OPERATOR'S MANUAL

MODEL 2366

UNIVERSAL PROGRAMMABLE LOGIC MODULE

CE

LeCroy

Innovators in Instrumentation

Corporate Headquarters
700 Chestnut Ridge Road
Chestnut Ridge, NY 10977-6499
Tel: (914) 578-6013 Fax: (914) 578-5984
E-mail: lrs_sales@lecroy.com
lrs_support@lecroy.com

European Headquarters
27 Blacklands Way
Abingdon Business Park
Abingdon Oxon OX14 1DY
United Kingdom
Tel: (1235) 533114 Fax: (1235) 528796
E-mail: lrs_europe@lecroy.com

Copyright© March 1996. LeCroy™ is a registered trademark of LeCroy Corporation. All rights reserved. Information in this publication supersedes all earlier versions.

CE CONFORMITY

CONDITIONS FOR CE CONFORMITY

Since this product is a subassembly, it is the responsibility of the end user, acting as the system integrator, to ensure that the overall system is CE compliant. This product was demonstrated to meet CE conformity using a CE compliant crate housed in an EMI/RFI shielded enclosure. It is strongly recommended that the system integrator establish these same conditions.

TABLE OF CONTENTS

1. General Information

Purpose 7
Unpacking & Inspection 7
Warranty 7
Product Assistance 7
Maintenance Agreements 7
Documentation Discrepancies 8
Software Licensing Agreement 8
Service Procedure 8

2. Operating Instruction

General 9
Specifications 10
Front Panel 12

3. Applications 15

How to Program the Model 2366 16
Software
Load 2366 18
LDGP 2366 20
EPRM 2366 22
59-Bit Input Module 24
59-bit Output Register Module 30
CAMAC Dataway Test Module 38
48-Input Majority Logic Unit 44
Long Range Multihit Time Digitizer 52

****

IMPORTANT NOTICE: THE 2366 ULM IS SHIPPED WITH BOTH INPUT AND OUTPUT LEVEL CONVERTERS INSTALLED. NO DAMAGE WILL OCCUR IF THE UNIT IS POWERED WITH THE STANDARD EPROM PROGRAM, WHICH DOES NOT USE THE FRONT PANEL I/O. HOWEVER, BEFORE ANY OTHER USER PROGRAM IS LOADED, THE UNIT MUST BE CONFIGURED WITH THE DESIRED I/O PATTERN. FOR EACH PAIR OF SOCKETS, FOR THE 10124 AND 10125, ONLY ONE SHOULD BE OCCUPIED. IN TOTAL, 17 ICs SHOULD BE REMOVED.

****

GENERAL INFORMATION

PURPOSE

This manual is intended to provide instruction regarding the setup and operation of the covered instruments. In addition, it describes the theory of operation and presents other information regarding its functioning and application.

UNPACKING AND INSPECTION

It is recommended that the shipment be thoroughly inspected immediately upon delivery. All material in the container should be checked against the enclosed Packing List and shortages reported promptly. If the shipment is damaged in any way, please notify the Customer Service Department or the local field service office. If the damage is due to mishandling during shipment, you may be requested to assist in contacting the carrier in filing a damage claim.

WARRANTY

LeCroy warrants its instrument products to operate within specifications under normal use and service for a period of one year from the date of shipment. Component products, replacement parts, and repairs are warranted for 90 days. This warranty extends only to the original purchaser. Software is thoroughly tested, but is supplied "as is" with no warranty of any kind covering detailed performance. Accessory products not manufactured by LeCroy are covered by the original equipment manufacturers' warranty only.

In exercising this warranty, LeCroy will repair or, at its option, replace any product returned to the Customer Service Department or an authorized service facility within the warranty period, provided that the warrantor's examination discloses that the product is defective due to workmanship or materials and has not been caused by misuse, neglect, accident or abnormal conditions or operations.

The purchaser is responsible for the transportation and insurance charges arising from the return of products to the servicing facility. LeCroy will return all in-warranty products with transportation prepaid.

This warranty is in lieu of all other warranties, express or implied, including but not limited to any implied warranty of merchantability, fitness, or adequacy for any particular purpose or use. LeCroy shall not be liable for any special, incidental, or consequential damages, whether in contract, or otherwise.

PRODUCT ASSISTANCE

Answers to questions concerning installation, calibration, and use of LeCroy equipment are available from the Customer Service Department, 700 Chestnut Ridge Road, Chestnut Ridge, New York, 10977-6499, (914) 578-6030.

MAINTENANCE AGREEMENTS

LeCroy offers a selection of customer support services. For example, Maintenance Agreements provide extended warranty that allows the customer to budget maintenance costs after the initial warranty has expired. Other services such as installation, training, on-site repair, and addition of engineering improvements are available through specific Supplemental Support Agreements. Please contact the Customer Service Department for more information.

DOCUMENTATION DISCREPANCIES

LeCroy is committed to providing state-of-the-art instrumentation and is continually refining and improving the performance of its products. While physical modifications can be implemented quite rapidly, the corrected documentation frequently requires more time to produce. Consequently, this manual may not agree in every detail with the accompanying product and the schematics in the Service Documentation. There may be small discrepancies in the values of components for the purposes of pulse shape, timing, offset, etc., and, occasionally, minor logic changes. Where any such inconsistencies exist, please be assured that the unit is correct and incorporates the most up-to-date circuitry.

SOFTWARE LICENSING AGREEMENT

Software products are licensed for a single machine. Under this license you may:

- Copy the software for backup or modification purposes in support of your use of the software on a single machine.
- Modify the software and/or merge it into another program for your use on a single machine.
- Transfer the software and the license to another party if the other party accepts the terms of this agreement and you relinquish all copies, whether in printed or machine readable form, including all modified or merged versions.

SERVICE PROCEDURE

Products requiring maintenance should be returned to the Customer Service Department or authorized service facility. If under warranty, LeCroy will repair or replace the product at no charge. The purchaser is only responsible for the transportation charges arising from return of the goods to the service facility. For all LeCroy products in need of repair after the warranty period, the customer must provide a Purchase Order Number before any inoperative equipment can be repaired or replaced. The customer will be billed for the parts and labor for the repair as well as for shipping. All products returned for repair should be identified by the model and serial numbers and include a description of the defect or failure, name and phone number of the user. In the case of products returned, a Return Authorization Number is required and may be obtained by contacting the Customer Service Department at (914) 578-6030.

OPERATING INSTRUCTIONS

GENERAL

The CAMAC Model 2366 is a general purpose programmable logic module,using state-of-the-art field programmable gate array technology. This CAMAC module can be used as a programmable LeCroy ECLine trigger processor module, among other uses. In addition to a full 24-bit CAMAC interface, there are 59 front panel differential ECL I/O signals, which (with some restrictions) can be independently selected to be either inputs or outputs. This module is also useful as a general purpose controller, as part of a test system or data acquisition system.

The desired logical operations are programmed in a Xilinx 4005 gate array chip. Any logic that can be implemented as a synchronous (clocked) state machine may be programmed, subject only to the limitations of the Xilinx 4005 gate array chip (approximately 5,000 equivalent gates). There are 3 clocks available on the board, 40, 20, and 10 MHz, or any of 3 special front panel inputs may be used as a clock. Input and output signals use standard 10124 and 10125 TTL-ECL level translators. Input signals as short as 5 nanoseconds can be latched and synchronized with the internal state machine logic.

The gate array must be programmed after power up, and can be reprogrammed at any time. An on-board EPROM socket may contain a program which is loaded on power up, or reloaded on CAMAC command. The Xilinx chip may also be programmed directly from the CAMAC dataway (the program information is stored in RAM in the Xilinx chip, so there is no limit to the number of times that it can be reprogrammed).

A few basic CAMAC function codes are implemented in hardware, and are available on power up. These are only used to program the Xilinx chip, and all but one of these functions disappear after a CAMAC clear operation (F9, C, or Z).

F30	A0-A15	Enter programming mode with eprom selected, enable all other hardware function codes.
F28	A0-A15	Select CAMAC programming mode.
F25	A0-A15	Program Xilinx chip (program pulse lasts until the next S1).
F16	A0-A15	Write 8 bits to the Xilinx.
F12	A0-A15	Test Xilinx READY line (not usually required).
F13	A0-A15	Test Xilinx program DONE.
F14	A0-A15	Test Xilinx INIT line.
F9	A0-A15	Disable function codes except for F30 (CAMAC C, Z have the same effect).

The Xilinx chip automatically loads itself if an EPROM is installed on the board. An F30 command followed by an F25 will cause the Xilinx chip to be reset and reloaded from the EPROM.

To load a program from CAMAC, the F30 command is followed by an F28 and F25. After the Xilinx INIT line is true (test with F14), the data is written 8 bits at a time using F16. F12 tests the Xilinx ready line before each write operation. This is not required unless the CAMAC host is capable of CAMAC operations at a rate greater than 500 kHz. This continues until all data is written and the Xilinx DONE line is true (test with F13). The Xilinx XACT software or any of several third party gate array software packages can be used to prepare the program file. An example basic program is supplied which reads the Xilinx .BIT file and uses standard ESONE CAMAC functions to load the program file into the 2366 module.

The initial program in the EPROM is T2366E (the schematic is included in the manual). This implements a simple divide chain to flash the LEDs with the 3 internal clocks (40 MHz, 20 MHz and 10 MHz), a 24 bit read write register and a register which latches the CAMAC F,A,Z,I,N on every S2. This last register is read by F0, A1. This emulates most of the test portion of the LeCroy 2050 CAMAC dataway display and test module.

A simple basic program which exercises the 2366 is EP2366.bas. It starts by forcing the reload of the Xilinx chip from the eprom (with T2366E), and implements a simple CAMAC system test.

The example program LOAD2366.bas will load an arbitrary xxxx.bit file into the Xilinx chip.

SPECIFICATIONS

Single width CAMAC module

1 LED indicates N line activity

2 programmable LEDs

59 I/O signals on front panel, all are differential ECL

All front panel I/Os to and from the Xilinx chip

Input or output is selectable in groups of 4. Inputs are terminated with 112 ohms, outputs will drive 100 ohm lines.

Programmable gate array: Xilinx 4005-5, 196 configurable logic blocks and 112 I/O blocks fast carry logic, wide decoding approximately 5000 gates: 616 flip-flops, 6272 Ram bits. This is in a 156-pin PGA socket.

The 4005-5 can be replaced with a 4005-4 or 4005-3, by the user, as they become available from Xilinx. The socket is also pin compatible with the Xilinx 4006.

40 MHz, 20 MHz, and 10 MHz crystal clocks on the board

Programmable by optional on board (socketed) Eprom on power up, or by CAMAC command to reload from Eprom.

Programmable via CAMAC, at any time, independent of Eprom. Xilinx XACT software system is required for programming. Uses Xilinx .BIT file for programming information.

Basic CAMAC programming software for IBM compatible included. 11875 CAMAC F16 write operations are required to program all logic must be clocked (synchronous logic).

CAMAC Interface: Programming, 8 bit write only interface. Test for successful programming. After programming, all CAMAC control and data lines (N, F, A, 24R/W, C, Z, S1, S2, Q, X, L) are available to Xilinx chip. Only 1 function code is reserved for reprogramming, all others are available for the user.

Possible Applications: Trigger logic, digital 48 input majority logic; trigger or readout controller; pipelined sequential logic; pulse sequence generator; any arbitrary state machine logic; fast memory, FIFO or LIFO, 16 bit digital adder.

CAMAC Interface Pin Assignments for Xilinx 4005-PGA156

CAMAC NAME	XILINX PIN		CAMAC NAME	XILINX PIN
C	R10		the 24 bit read-write bus	
Z	T9			
I	E14		RW1	T16
N	C5		RW2	T14
S1	T7		RW3	T10
S2	A2		RW4	R9
A1	B4		RW5	T8
A2	A3		RW6	P7
A4	B5		RW7	T3
A8	B6		RW8	P4
F1	A5		RW9	R1
F2	C7		RW10	P2
F4	B7		RW11	P1
F8	A6		RW12	N1
F16	A7		RW13	K3
			RW14	K2
X	G15		RW15	J2
Q	G16		RW16	J3
L	H16		RW17	H1
			RW18	G1
			RW19	F1
enables for bidirectional			RW20	F2
24 bit read write bus			RW21	E3
Re*	T11	r1-r24	RW22	C1
Wr1*	R11	w1-w8	RW23	B1
Wr2*	A8	w9-w24	RW24	A1

Direction for R1-R24, normally +5 V, pin T1

40 MHz crystal clock	pin B3 (PGCK1)
divide by 2 (20 MHz)	pin B16 (PGCK2)
divide by 4 (10 MHz)	pin T15 (PGCK3)

FRONT PANEL

Programmable LEDs, top = D14 (red), bottom = C16 (yellow)

Front Panel Input-Output pins:

PAIR NUMBER	A (8)	B (17)	C (17)	D (17)	connector (pairs)
1	F15	R6	B2	J15	
2	E16	T4	C9	J16	
3	F16	R4	B9	K14	
4	G14	R3	A9	K15	
5	P9	N2	C10	K16	
6	T6	M3	B10	L15	
7	R7	L2	A10	L16	
8	T5	L1	B11	M14	
9	K1	A11	M16		
10	J1	C12	N14		
11	G2	B12	N15		
12	G3	B13	P12		
13	E1	A13	P15		
14	E2	A14	P16		
15	C2	C15	R13		
16	D3	D15	T13		
17	R16	B14	T2		

The input (and output) polarity is such that a positive ECL edge (the odd numbered pin on the input connector is a positive-going edge) produces a negative TTL edge at the Xilinx input. This gives the best possible performance for capturing short input signals (as short as 5 nsec). The input signal should be inverted inside the Xilinx chip and used as the clock to a D Flip Flop (with a logic 1 as the D input). This does mean that a normal ECL logic 1 (odd pin +) becomes a logic 0 inside the Xilinx chip. Input and output are consistent, both are inverted. We suggest as a general rule to invert the data inside the Xilinx chip as it enters and leaves. This allows normal positive logic to be used inside the Xilinx chip. These extra inverters are not really extra, they are simply absorbed inside the logic and do not usually cause any extra propagation delay.

All front panel I/O is selectable as input or output in groups of 4, as indicated below (the 3 clocks inputs are separately selectable). This is accomplished by installing the socketed ECL-TTL or TTL-ECL level converters and the appropriate termination resistor networks. All inputs and outputs are differential ECL. The corresponding Xilinx pins must be programmed as either input or output.

For input only the 10125 ECL to TTL converters and the 56 ohm termination resistor SIPs are installed. The inputs are properly terminated for twisted pair cable.

For output, only the 10124 TTL to ECL converters and the 390 ohm pull-down resistor SIPs are installed.

Note: The module will work properly with short cables (but with reduced noise immunity) even if both the input termination SIPs and the output pull-down SIPs are installed.

The logical polarity of any signal is programmable in the Xilinx chip of course. There are a total of 59 I/O signals.

1 16 pin (8 pairs) header:

I/O A 1-4
I/O A 5-8

3 34 pin (17 pairs) headers:

I/O B 1-4
I/O B 5-8
I/O B 9-12
I/O B 13-16
I/O B 17 (can be connected to SGCK1)

I/O C 1-4
I/O C 5-8
I/O C 9-12
I/O C 13-16
I/O C 17 (can be connected to SGCK2)

I/O D 1-4
I/O D 5-8
I/O D 9-12
I/O D 13-16
I/O D 17 (can be connected to SGCK3)

SGCK1, 2 and 3 are Xilinx internal clock distribution networks.

It is possible to convert the front panel I/O to bidirectional TTL. This user modification cannot be wholeheartedly recommended, however, since little protection is provided for the Xilinx chip. This method will only work at low speeds (less than 1 MHz) and for very short cable lengths. Deglitching will be needed at all receivers. The Xilinx pins cannot drive 100 ohm terminated lines. For best results we recommend using the terminated ECL for the cable runs, and converting to TTL at the destination circuit board. This is necessary if high speed performance is required.

To provide TTL connections construct this 16 pin dip header with 4 100 ohm resistors, for each group of 4 to be used as TTL.

Install this header in the 10124 location after removing both resistor packs, and both the 10124 and 10125 chips.

This user modification connects the Xilinx pin directly to the odd numbered pin on the front panel connector. The corresponding even numbered pin is grounded. There is no buffer, ONLY a series resistor to prevent damage to the Xilinx chip.

The Xilinx chip can be programmed for either INPUT, OUTPUT or as a TRISTATE pin.

Please Use With Care!

Pin

1	connect to 16
2	connect to 16
3	100 ohm resistor to 7
4	100 ohm resistor to 5
5	100 ohm resistor to 4
6	
7	100 ohm resistor to 3
8	
9	
10	100 ohm resistor to 12
11	100 ohm resistor to 13
12	100 ohm resistor to 10
13	100 ohm resistor to 11
14	connect to 16
15	connect to 16
16	connect to 1,2,14,15 (ground)

Note: Only 4 100 ohm resistors are required.

APPLICATIONS

INTRODUCTION

The following section includes examples of programming the Model 2366 and examples of software to load programs into the 2366 from CAMAC. These examples have all been tested and work as described in the specifications. They are offered as-is, as examples only. They may be freely used, but they are not supported products.

These examples, and others, are available for downloading from the LeCroy ftp site (currently www.lecroy.com).

HOW TO PROGRAM THE LECROY MODEL 2366 UNIVERSAL LOGIC MODULE

There are 4 steps to programming the LeCroy Model 2366 Universal Logic Module. These four steps are:

Step	Action
1	Create Schematic†.
2	Convert from CAD/CAM drawing to XILINX formatted netlist using XILINX programs ... generates an ASCII output file: a) Convert to self contained netlist b) perform format conversion to XILINX compatible format.
3	Copy ASCII file to PC and compile it with XILINX programs to generate a binary file capable of being loaded into the XILINX chip.
4	Write binary file from PC into model 2366 CAMAC module and XILINX chip.

† *Other techniques for this step exist, such as VHDL or Verileg™.*

Step 1: Create Schematic

The very first step in programming the LeCroy Model 2366 Universal Logic Module is to produce a schematic using the XILINX parts software library *. The 2366 is based on the part identified in the XILINX library as part number 4005PLG156 which is described as a gate array. The pin assignments are described in the manual to the Model 2366 and correspond to the description given in the XILINX library. The XILINX library also includes many varieties of gates, flip-flops, counters, registers and so forth from which you are able to build up your design **.

In the following example, the file named test3377 represents the schematic design file which must be converted into a format compatible with the 2366 CAMAC module.

