NBSIR 79-1927

Recovery from Soft Errors in Triplicated Computer Systems Operating in Lock-Step

A. L. Koenig
A. W. Holt

System Components Division
Center for Computer Systems Engineering
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

June 30, 1979

Issued November 1979

Sponsored by
Defense Nuclear Agency
Washington, D.C. 20305

NBSIR 79-1927

RECOVERY FROM SOFT ERRORS IN TRIPLICATED COMPUTER SYSTEMS OPERATING IN LOCK-STEP

A. L. Koenig
A. W. Holt

System Components Division
Center for Computer Systems Engineering
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

June 30, 1979

Issued November 1979

Sponsored by
Defense Nuclear Agency
Washington, D.C. 20305

U.S. DEPARTMENT OF COMMERCE

Luther H. Hodges, Jr., ***Under Secretary***

Jordan J. Baruch, ***Assistant Secretary for Science and Technology***

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, ***Director***

RECOVERY FROM SOFT ERRORS IN TRIPLICATED COMPUTER SYSTEMS OPERATING IN LOCK-STEP

ABSTRACT

A Triply Modular Redundant (TMR) computer system operating in clocked lock-step is being investigated for an application requiring a Mean Time Between Failure of five years. No mechanical memories are used; this allows comparison of the outputs of the three computers to be made each clock period. The most novel contribution is the method of recovery from soft errors, such as those produced by lightning strokes or alpha particles. Data are provided on the uptime history of an experimental system, which uses three commercial microcomputers.

Key Words: Fault tolerant computer; soft errors; triply modular redundant; TMR.

INTRODUCTION

Where exceptionally high reliability is essential, an acceptable level of performance can often be achieved through the use of redundant structures or modules. Two common forms of modular redundancy are fault masking and spare switching. With fault masking, all channels are active all the time; majority logic is used to "mask" a failing channel, keeping the total system output correct. Spare switching, as the name implies, has spare channels in a standby or passive state ready to be switched in when an active channel fails. Digital systems have been implemented with these techniques with varying degrees of success. Fault masking systems are often difficult to maintain because the process that makes them fault tolerant also makes fault detection and isolation complicated. Using spare computers in standby mode presents the complex problem of making the spare processor reconstruct the current working status after the active computer has failed.

A third category of modular redundancy has been proposed, called Sift-Out Redundancy [1/]. With this technique there are no standby channels; all are active at starting time. If any one fails, however, the structure reconfigures itself so that the contribution of the failed channel to system output is eliminated. At least three modules are required to implement this kind of structure. If reduced to two modules and one fails, the system is unable to detect which module is in error. If three or more modules are used in the implementation, any errant one is "sifted out" automatically; when a failure occurs in another module, the process repeats itself until the system is sifted down to two. It must be assumed that no more than one module will fail at a given instant of time. The sifted out modules may be restored to operation at any convenient time by a command from the supervisory element of the system.

1/ P. T. de Sousa and F. P. Mathur, "Sift-Out Modular Redundancy," IEEE Transaction on Computers, Vol. C-27, pp. 624-627, July 1978

The experiment described here is an implementation of Sift-Out Redundancy applied to the data bus of a microcomputer. One unique aspect of the system is the requirement for synchronous operation of multiple computers in order to use the Sift-Out technique; another is the self-restoring action which takes place when an error has been detected. The latter takes advantage of the synchronous operation in its implementation.

In the triply modular redundant (TMR system described in this report,the modules are single-board microcomputers. Each module consists of a microprocessor, read/write random access memory (RAM), programmable read-only memory (PROM), buffers for the address bus and data bus, and various timing and control circuits. All three computers run identical programs which reside in PROM. The RAM is used to store data. The programs run "lockstep"; synchronism is achieved by driving all three processors from a common clock. The input/output (I/O) data bus of each computer is the logical OR of the individual bus outputs. The busses are checked for identical content once during each clock period. If one bus is not identical to the other two, it is logically disconnected from the combined output immediately and latched out. Outputs from the error detector identify which bus has failed. This information is used by the computers to take corrective action: restore erroneous data in RAM; resynchronize the programs; reset the error detector, reconnecting the failed bus to the system output. Thus the system is self-restoring for "soft" failures, i.e., transient faults on one bus. The system can be programmed to give an appropriate alarm for more serious or very repetitive faults.

Because of limitations on time and resources, this experiment is designed to demonstrate only the operation of the Sift-Out comparator hardware and the feasibility of automatic recovery through memory-to-memory transfer. Admittedly, a fully operational computer system is susceptible to many types of faults other than soft errors on the data bus. It is not within the scope of this experiment to remedy or even identify all failure modes, but rather to offer a means of improving reliability for one class of errors. Recognizing the limitations of the experiment, no attempt has been made to measure the improvement in Mean Time Between Failures resulting from TMR, though the uptime history of the system presents an encouraging picture.

HARDWARE DESCRIPTION

System Configuration

Figure 1 shows the configuration of the hardware for the TMR experiment. One line from the data bus of each computer is brought to the comparator and to the collector. The collector output is the "correct" system output, i.e., the combined output from which erroneous data has been sifted out. One one data bus line, the least significant bit (LSB), is used in this experiment in order to minimize the construction of special hardware; in a fully implemented system all data lines, and possibly address bus lines, would be similarly connected. However, comparison of one data line is sufficient to demonstrate the principles of the TMR

