IN9200194

BARC/1991/E/007

BARC/1991/E/007

सत्यमेव जयते

CC-3 CAMAC CRATE CONTROLLER FOR IBM PC

by

A. N. Khare, *M. D. Ghodgaonkar*, *B. R. Bairi*
Electronics Division

1991

BARC/1991/E/007

GOVERNMENT OF INDIA
ATOMIC ENERGY COMMISSION

BARC/1991/E/007

CC-3 CAMAC CRATE CONTROLLER FOR IBM-PC

by

A.N. Khare, M.D. Ghodgaonkar and B.R. Bairi
Electronics Division

BHABHA ATOMIC RESEARCH CENTRE
BOMBAY, INDIA

1991

BARC/1991/E/007

BIBLIOGRAPHIC DESCRIPTION SHEET FOR TECHNICAL REPORT

(as per IS : 9400 - 1980)

01	Security classification :	Unclassified
02	Distribution :	External
03	Report status :	New
04	Series :	BARC External
05	Report type :	Technical Report
06	Report No. :	BARC/1991/E/007
07	Part No. or Volume No. :	
08	Contract No. :	
10	Title and subtitle :	CC-3 CAMAC crate controller for IBM-PC
11	Collation :	38 p., 5 figs., 4 appendixes
13	Project No. :	
20	Personal author(s) :	A.N. Khare; M.D. Ghodgaonkar; B.R. Bairi
21	Affiliation of author(s) :	Electronics Division; Bhabha Atomic Research Centre, Bombay
22	Corporate author(s) :	Bhabha Atomic Research Centre, Bombay - 400 085
23	Originating unit :	Electronics Division, BARC, Bombay
24	Sponsor(s) Name :	Department of Atomic Energy
	Type :	Government
30	Date of submission :	June 1991
31	Publication/Issue date :	July 1991

Contd... (ii)

40	Publisher/Distributor :	Head, Library and Information Division, Bhabha Atomic Research Centre, Bombay
42	Form of distribution :	Hard Copy
50	Language of text :	English
51	Language of summary :	English
52	No. of references :	9 refs.
53	Gives data on :	

60 Abstract : The specifications and implementation details of CAMAC Crate Controller CC-3 for IBM-PC compatible as a host computer, having capability to transfer high speed data with direct memory access (DMA) scheme and logic to execute CAMAC cycles directly from the crate controller, to implement the block algorithms specified in ANSI/IEEE Std. 683-1976 (Reaff-1981) are described. The maximum data transfer rate measured with 8 bit interface of PC-AT is 240K byte per second. This work is carried out under Seventh Five Year Plan Project on "Modernisation of Reactor Control Instrumentation and Development of CAMAC and Fastbus Instrumentation".

70 Keywords/Descriptors : CAMAC SYSTEM; IBM COMPUTERS; MICRO-COMPUTERS; SPECIFICATIONS; EQUIPMENT INTERFACES; DATA ACQUISITION SYSTEMS; REACTOR CONTROL SYSTEMS; C CODES; DATA TRANSMISSION

71 Class No. : INIS Subject Category : F51.00; E24.00

99 Supplementary elements :

CONTENTS

1. INTRODUCTION	2
2. CC-3 CRATE CONTROLLER	2
3. CAMAC CYCLE SEQUENCERS	4
4. BLOCK TRANSFER CHANNEL	5
5. IMPLEMENTATION	7
6. SOFTWARE CONSIDERATIONS	8
7. PERFORMANCE	9
8. CONCLUSION	10

ACKNOWLEDGEMENTS
REFERENCES

APPENDIX A: CC-3 REGISTER DETAILS
APPENDIX B: DRIVER CAMAC.PAS
APPENDIX C: TEST PROGRAM CC1.PAS
APPENDIX D: TEST PROGRAM CC3.PAS

FIGURES

CC-3 CAMAC CRATE CONTROLLER FOR IBM-PC

1. INTRODUCTION

The CAMAC crate controller CC-3 has been designed and developed in the Electronics Division of BARC [4]. The PC based CAMAC system, using this controller yields data acquisition rate of about 240 Kb/sec. The controller also provides on-line data processing capability to the system. The crate controller CC-3 is described in this report.

The crate controller has dedicated interface to controlling computer, an IBM-PC [1]. In the case of computer is controlling execution of each of the CAMAC operation i.e. 'single CAMAC operation', frequent communication is involved between computer and crate. In such situations, the dedicated interface provides higher CAMAC cycle execution rate and hence higher data transfer rate, as compared to those provided by the standard interfaces like RS-232 or GPIB-488 between the controller and the computer. The dedicated interface of CC-3 uses DMA (direct memory access) channel of PC and improves the data transfer rate further.

The controller provides local intelligence to execute a sequence of CAMAC cycles directly i.e. a 'block CAMAC operation' [2], without involvement of computer. This improves the rate of CAMAC cycle execution and hence the effective data rate. The computer can do jobs concurrently with the CAMAC opereations handled by the crate controller. These features are useful in data acquisition applications where acquisition is done with higher data rates along with on-line processing.

Section 2 describes about IBM-PC based CAMAC systems and the functionality provided by CC-3 crate controller. The important blocks are described in section 3 and section 4, where details of CAMAC cycle generators and block transfer channel are given. Implementation details of CC-3 are given in section 5. The intended usage of controller along with the software aspects are described in section 6. The data transfer rates for single as well as for block CAMAC operations are reported in section 7. We conclude the report with Section 8, which says about limitations observed in CC-3 and future development plan to overcome these limitations.

2. CC-3 CRATE CONTROLLER

For medium size systems, the instrumentation can be accommodated in a single crate. A single crate CAMAC system, interfaced to IBM-PC compatible, on a dedicated interface at crate controller, forms a versatile and flexible instrumentation system. The vast variety of peripheral support, availability of interfaces for various communication standards, powerful displays and secondary storage, at lower cost, makes IBM-PC, the right choice as the controlling computer. The upgraded range of PC compatible machines are available with downward hardware and software compatibility. It is easy to upgrade the system for higher computing power without any major modifications in interface hardware and application software. The software for CAMAC system has to take care of configurability. It needs comprehensive user interface to tailor the program for a particular set up. The latest packages like WINDOWS are available on IBM-PCs to provide powerful and easy-to-use user interface.

As mentioned earlier, CC-3 has dedicated interface to IBM-PC and the above mentioned advantages of standalone CAMAC system with IBM-PC as control computer can be obtained by usage of CC-3. The controller has a set of registers, mapped in I/O space of PC. Controller and PC communicates information through these registers. Controller uses an interrupt line of PC interface to report asynchronous events and a DMA channel for high speed data transfer.

The controller works in two modes. In 'single CAMAC operation ' mode, controller initiates 'normal' or 'compressed' CAMAC cycles on software instructions from PC. The controller holds CAMAC vector i.e., N, A, F information and write data contents in the registers before execution of CAMAC cycle. The results of CAMAC cycle execution i.e. the read data contents and status information are stored in appropriate registers.

In 'block CAMAC operation' mode, the controller services the 'block commands', which request controller to execute a sequence of CAMAC cycles. The crate controller has 'block transfer channel' hardware. The sequence of CAMAC cycles is defined by programming the block transfer channel to work under the control of certain algorithm. The 'block command' can be asserted either by software instruction from PC or by the hardware signal generated on certain conditions. The PC communicates with block transfer channel through a command and a status registers. During the block command opereation, data transfer to PC is performed on the DMA channel of PC.

Apart from these two modes which perform synchronous operation of CAMAC system, a logic is defined to perform asynchronous opereation of CAMAC system. The crate controller has 'LAM' handling logic, which detects the asynchronous signals LAMs (Look At Me) from multiple modules. It has ability to resolve priority between multiple LAM sources occouring simulteneously and reporting the highest priority LAM number in LAM register. The LAM handling logic generates interrupt on the PC interface and writes the number of the highest priority LAM source in 'LAM register'. The interrupt service routine uses the 'LAM register' to detect the highest priority LAM source and performs the LAM service functions.

The block diagram of CC-3 is given in fig.1. The controller provides functionality as mentioned earlier. There are total sixteen registers of 8 bit width. They are mapped as I/O registers and interfaced to 8 bit data bus of PC. The details of registers are given in Appendix A. The single CAMAC cycles are performed with the CAMAC cycle generators, i.e. normal CAMAC cycle generator and compressed CAMAC cycle generator. The block transfer channel executes the block CAMAC cycles. The 'LAM handling logic' prioratises the multiple LAM requests from modules and provies functionality as described earlier. The clock unit provides stable 10 MHz clock to the synchronous state machines. The PC interface on crate controller provides decoding logic for I/O registers and buffers all signals coming from the PC interface cable.

3. CAMAC CYCLE SEQUENCERS

The CAMAC cycle sequencers provide the control of signals appearing on the CAMAC dataway, conforming to ANSI/IEEE Std. 583-1982 for Modular Instrumentation and Digital Interface System(CAMAC) [1].

3.1 NORMAL CAMAC CYCLE SEQUENCER:

The normal CAMAC cycle of one microsecond is shown in fig.2. During normal CAMAC cycle the strobes S1, S2 and Busy are generated and the N,A,F and data buses on dataway are enabled for appropriate duration. It is generated by 'normal CAMAC cycle sequencer', a synchronous state machine, running at 100 nanosecond per step. The sequencer is started by the performing the read 'status register' I/O instruction. This I/O read cycle is extended by asserting the 'peripheral ready' line for 1 microsecond on PC interface. The normal CAMAC cycle is executed during this time. The status of the CAMAC cycle which is just finished is stored in 'status register', which is read by PC in the extended read cycle. The sequencer also has interface to block transfer channel to generate CAMAC cycles from block transfer channel without any involvement of PC.

