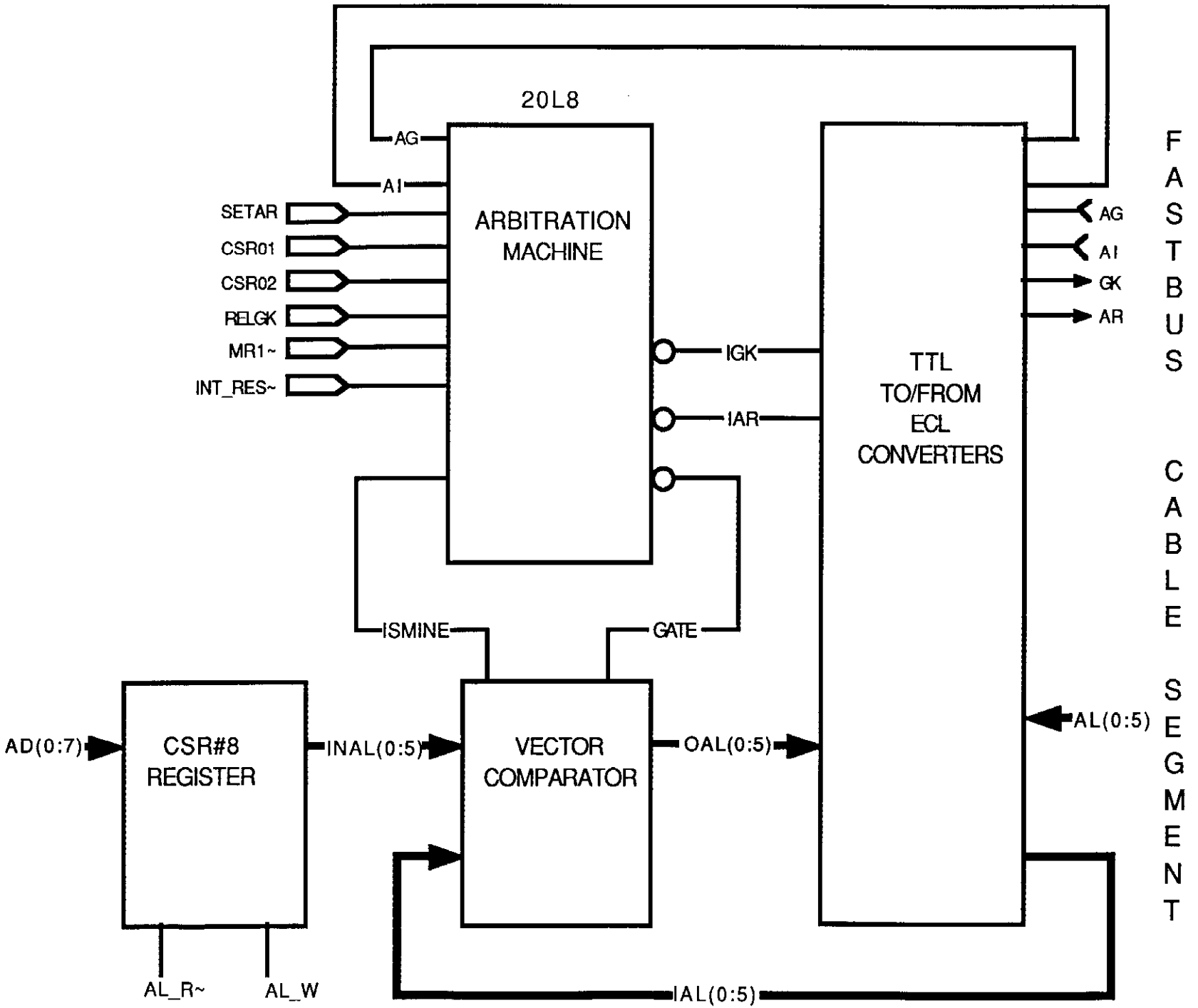
20L8
AG
AI
SETAR
CSR01
CSR02
RELGK
MR1~
INT_RES~
ARBITRATION MACHINE
IGK
IAR
ISMINE
GATE
TTL TO/FROM ECL CONVERTERS
AG
AI
GK
AR
FASTBUS
CABLE SEGMENT
AL(0:5)
AD(0:7)
CSR#8 REGISTER
INAL(0:5)
VECTOR COMPARATOR
OAL(0:5)
AL_R~
AL_W
IAL(0:5)

