A Module Maintenance Controller Prototype

BY

Jung-Cheun Lien

Technical Report No. CENG 90-14

June 1990

Electrical Engineering - Systems Department

University of Southern California

Los Angeles, CA. 90089-0781

A Module Maintenance Controller Prototype*

Jung-Cheun Lien
Department of EE-Systems
University of Southern California
Los Angeles, CA 90089-0781

Abstract

A Module Maintenance Controller (MMC) is a novel design that can test devices supporting the IEEE 1149.1 boundary scan standard and the interconnect among them. In this report we describe the implementation of an MMC prototype. Two ASICs have been designed and implemented. One contains the major building blocks of the MMC prototype. The other is a test chip employing the boundary scan architecture. Both chips were implemented using field programmable gate array technology. The MMC prototype has successfully executed the required test procedures for the test chip. Programs that describe these test procedures are easy to develop since they can be written in a high level languages such as BASIC. Compared to conventional ATEs, the MMC not only costs much less but also has some performance benefits.

*This work was supported by Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contract no. N00014-87-K-0861. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Contents

1 Introduction **1**

2 Design and Implementation Environment **3**

3 Design and Implementation of the Test Channel Chip **5**

3.1 Design changes 5

3.2 Functions of the Test Channel 6

3.3 Major blocks of the Test Channel Chip 9

3.3.1 CR 9

3.3.2 HOST_IF 9

3.3.3 FSM 10

3.3.4 CNTERS 15

3.3.5 TMSBL 16

3.3.6 XR2 16

3.3.7 SR 17

3.3.8 Interrupt Circuit 17

3.3.9 CIRC 17

3.3.10 M8X4 18

3.4 Design For Testability Aspects 18

3.5 Simulation results of the Test Channel 19

3.6 Physical Characteristics of the Test Channel chip 19

4 Design and Implementation of the App1 chip **21**

4.1 Architecture of the App1 Chip 21

4.1.1 TAP controller 21

4.1.2 IR 21

4.1.3 Core 21

4.1.4 APOL 22

4.2 Physical characteristics of the App1 chip 22

5 Data Bus Adapter **24**

6 Testing the MMC Prototype 26

7 Conclusions 29

8 Acknowledgment 30

A Schematic diagrams for the Test Channel 31

B Schematic diagrams for the App1 31

C Timing diagrams for the Test Channel 31

D Test programs in Turbo Basic 31

List of Figures

1 The architecture of an MMC. 1
2 Physical configuration of the prototype. 2
3 Major design steps for implementing as ASIC. 4
4 The architecture of the Test Channel 6
5 FSM state transition diagram. 11
6 State transition diagram for DTUR, DTCR and PTUR modes. 12
7 State transition diagram for PTCR mode. 13
8 State transition diagram for INS, RTEST, RSBUS and STBUS modes. . . . 14
9 The pinout of a 84 pin PLCC device. 20
10 The pinout of a 68 pin PLCC device. 23
11 The data bus adapter. 25

List of Tables

1 Operation modes of the Test Channel. 8
2 Initial values for counters in various operation modes. 8
3 Registers usage in various operation modes. 9
4 Addressable registers and associated control signals of the Test Channel. . . 10
5 FSM output signals activated in each state. 15
6 The order of the scan chain. 18
7 The pin assignment of the Test Channel chip. 20
8 Test operation modes for the App1 Chip. 22
9 Truth table for generating control signals. 23
10 The pin assignment of the App1 chip. 24
11 Primitives for test control programs. 27
12 Functional test vectors and expected results for the full adder. 29

1 Introduction

Boundary scan technique improves the testability of boards containing surface mounted devices [1]. However, different vendors may provide devices with incompatible boundary scan structures. To avoid this problem the IEEE has established a standard called IEEE 1149.1 [2], which guarantees the compatibility among devices that employing boundary scan. In this report, a boundary scan device is referred to as a device that supports the IEEE 1149.1.

A Module Maintenance Controller (MMC) can test boundary scan devices and the interconnect among them. It can also provide test data to the devices under test (DUTs) and analyze the test results. Thus, an MMC can completely test a board consisting of boundary scan devices. A design called a Test Channel greatly increases the performance of the MMC by efficiently accessing and controlling the built-in self-test (BIST) and design-for-testability (DFT) facilities availables on the DUTs. The architecture of the MMC design is shown in Figure 1.

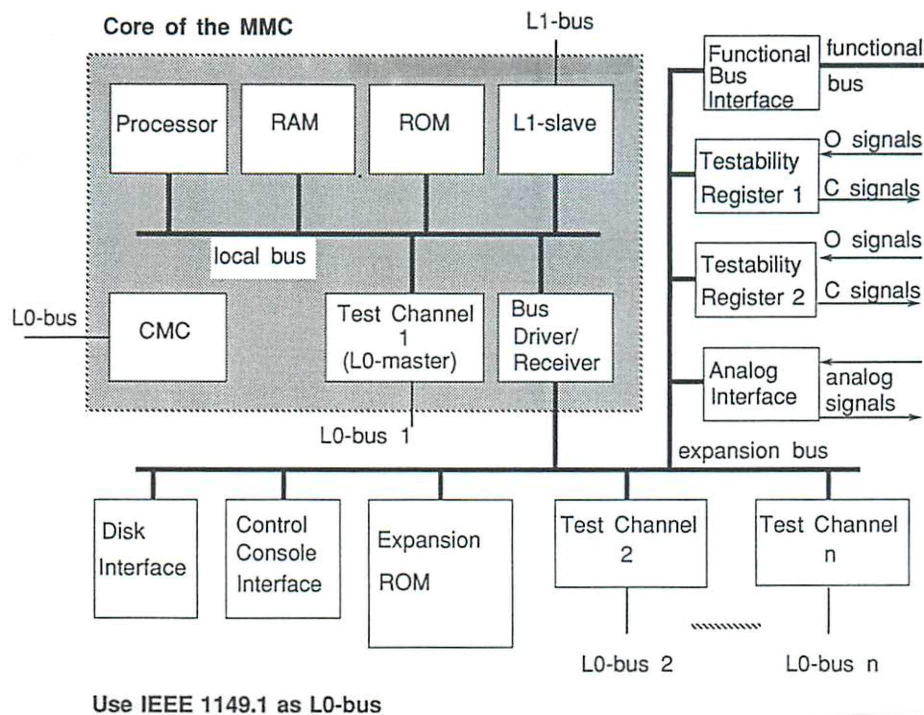


Figure 1: The architecture of an MMC.

In this report we describe the implementation aspect of the MMC prototype. A detailed description of the MMC design can be found in [3]. The major components of an MMC includes a processor, a memory unit and a Test Channel. An IBM AT computer is used as the host computer of the prototype. The physical configuration of the prototype is

illustrated in Figure 2 A board, which occupies a bus slot in the IBM AT, is used to provide an extension of the I/O bus. Another board, developed by Stanford University [4], is used to decode the bus signals and to accommodate the Test Channel chip. The Test Channel is implemented using the Actel field programmable gate array (FPLA) technology. The implementation was aided by the Actel Action Logic System, which automatically performs placement, routing and the programming of ACT 1020 devices.

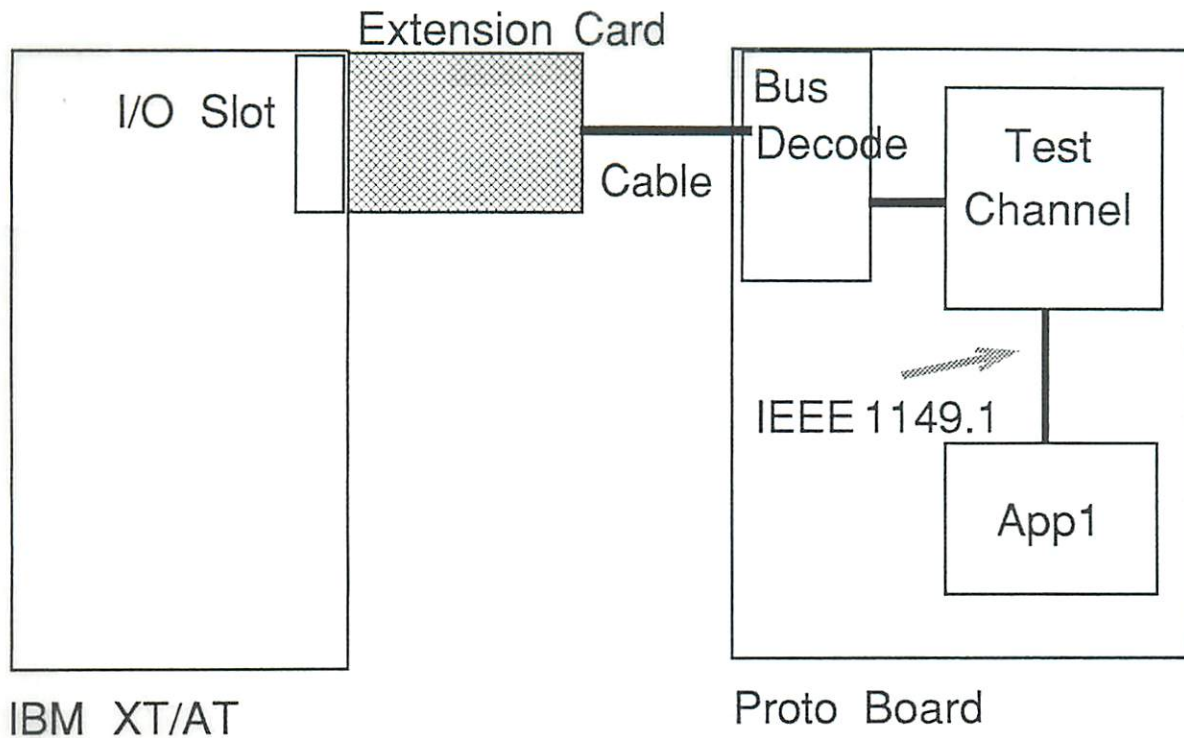


Figure 2: Physical configuration of the prototype.

Interfacing the Test Channel with the host processor requires little effort. Only the following signal lines need to be connected: a 16 bit bata bus, a 4 bit address bus, a chip enable line and two read/write control lines. These lines are available on the I/O bus of the host. Through these lines, the host can control the Test Channel by executing I/O read/write operations. Currently, control programs are written in BASIC. However, any language that provides the read/write to the absolute I/O address space can be used.

In order to verify the functions of the MMC prototype, an ASIC conformed with the IEEE 1149.1 is built. This chip, referred to as App1, has been successfully tested by the MMC prototype. During the testing of the App1, all functions of the MMC are exercised. Therefore, the prototype is tested functionally.

This report is organized as follows. The design and implementation environment for the prototype is described in Section 2. The implementation details of the Test Channel, which include design changes due to limited device capacity, the architecture, the DFT

aspects, simulation results and the physical characteristics, are described in Section 3. The implementation details of the Appl are described in Section 4. A data bus adapter for interfacing the 16 bits Test Channel to an 8 bits data bus is described in Section 5. The details for testing the MMC prototype is described in Section 6, followed by the conclusions in Section 7. A complete set of schematic diagrams of the Test Channel is presented in Appendix A, followed by the schematic diagrams for the Appl in Appendix B. Simulated timing waveforms for various operation modes of the Test Channel are listed in Appendix C. Test control programs for both Test Channel and Appl are listed in Appendix D.

2 Design and Implementation Environment

The major steps for implementing a circuit using the Actel FPGA technology include Functional Specification Design, Architecture Design, Logic Design, Testability Design, Schematic Entry, Logic Simulation, Device Selection, Pin Assignment, Validate, Place and Route, Timing/Delay Estimation, Device Programming and Device Testing. These steps are illustrated in Figure 3.

The high level design work, which includes the steps from Functional Spec Design to the Testability Design, is done manually. The low level design work is aided by a 386 PC running two major software systems, i.e., VIEWlogic and Action Logic System (ALS). VIEWlogic is a schematic capture, wirelist generation and logic simulation system. ALS not only allows a user to define I/O pins, but can also automatically perform the following tasks: the validation of the design, placement and routing, timing/delay parameters extraction, generation of the fuse file and programming of the device.

There are many problems involved in implementing a design. For example, there may be errors in the logic design; the design may be too large to fit into the selected device; the predicted timing/delay may not be acceptable. These problems will introduce one or more iterations in some of the design steps.

One drawback associated with this approach is that the software does not provide information about the number of the logic modules, which are the circuit primitives provided by the Actel devices, required for a design during the early stage of the design process. This information can only be provided in the validation step, where the entire circuit design is entered. If the circuit requires too many logic modules to fit in the selected device, major design changes may be required. The software should be improved to support the estimation of the number of logic modules whenever a building block is entered and simulated.

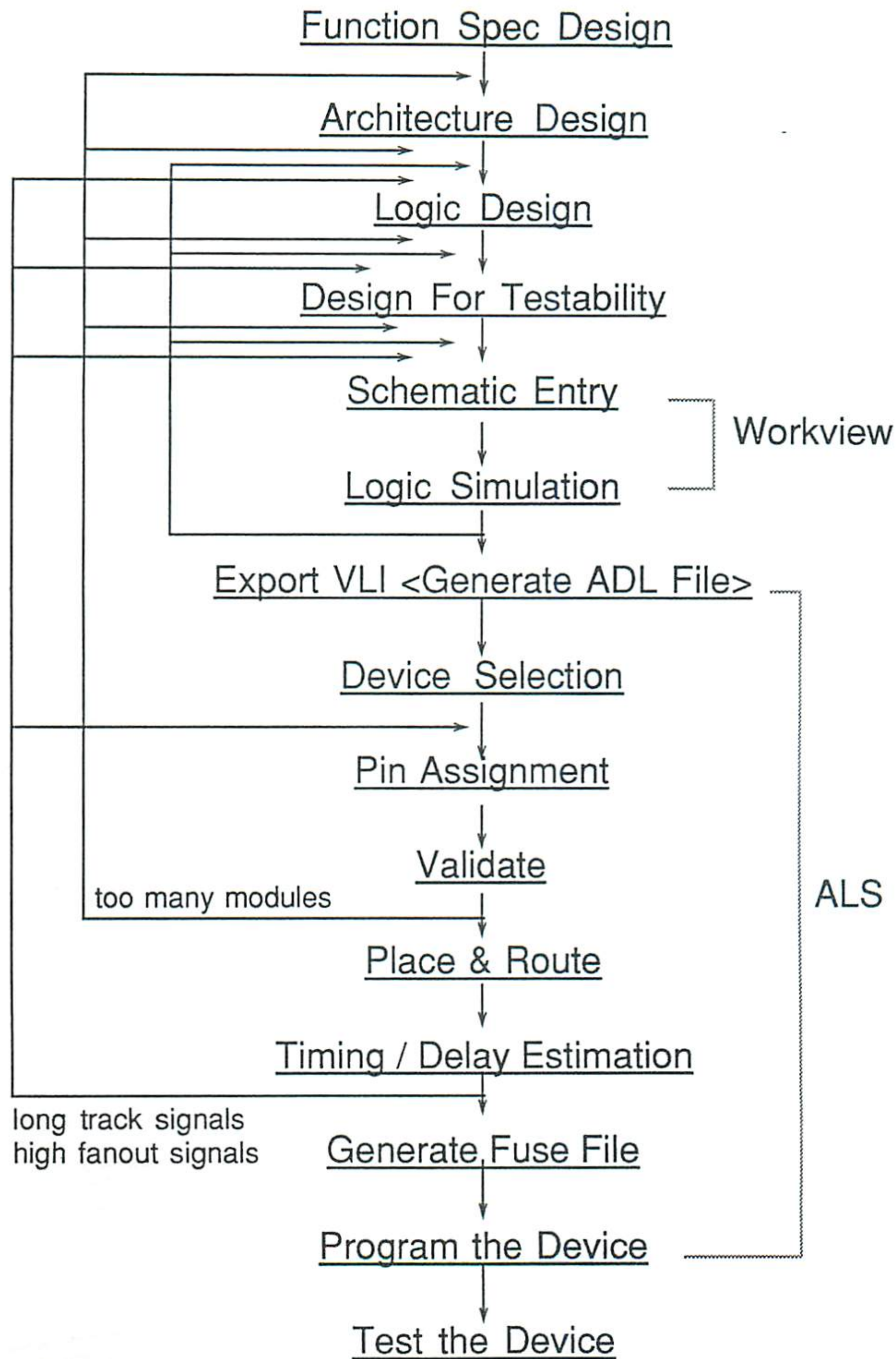


Figure 3: Major design steps for implementing an ASIC.

3 Design and Implementation of the Test Channel Chip

The major purpose of designing the Test Channel is to efficiently test chips that are conformed with the IEEE 1149.1 standard. A detailed description of the Test Channel design can be found in [3]. In this report we focus on the implementation aspects of the Test Channel, the design changes resulting from the limitations of the implementation and the enhancements on DFT.

Notations: Throughout this report, the following notation is used.

- Let SIGNAME be the name of a signal line, then /SIGNAME represents that the signal is active low, SIGNAME/O represents that the signal is a primary output, and SIGNAME/I represents that the signal is a primary input.
- The width of a bus is represented by its index. For example, PA[3:0] represents a bus consisting of four signal lines PA3, PA2, PA1 and PA0.

3.1 Design changes

Some design changes have been made in the implementation of the Test Channel. The main reasons for these changes are 1) the limited capacity of the device; 2) a change with the clocking scheme; and 3) the addition of DFT facilities. These design changes are listed below.

1. Registers PA and PB were not implemented. These two registers were used as (a) temporary storage for the output and input data, and (b) the feedback control of the two LFSRs (TxR and RxR). Without these two registers, the characteristic polynominals of the LFSRs (TxR and RxR) are therefore fixed.
2. The counter DC was not implemented. Therefore, the usage of the counters TC and SC has changed.
3. The length of the counters TC and SC was reduced from 22 and 12 bits to 12 and 4 bits, respectively.
4. A new block TMSBL was added. This block generates an arbitrary 6 bits sequence of values on the TMS line.
5. A new block Host_IF was added. This block is used to allow synchronization between the host and the Test Channel. Gated clocks can then be avoided.
6. The operation modes of the FSM were extended.

7. A scan chain was added for improving the testability of the design.

8. The FSM was modified to ensure correct operations during test mode.

The modified Test Channel is shown in Figure 4. In this figure the solid line boxes represent registers or sequential circuits; the dashed line boxes represent combinational circuits. Registers TxR, RxR and STR are related to the test bus signal lines. Register SR, CR, TC, SC and the sequential circuit sync are related to the control of the finite state machine FSM. Both data bus and address bus are buffered in the Test Channel. When executing an operation mode, the host first loads the proper data into these regsiters. The data transmission between the Test Channel and the test bus is then controlled by the FSM. The value of the SR indicates the current status of the operation. More details about the Test Channel are described next.

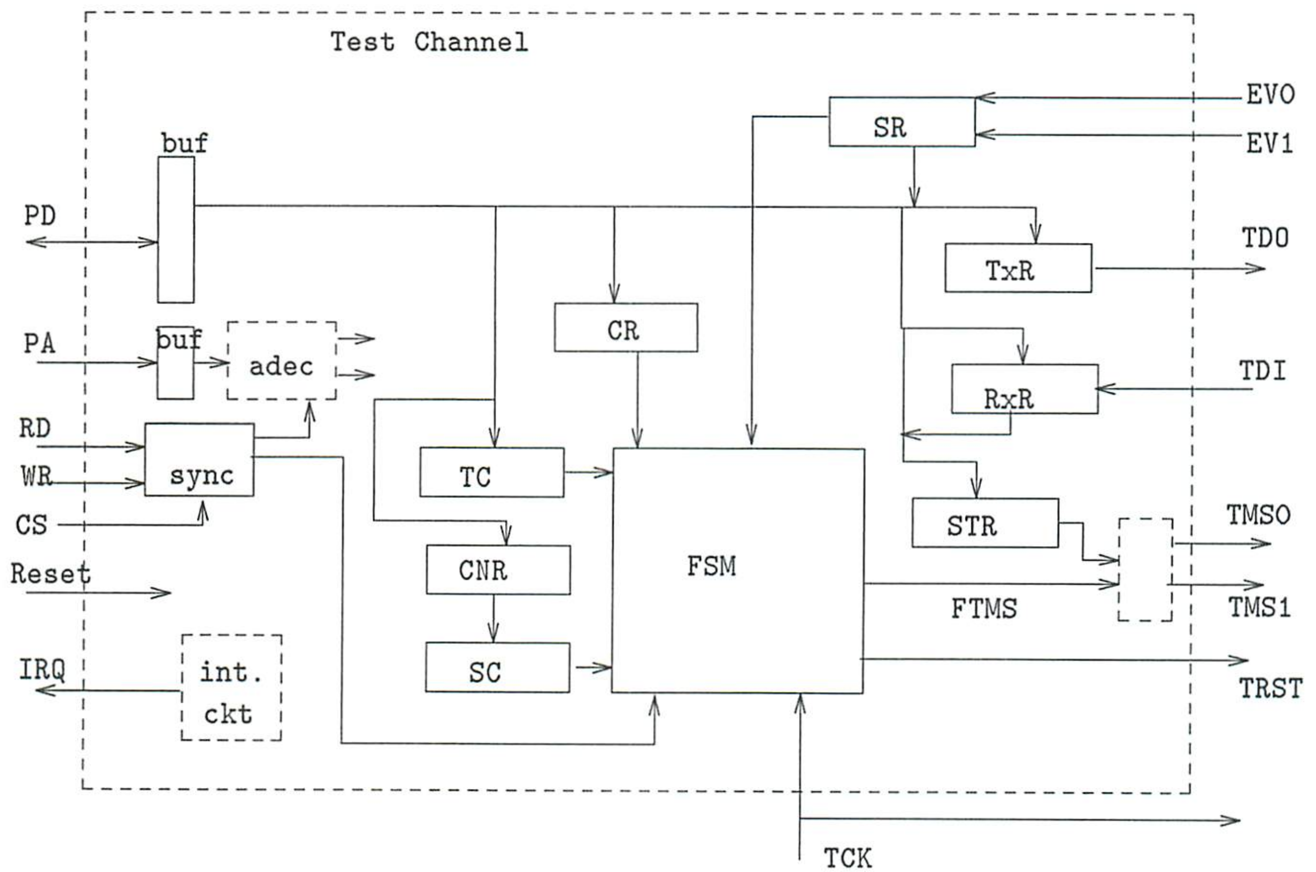


Figure 4: The architecture of the Test Channel

3.2 Functions of the Test Channel

The Test Channel can perform the following tasks:

1. control the bus state;
2. transmit instructions to and receive status from chips;
3. generate and transmit pseudorandom test data and receive and compact test results;
4. transmit deterministic test vectors to and receive test results from chips;
5. generate interrupts and also direct interrupts from chips to the host processor; and
6. keep count of the number of tests applied and the number of bits of each test or instruction transmitted.

From the above list it is easy to conclude that the Test Channel should be able to operate in any of the following eight modes, namely DTUR, DTCR, PTUR, PTCR, INS, RTEST, STBUS and RSBUS. These operation modes are described briefly below.

In the DTUR mode, deterministic test vectors are send to the chips under test; test results collected from these chips are sent to the host without compaction.

In the DTCR mode, deterministic test vectors are sent to the chips under test; test results collected from these chips are compacted. The resulting signature is then sent to the host for further analysis.

In the PTUR mode, test results (or status) of chips under test are collected and sent to the host without compaction. The collected test results are also sent back to the chips under test with a unit delay. In this way, the status can be read and data can be recirculated back to chips under test.

In the PTCR mode, pseudorandom test vectors are sent to the chips under test; test results collected from these chips are compacted. The resulting signature is then sent to the host for further analysis.

In the INS mode, instructions are sent to all the chips under test; the status of these chips are collected and sent to the host without compaction.

In the RTEST mode, all the chips under test are placed in a BIST mode for a pre-defined number of clock cycles.

In the STBUS mode, the value of the selected TMS lines (e.g., TMS0) are controlled by the content of the STR register. The TAP controllers of all chips under test must be in a stable state, such as the PAUSE or SHIFT state after the execution of this mode.

In the RSBUS mode, the signal /RST is activated for a pre-defined number of clock cycles. Test logic of the chips under test is reset.

The operation mode of the Test Channel is determined by a command register CR according to Table 1.

Two built-in counters and a register CNR are used to keep track of the operation of the Test Channel. The major counter TC is 12 bits long and the minor counter SC is 4

CR2	CR1	CR0	op mode
0	0	0	DTUR
0	0	1	DTCR
0	1	0	PTUR
0	1	1	PTCR
1	0	0	INS
1	0	1	RTEST
1	1	0	STBUS
1	1	1	RSBUS

Table 1: Operation modes of the Test Channel.

bits long. The control processor can load both TC and CNR with new values. The value of SC is loaded from CNR. CNR is used to restore the initial value of SC without help from the control processor. The usage of these counters depends on the operation modes of the Test Channel. Table 2 shows the initial values that need to be loaded prior to each operation. The value "t" represents the total number of test vectors to be applied. The value "s" represents the number of bits to be shifted in each test vector. The value "cnr" represents the current value in the CNR. A "-" indicates the don't care case. For most operation modes TC is used to keep track of the number of bits (to be shifted) in a test vector, and SC is used to keep track of the number of bits left in TxR and RxR. The host keeps track of the number of test vectors that have been applied. Note that this was done by TC in the previous design [3].

length	Register	DTUR	DTCR	PTUR	PTCR	INS	RTEST	STBUS	RSBUS
12	TC	s-2	s-2	s-2	t-1	s-2	t-1	s-1	s-1
4	CNR	14	14	14	s-2	14	-	-	-
4	SC	cnr	cnr	cnr	cnr	cnr	-	-	-

Table 2: Initial values for counters in various operation modes.

Three registers are directly related with the transfer of data to and from the test bus, namely TxR, RxR and STR. The host can write data into the TxR in parallel. The data in the TxR can then be shifted out to the primary output TDO serially. Data from the primary input TDI can be shifted into the RXR serially and then read by the host in parallel. The host can also write data into the STR in parallel. The data in the STR are then shifted to the primary output TMS0 or TMS1 during the STBUS operation mode. In all other modes, the TMS0 and TMS1 lines are controlled by the FTMS line from the FSM. Table 3 shows the function mode of each register for various operations.

length	Register	DTUR	DTCR	PTUR	PTCR	INS	RTEST	STBUS	RSBUS
16	TxR	shift	shift	TPG	TPG	shift	-	-	-
16	RxR	shift	SA	shift	SA	shift	-	-	-
6	STR	-	-	-	-	-	-	shift	-

Table 3: Registers usage in various operation modes.

3.3 Major blocks of the Test Channel Chip

The Test Channel contains 47 primary I/O pins (see Figure A.1). The top level building block of the Test Channel includes HOST_IF, CR, FSM, CNTERS, TMSBL, XR2, SR, CIRC and M8X4. Figure A.2 shows the connection among these building blocks. The functions of each building block are described next.

3.3.1 CR

The CR is a 6 bit register storing commands for the Test Channel. The host can write data into this register. The value of this register can also be altered during test mode since this register is a part of the scan chain. The lowest 3 bits, CR2, CR1 and CR0 are used to determine the operation mode of the FSM, which controls the operation of the Test Channel (see Table 1). The function of the other bits are listed next. CR5 controls the signal TT, which is used to enable/disable the FSM. The FSM is disabled when CR5=0. CR4 is used to enable one of the TMS0 and TMS1 lines. The TMS0 line is enabled when CR4=0. The CR3 is used to enable/disable the interrupt request signal IRQ. IRQ is disabled when CR3=0.

3.3.2 HOST_IF

This block allows the host processor to read and write the internal registers of the Test Channel. When reading an internal register, the content of the selected register is sent to the output data bus OD[15:0], which in turn is sent to the I/O pins PD[15:0] under the control of the signal OE (see Figure A.1). The signal OE is generated in this block by the host control signals /CS and /RD (see Figure A.3). Note that the host can read the internal registers at any time.

Writing to an internal register of the Test Channel requires a synchronization mechanism. This is because the host and the Test Channel are driven by separate clocks which are not in synchrony. The HOST_IF achieves synchronization via a handshaking scheme. When writing, the values on the data PD[15:0] and the address PA[3:0] lines are latched into the SYNDA block (see Figure A.5). A synchronization flip-flop (SYNFF) is then set to indicate that a write operation is pending. The setting of this flip-flop activates the signal AEN (see Figure A.6) at the next rising edge of the CLK. The signal AEN can

enable the address decoder ADEC to produce a proper control signal that will eventually load data into the register that is selected by the address bus PA[3:0]. The schematic of the ADEC is shown in Figure A.4. The ADEC allows the selection of registers according to Table 4. The loading of values to a selected register is controlled by a signal shown in the column "Asserted Signal" of the table.

Address	PA3	PA2	PA1	PA0	/CS	Register selected	Asserted Signal
-	0	0	0	0	1	none	none
0	0	0	0	0	0	CR	/CRam
1	0	0	0	1	0	CNR	/CNRam
2	0	0	1	0	0	STR	/STRam
3	0	0	1	1	0	none	none
4	0	1	0	0	0	TC	/TCLam
5	0	1	0	1	0	TxR	/TxRam
6	0	1	1	0	0	RxR	/RxRam
7	0	1	1	1	0	none	none
8	1	0	0	0	0	SYNR	/SYNRam
9	1	0	0	1	0	SR	/SRCLRam
10	1	0	1	0	0	none	/SOFTRSam

Table 4: Addressable registers and associated control signals of the Test Channel.

The SYNFF is cleared immediately after a write operation is completed. The next write operation can then be accepted. The control signals generated by the address decoder (ADEC), such as /CRam, is a single negative pulse. Note that the signal AEN is part of the scan chain. During test mode, AEN is altered by the values that go through the scan chain. It is possible that erroneous write operations may be generated. To avoid this problem all control signals generated by the ADEC are disabled during test mode. This is achieved by ANDing the signal AEN with the signal TEST (see Figure A.3).

A FSM control signal FEN is also generated by this block. The register SYNR, which controls FEN, can be set and reset by the host. Writing the value 1 into a selected address, i.e., PA[3:0]=1000, will set this register and thus activates the signal FEN. Similarly, writing a 0 into the same address will reset this register and thus deactivates FEN. Through the setting and resetting of this register, the FSM can be controlled.

3.3.3 FSM

Figure A.9 shows the finite state machine (FSM) of the Test Channel. The FSM contains five major blocks, namely CRDEC, NSDEC, STATE, OL and POST_OL. The CRDEC is a 3 to 8 demultiplexer (see Figure A.10). Only one of 8 output signals is active at a time.

These signals determine the operation mode of the FSM. For example, if the signal DTUR is active then the FSM is operated in the DTUR mode.

The state register of the FSM is a 5 bit register, referred to as STATE or PS, can represent 32 states. However, there are only 23 valid states labeled as S0,...,S22 in this machine. All other states are invalid and they have the same next state (S0) regardless the current control inputs. The FSM has 8 control inputs, 5 state variables and 7 outputs. The control inputs are CR2, CR1, CR0, TT, FEN, TCTC, SCTC and SR2. The output signals are /SCLDsm, SCsm, TCsm, TxRsm, STRsm, FTMS, and /RSTsm. The present state are represented by PS[4:0]. Each state Si is encoded by the binary code of i. For example, S0 is encoded as PS[4:0]=00000 and S2 is encoded as PS[4:0]=00010.

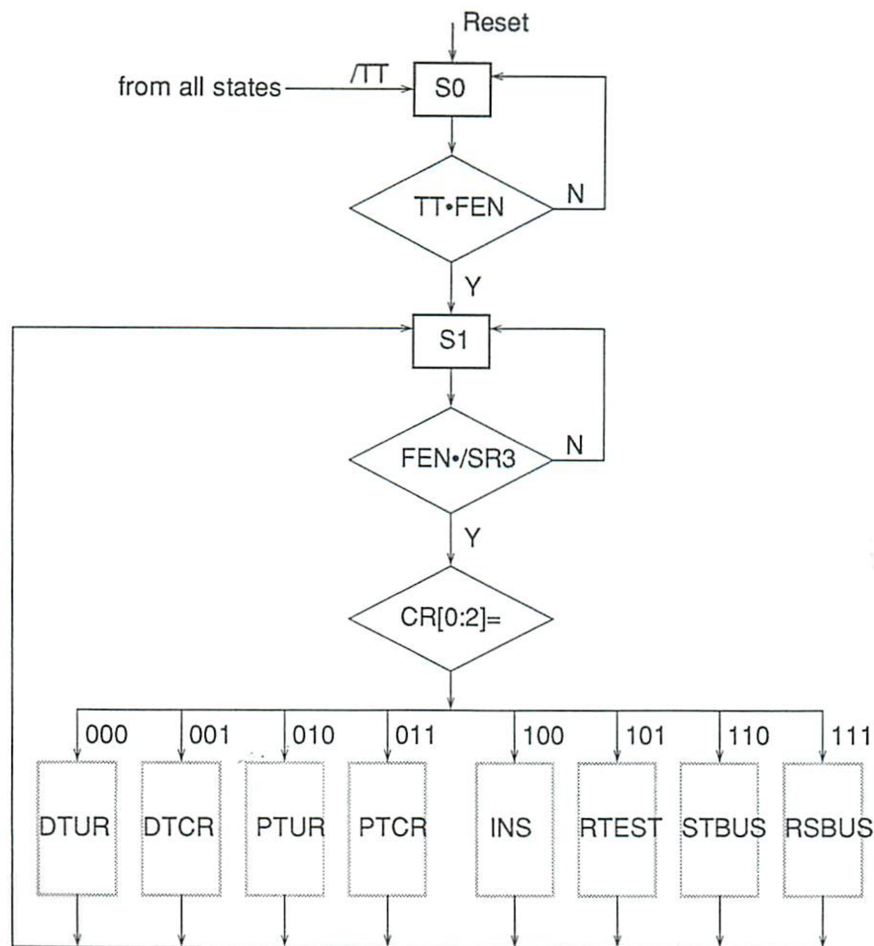


Figure 5: FSM state transition diagram.