To perform 'write to a CAMAC register' operation, the 24 bit data is written in W1, W2 and W3 registers by three I/O operations. (W1 containing lowest significant byte). The N, A, F vector is loaded in N, A and F registers respectively by three I/O instructions. The normal CAMAC cycle is initiated by giving the I/O instruction from PC. The PC reads the status during the same expanded cycle. To perform 'read data from CAMAC register', similar sequence is performed, except the 24 bit data is read from R1, R2 and R3 after CAMAC cycle executed. To perform a CAMAC cycle with functions involving no data transfer, the read or write registers are not manipulated.

3.2 COMPRESSED CAMAC CYCLE SEQUENCER:

The 'compressed CAMAC cycle', as shown in fig.3, is generated by writting appropriate command (i.e. initialise, clear etc.) in 'Z-C-I register'. The I/O write cycle is extended by asserting peripheral ready line for 800 nanoseconds and a compressed CAMAC cycle of 800 nanoseconds is executed during that period. It is generated by 'compressed CAMAC cycle sequencer'. The sequncer is also a synchronous state machine running at 10 MHz clock speed. It generates busy and S2 signals and enables the 'Z' or 'C' line on the dataway interface.

4. BLOCK TRANSFER CHANNEL

In many practical applications, there is a need of executing a well defined sequence of CAMAC instructions. The sequence has fixed conditions to start and stop. The pattern of CAMAC vector (N, A, F) modification for consecutive instructions in the sequence can be defined by an algorithm and the decision to advance to the next instruction in the sequence can be taken by finite conditions. The hardware which executes such sequences is called 'block transfer channel'. CAMAC standard ANSI/IEEE 683-1976 for Block Transfer in CAMAC Systems [2], suggests four algorithms to generate sequences by block transfer channel. The algorithms are general and are representatives of many practical situations.

The algorithms are:

1. Q stop:

starting condition	- on 'block command' issued by PC
stopping condition	- on invalid 'Q' response of a CAMAC cycle
NAF modification	- not modified after initial value is given by block command
CAMAC instruction advancing	- unconditional

2. Stop on terminal count:

starting condition	- on 'block command' issued by PC
stopping condition	- after fixed number of CAMAC instructions
NAF modification	- not modified after initial value is given by block command
CAMAC instruction advancing	- unconditional

3. Q repeat:

starting condition	- on 'block command' issued by PC
stopping condition	- after fixed number of CAMAC instructions
NAF modification	- not modified after initial value is given by block command
CAMAC instruction advancing	- advance only on valid Q response of CAMAC cycle

4. Address scan:

starting condition	- on 'block command' issued by PC
stopping condition	- after fixed number of CAMAC instructions OR NAF vector reaches to the last sub-address of last normal station in the crate i.e., N = 24, A= 15

NAF modification	- on (N = 15 or Q invalid)
	N = N + 1
	A = 0
	OR
	on (Q valid AND A = 15)
	A = A + 1

CAMAC Instruction advancing	- Unconditional

The block diagram of block transfer channel is shown in fig.4. The crate controller provides 'block transfer channel' hardware with the specifications mentioned in the preceeding paragraph. The block transfer channel has a `mode register' to select one of the algorithms mentioned earlier. After giving initial (N, A, F) vector, the sequence of block transfer channel is initiated by a I/O instruction from PC. The block transfer channel, after completion of execution of sequence, reports the occurence of `end -of-sequence', either by interrupt or by a status in `report register'. The `report register' also reports the important status information of the last CAMAC instruction in the sequence. If the sequence executes a data transfer instruction, then the data is transferred between controller and PC main memory through DMA channel.

Before initiation of block transfer channel following preparations have to be done,

The DMA controller is initialised with count and address pointer of PC memory buffer. If the 'end-of-sequence' is reported by an interrupt, the global variable, used to communicate between interrupt service routine and the main program, is initialised. The 'block transfer channel' operation is started by two optional conditions. Either the PC generates software 'dummy read' I/O instruction or front panel 'external trigger' is generated by external source. In case of external trigger the preparation of block channel must be ensured before triggering. Unexpected behaviour can be observed if this requirement is not fulfilled. The controller can even hang in an undetermined state in such situation.

The block transfer channel is an asynchronous sequencer, which performs a basic sequence of generating a CAMAC cycle followed by data transfers associated with it. For a CAMAC operation involving data transfer, three bytes of data are required to be moved between PC and controller. The block transfer channel transfers these three bytes by using direct memory access facility of PC. Hence the basic sequence is to generate a CAMAC cycle (in state 0) followed by transfer of three bytes of data (in states 1, 2 and 3). The algorithms are implemented by repeating the basic sequence. The state is advanced to the next asynchronously i.e., there is no common clock employed for this purpose. The "state advancement logic" monitors the status lines and generates a "advance pulse" when the required transitions take place on status lines. To implement some algorithms, basic sequence, as mentioned above, needs to be modified. It is modified by "sequence modification logic", which overrides the "advance pulses" from state advancement logic". The "start logic" and "stop logic" monitor the conditions to start and stop respectively and take appropriate actions. The "NAF modification logic" does the modification of "NAF vector" for consecutive CAMAC cycles.

In state 0, the sequence asserts CAMAC signal to generate the CAMAC cycle and monitors S1, S2, Busy signals. In state 1, 2 and 3 the sequence generates DREQ on PC interface and monitors the status of T/C, DACK, IORD. The strobe pulses for read and write registers are generated in these states, depending upon direction of data transfer. The 3 registers are accessed one by one, i.e., one each in the states 1, 2 and 3. The typical sequence of operation is shown in fig. 5.

5. IMPLEMENTATION

CC-3 crate controller is a two width CAMAC module. It provides CAMAC dataway interface at control station (Station 25) and at a normal station (Station 24), conforming to CAMAC electricall and mechanical standards. It is powered by dataway power distribution bus. It has interface to PC on a flat cable connector at front panel. The PC expansion bus signalsignals are buffered on a PC add on card, which fits in expansion slot of PC. The PC add-on buffer card provides switches to select DMA channel, interrupt vector and I / O base address of the registers. The details are given in [8]. The crate controller and the PC add-on buffer card are interconnected by buffered signals with a flat cable. The crate controller logic is implemented with SSI and MSI TTL building block ICs and custom logic on PROMs. The interconnections between components are done by mounting them on four layer printed circuit board.

6. SOFTWARE CONSIDERATIONS

In a typical data acquisition application, the data acquired from CAMAC is stored in buffers and then processed. The application program performs two functions. First is genetation of CAMAC cycles or managing CAMAC system in general (data acquisition) . Second is providing overall functionality to system (data processing).

The flow of the program is decided by the status information from crate controller and other parameters in the program. The asynchronous events are handled by interrupt service routine. It is linked to the incoming interrupts from the controller either on LAM occourance or on end-of-sequence from block channel.

For execution of single CAMAC cycles, PC executes a sequence of I/O instructions per CAMAC cycle. The PC is tightly coupled to the operation of executing of CAMAC cycles. Hence the data acquisition and data processing operations are sequential.

In the application program the block transfer channel facility should be used when sequencing of CAMAC operations is supported by the algotithms on block channel. It gives three advantages. First is the speed of CAMAC cycle execution is more as the hardware is employed for sequencing. The second advantage is the data is direcly transferred to the PC main memory buffer hence no overheads for data handling at PC. The third and the major advantage is the PC is free to do background data proceessing when block transfer channel is performing the sequence of CAMAC opereations i.e. data acquisition. The PC is informed about the end-of-sequence by interrupt. In this case only an interrupt service routine is required to handle data acquisition.

If the same sequence is to be initiated repeatedly, then block transfer channel can be restarted at the controller itself by the hardware trigger on the end-of-sequence of the previous sequence. In this case the interrupt can be supressed and the overheads on PC can be altogether eliminated.

In some cases the sequence is to be started on LAM request from module. This can be efficiently done by supressing the interrupt to PC on LAM and using the LAM to start the block transfer channel directly on the controller.

In these cases the 'cpu' involvement in executing CAMAC cycles is absolutely nill and the entire 'cpu' time can be utilised for background data processing.

The CAMAC related part in the application program can be isolated in the form of driver library. The driver can be upgraded independently and can even be substituted with diffrent drivers to use the same application program with different controllers. Driver software is provided along with CC-3, which follows ANSI /IEEE Std. 758-1979 Subroutines for CAMAC [7]. The driver is written to use with Turbo pascal ver.4, in the form of Turbo Pascal Unit. CAMAC.TPU is provided along with the source code CAMAC.PAS. The user can tailor the subroutines in the driver to improve performance of the driver for specific requirements. The source listing of driver is given in appendix B.

7. PERFORMANCE

In single CAMAC cycle execution mode the maximum rate of 40 Kb/sec. can be achieved, provided no other operations are done by PC. The 4 algorithms supported on block channel are implemented in software with single CAMAC operations for speed comparison. The speed is affected by the complxity of algorithm in these software implementations. The program to calculate single CAMAC cycle execution speed is given in appendix C.