Step 2: Convert from CAD/CAM to XILINX format netlist using XILINX programs

a) Convert to self contained netlist { lca_expand command below }
b) Convert to XILINX compatible format { erel2xnf command below }

Here are the commands used to perform these operations:

lca_expand *test3377*
erel2xnf *test3377 -p 4005pgl56-5*

The file output from the last command is called *test3377*.xnf and is in ASCII format. If you look at this file you would see something like the following:

```
LCANET,4
PROG,EREL2XNF,4.10,Created from //node_3ec48/disk_scsi5/local_user2/richard_b/test3377
01/25/1995 10:18
PART,4005pg156-5
SYM,I$5122/I$14,DFF,INIT=R
```

* We use a Mentor Graphics system, however, almost any of the popular CAD/CAM programs are capable of producing a compatibly formatted design file.
** Several examples of schematics are included in the Model 2366 user's manual.

```
PIN,C,I,CLOCK20
PIN,CE,I,N$5581
        :
        :
A large section of similar stuff here...this particular example was 83,064 bytes long.
        :
        :
EXT,P4-8,O,,LOC=M16
EXT,P4-9,O,,LOC=N14
PWR,0,GROUND
PWR,1,VCC
EOF
```

Step 3: Copy the *.XNF file to a PC (or compatible computer) and compile it with a XILINX program which generates a binary image file

The following command file (with the *.MAK extension) is used on the PC to convert the *.XNF file into a binary file with the extension *.BIN. This example uses the XILINX software version 5. Note that the lines beginning with a "#" are comment lines inserted to explain the contents of the *.MAK file.

```
########### Start of MAK file ######################
# test3377.mak PC MAKE file for XILINX builds

# Environment inherited macros, usually directories.
DESIGN_DIR=c:\xact\designs\test3377\

# Miscellaneous files...*.XNF is input, others are outputs
SCHEM_FILE = $(DESIGN_DIR)test3377.xnf
MERGED_FILE = $(DESIGN_DIR)test3377.xff
PREP_FILE = $(DESIGN_DIR)test3377.xtf
PPR_PARAFILE = $(DESIGN_DIR)test3377.ppr

# Compile and command macros
cmd_merge = xnfmerge $(SCHEM_FILE) $(MERGED_FILE)
cmd_prep = xnfprep $(MERGED_FILE)
cmd_ppr = ppr $(PREP_FILE) paramfile = $(PPR_PARAMFILE)
cmd_makebits = makebits test3377.lca

# Specify ultimate target of MAK.
all: $(DESIGN_DIR)test3377.bit

# Build the merged design.
$(MERGED_FILE):     $(SCHEM_FILE)
                    $(cmd_merge)

# Build final whatever.
$(DESIGN_DIR)test3377.bit:      $(MERGED_FILE)
                                $(cmd_prep)
                                $(cmd_ppr)
                                $(cmd_makebits)

########### End of MAK file ######################
```

Step 4: Write the binary image file directly into the LeCroy Model 2366

At this point there is a file named *test3377*.bit which can be written directly into the LeCroy Model 2366 Universal Logic Module via standard CAMAC commands.

```
DECLARE FUNCTION init% (file$, var$, value!)
'
'       LOAD2366.BAS
'
' this program is written for Microsoft QuickBasic
'
' NOTE!
' a single apostrophe indicates that the rest of the line is a comment
' a variable name ending in % is a 16 bit signed integer
' a variable name ending in & is a 32 bit signed integer
' a variable name ending in $ is a string
'

        PRINT
        fff$ = "load2366.ini"

        jj$ = "camslot%"
        isok% = init%(fff$, jj$, value)
        IF isok% = 0 THEN
            INPUT "Where is the 2366?  Camac slot #"; value
        END IF
        modl% = value

        ON ERROR GOTO fault
        IF COMMAND$ = "" THEN
        INPUT "Please enter the file name"; fil$
        ELSE
        fil$ = COMMAND$
        END IF

start:

        crate% = 1
        CALLS cdreg(tmod0%, 0, crate%, modl%, 0)
        CALLS cdreg(tmod1%, 0, crate%, modl%, 1)
        CALLS cdreg(mtslot%, 0, crate%, 2, 0)

        CALLS ccinit(0)
        CALLS cssa(9, tmod0%, dd%, qq%)

        PRINT : PRINT "RELOAD VIA CAMAC"
        OPEN fil$ FOR BINARY ACCESS READ AS #1
        jk$ = ""‘1 byte long

        CALLS ccinit(0)
        ' enter programming mode
        CALLS cssa(30, tmod0%, dd%, qq%)
        ' select camac mode
        CALLS cssa(28, tmod0%, dd, qq%)
        ' program pulse
        CALLS cssa(25, tmod0%, dd%, qq%)

        ' wait for init line
        ttt% = 0: PRINT "INIT ";
        DO
            CALLS cssa(14, tmod0%, dd%, qq%)
            PRINT qq%;
            ttt% = ttt% + 1
            IF ttt% > 10000 THEN EXIT DO
        LOOP WHILE qq% < > 1
        PRINT

        ' skip the first 16 bytes in the file
        FOR i% = 0 TO 15
            GET #1, , jk$
        NEXT

        np% = 0
DO
        GET #1, , jk$
        num% = ASC(jk$)
        IF num% = 255 THEN EXIT DO
        IF (num% > 31) AND (num% < 123) THEN
            PRINT jk$;
            np% = 0
        ELSE
            IF np% = 0 THEN PRINT "";
            np% = 1
        END IF
LOOP

        GOSUB sendit
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        num% = num% AND 15: lengt& = num%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        lengt& = lengt& * 256 + num%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        lengt& = lengt& * 256 + num%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        num% = (num% AND 240) / 16
        lengt& = lengt& * 16 + num%
        nbytes = lengt& / 8
        hide% = INT(nbytes)
        PRINT : PRINT lengt&; "(bits) "; nbytes; hide%; "bytes"

        FOR j% = 1 TO hide%
            GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
            CALLS cssa(13, tmod0%, dd%, qq%)
            IF qq% = 1 THEN PRINT "done at j%="; j%: EXIT FOR
        NEXT

        num% = ASC(jk$): GOSUB sendit ' one last write
        CLOSE 1

        PRINT "finished    ";

        CALLS cssa(13, tmod0%, dd%, qq%)
        IF qq% = 1 THEN
            PRINT "!!!success!!!"
        ELSE
            PRINT "***FAILURE***"
        END IF
        PRINT

        CALLS cssa(9, tmod0%, dd%, qq%)
        errnum% = 0
        SYSTEM

sendit:
        CALLS cssa(16, tmod0%, num%, qq%)
RETURN

fault:
        PRINT "error"
        SYSTEM

        END

        FUNCTION init% (file$, var$, value)
'
' read in an initialization file of the form:
'    varname=value
'    varname=value
' etc
'
' case is ignored
' one variable per line
' variables can be in any order
' init% returns as 0 for variables not found
'
```

```
    init% = 0
    ON LOCAL ERROR GOTO lerr
    OPEN file$ FOR INPUT AS #1
    WHILE NOT EOF(1)
      LINE INPUT #1, line$
      line$ = UCASE$(line$)
      IF (INSTR(line$, UCASE$(var$)) <> 0) AND (INSTR(line$, "=") <> 0) THEN
        jnk$ = RIGHT$(line$, LEN(line$) - INSTR(line$, "="))
        value = VAL(jnk$): init% = 1
      END IF
    WEND
lerr:
    CLOSE #1
    EXIT FUNCTION
    END FUNCTION
```

```
DECLARE FUNCTION init% (file$, var$, value!)
'
'       LDGP2366.BAS
'
'  this program is written for Microsoft QuickBasic
'
'  NOTE!
'  a single apostrophe indicates that the rest of the line is a comment
'  a variable name ending in % is a 16 bit signed integer
'  a variable name ending in & is a 32 bit signed integer
'  a variable name ending in $ is a string
'

restart:

        PRINT
        fff$ = "load2366.ini"

        jj$ = "camslot%"
        isok% = init%(fff$, jj$, value)
        IF isok% = 0 THEN
             INPUT "Where is the 2366 to load?  Camac slot #"; value
        END IF
        modl% = value

        IF COMMAND$ = "" THEN
        INPUT " Please enter the XXX.BIT file name to load"; fil$
        ELSE
        fil$ = COMMAND$
        END IF

start:

        CALLS gpinit
        CALLS gpsic
        CALLS gpron
        crate% = 1
        CALLS gpcini(crate%)
        CALLS gpcccz(crate%)

'       this is the syntax for the CAMAC subroutines
'       LeCroy 8901 and National Instruments GPIB card
'            CALLS GPCFSA(crate%, f%, a%, n%, int4&, xq%)
'            x=1, q=2, so x&q=3

        ulm% = modl%
        CALLS GPCFSA(crate%, 9, 0, modl%, dd&, qq%)
        GOSUB reload
        CALLS GPCFSA(crate%, 9, 0, modl%, dd&, qq%)

        INPUT "Load another? (y/n)"; yn$
        IF UCASE$(yn$) = "Y" THEN GOTO restart

        SYSTEM

reload:
        PRINT " RELOAD VIA CAMAC   ";
        OPEN fil$ FOR BINARY ACCESS READ AS #1
        jk$ = " " ' 1 byte long
rel2:
        ' enter programming mode
        CALLS GPCFSA(crate%, 30, 0, ulm%, dd&, qq%)
        ' select camac mode
        CALLS GPCFSA(crate%, 28, 0, ulm%, dd&, qq%)
        ' program pulse
        CALLS GPCFSA(crate%, 25, 0, ulm%, dd&, qq%)
        ' wait for init line
        ttt% = 0: PRINT " INIT ";
        CALLS GPCFSA(crate%, 14, 0, ulm%, dd&, qq%)
        IF qq% = 3 THEN GOTO rel2

        DO
        CALLS GPCFSA(crate%, 14, 0, ulm%, dd&, qq%)
        dd% = dd&
        PRINT qq%;
        ttt% = ttt% + 1
        IF ttt% > 100 THEN GOTO rel2
        LOOP WHILE qq% <> 3
        PRINT : PRINT " ";

        ' skip the first 16 bytes in the file
        FOR i% = 0 TO 15
             GET #1, , jk$
        NEXT
        np% = 0
DO
        GET #1, , jk$
        num% = ASC(jk$)
        IF num% = 255 THEN EXIT DO
        IF (num% > 31) AND (num% < 123) THEN
             PRINT jk$;
             np% = 0
        ELSE
             IF np% = 0 THEN PRINT " ";
             np% = 1
        END IF
LOOP
'       start with the FF word.....................
        GOSUB sendit
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        num% = num% AND 15: lengt& = num%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        lengt& = lengt& * 256 + num%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        lengt& = lengt& * 256 + num%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        PRINT num%;
        num% = (num% AND 240) / 16
        lengt& = lengt& * 16 + num%
        nbytes = lengt& / 8
        hide% = INT(nbytes)
        PRINT : PRINT lengt&; "(bits) "; nbytes; hide%; "bytes"

        FOR j% = 1 TO hide%
        GET #1, , jk$: num% = ASC(jk$): GOSUB sendit
        CALLS GPCFSA(crate%, 13, 0, ulm%, dd&, qq%)
        IF qq% = 3 THEN PRINT "done at j%="; j%; "  "; : EXIT FOR
        NEXT
        num% = ASC(jk$): GOSUB sendit '  one last write
        CLOSE 1

        PRINT "finished    ";

        CALLS GPCFSA(crate%, 13, 0, ulm%, dd&, qq%)
        IF qq% = 3 THEN
             PRINT "!!!success!!!"
        ELSE
             PRINT "***FAILURE***"
        END IF

        CALLS GPCFSA(crate%, 9, 0, ulm%, dd&, qq%)
RETURN

sendit:
        num& = num%
        CALLS GPCFSA(crate%, 16, 0, ulm%, num&, qq%)
RETURN

        END

        FUNCTION init% (file$, var$, value)
'
' read in an initialization file of the form:
```

```
'     varname=value
'     varname=value
' etc
'
'  case is ignored
'  one variable per line
'  variables can be in any order
'  init% returns as 0 for variables not found
'
        init% = 0
        ON LOCAL ERROR GOTO lerr
        OPEN file$ FOR INPUT AS #1
        WHILE NOT EOF(1)
             LINE INPUT #1, line$
             line$ = UCASE$(line$)
             IF (INSTR(line$, UCASE$(var$)) <> 0) AND (INSTR(line$, "=") <> 0) THEN
                  jnk$ = RIGHT$(line$, LEN(line$) - INSTR(line$, "="))
                  value = VAL(jnk$): init% = 1
             END IF
        WEND
lerr:
        CLOSE #1
        EXIT FUNCTION
        END FUNCTION
```

```
DECLARE FUNCTION init% (file$, var$, value!)
'
'       EPRM2366.BAS
'
' this program is written for Microsoft QuickBasic
'
' NOTE!
' a single apostrophe indicates that the rest of the line is a comment
' a variable name ending in % is a 16 bit signed integer
' a variable name ending in & is a 32 bit signed integer
' a variable name ending in $ is a string
'

        PRINT
        fff$ = "load2366.ini"

        jj$ = "camslot%"
        isok% = init%(fff$, jj$, value)
        IF isok% = 0 THEN
             INPUT "Where is the 2366?  Camac slot #"; value
        END IF
        modl% = value

start:

        crate% = 1
        CALLS cdreg(tmod0%, 0, crate%, modl%, 0)
        CALLS cdreg(tmod1%, 0, crate%, modl%, 1)
        CALLS cdreg(mtslot%, 0, crate%, 2, 0)

        CALLS ccinit(0)
        CALLS cssa(9, tmod0%, dd%, qq%)

        ' REprogram using EPROM

        ' enter programming mode
        CALLS cssa(30, tmod0%, dd%, qq%)
        ' program pulse
        CALLS cssa(25, tmod0%, dd%, qq%)
        CALLS cssa(14, tmod0%, dd%, qq%)
        ' wait for init line
        ttt% = 0: PRINT " INIT ";
        DO
        CALLS cssa(14, tmod0%, dd%, qq%)
        PRINT qq%;
        ttt% = ttt% + 1
        IF ttt% > 1000 THEN EXIT DO
        LOOP WHILE qq% <> 1
        PRINT

        PRINT "RELOADED FROM THE EPROM    ";
        ttt = TIMER
        DO
        IF (TIMER - ttt) > 1 THEN EXIT DO
        ' test DONE
        CALLS cssa(13, tmod0%, dd%, qq%)
        IF qq% = 1 THEN EXIT DO
        LOOP

        IF qq% = 1 THEN
        PRINT "!!!success!!!"
        ELSE
             PRINT "***FAILURE***"
        END IF
        PRINT
        PRINT "enter Q to quit or R to restart"
        dwrt& = 0
        RANDOMIZE TIMER

        CALLS cssa(9, tmod0%, dd%, qq%)
        errnum% = 0

        PRINT "enter Q to quit, or R to reprogram"
        RANDOMIZE TIMER

        FOR fff% = 0 TO 31
             FOR aaa% = 0 TO 15
                  CALLS cdreg(mtslot%, 0, crate%, 2, aaa%)
                  CALLS cfsa(fff%, mtslot%, dat&, qf%)
                  CALLS cfsa(1, tmod0%, dat&, qr%)
                  df& = dat& AND (256 + 128 + 64 + 32 + 16)
                  df& = df& / 16
                  IF df& <> fff% THEN PRINT "ftest "; fff%; dat&; df&; qf%; qr%;
                  da& = dat& AND 15
                  IF da& <> aaa% THEN PRINT "atest "; aaa%; dat&; da&; qf%; qr%;
             NEXT
        NEXT
        PRINT "Address and Function code test complete"

                  CALLS cccc(mtslot%)
                  CALLS cfsa(1, tmod0%, dat&, qr%)
                  PRINT dat& AND 3584;
                  IF dat& AND 2048 = 0 THEN PRINT "C failed   ";

                  CALLS cccz(mtslot%)
                  CALLS cfsa(1, tmod0%, dat&, qr%)
                  PRINT dat& AND 3584;
                  IF dat& AND 1024 = 0 THEN PRINT "Z failed  ";

                  CALLS ccci(mtslot%, 255)
                  CALLS cfsa(1, tmod0%, dat&, qr%)
                  PRINT dat& AND 3584;
                  IF dat& AND 512 = 0 THEN PRINT "I failed   ";

                  CALLS ccci(mtslot%, 0)
                  PRINT " C,Z,I test complete"
        PRINT "Begin 24 bit random write/read test"
        PRINT
        npass = 0
        DO
        FOR jklmn% = 0 TO 10000
         r1% = INT(RND * 4095.999)
        r2% = INT(RND * 4095.999)
        dwrt& = r1%
        dwrt& = dwrt& * 4096 + r2%
        CALLS cfsa(16, tmod0%, dwrt&, qw%)
        CALLS cfsa(0, tmod0%, drd&, qr%)
        IF dwrt& <> drd& THEN
             LOCATE 24, 1: errnum% = errnum% + 1
             PRINT "read write error"; errnum%; dwrt&; qw%; drd&; qr%
        END IF
        kkk$ = UCASE$(INKEY$)
        IF kkk$ = "R" THEN GOTO start
        IF kkk$ = "Q" THEN SYSTEM
        NEXT
        npass = npass + 1
        LOCATE , 1: PRINT "pass complete, #"; npass;
        LOOP
        SYSTEM

        END

        FUNCTION init% (file$, var$, value)
'
' read in an initialization file of the form:
'    varname=value
'    varname=value
' etc
'
' case is ignored
' one variable per line
' variables can be in any order
' init% returns as 0 for variables not found
'
        init% = 0
        ON LOCAL ERROR GOTO lerr
        OPEN file$ FOR INPUT AS #1
             WHILE NOT EOF(1)
```

```
            LINE INPUT #1, line$
            line$ = UCASE$(line$)
            IF (INSTR(line$, UCASE$(var$)) <> 0) AND (INSTR(line$, "=") <> 0) THEN
                jnk$ = RIGHT$(line$, LEN(line$) - INSTR(line$, "="))
                value = VAL(jnk$): init% = 1
            END IF
        WEND
lerr:
        CLOSE #1
        EXIT FUNCTION
        END FUNCTION
```

59 BIT INPUT MODULE

Xilinx bit file is T2366IN.BIT

The I/O connectors should be configured as all inputs. The 10125 ECL to TTL converters and the red termination SIPs should be installed. All 10124 ICs and all yellow pull-down SIPs should be removed. No damage will result if the configuration is incorrect, but the module will, of course, not function correctly.

CAMAC Function Codes

F0, A0	Read 8 bits from the A input connector.
F0, A1	Read 17 bits from the B input connector.
F0, A2	Read 17 bits from the C input connector.
F0, A3	Read 17 bits from the D input connector.

In all cases, the numbered inputs correspond to the CAMAC R lines.

This module will never return X or Q.

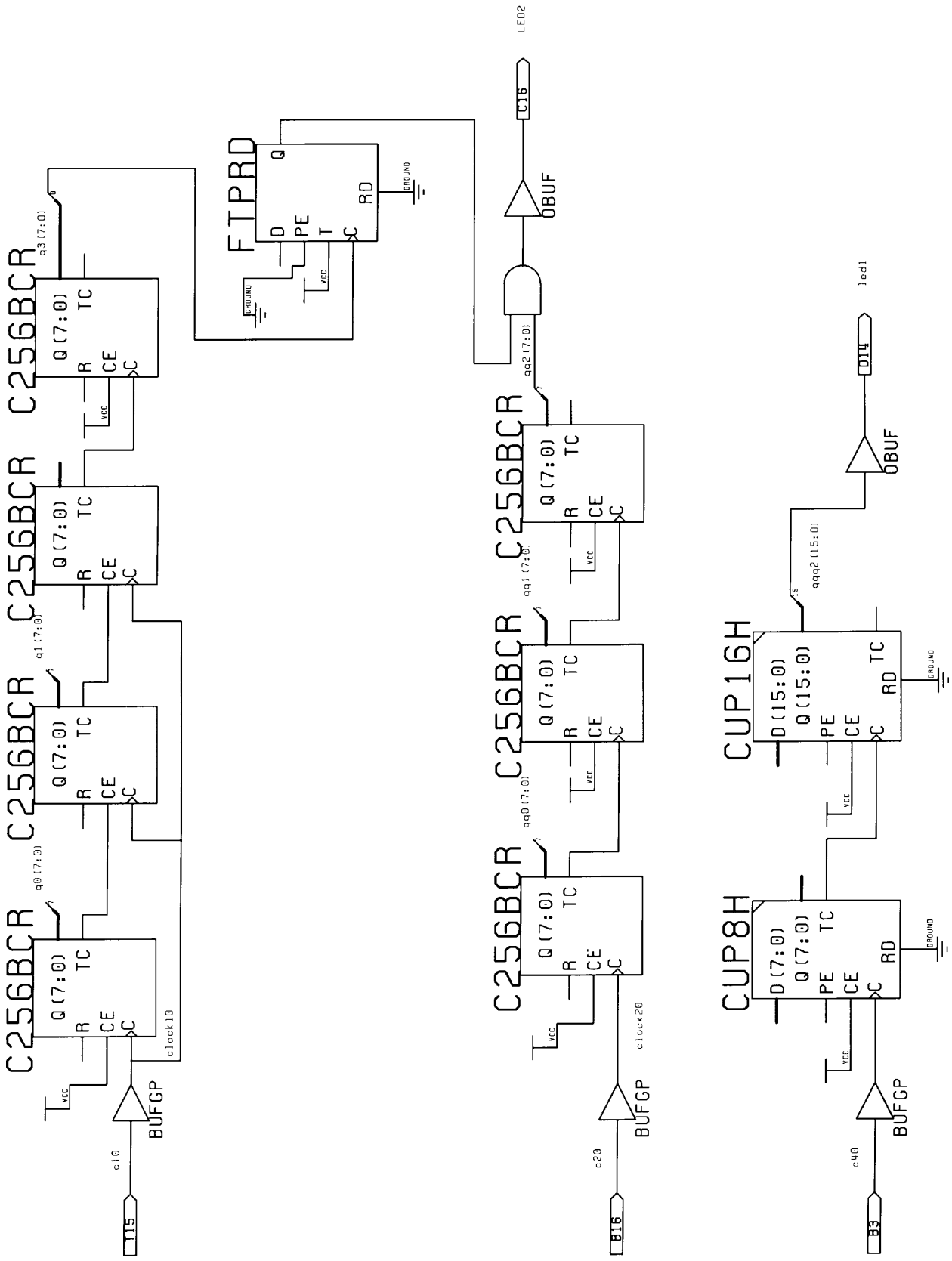
C256BCR
Q(7:0)
TC
R
CE
C
VCC
BUFGP
T15
c10
clock10
q0(7:0)
q1(7:0)
q3(7:0)
FTPRD
D
PE
T
RD
Q
GROUND
OBUF
LED2
C16
qq0(7:0)
qq1(7:0)
qq2(7:0)
B16
c20
clock20
CUP16H
D(15:0)
Q(15:0)
qqq2(15:0)
CUP8H
D(7:0)
B3
c40
led1
D14