Figure 9

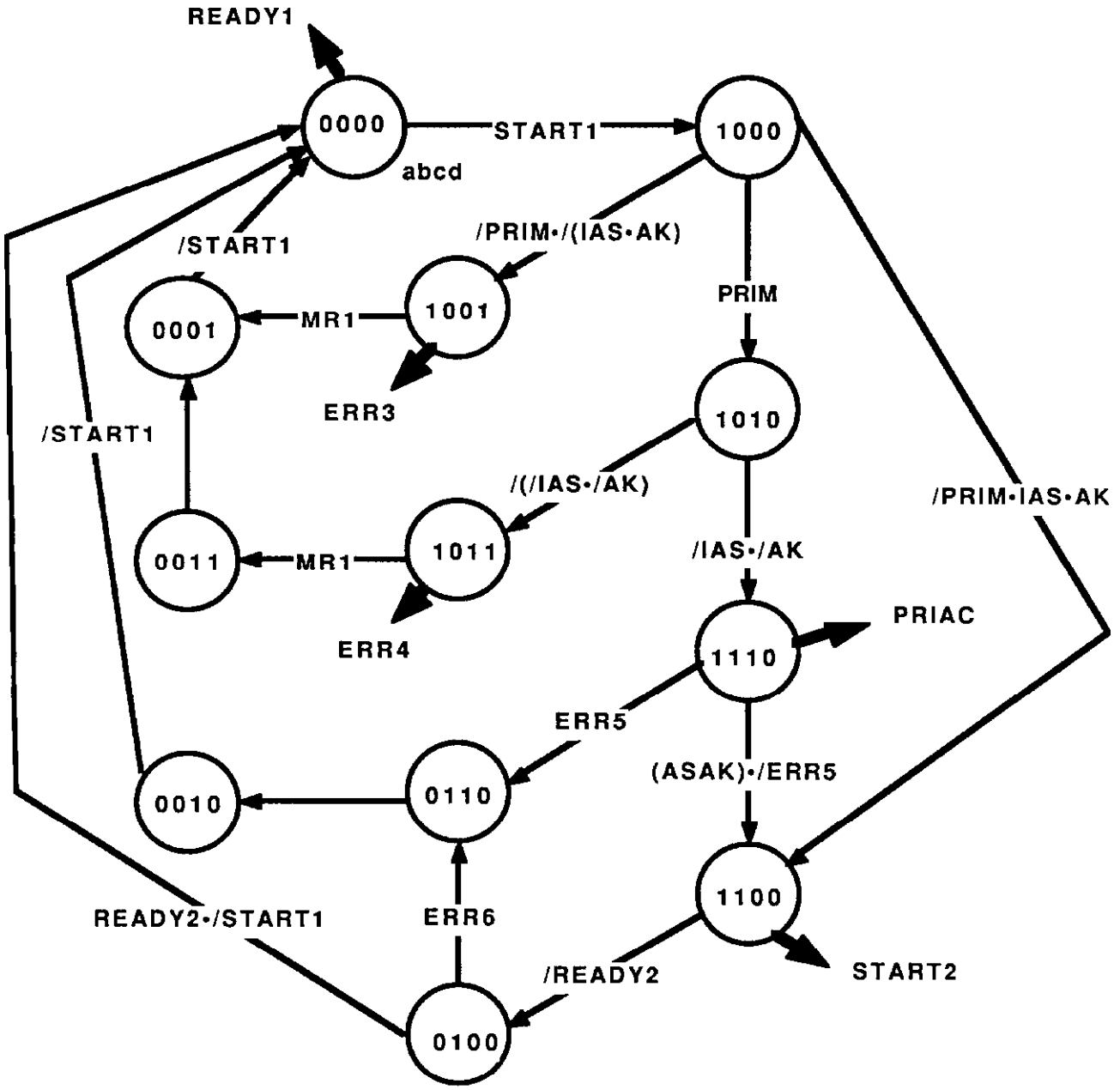
READY1
0000
START1
1000
abcd
/START1
/PRIM•/(IAS•AK)
PRIM
0001
MR1
1001
ERR3
1010
/START1
/(/IAS•/AK)
/PRIM•IAS•AK
0011
MR1
1011
/IAS•/AK
ERR4
PRIAC
1110
ERR5
(ASAK)•/ERR5
0010
0110
1100
READY2•/START1
ERR6
/READY2
START2
0100