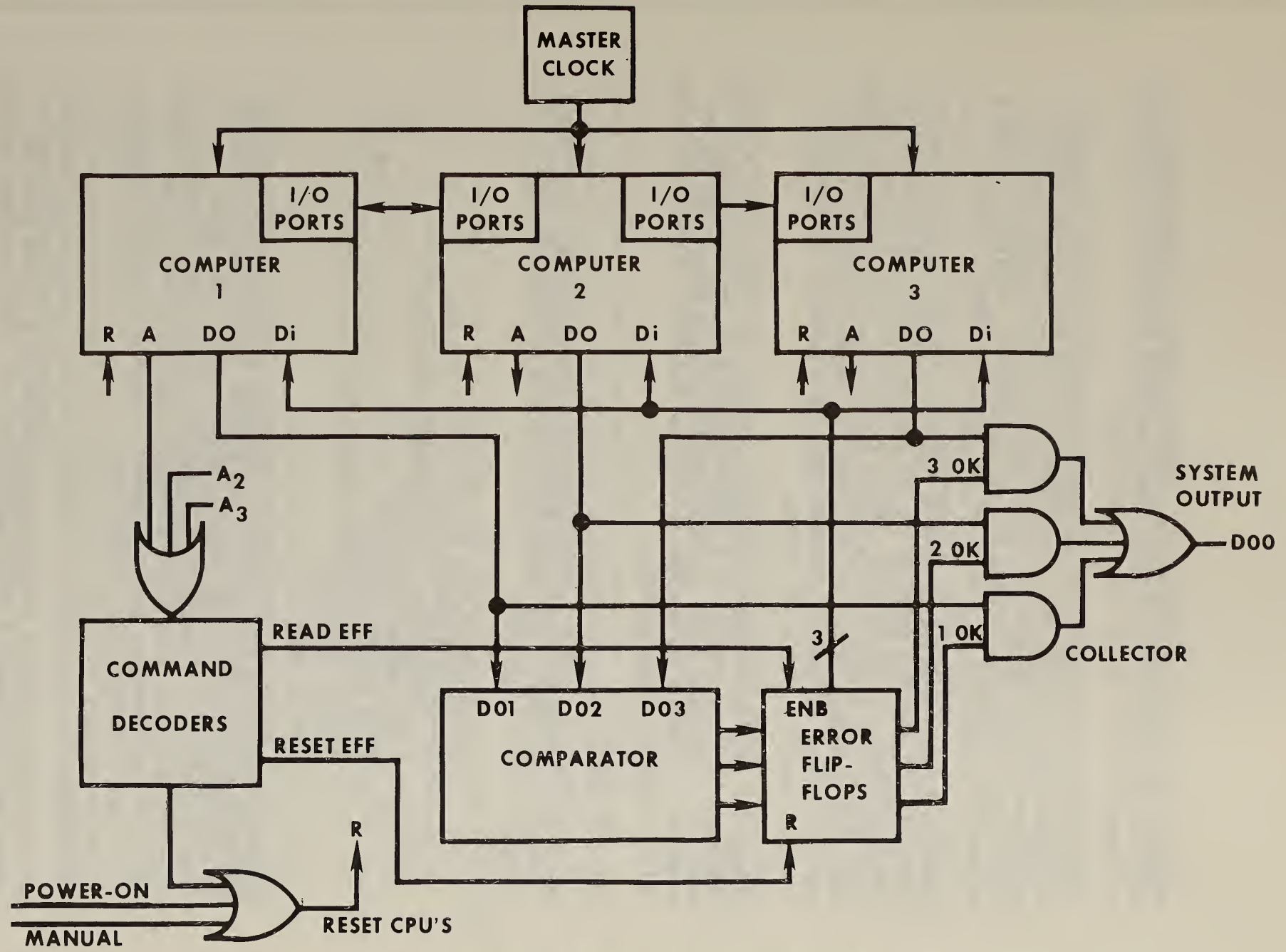

FIGURE 1. TMR CONFIGURATION

system. The choice of the LSB for comparison makes the comparator particularly sensitive to timing anomalies since this bit is most likely to change from one clock cycle to the next.

Other interconnections of the hardware are needed to effect automatic resynchronization after an error has been detected. Command decoders are connected to each address bus in order to assert reset lines and enable input gates for reading the error detector. Parallel I/O ports connect the computers to each other for memory-to-memory transfer.

Computer 1 has additional peripheral devices attached which are used primarily for software development and are not directly involved in the TMR experiment; A CRT display, keyboard, cassette tape recorder, and a 2400 baud serial I/O port. The serial port is used to down-load program code from computers in the NBS Experimental Computer Facility (ECF). The CRT display is used in the experiment as a means of verifying correct operation of the TMR system.

Computer 1 and its peripheral interface boards are installed on a single backplane which is mounted on a chassis along with the system power supplies; computers 2 and 3, their parallel output ports, and the comparator hardware are mounted on a remote chassis. There is about one meter of cable between chassis.

Microprocessors

The microprocessors used in this experiment are type 6800. The 6800 is a single chip, 8-bit N-MOS processor housed in a 40-pin package. Its important features include:

Bidirectional data bus (8 bits)
Sixteen-bit address bus
72 instructions (1 to 3 bytes/instruction)
1 MHz maximum clock rate
Two 8-bit accumulators
Three 16-bit registers (stack pointer, program counter, index register)
Memory-mapped I/O

Timing is controlled by a 2-phase clock which is external to the chip. During phase one, the contents of the program counter are transferred to the address bus; on the negative transition of this phase, the program counter is incremented. During phase two, data are put on the data bus, and, when this phase goes low, these data are latched into either the processor or external memory. Direction of data flow is determined by a read/write control line.

CPU Boards

Each CPU board consists of the microprocessor chip, 2,048 bytes of static RAM, 256 bytes of PROM, buffers for the address and data busses, and several timing and control circuits.

The 16-bit address bus is buffered with tri-state drivers. Since the processor uses memory mapped I/O, the same lines are used for memory address and I/O port addresses.

The bidirectional data bus is split into four parts by the buffering process; memory data out, memory data in, I/O data out, and I/O data in. Memory and I/O data out are not functionally different; they are simply connected to two different sets of tri-state buffers. The input buffers, however, are controlled differently. The memory inputs are gated with a clocked Read signal while I/O inputs are selected by the Read signal and a signal decoded from the upper half of the address bus. Thus one "page" of memory (hexadecimal addresses FE00 through FEFF) is reserved for I/O port addresses. Both the data out and address bus buffers can be disabled -- that is, put in the high impedance state -- when a direct memory access (DMA) transfer is requested. This allows the requesting device to have full control of the memory for the transfer.

The on-board memory consists of sixteen 1024 x 1 bit static MOS memory chips and one 256 byte ultraviolet erasable PROM. The address range of the RAM memory can be changed by jumpers on the board; the range of the PROM is fixed at hexidecimal addresses FF00 through FFFF.

As manufactured, the board contains a 1 MHz crystal controlled oscillator with fixed delay and drive circuits to provide two non-overlapping phases of clock for the microprocessor chip. For this experiment, the oscillator was disabled and a master oscillator provided on a separate board to drive all three CPU boards. The delay and driver circuits on each board were retained.

Other on-board features provided but not used in this experiment include interrupt and DMA Request control logic and Run/Step control. The latter is jumpered for Run condition only. Step forces a halt after each instruction cycle. This is of little use with this particular microprocessor because all of its address and data lines go into the high impedance state during a halt. A display panel with latches would be required to capture the status of the processor between steps.

On the original boards, a Power-On Reset function was provided by having a delay one-shot circuit attached to the processor Reset input, This one-shot could also be triggered by a remote contact closure for Manual Reset. Since the reset delay in the TMR system must be identical for all three processors, this feature was disabled on all boards and a Master Reset circuit built on a separate board.

Parallel I/O Ports

Eight-bit parallel ports are used in this system as a means of transferring RAM contents from one computer to another and to connect peripheral devices to computer 1.

As previously noted, data from the processors are buffered by tri-state drivers which are always enabled except when a DMA transfer is in progress. When data on the bus is intended for output to a port (addresses FE00-FEFF), an I/O Output Strobe pulse is generated on the CPU board. Thus, to capture this data, it is only necessary to decode the eight least significant bits of address and strobe the data into latches. The data remains in the latches until another data byte is sent to the port.