Figure 1

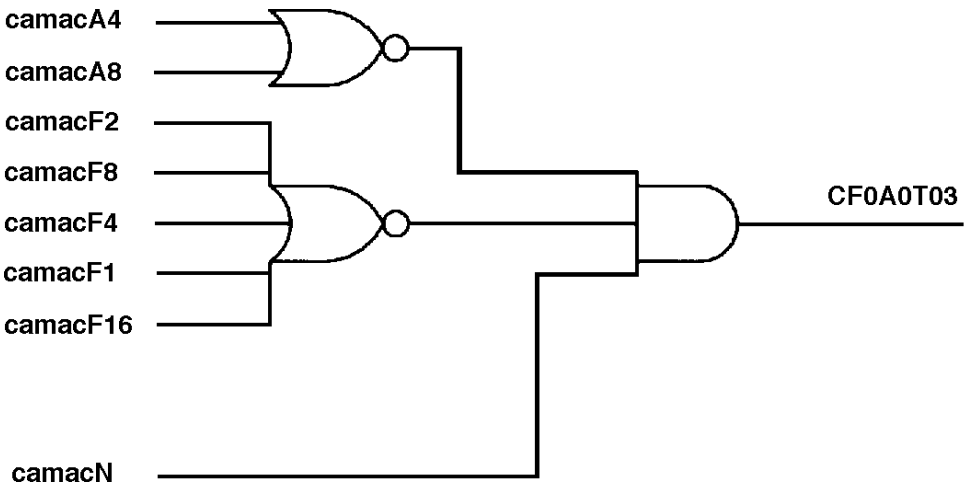
camacA4
camacA8
camacF2
camacF8
camacF4
camacF1
camacF16
camacN
CF0A0T03

Figure 2

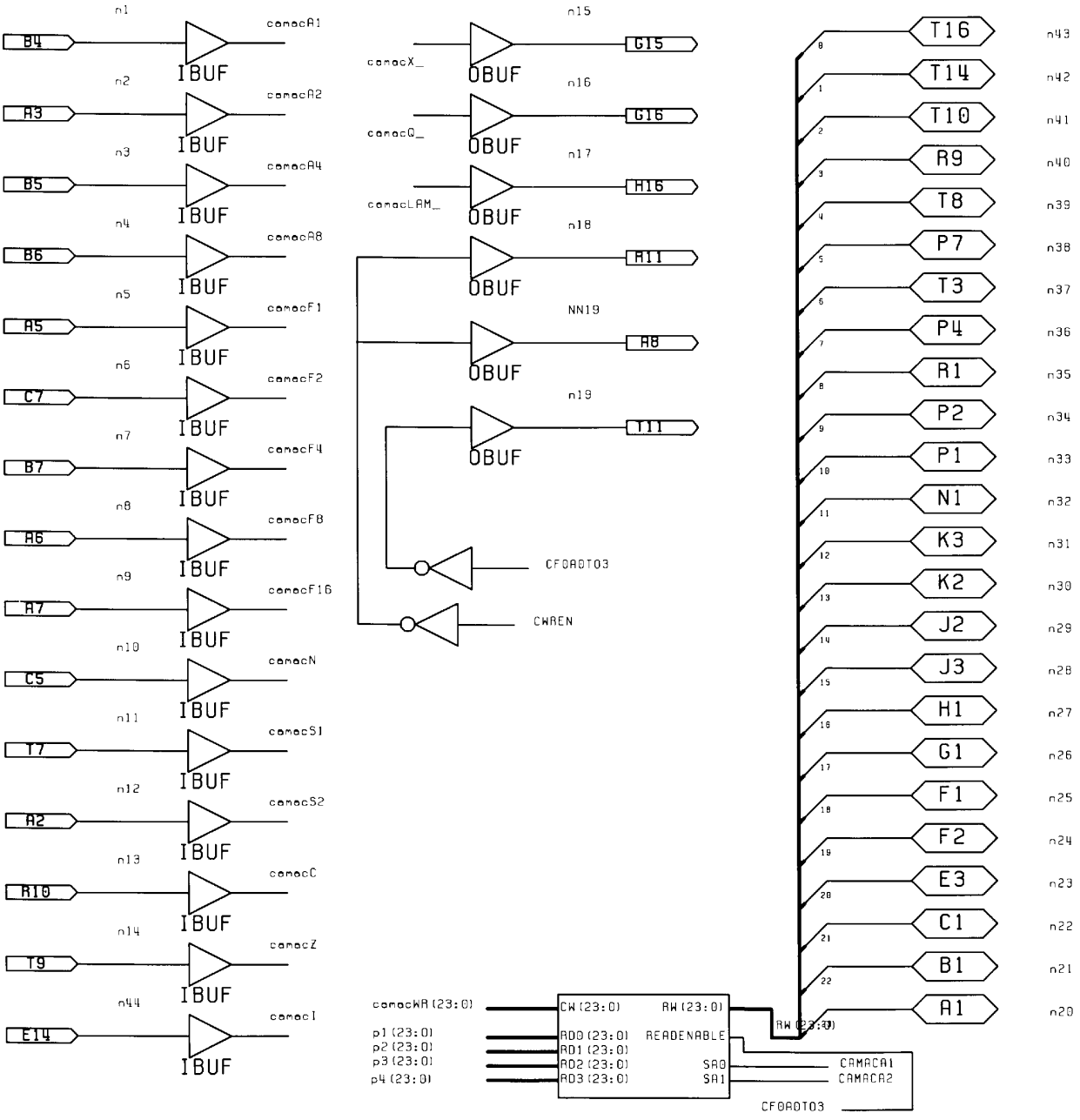
B4
A3
B5
B6
A5
C7
B7
A6
A7
C5
T7
A2
R10
T9
E14
IBUF
camacA1
camacA2
camacA4
camacA8
camacF1
camacF2
camacF4
camacF8
camacF16
camacN
camacS1
camacS2
camacC
camacZ
camacI
camacX_
camacQ_
camacLAM_
OBUF
G15
G16
R16
A11
A8
T11
NN19
CF0A0T03
CWREN
T16
T14
T10
R9
T8
P7
T3
P4
R1
P2
P1
N1
K3
K2
J2
J3
H1
G1
F1
F2
E3
C1
B1
A1
camacWR(23:0)
p1(23:0)
p2(23:0)
p3(23:0)
p4(23:0)
CW(23:0)
RW(23:0)
RD0(23:0)
RD1(23:0)
RD2(23:0)
RD3(23:0)
READENABLE
SA0
SA1
CAMACA1
CAMACA2
CF0A0T03

Figure 3

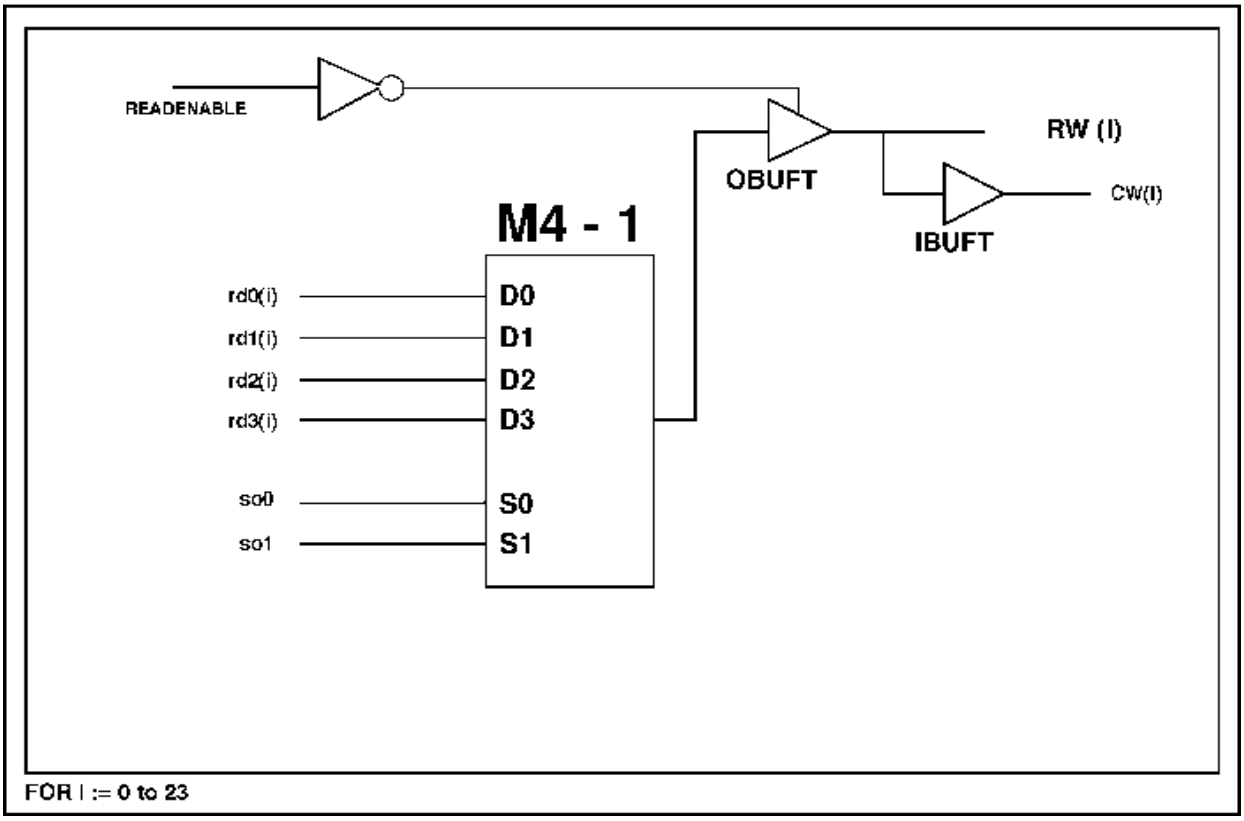
READENABLE
OBUFT
RW (I)
IBUFT
CW(I)
M4 - 1
rd0(i)
rd1(i)
rd2(i)
rd3(i)
D0
D1
D2
D3
so0
so1
S0
S1
FOR I := 0 to 23

Figure 4

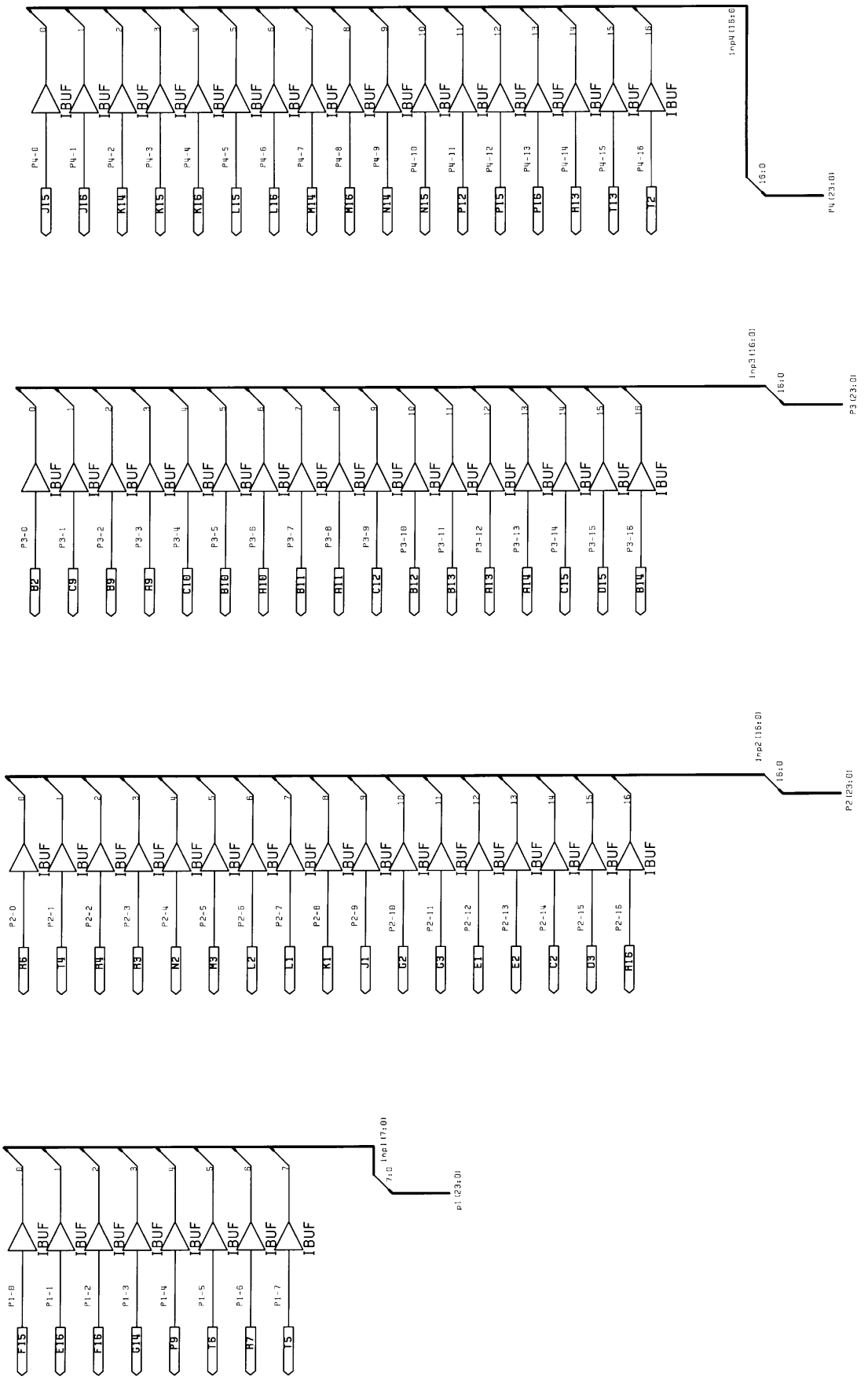
IBUF
P4-0
P4-1
P4-2
P4-3
P4-4
P4-5
P4-6
P4-7
P4-8
P4-9
P4-10
P4-11
P4-12
P4-13
P4-14
P4-15
P4-16
J15
J16
K14
K15
K16
L15
L16
M14
M16
N14
N15
P12
P15
P16
R13
T13
T2
Inp4[15:0]
I6:0
P4[23:0]
P3-0
P3-1
P3-2
P3-3
P3-4
P3-5
P3-6
P3-7
P3-8
P3-9
P3-10
P3-11
P3-12
P3-13
P3-14
P3-15
P3-16
B2
C9
B9
A9
C10
B10
A10
B11
A11
C12
B12
B13
A13
A14
C15
D15
B14
Inp3[15:0]
I6:0
P3[23:0]
P2-0
P2-1
P2-2
P2-3
P2-4
P2-5
P2-6
P2-7
P2-8
P2-9
P2-10
P2-11
P2-12
P2-13
P2-14
P2-15
P2-16
H6
T4
R4
R3
N2
M3
L2
L1
K1
J1
G2
G3
E1
E2
C2
D3
A16
Inp2[15:0]
I6:0
P2[23:0]
P1-0
P1-1
P1-2
P1-3
P1-4
P1-5
P1-6
P1-7
F15
E16
F16
G14
P9
T6
R7
T5
Inp1[7:0]
I7:0
P1[23:0]

Figure 5

59 BIT OUTPUT REGISTER MODULE

Xilinx bit file is T2366OUT.BIT

The I/O connectors should be configured as all outputs. All 10124 TTL to ECL converters and all yellow pull-down SIPS should be installed. The 10125 ECL to TTL converters and the red termination SIPs should be removed. This should be carefully checked, it is possible to damage the Xilinx chip if any channels are configured as inputs.

This module also contains a 24 bit read write register, and a 13 bit last command register. On every S2, the A line, F lines, I, Z, C, and N are latched (A1 is R1, F1 is R5, I is R10, etc). These can be read on the next CAMAC cycle.

CAMAC Function Codes

F0, A0	Read from the internal 24 bit register.
F1, A0	Read from the 13 bit last command register.
F16, A0	Write to the internal 24 bit register.
F17, A0	Write 8 bits to the A input connector.
F17, A1	Write 17 bits to the B input connector.
F17, A2	Write 17 bits to the C input connector.
F17, A3	Write 17 bits to the D input connector.

In all cases, the numbered inputs correspond to the CAMAC R lines.

The module returns Q and X for the F0, F1, and F16 commands. Neither Q nor X is returned for the F17 commands.