Figure 10

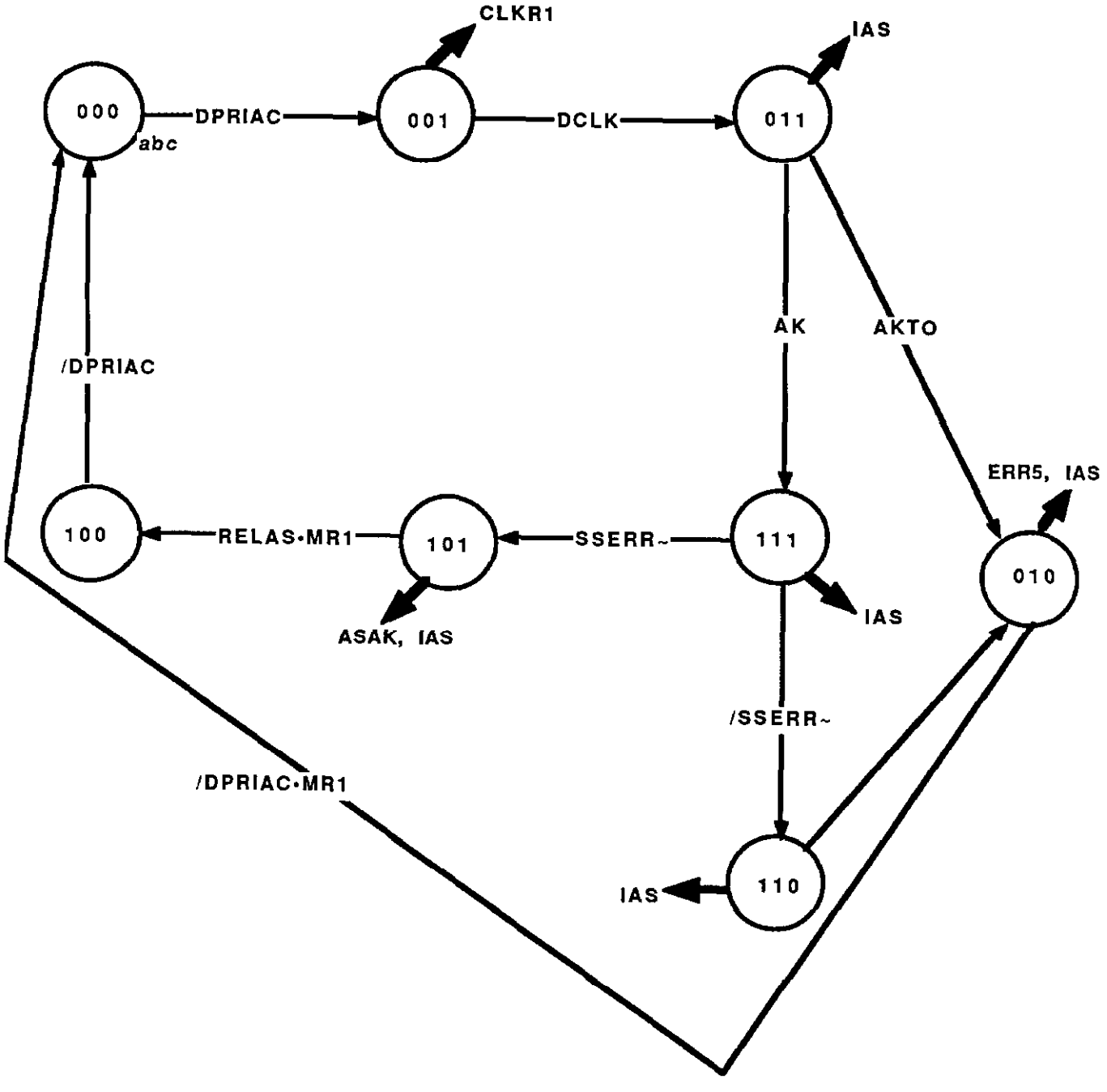
000
abc
DPRIAC
001
CLKR1
DCLK
011
IAS
AK
AKTO
/DPRIAC
100
RELAS•MR1
101
ASAK, IAS
SSERR~
111
IAS
ERR5, IAS
010
/SSERR~
/DPRIAC•MR1
IAS
110