Data input is handled in a similar manner. The address decoder used for output may be used for input. When the CPU executes the instruction, LOAD FROM ADDRESS FEnn, the I/O Input Strobe line is asserted. The AND of the strobe and the decoder output is used to gate the desired data onto the processor input bus. Open collector NAND gates are used for this function.

Note that a single address may be used to identify one input and one output port. Input and Output Strobes, generated by LOAD and STORE, respectively, define which function the port is to execute.

Clocking

Lock-step operation of the three TMR computers is achieved by having a common oscillator for all the computers. The circuit used is shown in figure 2.

The 10,000 ohm resistors across the inverters make them operate as linear amplifiers. The two amplifiers in series provide the 360 degree phase shift required for oscillation. The crystal frequency is 1 MHz. The third inverter provides enough drive for the three CPU boards. The output is a square wave.

As previously stated, the circuits necessary to produce the two non-overlapping clock phases to the microprocessors are located on each CPU board.

Reset Circuits

Initialization of the hardware and transfer of program control to a selected starting address is accomplished by assertion of the CPU Reset input line. This action takes place automatically when the system power is turned on. The same function can be activated at any time by pushbutton or on command from the software.

The CPU Reset circuit is shown in figure 3. The basic element is a delay one-shot with a pulse duration long enough to allow for power supply settling plus the required initialization time of the processor. The latter is specified by the manufacturer to be at least eight clock periods.