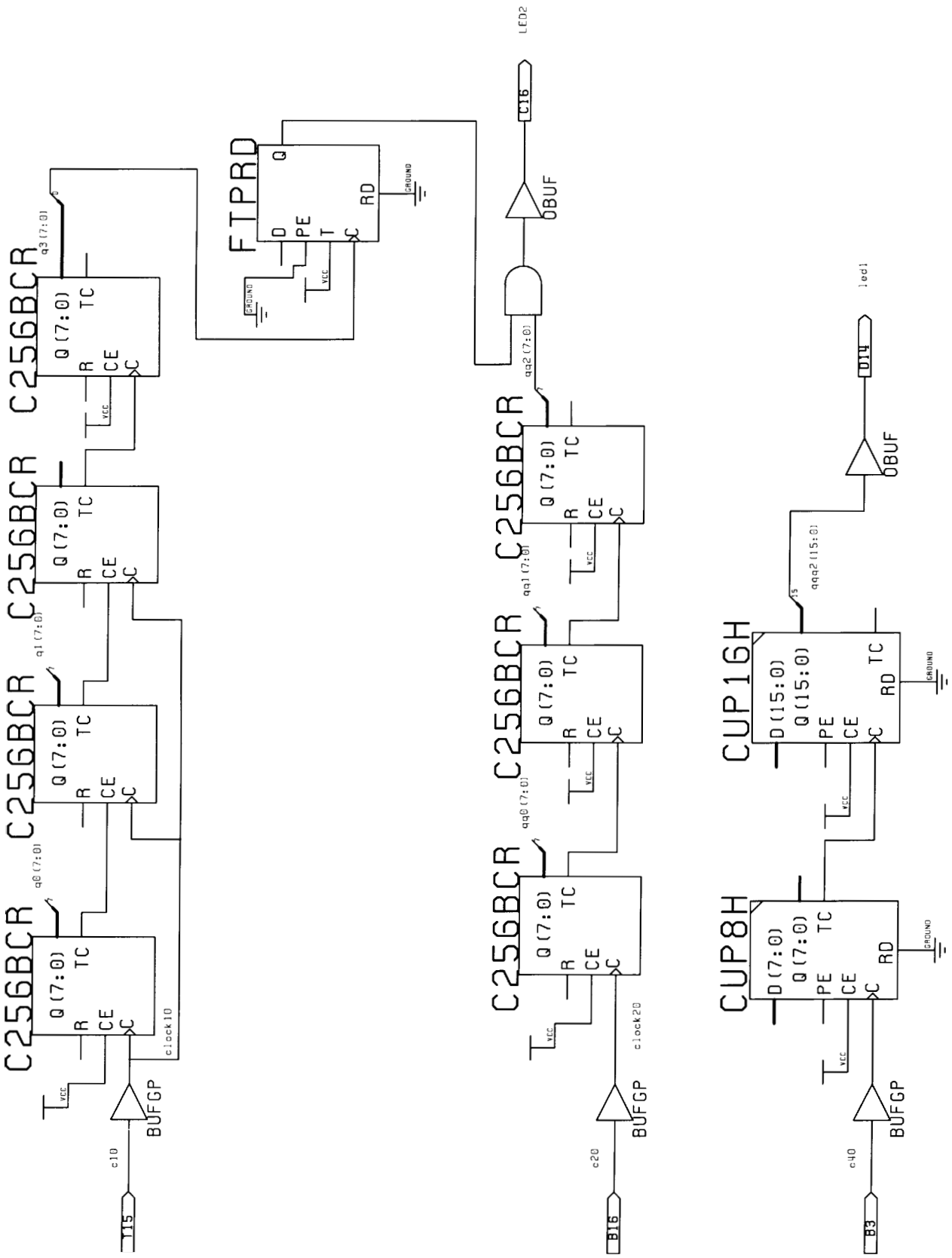
C256BCR
Q(7:0)
TC
R
CE
C
q3(7:0)
q2(7:0)
q1(7:0)
q0(7:0)
VCC
BUFGP
clock10
clock20
c10
c20
T15
B16
FTPRD
D
PE
T
RD
Q
GROUND
LED2
C16
OBUF
qq2(7:0)
qq1(7:0)
qq0(7:0)
CUP16H
CUP8H
D(15:0)
Q(15:0)
D(7:0)
Q(7:0)
qqq2(15:0)
led1
D14
c40
B3

Figure 6

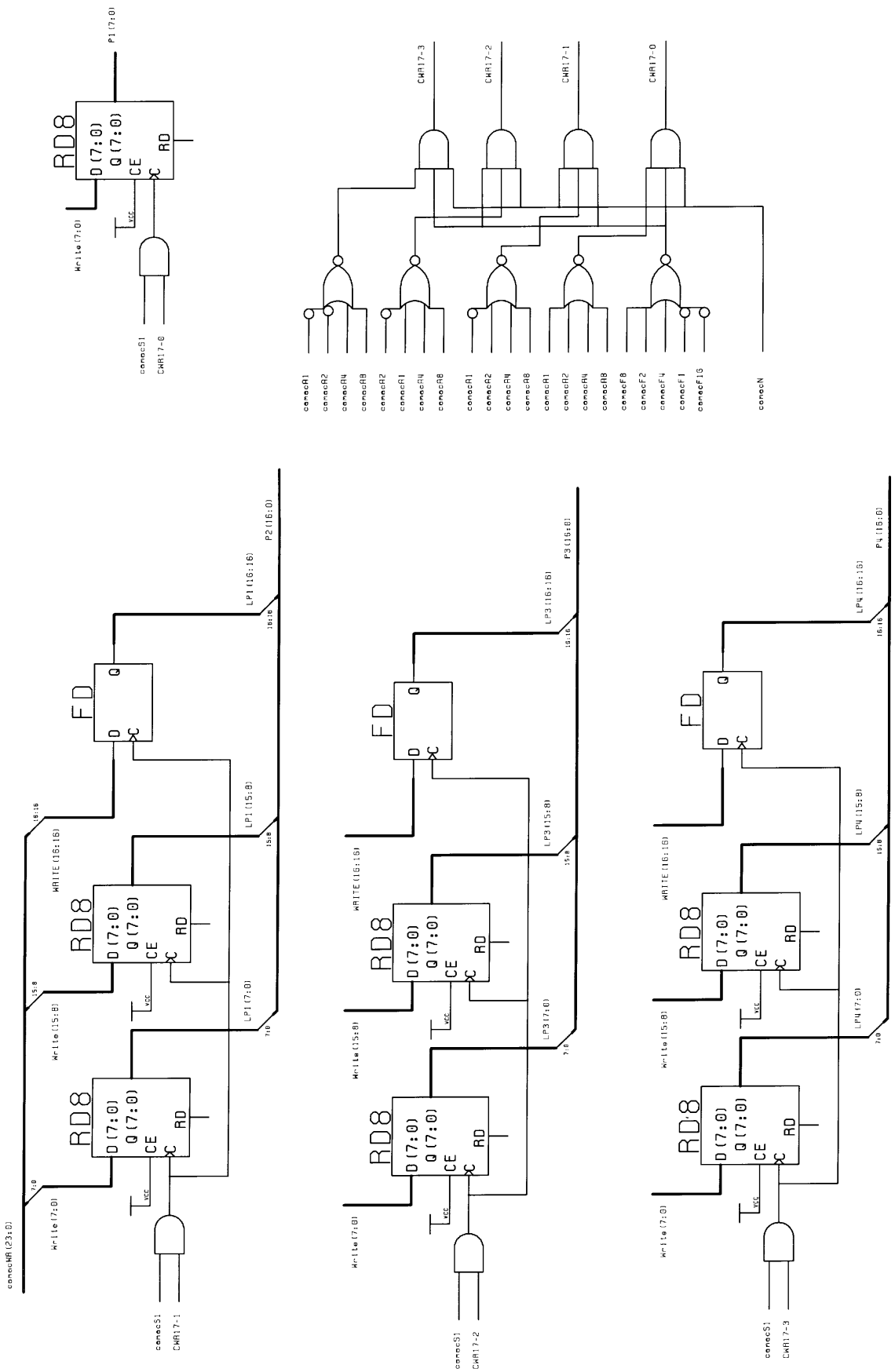
RD8
D(7:0)
Q(7:0)
CE
RD
FD
D
Q
VCC
P1(7:0)
P2(16:0)
P3(16:0)
P4(16:0)
LP1(16:16)
LP1(15:8)
LP1(7:0)
LP3(16:16)
LP3(15:8)
LP3(7:0)
LP4(16:16)
LP4(15:8)
LP4(7:0)
WRITE(16:16)
Write(15:8)
Write(7:0)
connNR(23:0)
connS1
CNR17-0
CNR17-1
CNR17-2
CNR17-3
connA1
connA2
connA4
connA8
connF2
connF4
connF1
connF16
connN

Figure 7

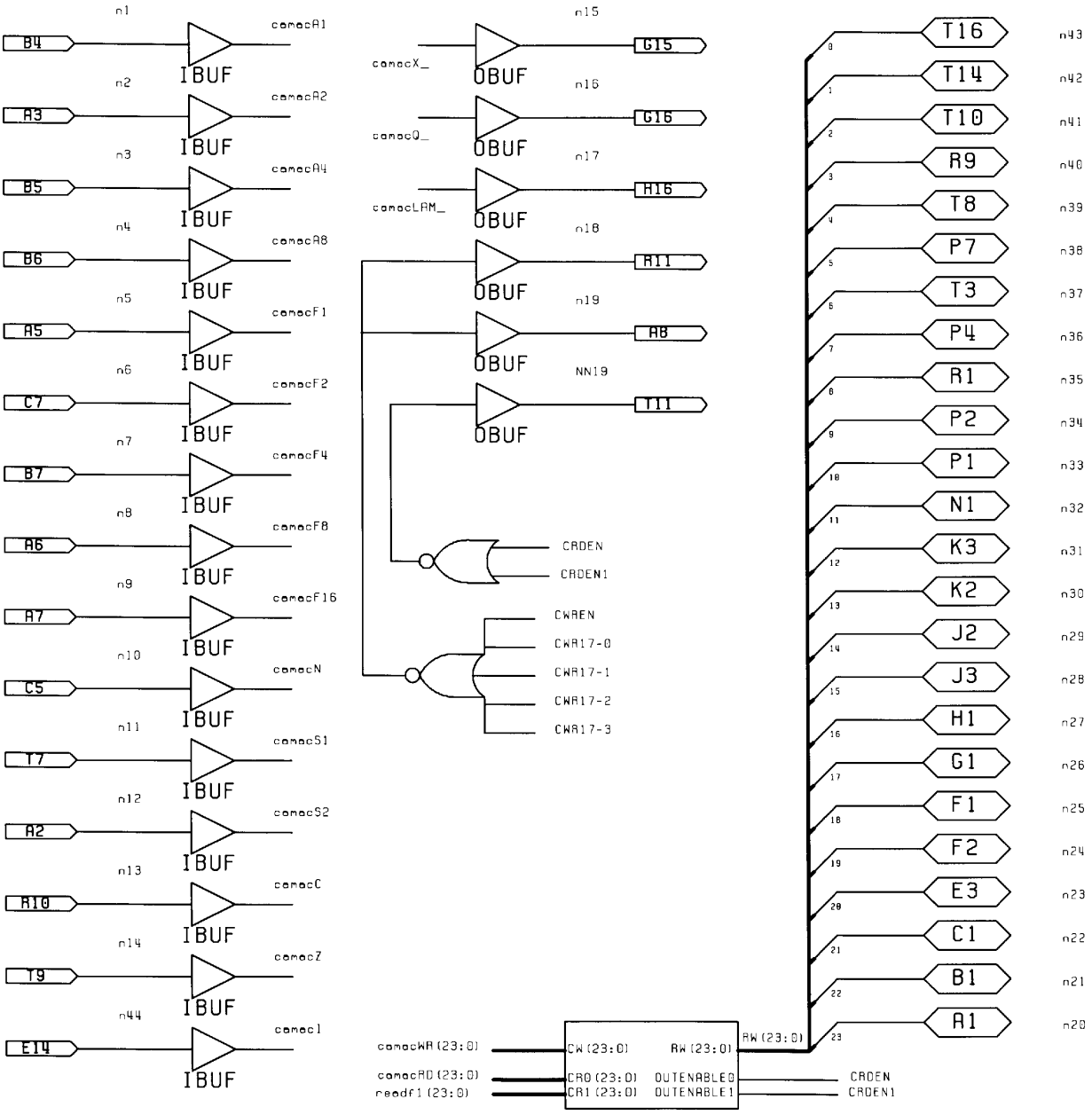
B4
A3
B5
B6
A5
C7
B7
A6
A7
C5
T7
A2
R10
T9
E14
IBUF
camacA1
camacA2
camacA4
camacA8
camacF1
camacF2
camacF4
camacF8
camacF16
camacN
camacS1
camacS2
camacC
camacZ
camacI
camacX_
camacQ_
camacLAM_
OBUF
G15
G16
R16
R11
A8
T11
NN19
CRDEN
CRDEN1
CWREN
CWR17-0
CWR17-1
CWR17-2
CWR17-3
T16
T14
T10
R9
T8
P7
T3
P4
R1
P2
P1
N1
K3
K2
J2
J3
H1
G1
F1
F2
E3
C1
B1
A1
camacWR(23:0)
camacRD(23:0)
readf1(23:0)
CW(23:0)
CR0(23:0)
CR1(23:0)
RW(23:0)
OUTENABLE0
OUTENABLE1

Figure 8

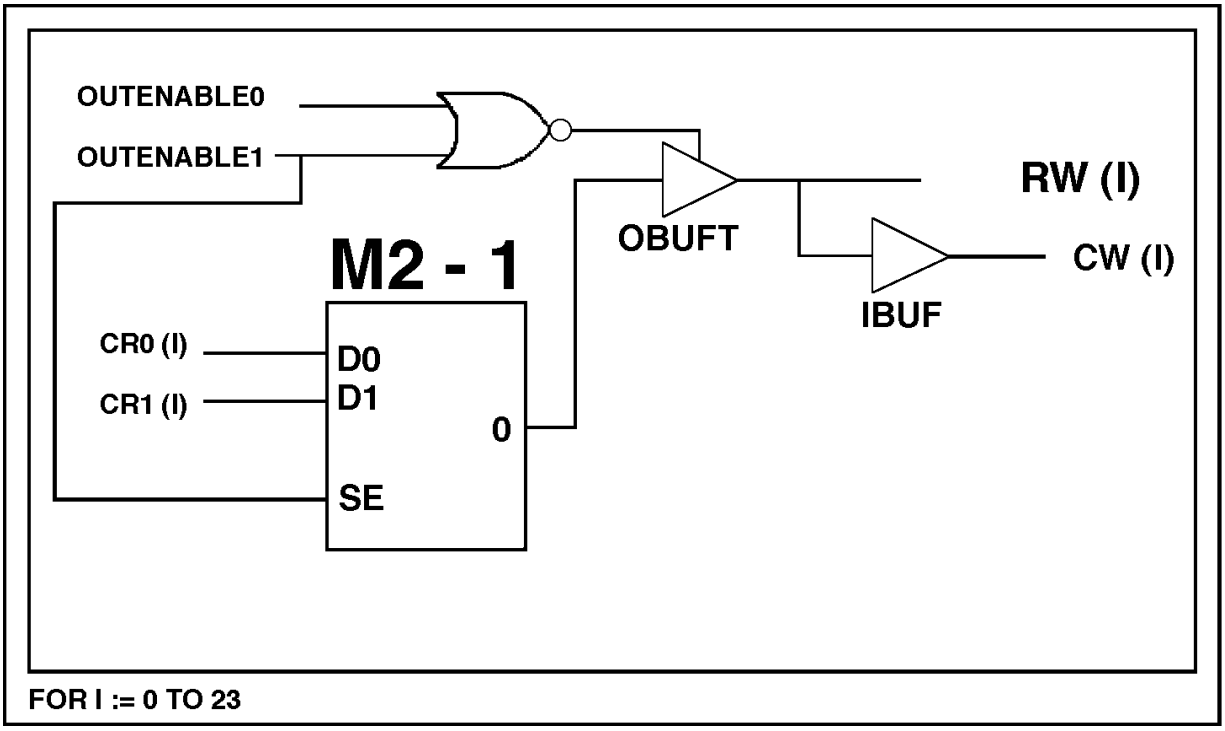
OUTENABLE0
OUTENABLE1
M2 - 1
OBUFT
RW (I)
CW (I)
IBUF
CR0 (I)
CR1 (I)
D0
D1
0
SE
FOR I := 0 TO 23

Figure 9

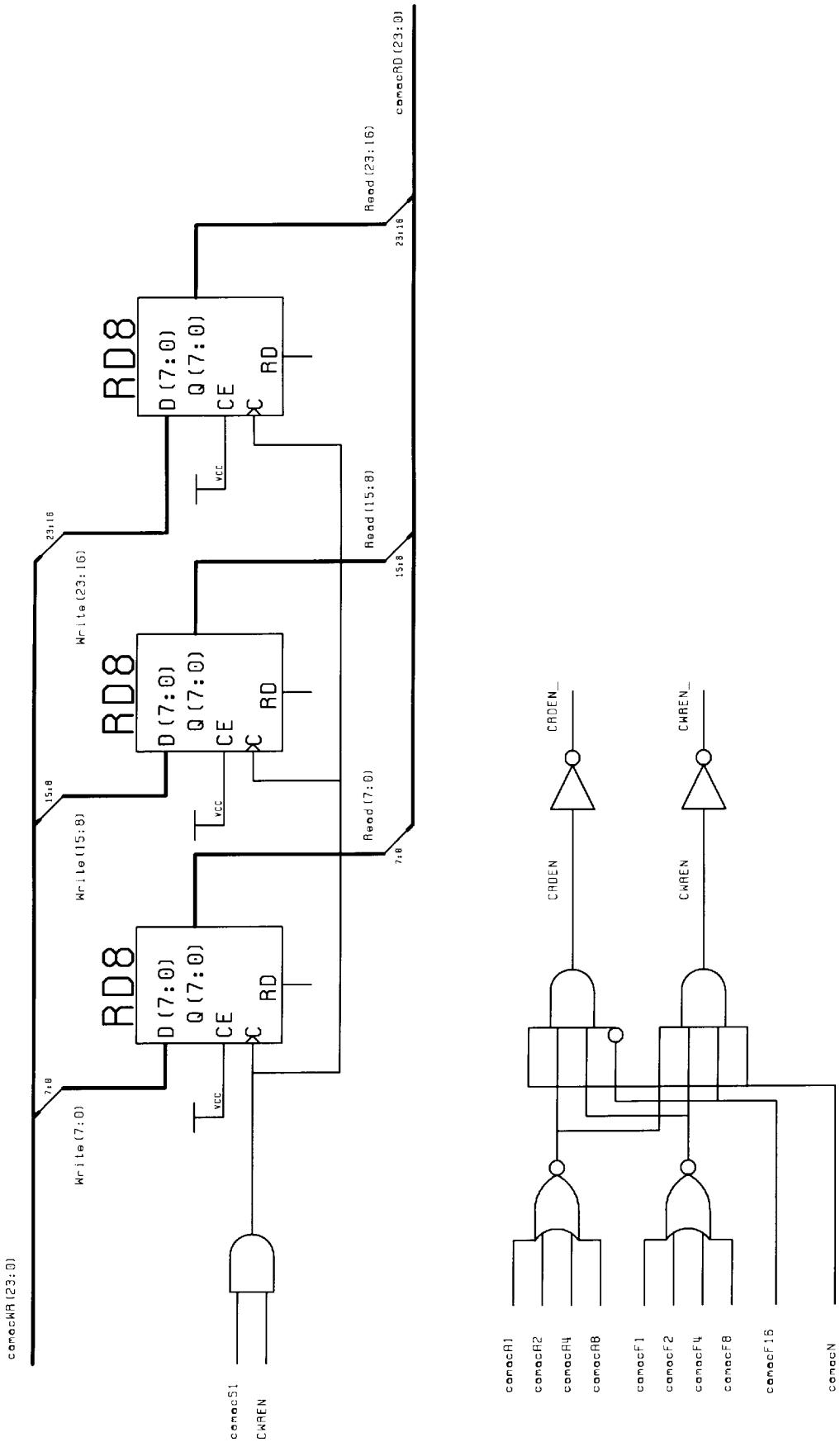
camacRD(23:0)
Read(23:16)
23:16
RD8
D(7:0)
Q(7:0)
CE
RD
VCC
Read(15:8)
15:8
Write(23:16)
23:16
Read(7:0)
7:0
Write(15:8)
15:8
Write(7:0)
7:0
camacWR(23:0)
camacS1
CWREN
CADEN_
CWREN_
CADEN
CWREN
camacA1
camacA2
camacA4
camacA8
camacF1
camacF2
camacF4
camacF8
camacF16
camacN

