

AD-A165 225

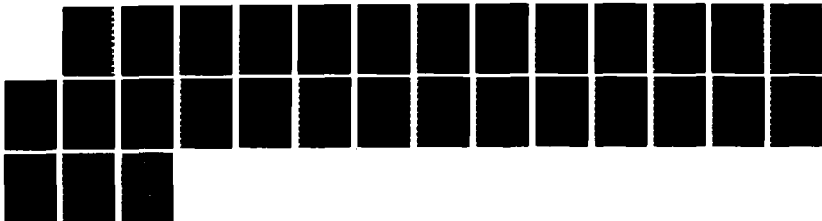
IRIG FORMAT 'B' DECODER(U) OKLAHOMA STATE UNIV
STILLWATER ELECTRONICS LAB J PEARSON ET AL JUN 85
SCIENTIFIC-4 AFGL-TR-85-0193 F19628-81-C-0079

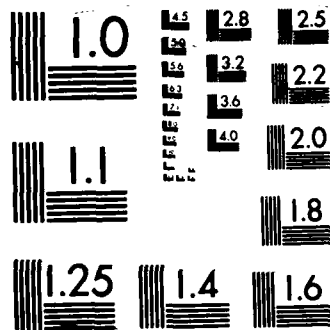
1/1

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART

85-
AFGL-TR-0193

IRIG FORMAT "B" DECODER

John Pearson
Jerry Grayson

Oklahoma State University
Electronics Laboratory - CEAT
Stillwater, Oklahoma 74078

June 1985

Scientific Report No. 4

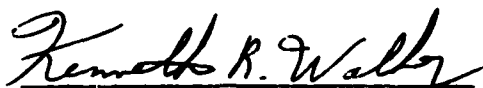
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

AIR FORCE GEOPHYSICS LABORATORY
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
HANSCOM AIR FORCE BASE, MASSACHUSETTS 01731

86 3 14 019

AD A165225

"This technical report has been reviewed and is approved for publication"



(Signature)

KENNETH R. WALKER

Contract Manager



(Signature)

RUSSELL G. STEEVES

Branch Chief

FOR THE COMMANDER



(Signature)

C. NEALON STARK

Division Director

This report has been reviewed by the ESD Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS).

Qualified requestors may obtain additional copies from the Defense Technical Information Center. All others should apply to the National Information Service.

If your address has changed, or if you wish to be removed from the mailing list, or if the addressee is no longer employed by your organization, please notify AFGL/DAA, Hanscom AFB, MA 01731. This will assist us in maintaining a current mailing list.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFGL-TR-85-0193	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) IRIG Format "B" Decoder		5. TYPE OF REPORT & PERIOD COVERED Scientific Report No. 4
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John Pearson Jerry Grayson		8. CONTRACT OR GRANT NUMBER(s) F19628-81-C-0079
9. PERFORMING ORGANIZATION NAME AND ADDRESS Electronics Laboratory - CEAT Oklahoma State University Stillwater, Oklahoma 74078		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62101F 765904BB
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Geophysics Laboratory Hanscom Air Force Base, Mass 01731 Attn: Ken Walker/LCR		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 22
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release: Distribution Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Microprocessor, IRIG "B" Time Code, DEcoder, Single Task, Multi-task.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes a special application of a microprocessor for decoding a serial bit stream of data from an IRIG "B" time code generator. The report covers theory of operation, suggestions for improvement and modification and a listing of the system software.		

SUMMARY

The Oklahoma State University Electronics Laboratory has developed a microprocessor-based IRIG format "B" real-time decoder. This circuit will provide a parallel BCD output to any device which requires this information. This decoder accepts standard unmodulated IRIG "B" code and provides the time in a continuously available parallel output. The 34 bit BCD output includes tenths of seconds, seconds, minutes, hours, and days. This output is updated ten times per second. The system has a backup timer which will be implemented when the system senses a failure of the IRIG input signal. However, the system will return to its normal mode of operation upon return of the IRIG signal.

National Semiconductor's NSC800 microprocessor has been implemented to control the all-CMOS decoder. The system is based on software for the purpose of reducing weight, parts count, power consumption and cost. Basically, the system is a microcomputer which is programmed for only one task. However, the system is not dedicated and other programs may be executed transparently. From a hardware point of view, the same general system could be used as a basis for other NSC800-controlled applications.