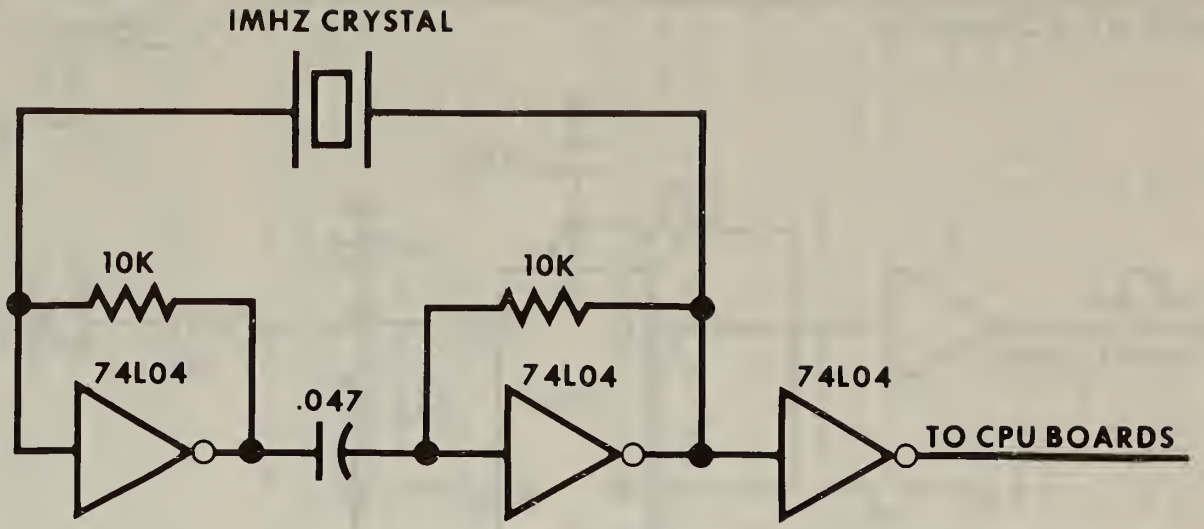

FIGURE 2. MASTER CLOCK

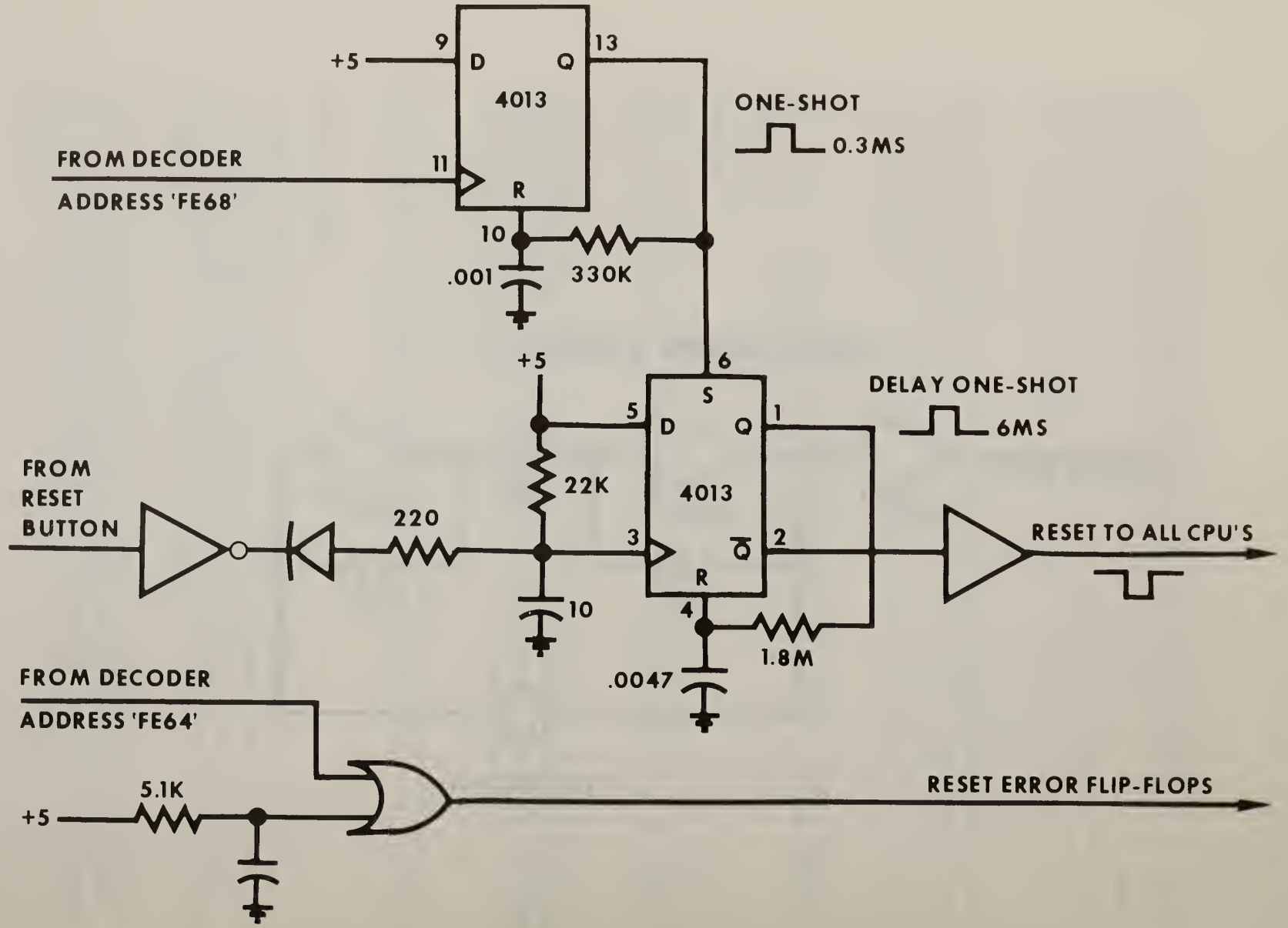

FIGURE 3. RESET CIRCUITS

The delay one-shot is constructed from a CMOS flip-flop and external timing components as shown. For the Power-On condition, triggering does not occur until the 10 μf capacitor has charged to at least half the supply voltage. The flip-flop is then in the set state until it is reset through the RC network connected to the dc reset (pin 4). Manual Reset by pushbutton involves the same components. The 10 μf capacitor is discharged when the button is depressed; release of the button starts the triggering sequence just described. For activation of Reset by software command, the path is slightly different. The command appears as an address (from any of the three computers) which must be decoded. The decoded address triggers the 0.3 millisecond one-shot which in turn triggers the delay one-shot through its dc set input (pin 6). With the component values shown, the pulse width of the delay one-shot is 6 milliseconds. For Power-On and Manual Reset, additional delay is provided by the charge time of the 10 μf capacitor.

Reset of the TMR error flip-flops is done via a separate reset line. Another address decoder output is used for command reset. Power turn-on reset delay is provided by an RC network connected to the OR gate as shown in figure 3.

Comparator

The comparator checks data bus lines to make sure they have identical information. If one line is in disagreement with the other two, only the information from the two in agreement is passed to the output. Furthermore, there is a feedback loop in the comparator which holds the "bad" line disabled until it is deliberately reset. A flip-flop in each feedback loop provides an output to identify the errant computer. The comparison of data lines is made at one instant during each clock cycle.

The comparator logic is shown in figure 4. Exclusive-OR (XOR) gates compare to a given data bus line to each of its like neighbors, that is, bit 0 of computer 1 to bit 0 of computer 2, etc. As long as all lines are identical, the XOR outputs are zero, the NOR gate outputs F1, F2, F3 will be zero, and all three data lines are allowed to contribute to the output (final NOR gate). Also, the D inputs to the flip-flops will all be zero so no flip-flops will get set.

Consider the following example to illustrate how errors are detected. Assume that data line D01 disagrees with the other two, lines E12 and E13 will now have a logical 1; output F1 will be a 1 while F2 and F3 will remain at zero. Computer 1 now will contribute a logical zero to the final NOR gate regardless of the state of line D01. Also, since F1 is 1, No. 1 error flip-flop will be set by the trailing edge of phase 2 clock. Note that the flip-flop is in a feedback loop so that F1 cannot return to zero, and thus bit 0 of computer 1 has been "sifted out" of the system. The comparator will return to normal when a Reset pulse is received to reset the flip-flop. The outputs of the error flip-flops are wired to the data input bus via tri-state buffers so that comparator status can be read by the computers. The buffers are enabled

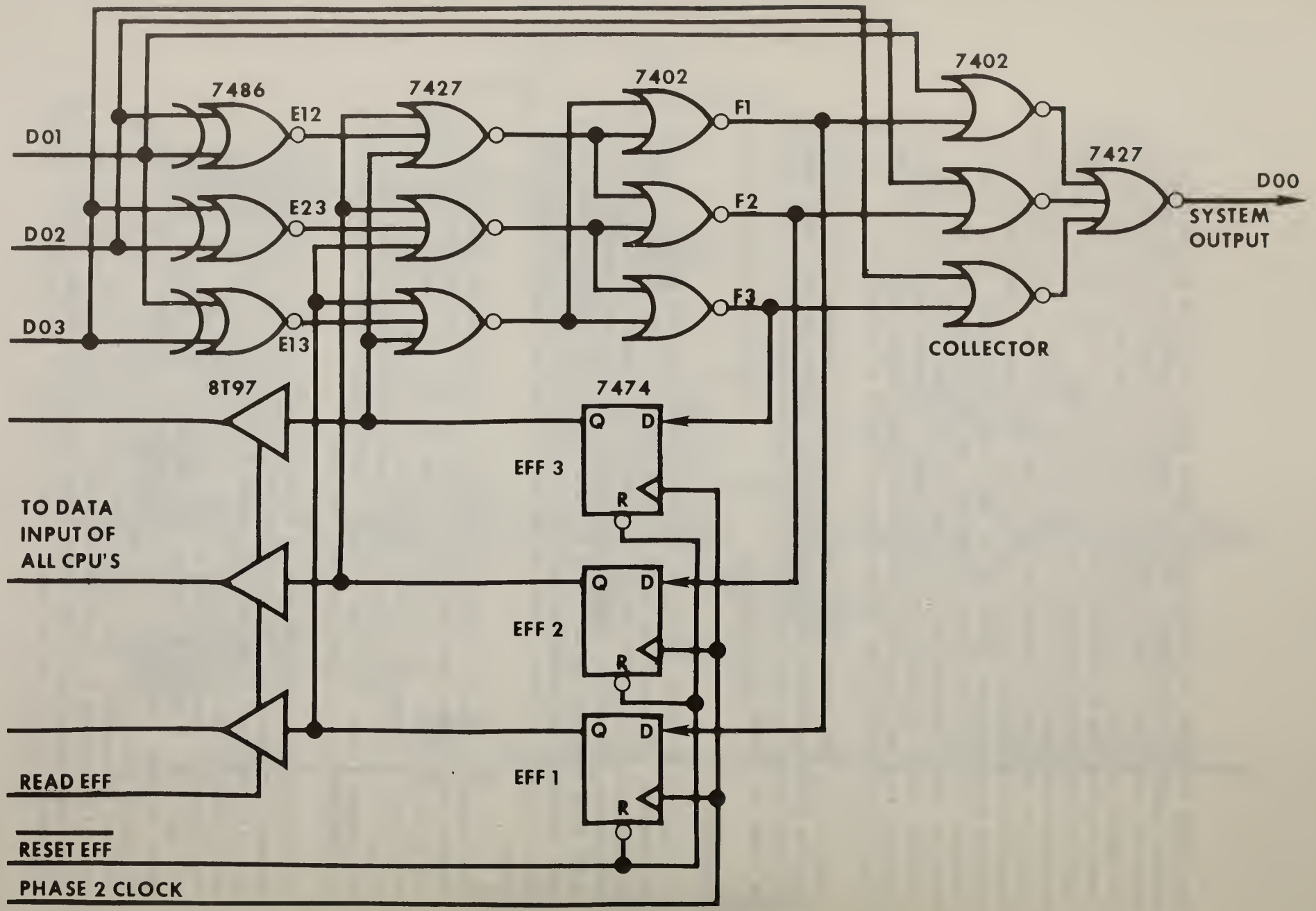

FIGURE 4. COMPARATOR

by an address decoder output.