Algorithm	Data Transfer Rate	
	Single Mode Kb/sec.	Block Mode Kb/sec.
Stop on t/c	40.3	245.7
Q stop	39.3	245.7
Address scan	28.2	245.7

Data transfer rate v/s algorithms for single and block mode
Block Size = 16K Word

The raw data transfer rate of block transfer channel measured on Logic Analyser is 250 Kb/sec. The practical achievable data transfer rate is 240 Kb/sec. The data transfer rate is not affected by complexity of algorithm as the implementation is done in hardware. The overheads associated with preparation of block transfer channel reduces the rate upto 225 Kb/sec., for smaller block sizes. The program to calculate block CAMAC cycle execution speed is given in appendix D.

Block size K Word	Block transfer Rate Kb / sec
4	223.4
8	234.1
16	245.7

Block size v / s data transfer rate for Block mode operation

The controller is used successfully for configuring a multiparameter data acquisition system at BARC - TIFR accelerator facility. In the system Lecroy 2259B ADCs with conversion time of 106 micro-sec are used. In a typical experiment with the counting rate of 3000 events / sec, with 8 parameters, writing data to the hard disk, 1D or 2D spectra building concurrently without data loss has been achieved [3]. The bottleneck in the system performance is the conversion time of ADCs and can be improved significantly by employing faster ADCs.

8. CONCLUSION

The controller provides fairly good performance for PC based CAMAC systems [9]. For a random sequence of CAMAC cycles the controller has to be used in slower single CAMAC cycle mode. A list processing crate controller which provides fast execution of random CAMAC cycles can solve this problem. The controller provides 8 bit interface with PC which can be improved to 16 bit to work with IBM-PC ATs which will give a major speed improvement. Both these improvements are incorporated in the crate controllers designed in Electronics Division of BARC to cater to the future requirements.

ACKNOWLEDGEMENT

The reliability and field testing of CC-3 performed by Dr. A. Chatterj Nuclear Physics Division, BARC at Pelletron facility, TIFR is gratefully acknowledged. Helpful suggestions and cooperation by members of CAMAC group, Electronics Division are also acknowledged.

REFERENCES

1. ANSI/IEEE Std. 583-1982
   Modular Instrumentation and Digital Interface System (CAMAC)

2. ANSI/IEEE Std. 683-1976 (Reaff.1981)
   Block Transfers in CAMAC Systems.

3. A. Chatterjee
   AC-2 A data Acquisition System For Pelletron
   To be published as B.A.R.C. Technical Report

4. A.N.Khare et.al.
   CAMAC Crate Controller For IBM-PC with DMA
   pp. 496 Proceedings of National Symposium on Nuclear Electronics and Instru mentation (BRNS-DAE-1989)

5. Lewis C. Eggebrecht
   Interfacing to the personal computer
   Howard W. Sams & Co. USA

6. V. Aruna et.al.
   Development of CAMAC and Fastbus Instrumemtation
   B.A.R.C. Technical Report (BARC-1394) (1987)

7. ANSI/IEEE Std. 758-1979 (Reaff.1981)
   Subroutines For CAMAC

8. Electronics Division Internal Circulation
   CC-3 User Manual

9. A. Chatterjee et.al.
   PC-Horizon Based Data Acquisition System with Ethernet Link
   Symposium on Nuclear Physics (India), 1990.

APPENDIX-A

CAMAC CONTROLLER IBM-PC DECODED ADDRESS:

HEX ADDRESS	DECIMAL ADDRESS	SIGNAL	DESCRIPTION
0300	768	NS	STATION NO.
0301	769	AS	SUBADRESS
0302	770	FS	FUNCTION
0303	771	CS1	PENDING
0304	772	SS	START CAMAC
0305	773	CS2	SOFT RESET
0306	774	ZS	START Z,C
0307	775	MS	BLOCK MODE
0308	776	RS1	READ1
0309	777	RS2	READ2
030A	778	RS3	READ3
030B	779	LS	LAM REGISTER
030C	780	WR1	WRITE1
030D	781	WR2	WRITE2
030E	782	WR3	WRITE3
030F	783	READ_DMA	START BLOCK

REGISTER INFORMATION

1) STATION REGISTER: N REGISTER PORT 768 (WRITE ONLY)

D0	D1	D2	D3	D4	D5	D6	D7
N1	N2	N4	N8	N16	-	-	-

2) SUBADDRESS REGISTER: A REGISTER PORT 769 (WRITE ONLY)

A1	A2	A4	A8	-	-	-	-

3) FUNCTION REGISTER: F REGISTER PORT 770 (WRITE ONLY)

F1	F2	F4	F8	F16	-	-	-

4) REPORT REGISTER: PORT 771 (READ ONLY)

BLOCK OVER	Q STATUS 0=VALID	LAM REQ	N>23	X STATUS 0=VALID	-	-	-

* MASKABLE INTERRUPT: BLOCK OVER
  i.e. D4 OF MODE REGISTER IS 0 THEN MASK.

* NON MASKABLE INTERRUPT: LAM REQUEST

* STATUS INFORMATION: Q STATUS & X STATUS,N>23

* X STATUS IS NOT PROVIDED IN THIS CONTROLLER

5) STATUS REGISTER: PORT 772 (READ ONLY)

X STATUS 1=VALID	Q STATUS 1=VALID	I STATUS 1=PRESENT	-	-	-	-	-

* STATUS REGISTER IS MAPPED ON PORT 772,WHICH ALSO EXECUTES CAMAC CYCLE.

6) SOFT RESET REGISTER: PORT 773 (WRITE ONLY)

BLOCK OVER BIT PEND REG	-	MODE REG	Q AND X BITS PEND REG + BLOCK CHANNEL CIRCUIT	-	-	-	-

7) Z-C-I REGISTER: PORT 774 (WRITE ONLY)

Z CYCLE 1=Z CYC	C CYCLE 1=C CYC	I STATUS 1=SET	-	-	-	-	-

* Z CYCLE OR C CYCLE IS EXECUTED BY WRITTING THE CORRESPONDING BIT IN THE REGISTER,WHILE I STATUS SHOULD BE MAINTAINED AS PREVIOUS.
* I STATUS CAN BE CHANGED BY TOGGLING THE CORRESPONDING BIT IN THIS REGISTER.

8) MODE REGISTER: PORT 775 (WRITE ONLY)

ADDRESS SCAN	Q STOP	STOP ON T/C	Q REPEAT	I MASK 0=DIS 1=ENA	DATA X'fer 0=YES 1=NO	READ/ WRITE 0=READ 1=WRITE	EXT TRIGGER 1=ON

9) READ REGISTER: R1 PORT 776 (READ ONLY)

R1	R2	R3	R4	R5	R6	R7	R8

10) READ REGISTER: R2 PORT 777 (READ ONLY)

R9	R10	R11	R12	R13	R14	R15	R16

11) READ REGISTER: R3 PORT 778 (READ ONLY)

R17	R18	R19	R20	R21	R22	R23	R24

12) LAM REGISTER : PORT 779 (READ ONLY)

L1	L2	L4	L8	L16	-	-	-

13) WRITE REGISTER: W1 PORT 780 (WRITE ONLY)

W1	W2	W3	W4	W5	W6	W7	W8

14) WRITE REGISTER: W2 PORT 781 (WRITE ONLY)

W9	W10	W11	W12	W13	W14	W15	W16

15) WRITE REGISTER: W3 PORT 782 (WRITE ONLY)

W17	W18	W19	W20	W21	W22	W23	W24

16) START BLOCK: DUMMY REGISTER PORT 783 (READ ONLY):