The NSDEC is used to determine the next state for the FSM. The state transition diagram for the FSM is shown in Figure 5, 6, 7 and 8. Each decision box is represented by a combination of control inputs. For example, the box labeled with FEN*/SR3 represents the condition that FEN=1 and SR3=0. The state transition diagram contains 8 major branches, namely DTUR, DTCR, PTUR, PTCR, INS, RTEST, STBUS and RSBUS. Each

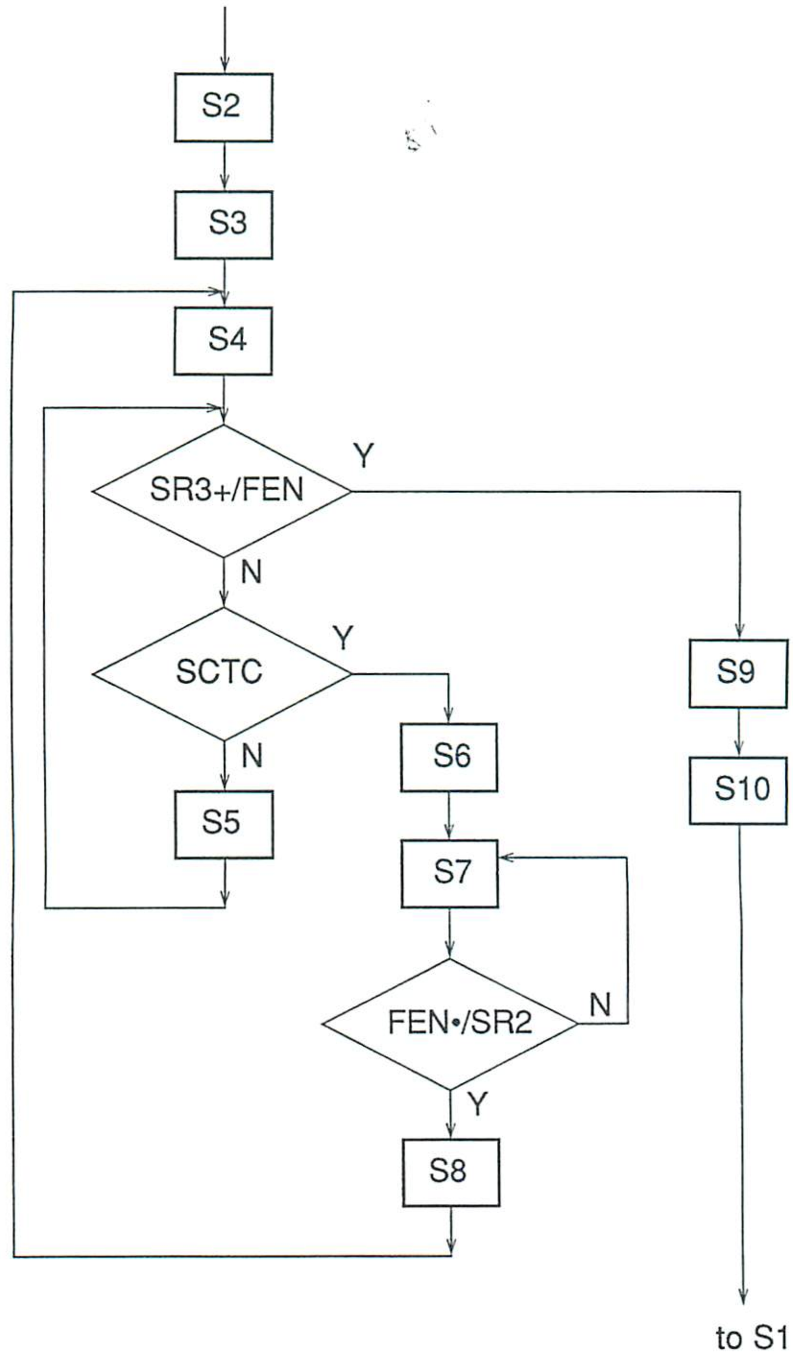


Figure 6: State transition diagram for DTUR, DTCR and PTUR modes.

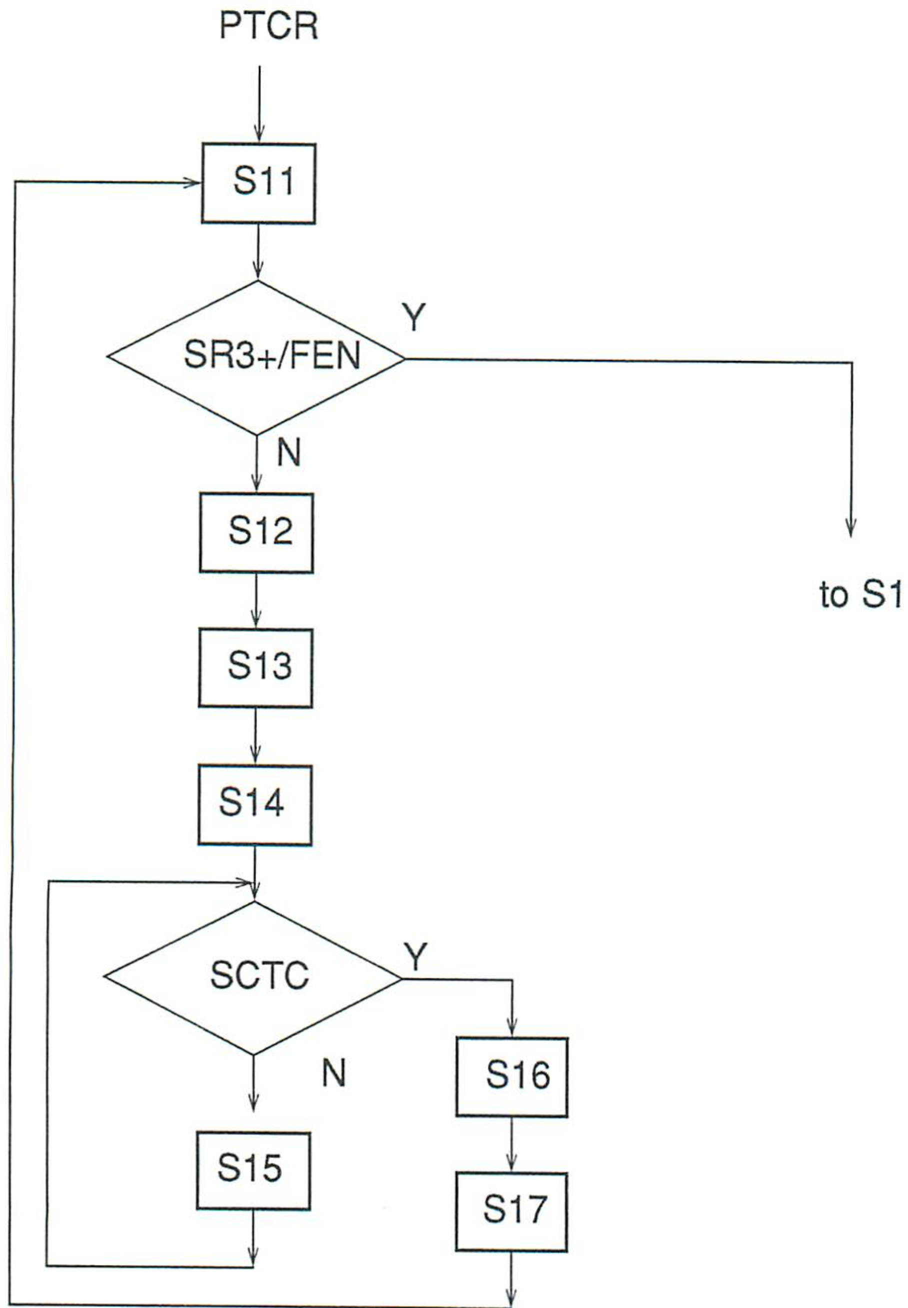


Figure 7: State transition diagram for PTCR mode.

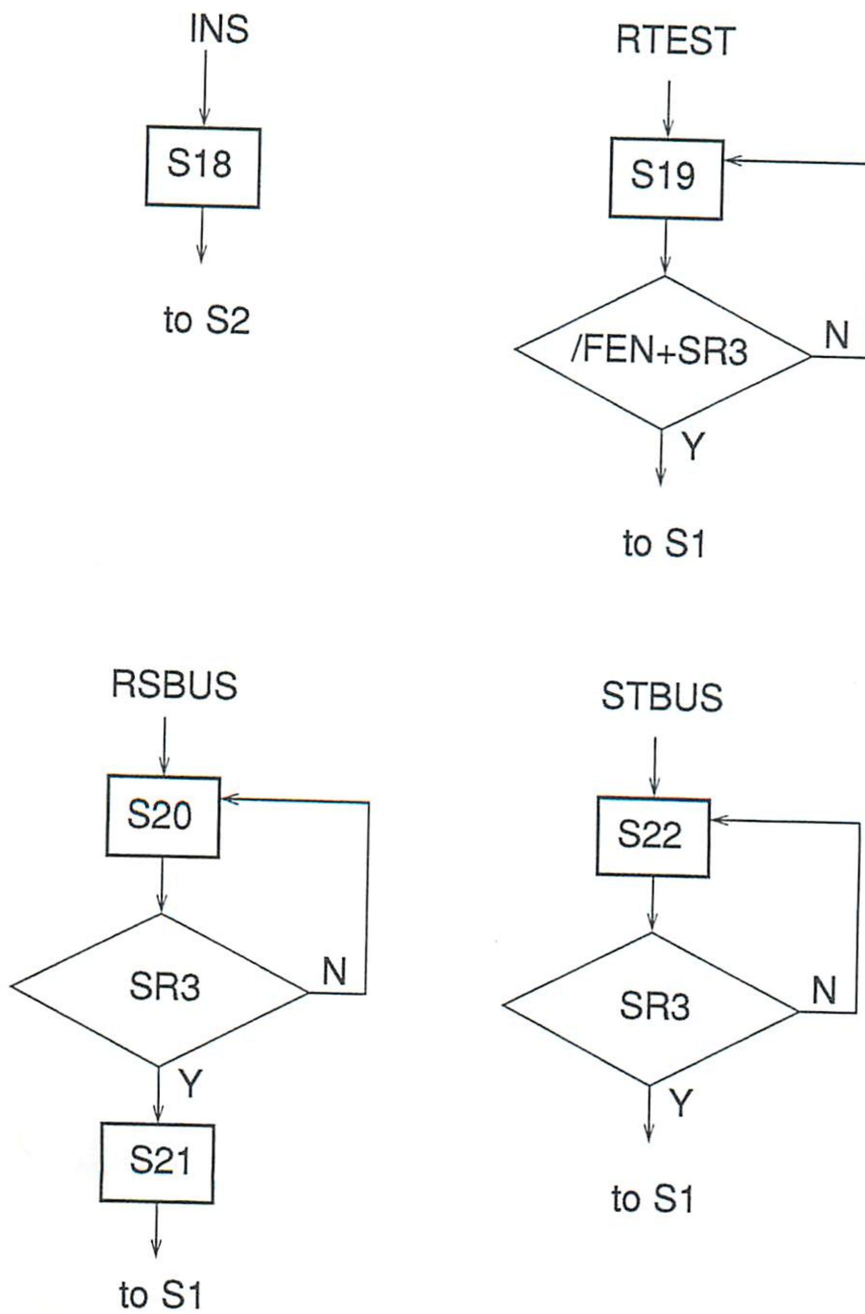


Figure 8: State transition diagram for INS, RTEST, RSBUS and STBUS modes.

branch corresponds to an operation mode. Note that DTUR, DTCR and PTUR share the same branch. This branch is also used during the INS mode. The control signals CR2, CR1, CR0, TT, FEN, SR2 and SR3 are controlled by the host via write operations. The host thus can control the FSM. For example, the host can terminate the operation of the Test Channel by resetting the signal TT. This will force the next state of the FSM to S0, which is an idle state.

The OL is used to generate output signals for each state. An output signal can be activated in one or more states. Table 5 shows the output signals that are activated for each state. For example, the signal /SCLDsm is active (=0) only in states S4 and S14. For other states /SCLDsm is inactive (=1). Similarly, the signal TCsm is active (=1) only in states S5, S6, S17, S19, and S20. For other states TCsm is inactive (=0).

State	Signals	State	Signals
S0	FTMS=1	S1	FTMS=0
S2	FTMS=1	S3	FTMS=0
S4	FTMS=0, /SCLDsm	S5	FTMS=0, TCsm, SCsm, TxRsm
S6	FTMS=0, TCsm, TxRsm	S7	FTMS=0
S8	FTMS=1	S9	FTMS=1, TxRsm
S10	FTMS=1	S11	FTMS=0
S12	FTMS=1	S13	FTMS=0
S14	FTMS=0, /SCLDsm	S15	FTMS=0, SCsm, TxRsm
S16	FTMS=1, TxRsm	S17	FTMS=1, TCsm
S18	FTMS=1	S19	FTMS=0, TCsm
S20	FTMS=1, TCsm, /RSTsm	S21	FTMS=1
S22	FTMS=1, STRsm, TCsm	others	FTMS=1

Table 5: FSM output signals activated in each state.

Note that the STATE register is part of the scan chain. Erroneous output signals may be activated during scan operation. To avoid this problem, a POST_OL is inserted to disable all output signals during test mode (see Figure A.14).

3.3.4 CNTERS

This block contains three components that are used to control the state transition of the FSM (see Figure A.15). These components are CNR, SC and TC. Only the CNR is part of the scan chain. The host can write to both the CNR and the TC directly. The CNR is used to store the initial value for the SC. The SC is a 4 bit programmable down counter that can generate a terminal count signal SCTC when its value is 0 (see Figures A.18, A.19). After the value of the SC reaches 0, the initial value can be restored by loading

the current value from the CNR. The TC is a 12 bit programmable down counter that can generate a terminal count signal TCTC when its value is 0 (see Figure A.17). Both SCTC and TCTC are used to control the FSM.

The usage of these counters has been shown in Table 2.

3.3.5 TMSBL

This block is used to generate the control signals for the test bus. These control signals include TMS0, TMS1 and /RST (see Figure A.20). The signal /RST will remain active as long as the signal /RSTsm is active. /RST is used to reset the slaves on the test bus. Both TMS0 and TMS1 are used to control the state of the test access port (TAP) of all slaves on the test bus. Only one of them can be active at a time. When the signal EN1 (which is driven by CR4) is active TMS1 will be active; otherwise TMS0 will be active. Two sources drive TMS0, namely the signal FTMS from the FSM and the signal Q5 from the STR. During the STBUS mode (CR[2:0]=110), TMS0 is driven by Q5. The content of the STR can be first written by the host and then shifted out to the TMS0 line. In this way the value of TMS0 can be controlled. Hence, the TAP controller state of the slaves on the test bus can be completely controlled. However, the TAP controller must be in a stable state at the end of the STBUS operation. The schematic of STR is shown in Figure A.21.

3.3.6 XR2

This block contains two Linear Feedback Shift Registers (LFSRs), namely the TxR and the RxR. The basic building block for both registers is a D-type flip-flop DFHL, which is shown in Figure A.7. The DFHL is in shift mode if the signal T is high; otherwise, it can be in either LOAD mode (/LD=0) or HOLD mode (/LD=1). The TxR can operate in any one of the five functional modes, namely LOAD, SHIFT, HOLD, TPG and CLEAR (see Figure A.25). During the TPG mode, the TxR is configured as a maximal sequence test pattern generator that has a characteristic polynominal as $f(x) = x^{16} + x^5 + x^3 + x^2 + 1$. It is necessary to load a non-zero seed into the register before entering the TPG mode. This can be done by a host write operation.

The RxR can operate in any of the five functional modes, namely LOAD, SHIFT, HOLD, SA, CLEAR (see Figure A.26). During the SA mode, the RxR is also configured as a serial signature analyzer that has the same characteristic polynominal as the TxR. It is necessary to load a known pattern into this register before entering the SA mode. This can be done in the CLEAR mode or by a host write operation.

During test mode, both the TxR and the RxR are in the scan chain. Since both registers already have SHIFT capability, only a multiplexer is required to include each of them in the scan chain.

3.3.7 SR

The status register SR is a 4 bit register (see Figure A.27) which has input signals EV0, EV1, SCTC and TCTC for SR0, SR1, SR2 and SR3, respectively. These signals drive the Preset line of the flip-flop instead of the D line. Once being set, a flip-flop can hold its content until a clear signal, which can be either /CLR1 or /CLR2, is activated.

Both EV0 and EV1 are used by the chips under test to indicate the occurrence of important events. SCTC is the terminal count signal of SC and TCTC is the terminal count signal of TC. The control processor can read SR at any time to determine the current status of the Test Channel.

During test mode, the SR is configured as a shift register which is part of the scan chain. The data inputs must be disabled during test mode to ensure proper shift operation. A specific data pattern thus can be loaded into the SR during test mode. The host cannot write an arbitrary pattern into the SR, but it can clear the SR by activating the signal /SRCLRam. The host can read the SR at any time without considering the synchronization problem.

3.3.8 Interrupt Circuit

Under certain circumstances, the Test Channel should inform the host about the occurrence of important events. The condition for activating the interrupt signal is $IRQ = (/TEST * SR3 + EV0 + EV1) * CR3$ (see Figure A.2). Note that the host can disable the interrupt by resetting the CR. To avoid erroneous operations, the interrupt generated by the signal SR3 is disabled during test mode. It is the user's responsibility to enable the interrupt during the operation modes that requires the interrupt to establish the handshaking operation between the host and the Test Channel. For example, while running the RTEST mode, the host must enable the interrupt such that it can be informed when the value of the TC is 0.

3.3.9 CIRC

The schematic diagram of the CIRC is shown in Figure A.22. The CIRC selects data from the RxR or the TDI line to drive the output TDO. Since TDO must be synchronized with the falling edge of the CLK, a falling edge flip-flop is used. During the PTUR mode the data in the TDI line is sent out to the TDO line after a CLK cycle delay. This recirculation facility can restore the original value of a scan chain in the slaves while letting the host read out its content.

3.3.10 M8X4

This circuit contains four 2 to 1 multiplexers. It selects the data to be output to the output data bus OD[15:0] from either the SR or the RxR. Since only 4 bits are multiplexed, the higher 12 bits are always taken from the RxR.

3.4 Design For Testability Aspects

Serial scan is a well suited DFT technique for designing testable circuits using the Actel FPLA Technology. This is due to that both a D type flip-flop (DFC1B) and a MUXed D type flip-flop (DFMB) are implemented by the same number of logic modules. Since the latter can supersede the former, a DFMB is used whenever a D type flip-flop is required. Scan capability thus is provided without adding extra logic modules.

Scan Registers:
In the Test Channel design, registers that are designed with DFC1B are replaced by DFMB. These registers include CR, STATE (in FSM), CNR (in CNTERS) and AEN (in SYNDA of HOST_IF). Furthermore, registers that have **shift** capability can be easily included in the scan chain. These registers include RxR, TXR (in XR2) and STR (in TMSBL). Other registers such as FEN (in HOST_IF) and SR are designed to have scan capability by adding extra logic modules.

A scan chain is formed during test mode, i.e., when the signal TEST is set high. Starting from primary input SI/I and ending at primary output SO/O, the scan chain consists of 60 scan cells in a predefined order (see Table 6).

```
SI/I ->
AEN -> FEN -> CR0 -> CR1 -> CR2 -> CR3 -> CR4 -> CR5 -> PS0 -> PS1 ->
PS2 -> PS3 -> PS4 -> CNR0 -> CNR1 -> CNR2 -> CNR3 -> STR0 -> STR1 ->
STR2 -> STR3 -> STR4 -> STR5 -> BSCELL -> SR0 -> SR1 -> SR2 -> SR3 ->
TxR0 -> TxR1 -> TxR2 -> TxR3 -> TxR4 -> TxR5 -> TxR6 -> TxR7 ->
TxR8 -> TxR9 -> TxR10 -> TxR11 -> TxR12 -> TxR13 -> TxR14 -> TxR15 ->
RxR0 -> RxR1 -> RxR2 -> RxR3 -> RxR4 -> RxR5 -> RxR6 -> RxR7 ->
RxR8 -> RxR9 -> RxR10 -> RxR11 -> RxR12 -> RxR13 -> RxR14 -> RxR15 ->
SO/O
```

Table 6: The order of the scan chain.

Non-scan Registers:
Some registers, such as TC and SC, use both data inputs of the DFMB. Another MUX is required to make them scannable. Therefore, these registers are not made scannable. The controllability and observability of these non-scan registers are achieved by reading and writing through the data bus. For example, the value of the TC can be set by loading new

data from data bus. The SC can be loaded from the CNR by seeding a proper value into the STATE register (in FSM).

The value of these non-scan registers may be affected during scan. Since their control signals are provided by the FSM, undesired control sequences may be generated when new values are shifted into the STATE register of the FSM. Test data that have been loaded into the non-scan registers may be altered. To alleviate this situation, all control signals should be suppressed during scan. The functional block POST_OL in the FSM serves this purpose.

3.5 Simulation results of the Test Channel

Only some of the important logic timing waveforms are included in this report. These timing waveforms show the behavior of some important signals in various operation modes. Figure C.1 shows part of the timing waveform during the DTUR mode. All primary inputs of the Test Channel except the data bus are represented by the signal bus INS. The value of INS is set properly so that the internal registers of the Test Channel can be loaded. The FSM then operates in the DTUR mode. The value on the line TMS0 is properly controlled so that the slaves on the test bus can be controlled. During data transmission, the value on the TxR is output to the signal line TDO. The high value on the signal line SCTC terminates the first data transmission, and sets low the signal TCSM. The signal line OE is used to enable the reading of internal registers. After loading appropriate values into the internal registers, the FSM is again enabled. The data transmission is terminated by the high value on the TCTC line.

Figure C.2 shows part of the timing waveform during the DTCR mode. Figure C.3 shows part of the timing waveform during the PTUR mode. Figure C.4 shows part of the timing waveform during the PTCR mode. Figure C.5 shows part of the timing waveform during the INS mode. Figure C.6 shows part of the timing waveform during the RTEST mode. Figure C.7 shows part of the timing waveform during the STBUS mode.

3.6 Physical Characteristics of the Test Channel chip

The Test Channel is implemented using an ACT1020 device which is packaged in a 84 pin Plastic Leaded Chip Carrier (PLCC). The module utilization of this device is high. 513 out of a total of 548 logic modules are used, and so are 47 out of 67 I/O modules. This chip can operate at 2.5 MHz and consumes less than 250 mW of power. Figure 9 shows the pinout of a 84 pin PLCC package. The pin assignment of the Test Channel chip is listed in Table 7.

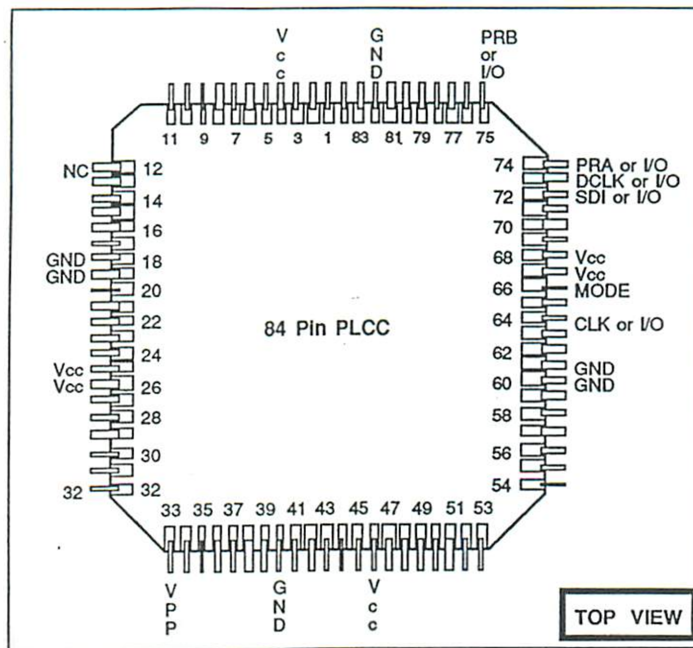


Figure 9: The pinout of a 84 pin PLCC device.

Pin	Name	I/O	Pin	Name	I/O	Pin	Name	I/O	Pin	Name	I/O
1	SI/I	I	2	IRQ/O	O	3	TDO/O	O	4	VCC	I
5	TDI/I	I	6	EV0/I	I	7	EV1/I	I	8	TMS0/O	O
9	TMS1/O	O	10	/RST/O	O	11	PA0	I	12	NC	-
13	PA1	I	14	PA2	I	15	PA3	I	16	PD0	I/O
17	PD1	I/O	18	GND	I	19	GND	I	20	PD2	I/O
21	PD3	I/O	22	PD4	I/O	23	PD5	I/O	24	PD6	I/O
25	VCC	I	26	VCC	I	27	PD7	I/O	28	PD8	I/O
29	PD9	I/O	30	PD10	I/O	31	PD11	I/O	32	PD12	I/O
33	VCC	I	34	PD13	I/O	35	PD14	I/O	36	PD15	I/O
37	SO/O	O	38	TEST/I	I	39	/RD/I	I	40	GND	I
41	/WR/I	I	42	/CS/I	I	43	RESET/I	I	44	NC	-
45	NC	-	46	VCC	I	47	AENG/O	O	48	PS1	O
49	PS0	O	50	PS1	O	51	PS2	O	52	PS3	O
53	PS4	O	54	DTUR	O	55	SCTC	O	56	SR3	O
57	SR2	O	58	NC	-	59	NC	-	60	GND	I
61	GND	-	62	NC	-	63	NC	-	64	CLK/I	I
65	NC	-	66	GND	-	67	VCC	I	68	VCC	I
69	NC	-	70	NC	-	71	NC	-	72	GND	I
73	GND	I	74	NC	-	75	NC	-	76	NC	-
77	NC	-	78	NC	-	79	NC	-	80	NC	-
81	NC	-	82	GND	I	83	NC	-	84	NC	-

Table 7: The pin assignment of the Test Channel chip.

4 Design and Implementation of the Appl chip

The Appl chip is used as part of the functional testing of the Test Channel. This chip conforms to the IEEE Std. 1149.1. The Appl chip contains a one bit full adder and a two-bit feedback register. The feedback register is scannable. There are 4 scan registers in this chip, namely Boundary Scan Register (BSR), Feedback Register (FB), Bypass Register (BPR) and Instruction Register (IR). These registers are separately scannable.

4.1 Architecture of the Appl Chip

Figure B.1 shows the top level schematics of the Appl chip. Many primary output pins are added for the enhancement of observability. These pins include the state variables of the TAP, namely A/O, B/O, C/O and D/O; the content of the IR, namely EXTEST/O, INTEST/O, SAMPLE/O, SCANFB/O and BYPASS/O. The major building blocks of the Appl include TAP controller, IR, CORE and APOL.

4.1.1 TAP controller

The TAP controller consists of two parts, namely APNS and APST. APST is a 4 bit register that stores the current value of the state variables D, C, B and A (see Figure B.2). Each state is represented by a binary value of DCBA, where D is the most significant bit. The TAP controller can be in any of the 16 states. APNS is the logic network that determines the next state of the TAP controller (see Figure B.3). The next state logic is derived from the state transition diagram defined in [2].

4.1.2 IR

The instruction register (IR) is a 3 bit registers (see Figure B.5). Each bit of the IR consists of two flip-flops (see Figure B.6). Note that unlike the proposed example shown in the IEEE 1149.1, the clock for the IR is not gated. The clocks for both flip-flops are driven by the primary input CLK directly. The control signals for the IR thus are generated differently. The content of the IR determines the test operation mode of the Appl according to Table 8

The status, which is latched into IR during the CaptureIR state, are 1, 0, and the value on the Enable line. Since the Enable line is active at this state, the correct status should always be 101.

4.1.3 Core

This circuit consists of the full adder, the boundary scan register and the feedback register (see Figure B.7). The full adder has 3 inputs, namely PD, PSUM and PCO, and 2 outputs,

IR0	IR1	IR2	op-mode
0	0	0	EXTEST
0	0	1	INTEST
0	1	0	SAMPLE
0	1	1	SCANFB
1	-	-	BYPASS

Table 8: Test operation modes for the App1 Chip.

namely NCO and NSUM. The full adder is connected to one primary input (DIN) and two primary outputs (CO and SUM). The connection is done via boundary scan cells. For example, DIN and PD are connected via an input scan cell, PSUM and SUM are connected via an output scan cell, and PCO and CO are connected via an output scan cell.

The boundary scan register (BSR) consists of 3 scan cells (JTCELL), which are associated with the primary inputs/outputs. The MUX control signal for the input cell (/BPI) are different from that of the output cells (/BPO).

The feedback register consists of 2 DRCELLs. Each DRCELL consists of 2 flip-flops. The clock inputs for these flip-flops are driven by the primary input CLK.

4.1.4 APOL

This circuit generates control signals for the CORE. The input signals for APOL are the TAP state and the IR. Unlike the example proposed in the IEEE 1149.1, there are no gated clocks used in this design. Therefore, we have modified the output logic design. Figure B.4 shows the schematics of the output logic APOL. Table 9 shows the truth table for generating the output control signals.

4.2 Physical characteristics of the App1 chip

The App1 chip is implemented using a 68 pin ACT1010 PLCC device. The chip can operate at 10 MHz. In the worst case the power consumed by this chip is less than 100mW. Figure 10 shows the top view of the device pinout. The pin assignment of the App1 is shown in Table 10.

		BSR					FB				BPR
op-mode	state	/BPI	/BPO	/E1	S	/E2	/FM	/E1	S	/E2	S
EXTEST	Capture	0	1	0	0	1	1	1	-	1	-
	Shift	0	1	0	1	1	1	1	-	1	-
	Update	0	1	1	-	0	1	1	-	1	-
INTEST	Capture	1	0	0	0	1	0	-	-	0	-
	Shift	1	0	0	1	1	0	-	-	0	-
	Update	1	0	1	-	0	0	-	-	0	-
SAMPLE	Capture	0	0	0	0	1	0	-	-	0	-
	Shift	0	0	0	1	1	0	-	-	0	-
	Update	0	0	1	-	0	0	-	-	0	-
SACNFB	Capture	1	0	1	-	1	1	0	0	1	-
	Shift	1	0	1	-	1	1	0	1	1	-
	Update	1	0	1	-	1	1	1	-	0	-
BYPAPSS	Capture	0	0	-	-	-	0	-	-	0	0
	Shift	0	0	-	-	-	0	-	-	0	1
	Update	0	0	-	-	-	0	-	-	0	0

Table 9: Truth table for generating control signals.

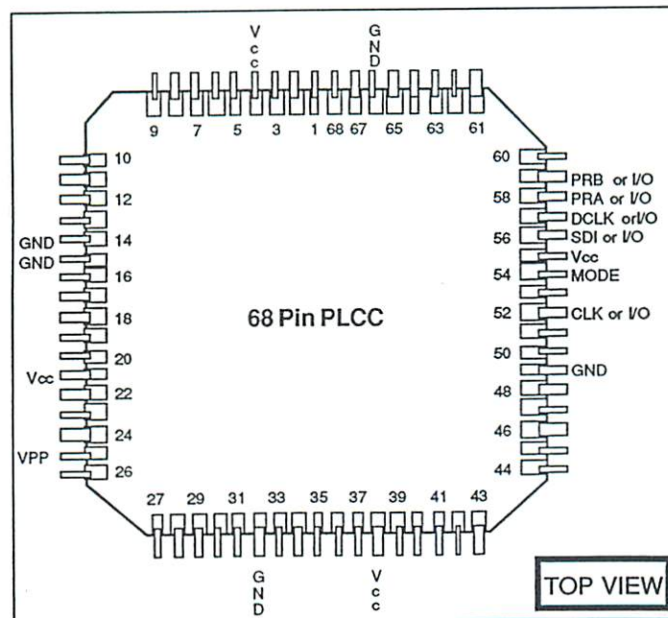


Figure 10: The pinout of a 68 pin PLCC device.

Pin	Name	I/O	Pin	Name	I/O	Pin	Name	I/O	Pin	Name	I/O
1	PD	O	2	PCO	O	3	SUM	O	4	VCC	I
5	CO	O	6	NSUM	O	7	NCO	O	8	DIN	I
9	NC	-	10	NC	-	11	NC	-	12	NC	-
13	NC	-	14	GND	I	15	GND	I	16	BYPASS	O
17	SCANFB	O	18	SAMPLE	O	19	INTEST	O	20	EXTEST	O
21	VCC	I	22	NC	-	23	NC	-	24	NC	-
25	VCC	I	26	NC	-	27	A	O	28	B	O
29	C	O	30	D	O	31	NC	-	32	GND	I
33	NC	-	34	NC	-	35	NC	-	36	NC	-
37	NC	-	38	VCC	I	39	NC	-	40	NC	-
41	NC	-	42	NC	-	43	NC	-	44	TDO	O
45	TDI	I	46	TMS	I	47	/TSRT	I	48	RESET	I
49	GND	I	50	NC	-	51	NC	-	52	CLK	I
53	NC	-	54	GND	I	55	VCC	I	56	GND	I
57	GND	I	58	NC	-	59	NC	-	60	NC	-
61	NC	-	62	NC	-	63	NC	-	64	NC	-
65	NC	-	66	GND	I	67	NC	-	68	PSUM	O

Table 10: The pin assignment of the App1 chip.

5 Data Bus Adapter

A proto-board [4] is used in the MMC prototype. This board provides I/O connection and bus decoding logics for interfacing with an IBM XT or AT computers that serves as a host for the Test Channel chip. The width of the data bus on the proto-board is 8 bit only, which is incompatible with the 16 bit design of the Test Channel chip. Hence, a data bus adapter that provides data buffering between the 8 bit and 16 bit bus is required.

Figure 11 shows the data bus adapter used in the prototype. The signal lines available on the proto-board include HD[7:0], HA[3:0], /PIOR, /PIOW and /POR. HD[7:0] is the 8 bit data bus from the host. HA[3:0] is the 4 bis address bus from the host. The /PIOR and /PIOW signal lines are derived from the host signal lines /IOR, /IOW and address lines. Both /PIOR and /PIOW are active only when I/O is selected in the address range from 300H to 30FH.

The signal lines of the Test Channel chip that need to be controlled by the host are PD[15:0], PA[3:0], /RD, /WR, /CS and Reset. PD[15:0] is a 16 bit data bus. PA[3:0] is a 4 bit address bus that selects the internal register to be accessed. A data buffer consists of two 74LS373 is used to interface PD[15:8] with HD[7:0]. A bus transceiver 74LS245 is used to interface PD[7:0] with HD[7:0].