This report covers design details, theory of operation , and suggestions on modifying the software to accomodate different clock speeds. In addition , a listing of the system software is provided in the appendix.

TABLE OF CONTENTS

Topic	Page
1.0 INTRODUCTION.....	1
2.0 SPECIFICATIONS.....	2
2.1 INPUTS.....	2
2.1.1 Power Requirements.....	2
2.1.2 Code Input.....	2
2.2 OUTPUTS.....	2
2.3 SYSTEM TIME BASE.....	2
3.0 THEORY OF OPERATION.....	3
3.1 Hardware.....	3
3.2 Software.....	4
3.2.1 Primary Decoder Software.....	4
3.2.2 Back-up Timer Software.....	7
4.0 ADJUSTMENTS.....	14
4.1 Accuracy of Back-up Timer.....	14
4.2 Changing System Clock Frequency.....	14
5.0 SYSTEM DEDICATION.....	15
Appendix A Complete software listing.....	16

LIST OF ILLUSTRATIONS

Figure No.	Title	Page
3.3	Block Diagram.....	9
3.4	Flow Chart.....	10
3.5	IRIG-B Code Illustration.....	12
3.6	System Memory Map.....	13

1.0 INTRODUCTION

The IRIG format 'B' decoder accepts serial unmodulated IRIG time code from an external source and provides a parallel BCD time-of-year output. This output may be used at any time by any device that requires this information.

Upon power-up, the circuit will begin searching for the IRIG framing bits. When these bits are found, the system begins reading the incoming bits and formatting them into digits. As each group of digits is formatted into seconds, minutes, hours, and days, they are sent to the output ports and software buffers in RAM. This is the only frame which will exhibit significant delays. The output for all further frames is controlled by the software buffers and only refreshed by the formatted IRIG code. In this way, the great majority of input-to-output delay is eliminated.

In the event of IRIG failure, an on-board backup timer will respond within seventeen milliseconds to continue providing the necessary output, assuming that the software buffers contained the correct time prior to failure. The backup timer will continue until the IRIG code is resumed.

The system also incorporates a program-controlled hardware overseer which checks to see that the CPU does not ever stay 'lost'. If the system becomes 'lost' for more than 35 milliseconds, the system will be totally reset and begin searching for the framing bits again. If no bits are found while searching for the frame, the backup timer will take over.

2.0 SPECIFICATIONS

2.1 INPUTS

2.1.1 Power Requirement

- i. 5 volts D.C. at approximately 32 mA

2.1.2 Code Input

- i. Format: IRIG B BCD serial time of year. (Control functions and straight binary seconds may be included in the code and will not affect the operation of the decoder).

- i.i Amplitude: input logic high - above 3.5 vdc
input logic low - below 1.5 vdc

- i.i.i. Input Capacitance: 10-15 picofarad

iv. Input to Output

- Accuracy: Primary: 5.4ms delay max
Backup: accuracy is dependent upon the system time base.

2.2 OUTPUTS

2.2.1 Code Output

- i. Format: Parallel 34 bit BCD time of
year
seconds/10 4 bits
seconds 7 bits
minutes 7 bits
hours 6 bits
days 10 bits

- i.i Amplitude: logic high, 5 VDC
logic low, 2mV DC

- iii. External load: 500 ohms maximum

2.3 SYSTEM TIME BASE

- i. Type Crystal Controlled Oscillator
- ii. Frequency 2 MHz

3.0 THEORY OF OPERATION

3.1 Hardware

Refer to the block diagram in Figure 3.3 or Schematic drawing number D42CR02. The heart of the decoder is the NCS800N CPU. In this application, the CPU will respond to one of four possible conditions:

1. If IRIG is present, the CPU will be interrupted on every negative edge of the incoming code.
2. If the IRIG input fails, the CPU will be interrupted by a failure timer that has sensed the absence of the IRIG code.
3. If IRIG has failed and the CPU has been interrupted by the failure timer, it will then be interrupted by a 100 Hz self-generated pulse stream.
4. If the CPU gets lost, an automatic system reset will occur.