Figure 11

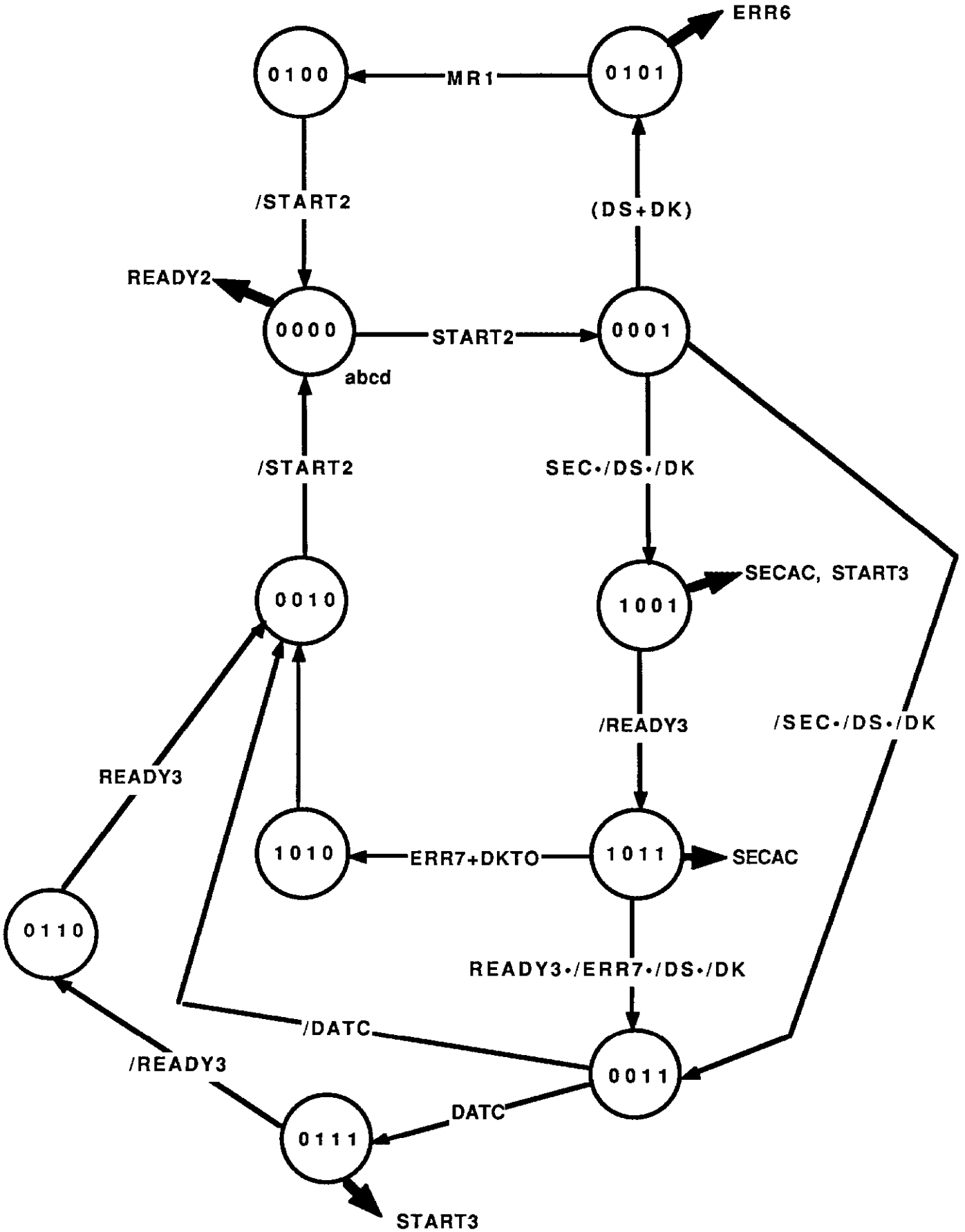
0100
MR1
0101
ERR6
/START2
(DS+DK)
READY2
0000
START2
0001
abcd
/START2
SEC•/DS•/DK
0010
1001
SECAC, START3
/READY3
/SEC•/DS•/DK
READY3
1010
ERR7+DKTO
1011
SECAC
0110
READY3•/ERR7•/DS•/DK
/DATC
/READY3
0011
DATC
0111
START3