The above circuit has been implemented only for bit 0 (LSB) of the data bus. A fully implemented system would have a replication of the circuit of figure 4 for every data bus line. Any line not passing the comparison test would result in that line being locked out until the arrival of a Reset pulse; further, the identity of the failed lines is maintained by the error flip-flops whcih are read by each computer prior to issuing the Reset. Thus a comparator for a complete data bus will provide a correct system output as long as at least two of the three lines to each cell of the comparator are correct.

Considering multiple errors, a failure of more than one bit on any given bus can be handled successfully as long as the other two computers maintain correct output during the prescribed interval. However, if the multiple error affects different bits of two or more computers, the situation is more difficult. The individual comparator circuits will still function as described above, locking out the erroneous lines; however, since data of more than one computer is now suspect, this must be considered an alarm, or non-recoverable situation. The subject of error recovery is treated more fully in the Software section of this report.

Peripheral Device Interfaces

Computer 1 has several peripheral device interfaces connected to it, primarily used for program development. Since these do not directly affect the TMR experiment, they will not be treated in detail here. A keyboard, video display, and a cassette tape are interfaced via parallel ports. A serial I/O channel is connected directly to the I/O Data Bus. Table 1 lists these interfaces and their port addresses.

TABLE 1. PERIPHERAL DEVICE INTERFACES

PERIPHERAL DEVICE	PORT ADDRESS
Keyboard	FE00, Input
Video Display	FE00, Output
Cassette Tape	FE01, Input and Output
Serial I/O, Data	FE60, Input and Output
Serial I/O, Status	FE61, Input

SOFTWARE DESCRIPTION

Software for the TMR experiment was designed to demonstrate very simply the synchronous operation of the computers and the technique for automatic recovery from soft errors. The program performs a task (display memory), periodically checks for errors, and, if an error has occurred, updates the suspect memory and resynchronizes the computers. All of this, plus initialization and utility subroutines, occupies less than

256 bytes of ROM. Program flow is shown in figure 5.; a program listing is given in the appendix.

Initialization

There are always certain housekeeping tasks to be performed at the beginning of a program. In this case, all that is necessary is to set the stack pointer to a convenient RAM address and to make sure that interrupts are disabled. Clearing the CRT screen may be considered an initializing task but this is not done immediately; it is important to read the status of the error flip-flops (EFF) to determine if resynchronization is required. Note that the same path is used for initial start and restart after an error, hence the necessity for an EFF check at this time. This is not the periodic error check mentioned above.

If any of the EFF are set, the program branches to the appropriate error service routine. The EFF are tested in numerical order,therefore computer 1 can be said to have top priority. This is not significant unless more than one flip-flop is set, in which case the lower numbered computer will be assumed to be the errant one. However, more than one computer in error is a non-recoverable case so other procedures would have to be invoked in this situation.

If no EFF are found set, the program goes through a "dummy" error routine which makes the computer idle for a length of time equal to a real error routine. This is done only to make all paths in this part of the program of equal length. The significance of this will be discussed later.

Task

For demonstration purposes, the TMR system is programmed to display the contents of a 128-byte block of RAM on a CRT screen. To make it easy to detect an error visually, the bytes of the block are loaded with sequential numbers. Thus the CRT display is a count (in hexadecimal notation) from 0 through 127 (hex 7F). This count is generated only when a pushbutton is depressed by the operator -- that is, the program will not autonomously correct an error by reloading the byte. This point is essential to demonstrate that the system self-corrects only by transfer of data from another computer.

To display RAM contents, a byte is brought to the accumulator with an indexed load instruction, converted to two ASCII characters and transferred to the video interface memory. The block is limited to 128 bytes because of the size of the display.

After 128 bytes have been displayed, the EFF are read. If none are set, the program loops back to display the block again. If an error has occurred, the CPU Reset line is asserted and the computers are all forced to restart.