Figure 10

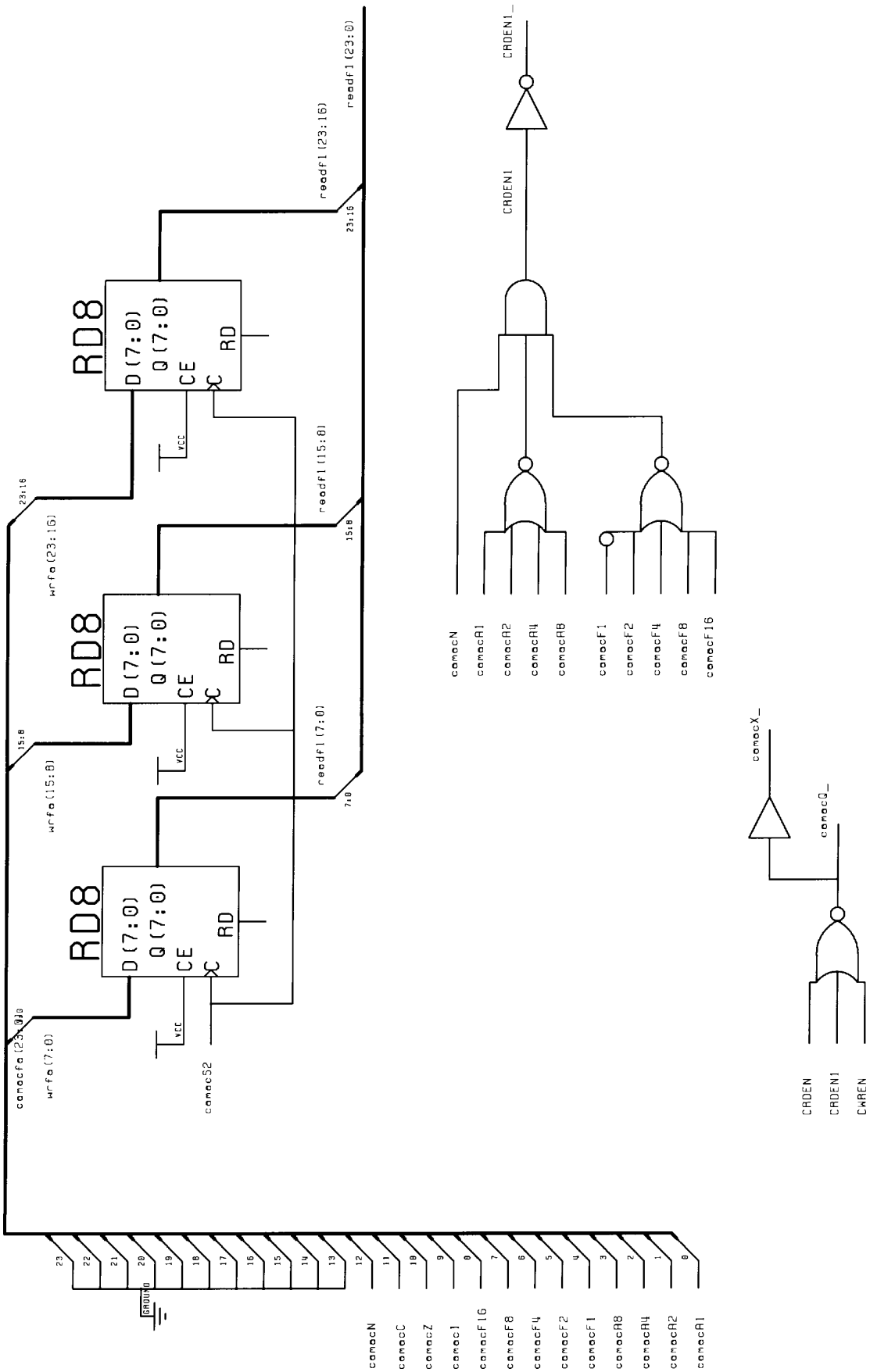
RD8
D(7:0)
Q(7:0)
CE
RD
VCC
wrfa(23:16)
wrfa(15:8)
wrfa(7:0)
readf1(23:16)
readf1(15:8)
readf1(7:0)
readf1(23:0)
cmacfa(23:0)
cmac52
CADEN1_
CADEN1
cmacN
cmacA1
cmacA2
cmacA4
cmacA8
cmacF1
cmacF2
cmacF4
cmacF8
cmacF16
cmacX_
cmacG_
CADEN
CADEN1
CWREN
GROUND
cmacC
cmacZ
cmac1

Figure 11

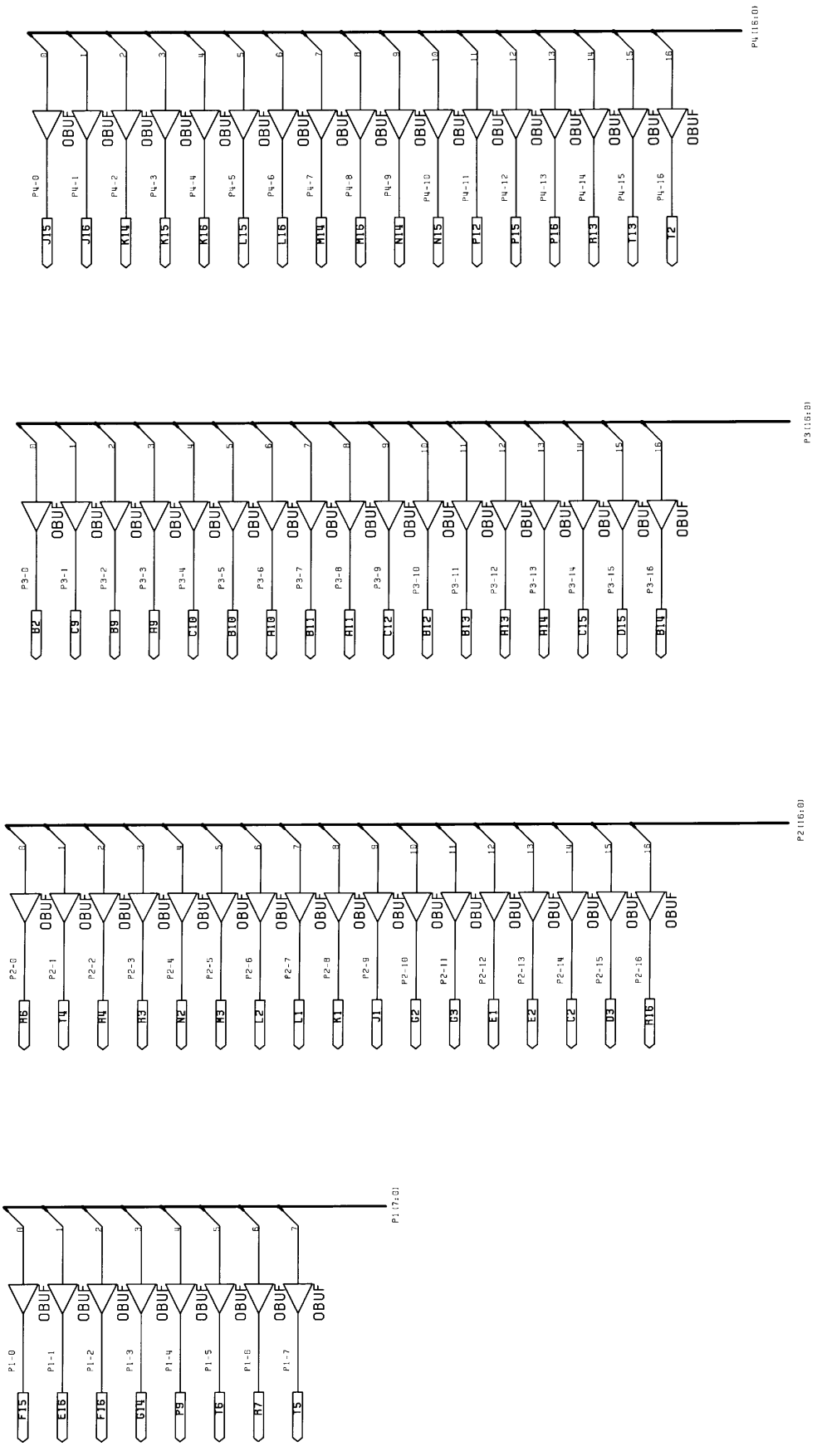
P4[15:0]
P4-0
P4-1
P4-2
P4-3
P4-4
P4-5
P4-6
P4-7
P4-8
P4-9
P4-10
P4-11
P4-12
P4-13
P4-14
P4-15
P4-16
J15
J16
K14
K15
K16
L15
L16
M14
M16
N14
N15
P12
P15
P16
R13
T13
T2
P3[15:0]
P3-0
P3-1
P3-2
P3-3
P3-4
P3-5
P3-6
P3-7
P3-8
P3-9
P3-10
P3-11
P3-12
P3-13
P3-14
P3-15
P3-16
B2
C9
B9
H9
C10
B10
H10
B11
H11
C12
B12
B13
H13
H14
C15
D15
B14
P2[15:0]
P2-0
P2-1
P2-2
P2-3
P2-4
P2-5
P2-6
P2-7
P2-8
P2-9
P2-10
P2-11
P2-12
P2-13
P2-14
P2-15
P2-16
M5
T4
M4
R3
N2
M3
L2
L1
K1
J1
G2
G3
E1
E2
C2
D3
M16
P1[7:0]
P1-0
P1-1
P1-2
P1-3
P1-4
P1-5
P1-6
P1-7
F15
E16
F16
G14
P9
T6
R7
T5
OBUF

Figure 12

CAMAC DATAWAY TEST MODULE

Xilinx bit file is T2366E.BIT

This module is loaded in the eprom which is installed in the 2366 when shipped.

This module contains a 24 bit read write register, and a 13 bit last command register. On every S2, the A line, F lines, I, Z, C, and N are latched (A1 is R1, F1 is R5, I is R10, etc). These can be read on the next CAMAC cycle.

The I/O connectors can have any configuration, including all ICs and SIPs installed (the shipping configuration). No damage will result, but the power dissipation will be higher than normal. None of the I/O lines are driven by the Xilinx chip.

CAMAC Function Codes

F0, A0	Read from the internal 24 bit register.
F1, A0	Read from the 13 bit last command register.
F16, A0	Write to the internal 24 bit register.

The module returns Q and X for the F0, F1, and F16 commands.

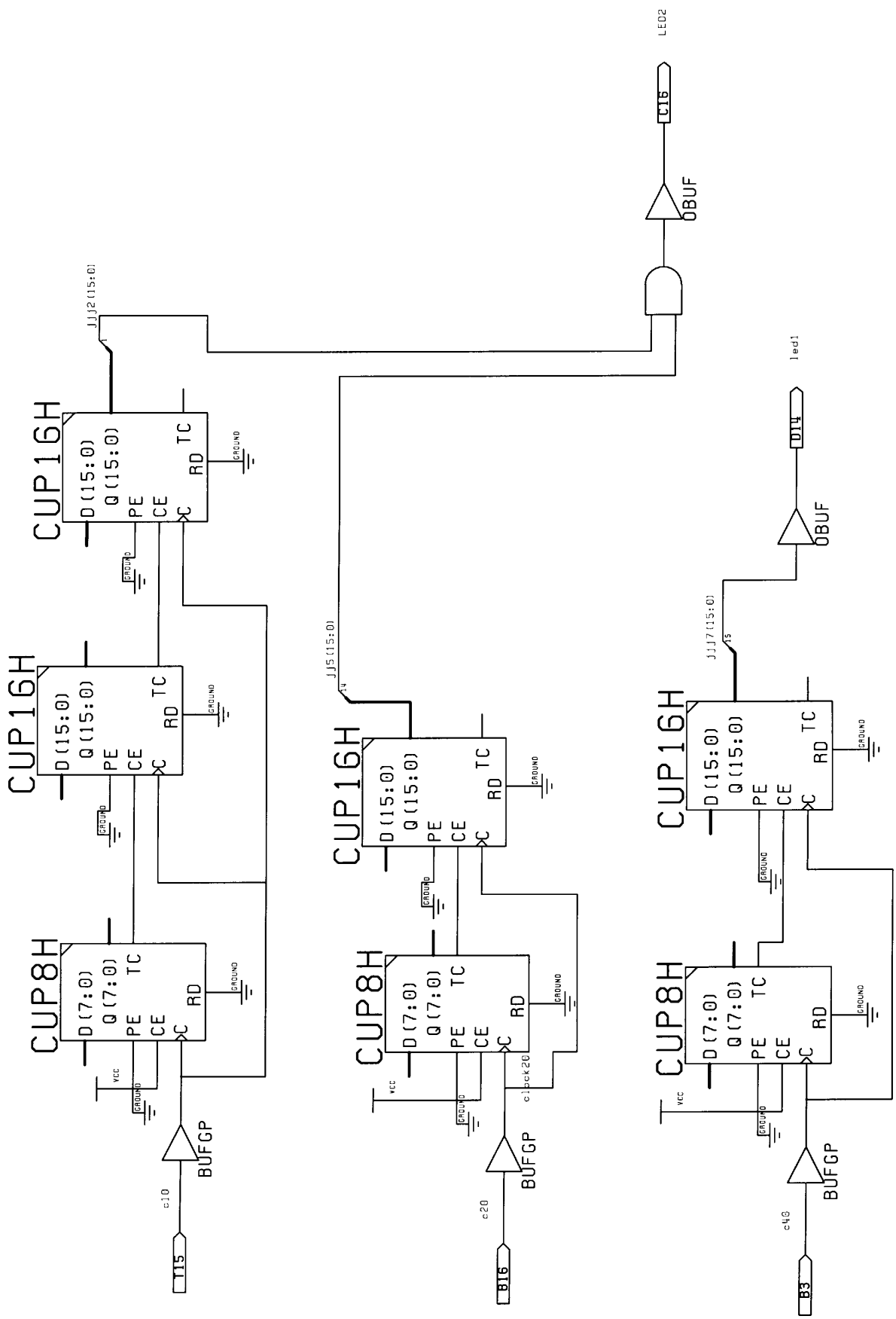
LED2
C16
OBUF
led1
D14
OBUF
JJJ2(15:0)
JJJ5(15:0)
JJJ7(15:0)
CUP16H
D(15:0)
Q(15:0)
PE
CE
C
RD
TC
GROUND
CUP8H
D(7:0)
Q(7:0)
VCC
BUFGP
clock20
c10
c20
c40
T15
B16
B3

Figure 13

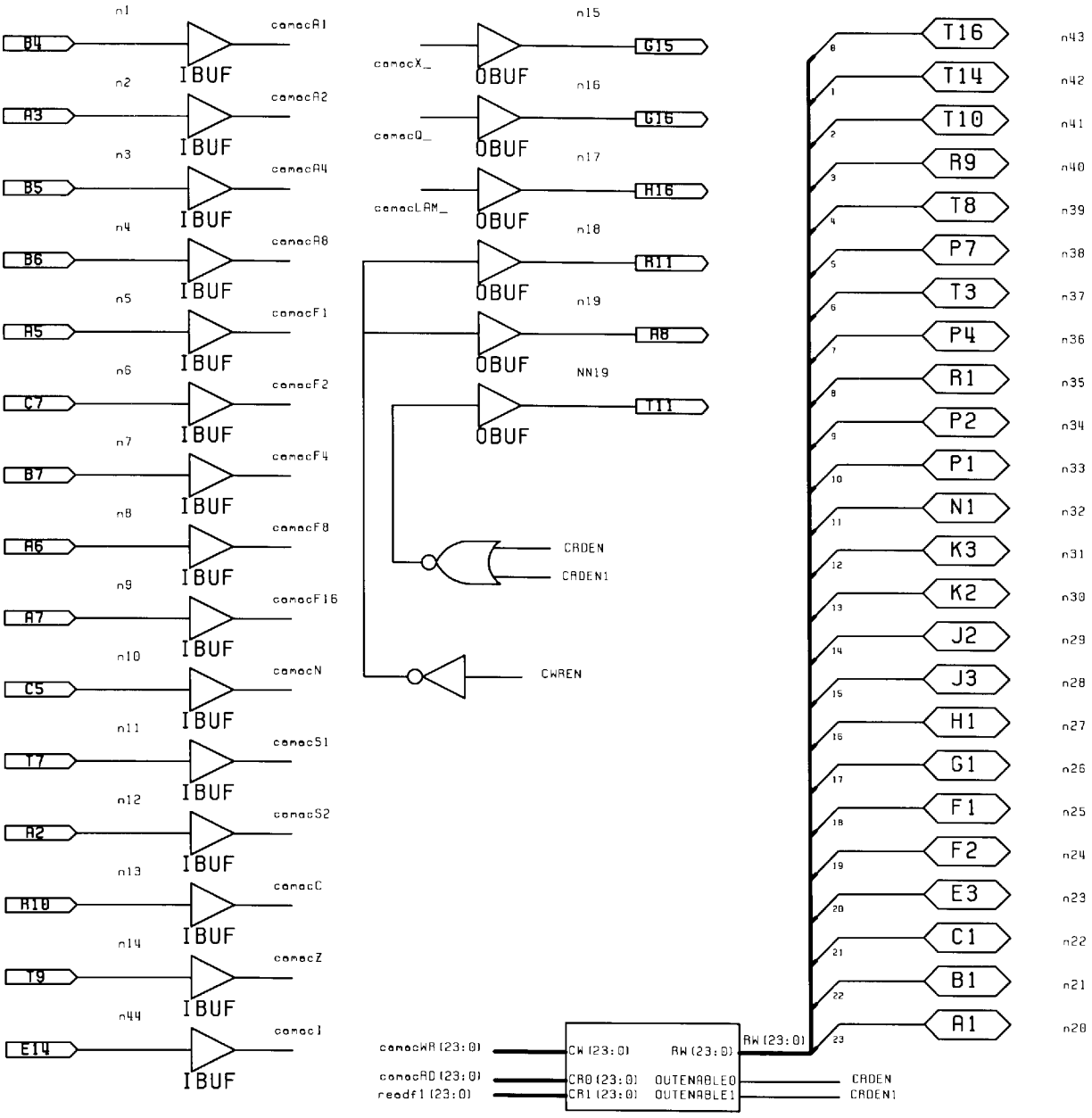
B4
A3
B5
B6
A5
C7
B7
A6
A7
C5
T7
A2
R10
T9
E14
IBUF
camacA1
camacA2
camacA4
camacA8
camacF1
camacF2
camacF4
camacF8
camacF16
camacN
camacS1
camacS2
camacC
camacZ
camacI
camacX_
camacQ_
camacLAM_
G15
G16
R16
R11
A8
T11
OBUF
NN19
CRDEN
CRDEN1
CWREN
camacWR[23:0]
camacRD[23:0]
readf1[23:0]
CW[23:0]
CR0[23:0]
CR1[23:0]
RW[23:0]
OUTENABLE0
OUTENABLE1
T16
T14
T10
R9
T8
P7
T3
P4
R1
P2
P1
N1
K3
K2
J2
J3
H1
G1
F1
F2
E3
C1
B1
A1

Figure 14

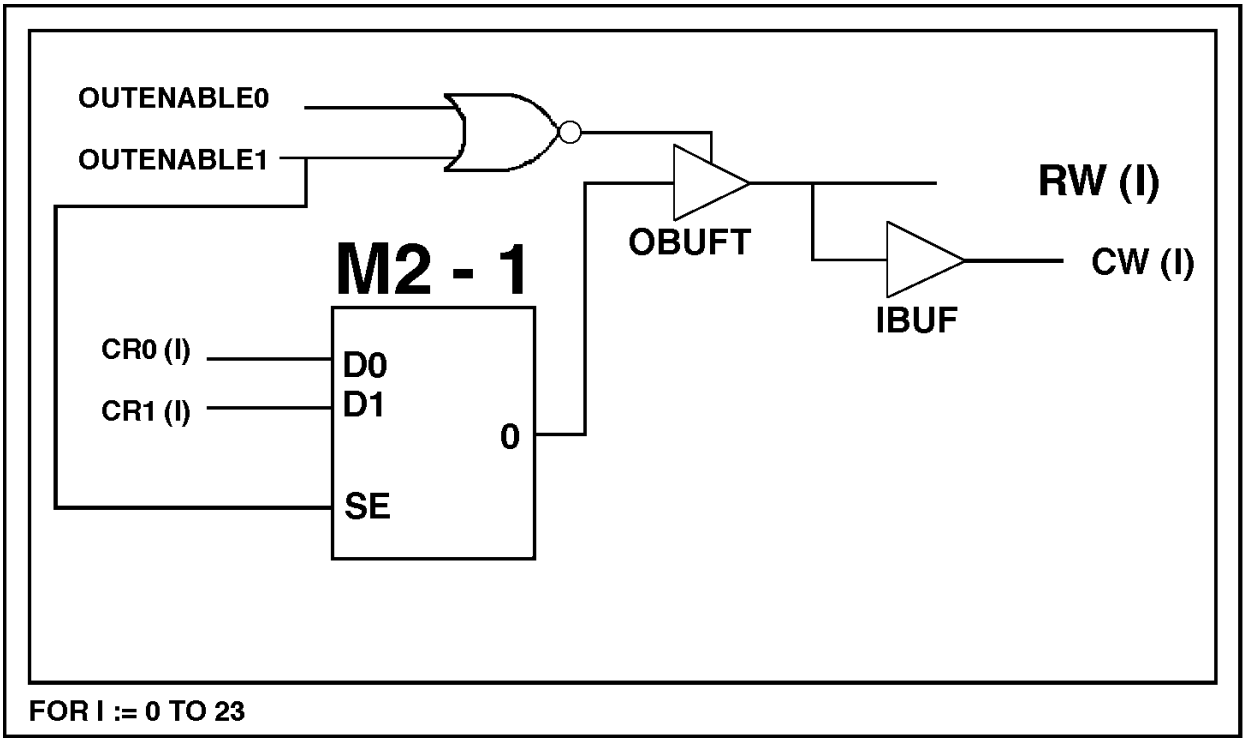
OUTENABLE0
OUTENABLE1
RW (I)
OBUFT
CW (I)
IBUF
M2 - 1
CR0 (I)
CR1 (I)
D0
D1
0
SE
FOR I := 0 TO 23