APPENDIX-B

{listing of CAMAC.PAS }

```
unit camac;
interface
uses dos,crt;
const
            buff_length = 2048;
Type
    LongData  = array[1..3] of integer;
    GroupAdrs   = Record
                          Branch,
                          Crate,
                          Station,
                          Adrs  : integer
                         End;
    LAMPointer   = ^LAMadrs;

         LAMadrs  = Record
                          Branch,
                          crate,
                          station,
                          adrs     : integer;
                          service  : pointer;
                          NextLAM: LAMPointer;
                          Case m: integer of
                               1,2,3,4,5,6,7,8,9,10,
                               11,12,13,14,15,16,17,
                               18,19,20,21,22,23,24: (ladrs: integer);

                               -24,-23,-22,-21,-20,
                               -19,-18,-17,-16,-15,
                               -14,-13,-12,-11,-10,
                               -9,-8,-7,-6,-5,-4,-3,
                               -2,-1:(data24: LongData);
                         end;

    buffer     = array[1..buff_length,1..3] of byte;

    conadrs    = record
                       repcount:longint;
                       tally  :longint;
                       end;

  Procedure CDREG(b,c,n,a: integer; Var Ext: GroupAdrs);
  Procedure CFSA(f:integer;ext:GroupAdrs;var int:LongData;
                       var q:boolean);
  Procedure CSSA(f:integer;Ext:GroupAdrs;Var int:integer;
                       var q:boolean);
  Procedure CCCZ(ext: GroupAdrs);
  Procedure CCCC(ext: GroupAdrs);
  Procedure CCCI(ext: GroupAdrs; l: boolean);
  Procedure CTCI(ext: GroupAdrs; var l: boolean);
  Procedure CCCD(ext: GroupAdrs; l: boolean);
  Procedure CTCD(ext: GroupAdrs; var l: boolean);
  Procedure CTGL(ext: GroupAdrs; var l: boolean);
  Procedure CDLAM(b,c,n,m: integer;var inta: integer;
                        var lam: lamAdrs);
  Procedure CCLM(lam: lamAdrs; l: boolean);
  Procedure CCLC(lam: lamAdrs; var l: boolean);
```

```
    Procedure CTLM(lam: lamAdrs; var l: boolean);
    Procedure CLLNK(LAM: lamAdrs; ProcedureEntry: pointer);
    procedure CFMAD(f: integer;ext: GroupAdrs; var intc:buffer;
                              var cb:conadrs);
    procedure CFUBC(f: integer;ext: GroupAdrs; var intc:buffer;
                              var cb:conadrs);
    procedure CFUBR(f: integer;ext: GroupAdrs; var intc:buffer;
                              var cb:conadrs);
    procedure CFGA(f: integer;ext: GroupAdrs; var intc:buffer;
                              var cb:conadrs);

implementation
   type
       LAMArray    = Array[1..24] of LAMpointer;
       buffrecord = record
                              addlo:longint;
                              addhi:longint;
                              page :longint;
                        end;
   Var
         StationLAMs: LAMArray;
         inhibit:boolean;
         dma,lam:boolean;
         stn:integer;
         inter:integer;
                x:char;
         ii:integer;
         d:integer;
         hport:longint;
   label rr,rrr;

   Procedure CDREG(b,c,n,a: integer; Var Ext: GroupAdrs);
               (* CAMAC register declaration*)
     Begin
       With Ext do
         begin
           Branch := b ; Crate  := c;
           Station := n;   Adrs := a;
         end
     End;

   Procedure CFSA(f: integer;ext: GroupAdrs; var int:LongData;
                          var q:boolean);
            (* CAMAC single action execute *)
     Begin
       Port[hport]:= Ext.Station-1;
       Port[hport+1]:= Ext.Adrs;
       Port[hport+2]:= f;
       Case f div 08 of
         (* read function *)     0: Begin
                                                      q:= (Port[hport+4] and 0002)  = 0;
                                                      int[1]:= Port[hport+8];
                                                      int[2]:= Port[hport+9];
                                                      int[3]:= Port[hport+10];
                                                    End;
        (* control function *) 1,3: q:= (Port[hport+4] and 0002) =0;
        (* write function *)     2: Begin
                                                      Port[hport+12]:= int[1];
                                                      Port[hport+13]:= int[2];
                                                      Port[hport+14]:= int[3];
                                                      q:=(Port[hport+4] AND $0002)  = 0;
```

```
                                                  End;
        End
      End;

    Procedure CSSA(f:integer;Ext:GroupAdrs;Var int:integer;
                         var q:boolean);
        (* CAMAC single action execute with short word *)
      Begin
        Port[hport]:= Ext.Station-1;
        Port[hport+1]:= Ext.Adrs;
        Port[hport+2]:= f;
        Case f div 8 of
           (* read function *)  0: Begin
                                               q := (Port[hport+4] and 0002)=0;
                                             int :=Port[hport+9] shl 8+Port[hport+8];
                                                         End;
          (* control function *) 1,3:
                                                     q := (Port[hport+4] and 0002)=0;
          (*write function *)     2: Begin
                                                  Port[hport+14]:= 0;
                                                  Port[hport+13]:= int shr 8;
                                                  Port[hport+12]:= int and 255;
                                                  q:=(Port[hport+4] and 0002)=0 ;
                                                    End;
        End;
      End;

    Procedure CCCZ(ext: GroupAdrs);
        (* generate dataway initialize *)
      var
        dummy: integer;
      Begin
        Port[hport+6]:= 05;
        inhibit := true;
            {dummy := Port[hport+4];}
      End;

    Procedure CCCC(ext: GroupAdrs);
        (* generate dataway clear *)
      var
        dummy : integer;
      Begin
        If inhibit = true then Port[hport+6]:= 06
                                  else Port[hport+6]:= 02;
            {dummy:= Port[hport+4];}
      End;

  Procedure CCCI(ext: GroupAdrs; l: boolean);
      (* set/reset Inhibit *)
    Begin
      If l = true then (* set I*) Port[hport+6]:= 04
                  else (* reset I*) Port[hport+6]:= 00;
      inhibit:= l
    End;

  Procedure CTCI(ext: GroupAdrs; var l: boolean);
      (* test Inhibit *)
    Begin
      l:= inhibit;
    End;
```

```
Procedure CCCD(ext: GroupAdrs; l: boolean);
     (* enable/disable crate demand *)
  Begin
    If l = true then(*enable demand*)
                                  port[33]:=port[33] and $fb
                     else(*disable demand*)
                     port[33]:= port[33] or $04;
  End;

Procedure CTCD(ext: GroupAdrs; var l: boolean);
       (* test crate demand enabled/disabled *)
  Begin
    If Port[$21] and 04(*lam mask*) = 0
        then l:= true   (*demand enabled*)
        else l:=false;(*demand disabled*)
  End;

Procedure CTGL(ext: GroupAdrs; var l: boolean);
      (* test crate demand *)
  Begin
    If Port[hport+11] and $1F = $1f
                    then   (*no demand*)  l:= false
                    else(*demand present*)  l:= true ;
  End;

Procedure CDLAM(b,c,n,m: integer;var inta: integer;
                              var lam: lamAdrs);
         (* declare lam *)
var bitposn:integer;
  Begin
    With lam do
      Begin
        branch := b;
        crate  := c;
        station:= n;
        adrs   := m;
        case m<0 of
             true:  begin
                              Bitposn  := not(m);
                              data24[1]:= 1 shl (bitposn);
                              data24[2]:= 1 shl (bitposn-8);
                              data24[3]:= 1 shl (BitPosn-16);
                            end;
             false: begin   end;
           end; {of case}
        nextlam:=nil;
      End;
  End;

Procedure CCLM(lam: lamAdrs; l: boolean);
       (* enable/disable lam *)
  var
    module  : groupAdrs;
    status :boolean;
    fnctn:integer;
    data24:longdata;
  Begin
    Module.branch:= lam.branch;
    Module.crate:= lam.crate;
    Module.station:= lam.station;
    Module.adrs:= lam.adrs;
```

```
   If lam.adrs < 0 then begin
                                         If l = true then fnctn:= 23
                                                       else fnctn:= 19;
                                         Module.adrs:= 13;
                                         Data24[1] := lam.data24[1];
                                         Data24[2] := lam.data24[2];
                                         Data24[3] := lam.data24[3];
                                       End
                                else If l = true then fnctn:= 26
                                                    else fnctn:= 24;
   CFSA(fnctn,module,data24,status);
  End;

Procedure CCLC(lam: lamAdrs; var l: boolean);
     (* clear lam *)
  Var
   Module : GroupAdrs;
   status :boolean;
   fnctn:integer;
   data24:longdata;
  Begin
   Module.branch:= lam.branch;
   Module.crate:= lam.crate;
   Module.station:= lam.station;
   Module.adrs:= lam.adrs;
   If lam.adrs < 0 then Begin
                                         fnctn    := 23;
                                         Module.adrs:= 12;
                                         Data24[1] := lam.data24[1];
                                         Data24[2] := lam.data24[2];
                                         Data24[3] := lam.data24[3];
                                        End
                                 else fnctn:= 10;
   CFSA(fnctn,module,data24,status);
  End;

Procedure CTLM(lam: lamAdrs; var l: boolean);
     (* test specified lam *)
  var
   Module: GroupAdrs;
   lamadrs1,data24: longdata;
   fnctn:integer;
   status :boolean;
  Begin
   Module.branch:= lam.branch;
   Module.crate:= lam.crate;
   Module.station:= lam.station;
   Module.adrs:= lam.adrs;
   If lam.adrs < 0 then Begin
                                       Data24[1] := lam.data24[1];
                                       Data24[2] := lam.data24[2];
                                       Data24[3] := lam.data24[3];
                                       fnctn    := 1;
                                       Module.adrs:= 12;
                                       CFSA(fnctn,module,lamadrs1,status);
                                                 l:= (lamadrs1[1] and data24[1])
                                                   or(lamadrs1[2] and data24[2])
                                                   or(lamadrs1[3] and data24[3])<>0;
                                     End
                                else Begin
                                       fnctn    := 8;
```

```
                                              CFSA(fnctn,module,data24,status);
                                                           l:= status;
                                            End
  End;

 Procedure CLLNK(LAM: lamAdrs; ProcedureEntry: pointer);
  var findinsert:lampointer;
  Begin
   lam.Service:= ProcedureEntry;
   if stationlams[lam.adrs] = nil
       then     stationlams[lam.adrs] := @lam
       else
              begin
                 findinsert:= StationLAMs[lam.adrs];
                 while (findinsert^.nextlam  <> nil)
              do findinsert:= findinsert^.nextlam;
                 findinsert^.nextlam:= @lam;
              end;
end;

Procedure IntrProcess;interrupt;
label
    next;
var
    findinsert:lampointer;
    stn:integer;
    l:boolean;
    testlam:lamadrs;
    route:pointer;
    camword:word;
  Begin
    Inline($fa);
    next:
       camword:=port[hport+3];
     If ((camword and 1)<>0) then
           begin
                port[hport+5]:=13; { reset pending interrupt}
                inter:=1 ;
           end;
     {if ((camword and 4)<>0) then
                           begin
                              writeln(camword);
                              sound(1024);
                              delay(1000);
                              nosound;
                              stn:= (Port[hport+11] and $1f)+1;
                              if (stn = $1f) then
                                  begin
                                      if stationLAMs[stn] = nil then
                                      else
                                          begin
                                              findinsert:= stationLAMs[stn];
                                              repeat
                                                  TestLAM.adrs:= findinsert^.adrs;
                                                  TestLAM.station:= stn;
                                                  route:=findinsert^.service;
                                                  CTLM(TestLAm,l);
                                                  If l Then
                                                      begin
                                                         inline($9c/$3e/$ff/$1e/route);
                                                      end
```

```
                                        else
                                            findinsert:=findinsert^.NextLAM;
                                    until (l or (findinsert = nil));
                                end;
                            goto next;
                        end;
                    end;}


        Inline($fb);
        port[$20]:=$20;
      end;

  procedure dma_init(buffrec:buffrecord;countlo,counthi:longint;wr:integer);
  begin
        if dma then begin
        port[10]:=7;
        port[11]:=71+(4*wr);  { channel 3 ,single ,read}
        port[12]:=0;
        port[6]:=buffrec.addlo;
        port[6]:=buffrec.addhi;