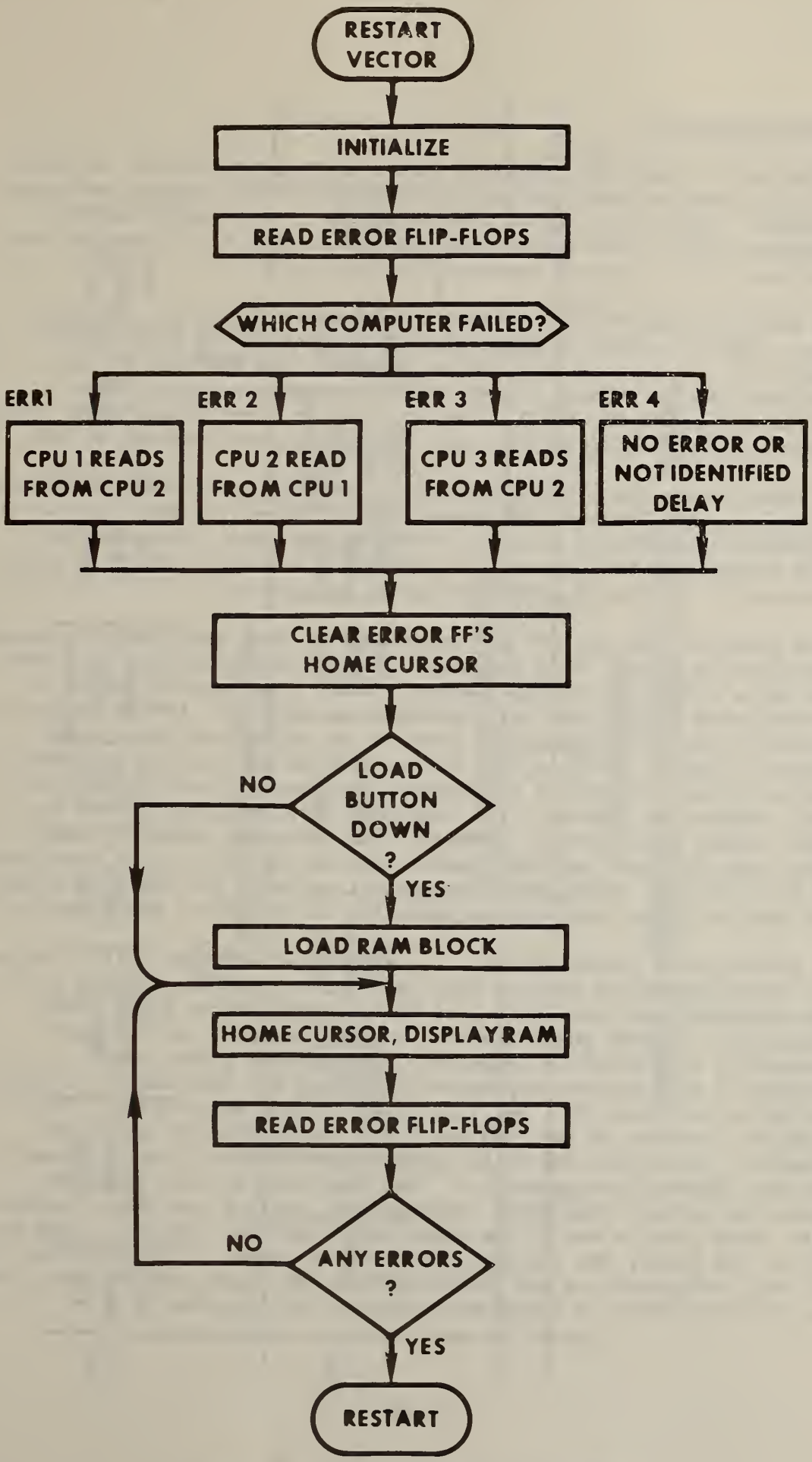

FIGURE 5. TMR PROGRAM FLOW

Resynchronization

Bringing the computers back into full synchronization after an error has been detected requires two distinct operations, Restart and Memory Transfer. The former forces the processors to a common program reference point; the latter gives assurance that all processors have the same data to operate on.

The Restart sequence is initiated by the assertion of the CPU Reset line. While the line is asserted, the processors are in a halt condition; as soon as the line is released, the contents of last two locations in ROM are loaded into the program counter. These locations contain the Restart Vector, that is, the address where program execution must begin, in this case FF00, the lowest address in ROM.

As noted above, the program path after Restart is the same as for the initial start. However, CPU Reset does not clear the EFF; hence, after a Restart one EFF will be set, indicating the error just detected. This information will be used to initiate a memory-to-memory transfer from one of the "good" computers to the errant one.

Data transfer from one CPU RAM to another is done synchronously through parallel I/O ports. This technique takes advantage of the fact that the three computers are all running off the same clock. Timing is accomplished by the order of load/store instructions in the respective programs, thus eliminating the need for an exchange of busy/ready control signals. In this part of the program the code is necessarily non-identical; to effect a transfer, one computer must be reading while another is writing. The comparator will indicate errors for all three during this process. However, the computers actually remain in step, that is, each program segment executes in exactly the same number of clock cycles. This fact is essential not only for the data transfer but also to insure synchronous operation at the end of the transfer.

To examine this process in detail, assume that computer 1 is in error. All three processors receive this information when the EFF are interrogated; all then branch to the routine labeled ERR1. (See figure 5). At this point the code in each computer is different; computer 2 will write to computer 1, so computer 1 must read. Computer 3 will execute a program which uses the same number of clock cycles as the other two but is neither reading nor writing. The relative position of the read and write instructions is shown in figure 6. Note that the Write Strobe from computer 2 occurs ahead of the Read Strobe from computer 1. The computers at this point are executing indexed LOAD/STORE instructions, testing the index register after each pass to see if the upper limit has been reached. When not yet at the limit, the program loops back to repeat. The execution time of each instruction is not identical, but the total time for each loop is the same; hence, a synchronous transfer is achieved.

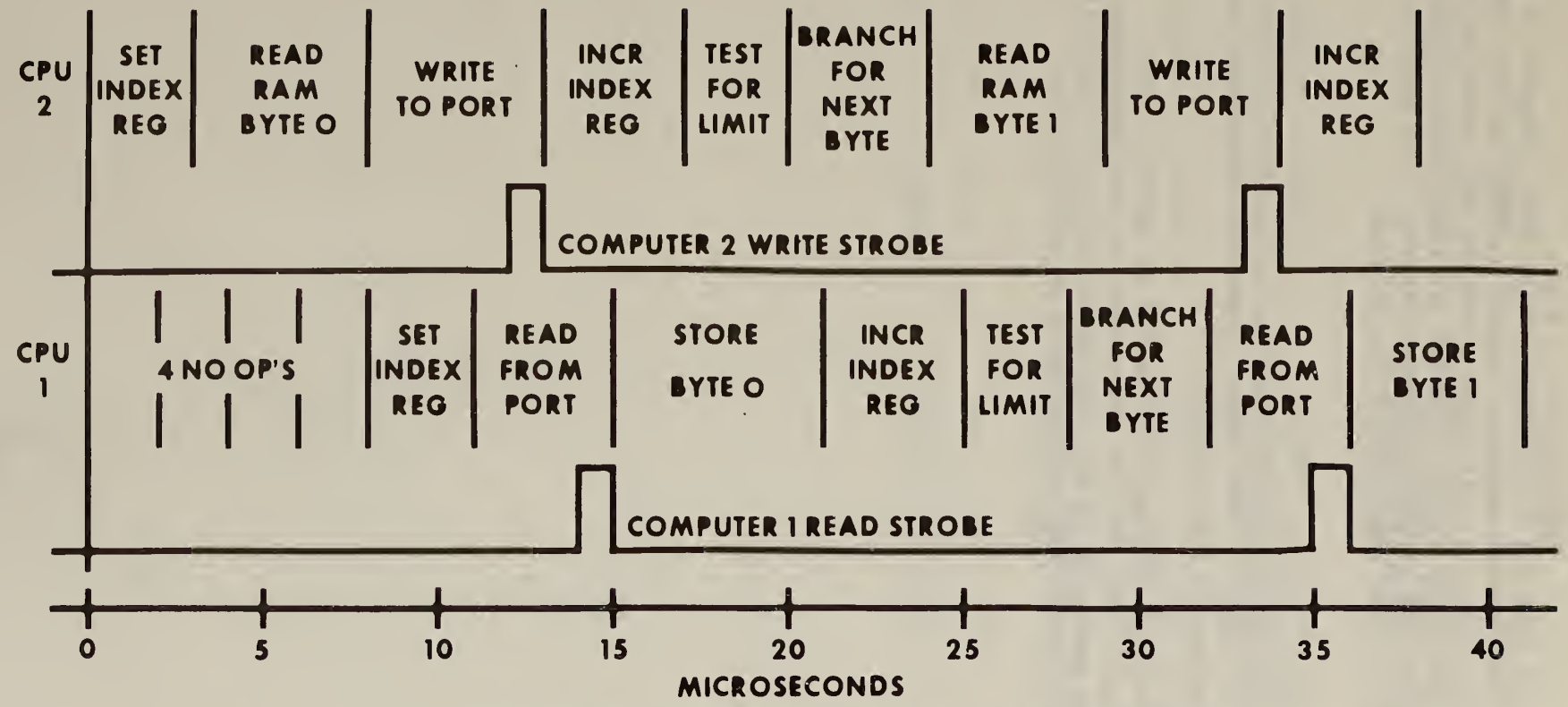

FIGURE 6. MEMORY-TO-MEMORY TRANSFER TIMING

While computer 1 and computer 2 are transferring data, computer 3 is executing instructions chosen solely to expend an equal amount of time. When the transfer is complete, all three computers will arrive at an end point on the same clock cycle. The EFF are then cleared, which is the final function of the resynchronization process. The code for all three is identical for the remainder of the program.