In case # 1, when IRIG code is present, the NSC810 measures the duration of the incoming pulses and makes this data available to the CPU. The CPU then determines whether the pulse is a logic '1', a logic '0', or a position marker. (see IRIG format illustration, page 12). Section 3.2 will discuss how this is done.

The interrupt caused by IRIG failure is produced by the NSC810 PIO, (case # 2). This chip will interrupt the CPU if the expected interrupt caused by incoming IRIG does not occur within 17ms. Seventeen milliseconds was chosen since IRIG should interrupt the system at least once every 16ms. When the NSC810 does interrupt the CPU, the NCS810 is reconfigured to produce a 100Hz pulse stream that replaces the missing 100Hz IRIG code and allows the decoder to continue the time from where it left off prior to IRIG failure, (Case #3)

In case # 4, the system will be totally reset if a certain "known" subroutine is not executed periodically. This insures that if the CPU becomes "lost", the entire system will be reset. This is accomplished by restarting the 35ms CD4098 one-shot every time the "known" subroutine is executed. Thus, if the CPU becomes lost, the first half of the one-shot will "time-out" and trigger the second half of the CD4098 which in turn will hold the RESET line low for a short period of time.

The NSC810 PIO contains 128 bytes of useable R/W memory, 22 I/O ports, and two internal timers designated as a timer zero and timer one. The RAM is used for the system stack, software flag storage, and timer buffers. Nineteen of the 22 I/O ports are used for IRIG input and timer I/O. Timer zero is used to determine the duration of the incoming IRIG pulses. Timer one is used to look for the possible 17ms absence of IRIG in

the case of failure . Timer one is also used to produce the 100 HZ pulse stream when the system is in its backup mode.

The NSC-831 is also a PIO, but contains only ports. The NSC831 (no ROM) was chosen over the NSC830 (with ROM) since the development of the system would require many software changes. This method will also allow the addition of other programs that may be added to the existing 27C16 EPROM.

Since the NSC800 family utilizes multiplexed address and data lines, the 82PC12 demultiplexer was used to provide separate address and data lines to the 27C16 EPROM, which stores the decoder software.

3.2 Software

The system software can be broken into two main segments- primary and backup. Refer to the flowchart in Figure 3.4 (a) and (b). Under normal conditions, IRIG is being read. The decoder output is a reflection of the incoming code and only primary software will be executing. In the case of IRIG failure, the backup software will take over and begin producing the necessary output until the IRIG code input is resumed. (See page 16 for a complete software listing.)

3.2.1 Primary Decoder Software

The primary or main program begins by initializing the output ports. These ports may be used as inputs or outputs. In this case, all are configured as outputs except the NSC810's port C bits 3 and 4 which are used as inputs by the timers. Upon power-up of the system, all ports are automatically configured as inputs for the purpose of preventing any ambiguous output levels from triggering devices that may be connected to these outputs. A port in the input mode is in a high impedance state. In addition to initializing the ports, two flags described below are set to zero:

'OUTENBL' This flag keeps the software buffers disabled until the first frame of IRIG has been decoded and sent to the ports.

OUTENBL = \$00 during the first pass of IRIG after power-up. The software buffers are disabled.

OUTENBL = \$FF after the first pass of IRIG. The software buffers will be enabled until the system restarts from power-up again.

'BAKFLAG' The NSC-810's timer 1 is used for two purposes. It is used as an IRIG failure timer during the primary program sequence and as the 100 Hz pulse generator in the event of IRIG failure. Since both timer functions use the same output, and hence the same interrupt line, this flag is needed to differentiate between the two functions.

BAKFLAG = \$00 when timer 1 is configured as the IRIG failure sensor. The 100Hz pulse generator is disabled.

BAKFLAG = \$FF when IRIG has failed, timer 1 has been configured as the 100Hz pulse generator, and the program is in the backup mode. The 17ms IRIG failure timer is disabled.