        port[12]:=0;
        port[7]:=countlo; {count}
        port[7]:=counthi;
        port[130]:=buffrec.page;
        port[10]:=3;
        end
        else begin
        port[10]:=5;
        port[11]:=69+(4*wr);  {******check channel 1 ,single ,read}
        port[12]:=0;
        port[2]:=buffrec.addlo;
        port[2]:=buffrec.addhi;
        port[12]:=0;
        port[3]:=countlo; {count}
        port[3]:=counthi;
        port[131]:=buffrec.page;
        port[10]:=1;
        end;
  end;

  procedure convert_count(size:longint;
                                   var countlo,counthi:longint;wr:integer);
   begin
        counthi:= (size-(wr*3)-((1-wr)*1)) div 256;
        countlo:= (size-(wr*3)-((1-wr)*1)) mod 256;
   end;

  procedure convert_buffer(var buff:buffer;
                                    var buffrec:buffrecord;wr:integer);
  var
       k:longint;
       offset,segment:longint;
       add,dadd, k64,k8,k64_8: longint;
  begin
          k64:= 256*256;
          k8 := 256*8;
          k64_8 := ((k64 - k8) div 256);
        offset := ofs(buff)+(wr*3);
        segment := seg(buff);
        add := (segment*16) + offset ;
        dadd:= add mod k64;
        buffrec.addhi:= dadd div 256;
        buffrec.addlo:= dadd mod 256;
        buffrec.page:= add div k64;
end;
procedure invert_count(var cc:longint;wr:integer);
var counthi,countlo:longint;
        comp,i:integer;
        ana:integer;
begin
    comp:=0;

        if dma then begin
        port[10]:=7;
        port[12]:=0;
        countlo:=port[7]; {count}
        counthi:=port[7];
        {if((countlo <> 255) and ( counthi<> 255)) then
        begin
            for i:=1 to 10000 do begin  end;
            writeln('ERROR early reset:wr:',wr);
            writeln('lo:',countlo,'  hi:',counthi);
        end;}
        ana := port[8];
        port[7]:=0;
        port[7]:=0;{ dma channel loaded with zero count for safety}
        {port[10]:=3;}
        end
        else
        begin
        port[10]:=5;
        port[12]:=0;
        countlo:=port[3]; {count}
        counthi:=port[3];
        port[3]:=0;
        port[3]:=0;{ dma channel loaded with zero count for safety}
        {port[10]:=1;}
        end;

        {if ((counthi=$ff) and (countlo=$ff))
                then begin
                                counthi:=0;
                                countlo:=0;
                                if (wr=0) then comp:=1;
                        end;}
        cc:=(counthi*256)+countlo - comp+(1*(1-wr));
end;

procedure get_control_word(mo:integer;bo:boolean;
                                        fu:integer;var wr:integer;var cw:byte);
begin
   wr:=0;
  case mo of
          1:cw:=1;
          2:cw:=2;
          3:cw:=4;
          4:cw:=8;
  end;{of case}
  if (bo) then cw:=cw+16;
  if ((fu div 16)>0) then begin cw:=cw+64;wr:=1;end;
  if (((fu mod 16) div 8)>0) then cw:=cw+32;
```

```
end;

procedure block_camac(f:integer;ext:groupadrs;cw,wr:integer;
                       var     cb:conadrs;var      intc:buffer;var q,n_23:integer);
var
   countlo,counthi:longint;
   camword:byte;
   buffrec:buffrecord;
   i,j:integer;
   store:array[1..100] of byte;
 begin
      port[hport+5]:=15; { reset pending interrupt}
      port[hport+7]:=cw; { stop on t/c}
      port[hport+2]:=f;{ function}
      port[hport+1]:=ext.adrs;{ sub address}
      port[hport]:=ext.station-1;{station no.}
      convert_buffer(intc,buffrec,wr);
      cb.repcount:=cb.repcount * 3;
      convert_count(cb.repcount,countlo,counthi,wr);
      dma_init(buffrec,countlo,counthi,wr);
      camword:=port[hport+15];
        repeat
           {port[10]:=7;}
           camword:=port[hport + 3 ];
           {port[10]:=3;}
         until (((camword and 1 )<>0) ) ;

         {repeat
         until(inter=1);
         inter:=0;}

         {j:=1;
         repeat
begin
port[10]:=7;
camword:= port[8];
     port[10]:=3;
j:= j+1;
end
         until( ((camword and 8)<> 0) );}

      if( (camword AND $0008)= 0) then n_23:=0 else n_23:=0;
      invert_count(cb.tally,wr);

      cb.tally:=cb.repcount-cb.tally-(wr*q*3);
      cb.repcount:=cb.repcount div 3;
      cb.tally:=cb.tally div 3;
 end;

 procedure block_camac1(f:integer;ext:groupadrs;cw,wr:integer;
                       var     cb:conadrs;var      intc:buffer;var q,n_23:integer);
var
   countlo,counthi:longint;
   camword:byte;
   buffrec:buffrecord;
   store: array[1..100] of byte;
   i,j:integer;
 begin
      port[hport+5]:=15; { reset pending interrupt}
      port[hport+7]:=cw; { stop on t/c}
      port[hport+2]:=f;{ function}
```

```
        port[hport+1]:=ext.adrs;{ sub address}
        port[hport]:=ext.station-1;{station no.}
        convert_buffer(intc,buffrec,wr);
        cb.repcount:=cb.repcount * 3;
        convert_count(cb.repcount,countlo,counthi,wr);
        dma_init(buffrec,countlo,counthi,wr);
        camword:=port[hport+15];
          repeat
             {port[10]:=7;}
             camword:=port[hport + 3];
             {port[10]:=3;}
           until (((camword and 1 )<>0) ) ;

          {repeat
           until(inter=1);
           inter:=0;}

           {j:=1;
           repeat
begin
port[10]:=7;
camword:= port[8];
port[10]:=3;
j:= j+1;
end
           until(( (camword and 8)<> 0)  );}

       if( (camword AND $0002) = 0) then q:=0 else q:=1;
       if( (camword AND $0008)= 0) then n_23:=0 else n_23:=0;
       invert_count(cb.tally,wr);

       cb.tally:=cb.repcount-cb.tally-(wr*q*3);
       cb.repcount:=cb.repcount div  3;
       cb.tally:=cb.tally div 3;
  end;

procedure CFMAD(f: integer;ext: GroupAdrs; var intc:buffer;
                              var cb:conadrs);
{procedure for address scan}
var wrt,q,n_23:integer;
       controlw:byte;
begin
       {1= address scan / 2= Q stop / 3= t\c / 4= Q repeat}
       get_control_word(1,false,f,wrt,controlw);
       if (wrt=1) then begin
                                port[hport+12]:=intc[1,1];
                                port[hport+13]:=intc[1,2];
                                port[hport+14]:=intc[1,3];
                           end;
       block_camac(f,ext,controlw,wrt,cb,intc,q,n_23);
       {only for address scan}
       if ((q=0) and (wrt=0) and (n_23=1)) then
                  begin
                      intc[cb.tally+1,1]:= Port[hport+8];
                      intc[cb.tally+1,2]:= Port[hport+9];
                      intc[cb.tally+1,3]:= Port[hport+10];
                      cb.tally:=cb.tally+1;
                  end;
end;

procedure CFUBC(f: integer;ext: GroupAdrs; var intc:buffer;
                         var cb:conadrs);
{procedure for q stop}
var wrt,q,n_23:integer;
      controlw:byte;
begin
      {1= address scan / 2= Q stop / 3= t\c / 4= Q repeat}
      get_control_word(2,false,f,wrt,controlw);
      if (wrt=1) then begin
                                    port[hport+12]:=intc[1,1];
                                    port[hport+13]:=intc[1,2];
                                    port[hport+14]:=intc[1,3];
                              end;
      block_camac(f,ext,controlw,wrt,cb,intc,q,n_23);
end;

procedure CFUBR(f: integer;ext: GroupAdrs; var intc:buffer;
                         var cb:conadrs);
{procedure for q repeat}
var wrt,q,n_23:integer;
      controlw:byte;
begin
      writeln('Press a key to terminate q repeat cycle');
      {1= address scan / 2= Q stop / 3= t\c / 4= Q repeat}
      get_control_word(4,false,f,wrt,controlw);
      if (wrt=1) then begin
                                    port[hport+12]:=intc[1,1];
                                    port[hport+13]:=intc[1,2];
                                    port[hport+14]:=intc[1,3];
                              end;
      block_camac1(f,ext,controlw,wrt,cb,intc,q,n_23);
end;

procedure CFGA(f: integer;ext: GroupAdrs; var intc:buffer;
                         var cb:conadrs);
{procedure for stop on t/c}
var wrt,q,n_23:integer;
      controlw:byte;
begin
      {1= address scan / 2= Q stop / 3= t\c / 4= Q repeat}
      { false = interrupt off / true= interrupt on}
      get_control_word(3,false,f,wrt,controlw);
      if (wrt=1) then begin
                                    port[hport+12]:=intc[1,1];
                                    port[hport+13]:=intc[1,2];
                                    port[hport+14]:=intc[1,3];
                              end;
      block_camac(f,ext,controlw,wrt,cb,intc,q,n_23);
end;

{ main program }
var

   man:longint;
   i,j:longint;
   k64:longint;
   int_b:byte;
   t:array[1..12] of longint;
begin
       clrscr;
       window(1,1,80,3);
       textcolor(lightgreen);
```

```
 gotoxy(1,1);
 write('ANSI/IEEE Std 758-1979 DRIVER for CC-3 crate controller.
     version 2.0 :07/11/1990');
 gotoxy(10,2);
 write('CAMAC Group,Electronics Division,
               BARC Bombay 400085.');
 gotoxy(1,3);
 textcolor(black);
 textbackground(white);
 for ii:=1 to 79 do
 write(' ');
 textcolor(white);
 textbackground(black);
 window(1,4,80,25);
 writeln('This is version 2.0 .');
 writeln('The interrupt option is provided.');
 writeln('The buffer size is ',
                      buff_length,' x 24 bit(camac word)');
 gotoxy(25,11);
 write('press a key');
 x:=readkey;
 clrscr;
 rr:
 GOTOXY(46,2);
 write('          ');
 gotoxy(28,2);
 write('DMA CHANNEL NO.1/3:');
 x:=readkey;
 if not((ord(x)>=48) and (ord(x)<=57))
        then begin sound(1024);delay(200);nosound;goto rr;end;
 d:=ord(x)-48;
 case d of
 1:begin dma:=false;write('1');
                    gotoxy(20,4);writeln('IC 20 switches 2 & 8 close');
    end;
 3:begin dma:=true;write('3');
                    gotoxy(20,4);writeln('IC 20 switches 1 & 7 close');
    end;
 else begin sound(1024);delay(200);nosound;goto rr;end;
 end;{of case}
 write('press a key');
 x:=readkey;
 clrscr;
 rrr:
 GOTOXY(46,2);
 write('          ');
 gotoxy(28,2);
 write('INTERRUPT   NO.2/3:');
 x:=readkey;
 if not((ord(x)>=48) and (ord(x)<=57))
     then begin sound(1024);delay(200);nosound;goto rrr;end;
 d:=ord(x)-48;
 case d of
 2:begin lam:=false;write('2');
                    gotoxy(20,4);writeln('IC 17 switch 1 close');
    end;
 3:begin lam:=true;write('3');
                    gotoxy(20,4);writeln('IC 17 switch 2 close');
    end;
 else begin sound(1024);delay(200);nosound;goto rrr;end;
 end;{of case}
```

```
          write('press a key');
          x:=readkey;
          clrscr;
          gotoxy(15,2);
 write('PC ADD-IN CARD BASE ADDRESS SETTINGS');
 gotoxy(10,14);
 hport:=768;
 write('BASE ADDRESS:decimal ',hport);
 gotoxy(5,16);
 write('Use  Curser movement arrow keys to set the adress bits.');
 gotoxy(5,17);
 write('cr> to save address');
 for i := 5 to 12 do
 begin
 gotoxy((5*i)+5,12);
 write('A',(16-i));
 end;
 gotoxy(52,11);
 write('IC 4');
 textcolor(black);
 textbackground(white);
 for j := 1 to 6 do
 begin
      for i:= 5 to 12 do
        begin
           gotoxy((5*i)+5,j+3);
           write('  ');
        end;
 end;
 man:=hport;
 k64:=256*128;

 for i:= 1 to 12 do
 begin
     t[i]:=man div k64;
     man:=man mod k64;
     k64:=k64 div 2;
     gotoxy((5*i)+6,(3*t[i])+5);
     if (i>4) then if (t[i]=0) then write('0') else write('1');
 end;

 i:=5;
 gotoxy((5*i)+6,6);
 repeat
 x:=readkey;
 if(x=chr(0)) then
                  begin
                     x:=readkey;
                     if (x=chr(80)) then
                                                   begin
                                                   gotoxy((5*i)+6,8);
                                                   write('1');
                                                   gotoxy((5*i)+6,5);
                                                   write(' ');
                                                   t[i]:=1;
                                                   end;
                     if (x=chr(72)) then
                                                   begin
                                                   gotoxy((5*i)+6,5);
                                                   write('0');
                                                   gotoxy((5*i)+6,8);
```

```
                                           write(' ');
                                           t[i]:=0;
                                           end;
                    if (x=chr(77)) then
                                             begin
                                             i:=i+1;
                                             if (i>12) then i:=12;
                                             end;
                    if (x=chr(75)) then
                                             begin
                                             i:=i-1;
                                             if (i<5) then i:=5;
                                             end;
                    end;
 gotoxy((5*i)+6,6);
 until(x=chr(13));
 textcolor(white);
 textbackground(black);
 man:=0;
 k64:=256*128;
 for i:= 1 to 12 do
 begin
    man:=t[i]*k64+man;
    k64:=k64 div 2;
 end;
 hport:=man;
 gotoxy(32,14);
 write('      ');
 gotoxy(10,14);
 write('BASE ADDRESS:decimal ',hport);
 gotoxy(2,20);
 write('press a key..');
 x:=readkey;
 clrscr;

      textbackground(lightgreen);
      for ii:= 1 to 1760 do  write(' ');
      textbackground(black);
      gotoxy(28,8);
      write(' THIS IS USABLE AREA ');
      gotoxy(40,10);
      write('press a key');
      x:=readkey;
      clrscr;
      for stn:= 1 to 24 do stationlams[stn]:= nil;
      inter:=0;
      port[hport+5]:=15; { reset pending interrupt}
      int_b := port[$21];

      if lam then begin
                              port[$21] := int_b and $f7;
                              setintvec($0b,@intrprocess);
                              end
                                else
                                begin
                                port[$21] := int_b and $fb;
                                setintvec($0a,@intrprocess);
                                end;
end.
```