Figure 12

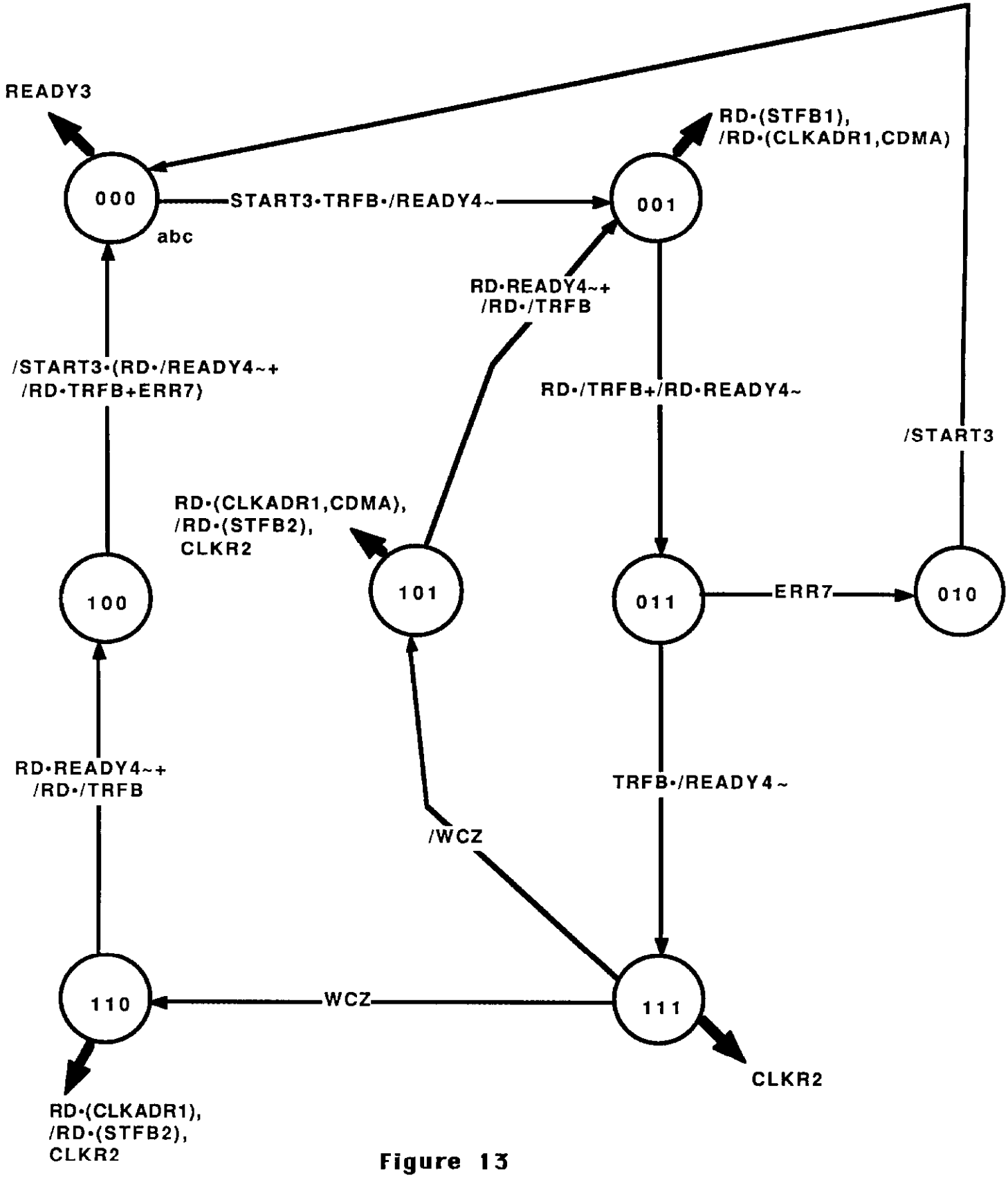
READY3
000
abc
START3•TRFB•/READY4~
001
RD•(STFB1),
/RD•(CLKADR1,CDMA)
RD•READY4~+
/RD•/TRFB
/START3•(RD•/READY4~+
/RD•TRFB+ERR7)
RD•/TRFB+/RD•READY4~
/START3
RD•(CLKADR1,CDMA),
/RD•(STFB2),
CLKR2
100
101
011
ERR7
010
RD•READY4~+
/RD•/TRFB
TRFB•/READY4~
/WCZ
110
WCZ
111
CLKR2
RD•(CLKADR1),
/RD•(STFB2),
CLKR2