The address PA[3:0]=1111 is reserved for accessing the data buffer, which is used to

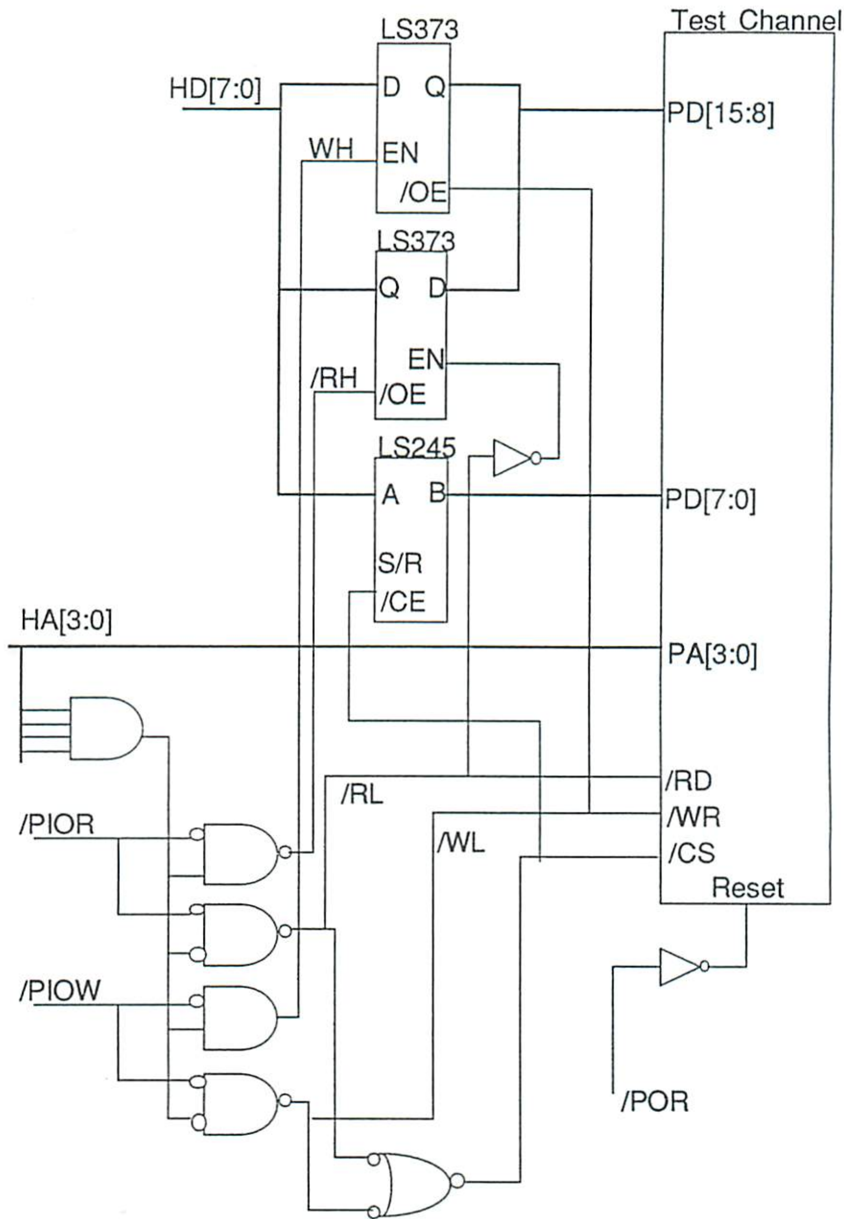


Figure 11: The data bus adapter.

store the high byte data. Two write operations are required to move a 16 bit data from the host to an internal register of the Test Channel. The first write operation, which is "OUT &H30F, high_byte_data" in BASIC, loads the high byte data into the data buffer. The second write operation "OUT &H30i, low_byte_data" loads both the high byte and low byte data into the internal register that is addressed by PA[3:0]=i.

Similarly, two read operations are required to move a 16 bit data from an internal register of the Test Channel to the host. The first read operation "low_byte_data=inp(&H30i)" moves the low byte of the internal register into the host data bus and the high byte into the data buffer. The second read operation "high_byte_data=inp(&H30F)" moves the high byte from the data buffer into the host data bus.

6 Testing the MMC Prototype

Testing the MMC prototype functionally is easy since all its functions can be exercised in testing the Appl. Programs executed by the host can exercise all the operation modes of the Test Channel, which include DTUR, DTCR, PTUR, PTCR, INS, RTEST, STBUS and RSBUS. These functions can be verified if the Appl passes all of its tests.

Host programs that control the Test Channel mainly consist of I/O write and read operations. Write operations are used to load data into the internal registers of the Test Channel. Read operations are used to get status and test results from the Test Channel. Two write operations are required for loading a 16 bit register because that the host interface board has a 8 bit data bus while the Test Channel has a 16 bit data bus.

The handshaking circuit in the Test Channel requires two clock cycles to synchronize a write operation. Since the clock speed of the host processor is faster than that of the Test Channel, the host processor must wait for the completion of each write operation. For example, in the prototype, the clock speed of the Test Channel is 1 MHz, thus a 2 us delay is required between two write operations. Table 11 shows three primitives in Turbo BASIC. The primitive *writeReg* is used to write a 2 bytes data to a I/O location. The primitive *readReg* is used to read a 2 bytes data from a specified I/O location. When the primitive *polling* is executed, the control program is looping until the value of SR is non-zero. These primitives are used to construct all test control programs.

To set up an operation mode, the host must load data to various internal registers of the Test Channel. These registers include SYNR, CR, CNR, TxR, TC, RxR and SR. The signal FEN must be disabled to ensure that the FSM is in a specific state which will allow the write operation to be done correctly. Once all registers are loaded with proper values, the signal FEN is then enabled, so the FSM can operate in the newly determined mode.

Testing the Test Channel
To test the Test Channel it is necessary to execute its eight operation modes. The operation mode INS can be tested if a predefined instruction is sent to the IR of the Appl and the received status is 101. Figure D.1 shows a program that can load 000 to the IR of the

```
SUB writeReg(addr%, highValue%, lowValue%)
LOCAL i, j
OUT &h30f, highValue%
OUT addr%, lowValue%
FOR i=0 to 30
j=j
NEXT
END SUB

SUB readReg(result%, addr%)
LOCAL lowB%, highB%
lowB%=INP(addr%)
highB%=INP(&h30f)
result%=highB%*256+lowB%
END SUB

SUB polling
DO
sr%=INP(&h300)
sr%=sr% and &h0f
LOOP UNTIL sr% <> 0
PRINT "SR [tctc sctc ev0 ev1]="; bin$(sr%)
RETURN
```

Table 11: Primitives for test control programs.

Appl chip. The output pin EXTEST should be active after this program is executed. In addition, the received status, which is in RxR, should be 101. Figure D.2 shows a program that loads 011 to the IR of the Appl chip. The output pin SCANFB should be active after this program is executed. We have tested the INS operation mode by sending all 8 possible patterns for the IR and then observe the primary outputs of the Appl.

The operation mode DTUR can be tested if a predefined vector is sent to a data register of the Appl correctly. Figure D.3 shows a program that can send data to the a selected data register. In this example the boundary scan register is loaded with the data 011, the first two bits can be observed at the primary outputs of the Appl. After this program is executed, the values on the SUM and CO pins should be 1,0 respectively. We have tested the DTUR operation mode by sending different patterns and observe the primary outputs.

The reset operation modes can be tested similarly. For example, the operation mode PTCR are tested if pseudorandom test patterns are sent to the Appl and the signature of the test results are correct. Another example is that the operation mode RSBUS are tested if the slaves can be reseted after this operation is executed.

The Test Channel has been tested and verified by executing control programs for all operation modes. The observed results are correct.

Testing the Appl

The functional circuit of the Appl contains only a 1 bit full adder. The inputs to the adder is PD, PSUM, PCO and the outputs of the adder is NSUM and NCO. The input PD can be controlled by the boundary scan register (BSR) during the INTEST mode, while the inputs PSUM and PCO can be controlled by the feedback register (FB) during the SCANFB mode. Therefore, to apply a 3 bits test vector, the Appl must be in the INTEST mode so that the value for PD can be scanned in, followed by the SCANFB mode during this time the values for PSUM and PCO can be scanned in. It is not necessary to scan out the test results since during both modes the values of the NSUM and NCO can be observed at the output pins SUM and CO, respectively.

To test the full adder functionally, 8 test patterns are required. Table 12 shows the truth table for the inputs and the expected values on the output pins SUM and CO.

Figure D.4 shows a program, which when executed by the host, can sets the PD to 0, the PSUM to 0 and the PCO to 1. The Test Channel first operates in the INS mode and sends 001 to the IR in order to set the Appl to the INTEST mode. The Test Channel then operates in the DTUR mode and sets the input PD of the full adder to 0. Next, the Test Channel operates in the INS mode again and sends 011 to the IR in order to set the Appl to the SCANFB mode. The Test Channel then operates in the DTUR mode and sets the inputs PSUM and PCO of the full adder to 0 and 1, respectively. If the full adder is fault free, the values of the output pins SUM and CO should be 1 and 0, respectively.

The full adder can be tested functionally by repeating this process for the remaining test vectors.

PD	PSUM	PCO	NSUM	NCO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 12: Functional test vectors and expected results for the full adder.

7 Conclusions

The prototype has successfully executed the required test procedures for the test chip. We have shown that programs that describe these test procedures can be easily developed since they can be written in a high level language such as BASIC. Since the prototype has been implemented using a PC, it costs much less than a conventional ATE. Furthermore, the performance of the prototype is superior to an ATE in testing boundary scan devices thanks to the Test Channel chip.

Using the developed Test Channel it is possible to implement a minimal MMC by adding two chips such as an off-the-shelf RAM chip (e.g. Hitachi 6116), and a micro-controller (e.g. Intel 8048). Thus at most three chips are required to make a board completely self-testable.

The major drawback of the field programmable gate array technology is its limited capacity. Because the maximal capacity of the selected device (ACT1020) is less than 2000 gates, many functions in the original designs have been omitted. It is possible to implement the MMC using a different technology that has a larger capacity than an ACT1020. In such case the performance of the MMC can be improved as follows.

1. Add to the Test Channel parts omitted in this implementation, such as PA and PB registers, and the DC counter as described in [3].
2. Increase the length of the two counters TC and SC, such that both the maximal number of test vectors and the number of bits in each vector can be increased.
3. Add a memory control circuit so that the Test Channel can access a local RAM. In this way it is possible for the Test Channel to send a long sequences of instructions and data without interruption.

4. Integrate the processor, the Test Channel and a memory unit into a chip. Handshaking circuit among the processor and the Test Channel can be avoided since they can be designed to be synchronous. However, to fit into a chip, the size of the processor instruction set and the on-chip memory should be bounded.

8 Acknowledgment

The author wishes to thank Professor Abbas El Gamal of Information Systems Laboratory at Stanford University for providing the facilities to use for designing the Test Channel and App1 chips and for obtaining the prototype hardware. Help from Mr. Yi-Hung Chee is also appreciated.

References

[1] C. Maunder and F. Beenker, "Boundary-Scan: A Framework for Structured Design-For-Test", *Proc. Int'l Test Conf.*, 1987, pp. 714-723.

[2] IEEE Standard 1149.1-1990, "IEEE Standard Test Access Port and Boundary Scan Architecture," *IEEE Standards Board, 345 East 47th Street, New York, NY 10017,* May, 1989.

[3] Jung-Cheun Lien and Melvin A. Breuer, "A Universal Test and Maintenance Controller for Modules and Boards," *IEEE Trans. on Industrial Electronics*,Vol. 36, No. 2, May 1989, pp. 231-240.

[4] A. El Gamal, "Protozone: The PC-Based ASIC Design Frame", *EE218 Handout No.7*, Stanford University, Winter 1990.

APPENDIX:

A Schematic diagrams for the Test Channel

B Schematic diagrams for the App1

C Timing diagrams for the Test Channel

D Test programs in Turbo Basic

A Schematic of the Test Channel

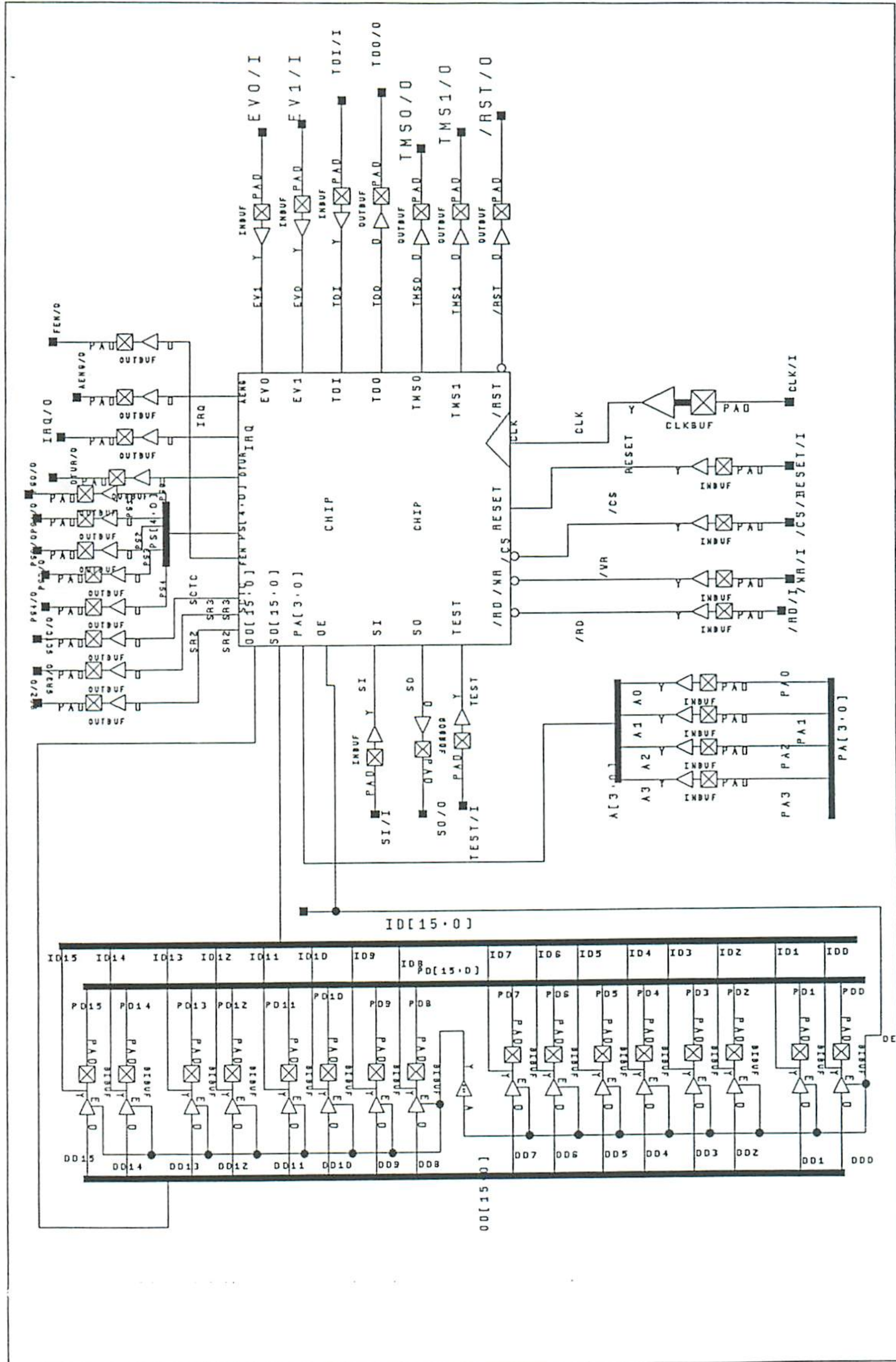


Figure A.1: Schematic shows the I/O pads of the Test Channel.

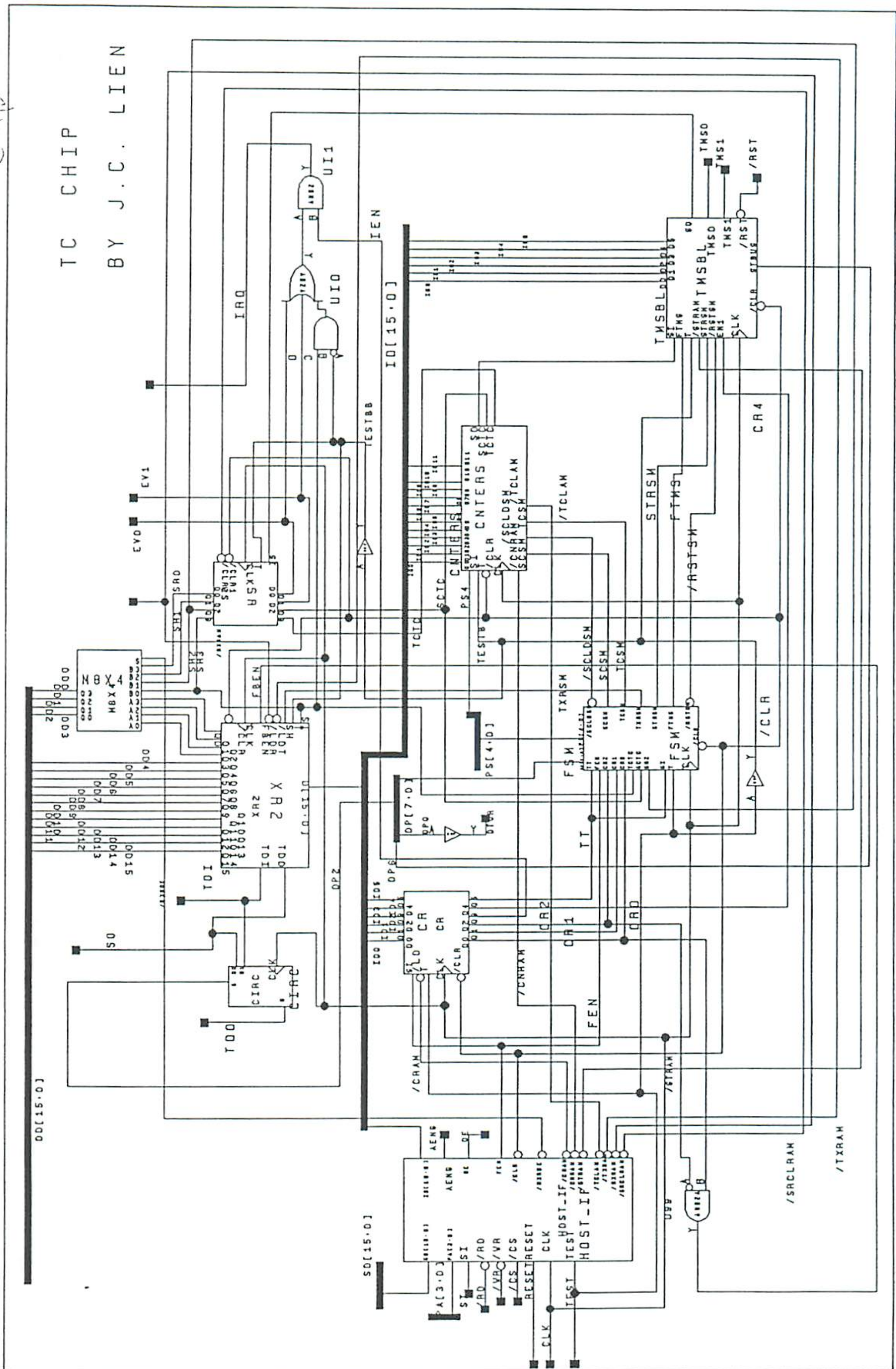


Figure A.2: Top level schematic of the Test Channel.

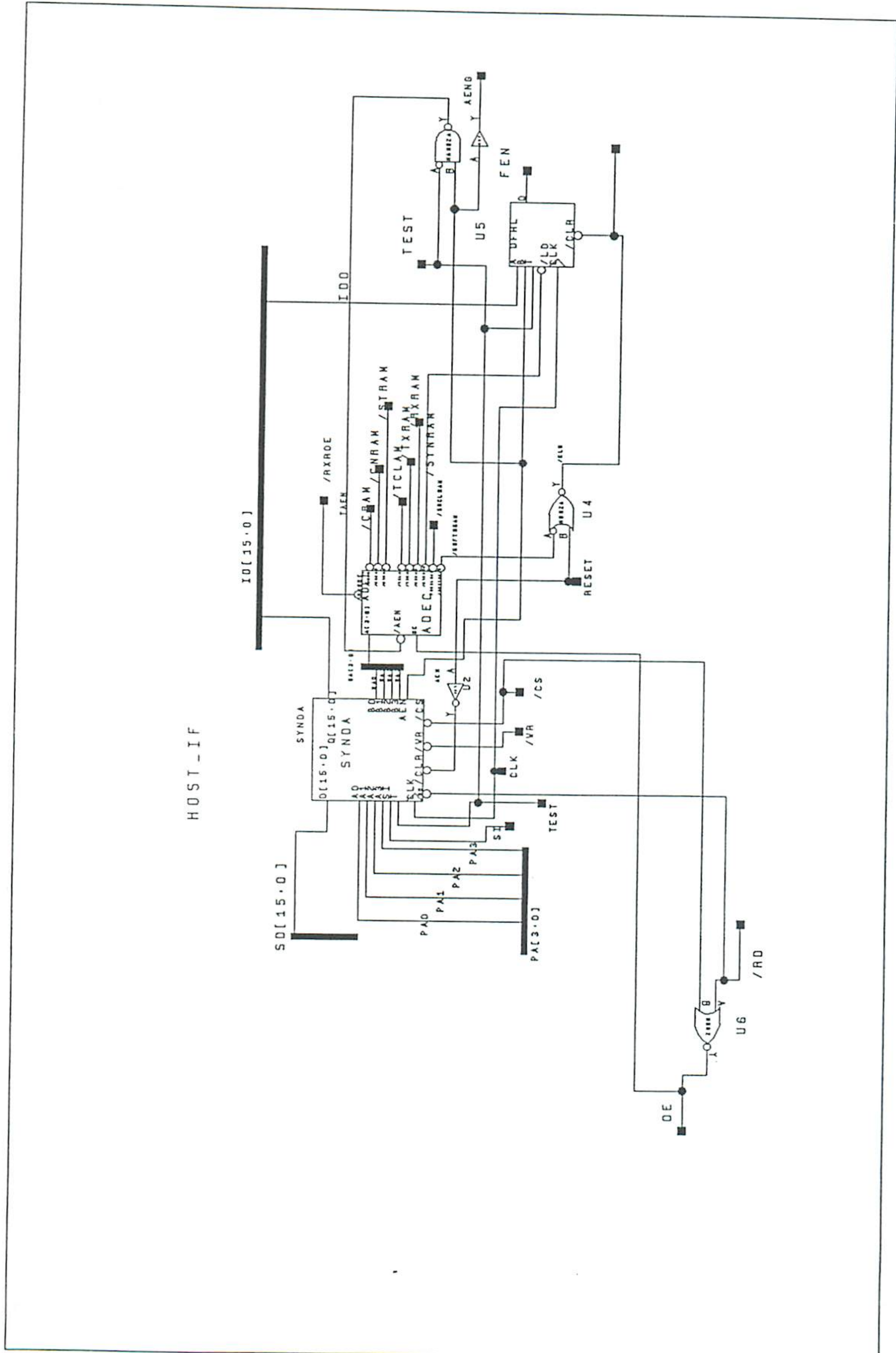


Figure A.3: Schematic of the HOST_IF.

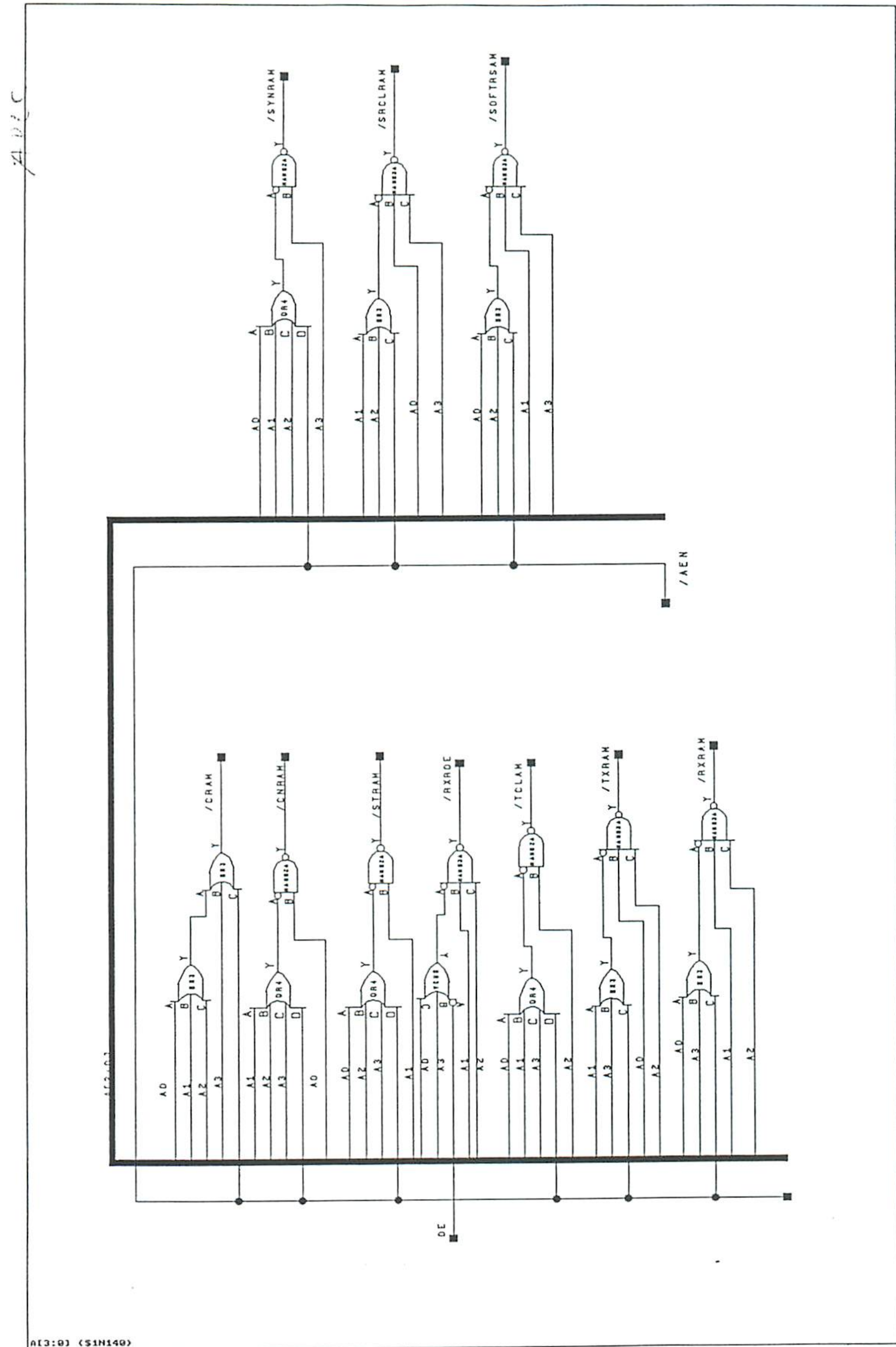


Figure A.4: Schematic of the ADEC.

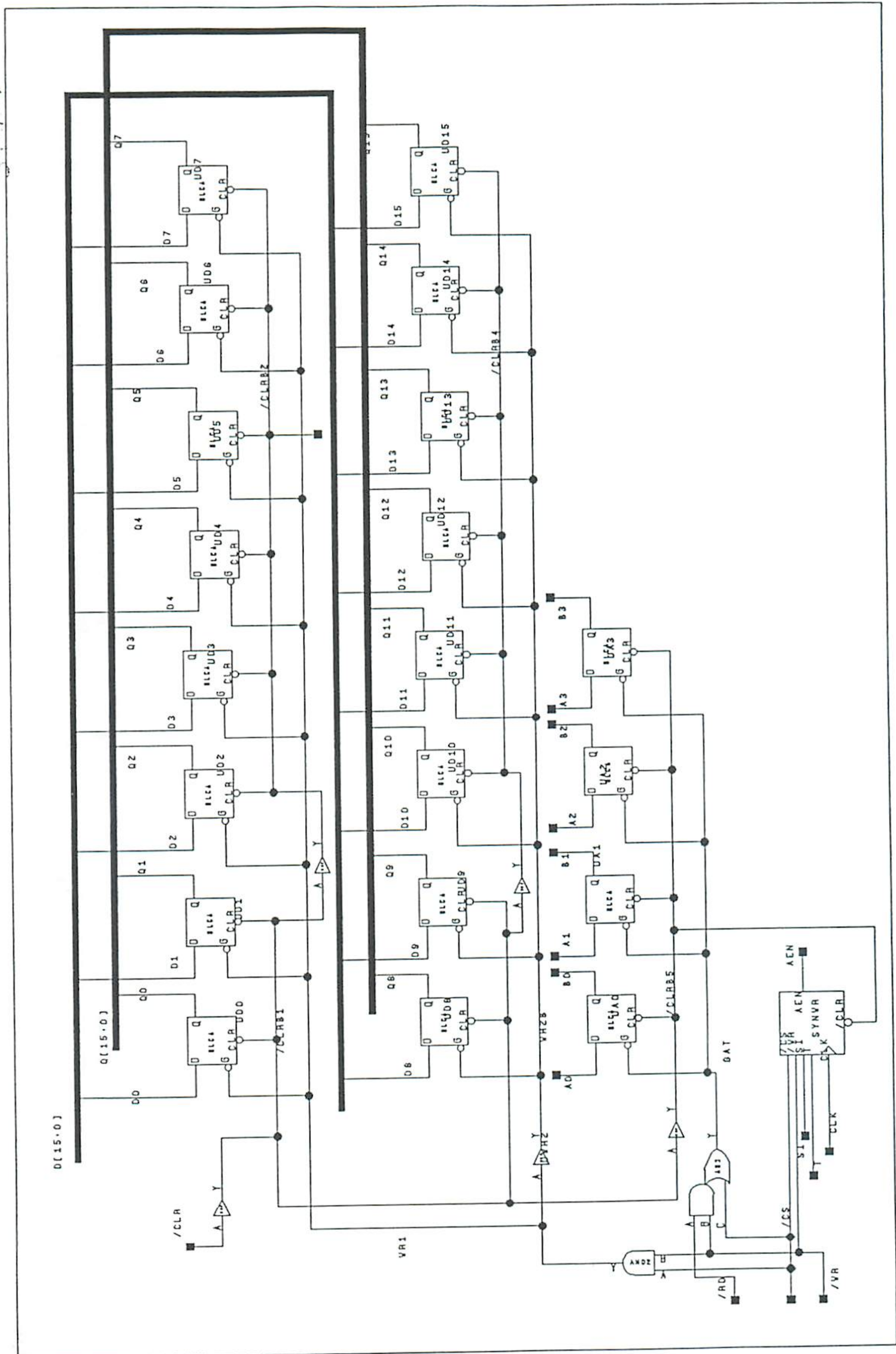


Figure A.5: Schematic of the SYNDA.

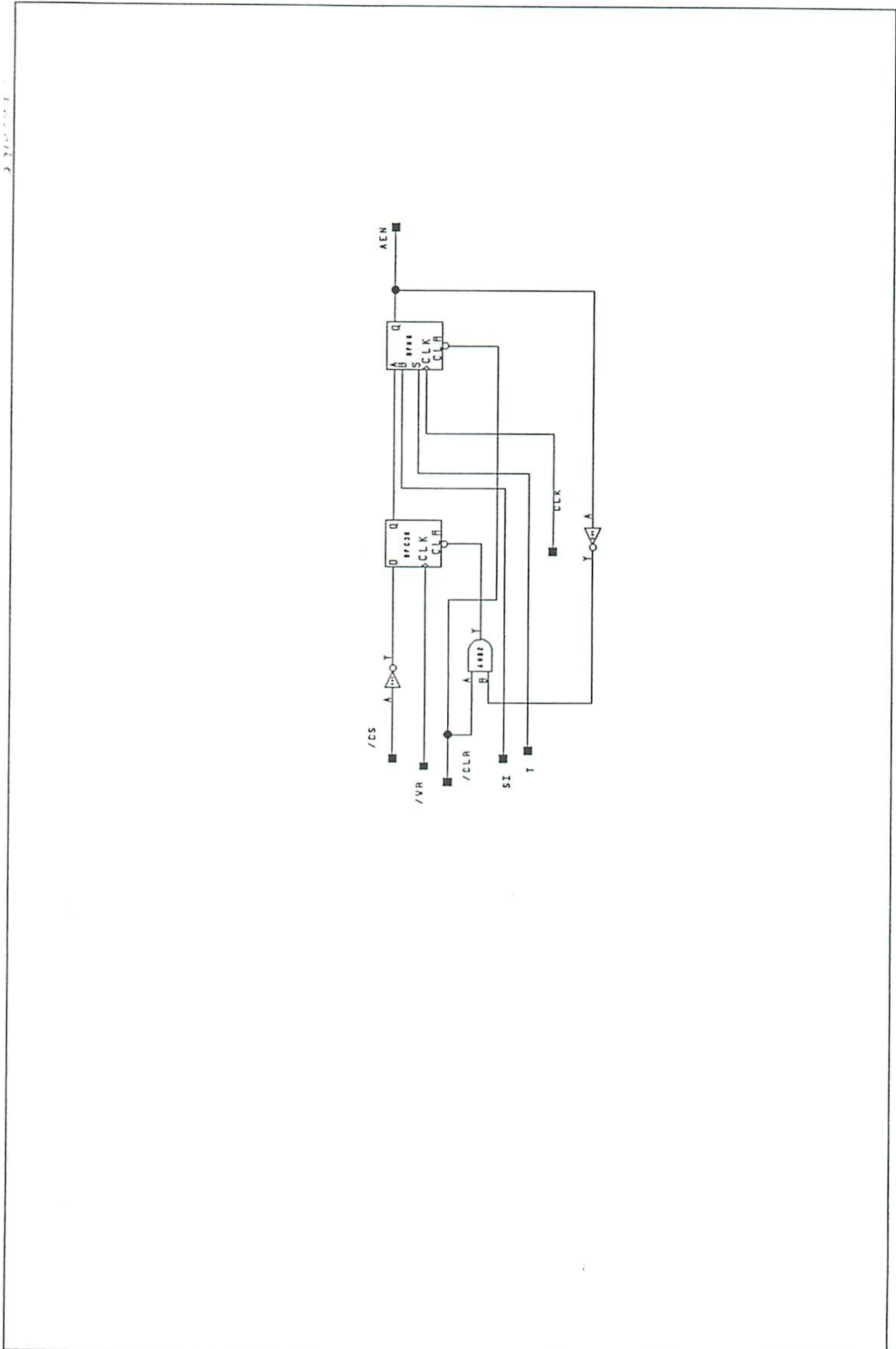


Figure A.6: Schematic of the SYNWR.

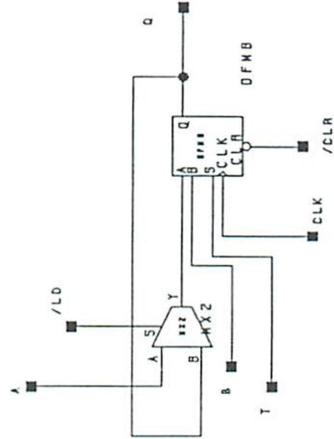


Figure A.7: Schematic of the DFHL.