Figure 15

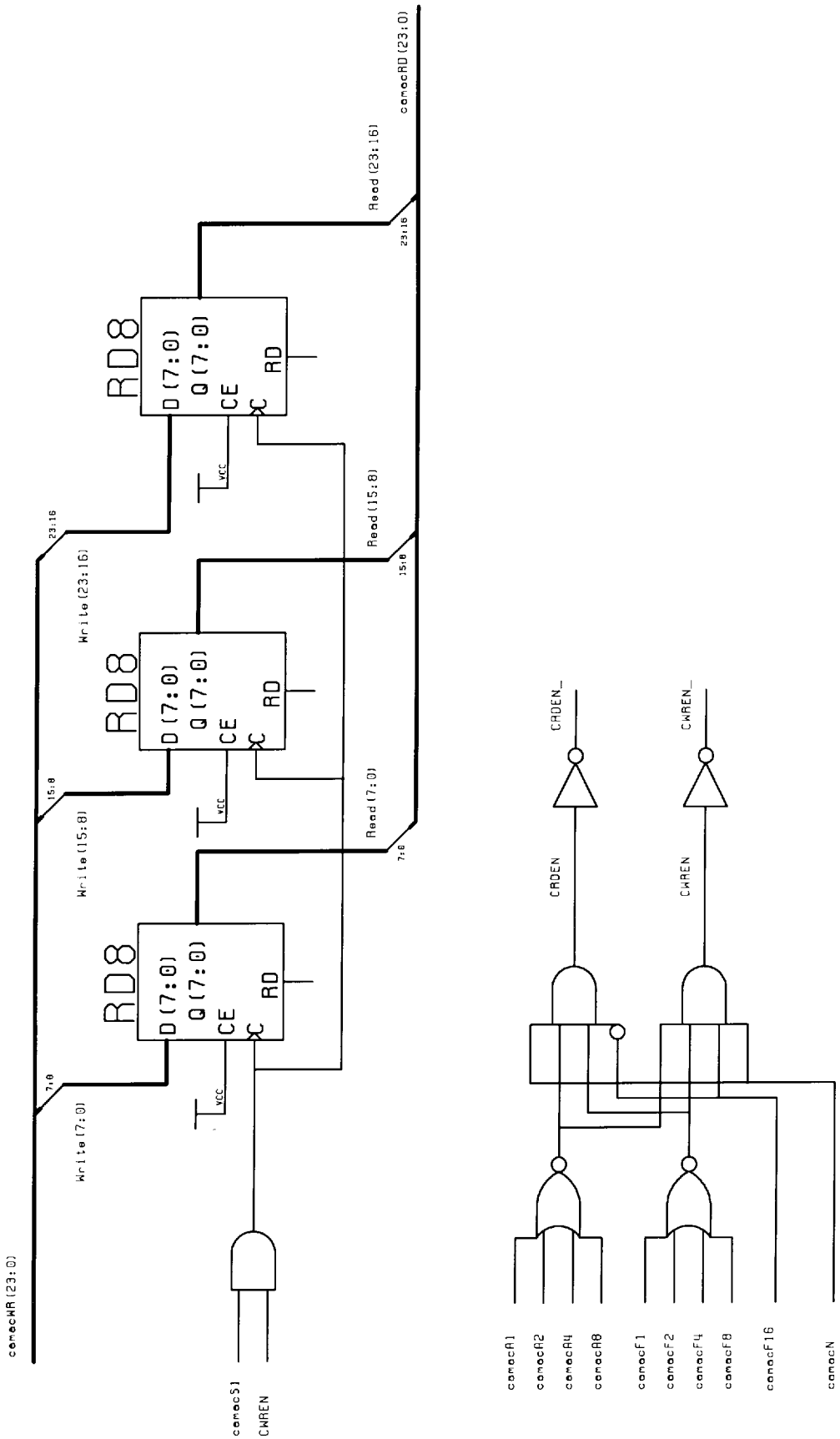
RD8
D(7:0)
Q(7:0)
CE
RD
VCC
Write(23:16)
Write(15:8)
Write(7:0)
Read(23:16)
Read(15:8)
Read(7:0)
23:16
15:8
7:0
cmacWR(23:0)
cmacRD(23:0)
cmacSI
CWREN
CRDEN_
CWREN_
CRDEN
CWREN
cmacA1
cmacA2
cmacA4
cmacA8
cmacF1
cmacF2
cmacF4
cmacF8
cmacF16
cmacN

Figure 16

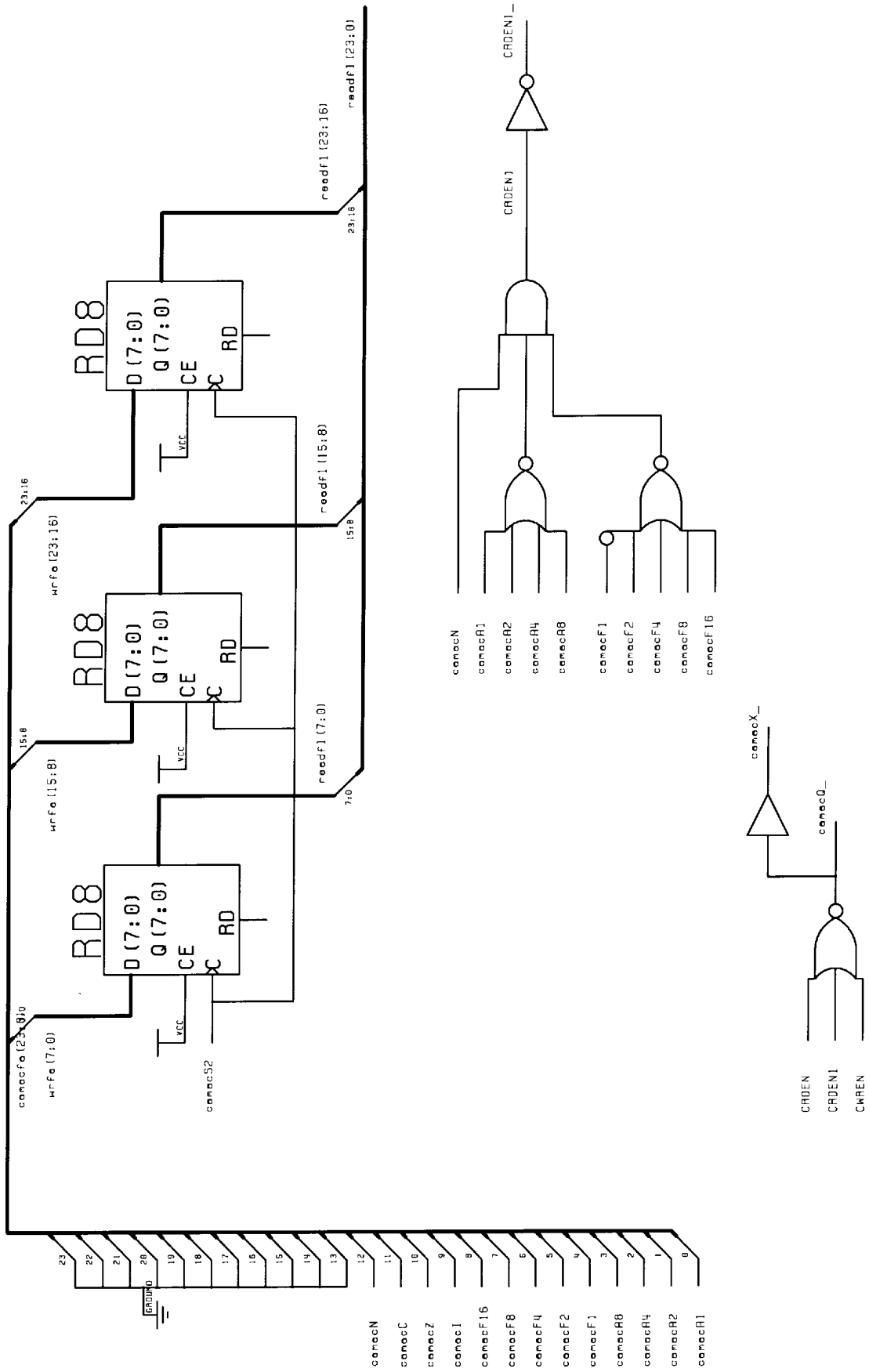
RD8
D(7:0)
Q(7:0)
CE
RD
VCC
wrfo(23:16)
wrfo(15:8)
wrfo(7:0)
readf1(23:16)
readf1(15:8)
readf1(7:0)
readf1(23:0)
CADEN1_
CADEN1
cmacN
cmacA1
cmacA2
cmacA4
cmacA8
cmacF1
cmacF2
cmacF4
cmacF8
cmacF16
cmacX_
cmacQ_
CADEN
CADEN1
CWREN
GROUND
cmacS2

Figure 17

A 48 INPUT DIGITAL MAJORITY LOGIC UNIT

Bit File: MAJOR48.BIT

Specifications

48 inputs, differential ECL. Pulse width greater than 5 nsec. Synchronized internally with the clock.

48 bit input mask register to enable/disable individual inputs.

Output: 6 bit digital word equal to the number of inputs true within the coincidence resolving time.

Selectable clock rate, internal 10, 20 or 40 MHz or a choice of 3 external clock inputs (40 MHz maximum).

40 MHz output rate (40 MHz clock)

Output pipeline delay: 175 nsec (40 MHz clock)

Coincidence resolution: minimum 25 nsec, maximum 50 nsec (40 MHz clock)

Configuration: connector A, all output, Connectors B, C, and D are all input.

Complete schematic and Xilinx programming files are available on request.

Description

The LeCroy CAMAC Model 2366 Universal Logic Module is a general purpose logic module using a Xilinx 4005 field programmable gate array. The module has 59 front panel i/o signals, which can be configured as either input or output in 7 groups of 4 channels, and 3 individual channels. In addition there is a full 24 bit CAMAC interface. The Xilinx gate array can be programmed with an on board EPROM, or from CAMAC.

For use as the 48 input majority logic module, the 2366 is configured as 48 inputs and 8 outputs. Three of the 59 possible i/o are used for an external clock. The 48 inputs are synchronized with the clock and trigger a 2 clock period one-shot. The inputs are edge sensitive, triggering on the leading edge of pulses as short as 5 nsec, Seven pipelined adder stages sum the 48 input signals to produce the binary output word. The output word is updated at the clock rate, with a 7 clock period pipeline delay. A 48 bit mask register (CAMAC F16, A0 and A1, read with F0) is used to enable (=1) or disable (=0) the individual inputs. The coincidence resolution for determining the accidental triggers is 1.5 clock periods. Real events must have all inputs within a span of one clock period or less to trigger with 100% efficiency.

This program uses most of the resources of the Xilinx 4005 in the 2366 module. It uses 243 function generators (62%) and 352 CLB flip-flops (90%). In order to operate at 40 MHz, adders with more than 3 input bits are split into 2 pipeline stages.

Several variations of this circuit are possible. By slowing the maximum clock rate to 20 MHz, the circuit can be simplified to four pipeline stages. The coincidence resolution for a 20 MHz clock will be 75 nsec for

accidentals, and 50 nsec for real events. The pipeline delay would be 200 nsec, with a 20 MHz output word rate.

If the input signals are wide pulses, a different synchronizing circuit is possible. The input can simply be clocked with the selected clock using the i/o flip flop, then differentiated to produce a short pulse. This method frees up two extra flip flops per input, which can be used to lengthen the pulse from 2 to 3 clock periods. This changes the coincidence resolution to 2.5 clock periods for accidentals and 2 clock periods for real events.

The output word is a 6 bit binary number which is the number of inputs that were true during the clock period 7 clocks earlier. If 48 inputs is sufficient, the 6 bit output word of the majority logic can be digitally compared with a CAMAC loaded register and produce a trigger signal directly.

For systems which require a larger majority logic unit, multiple 48 input majority logic units can be combined. The 6 bit output word can be time aligned (in clock steps) and added to the outputs of other majority logic modules, using another 2366 module. A single 2366 module configured as an adder tree can add eight 6 bit input words in 7 pipeline stages. The output word is limited to 8 bits in this example, so the sum will limit at 255 (there can be as many as 384 inputs). If counting to beyond 255 is required, The 2366 can be configured to have fewer inputs and a wider output word (in units of 4 bits).

Extending this to one more level (adding another 2366 to sum the outputs of the adders) allows six 8 bit inputs to be counted in 9 pipeline stages (again limiting at 255). This produces a 2304 input majority logic in a total of 25 pipeline stages (allowing one stage for going from module to module). When using multiple 2366 modules, a common clock should be used to synchronize the modules. Each 2366 has 3 inputs which can be used as the external clock source.

This majority logic program consumes most of the resources of the Xilinx chip. There is very little left with which to implement test features. A trigger system for a large high energy physics experiment is usually very complex, with many modules and very, very many cables and connectors which combine the logic modules to produce the desired trigger algorithm. Testing a system like this usually requires looking at signals with oscilloscopes and logic analyzers. It also usually requires disconnecting cables. It is usually not possible to test the system completely in situ. This allows many possibilities for errors, and for undetected (until long after the run) problems such as intermittent or missing connections.

The 2366 universal logic module solves this problem by allowing COMPLETE reprogrammability of the trigger logic. This is dynamic reprogramming, on the fly, without moving a single cable of jumper anywhere in the system. Test logic can be downloaded over CAMAC in a second or two per module, completely replacing the trigger logic. This test logic can be as simple as read write registers which drive the cable connections (testing both the integrity of the connections AND verifying the topology) to complex pattern generators which test the trigger algorithms of other modules. When testing is finished (which could require several different downloads of test logic) the trigger logic is simply reprogrammed with the

trigger algorithms and the experiment is ready to run, with a much higher confidence level than normally found in trigger systems.

This reprogrammability also allows the trigger to be easily and quickly changed for new requirements. If the topology is the same, or a superset of the previous topology, then switching back and forth between the old and new triggers can be done quickly AND safely, without touching a cable.

CAMAC Function Codes

F0, A0	Read enable mask for bits 1-24
F0, A1	Read enable mask for bits 25-48
F0, A2	Read CLOCK select register, bits 1-8
F16, A0	Write enable mask for bits 1-24
F16, A1	Write enable mask for bits 25-48
F16, A2	Write CLOCK select register, bits 1-8

CLOCK select register

0 = internal 10 MHz clock
1 = internal 20 MHz clock
2 = internal 40 MHz clock
3 = external clock, pair B17
4 = external clock, pair C17
5 = external clock, pair D17

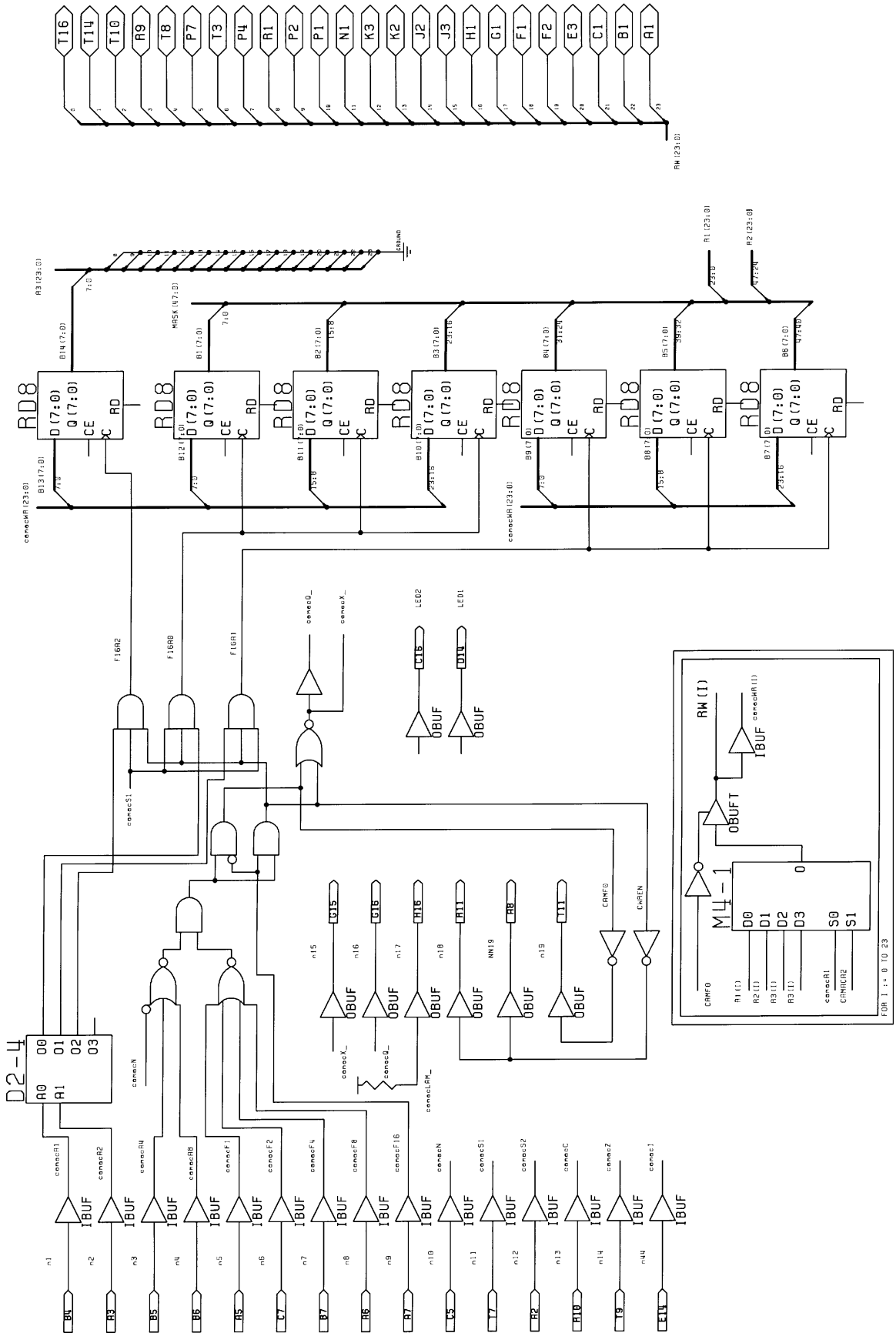
T16
T14
T10
A9
T8
P7
T3
P4
R1
P2
P1
N1
K3
K2
J2
J3
H1
G1
F1
F2
E3
C1
B1
A1
RD8
D(7:0)
Q(7:0)
CE
RD
F16A2
F16A0
F16A1
LED2
LED1
OBUF
IBUF
D2-4
A0
A1
O0
O1
O2
O3
M4_1
D0
D1
D2
D3
S0
S1
OBUFT
FOR I := 0 TO 23