APPENDIX-C

The program to measure the single CAMAC cycle execution rate CC1.PAS is given here.

{listing of CC1.PAS}

```
Program performance;
uses dos,crt,camac;
var
  data:integer;
  data24:longdata;
  status, cur: boolean;
  Display,display1: GroupAdrs;
  datasp:buffer  absolute $5000:$0000 ;
  datasp1:buffer absolute $6000:$0000;
  cnt:conadrs;
  x:char;
  i,k:integer;
  cc1:integer;
  my_length:integer;
  N,A:byte;
  store:array [1..10] of word;
  hour,min,sec,sec_100 :word;
  sum:word;
  ave_sum,sec_sum:real;
  label  exit;

Begin (* main procedure *)

    my_length := buff_length;
    N := 4; A:= 1;
    CDREG(0,0,4,0,display);
    CDREG(0,0,N,A,display1);

for k:= 1 to 10 do
  begin
      CFSA(9,display1,data24,status);
      CFSA(19,display,data24,status);

      data24[1]:=255;
      data24[2]:=255;
      data24[3]:=255;
      CFSA(16,display,data24,status);

      for i:=1 to my_length do
   begin
    datasp[i][1]:=234;
    datasp[i][2]:=13;
    datasp[i][3]:=90;

    datasp1[i][1]:=85;
    datasp1[i][2]:=85;
    datasp1[i][3]:=85;
                                    end;
      for i := 1 to my_length do
        begin
            data24[1] := datasp[i][1];
            data24[2] := datasp[i][2];
            data24[3] := datasp[i][3];
            CFSA(16,display1,data24,status);
        end;

            data24[1]:=255;
            data24[2]:=255;
            data24[3]:=255;
            CFSA(16,display,data24,status);

            settime(0,0,0,0);

            for i := 1 to my_length do
              begin
                CFSA(0,display1,data24,status);
                datasp1[i][1]  := data24[1] ;
                datasp1[i][2]  := data24[2] ;
                datasp1[i][3]  := data24[3] ;
            end;
                exit:
            gettime(hour,min,sec,sec_100);
            store[k]:=(((((hour * 60 )+min)*60)+sec)*100)+sec_100;

        cur := true;
        for i := 1 to my_length do
          begin
            if ((datasp1[i][1] = datasp[i][1]) and
                  (datasp1[i][2] = datasp[i][2]) and
                  (datasp1[i][3] = datasp[i][3]) )
            then cur:=cur
            else begin cur:=false;
 cc1:=cc1+1;end;
            end;
        if cur
                then writeln('O.K.')
                else begin writeln('not O.K.:count',cc1);end;
  end; { of ten iterations}
sum := 0;
for k := 1 to 10 do
  begin
    writeln(k,'  ',store[k]);
    sum := sum + store[k];
  end;
  ave_sum:=sum /10;
  sec_sum := ave_sum/100;
  writeln('ave:',sec_sum ,' per ',buff_length,'camac words');
  writeln('per word',sec_sum/buff_length);
  writeln('data rate',buff_length/sec_sum,' camac words per sec');
end.
```

APPENDIX-D

The program to measure the block CAMAC cycle execution rate CC3.PAS is given here.