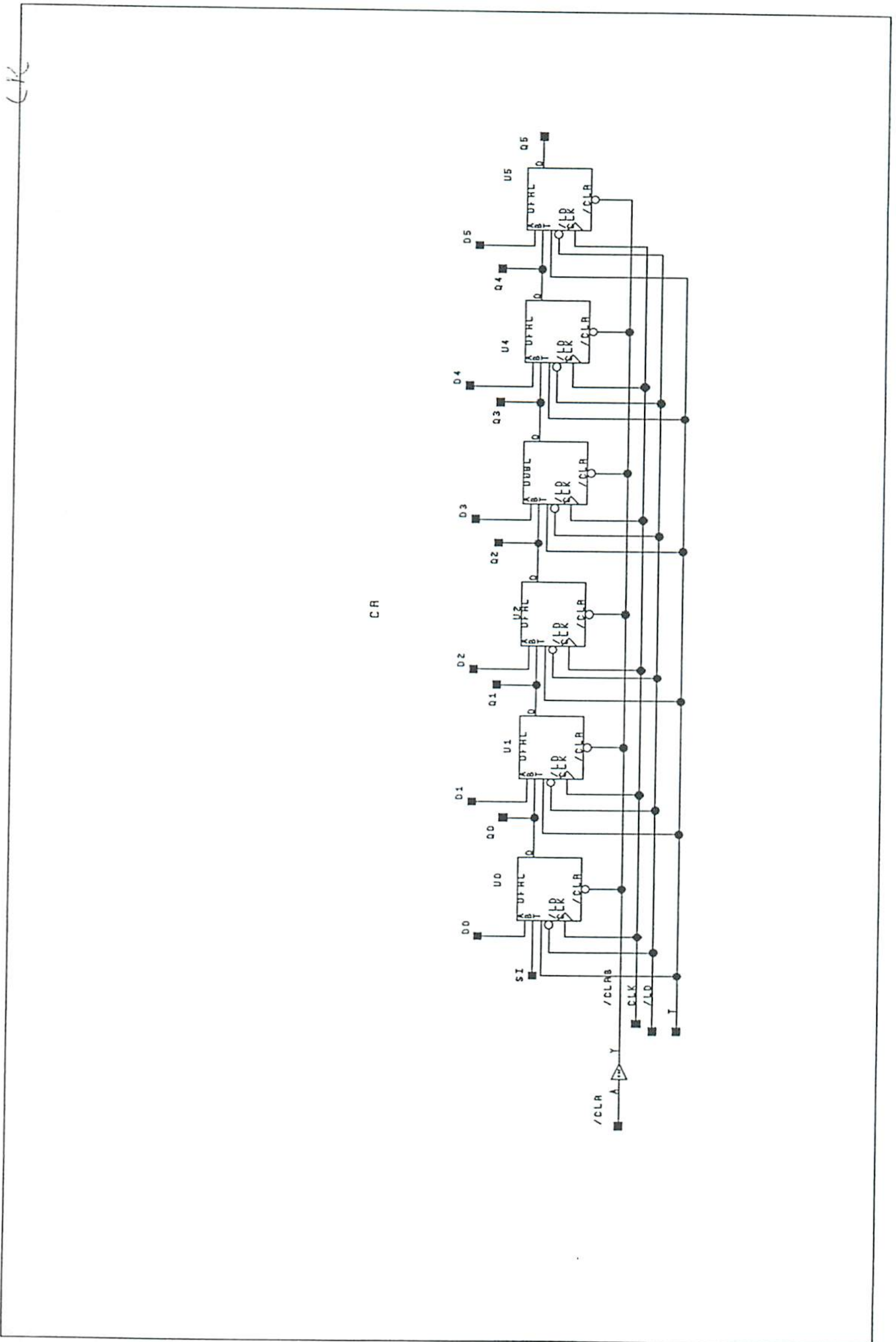


Figure A.8: Schematic of the CR.

Figure A.9: Schematic of the FSM.

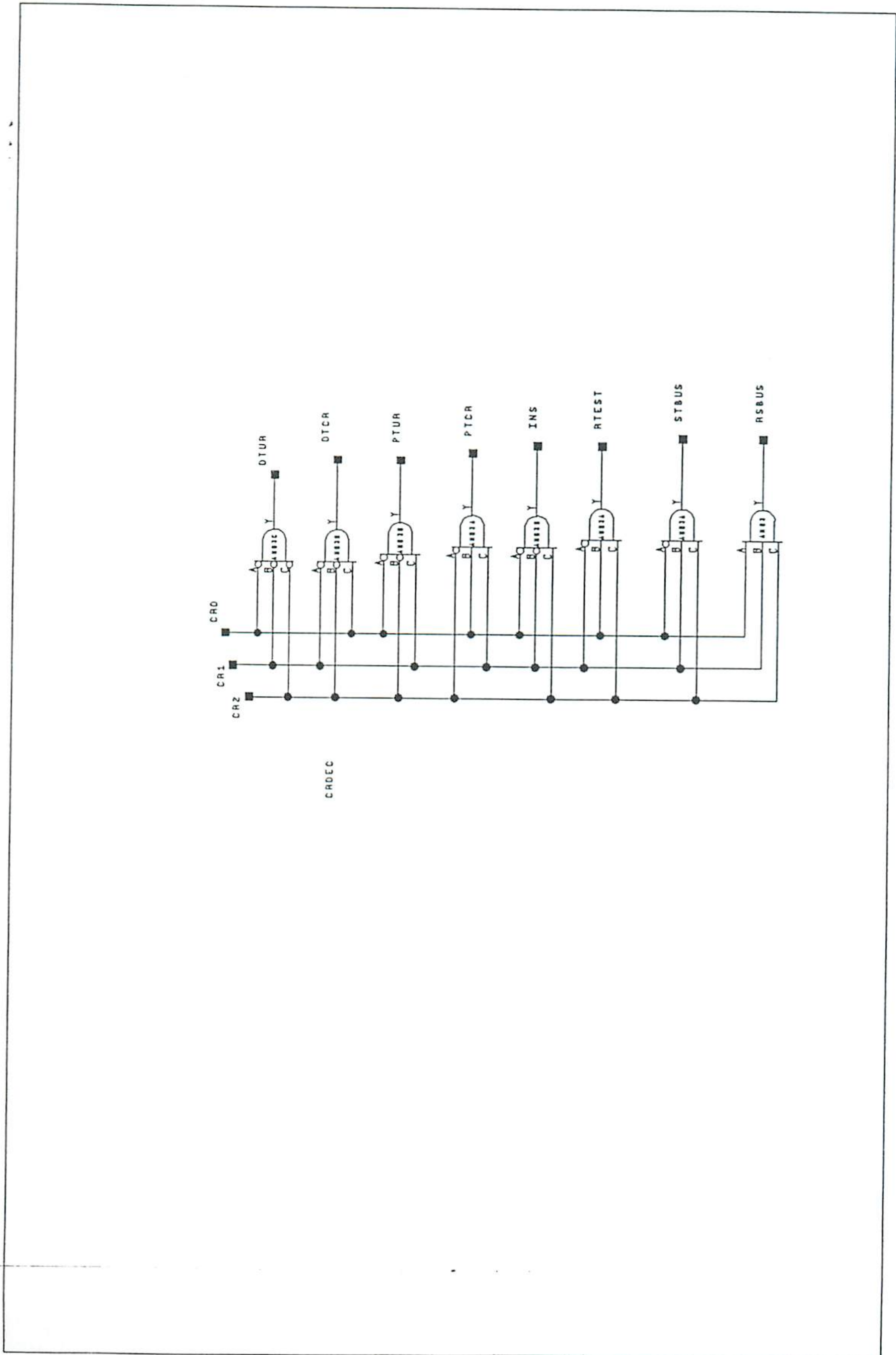


Figure A.10: Schematic of the CRDEC.

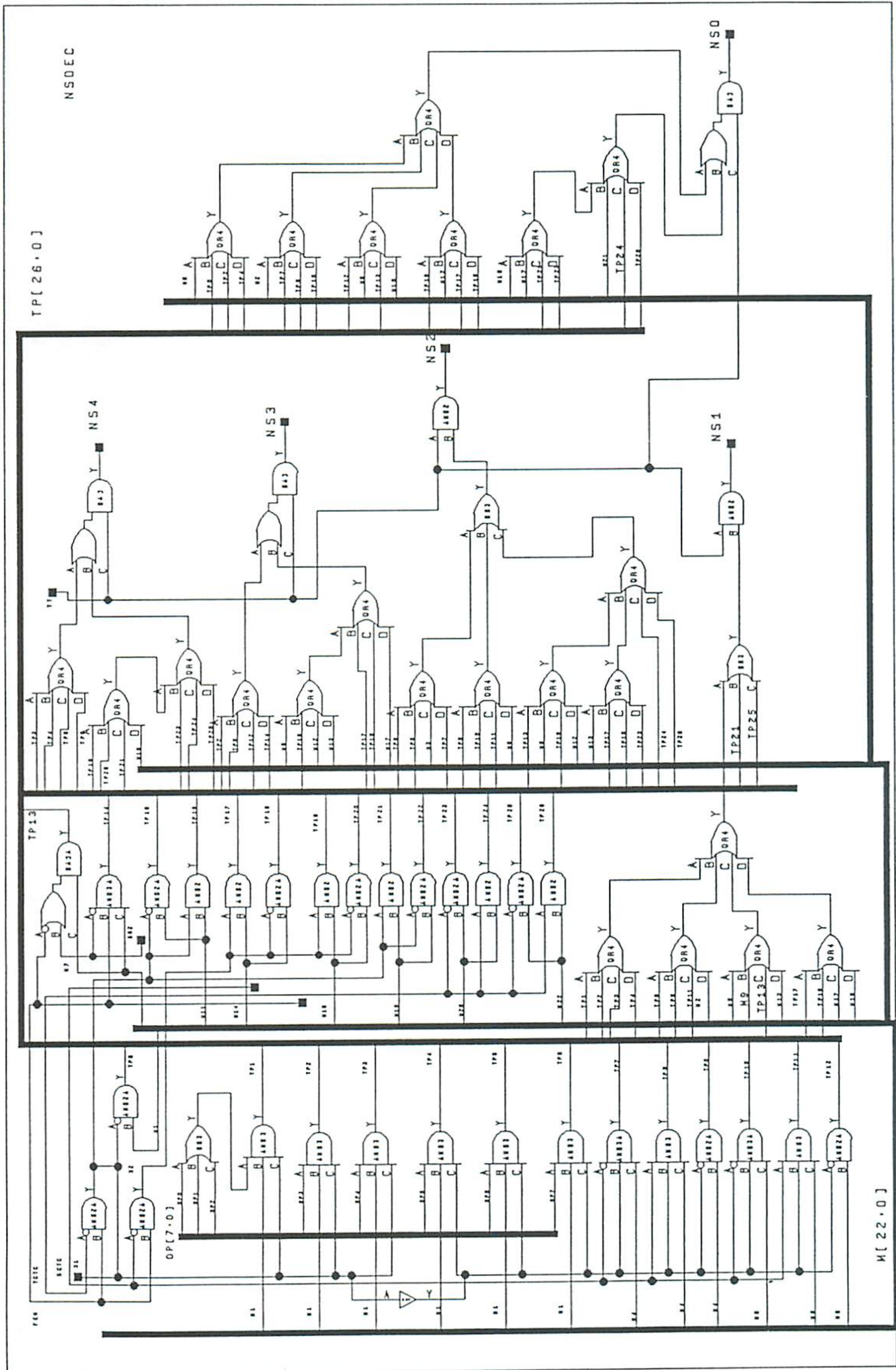


Figure A.11: Schematic of the NSDEC.

Figure A.12: Schematic of the STATE.

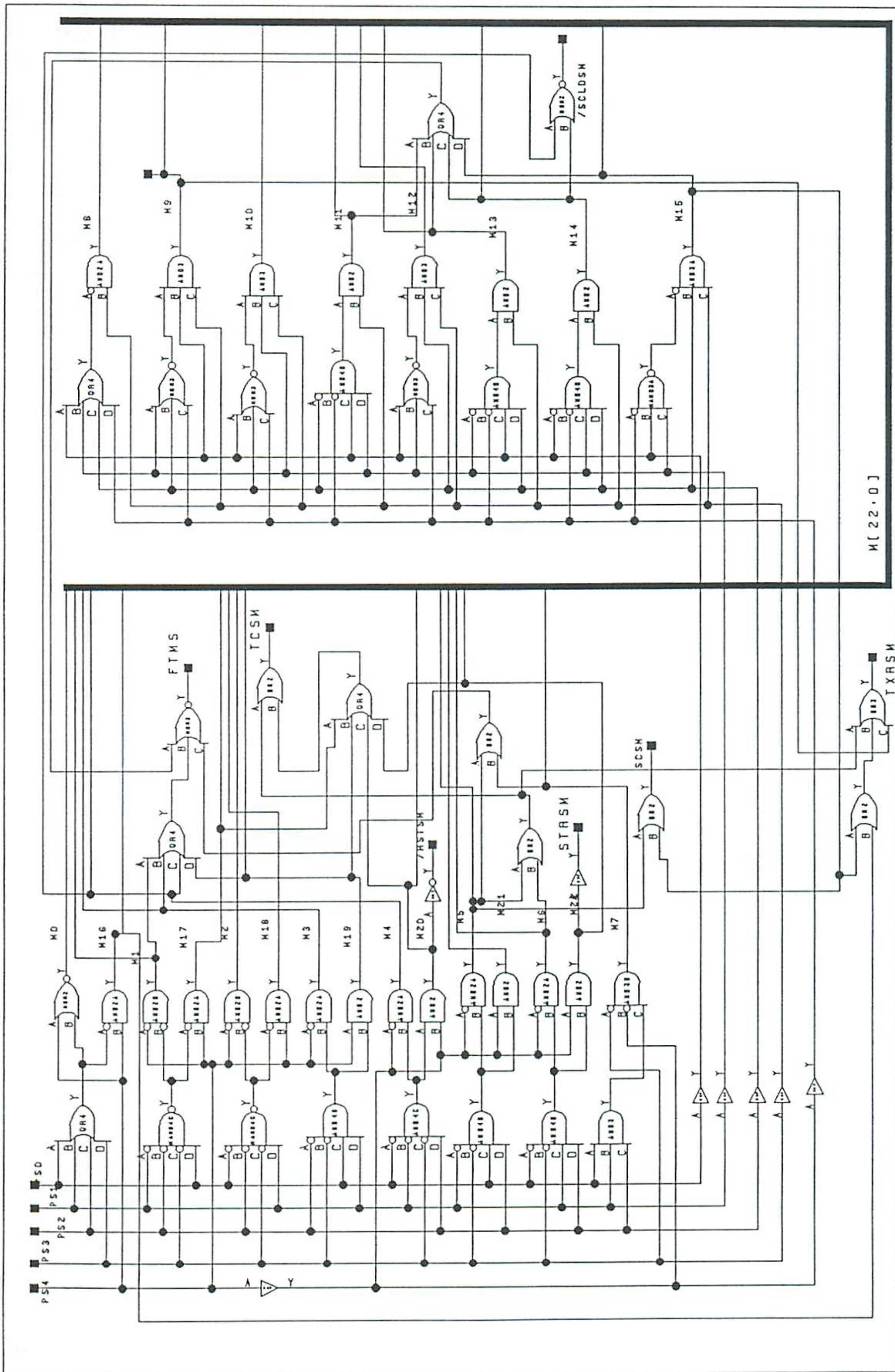


Figure A.13: Schematic of the OL.

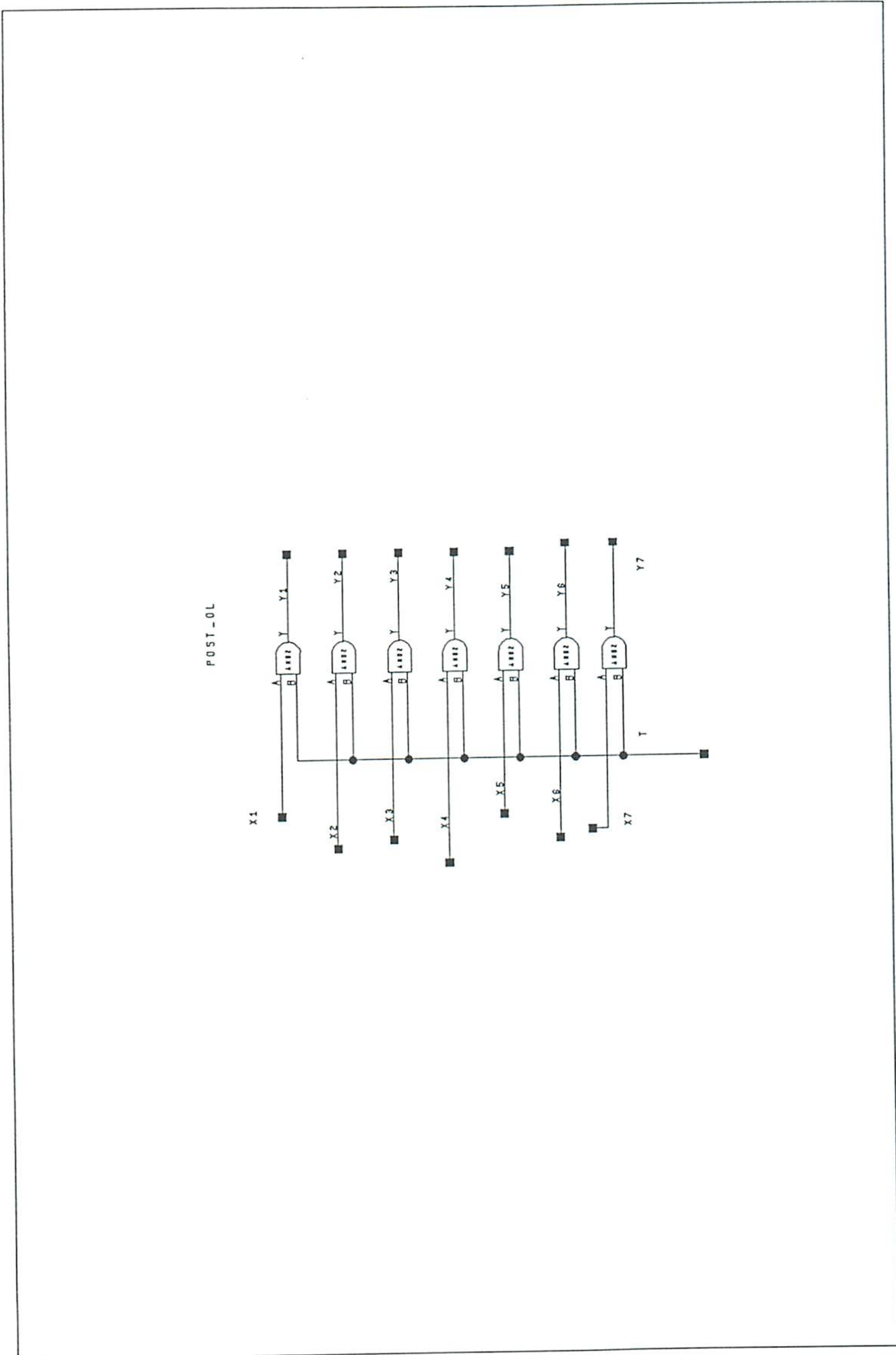


Figure A.14: Schematic of the POST_OL.

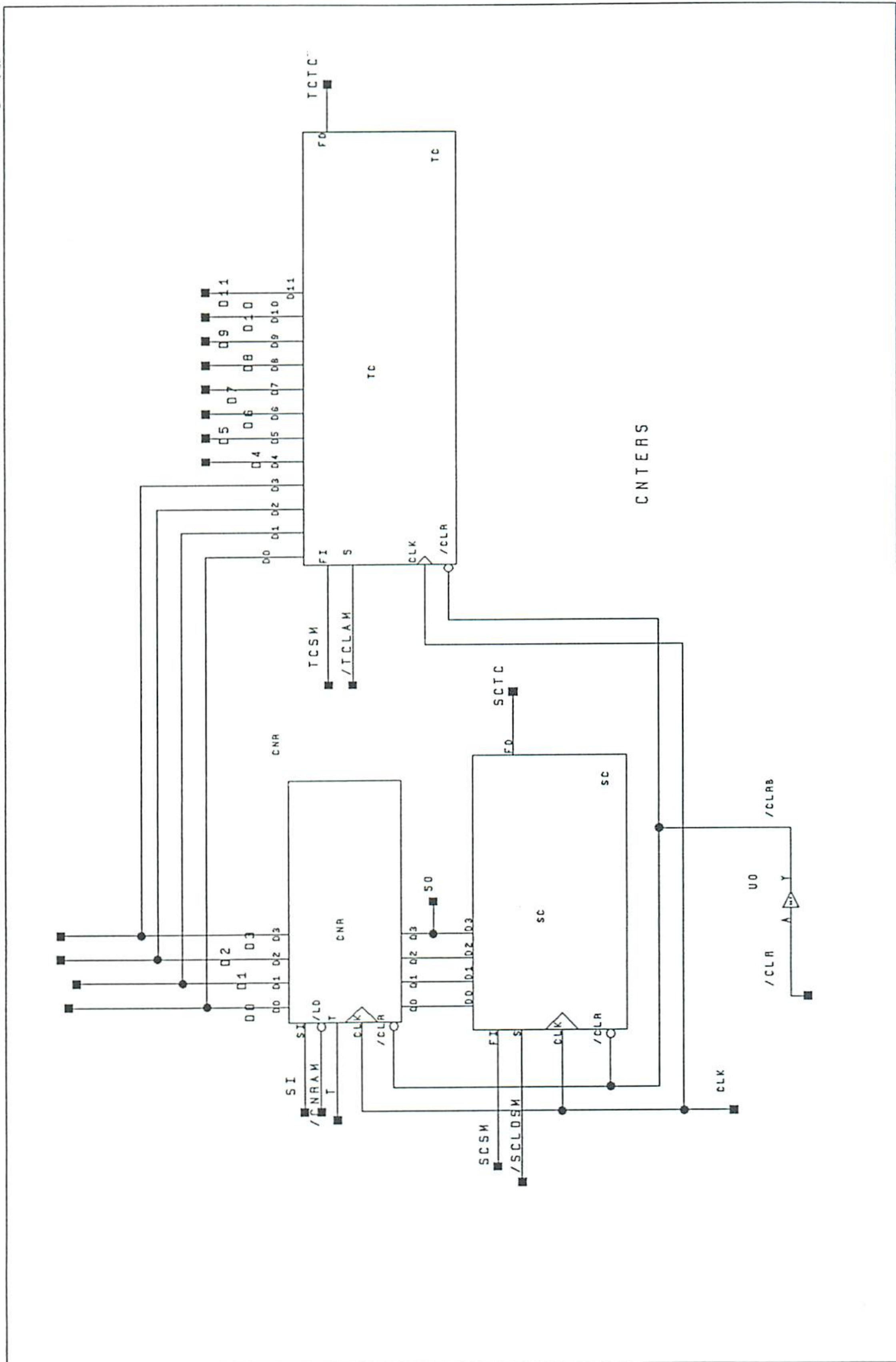


Figure A.15: Schematic of the CNTERS.

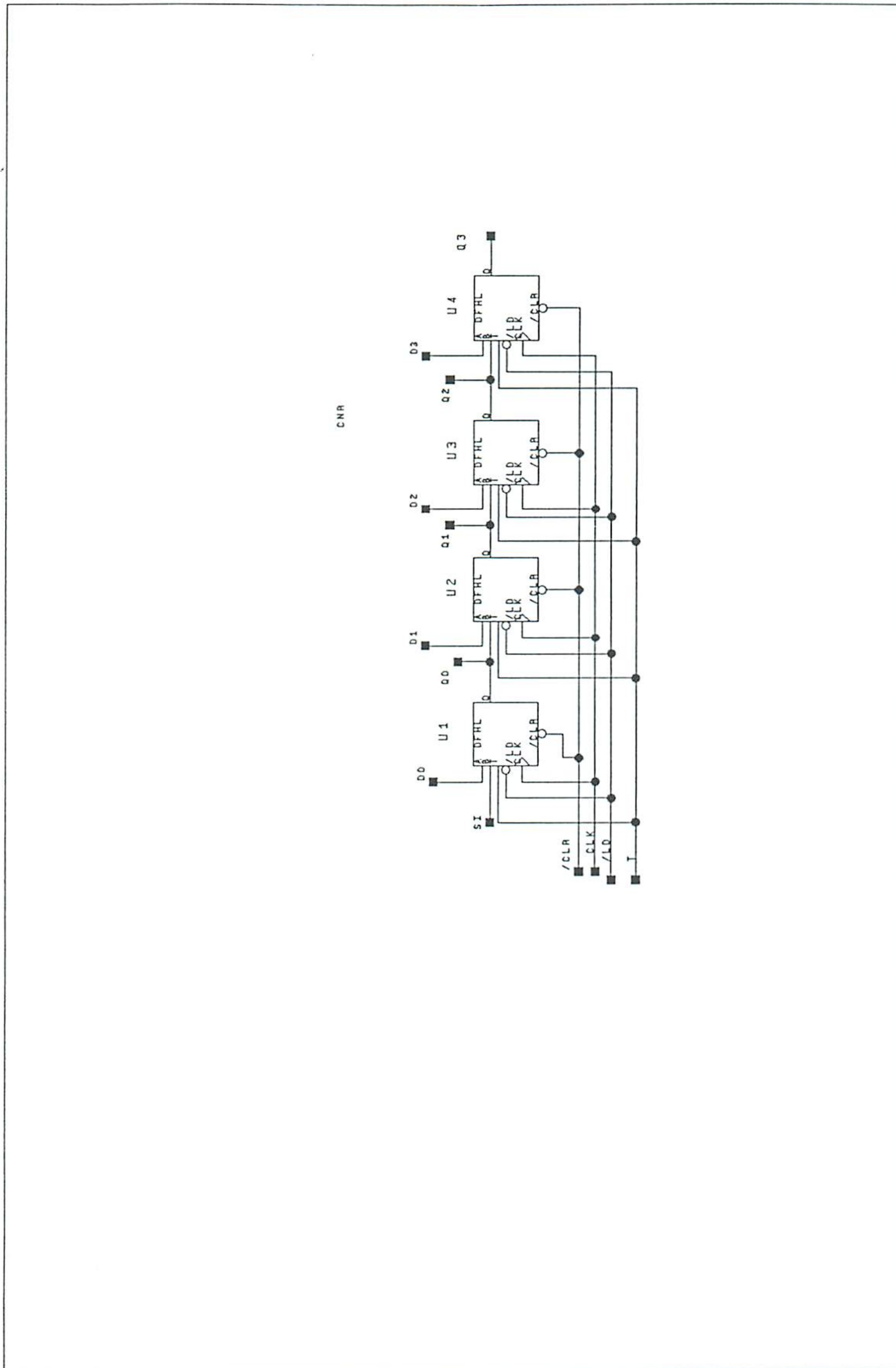


Figure A.16: Schematic of the CNR.

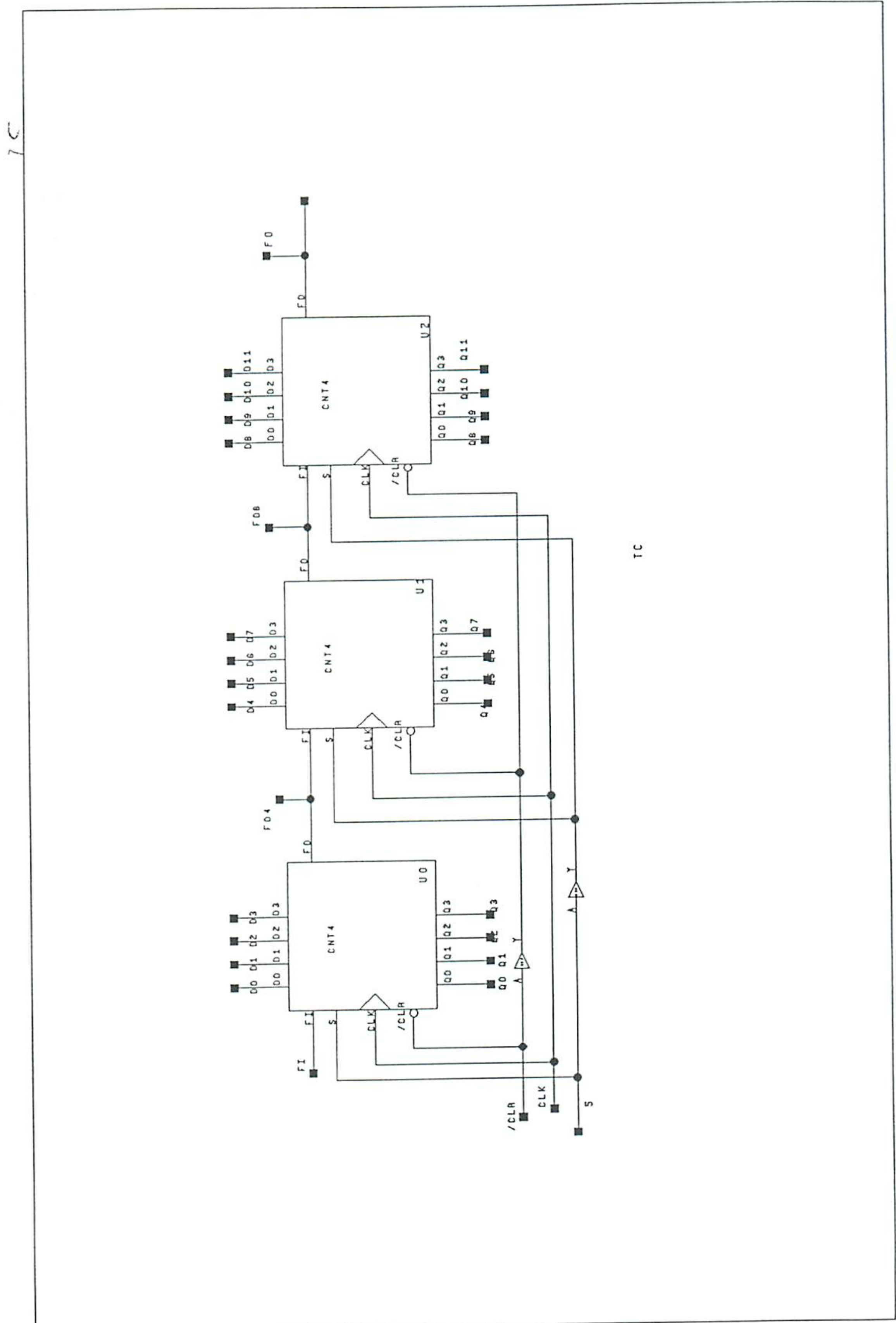


Figure A.17: Schematic of the TC.

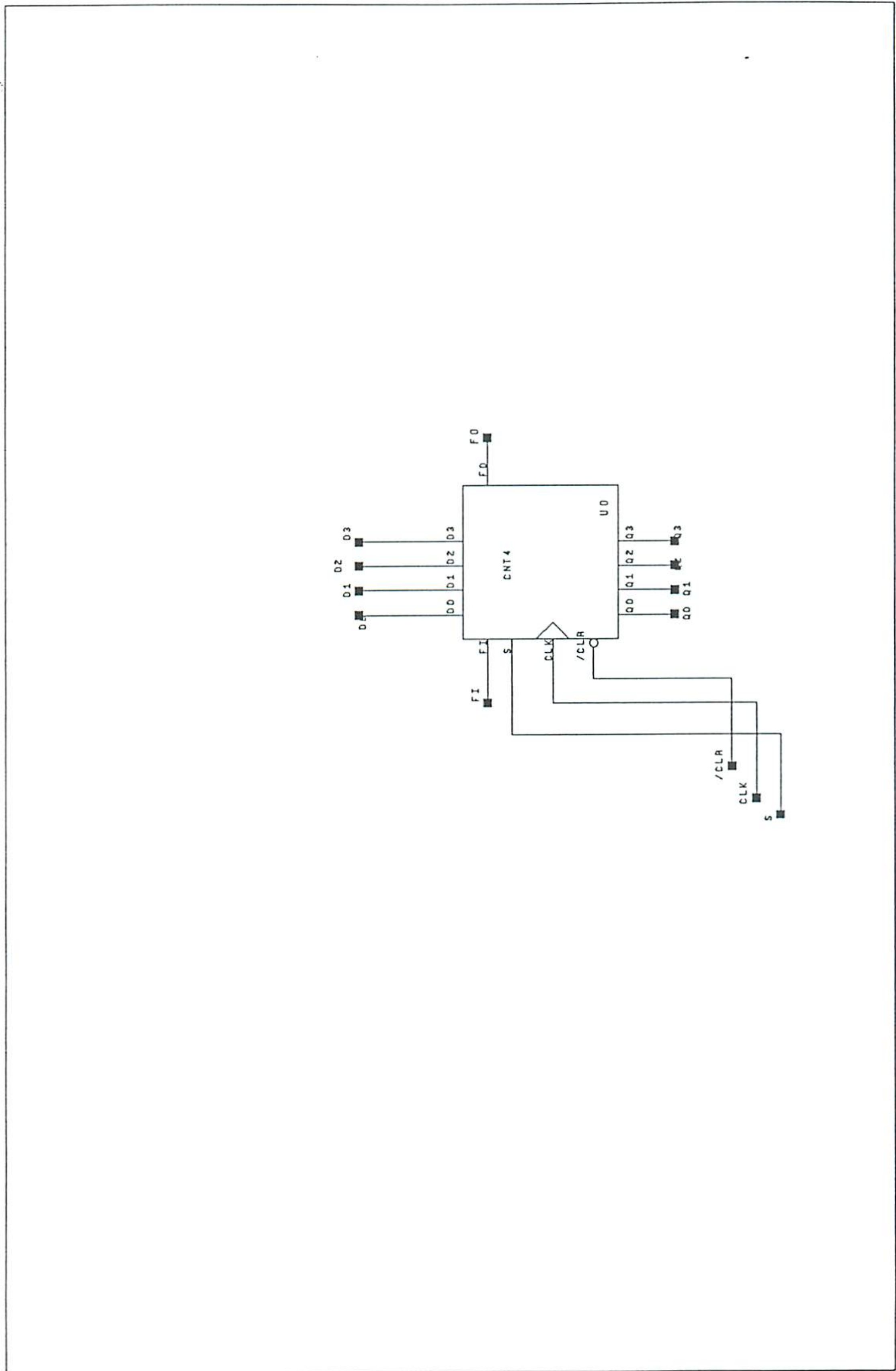


Figure A.18: Schematic of the SC.