Figure 13

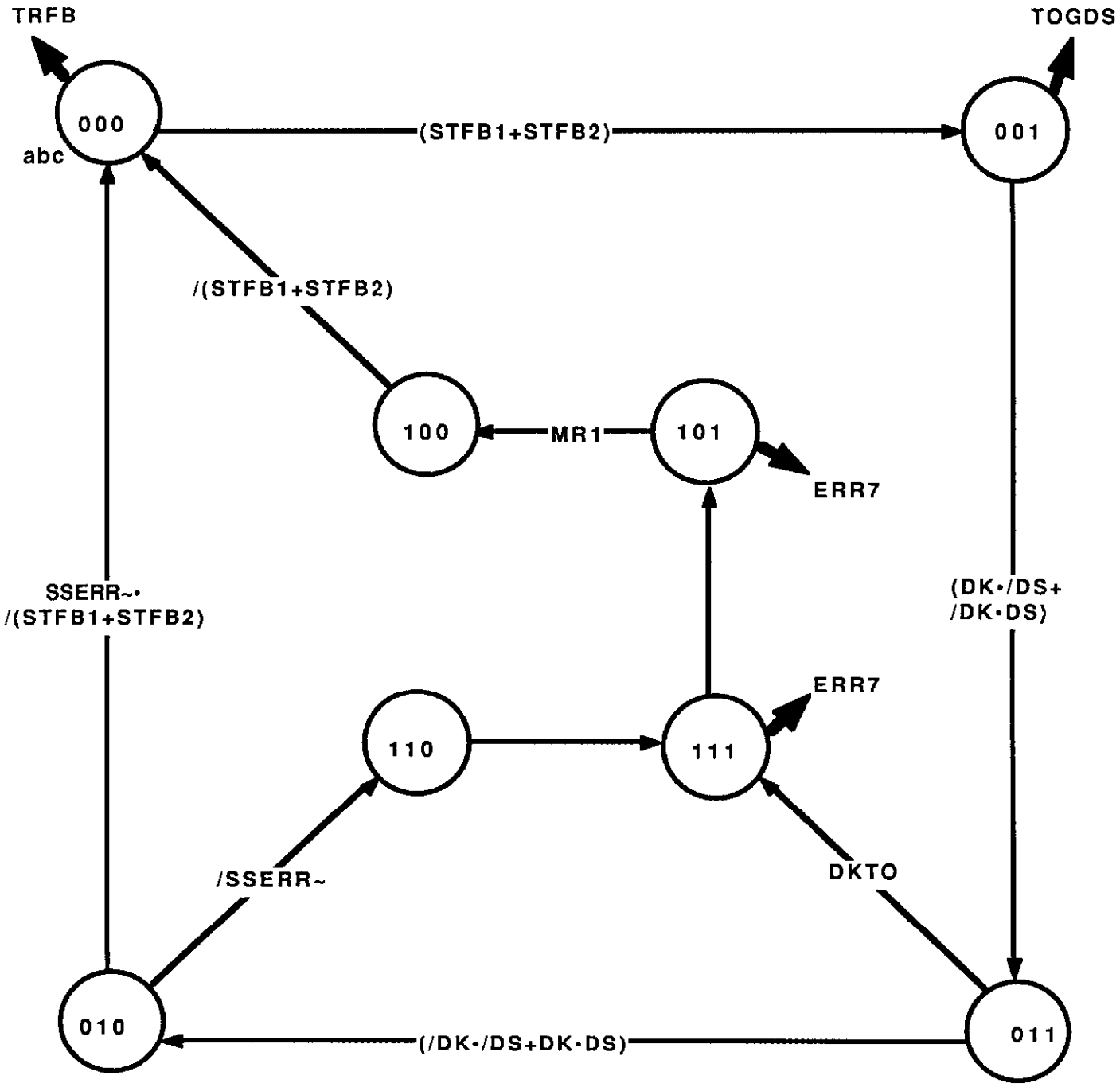
TRFB
TOGDS
000
abc
(STFB1+STFB2)
001
/(STFB1+STFB2)
100
MR1
101
ERR7
SSERR~•
/(STFB1+STFB2)
(DK•/DS+
/DK•DS)
ERR7
110
111
/SSERR~
DKTO
010
(/DK•/DS+DK•DS)
011