Figure 18

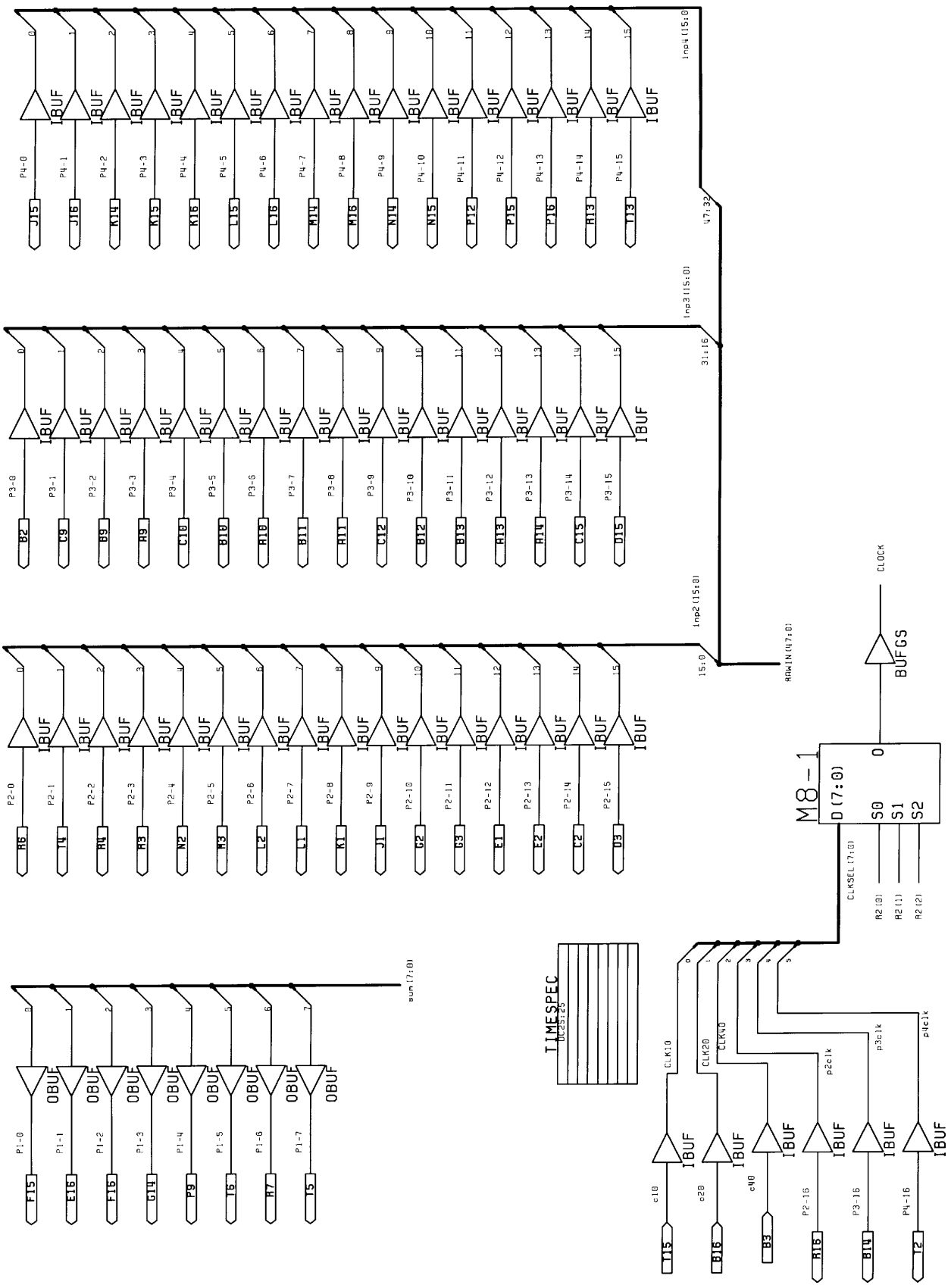
inp4[15:0]
inp3[15:0]
inp2[15:0]
RAWIN[47:0]
47:32
31:16
15:0
CLOCK
BUFGS
M8_1
D[7:0]
S0
S1
S2
CLKSEL[7:0]
TIMESPEC
sum[7:0]
IBUF
OBUF

Figure 19

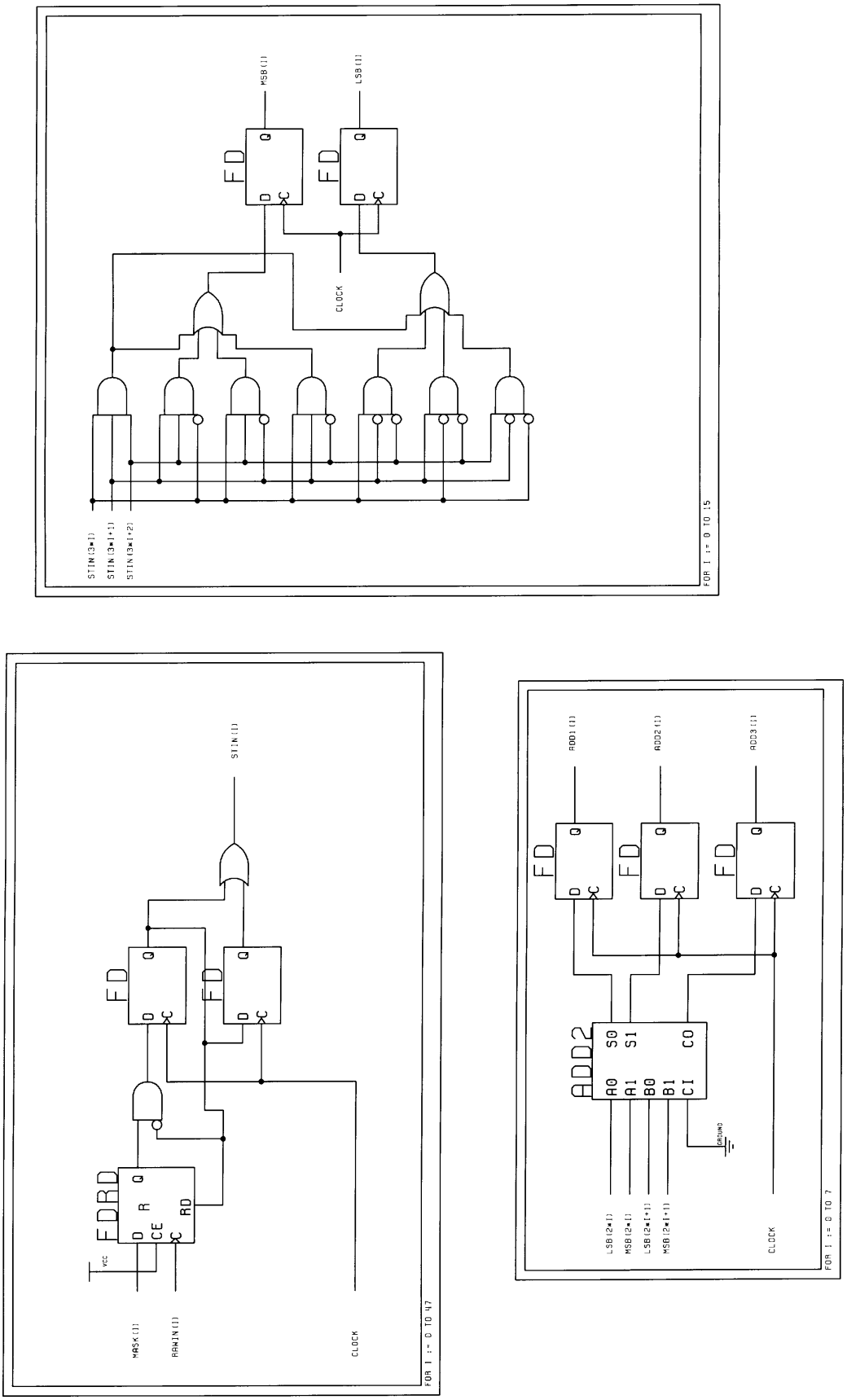
FOR I := 0 TO 15
MSB(I)
LSB(I)
FD
CLOCK
STIN(3*I)
STIN(3*I+1)
STIN(3*I+2)
FOR I := 0 TO 47
STIN(I)
FDRD
MASK(I)
RAWIN(I)
CLOCK
FOR I := 0 TO 7
ADD1(I)
ADD2(I)
ADD3(I)
ADD2
A0
A1
B0
B1
CI
S0
S1
CO
GROUND
LSB(2*I)
MSB(2*I)
LSB(2*I+1)
MSB(2*I+1)

Figure 20

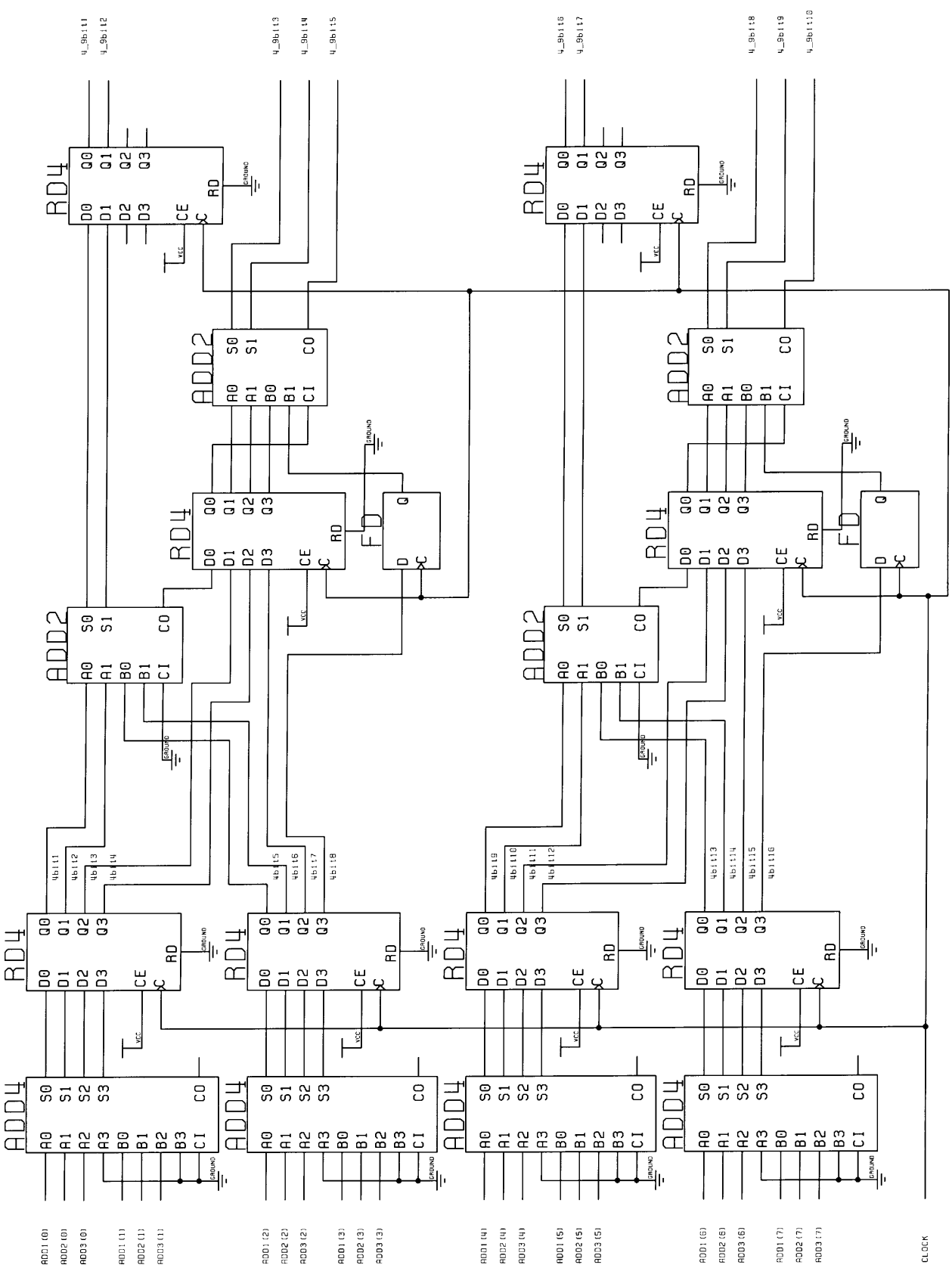

Figure 21

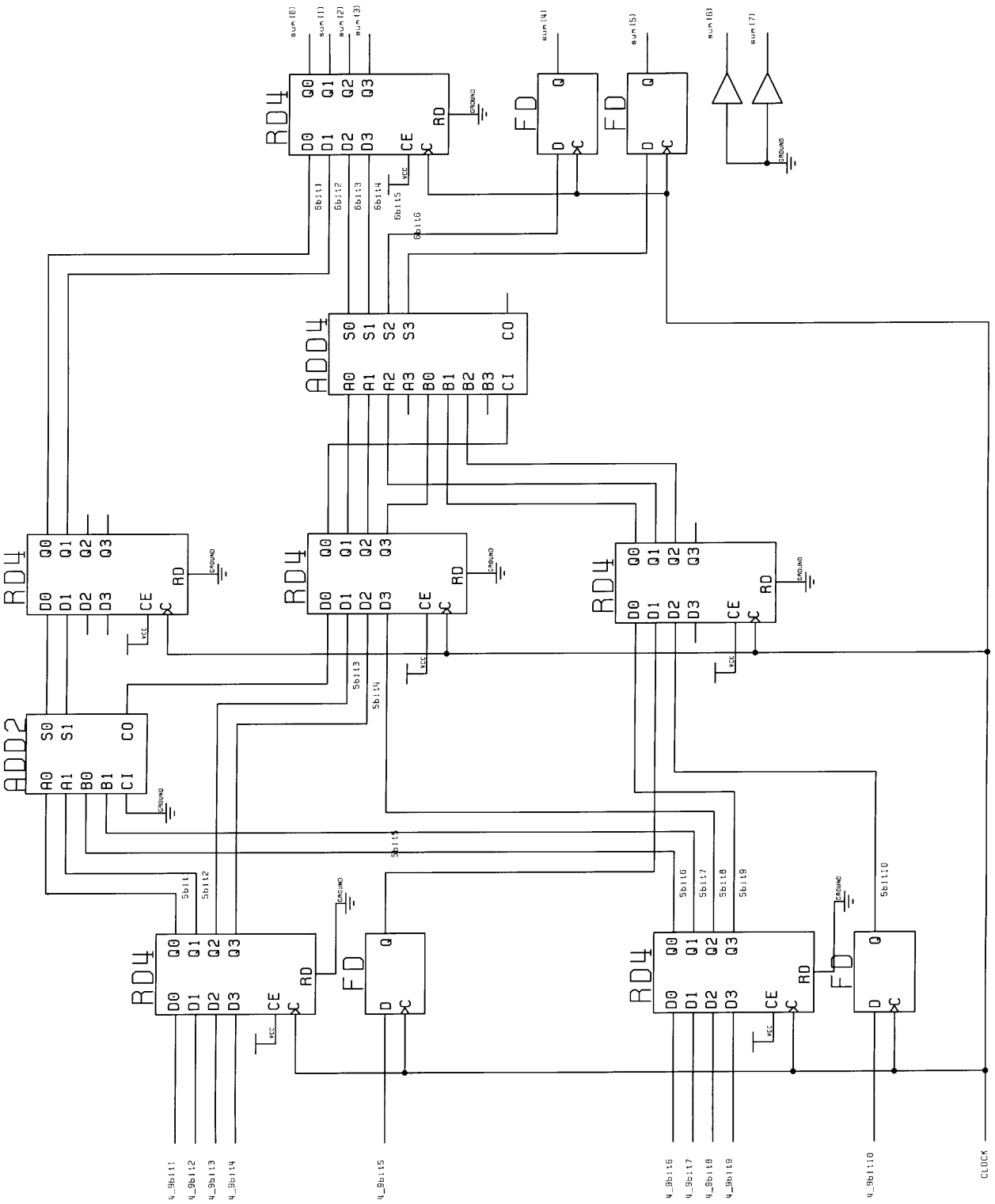

Figure 22

A LONG RANGE MULTIHIT TIME DIGITIZER WITH 100 NS RESOLUTION

The Xilinx bit file is SLOWTDC.BIT

The top connector on the 2366 ULM, I/O A, must be configured for input. The other connectors are not used and can be configured either as in or out without damage to the module.

The START pulse is the OR of I/O A pins 1, 2, 3 and 4.
The HIT pulses are the OR of I/O A pins 5, 6, 7 and 8.
The minimum width for the START and HIT inputs is 10 nsec.
The inputs are edge triggered and inverted, a negative edge (at the left hand pin) is the time to be measured.

The module should be cleared with F9, and enabled with F26. The first START pulse is synchronized with the internal 40 MHz clock. Then the HIT inputs, the 24 bit counter and a divide by 4 counter (with 40 MHz input) are enabled. The divide by 4 provides a 10 MHz clock for the 24 bit counter and the data storage FIFO. The clock remains running until the next F9. Note that the START time is quantized with the 40 MHz clock, and resets the phase of the 10 MHz clock.

When a HIT arrives, the pulse is synchronized with the 40 MHz clock and triggers an updating one shot with 100 nsec output width. The one shot can retrigger after 50 nsec.

The one shot output is synchronized with the 10 MHz clock and causes the contents of the 24 bit counter to be stored in the FIFO. The FIFO can hold up to 31 HIT times.

Note that the HIT time is quantized with the 10 MHz clock. The RMS quantizing error for the time difference, HIT - START, is 30 nsec, the sum in quadrature of the START error and the HIT error. The time zero must be calibrated, of course.

The module must be disabled with F24,A0 before readout. The data is read from the FIFO with F0,A0. This command returns Q true if there was a valid data word in the FIFO.

CAMAC Function Codes

F9,A0	Clear data, reset clocks, disable inputs.
F26,A0	Enable module (arm the start logic).
F27,A0	Test Enabled state.
F27,A1	Test FIFO not empty.
F27,A2	Test Start pulse received.
F24,A0	Disable Inputs (stop acquisition, put end mark in FIFO).
F0,A0	Read one data word from FIFO, Advance FIFO.

The proper sequence is:
F9, F26
a start pulse
up to 31 timing pulses within 1.6 seconds after the start
F24
F0 until no Q
The LSB of the 24 bit data is 100 nsec

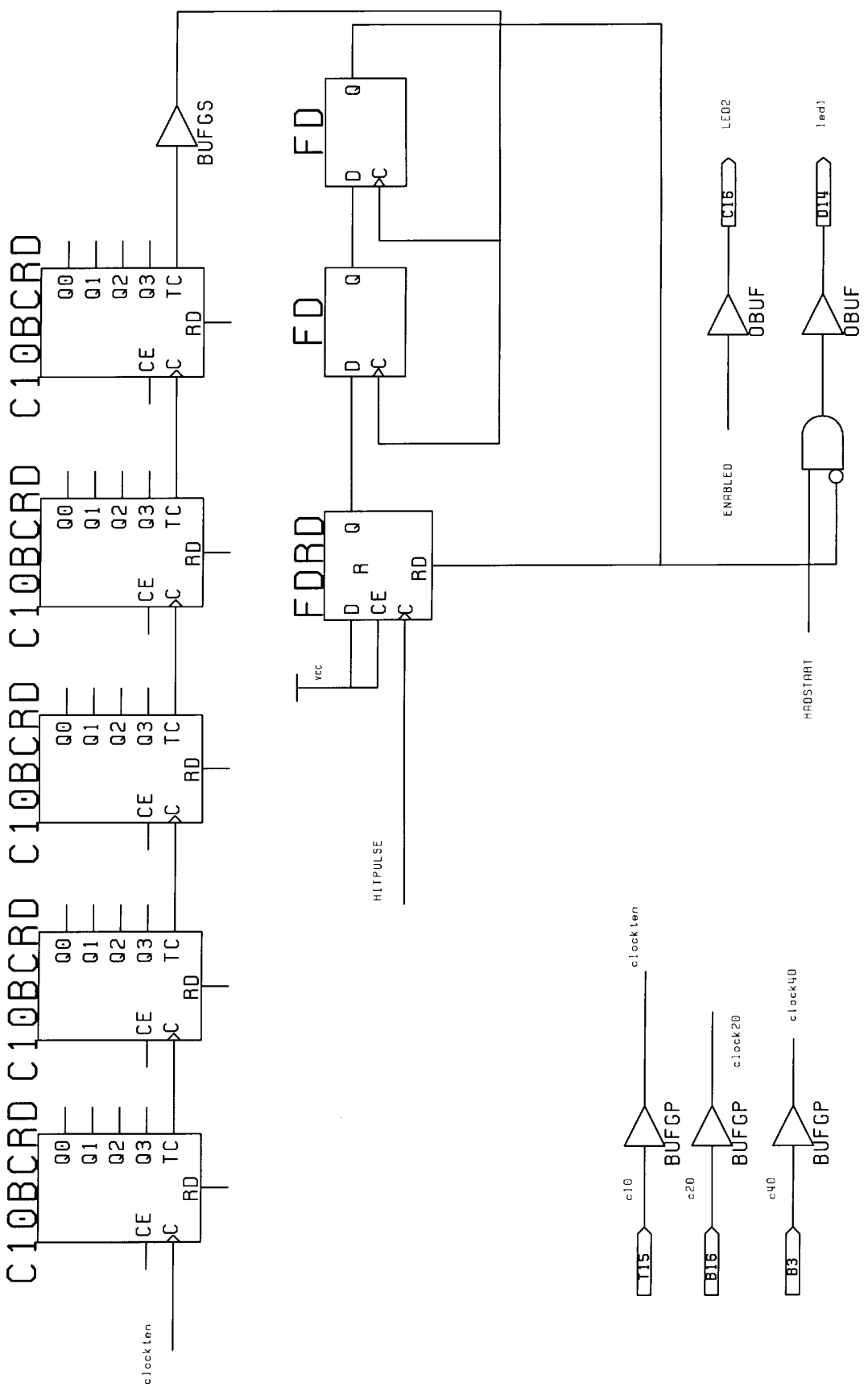
C10BCRD
C10BCRD
C10BCRD
C10BCRD
C10BCRD
Q0
Q1
Q2
Q3
TC
CE
C
RD
clocken
BUFGS
FD
D
Q
C
FDRD
R
CE
vcc
HIPULSE
ENABLED
OBUF
LED2
C16
led1
D14
HADSTART
clocken
clock20
clock40
BUFGP
c10
c20
c40
T15
B16
B3