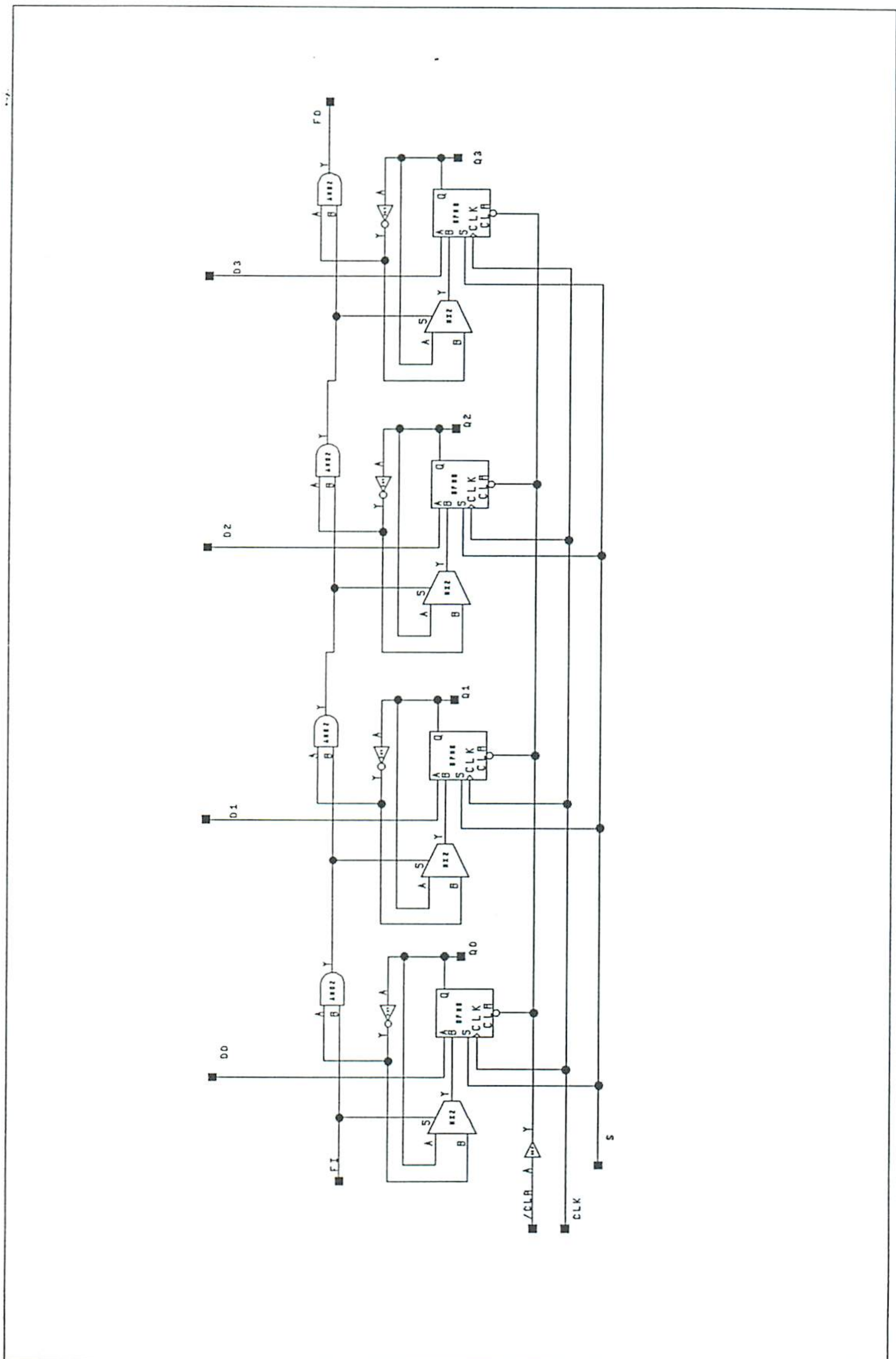


Figure A.19: Schematic of the CNT4.

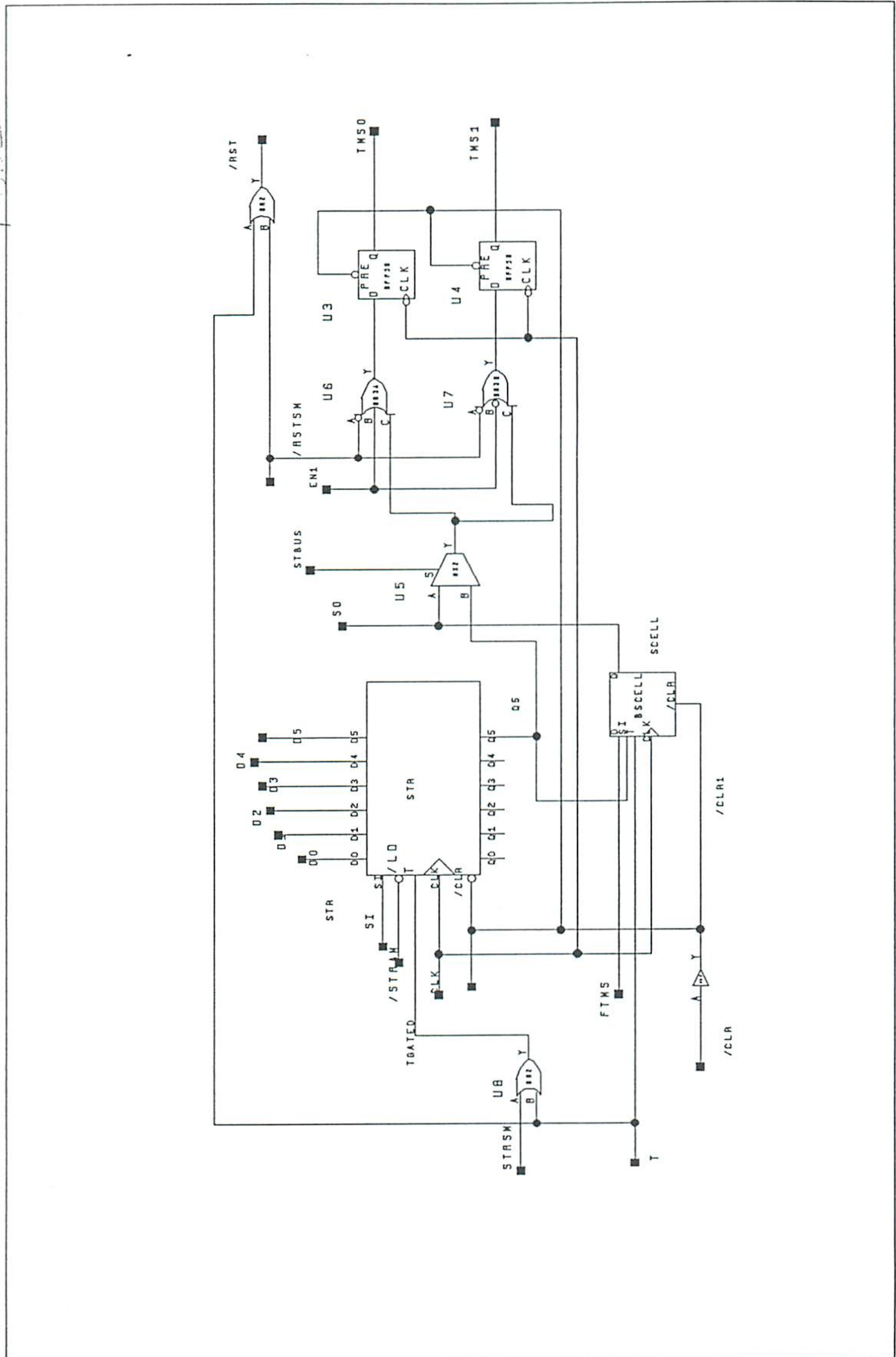


Figure A.20: Schematic of the TMSBL.

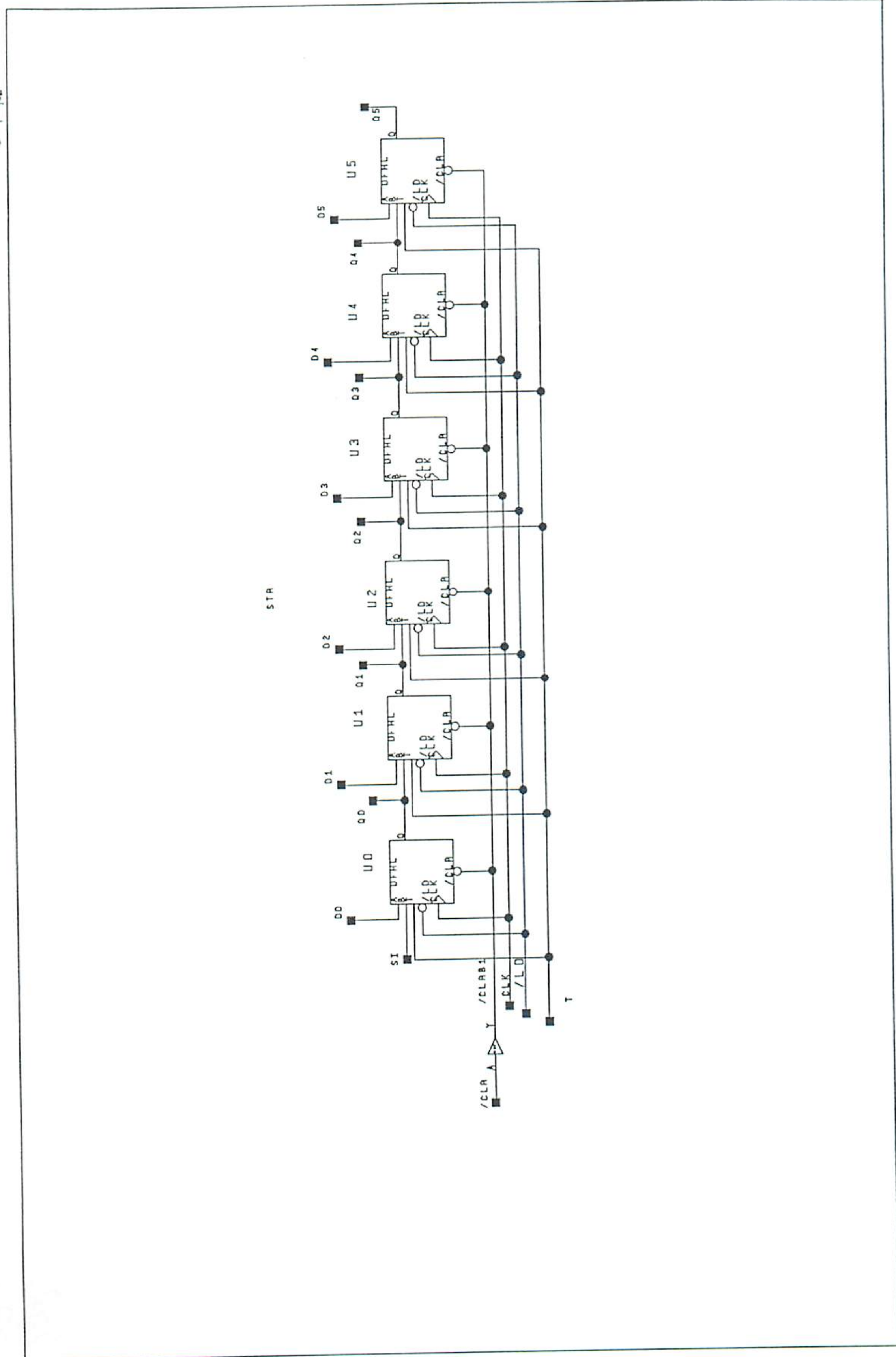


Figure A.21: Schematic of the STR.

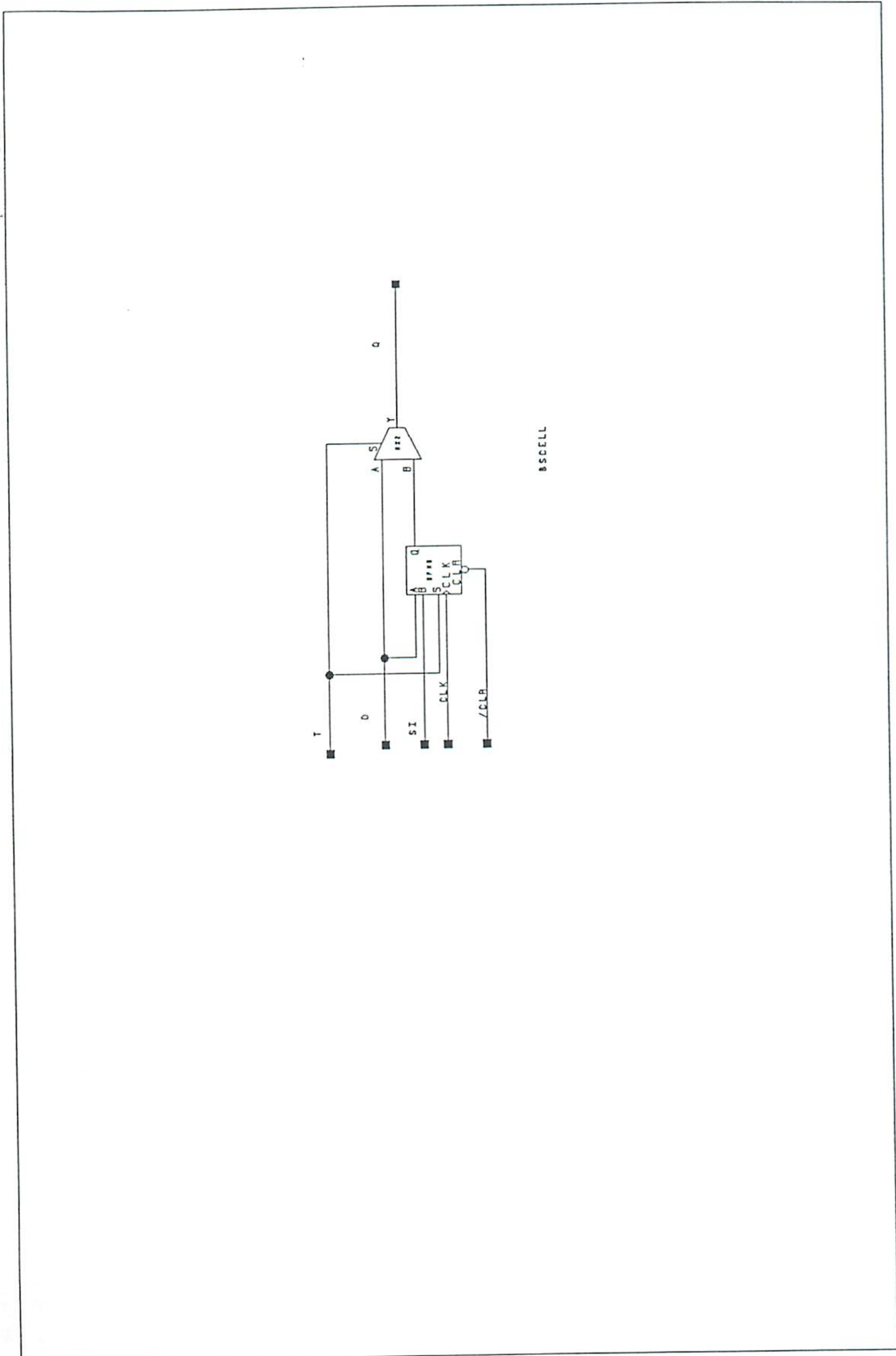


Figure A.22: Schematic of the BSCELL.

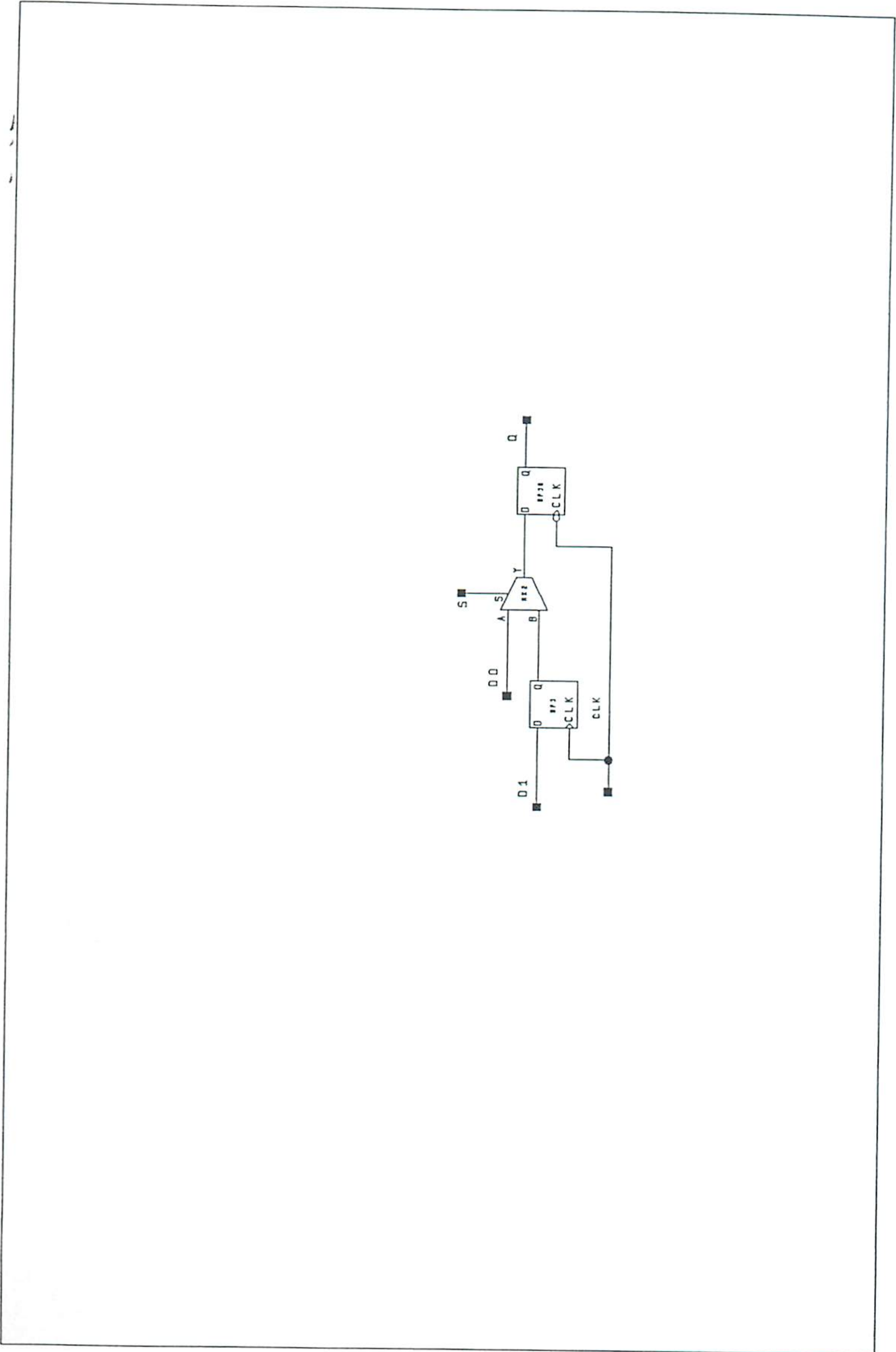


Figure A.22: Schematic of the CIRC.

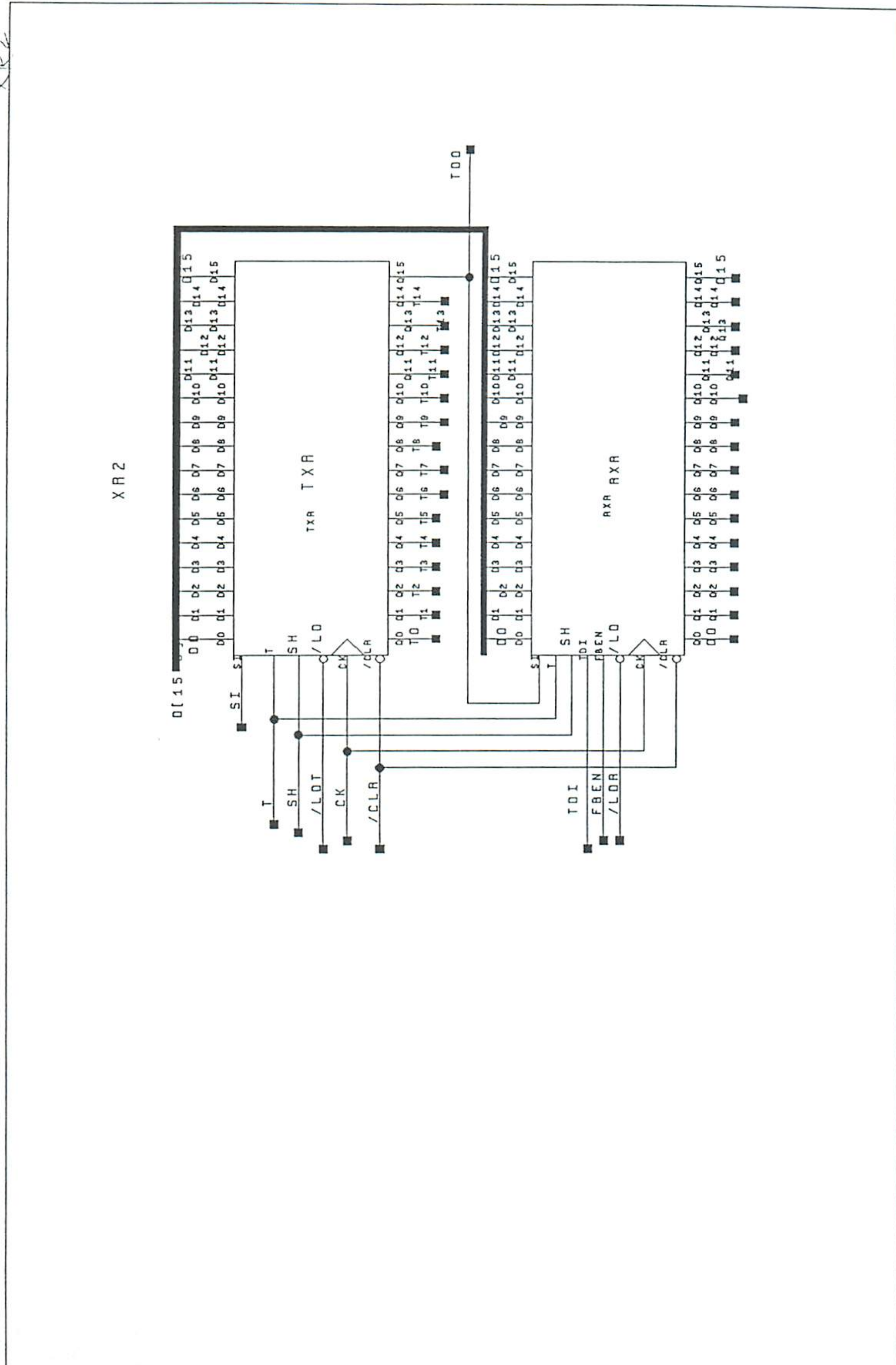


Figure A.24: Schematic of the XR2.

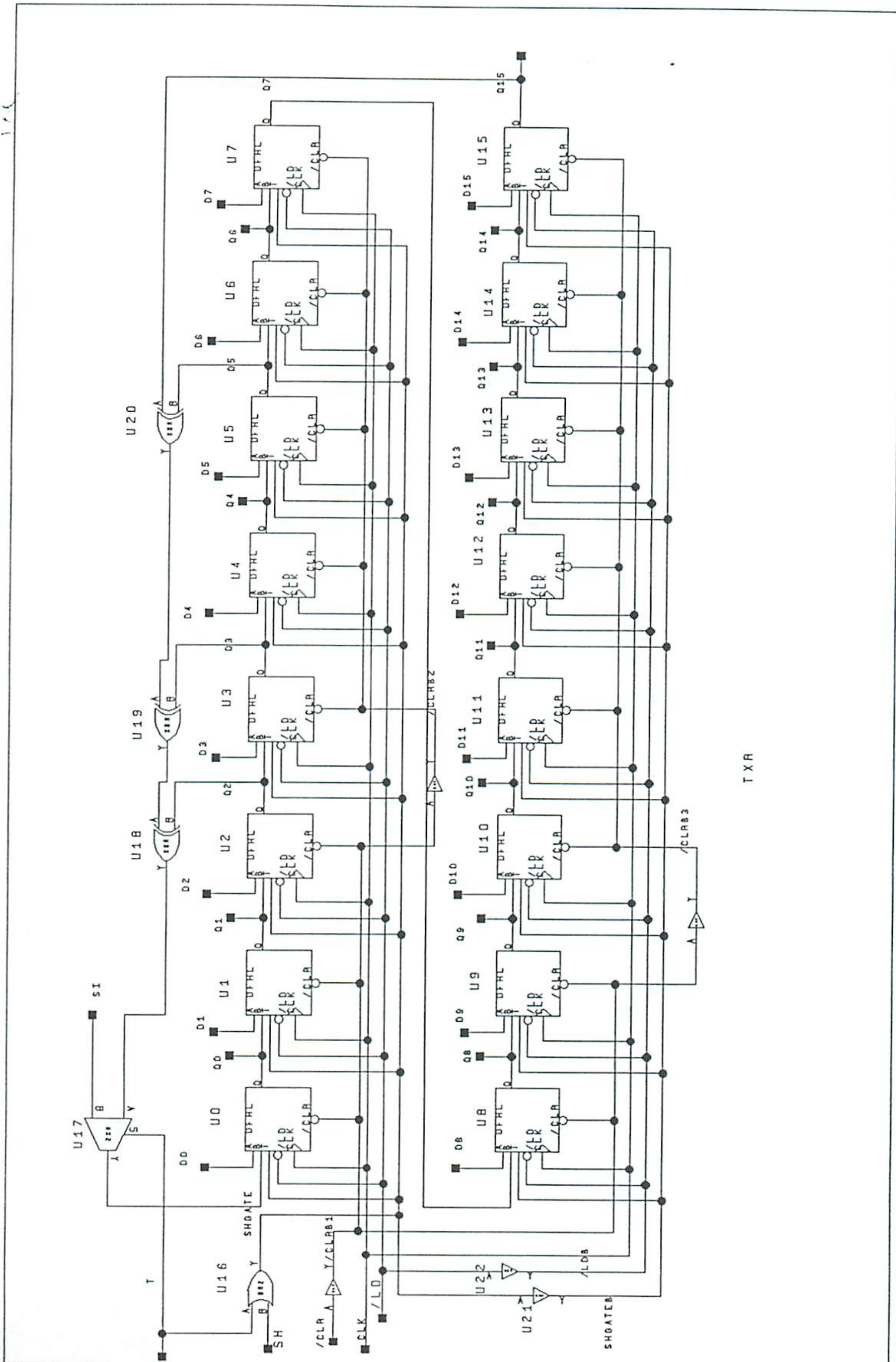


Figure A.25: Schematic of the TxR.

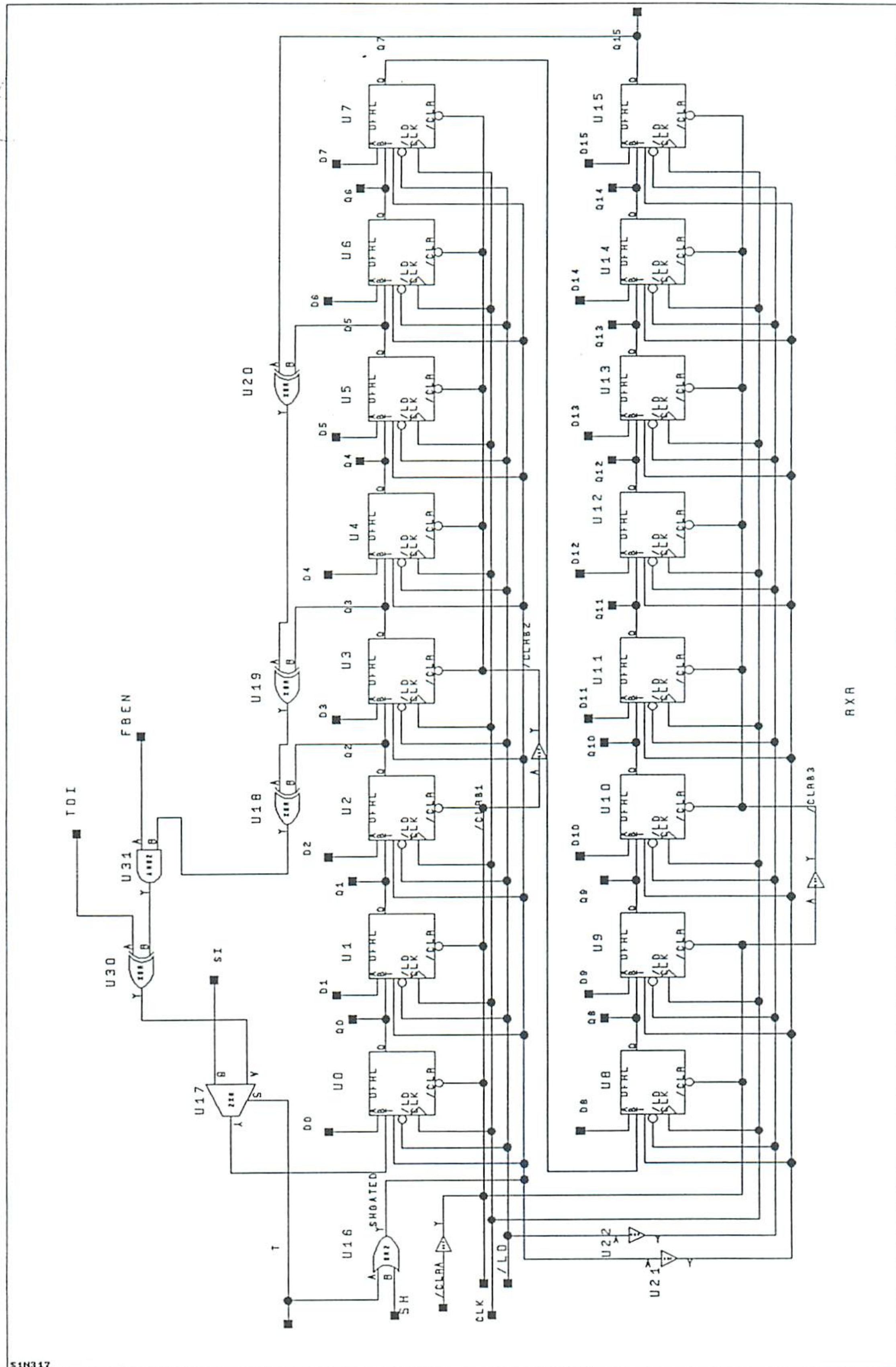


Figure A.26: Schematic of the RxR.

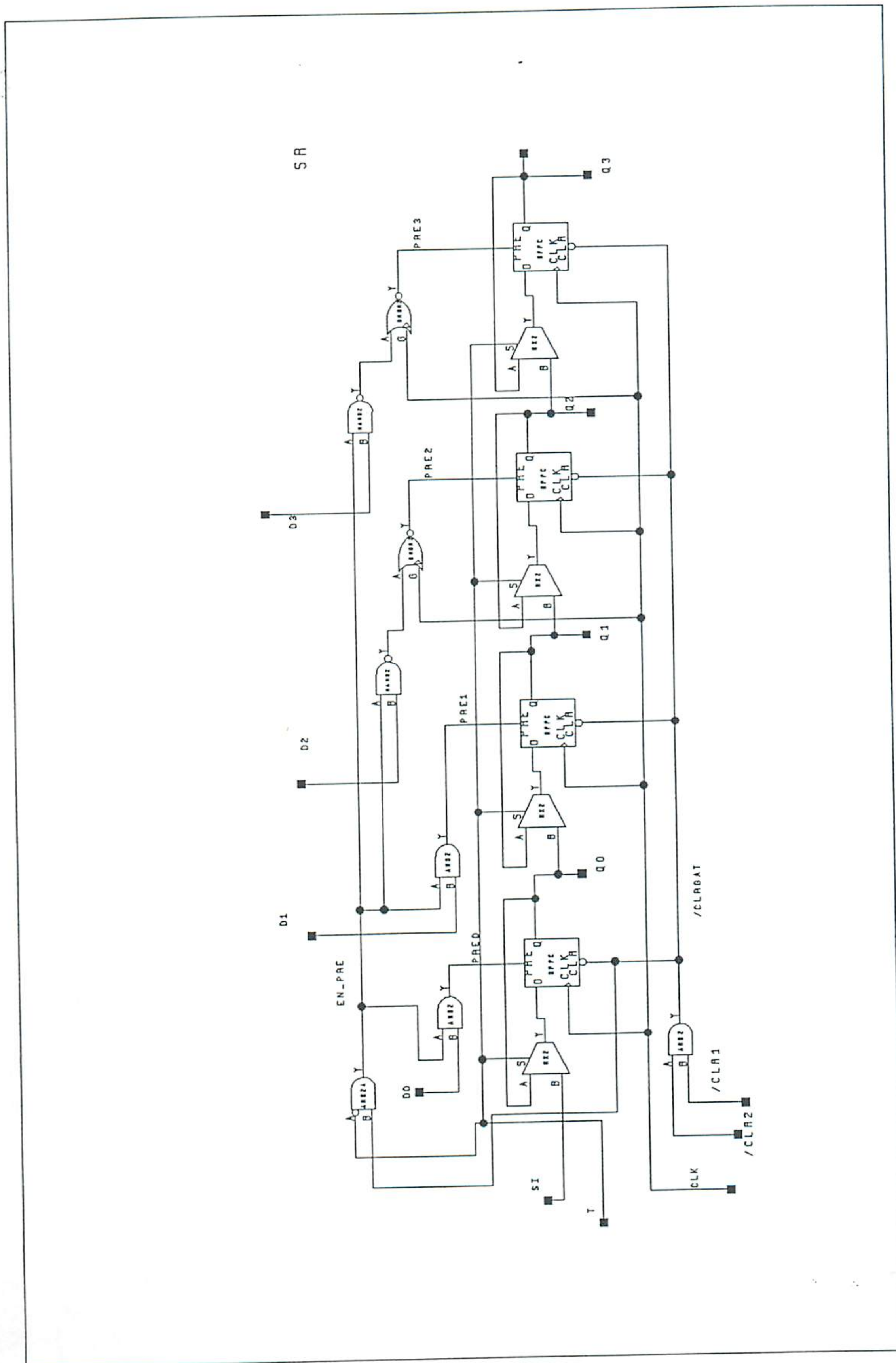


Figure A.27: Schematic of the SR.

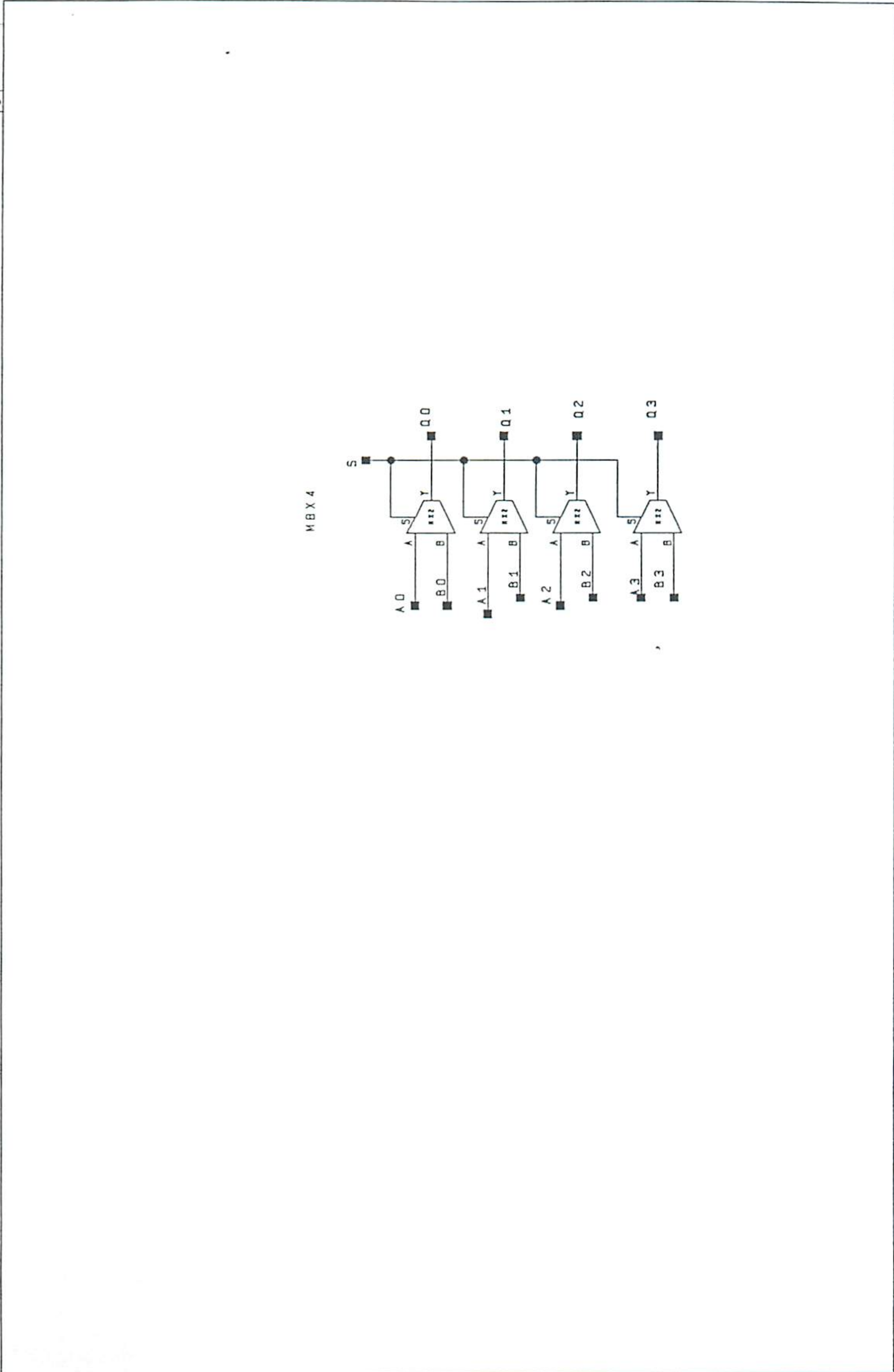


Figure A.28: Schematic of the M8X4.