Figure 14

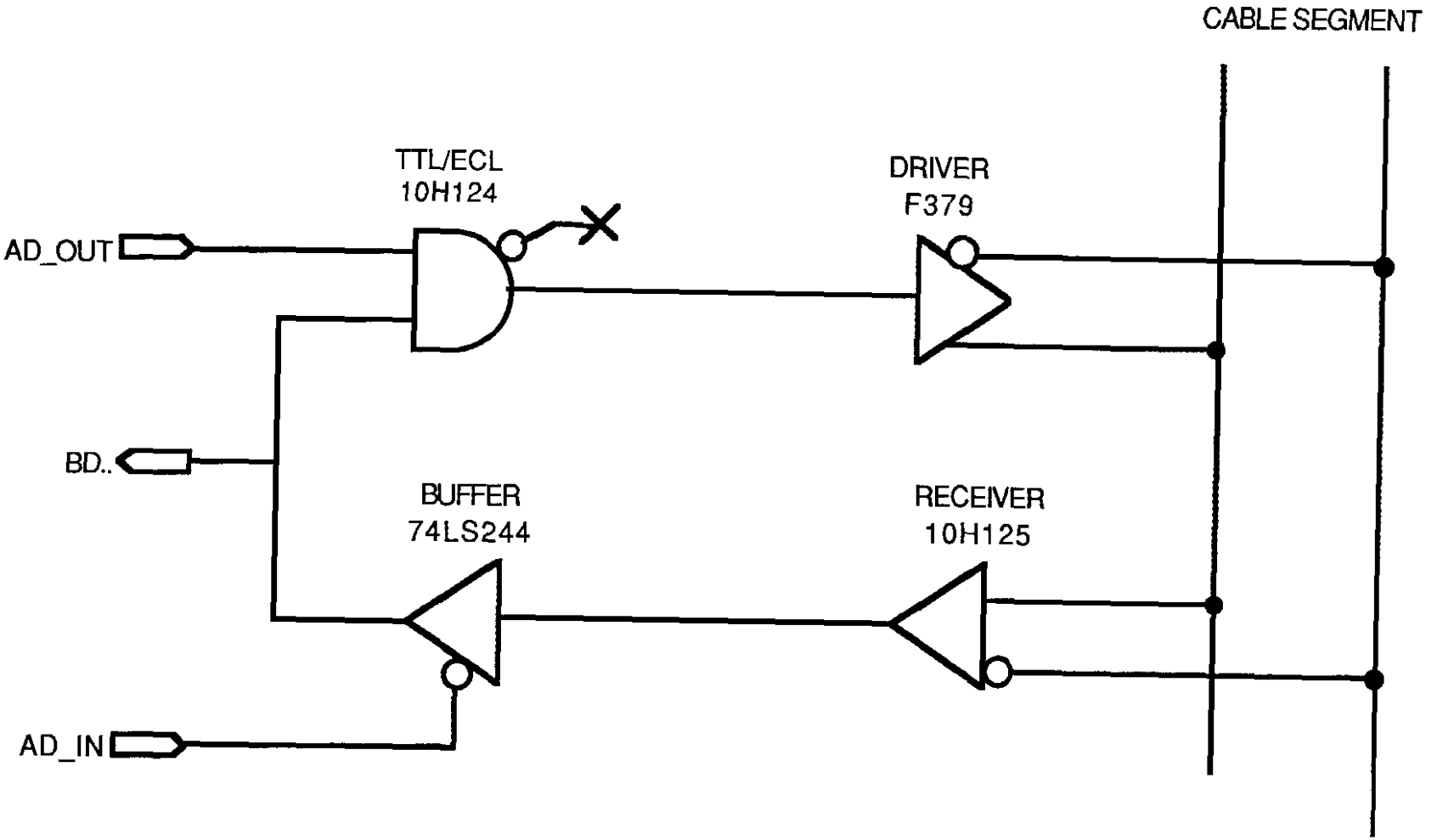
CABLE SEGMENT
TTL/ECL
10H124
DRIVER
F379
AD_OUT
BD..
BUFFER
74LS244
RECEIVER
10H125
AD_IN