Figure 23

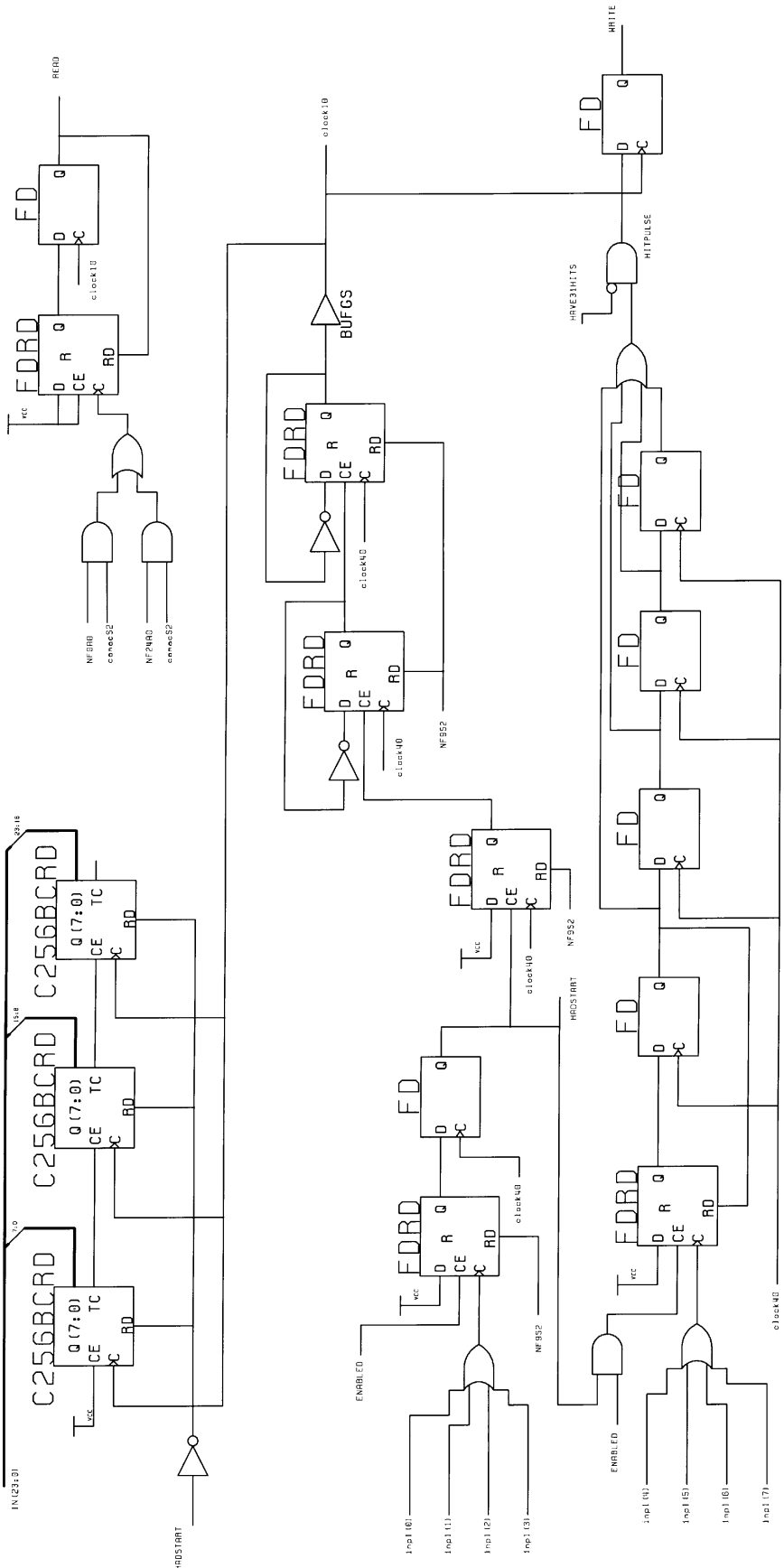
READ
WRITE
FD
FDRD
BUFGS
C256BCRD
clock10
clock40
HITPULSE
HAVE3HITS
NF952
NF580
NF2940
conac52
ENABLED
HADSTART
IN(23:0)
Inpl(0)
Inpl(1)
Inpl(2)
Inpl(3)
Inpl(4)
Inpl(5)
Inpl(6)
Inpl(7)

Figure 24

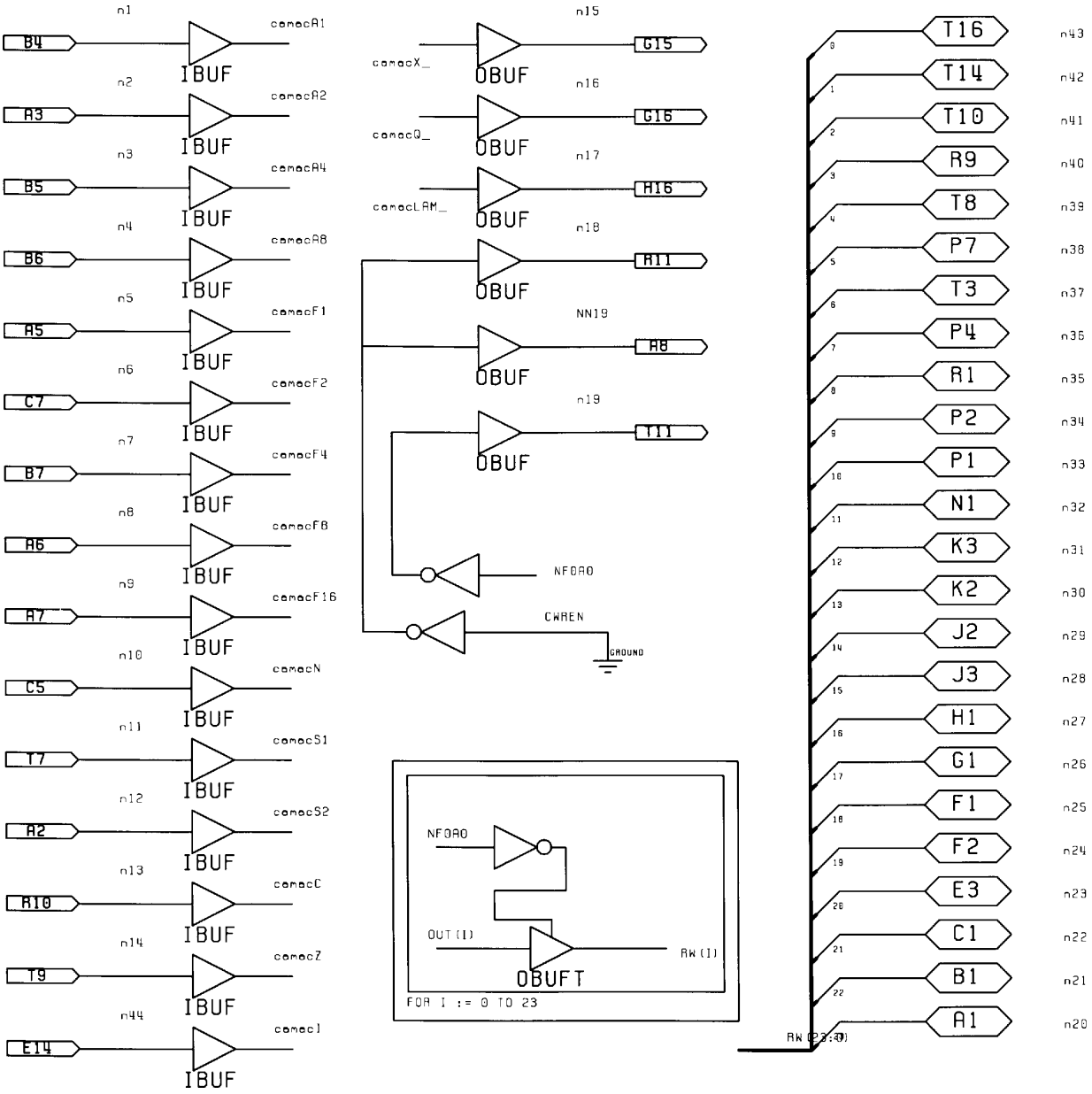
B4
A3
B5
B6
A5
C7
B7
A6
A7
C5
T7
A2
R10
T9
E14
IBUF
camacA1
camacA2
camacA4
camacA8
camacF1
camacF2
camacF4
camacF8
camacF16
camacN
camacS1
camacS2
camacC
camacZ
camacI
camacX_
camacQ_
camacLAM_
G15
G16
H16
R11
A8
T11
OBUF
NN19
NFDA0
CWREN
GROUND
OBUFT
OUT(I)
RW(I)
FOR I := 0 TO 23
T16
T14
T10
R9
T8
P7
T3
P4
R1
P2
P1
N1
K3
K2
J2
J3
H1
G1
F1
F2
E3
C1
B1
A1
RW(23:0)

Figure 25

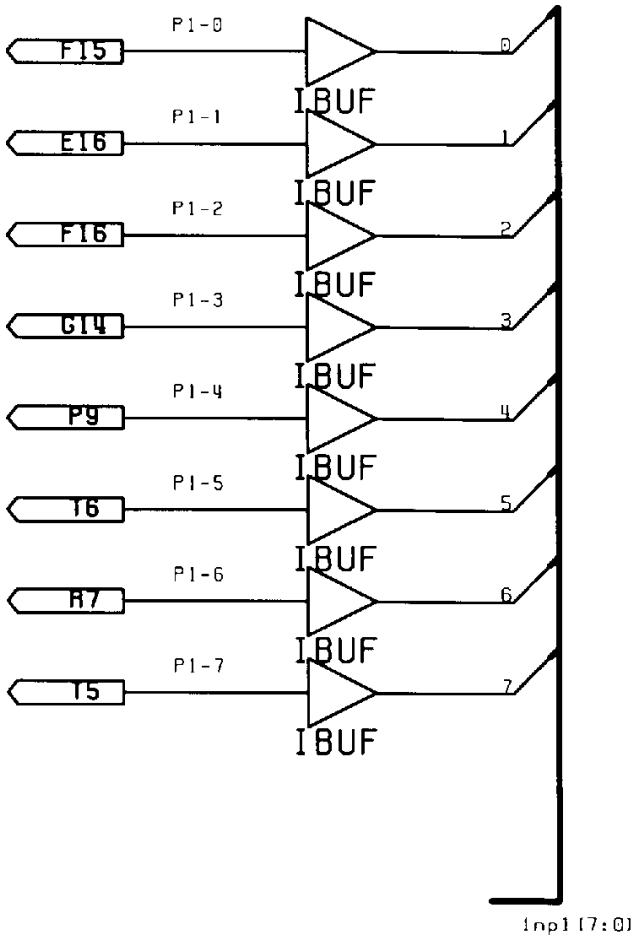
P1-0
F15
IBUF
0
P1-1
E16
IBUF
1
P1-2
F16
IBUF
2
P1-3
G14
IBUF
3
P1-4
P9
IBUF
4
P1-5
T6
IBUF
5
P1-6
R7
IBUF
6
P1-7
T5
IBUF
7
inp1[7:0]

Figure 26

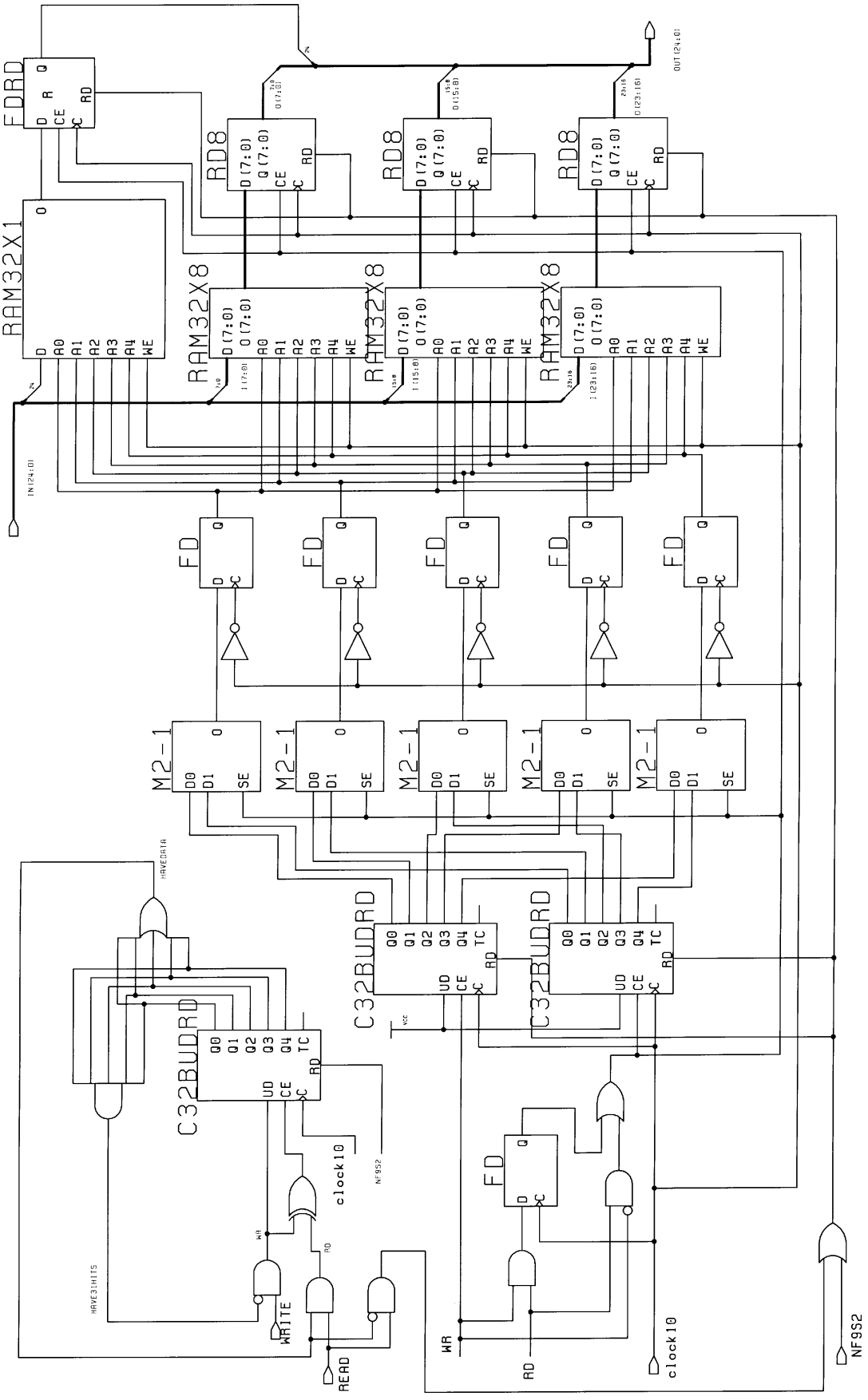
FDRD
RD8
RAM32X1
RAM32X8
FD
M2_1
C32BUDRD
clock10
WRITE
READ
WR
RD
NF952
HAVEHIA
HAVEHITS
OUT(24:0)
IN(24:0)

Figure 27

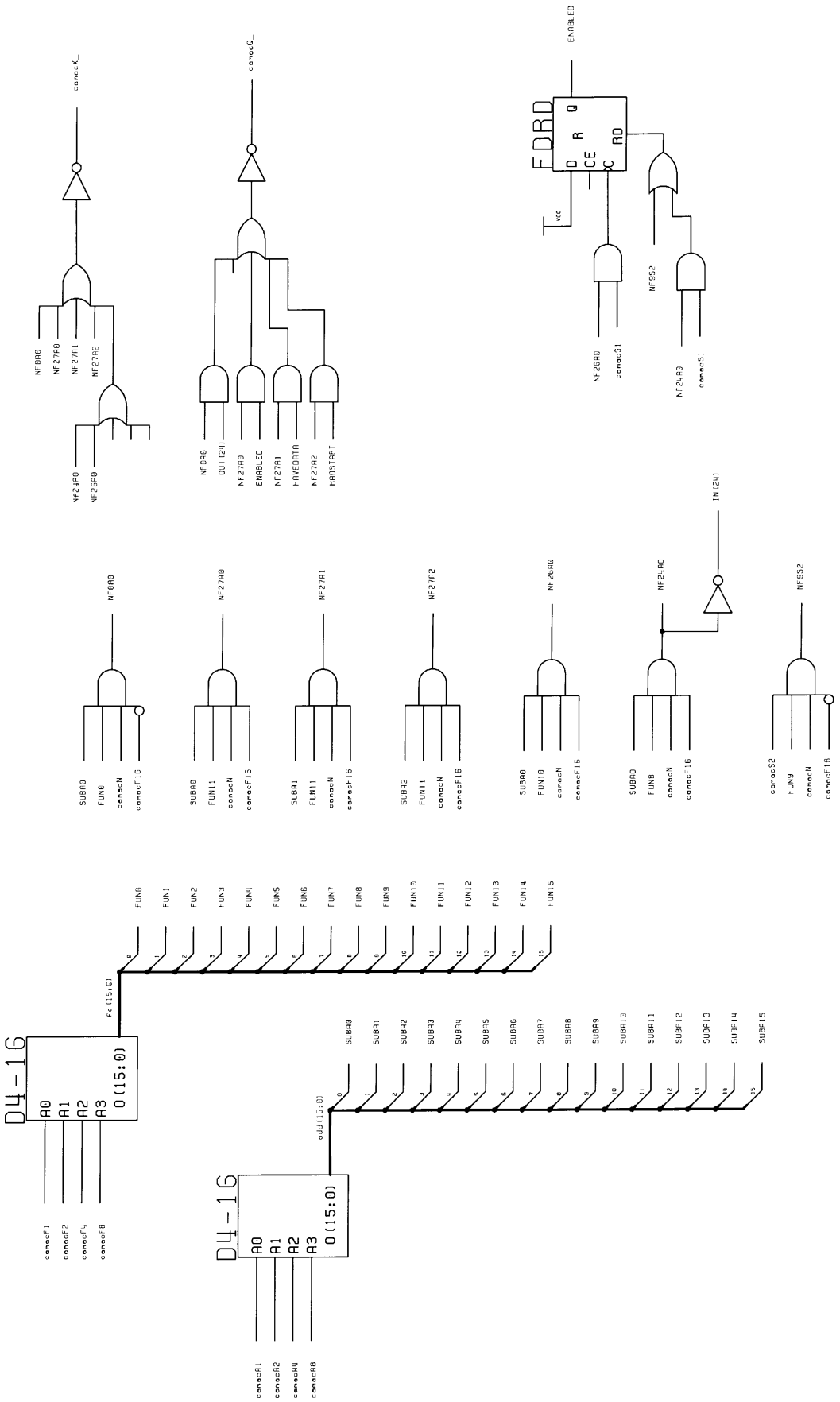
FDRD
D
CE
C
R
Q
RD
ENABLED
vcc
NF952
NF26A0
NF24A0
NF6A0
NF27A0
NF27A1
NF27A2
NF24A0
NF26A0
NF6A0
OUT(24)
NF27A0
ENABLED
NF27A1
HAVEDATA
NF27A2
HARDSTART
SUBA0
FUN0
FUN1
FUN2
FUN3
FUN4
FUN5
FUN6
FUN7
FUN8
FUN9
FUN10
FUN11
FUN12
FUN13
FUN14
FUN15
Fc(15:0)
D4-16
A0
A1
A2
A3
O(15:0)
concF1
concF2
concF4
concF8
add(15:0)
SUBA0
SUBA1
SUBA2
SUBA3
SUBA4
SUBA5
SUBA6
SUBA7
SUBA8
SUBA9
SUBA10
SUBA11
SUBA12
SUBA13
SUBA14
SUBA15
concA1
concA2
concA4
concA8

Figure 28