{listing of CC3.PAS}

```
Program performance;
uses dos,crt,camac;
var
  data:integer;
  data24:longdata;
  status, cur: boolean;
  Display,display1: GroupAdrs;
  datasp:buffer  absolute $5000:$0000 ;
  datasp1:buffer absolute $6000:$0000;
  cnt:conadrs;
  x:char;
  i,k:integer;
  cc1:integer;
  my_length:integer;
  N,A:byte;
  store:array [1..10] of word;
  hour,min,sec,sec_100 :word;
  sum:word;
  ave_sum,sec_sum:real;
  label  exit;

Begin (* main procedure *)

    my_length := buff_length;
    N := 4; A:= 1;
    CDREG(0,0,4,0,display);
    CDREG(0,0,N,A,display1);

for k:= 1 to 10 do
  begin
      CFSA(9,display1,data24,status);
      CFSA(19,display,data24,status);

      data24[1]:=255;
      data24[2]:=255;
      data24[3]:=255;
      CFSA(16,display,data24,status);

      for i:=1 to my_length do
  begin
   datasp[i][1]:=234;
   datasp[i][2]:=13;
   datasp[i][3]:=90;

   datasp1[i][1]:=85;
   datasp1[i][2]:=85;
   datasp1[i][3]:=85;
                                  end;
      cnt.repcount:=my_length;
      CFGA(16,display1,datasp,cnt);

            data24[1]:=255;
            data24[2]:=255;
            data24[3]:=255;
            CFSA(16,display,data24,status);
            settime(0,0,0,0);

              cnt.repcount:=my_length;
              CFGA(0,display1,datasp1,cnt);
            gettime(hour,min,sec,sec_100);
            store[k]:=(((((hour * 60 )+min)*60)+sec)*100)+sec_100;

        cur := true;
        fc-i := 1 to my_length do
          begin
            if ((datasp1[i][1] = datasp[i][1]) and
                  (datasp1[i][2] = datasp[i][2]) and
                  (datasp1[i][3] = datasp[i][3]) )
            then cur:=c'ir
            else begin cur:=false;
  cc1:=cc1+1;end;
            end;
        if cur
              then writeln('O.K.')
              else begin writeln('not O.K.:count',cc1);end;
    end; { of ten iterations}
  sum := 0;
  for k := 1 to 10 do
    begin
        writeln(k,' ',store[k]);
        sum := sum + store[k];
    end;
    ave_sum:=sum /10;
    sec_sum := ave_sum/100;
    writeln('ave:',sec_sum ,' per ',buff_length,'camac words');
    writeln('per word',sec_sum/buff_length);
    writeln('data rate',buff_length/sec_sum,' camac words per sec');
  end.
```

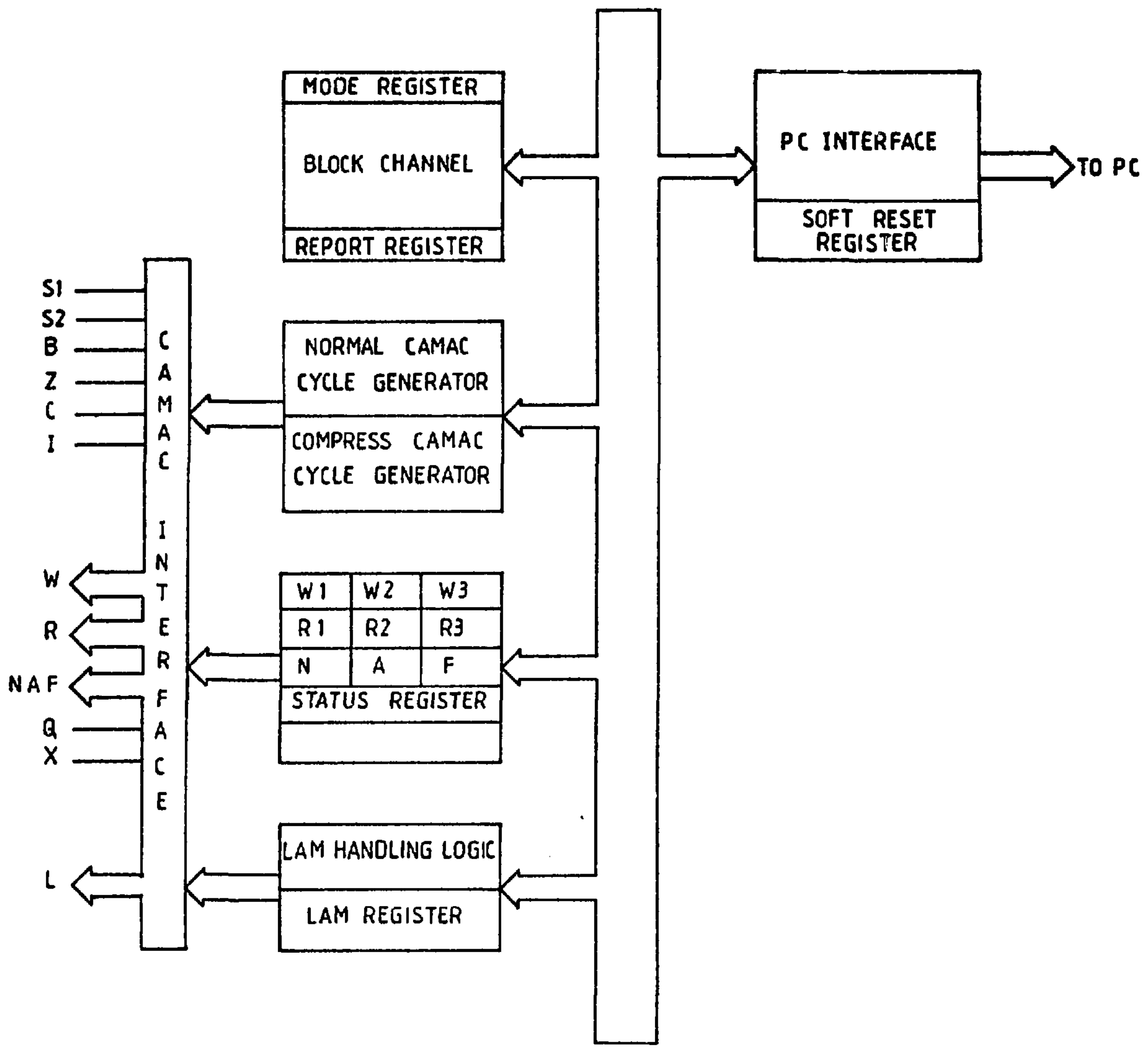
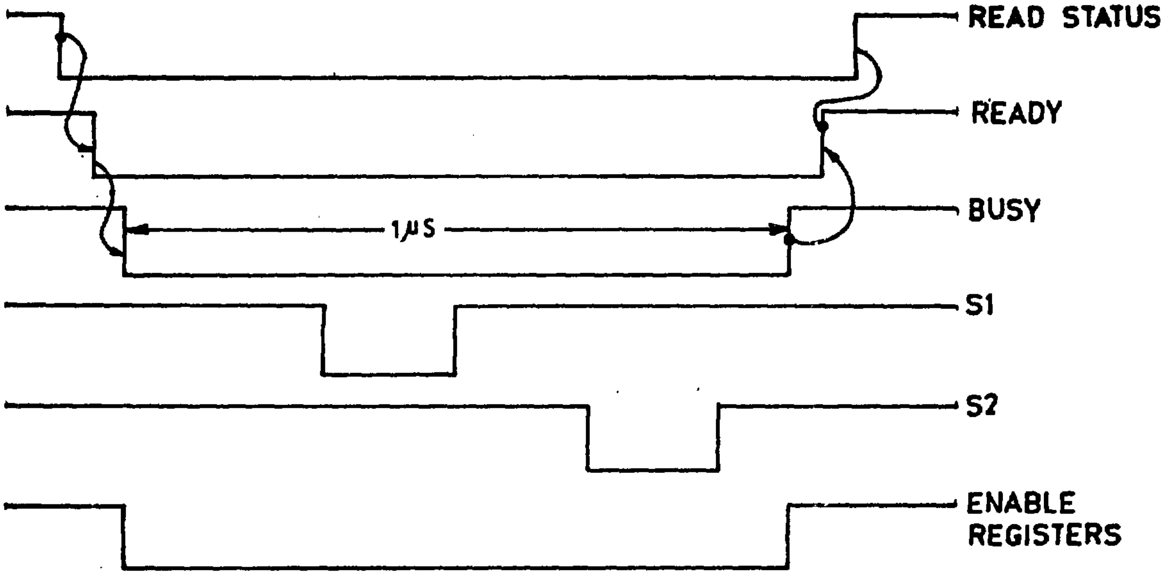
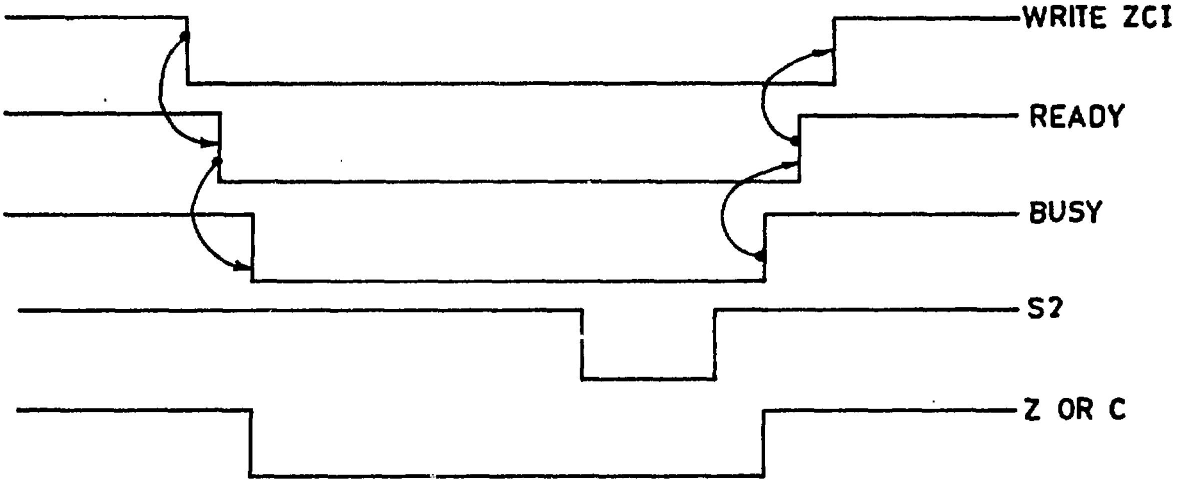