The next step of the main program is to find the framing bits of the IRIG code. This is accomplished by testing each incoming bit until two consecutive position markers (8ms) are sensed. This segment uses a subroutine called 'GET' which is responsible for reading one incoming bit and determining what it is (i.e. logic 0 or 1, or position marker). After the program is synchronized, the tenths and hundredths of seconds counters are reset to zero. The hundredths counter increments once for every incoming pulse. That pulse will either be due to incoming IRIG code or the system-generated 100Hz back-up pulse generator. When the hundredths count buffer, 'HUN', reaches ten, it resets to zero and increments the tenths count buffer, 'TEN'. Then, as more pulses are read, the tenths count increments until it is time to increment the seconds count, minutes count, etc. Therefore, after the first IRIG code stream initializes these software buffers, they continue to calculate the present time every hundredth of a second and send it to the output ports every tenth of a second. This way, the program does not have to wait for the delayed IRIG serial data to be read and decoded each time before it is sent to the ports. The IRIG time is simply sent to the software buffers once per second as a reminder of what the buffers should contain. This insures that if the incoming IRIG code is changed, or an error is produced by the buffers, the error will be corrected during the next IRIG frame.

Now that all ports and buffers have been initialized and the program has recognized the framing bits, the program will begin reading code. Since the incoming code is pulse width logic, the NSC810's timer 0 is used to measure the length of each incoming pulse. The process begins by calling the subroutine, 'GET'. This subroutine sets timer 0 to a mode that causes it to act as a down-counter. It begins decrementing from a predefined 'modulus' value whenever a positive transition occurs during the incoming IRIG code. The modulus will decrement once for each 64 system clock pulses, and stop decrementing after the IRIG pulse has dropped back low again. This function is also determined by the timer mode selection.

Not only does the negative transition cause the modulus to stop decrementing, but it also causes a CPU interrupt via RSTA. (Case #1 as discussed on page 3). When this interrupt occurs, the decremented modulus value is read from the NSC810 and used to calculate the width of the pulse and hence, the pulse's logic value.

In our case, the modulus was initially loaded with \$FFFF. The system clock frequency is 2 MHz which makes the timer effectively see an input frequency of $2 \text{ MHz} / 64 = 31.250 \text{ KHz}$, resulting in an effective timer period of 32 microseconds. Therefore,

$$\text{PULSE WIDTH} = (\text{MODULUS} - \text{DECREMENTED VALUE}) \times 32\mu\text{s}.$$

This width can then be compared to expected IRIG pulse widths to determine what the pulse is. For this software the width definitions are:

	Accepted	Expected
Definition:	Pulse Duration:	Duration:
logic '0'	less than 3.5ms	2ms
logic '1'	3.5ms to 6.5ms	5ms
position marker	greater than 6.5ms	8ms

Before the defined bit is sent back to the calling routine, the software counters are incremented by one hundredth.

Since IRIG code comes in the same code word structure every frame, (e.g. seconds, then minutes, then hours etc.) we know when to expect each digit of the BCD time. These BCD digits are then sent to both the output ports and to the "look-ahead" software timer buffers where they overwrite any previous value. It is assumed that under the expected conditions, though, the software-calculated time and decoded IRIG

time will be equal. After the DAYS value has been read and sent to the ports and buffers, the program jumps back to the framing routine to get ready to read the next frame.

The "GET" subroutine also sends a pulse to the automatic system reset circuitry composed of the CD4098 monostable multivibrator. This circuitry is activated by reading or writing to any memory location on page \$30. If the system is operating in either the primary or backup mode properly, the "GET" subroutine will be executed every 10ms. Therefore, every 10ms a pulse is sent to the first half of the CD4098. These continuously-applied pulses keep the one-shot in its high state and prevent it from timing out and producing a high-to-low transition. If the CPU gets lost and does not send the retrigger pulse, after 35ms the first half of the one-shot will time out and fall from high to low. This negative transition causes the second half of the CD4098 to pull low for a short period of time. Since the output is connected to the CPU's reset input, the entire system will be reset and the program will begin from location \$0000 in ROM. This reset, however, will not destroy the contents of RAM. Thus, even if the system was in its backup mode, a reset will only cause a small delay and will not destroy the timer buffers or the software flags.

3.2.2 Back-up Timer Software

In order for the back-up timer to function properly, it must first sense the absence of IRIG code, update the software buffers, and then check for the return of the IRIG code.

The absence of IRIG input is detected by first realizing that the longest delay between any two IRIG negative pulse transitions is 16 milliseconds. (See IRIG illustration, page 12). On the negative transition of every IRIG pulse, a 17 millisecond timer is started. When the 17ms timer 'times out', a pulse is sent to the CPU via the RSTA interrupt. Again, since the longest delay expected when reading RIG code is 16ms, if IRIG is still present, it should have already interrupted the CPU before the 17ms timer did. The IRIG interrupt also disables all further interrupts, and thus the 17ms 'time out' pulse will be ignored.