B Schematic of the App1

Figure B.1: Top level schematic of the App1.

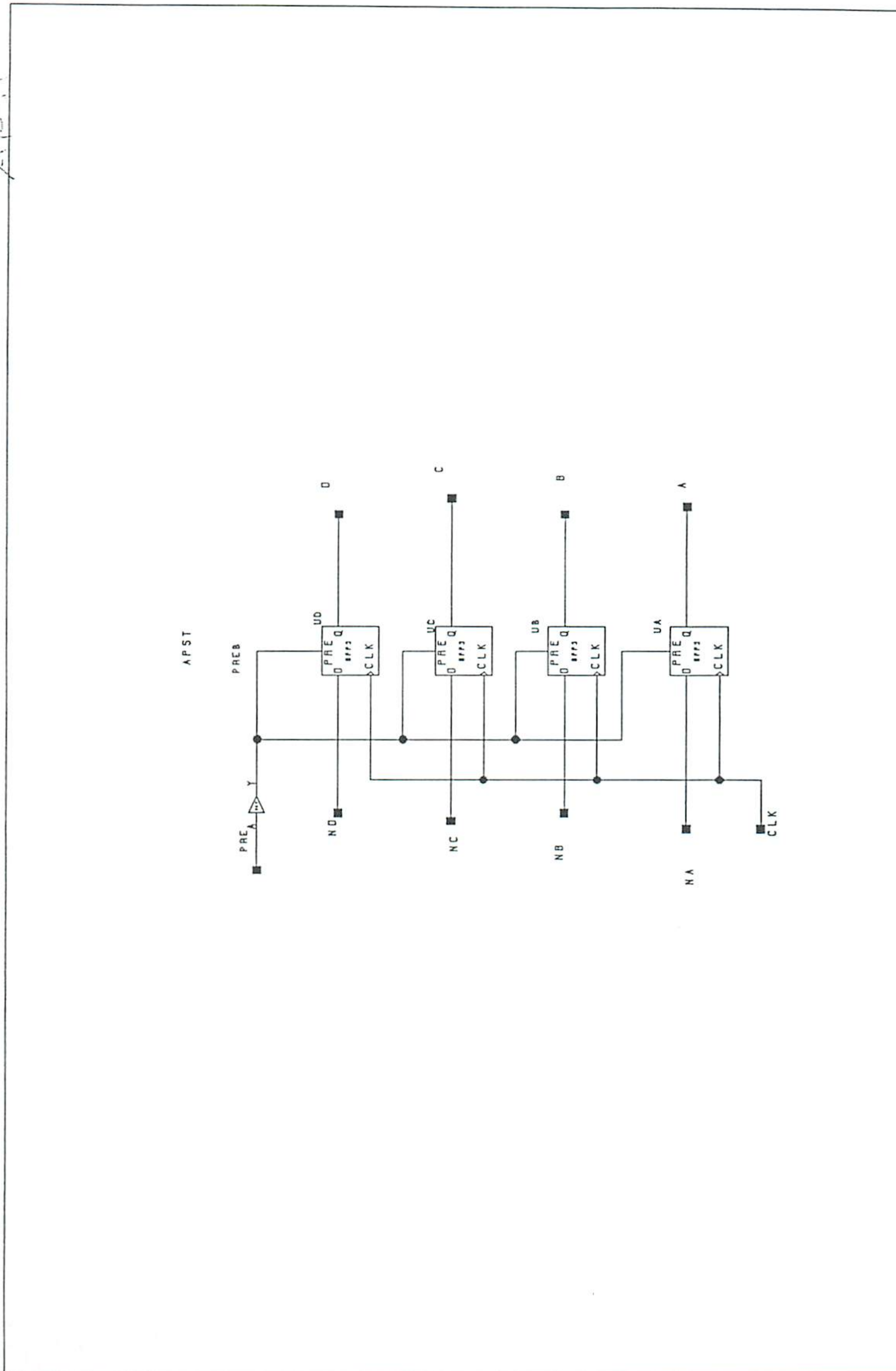


Figure B.2: Schematic of the APST.

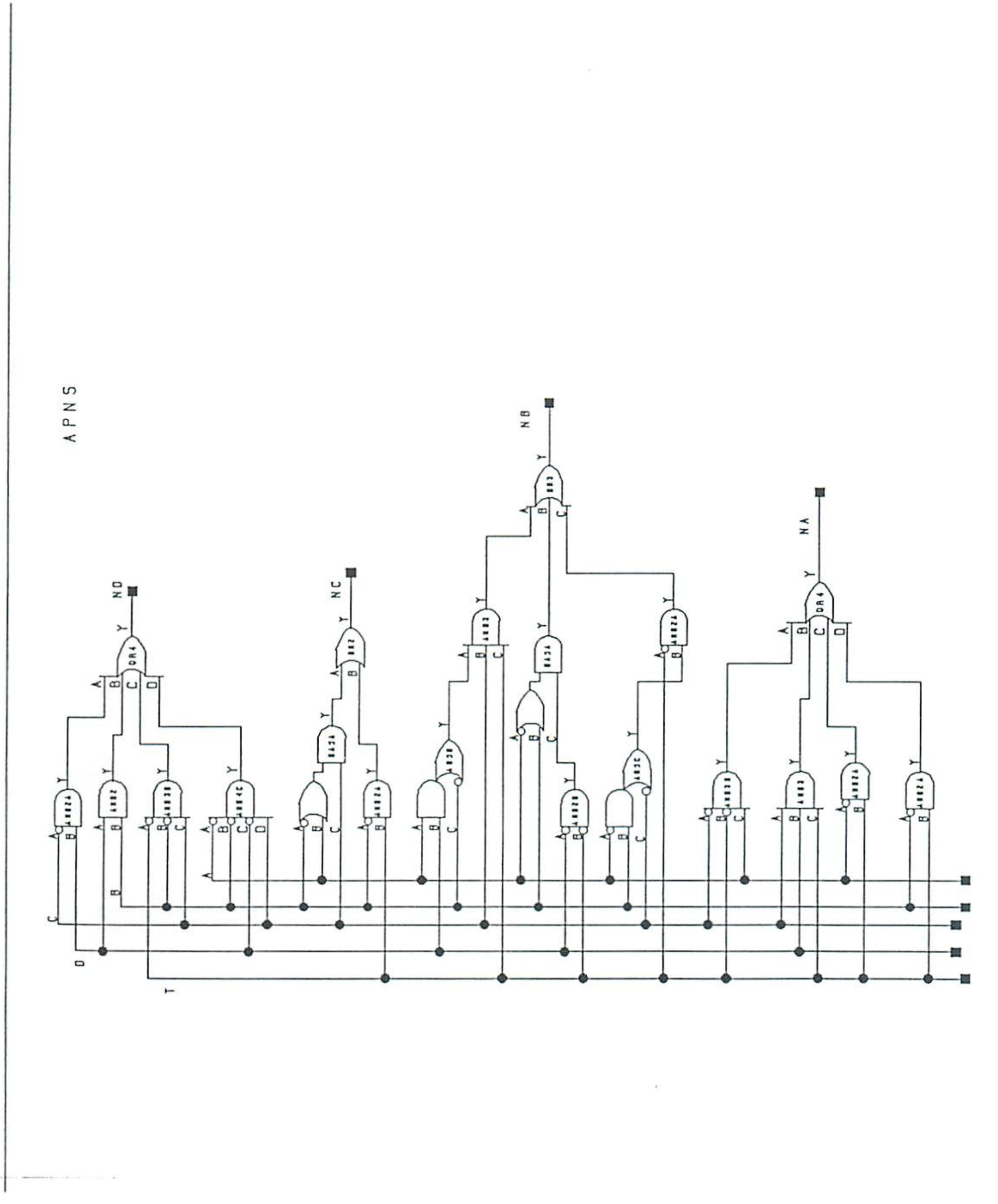


Figure B.3: Schematic of the APNS.

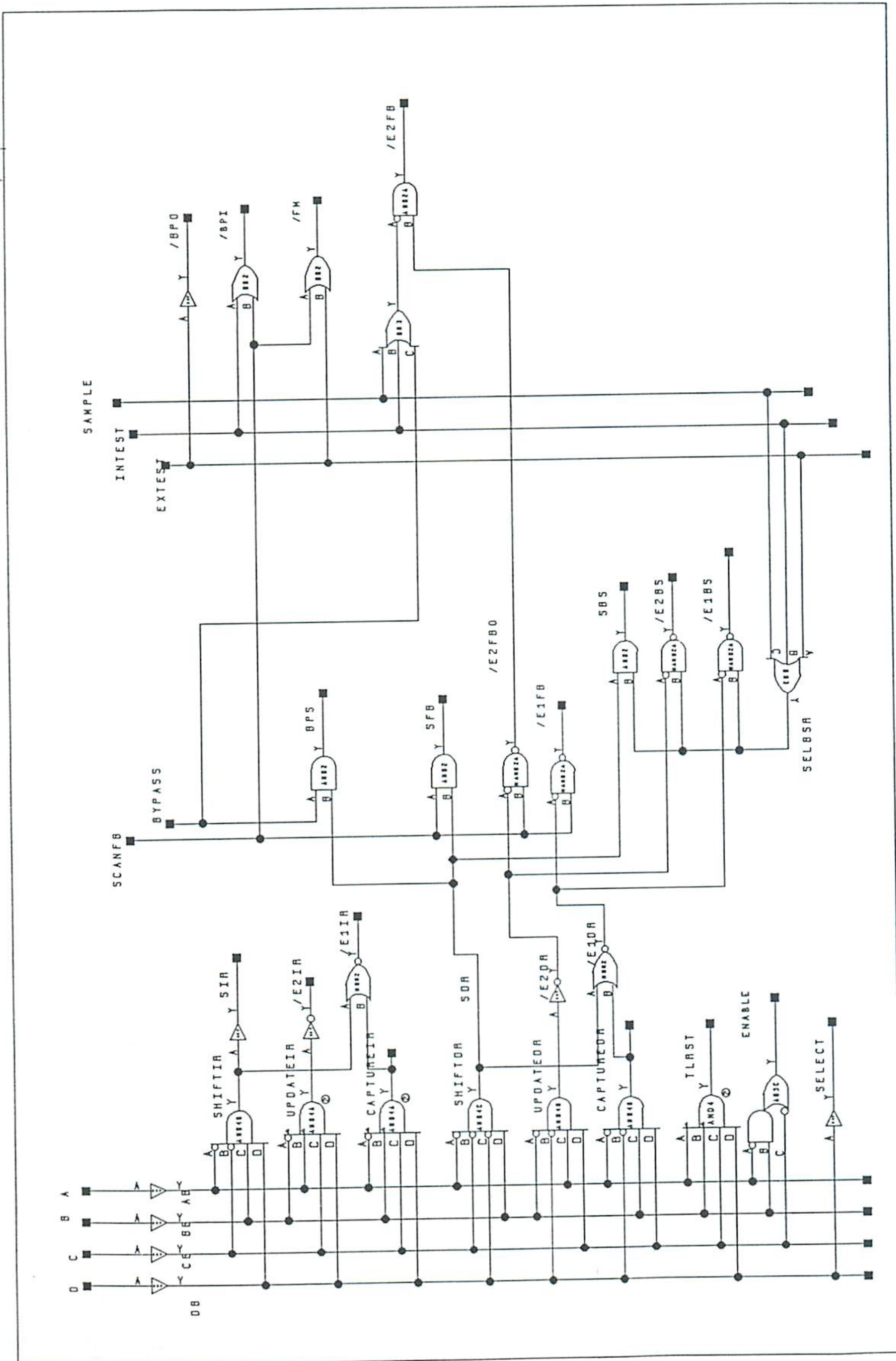


Figure B.4: Schematic of the APOL.

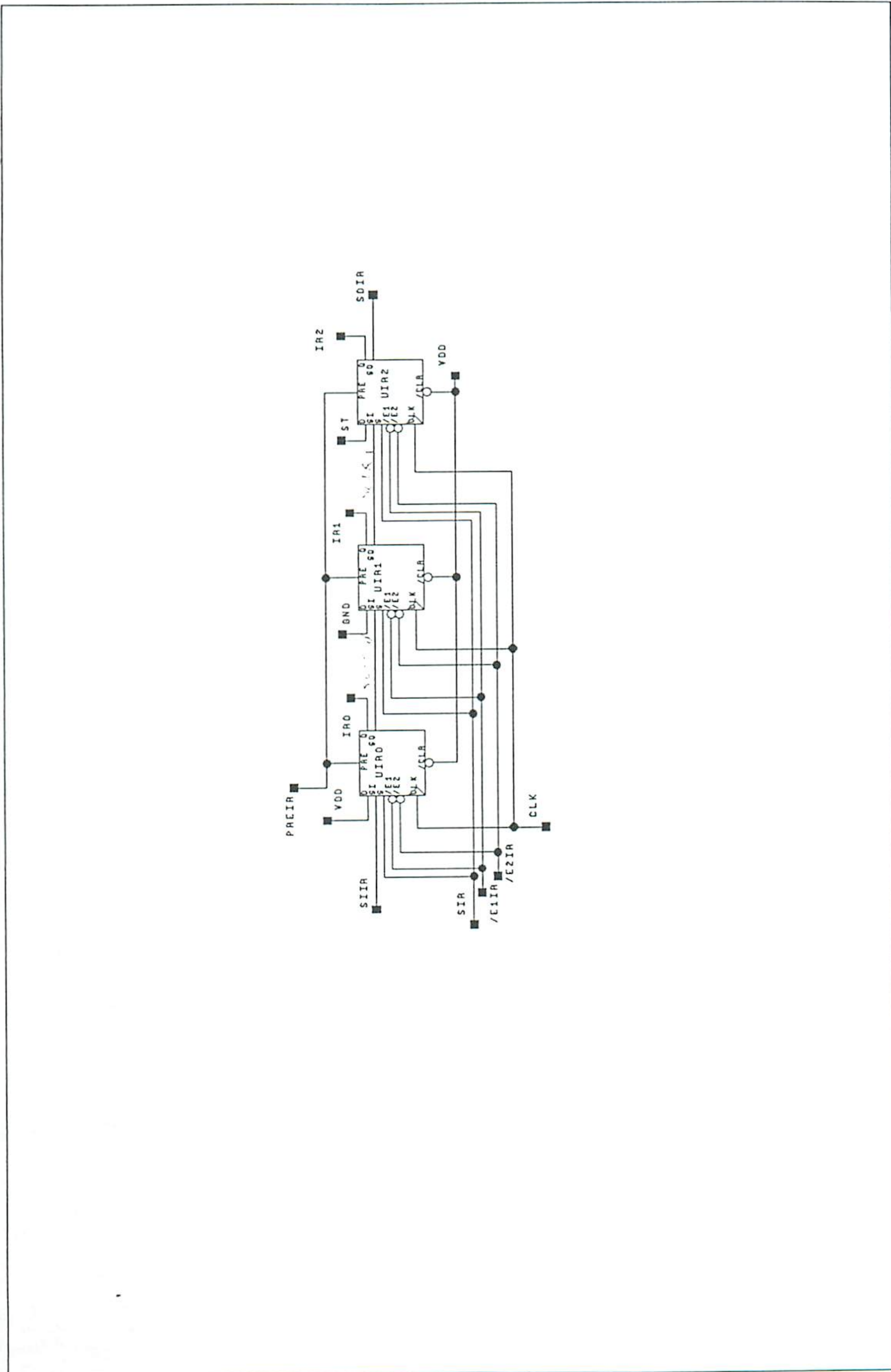


Figure B.5: Schematic of the IR.

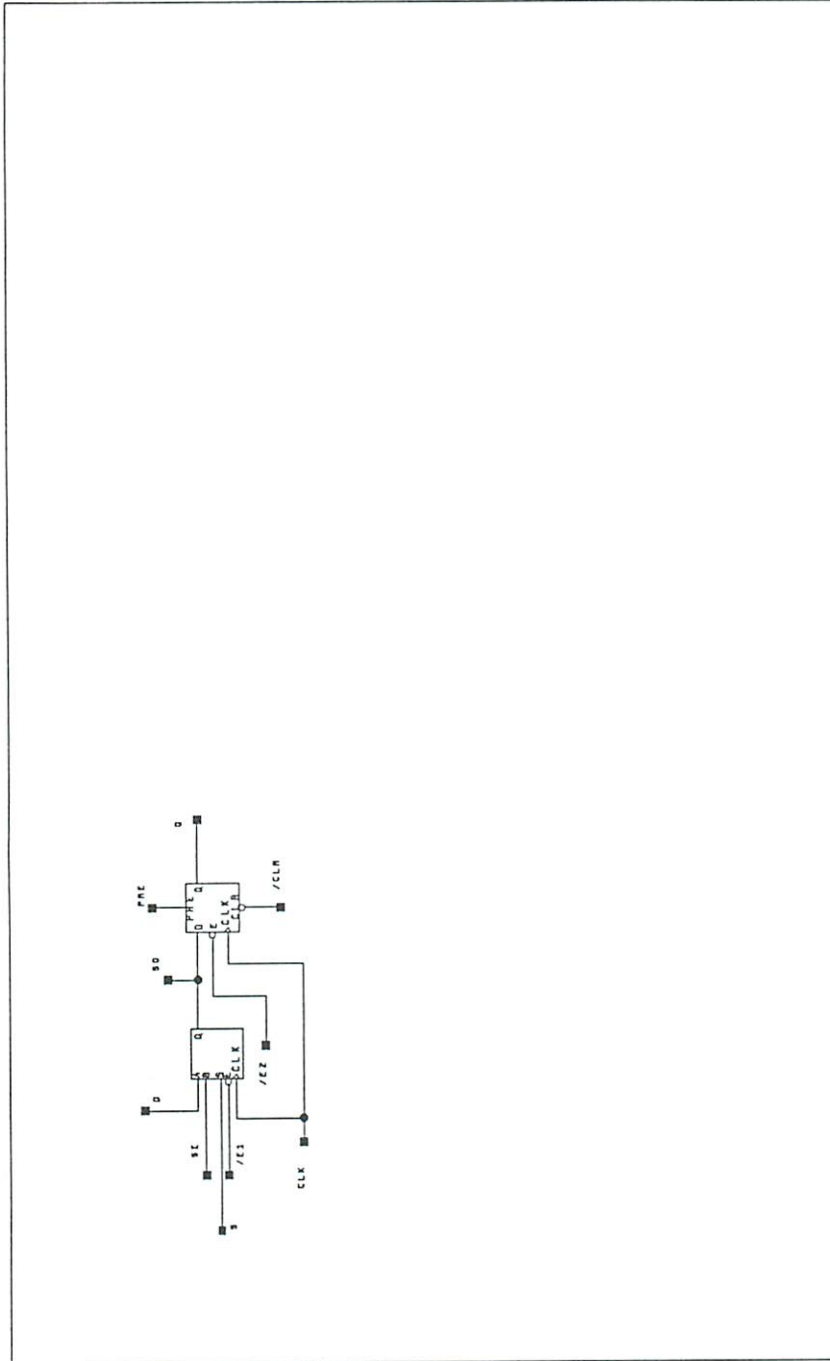


Figure B.6: Schematic of the IRCELL.

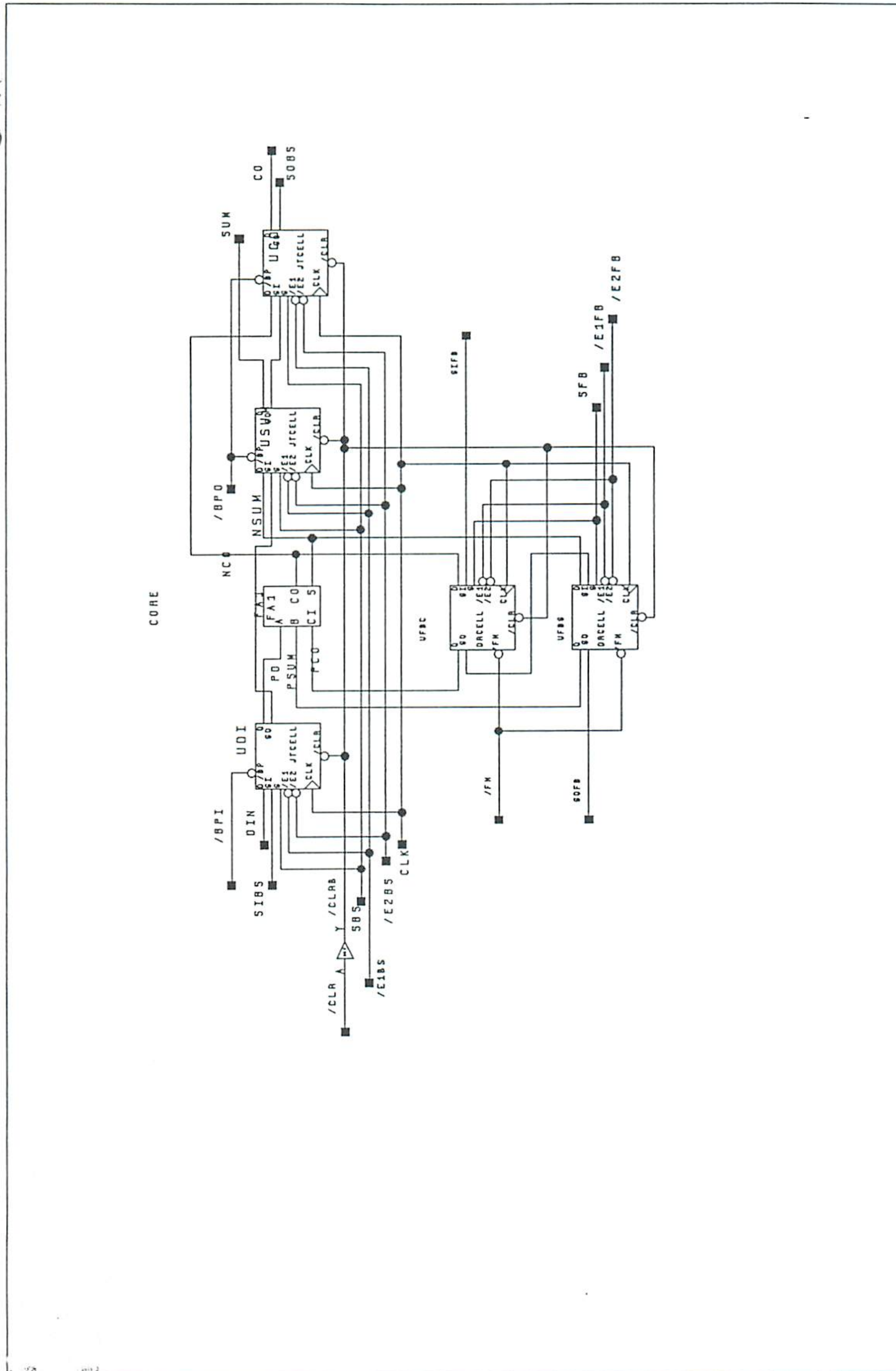


Figure B.7: Schematic of the CORE.

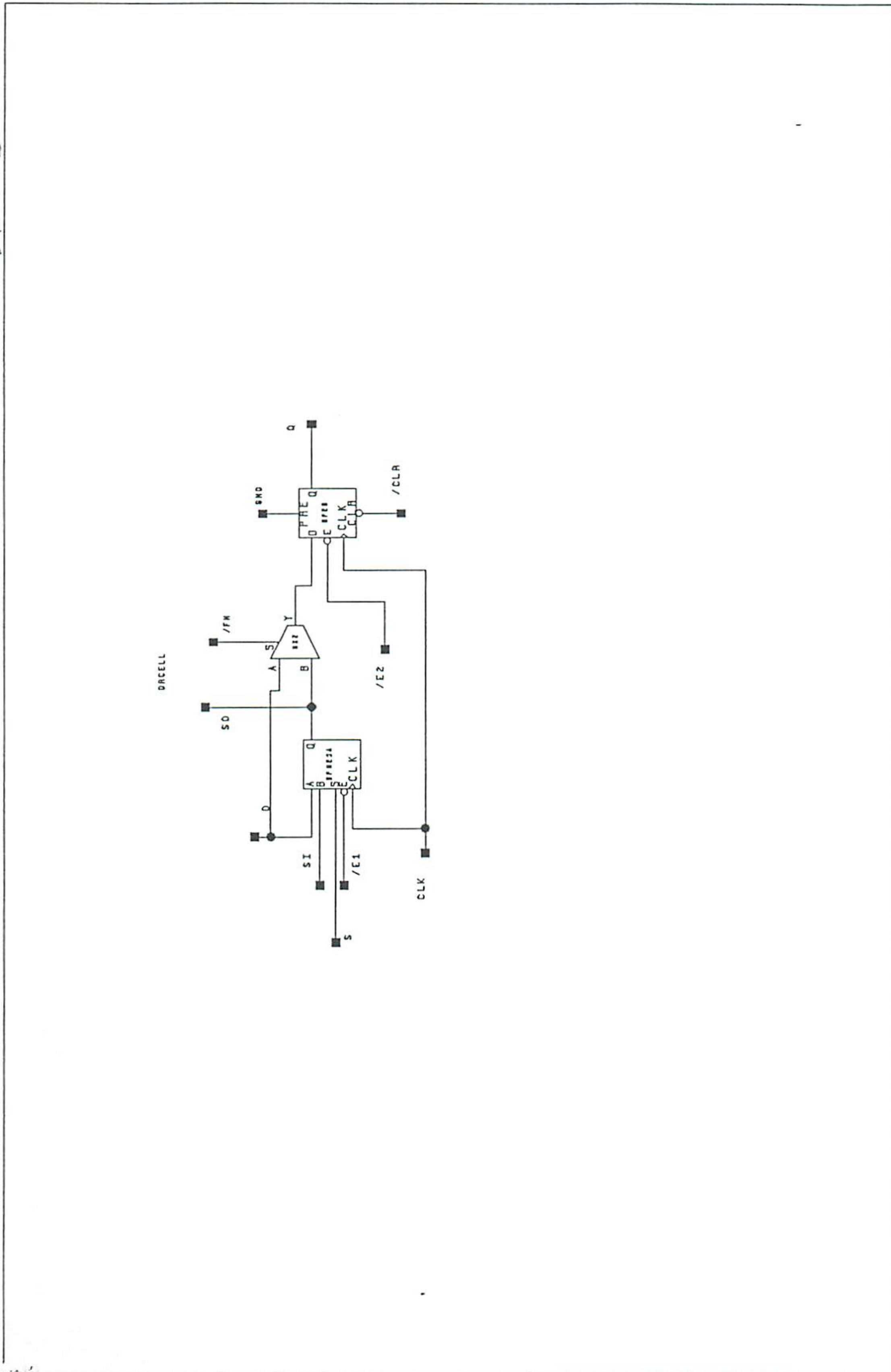


Figure B.8: Schematic of the DRCELL.

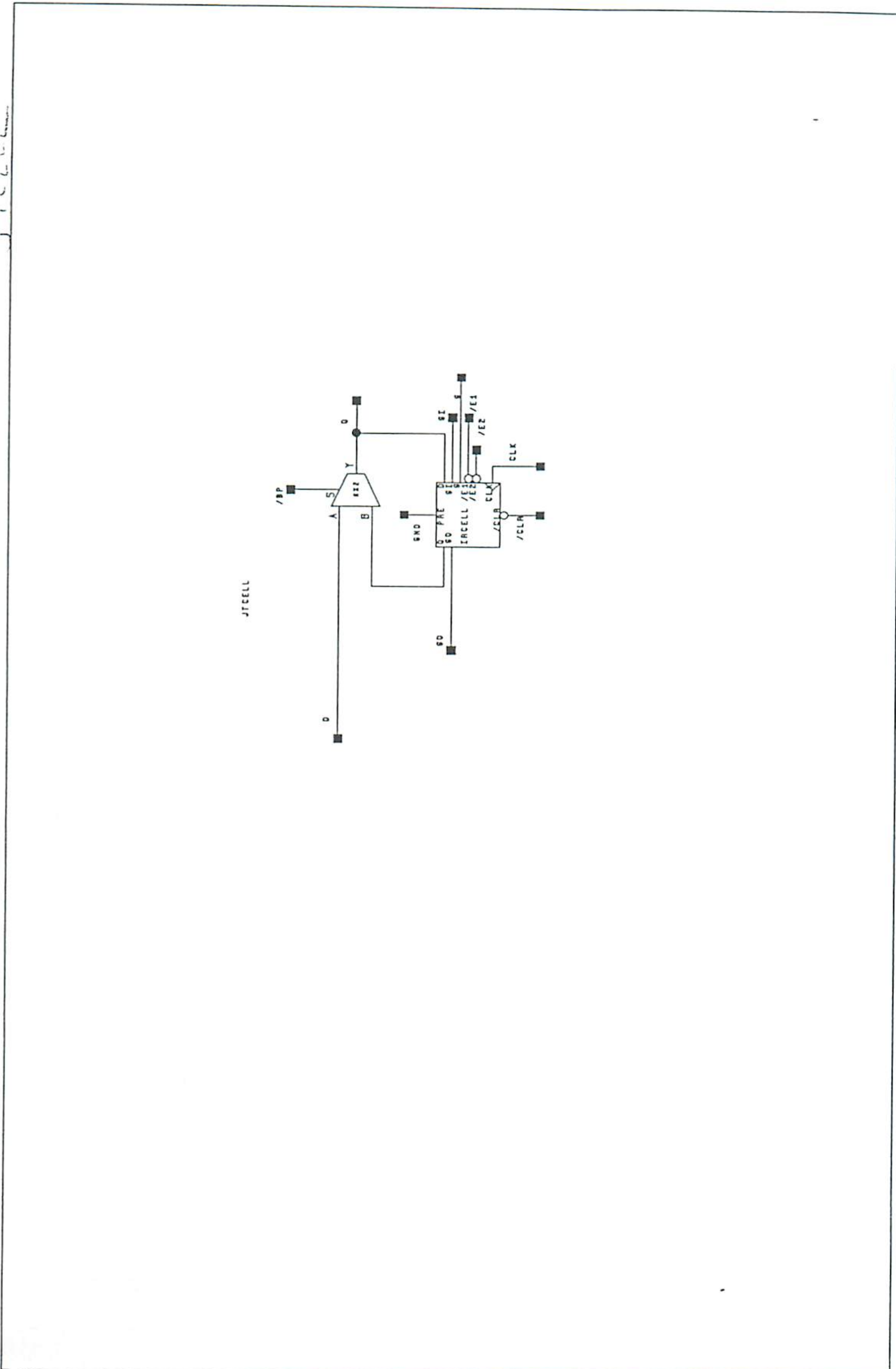


Figure B.9: Schematic of the JTCELL.

C Timing diagrams for the Test Channel

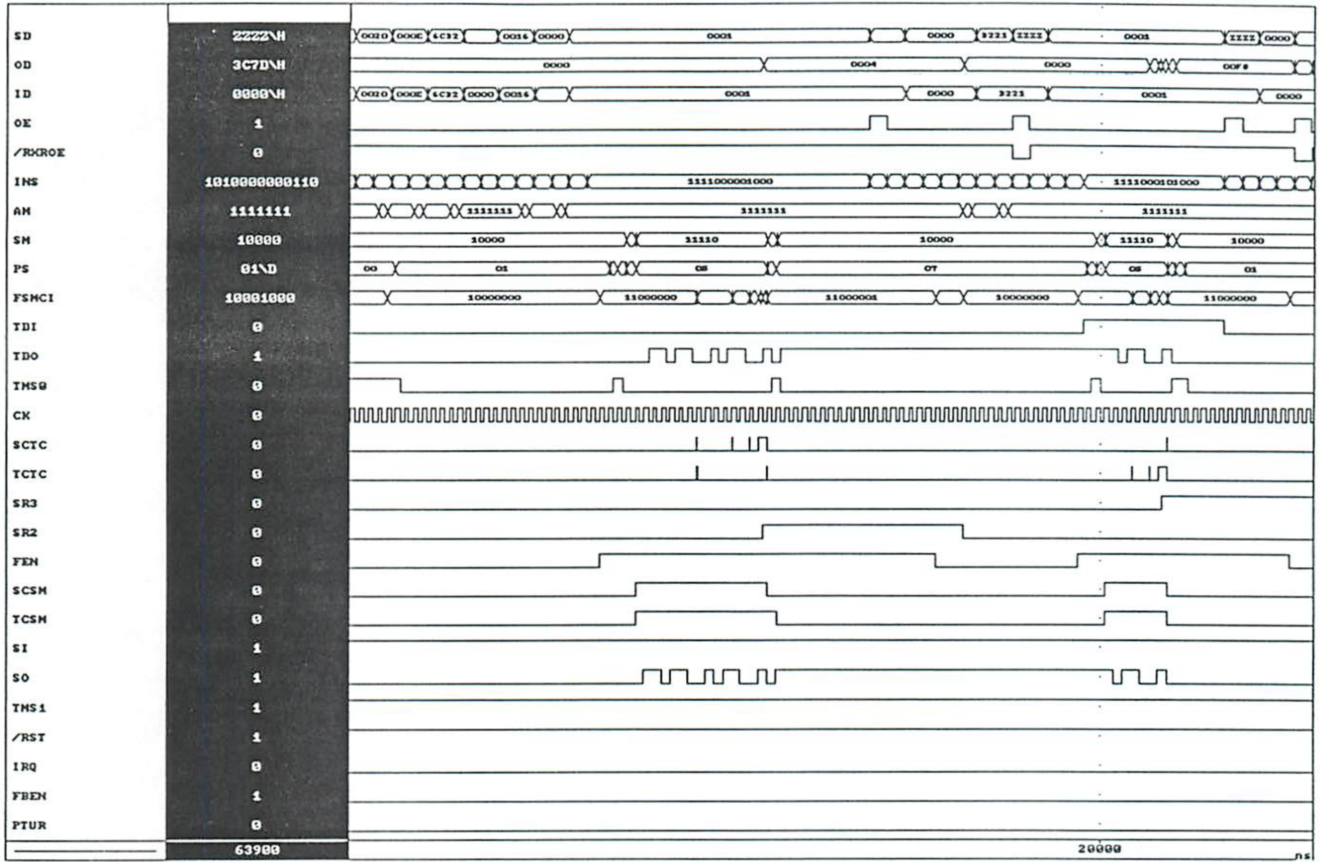


Figure C.1: Timing waveform of the Test Channel in DTUR mode.