Figure 15

```
10    !*************************************************************************
20    !    Parameters: PrAdd - primary address
30    !                ScAdd - secondary address
40    !                BfAdd - Memory location of VME data buffer
50    !                Cnt   - Number of words to transfer
60    !
70    !*************************************************************************
80    SUB vfi_read_dat_block(IN INT32 pradd, scadd, bfadd, cnt)
90
100       ! Does a block transfer to BfAdd, transfering Cnt words.
110
120       CONST read_dat_block_opcode = 8%1210177  ! block read opcode
130
140       ! ...................                    ! part checking Cnt
150
160       vmec_run_block(read_dat_block_opcode, pradd, scadd, bfadd, cnt)
170
180   ENDSUB vfi_read_dat_block
190   !*************************************************************************
200   SUB vmec_run_block(IN INT32 contval, primadd, secadd, dmaadd, dmawc)
210
220       ! This routine loads the various interface registers with the given
230       ! parameters, starts the interface, and waits for the inteface to
240       ! complete the operation.
250
260       CONST prim_reg = 16%FFF040          ! primary address register
270       CONST sec_reg = 16%FFF044           ! secondary address register
280       CONST fb_cont_reg = 16%FFF084       ! FASTBUS control register
290       CONST dma_add_reg = 16%FFF01C       ! DMA address register
300       CONST dma_wc_reg = 16%FFF020        ! DMA word count register
310       INT32 temp
320
330       put32(fb_cont_reg, contval)         ! Load operation code
340       put32(prim_reg, primadd)            ! Load primary address
350       put32(sec_reg, secadd)              ! Load secondary address
360       put32(dma_add_reg, dmaadd)          ! Load address for DMA transfer
370       put32(dma_wc_reg, dmawc)            ! Load word count for DMA transfer
380
390       vmec_go                             ! execute operation
400       vmec_wait                           ! wait till done
410
420   ENDSUB vmec_run_block
```

Figure 16

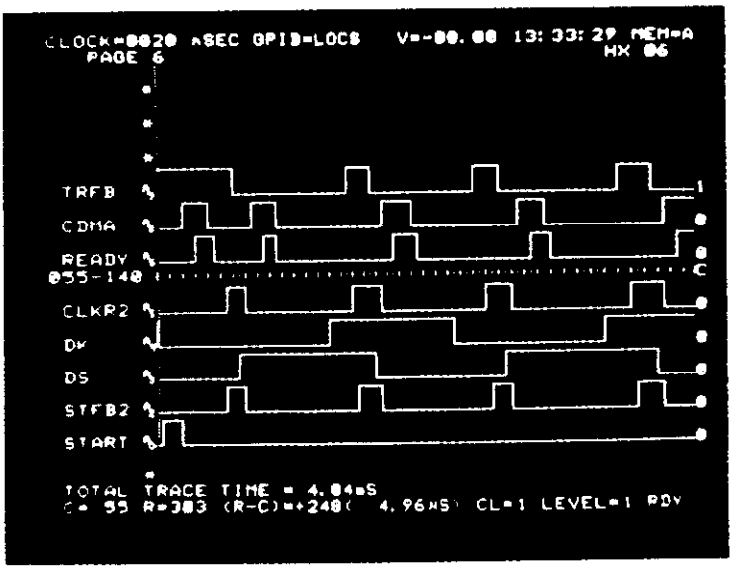
CLOCK=0020 xSEC GPIB=LOCS V=-00.00 13:33:29 MEM=A
PAGE 6
HX 06
TRFB
CDMA
READY
@55-140
CLKR2
DK
DS
STFB2
START
TOTAL TRACE TIME = 4.04mS
C= 55 R=383 (R-C)=+240(4.96mS) CL=1 LEVEL=1 RDY

Figure 17

Figure captions

Figure 1 : Block diagram of board #1 in the VME to FASTBUS master coprocessor.

Figure 2 : Block diagram of board #2 in the VME to FASTBUS master coprocessor.

Figure 3 : The FSM running the VME slave port, supporting 16 or 32 bit transfers.

Figure 4 : The FSM running vectored interrupts.

Figure 5 : The Interrupt logic implemented as two PLDs.

Figure 6 : The interconnections of the different FSMs in the FASTBUS master control.

Figure 7 : The Arbitration Supervisor, with the external process interface.

Figure 8 : The Arbitration Machine.

Figure 9 : The Arbitration logic implemented as three PLDs.

Figure 10 : The Primary Address Supervisor.

Figure 11 : The Primary Address Machine.

Figure 12 : The Secondary Address and Data Cycle Supervisor.

Figure 13 : The Data Cycle Machine.

Figure 14 : The Data Transfer Machine.

Figure 15 : A part of the TTL to ECL conversion and differential driving for the FASTBUS cable segment.

Figure 16 : An example on how a FASTBUS routine is run using PILS.

Figure 17 : Logic analyzer photo showing the FASTBUS port timing.

Table captions

Table 1 : The address map of the VME to FASTBUS master coprocessor.

Table 2 : (a) The VME control and status register bit map,
(b) The FASTBUS status register status codes.

Table 3 : The FASTBUS operation register bit map.