Subroutines

The software for driving the display consists primarily of subroutines. These are listed in table 2. The symbols given are those which appear in the program listings in the appendix.

The 128 bytes of RAM are displayed on the CRT in sixteen lines, eight bytes per line. Each byte is represented by two hexadecimal digits; bytes are separated in the line by two spaces. This fills the entire 1024 positions that are available on this display.

TABLE 2. SUBROUTINES

Name	Symbol	Function
Hex Dump	hdump	Translate a byte from RAM into two hex digits, format for display by adding two spaces after each pair of digits.
Hex Output	hexout	Convert binary half-byte in accumulator to an ASCII hex digit for display on the CRT.
Home Erase	homer	Clear CRT screen by writing 1024 spaces. Return with cursor set at upper left corner of screen.
Print	print	Output character in accumulator to the CRT.
Print Space	prspc	Output space code to CRT.
Spacer	spacer	Output 512 spaces to CRT.

EVALUATION

Testing Procedure

During the course of the experiment, several programs have been run on the computers to test the hardware and to try various software concepts. The first programs were kept very short and simple so that synchronism could be easily verified by monitoring the address bus with an oscilloscope. Once design and wiring errors were eliminated, other programs were written to more fully exercise the system and to test features such as automatic restart and memory-to-memory transfers. During each of these intermediate steps observations were made of error rates and sources of errors. Some errors were found to be caused by slow or weak RAM chips, while others were caused by induced noise in the system wiring. After correcting these problems, the system failures have become so infrequent that none are observable in a time frame of days or weeks. Errors have to be artificially induced in order to demonstrate that the system has the capability of automatic error recovery.

The most extensive reliability testing of the system has been done with the software described in this report. The error recording apparatus consists of a logic analyzer and a counter. The logic analyzer is attached to the address bus of one computer; its trigger word is set to a program location that will be encountered only if an error occurs and the system resynchronizes. The counter input is connected to the trigger pulse output of the analyzer; thus, the counter indicates the number of resynchronization attempts of the system.

For demonstration, errors are induced by shorting a data or address bus line to ground momentarily or by turning on an external noise source. A very effective device for the latter is a high voltage arc generator. An arc of a few centimeters length in the general vicinity of the computers will produce errors readily.

A power line failure will, of course, cause an error indication. The computers will restart automatically after power returns to normal but the RAM contents will be random until the load button is depressed. This is equivalent to reloading the system from non-volatile memory, a procedure which must involve operator intervention. In order to isolate this type of error, the test setup includes a power line monitor which has a strip-chart recorder and a latching lamp circuit to indicate line failures.

Results

As noted above, naturally occurring errors in the TMR system are rare. Over sixty days of running time have been accumulated with the system in its final configuration. During that time only three failures have been observed. One of these affected only the format of the display, a device which is not really a part of the TMR experiment. A software modification has since been made to periodically reset the display cursor, thus eliminating this type of error from the experimental data. Another error resulted in computer 1 being latched into a halt mode, again a situation not covered by the scope of the experiment. This type of error requires operator action, that is, manual reset. The third error was a "soft" error from which the system recovered automatically.

It is important to note that this low error rate has been achieved without extraordinary measures in selection or construction of hardware. As noted above, some weak memory chips were eliminated during the course of development, and system wiring had to be done carefully. Neither of these aspects, however, are above ordinary good commercial practice. The memory chips are inexpensive static RAM, type 2102. No exhaustive testing or preselection was done; there were simply a few chips that did not perform well and they were replaced.

The inter-chassis wiring consists of twisted pair for bus and control signals, miniature coaxial cable for the clock. No special line terminations were used. Bus pull-up resistors, bypass capacitors on the boards, termination of unused inputs, etc., are all according to standard TTL rules.

Software for the TMR experiment is specialized only in the error recovery routines, the remainder is programmed in the conventional manner. No special precautions or procedures are required. Thus, once the recovery programming has been done for a TMR system, the applications. programming can proceed in a normal manner. As the final step, ROMs containing the applications software are produced in triplicate.

Conclusions

The experience with the TMR system indicates that (a) synchronous operation of the three computers is a feasible approach to modular redundancy; (b) off-the-shelf computer hardware can be used in the construction of a TMR system; (c) soft errors which might otherwise crash a system or cause erroneous results can be corrected automatically; and (d) all the above can be achieved without significant increase in software complexity.

APPENDIX

The following is an assembly language listing of the TMR programs. Programs for the three computers are identical axcept for the error routines (err1 through err4); routines for computer 1 are listed as they appear in the actual program while computer 2 and computer 3 error routines are listed separately at the end.

INITIALIZE

```
        org $ff00       ;start at bottom of ROM

        sei             ;disable interrupts
        lds #$7ff       ;load stack pointer
        ldaa $fe6f      ;read error flip-flops
        bita #1         ;error in computer 1?
        bne err1        ;branch to err1
        bita #2         ;in computer 2?
        bne err2        ;branch to err2
        bita #4         ;in computer 3?
        bne err3        ;branch to err3
        bra err4        ;no error or not identified
                        ;branch to err4
```