FIG.1 CC-3 CRATE CONTROLLER BLOCK DIAGRAM



SEQUENCE OF OPERATION

FIG. 2 NORMAL CAMAC CYCLE SEQUENCER



SEQUENCE OF OPERATION

FIG. 3 COMPRESSED CAMAC SEQVENCER

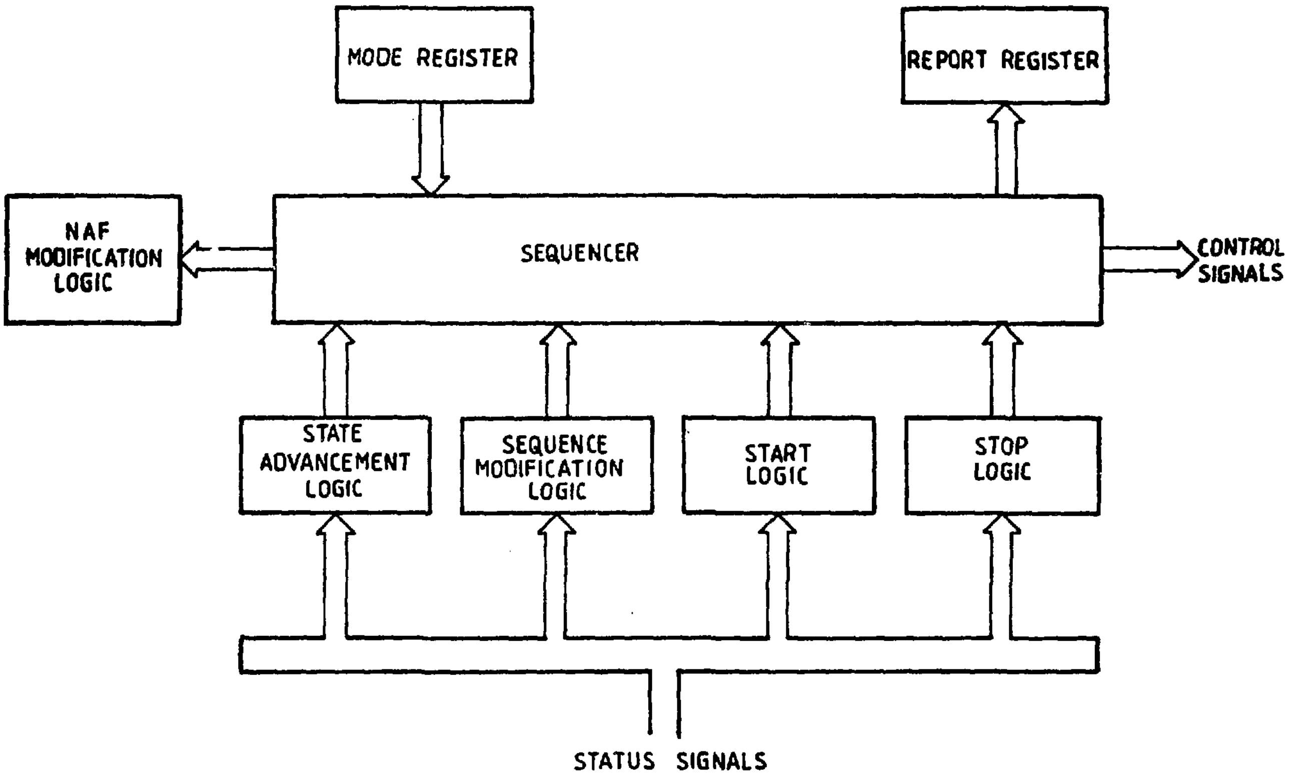


FIG.4 BLOCK TRANSFER CHANNEL– BLOCK DIAGRAM

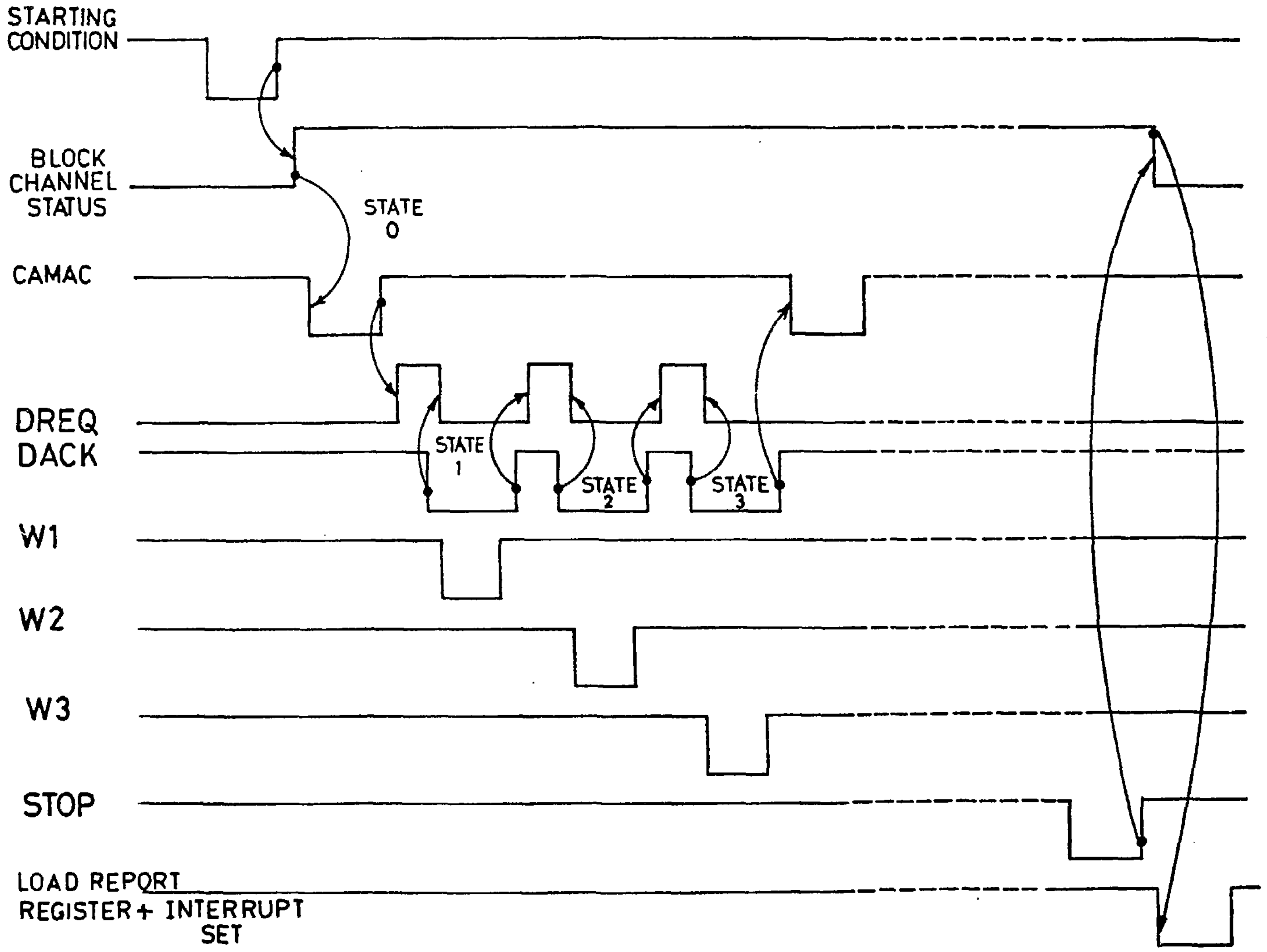


FIG. 5 BLOCK TRANSFER CHANNEL-SEQUENCE OF OPERATION

Published by : M. R. Balakrishnan Head, Library & Information Services Division
Bhabha Atomic Research Centre Bombay 400 085