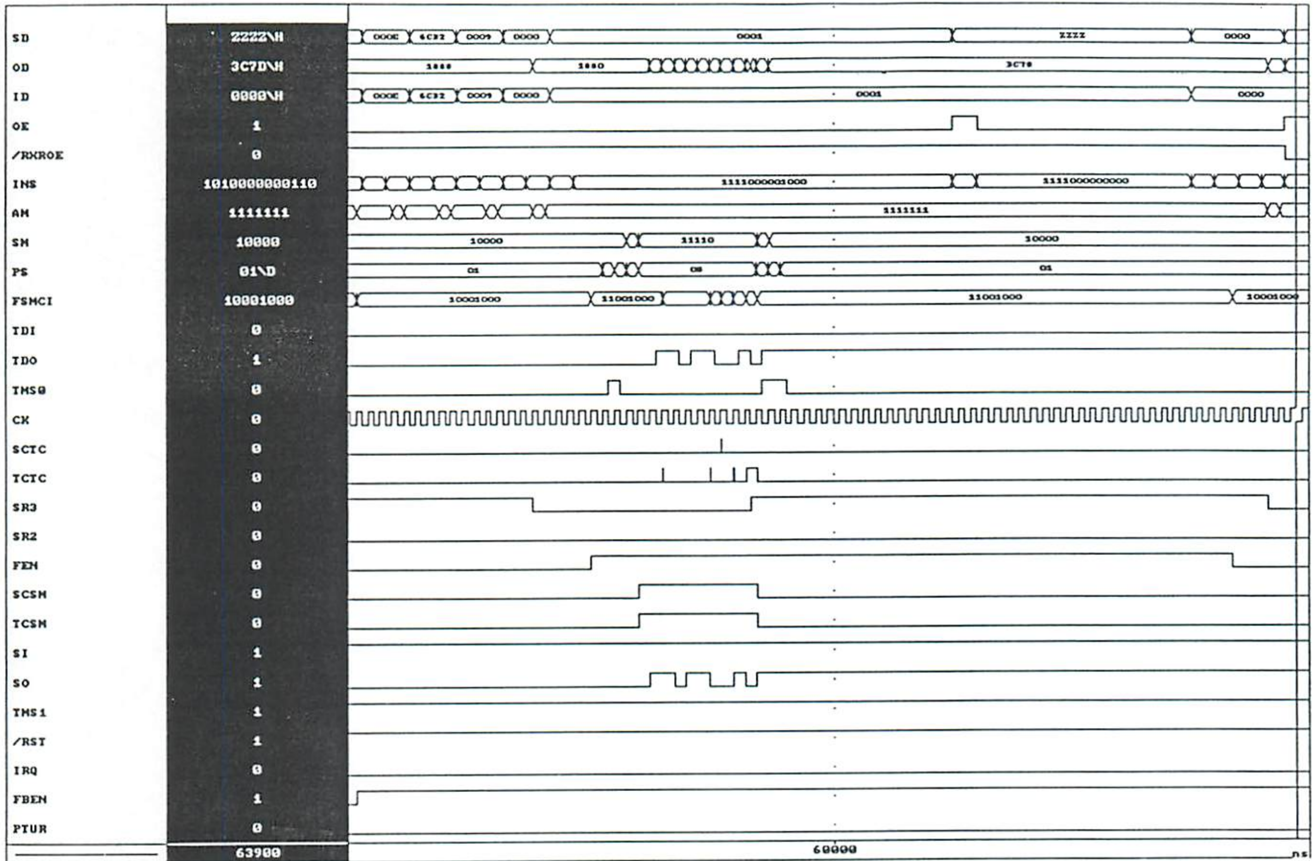


Figure C.2: Timing waveform of the Test Channel in DTCR mode.

Figure C.3: Timing waveform of the Test Channel in PTUR mode.

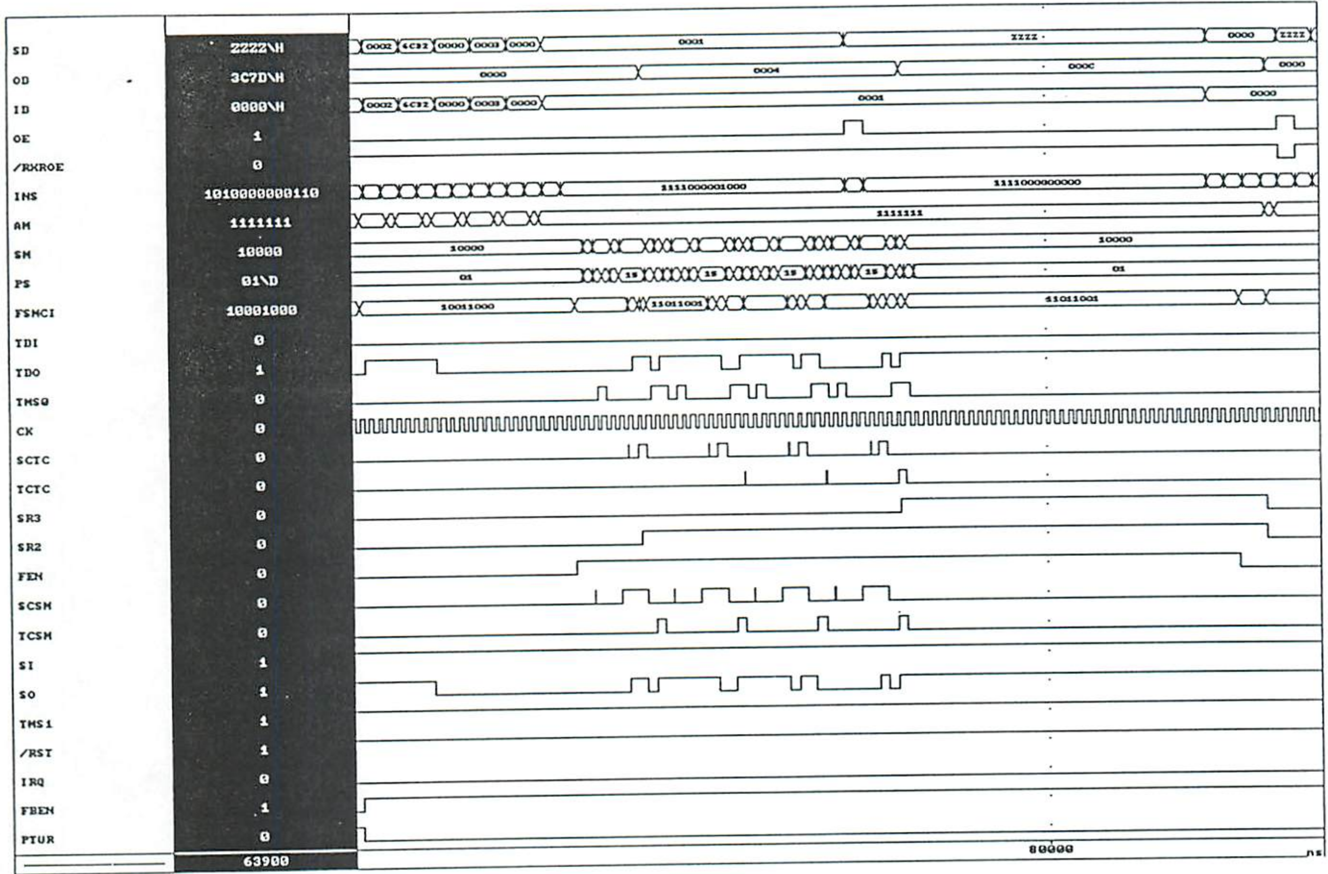


Figure C.4: Timing waveform of the Test Channel in PTCR mode.

Figure C.5: Timing waveform of the Test Channel in INS mode.

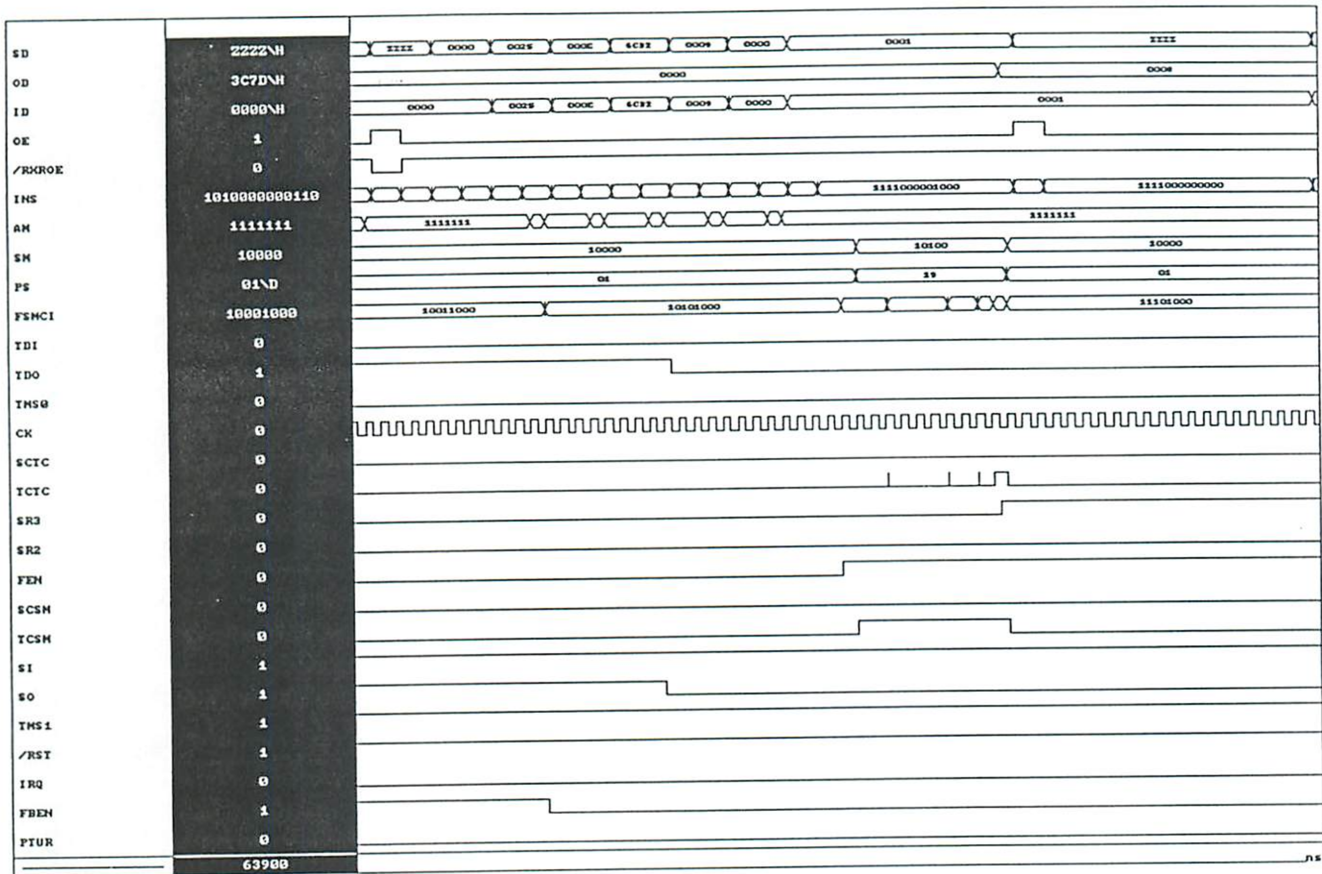


Figure C.6: Timing waveform of the Test Channel in RTEST mode.

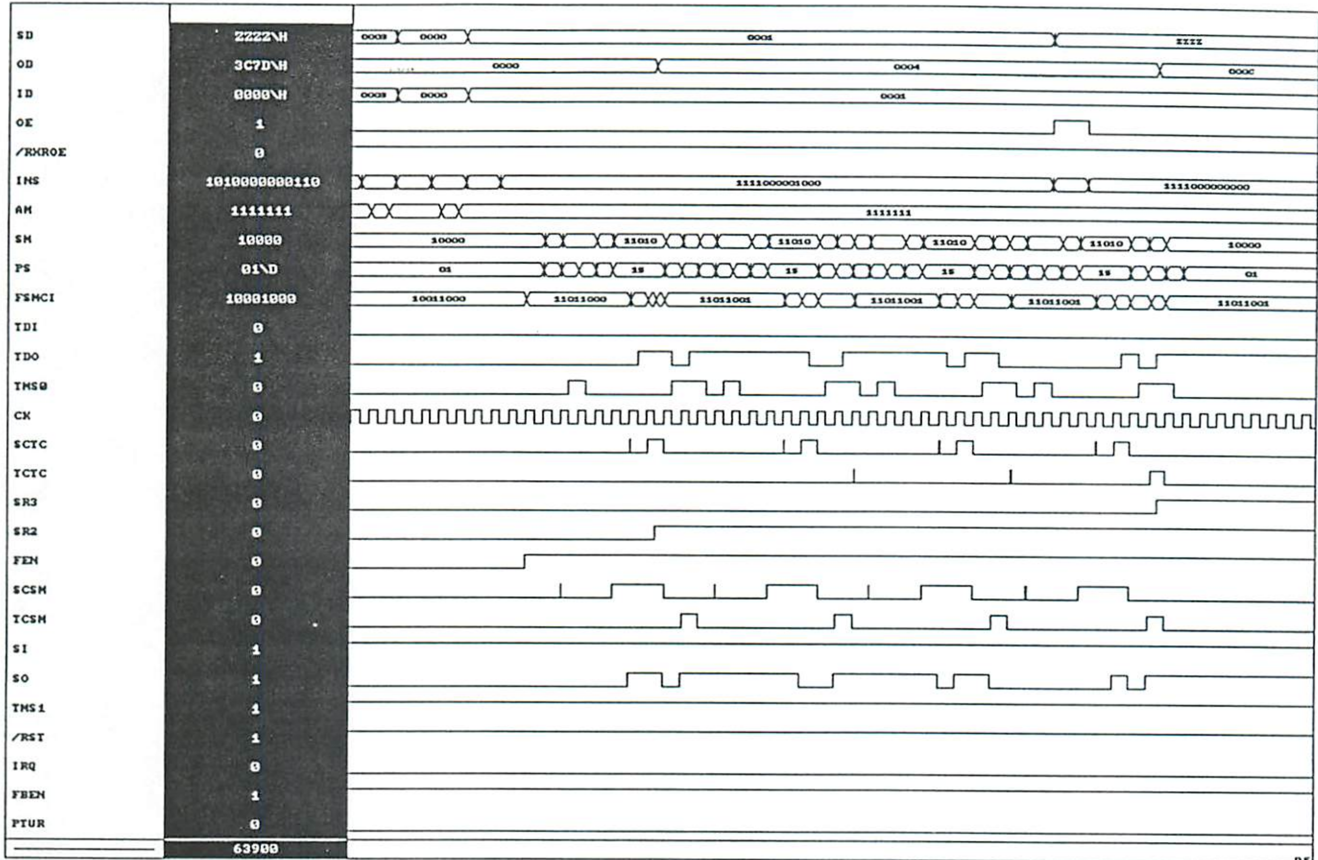


Figure C.7: Timing waveform of the Test Channel in STBUS mode.

D Test programs in Turbo Basic

```
REM extest.BAS   ---- set extest mode --- by Lien

REM  INS OPERATION +++ put app1 ir=000

REM DISABLE FEN

call writeReg(&h308, 0, 0)

REM LOAD CR

call writeReg(&h300, 0, &h24)

REM LOAD CNR

call writeReg(&h301, 0, &h0e)

REM LOAD TXR

call writeReg(&h305, 0, &h6c)

REM LOAD TCL = &H1A, high byte =0

call writeReg(&h304, 0, &h1a)

REM CLEAR SR

call writeReg(&h309, 0, 0)

REM ENABLE FEN

call writeReg(&h308, 0, 1)

GOSUB polling

REM ------ do again -------------------------

REM DISABLE FEN

call writeReg(&h308, 0, 0)

REM CLEAR SR

call writeReg(&h309, 0, 0)

REM LOAD TXR = 000C\H, SELECT BS CHAIN, IR0=[0,1,2]=000,

call writeReg(&h305, 0, &h0c)

rem -------- the above line : 0=0000 the 000 decide ir=000

call readReg(rxr, &h306)

PRINT "RXR="; bin$(rxr)

REM ENABLE FEN

call writeReg(&h308, 0, 1)

GOSUB polling

REM IF DONE, THEN APP1 IR[0,1,2]=011
```

```
REM    --- set to SCANFB mode  --- Lien
REM DISABLE FEN
call writeReg(&h308, 0, 0)
REM LOAD CR
call writeReg(&h300, 0, &h24)
REM LOAD CNR
call writeReg(&h301, 0, &h0e)
REM LOAD TXR; high byte=0
call writeReg(&h305, 0, &h6c)
REM LOAD TCL = &H1A
call writeReg(&h304, 0 &h1a)
REM CLEAR SR
call writeReg(&h309, 0, 0)
REM ENABLE FEN
call writeReg(&h308, 0, 1)
GOSUB polling
REM ------ do again -------------------------
REM DISABLE FEN
call writeReg(&h308, 0, 0)
REM CLEAR SR
writeReg(&h309, 0, 0)
REM LOAD TXR = 006C\H, SELECT fb CHAIN, IR0=[0,1,2]=011,
call writeReg(&h305, 0, &h6c)
rem -------- the above line : 6=0110 the 110 decide ir=011
REM READ RXR
call readReg(rxr, &h306)
REM ENABLE FEN
call writeReg(&h308, 0, 1)
GOSUB polling
REM IF DONE, THEN APP1 IR[0,1,2]=011
REM READ RXR again ---
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
```

Figure D.2: Program that sets the IR of the App1 to 011.

```
REM ---- send 011 to the BSR --- Lien
REM DISABLE FEN
call writeReg(&h308, 0, 0)
REM LOAD CR=20\H, MODE IS DTUR
call writeReg(&h300, 0, &h20)
REM LOAD CNR
call writeReg(&h301, 0, &h0e)
REM LOAD TXR=006c
call writeReg(&h305, 0, &h6c)
REM LOAD TCL = &H001A
call writeReg(&h304, 0, &h1a)
REM CLEAR SR
call writeReg(&h309, 0, 0)
REM ENABLE FEN
call writeReg(&h308, 0, 1)
REM wait for conditions occur
GOSUB polling
REM ------ do again -------
REM DISABLE FEN
call writeReg(&h308, 0, 0)
REM CLEAR SR
call writeReg(&h309, 0, 0)
REM load txr =003c
call writeReg(&h305, 0, &h3c)
rem -------- the above : 3=0011, the 011 decide bs=110,
REM READ RXR again ---
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
rem clear rxr
call writeReg(&h306, 0, 0)
REM ENABLE FEN
call writeReg(&h308, 0, 1)
REM wait for conditions occur
GOSUB polling
REM READ RXR again ---
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
rem --- end of dtur.bas ----
```

Figure D.3: Program that sends data to the App1.

```
REM ---- set extest mode ------------------------- by Lien
call writeReg(&h308, 0, 0)
call writeReg(&h300, 0, &h24)
call writeReg(&h301, 0, &h0e)
call writeReg(&h304, 0, &h0a)
call writeReg(&h305, 0, &h0c)
call writeReg(&h309, 0, 0)
call writeReg(&h308, 0, 1)
GOSUB polling
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
REM ---- send 010 to the data register --------------
call writeReg(&h308, 0, 0)
call writeReg(&h300, 0, &h20)
call writeReg(&h301, 0, &h0e)
call writeReg(&h304, 0, &h0a)
call writeReg(&h305, 0, &h2c)
call writeReg(&h306, 0, 0)
call writeReg(&h308, 0, 1)
GOSUB polling
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
REM set IR = 011 ------------------------------------
call writeReg(&h308, 0, 0)
call writeReg(&h300, 0, &h24)
call writeReg(&h301, 0, &h0e)
call writeReg(&h304, 0, &h0a)
call writeReg(&h305, 0, &h0c)
rem ----- the above line : 0=0110 the 110 decide ir=011
call writeReg(&h309, 0, 0)
call writeReg(&h308, 0, 1)
GOSUB polling
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
REM ---- send 010 to the data register ------------------
call writeReg(&h308, 0, 0)
call writeReg(&h300, 0, &h20)
call writeReg(&h301, 0, &h0e)
call writeReg(&h304, 0, &h0a)
call writeReg(&h305, 0, &h3c)
rem -------- the above : 2=0010, the 010 decide bs=010,
call writeReg(&h308, 0, 1)
GOSUB polling
call readReg(rxr, &h306)
PRINT "RXR="; bin$(rxr)
```

Figure D.4: Program that applies a test vector to the full adder.

Efficient Testing of Acyclic Structures in Partial Scan Designs

BY

Rajesh Gupta and Melvin A. Breuer

Technical Report No. CENG 90-13

April 1990

Electrical Engineering - Systems Department

University of Southern California

Los Angeles, CA. 90089-0781

Efficient Testing of Acyclic Structures in Partial Scan Designs*

Rajesh Gupta and Melvin A. Breuer

Department of Electrical Engineering–Systems
University of Southern California
Los Angeles, California 90089-0781
Phone: (213) 743-7258
Fax: (213) 745-7284
E-mail: gupta@usc.edu

April 1990

*This work was supported in part by the Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under Contract No. N00014-87-K-0861, and in part by the Semiconductor Research Corporation under Contract No. 88-DP-075. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U. S. Government.

Abstract

It is well known that acyclic sequential structures are considerably easier to test than arbitrary sequential circuits. Hence some partial scan techniques attempt to simplify the test generation problem by ensuring that the portion of the circuit effectively under test is acyclic. In such designs the test time is dominated by the shifting of test patterns into and out of the scan path. We present a compacting technique that minimizes the number of test patterns required to detect an arbitrary fault. A modified test schedule is used in which each compacted pattern is held in the scan path until the next pattern is ready to be shifted in. An optimal test scheduling algorithm is presented which determines the minimum possible number of compacted test patterns required for an arbitrary fault based on the circuit structure. Using the optimal schedule we derive a condensed combinational test generation model (TGM) for combinational ATPG under a multiple fault model. This TGM replaces the iterative array used in traditional sequential ATPG. This model allows the detection of any arbitrary fault using a minimum possible number of distinct test patterns.

1 Introduction

Automatic test pattern generation (ATPG) for acyclic sequential structures is known to require substantially lower computation effort than for arbitrary sequential structures [1]. Partial scan approaches that make use of this fact have recently been proposed [2,3]. Essentially they select flip-flops (FFs) to be included in the scan path such that the portion of the circuit effectively under test, the **kernel**, is either acyclic or close to acyclic. Test generation is then carried out for the kernel to determine test sequences; the complexity of this computation is similar to that for combinational circuits. This approach requires that while in the test mode the clock signals for scan FFs should be controllable independently of the clock signals for the non-scan FFs. A test sequence can then be applied to the kernel using the following two steps alternately:

1. Serially shift a test pattern into the scan path while disabling the clock signal feeding the non-scan FFs (this effectively puts the non-scan FFs in a HOLD mode);
2. While disabling the clock signal feeding the scan FFs (putting them in a HOLD mode), activate the clock signal for the non-scan FFs for one clock cycle (this enables test data to propagate through one level in the kernel).

A *test sequence* for a fault in a sequential circuit consists of a set of consecutive time frames, in each of which patterns containing both specified and don't-care values may need to be applied at the various inputs. The *length* of a sequence is the total number of time frames in it. For an acyclic structure the length is related to the **depth** or the highest number of FFs in any path in the structure. If d is the depth of a structure, the test sequence length is bounded by $d + 1$. However, a given test sequence may contain unassigned or don't-care input values such that not all primary inputs need to be provided with new data at each of the $d+1$ clock cycles. If the inputs and outputs of the structure under test are directly accessible, the time for applying the sequence is $d + 1$ clock cycles and is not affected by the presence of don't-care inputs. However, in a partial scan design many of the inputs and outputs of the structure are accessed by shifting data serially. Hence the presence of don't-care input values could potentially lead to a great saving in test time. In such circuits, where the length of the scan path is usually much higher than d, the test time is dominated by the time to shift new patterns into and out of the scan path.

An example of acyclic structures which need less than $d + 1$ input patterns are *balanced structures* [2]; these are a class of structures that require only a single-pattern test for any fault. The BALLAST methodology uses this fact by selecting scan path FFs so as to make the kernel balanced, i.e., every path between any two points in the kernel has the same number of FFs. Each test pattern is held in the scan path for $d + 1$ clock cycles. (In some cases the need to hold the test data can be eliminated by ordering the scan path appropriately and manipulating the test data [4].) This technique guarantees

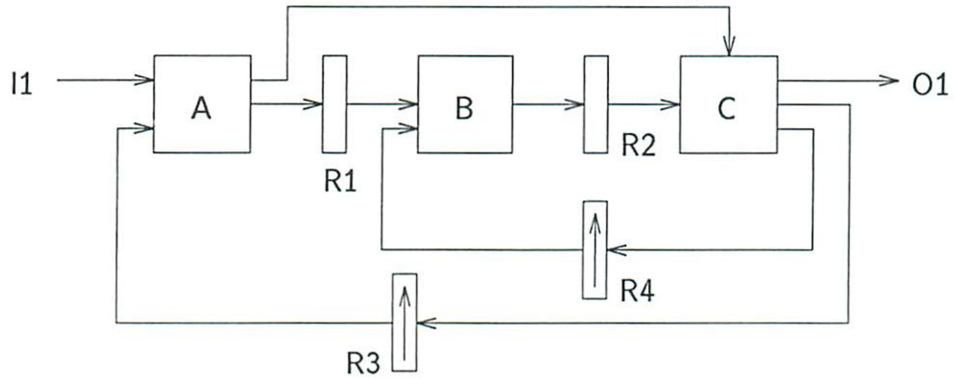
that every detectable fault is single-pattern testable. Test patterns are obtained simply by replacing all FFs in the kernel by wires and running combinational ATPG on the resulting **combinational equivalent**. The number of FFs to be placed in the scan path in this approach, however, is higher than that required to just make the kernel acyclic.

In this paper we study the implications of using a kernel that has an acyclic structure but may be unbalanced. A branch-and-bound algorithm for determining a minimal set of registers to be made scannable such that the kernel is acyclic can be found in [2]. Clearly the area overhead for this case is lower than for a balanced kernel. But a simple combinational equivalent cannot be used for ATPG. Further, any given fault may require up to $d+1$ patterns to detect it.

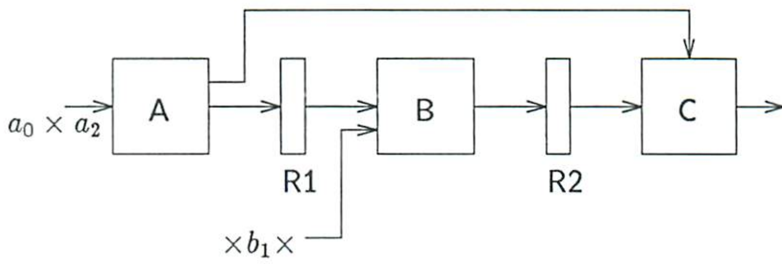
For example, consider the circuit shown in Figure 1(a) consisting of combinational logic blocks A, B, C interconnected with registers. Each connection shown may consist of any number of wires. Registers R3 and R4 are selected to be made scannable since the resulting kernel, shown in Figure 1(b), is acyclic. Note that the kernel is obtained from the original circuit simply by removing the scan registers and replacing them by pseudo-inputs/outputs. In the kernel, inputs/outputs connected to the same combinational logic block are merged together; thus the kernel effectively has one output at C which feeds scan path registers and the primary output O1, and inputs at both A and B are fed by scan registers and/or the primary input I1. The depth of this kernel is 2.

Typical sequential ATPG programs would construct an *iterative array* consisting of up to 3 copies of the kernel, each representing one time frame, and attempt to find a test sequence for a given fault (if one exists) within these time frames. Since the kernel has only one primary output at C, any test sequence for a fault must propagate an error to this output. Assume that the error is visible at the output at time frame 2. Because of the topology of the circuit this output value must depend only on the input values at A at time steps 0 and 2 and on the input value at B at time frame 1. All other input values are essentially "don't-cares" in all possible test sequences and are indicated by '×' in the input sequences in Figure 1(b) (to be applied in the order from left to right). Note that each '×' represents a vector of don't-care values. An ATPG program would normally assign random logic values to these inputs. This means that approximately half the test time in this case could be taken up in shifting random data into the scan path.

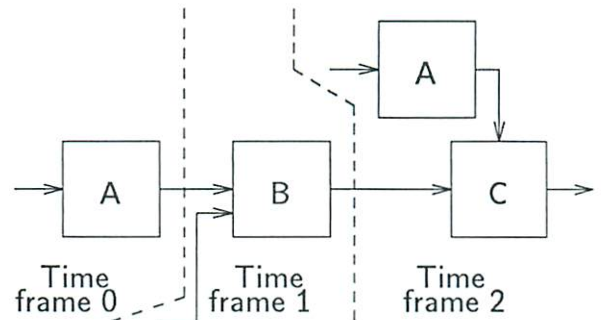
The effectiveness of this test process can be improved in two ways. First, the test pattern generator can be enhanced to fill in the unassigned input values with deterministic patterns that detect one or more additional faults, rather than with random data. In general there is no guarantee that any additional faults can be detected in this way, especially if most of the circuit faults have already been covered. Second, the test sequence can be compacted so that the shifting time is reduced without reducing its effectiveness. The latter approach is the subject of this paper. In the simple example of Figure 1, A has its input value specified at time frame 0 but not at 1, while B has its input value specified at 1 but not at 0. Hence we can combine the first two patterns by shifting a_0 and b_1 simultane-



(a) Partial scan design



(b) Acyclic kernel



(c) Test generation model

Figure 1: Example of partial scan design.

ously into the scan path, holding them there for two clock cycles instead of one, and then shifting in a_2. This is a modification of the basic test procedure described in Section 1. Thus the number of *shift cycles* of the scan path (i.e., the number of times a new pattern is shifted in) is reduced from three to two without losing any of the deterministic part of the test sequence. Note that this applies irrespective of which fault is under test. Intuitively, the fact that there are two "unbalanced" paths from A to D with unequal delays indicates that in general two distinct input patterns at A will be required to guarantee detection of an arbitrary fault.

In Section 2 we present a more formal and general discussion of how to test unbalanced acyclic structures. We use a formal model to compute a lower bound on the number of shift cycles required to test for an arbitrary fault, and present a test compaction algorithm that achieves the lower bound. In Section 3 we study the problem of simplifying the iterative array model used for ATPG by making use of the fact that the kernel is acyclic. For example, Figure 1(c) shows a test generation model (TGM) consisting of a reduced iterative array that is sufficient for any fault in the kernel. The TGM can be further condensed based on the compacted schedule that will be used for test application, as described in Section 3. Conclusions are presented in Section 4.

2 Optimal Test Scheduling

A given test sequence for a partial scan design whose kernel is an acyclic structure of depth d may consist of up to $d+1$ time frames. Our objective is to find a way of compacting the patterns in a test sequence so as to minimize the number of time frames at which new data needs to be applied. This ensures that when the test patterns are applied using the scan path, the shifting time (which usually dominates the test time) is minimized. Assume that the time frames are numbered from 0 (the earliest) to d (the latest, at which the fault gets detected). We define the **schedule** as the list of time frames in the test sequence that require new data to be shifted in, in ascending order. Thus a schedule $(0, 1, 2, \ldots, d)$ means that new data is shifted in at every time frame, while (0) represents a single-pattern test. In the example of Figure 1 presented earlier time frames 0 and 1 are combined together, hence the schedule is $(0, 2)$.

We shall refer to each element in the schedule as a **shift step**, since in the corresponding time frame a new pattern needs to be shifted into the scan path, and each element not in the schedule as a **hold step**, since it requires the contents of the scan path to be held for an additional clock cycle. Note that the test pattern scanned in during a given shift step i in the schedule $(\ldots, i, j, \ldots)$ is actually the result of compacting the test patterns for time frames i, $i+1$, $i+2$, $\ldots$, $j-1$.

The Compaction Principle

In our earlier example using Figure 1, we combined the test patterns for time frames 0 and 1 because neither of the inputs at A or B need to have a value specified in *both* frames. This compaction applies to all test sequences in this example. Before studying more complex cases we state the following principle that governs our compaction problem. The term *minimal test sequence* refers to a test sequence in which all unspecified input values are left as don't-care values.

Compaction Principle: *A set of consecutive time frames in a minimal test sequence may be compacted together into a single shift step in a schedule only if no input is required to be assigned values in more than one of these time frames.*

We will apply this compaction principle before test pattern generation is actually carried out. Note that the principle does not make use of the actual values of the test patterns; it uses only the information, derived from the circuit structure, about which input values can be specified and which must be don't-cares in various time frames. In a single-output acyclic structure, such as our previous example, the required information about the input values can be derived using the following rule: An input can be assigned a value in time frame x if there exists a path from that input to the output that passes through $d - x$ FFs (assuming the error is first observed at the output in time frame d). Later in this section we describe how to determine an optimally compacted schedule that satisfies the compression principle based on this information.

Modeling Schedule Constraints

Let us now consider the multi-output structure in Figure 2 in which blocks A, B, C, etc. are combinational and unlabeled blocks are registers. Its depth d is 4 and it has three inputs and four outputs, all of arbitrary width. As before the inputs and outputs are accessible only through a scan path which is not shown. Given an arbitrary fault in the circuit, a test sequence may propagate the fault to any of the outputs. At some outputs it may be possible to detect a fault at a time frame earlier than d. However, note that the same test sequence displaced in time can be used to detect a fault at different time frames. Hence we shall assume without loss of generality that a fault is to be detected at time frame d. This is justified since any fault that is propagated into the scan path at time frame d will be observed during the first shift step for the subsequent test sequence. This assumption will lead to a simplified test generation model discussed in Section 3.