****

ERROR ROUTINES FOR COMPUTER 1

```
err1:   nop             ;Computer 1 must read from 2's memory
        nop
        nop
        nop
        ldx  #lolim     ;set index
loop1:  ldaa port       ;read from port
        staa 0(x)       ;store in ram
        inx             ;increment index
        cpx #hilim      ;block done?
        bne loop1       ;loop if not
        bra clref       ;if so, continue

                        ;Computer 1 must write to 2's memory
err2:   ldx #lolim      ;set index
loop2:  ldaa 0(x)       ;read from memory
        staa port       ;write to port
        inx             ;increment index
        cpx #hilim      ;done with block?
        bne loop2       ;loop if not
        nop             ;delay
        nop
        nop
        nop
        bra clref       ;continue
```

```
err3:   nop             ;Computer 3 must read from 2's memory
        nop             ;Computer 1 will idle
        nop
        nop
        ldx #lolim
loop3:  adda 0(x)       ;instructions to take up time
        anda 0(x)
        inx
        cpx #hilim
        bne loop3
        bra clref       ;continue
        nop

err4:   nop             ;make computer idle
        nop             ;time must equal other error times
        nop
        nop
        ldx #lolim
loop4:  adda 0(x)       ;form loop to take up time
        anda 0(x)
        inx
        cpx #hilim
        bne loop4

                        ;end of error routines
*****************************************************************


clref:  staa $fe64      ;clear error flip-flops
        jsr homer       ;clear screen
        ldaa $fe6f      ;read load switch
        bita #8         ;switch closed?

        beq prmem       ;if not, skip ahead

                        ;LOAD RAM WITH COUNT PATTERN
        ldx #lolim      ;set index
        ldaa #0         ;clear acc A
count:  staa  0(x)      ;store count
        inca            ;increment A
        inx             ;increment index
        cpx #hilim      ;done?
        bne count       ;loop if not


                        ;DISPLAY RAM BLOCK
prmem:  ldx #lolim      ;reset index
        ldaa #$ff       ;home cursor
        jsr print
1:      ldaa 0(x)       ;load byte from ram
        jsr hdump       ;display it
        inx             ;increment index
        cpx #hilim      ;done?
        bne 1b          ;loop if not
```

```
                        ;CHECK FOR ERRORS
        ldaa $fe6f      ;read error flip-flops and switch
        beq prmem       ;loop if all zero
rstrt:  staa $be68      ;output restart pulse
        wai             ;wait, restart pulse will force
                        ;program back to beginning


                        ;SUBROUTINES

homer:  ldaa #$ff       ;Home Erase subroutine
        jsr print       ;write home code
        ldaa #2         ;set t1 for 2 passes
        staa t1
        clra            ;set t2 for 512 spaces
        staa t2
1:      jsr spacer      ;call for space
        dec t1
        bne 1b          ;repeat until t1=0
        rts             ;return

spacer: jsr prspc       ;call print space
        dec t2          ;decrement space counter
        bne spacer      ;repeat if not zero
        rts             ;return

prspc:  ldaa #$a0       ;Print Space subr.
print:  staa out0       ;Print subr., out to TV
        clra
        staa out0       ;clear strobe
        rts             ;return


hdump:  tab             ;save A in B
        lsra            ;shift right four
        lsra
        lsra
        lsra
        jsr hexout      ;print one char.
        tba             ;B to A
        jsr hexout      ;print second char.
        jsr prspc       ;print space
        jsr prspc
        rts             ;return

hexout: anda #$f        ;strip upper half of byte
        cmpa #$a        ;is char 0-9?
        bmi 1f          ;branch if yes
        suba #9         ;subtact 9
        oraa #$c0       ;add ascii bits
        jsr print       ;out to tv
        rts             ;return
```

```
1:      oraa #$b0       ;add ascii bits
        jsr print       ;out to tv
        rts             ;return

                        ;END


        org $fffe       ;restart vector stored at top
        byte $ff,0      ;of ROM, points to bottom of ROM
```

****

ERROR ROUTINES FOR COMPUTER 2

```
                        ;Computer 2 must write to 1's memory
err1:   ldx #lolim      ;set index
loop1:  ldaa 0(x)       ;read from memory
        staa port1      ;write to port
        inx             ;increment index
        cpx #hilim      ;done with block?
        bne loop1       ;loop if not
        nop             ;delay
        nop
        nop
        nop
        bra clref       ;continue

err2:   nop             ;Computer 2 must read from 1's memory
        nop
        nop
        nop
        ldx #lolim      ;set index
loop2:  ldaa port1      ;read from port

        staa 0(x)       ;store in ram
        inx             ;increment index
        cpx #hilim      ;done with block?
        bne loop2       ;loop if not
        bra clref       ;continue

                        ;Computer 2 must write to 3's memory
err3:   ldx #lolim      ;set index
loop3:  ldaa 0(x)       ;read from memory
        staa port2      ;write to port
        inx             ;increment index
        cpx #hilim      ;done with block?
        bne loop3       ;loop if not
        nop             ;delay
        nop
        nop
        nop
        bra clref       ;continue
```

```
err4:   nop             ;make computer idle
        nop
        nop
        nop
        ldx #lolim
loop4:  adda 0(x)       ;form loop to take up time
        anda 0(x)
        inx
        cpx #hilim
        bne loop4

*************************************************************************

                ERROR ROUTINES FOR COMPUTER 3

err1:   nop             ;Computer 1 must read from 2's memory
        nop             ;Computer 3 will idle
        nop
        nop
        ldx  #lolim     ;set index
loop1:  adda 0(x)       ;instructions to take up time
        anda 0(x)
        inx             ;increment index
        cpx #hilim      ;block done?
        bne loop1       ;loop if not
        bra clref       ;if so, continue
        nop

err2:   nop             ;Computer 2 must read from 1's memory
        nop             ;Computer 3 will idle
        nop
        nop
        ldx #lolim      ;set index
loop2:  adda 0(x)       ;instructions to take up time
        anda 0(x)

        inx             ;increment index
        cpx #hilim      ;done with block
        bne loop2       ;loop if not
        bra clref       ;continue
        nop

err3:   nop             ;Computer 3 must read from 2's memory
        nop
        nop
        nop
        ldx #lolim
loop3:  ldaa port
        staa 0(x)
        inx
        cpx #hilim
        bne loop3
        bra clref       ;continue
```

```
err4:   nop             ;make computer idle
        nop
        nop
        nop
        ldx #lolim
loop4:  adda 0(x)       ;form loop to take up time
        anda 0(x)
        inx
        cpx #hilim
        bne loop4
*****************************************************************
```

NBS-114A (REV. 9-78)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBSIR 79-1927	2. Gov't. Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

Recovery from Soft Errors in Triplicated Computer Systems Operating in Lock-Step

5. Publication Date
November 1979

6. Performing Organization Code

7. AUTHOR(S)
A. L. Koenig and A. W. Holt

8. Performing Organ. Report No.
NBSIR 79-1927

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, DC 20234

10. Project/Task/Work Unit No.

11. Contract/Grant No.
IACRO 79-802

12. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

Defense Nuclear Agency
Washington, D. C. 20305

13. Type of Report & Period Covered
Topical

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

A Triply Modular Redundant (TMR) computer system operating in clocked lock-step is being investigated for an application requiring a Mean Time Between Failure of five years. No mechanical memories are used; this allows comparison of the outputs of the three computers to be made each clock period. The most novel contribution is the method of recovery from soft errors, such as those produced by lightning strokes or alpha particles. Data are provided on the uptime history of an experimental system, which uses three commercial microcomputers.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)*

Fault tolerant computer; soft errors, triply modular redundant; TMR.

18. AVAILABILITY ☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☐ Order From Sup. of Doc., U.S. Government Printing Office, Washington, DC 20402, SD Stock No. SN003-003-

☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

19. SECURITY CLASS (THIS REPORT)	21. NO. OF PRINTED PAGES
X UNCLASSIFIED	27
20. SECURITY CLASS (THIS PAGE)	**22. Price**
UNCLASSIFIED	$4.00

USCOMM-DC