The patterns shown at each input in Figure 2 indicate the time frames at which an input may possibly need to be specified in order to detect a fault at one of the outputs at time frame d. This information is determined in the same way as in the single-output case, except that for a given input, paths to all outputs have to be taken into account. Thus

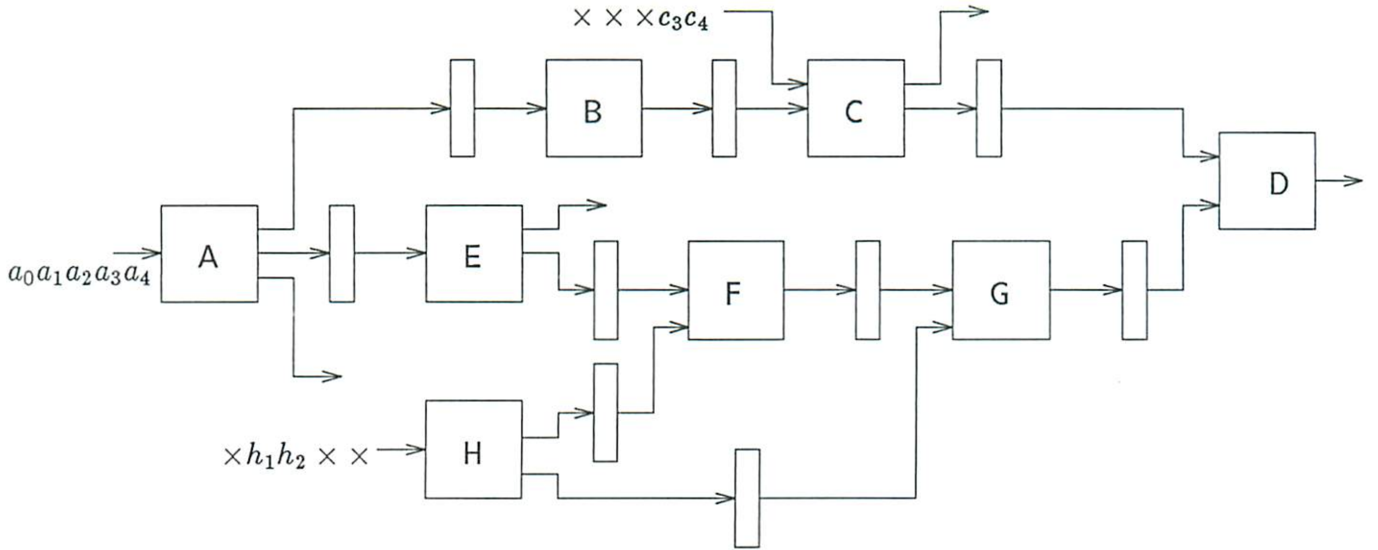


Figure 2: Example of acyclic kernel.

Output	Input values on which the output depends		
	Sequence at A	Sequence at C	Sequence at H
A	× × × × a_4	—	—
C	× × a_2 × ×	× × × × c_4	—
D	a_0 a_1 × ××	× × ×c_3×	×h_1 h_2 × ×
E	× × ×a_3×	—	—

Table 1: Relationships among inputs and outputs.

for example the input to C has values at time frames 3 and 4 but is always unspecified at all other times. The input to A may need specified values at any time frame, because corresponding to each time frame there is some path with the appropriate number of FFs ending in time frame d at one of the outputs. Hence it appears at first glance that the compaction principle will not allow a reduced test schedule.

However, to detect any fault it is sufficient to propagate it to just one of the outputs. If some test sequence for a fault propagates an error to more than one output, this implies that there may exist a reduced form of the same sequence that propagates it to only one output. Table 1 shows, for each output, what input values need to be specified such that the fault is observed at that output at the end of the 4th time frame. The table shows that no test sequence would require all 5 values at A to be specified. Also it is clear that all 5 frames cannot be compacted together, since the output at D (which we shall refer to as D for short) requires two different values at A and also two different values at H. Under these constraints it seems intuitively clear that a bare minimum of two shift steps will be needed in the schedule for an arbitrary test sequence in order to satisfy the compaction principle stated earlier. A model for representing the schedule constraints is described below.

Given an input x and an output y, let $\sigma(x, y)$ be defined as the ordered list of time frames at which the input sequence at x for output y can have specified values. Thus for example Table 1 shows that $\sigma(A, D) = (0, 1)$ and $\sigma(H, C) = (\,)$. $|\sigma(x, y)|$ denotes the number of elements in $\sigma(x, y)$. We shall attempt to find a minimal schedule by constructing a **schedule constraint graph** (SCG). We define an SCG as a directed graph $G = (V, A)$ where $V = (0, 1, 2, \ldots, d)$ represents the set of time frames and an arc (f_1, f_2) in A implies that frame f_1 must occur strictly before frame f_2 in any compacted test sequence. An SCG is constructed using the following procedure, which takes as input the values $\sigma(x, y)$ for all inputs x and all outputs y.

procedure constructSCG (σ): Returns schedule constraint graph, $G = (V, A)$.

```
{
  V ← {0,1,2,...,d}, where d = depth of the circuit;
  A ← { };
  For all input-output pairs (x,y) of the circuit such that |σ(x,y)| ≥ 2 do:
  /* Add constraints corresponding to this input-output pair */
  {
    L ← σ(x,y);
    While |L| ≥ 2 do:
    {
      i ← first element of L;
      j ← second element of L;
      Remove i from L;
      /* Time frames i and j cannot be compacted together */
      For each k, 0 ≤ k ≤ i, do:
        A ← A ∪ {(k,j)};
      For each k, j < k ≤ d, do:
        A ← A ∪ {(i,k)};
    }
  }
}
```
□

For the circuit of Figure 2 the construction of the SCG is illustrated in Figure 3. We begin with the set of nodes $V = \{0, 1, 2, 3, 4\}$ and no arcs in A. Referring to Table 1, there are two input-output pairs that may contribute to arcs in the SCG: $\sigma(H, D) = (1, 2)$ and $\sigma(A, D) = (0, 1)$. The fact that $\sigma(H, D) = (1, 2)$ implies that there must be a shift step separating time frames 1 and 2 since distinct test patterns may be required at input H. In terms of constraints on the schedule, this implies that:

1. All time frames up to and including time frame 1 must occur before time frame 2 in the schedule; and
2. All time frames including 2 and beyond must occur after time frame 1 in the schedule.

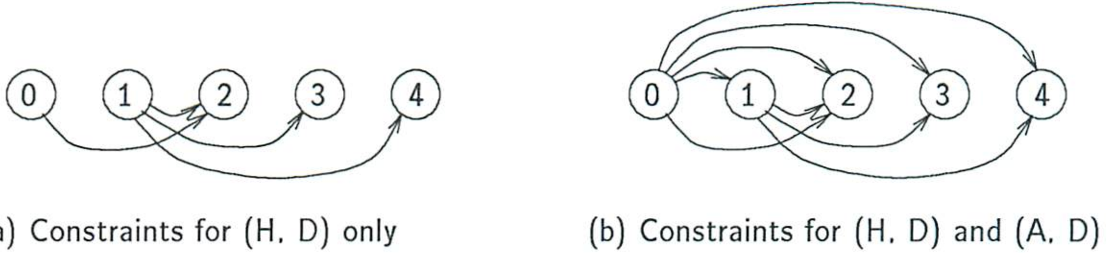


Figure 3: Construction of schedule constraint graph.

The first item above contributes arcs (0, 2) and (1, 2), while the second contributes arcs (1, 3) and (1, 4). Thus the constraints due to the input-output pair (H, D) translate into the arcs shown in Figure 3(a), which is the result of the first iteration of the outer 'for' loop in the procedure. In the second iteration the constraints due to the pair (A, D) are added, resulting in the completed SCG shown in Figure 3(b). Note that in the above example it is not sufficient to have only the arcs (0, 1) and (1, 2) in the SCG. By adding the other arcs we are explicitly encoding the fact that although some time frames represented by nodes in V may be compacted with others, they can never be scheduled in reverse order.[1]

In the procedure **constructSCG**, the outer 'for' loop may be repeated for all N_I inputs and all N_O outputs. Within the loop the time complexity is $O(d^2)$, hence the overall time complexity is $O(N_I N_O d^2)$.

Picking a Schedule

The SCG is essentially a representation of information on which time frames may be compacted together and which may not. Based on the SCG we are in a position to make the following statements about the schedules resulting from compaction.

Lemma 1 *Given a sequence of frame numbers, $S = (f_1, f_2, \ldots, f_n)$, where $0 = f_1 < f_2 < \cdots < f_n \leq d$, the compacted schedule denoted by S satisfies the compaction principle if for any arc (a, b) in the SCG, there is some f_i in S such that $a < f_i \leq b$.*

Proof Assume that for all arcs (a, b) in the SCG, there is some f_i in S such that $a < f_i \leq b$. Assume for the purpose of contradiction that the compaction principle is violated by S. Then there must be some input of the circuit that needs distinct values in some time frames a and b that are compacted into the same shift step in S. This implies that the SCG has an arc (a, b). But the fact that a and b are in the same shift step also implies that there is no f_i in S such that $a < f_i \leq b$, which is a contradiction. □

[1]With these constraints encoded explicitly, our problem is actually a special restriction of the equal execution time job scheduling problem [5][p. 402] with the number of processors not less than the number of jobs.

The above lemma essentially means that a given schedule S is valid, i.e., does not violate the compaction principle, if no two time frames that have an arc between them in the SCG are merged within the same shift step.

Lemma 2 *The number of nodes in the longest directed path in the schedule constraint graph is a lower bound on the number of steps in any schedule that satisfies the compaction principle.*

Proof Let P be a longest path in the SCG and let it consist of the δ nodes f_1, f_2, ..., f_δ in sequence. From the construction of the SCG, every arc (f_i, f_{i+1}) implies that if the frames f_i and f_{i+1} were compacted into the same shift step, the compaction principle would be violated. Hence there cannot be less than δ shift steps in any valid schedule. □

In Figure 3(b) the path consisting of nodes 0, 1, 2 is the longest, hence at least three shift steps are required in any compacted schedule.

Our problem is now to find a schedule $(f_1, f_2, \ldots, f_\delta)$ of minimum length that satisfies the following condition: given any arc (a, b) in A, the nodes a and b must not be compacted into the same shift step in the schedule, i.e., there must be an f_i in the schedule such that $a < f_i \leq b$. We present below a greedy algorithm that achieves the lower bound of Lemma 2. It essentially places the nodes of the SCG in levels such that the nodes in the longest path lie in consecutive levels, and all arcs that begin at a particular level end at some higher-numbered level. Then all nodes (time frames) at the same level can be compacted into the same shift step in the schedule.

algorithm schedule $(G = (V, A)$: schedule constraint graph): Returns S, a schedule of minimum length satisfying G.

```
{
  l ← 0;
  While |V| > 0 do:
  {
    l ← l + 1;
    R_l ← nodes in V having no incoming arcs;
    /* R_l consists of consecutively numbered time frames starting with the
         lowest-numbered time frame in V; see proof of correctness */
    Remove the nodes in R_l, along with adjacent arcs, from G;
  }
  /* Final value of l represents number of steps in schedule */
  Return schedule S = (m_1, m_2, ..., m_l) where
    m_i = lowest-numbered time frame in R_i, 1 ≤ i ≤ l.
}                                                                    □
```

The sets R_l determined by this algorithm for the SCG of Figure 3 are $\{0\}$, $\{1\}$ and $\{2, 3, 4\}$, hence the schedule is (0, 1, 2). The computation involved in computing R_l in

each iteration is of order $O(d^2)$ assuming that an adjacency matrix is used to represent the SCG. Since the number of iterations is bounded by d, the overall complexity is $O(d^3)$. Below we demonstrate that the algorithm **schedule** works correctly in all cases.

Proof of Correctness We need to prove two assertions: first, that S is a schedule satisfying the compaction principle; second, that the resulting schedule is optimal.

Consider the first iteration of the 'while' loop. By construction, the lowest-numbered node in G (i.e., 0 for $l = 1$) cannot have incoming arcs, hence it must be included in R_l. Let this node be r_1. Let the highest-numbered node in R_l be r_2. We will now show that all nodes r such that $r_1 < r < r_2$ must be in R_l. Assume that there is in fact a node v, $r_1 < v < r_2$, that is not in R_l. Then there must be a node $u < v$ with an arc (u, v) in G. Then by construction of G, u must have outgoing arcs to all nodes $v' \geq v$. Hence there must be an arc (u, r_2) in G, which is a contradiction since r_2 is in R_l. Thus R_l represents a group of consecutively numbered time frames starting with the lowest-numbered one currently in V.

After the nodes in R_l are removed from G, the resulting graph is similar in form to G since only a consecutive set of lowest-numbered nodes has been removed. Hence the arguments above can be applied recursively to the resulting graph for the subsequent iterations. Thus every set R_l consists of consecutive time frames. Note also that for any arc in G, the two adjacent nodes cannot be in the same R_l. From Lemma 1 it follows that S is a valid schedule satisfying the compaction principle.

Since all nodes with no incoming arcs are removed in each iteration of the 'while' loop, the length of the longest path must decrease by 1 each time. Thus the number of iterations is equal to the number of nodes in the longest path. According to Lemma 2, this is in fact a lower bound on the number of steps in any valid schedule. Hence the schedule S returned by the algorithm is optimal. □

In this section we have shown how to determine an optimally compacted schedule based on the structure of the acyclic circuit under test. This schedule can be utilized in two ways. First, it can be used in conjunction with a traditional sequential ATPG program to compact each test sequence produced before random data is used to fill in unspecified input values. In the sequences produced by ATPG, the time frame at which the fault is detected may be treated as frame d, and the sequence can be compacted according to the schedule. Some test sequences produced by ATPG may propagate a fault to more than one output. Such sequences should be preprocessed by selecting any one of those outputs and then forcing any input value to don't-cares if the value in that time frame does not influence the selected output at the time of detection.

The second and more efficient way to utilize the schedule is to use it as a guide for test generation itself. In the following section we will show how to construct a restricted test generation model to replace the traditional iterative array used in sequential ATPG.

Test generation on this model will directly result in compacted test sequences for the desired schedule.

3 Test Generation Model

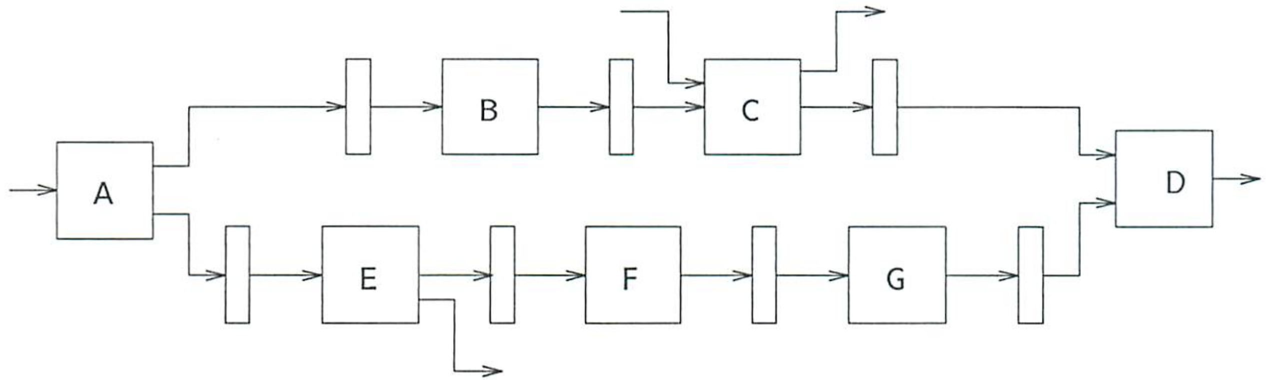
We now turn to the problem of test pattern generation for an acyclic structure. Given an optimized schedule with the smallest number of shift steps, we shall use it to influence the test generation process and simplify it if possible.

In test generation for general cyclic circuits, sequential ATPG programs typically construct an iterative array containing repeated copies of the circuit in order to represent the behavior of the circuit in different time frames [6]. With cyclic circuits the size of the iterative array required to detect an arbitrary fault may grow exponentially with the number of FFs in the circuit. However, in an acyclic circuit every irredundant fault must be detectable within $d+1$ clock cycles, where d is the depth of the structure, and the complexity of the test generation process is comparable to that for combinational circuits [1]. In fact a simple combinational **test generation model** (TGM) can be derived from the circuit structure, and any combinational ATPG program capable of dealing with multiple faults can be used. Not only is a sequential ATPG program unnecessary, this also avoids the execution overhead in maintaining iterative arrays of various lengths.

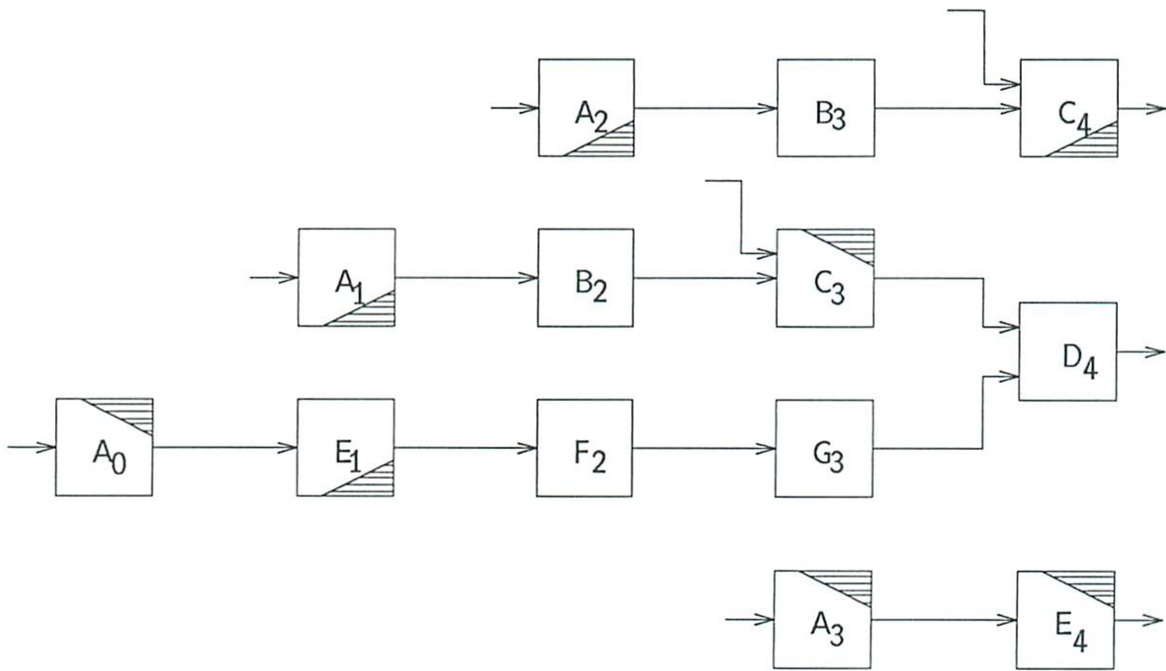
The concept of combinational TGMs is illustrated in Figure 4. Figure 4(a) shows a simplified version of the structure in Figure 2. It has three outputs at C, D and E respectively. We assume that any fault under test will be detected at one of the outputs at time frame 4. Figure 4(b) shows both outputs placed in time frame 4, and the portion of the circuit that feeds each output is laid out in a levelized fashion corresponding to the time frames. Blocks that are required to be in more than one time frame are replicated; thus for example A occurs at several different time frames in the expanded structure since the output values may depend on the behavior of A in various time frames.

Each copy of a repeated block has been pruned to remove any logic that will not be used for test generation; this is indicated by shaded regions but will not be explicitly shown from now on. The subscripts on A_0, E_1, etc. refer to the time frames in which the corresponding instances of the logic blocks exist; the highest subscript is clearly the depth $d = 4$. All registers in the expanded structure have been replaced by wires and the resulting TGM is combinational. This is the general form of the TGM before any compaction; we shall refer to it as the **basic TGM** and it represents the schedule $(0, 1, 2, \ldots, d)$.

In order to generate a test for a fault in the original sequential circuit, the fault must first be mapped to the set of corresponding fault instances in the combinational TGM. (This is analogous to the modeling of faults in iterative arrays.) Ordinary combinational ATPG can now be carried out on the TGM. The test pattern obtained can be transformed into a test sequence for the sequential circuit using the following rule: all input patterns



(a) Acyclic structure



(b) Basic TGM

Figure 4: Basic test generation model.

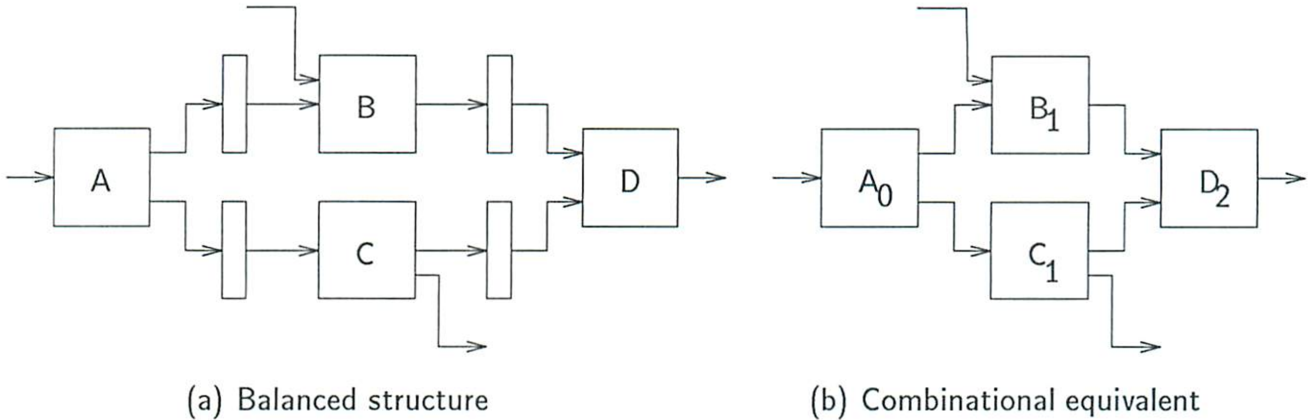


Figure 5: Balanced structure and its test generation model.

at logic blocks with subscript i must be used as the ith pattern in the test sequence. This of course applies if the schedule $(0, 1, 2, \ldots, d)$ is used with no compaction.

Condensing the Test Generation Model

When the test schedule is compacted as described in Section 2, not only is the test time per test sequence minimized, but we can also take advantage of the compacted schedule to condense the TGM. For the special case of *balanced structures* [2] such as the one shown in Figure 5(a) it has been shown that a single pattern is always sufficient for detecting any fault, i.e., the optimal schedule is always (0). The TGM for this class of structures is simply the combinational equivalent of the structure formed by replacing all FFs by wires as shown in Figure 5(b). Thus each logic block appears only once in the TGM, and only single faults need to be considered during combinational ATPG. However, this is not the case with general unbalanced structures. Given the schedule to be used, we shall show how a maximally condensed TGM can be derived. We shall prove that provided the schedule satisfies the compaction principle, the condensed TGM is sufficient for complete test pattern generation.

In condensing the TGM we begin with the basic TGM for the schedule $(0, 1, 2, \ldots, d)$ and modify it based on the schedule provided. We essentially utilize the fact that each input pattern applied to the kernel at a shift step in the schedule is also applied during the subsequent hold steps. The condensation process can be carried out by repeating the following two steps which are illustrated in Figure 6 for the circuit of Figure 4. The term **schedule interval** refers to a shift step and its subsequent hold steps.

Step 1: Repeat the following for each schedule interval. If any input signal occurs at more than one time frame in the same interval, connect the different copies together by fanning out the earliest copy of the signal (i.e., the one

occurring in the lowest-numbered time frame) to the other copies so that only one copy of the input signal remains within the interval.

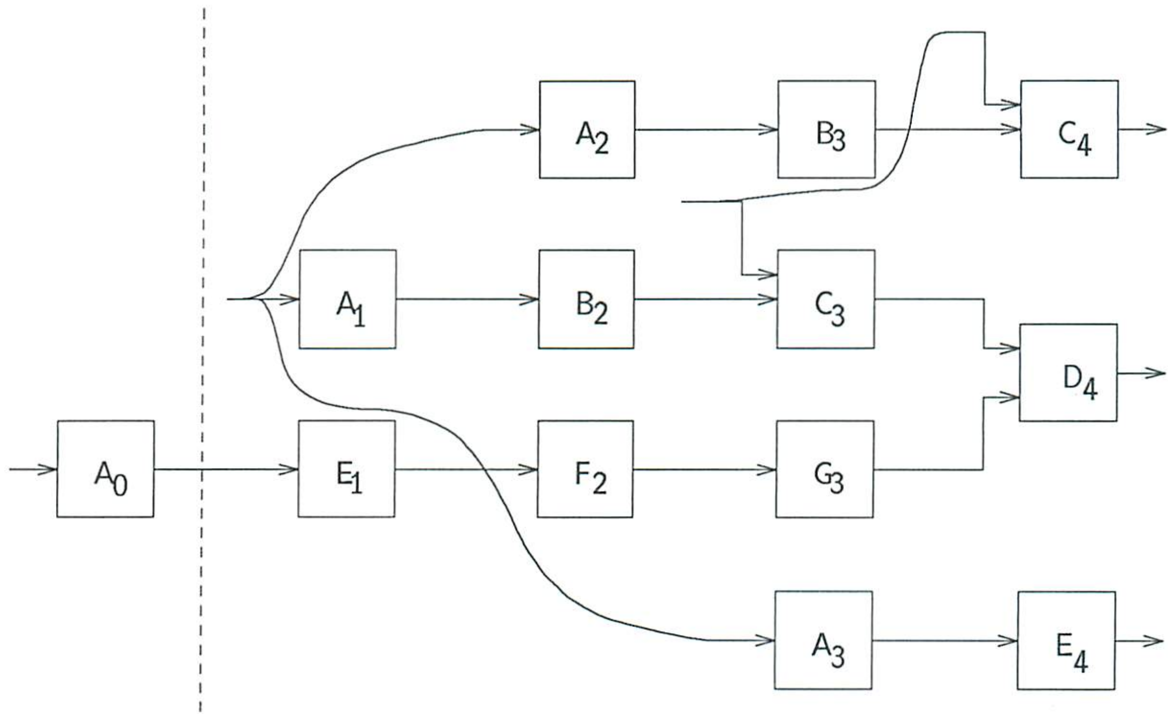
For example, consider the circuit of Figure 4(a) for which $(0, 1)$ is an optimal schedule. Figure 4(b) shows the basic TGM which is to be condensed. The schedule $(0, 1)$ has only one interval containing more than one step, namely the one consisting of time frames 1, 2, 3 and 4. In this interval the input feeding logic block A occurs three times, hence these inputs are connected together as shown in Figure 6(a). Similarly the inputs to C in time frames 3 and 4 are connected together. Note that A_1, A_2 and A_3 now receive identical inputs and in fact they represent exactly the same behavior extended over three clock cycles. The following operation will replace them with one merged copy in A_1.

Step 2: Repeat the following operation until no further changes can be made in the TGM. Let β be a logic block in the circuit and let $\beta_{i_1}, \beta_{i_2}, \ldots, \beta_{i_n}$ be different copies of β such that $i_1 < i_2 \cdots < i_n$ and every signal feeding an input of β_{i_1} also fans out to the corresponding input of each of $\beta_{i_2}, \ldots, \beta_{i_n}$. Then remove $\beta_{i_2}, \ldots, \beta_{i_n}$ from the TGM and fan out each output of β_{i_1} to all the signals originally fed by the corresponding outputs of $\beta_{i_2}, \ldots, \beta_{i_n}$.

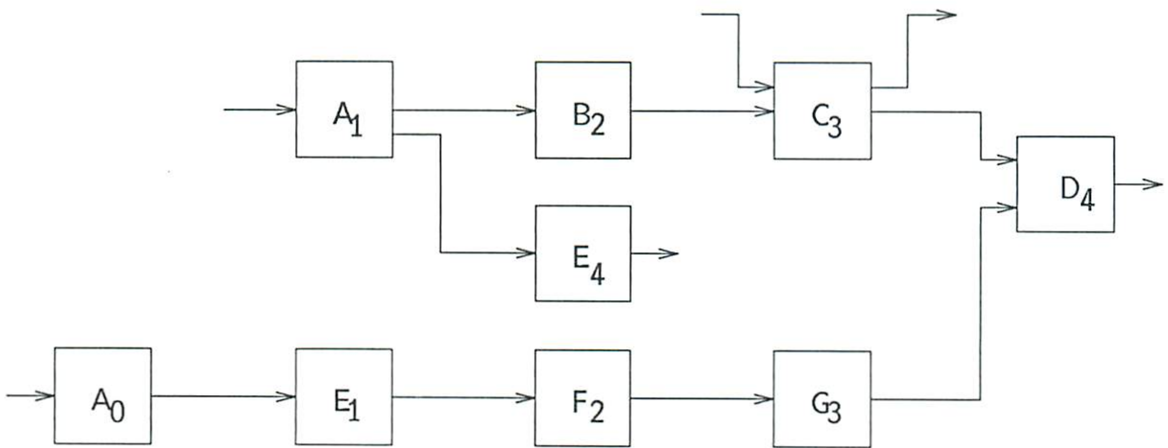
At first this step can be applied to remove A_2 and A_3 and fan out the output of A_1 to B_3 and E_4 as well. Because block A must have exactly the same behavior in time frames 1, 2 and 3, we have simply combined the three copies for the purpose of ATPG, and fanned out the outputs appropriately. Note that this does not alter the execution of the test in any way; it only incorporates some information present in the test schedule into the test generation process, reducing the amount of analysis carried out during ATPG. In the resulting structure both B_2 and B_3 are fed by A_1, hence they can be merged into B_2. Finally in a similar way C_4 can be merged into C_3. No further merging is possible and the final condensed TGM for the schedule (0, 1) is shown in Figure 6(b).

The condensation steps can be applied to any basic TGM, given a schedule, to yield a condensed TGM. Figure 5(a) shows an acyclic structure that has the balanced property, for which the optimal schedule is (0); Figure 5(b) shows the condensed TGM resulting from the procedure described above. As expected the TGM is a simple combinational equivalent of the original structure.

The computation complexity of the condensation steps depends on the implementation and on the level of description of the circuit. The computation in Step 2 can be minimized by forming maximally connected clusters of combinational logic blocks, and carrying out the circuit manipulations on this form of the circuit. Later the lower-level logic descriptions can be filled in for each high-level block, pruned where necessary as described earlier.



(a) Condensation step 1



(b) Condensation step 2

Figure 6: Condensation of test generation model.

Test Pattern Generation

Given an acyclic structure C, a schedule $S = (\phi_1, \phi_2, \ldots, \phi_n)$, and a condensed TGM T_C for C based on the schedule S, the following procedure can be used to generate a test for an arbitrary fault f in C. Let f_C be the corresponding fault (possibly a multiple fault since some logic may be replicated) in T_C. Let us assume that f_C is detectable in T_C; then ordinary combinational ATPG can be used to derive a test pattern for f_C in T_C. Note that due to the nature of the condensation process, any given input signal to C can occur in T_C only at time frames ϕ_1, ϕ_2, etc. in S. For each ϕ_i, let p_i be an input pattern containing the values of all inputs that occur in time frame ϕ_i in T_C, and containing don't-care values for all other inputs. Then the sequence of patterns $(p_1, p_2, \ldots, p_n)$, if applied according to the schedule S, will detect f in C.

In order to justify the use of the condensed TGM we need to validate the assumption that if f is detectable in C then f_C is detectable in T_C. This is done by the following theorem.

Theorem 1 *Given a fault f detectable in an acyclic circuit C, and given a schedule S that satisfies the compaction principle, the corresponding fault f_C (possibly multiple) is detectable in the condensed TGM T_C.*

Proof Let T_B be the basic TGM of C and let f_B be the fault (possibly multiple) corresponding to f in T_B. Since f is detectable in C, f_B must be detectable in T_B using some test pattern τ_B. Suppose the error is propagated to output Ω_d in T_B. Then the cone of logic feeding Ω_d in T_B has certain input values in τ_B that constitute a sufficient test pattern τ'_B for f_B, irrespective of the other input values. Note that in τ'_B, no input signal takes on more than one distinct value within the same schedule interval, otherwise the compaction principle would be violated by the schedule S. Hence for every input signal in the condensed TGM T_C there is a unique value that can be applied to it in every schedule interval in order to simulate the behavior of T_B. Let the input pattern formed by these values be τ_C; note that it is a condensed form of τ'_B. Since τ'_B detects f_B, it must cause different output values at Ω_d in the good and faulty versions of T_B. Hence τ_C must cause different output values at Ω_d in the good and vaulty versions of T_C. Thus f_C is detectable in T_C. □

The above theorem proves that for any detectable fault in C, a test sequence that follows the schedule S can be generated using combinational ATPG on T_C. This leads to the following corollary.

Corollary *Given an acyclic circuit C and a schedule S that satisfies the compaction principle, a complete test pattern set for the condensed TGM T_C results in a complete test sequence set for C using the schedule S.* □

We have thus shown that the condensed TGM derived in this section is a sufficient and complete model for test generation. The size of the TGM is lower than the corresponding iterative array used by traditional sequential ATPG programs. By condensing the TGM for the given schedule to be used in applying the test, some redundant computations during the test generation process are eliminated. If the schedule is minimal, the model guarantees that an arbitrary fault can be detected using the smallest possible number of shift steps.

Note that in this paper we have focussed on the problem of generating minimal test sequences for given faults. Test generation programs typically assign random values to unspecified values in the input patterns, and then run fault simulation to drop additional faults. The TGM derived here is not suited to fault simulation, except when the schedule is (0), since it does not consider errors propagated to the scan path in intermediate shift steps. Note however that sequential fault simulation is not as hard a problem as test generation and any sequential fault simulation tool can be used in conjunction with ATPG using the condensed TGM derived here.

4 Conclusion

In this paper we have studied the problem of testing acyclic structures in partial scan designs. We have presented a new approach to test sequence compaction in which the objective function is the number of distinct patterns to be shifted into the scan path per test sequence. In our approach, each test sequence is compacted into the smallest number of patterns needed to be shifted into the scan path. This leads to the lowest test time to detect an arbitrary fault. An algorithm for determining the optimal schedule, based on the structure of the circuit, was presented.

We have also presented a specialized test generation model (TGM) for acyclic structures. Like the iterative array model, this model reduces the ATPG problem to that of combinational ATPG with multiple faults. A special feature of this model is that it uses the optimal schedule determined separately in order to derive a condensed TGM. This leads to fewer redundant computations during ATPG. However, a separate sequential fault simulator is required. The optimal scheduling algorithm and the test generation model can be used to significantly reduce the testing costs in partial scan designs, especially for signal processing and pipelined circuits.

Acknowledgement

The authors wish to thank Kuen-Jong Lee and Rajiv Gupta for helpful suggestions.

References

[1] A. Miczo. *Digital Logic Testing and Simulation.* Harper & Row, New York, 1986.

[2] Rajesh Gupta, Rajiv Gupta, and M. A. Breuer. BALLAST: a methodology for partial scan design. In *Proceedings, Fault-Tolerant Computing Symposium (FTCS-19)*, pages 118–125, June 1989.

[3] K.-T. Cheng and V. D. Agrawal. An economical scan design for sequential logic test generation. In *Proceedings, Fault-Tolerant Computing Symposium (FTCS-19)*, pages 28–35, June 1989.

[4] Rajesh Gupta, Rajiv Gupta, and M. A. Breuer. An efficient implementation of the BALLAST partial scan architecture. In *Proceedings, International Conference on Very Large Scale Integration (VLSI-89)*, pages 133–142, Munich (West Germany), August 1989.

[5] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

[6] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, Rockville, Md., 1976.