Now, assuming that the CPU has been interrupted by the IRIG failure timer, the program jumps to the back-up timer routine. This routine generates a 100 Hz pulse stream by reconfiguring the same timer that was just used for IRIG failure detection. This new pulse stream simulates the 100 Hz IRIG pulse stream. Once this independent pulse generator is started, the program returns to the framing routine in hopes of

detecting IRIG code again. Since the back-up routine was executed, 'BAKFLAG' has been reset so that the IRIG failure detector is disabled, thus preventing the reconfiguration of the 100Hz timer again. Both NMI and RSTA are enabled during the back-up routine execution. This allows IRIG to regain control upon its return by interrupting the CPU via the NMI interrupt input.

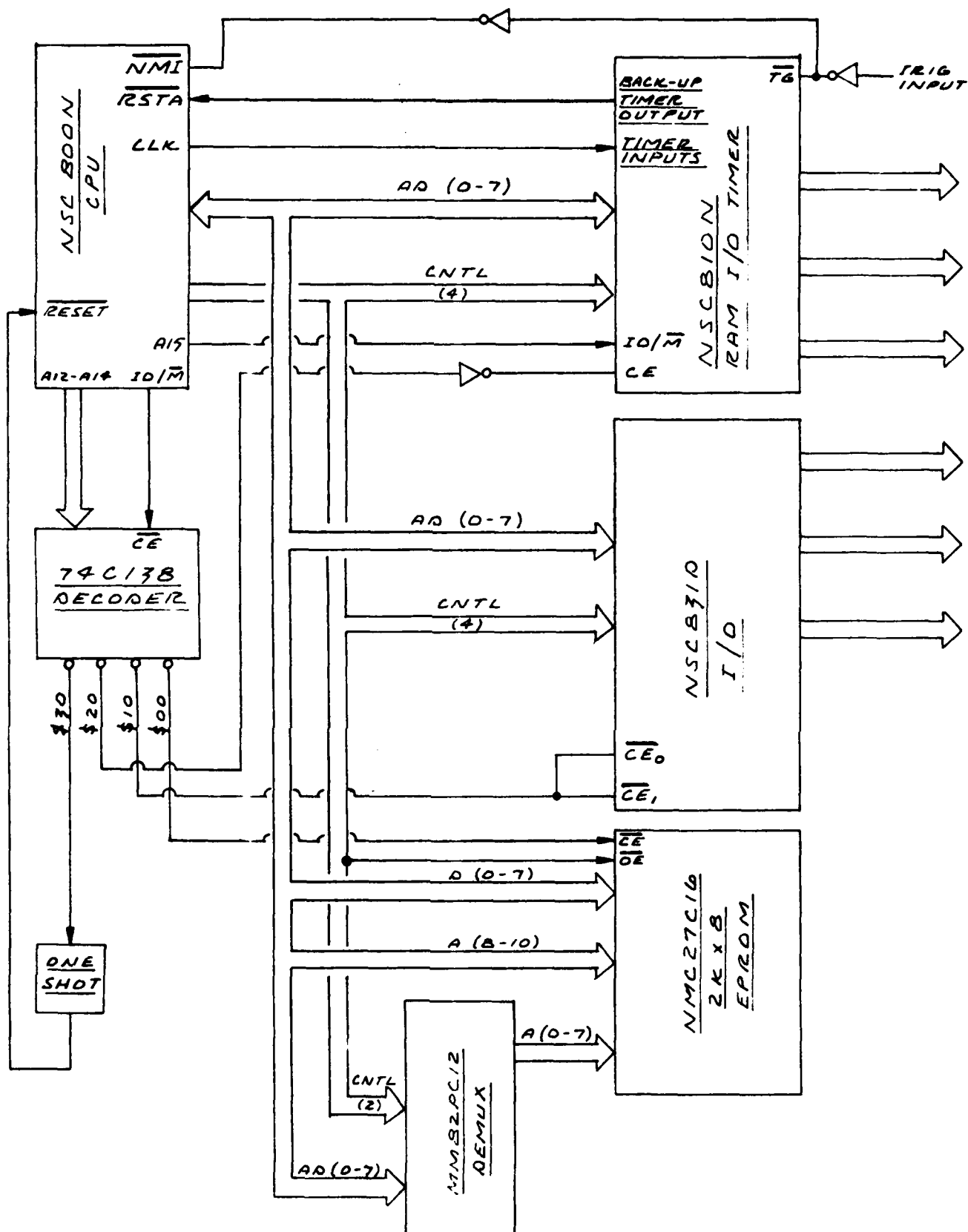


Figure 3.3 IRIG 'B' Decoder Block Diagram

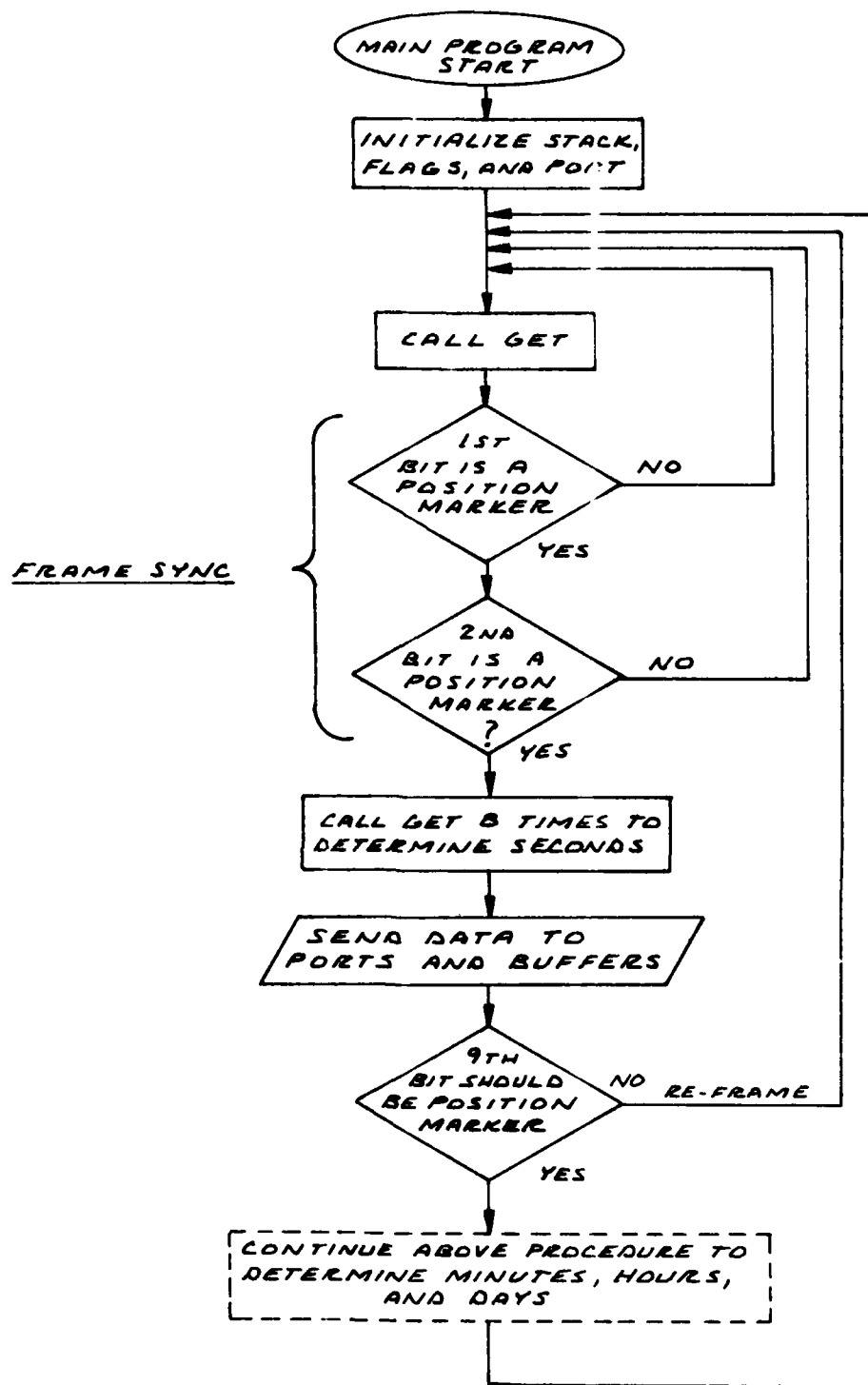


Figure 3.4 (a) Main Program Flowchart

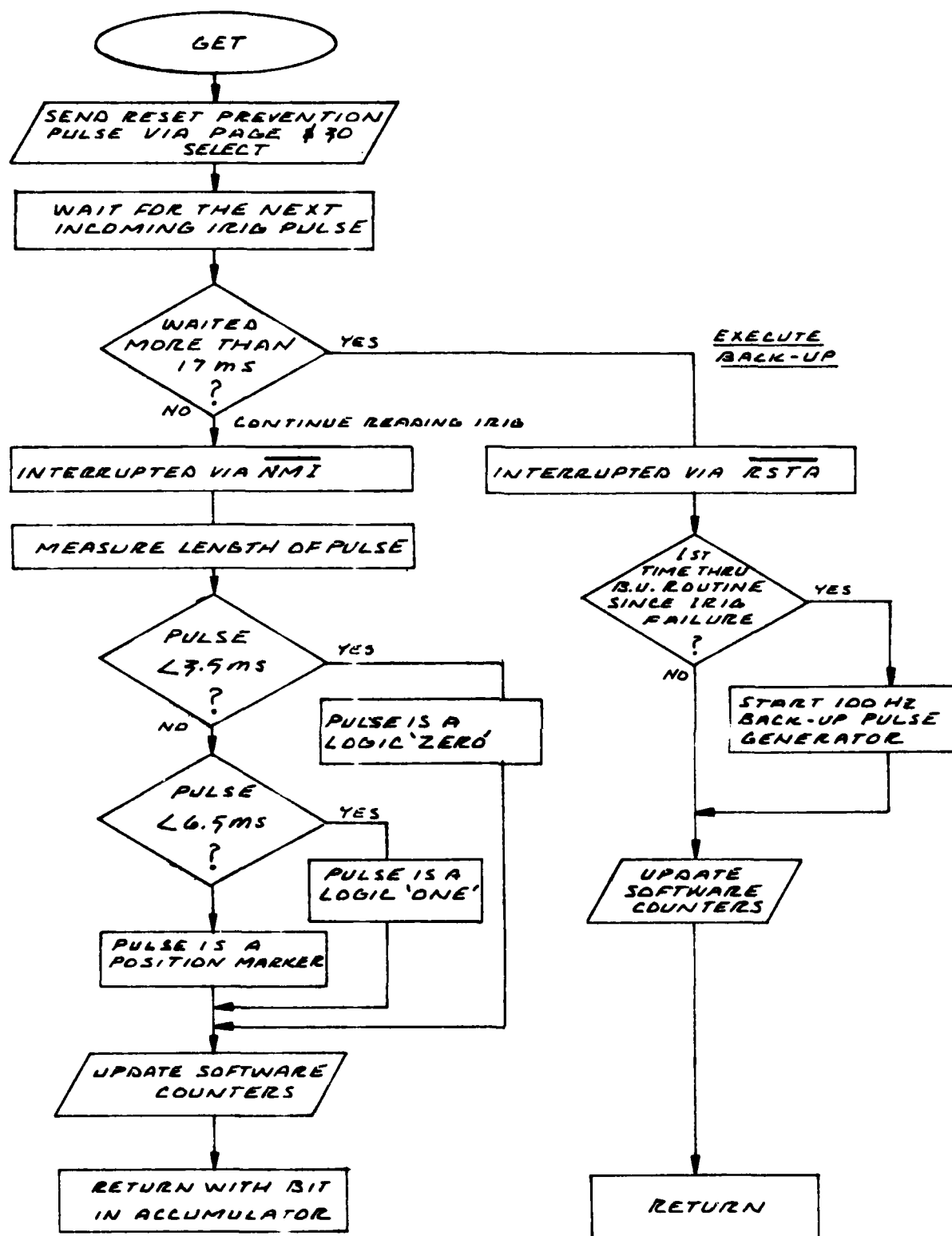


Figure 3.4 (b) Flowchart of Subroutines

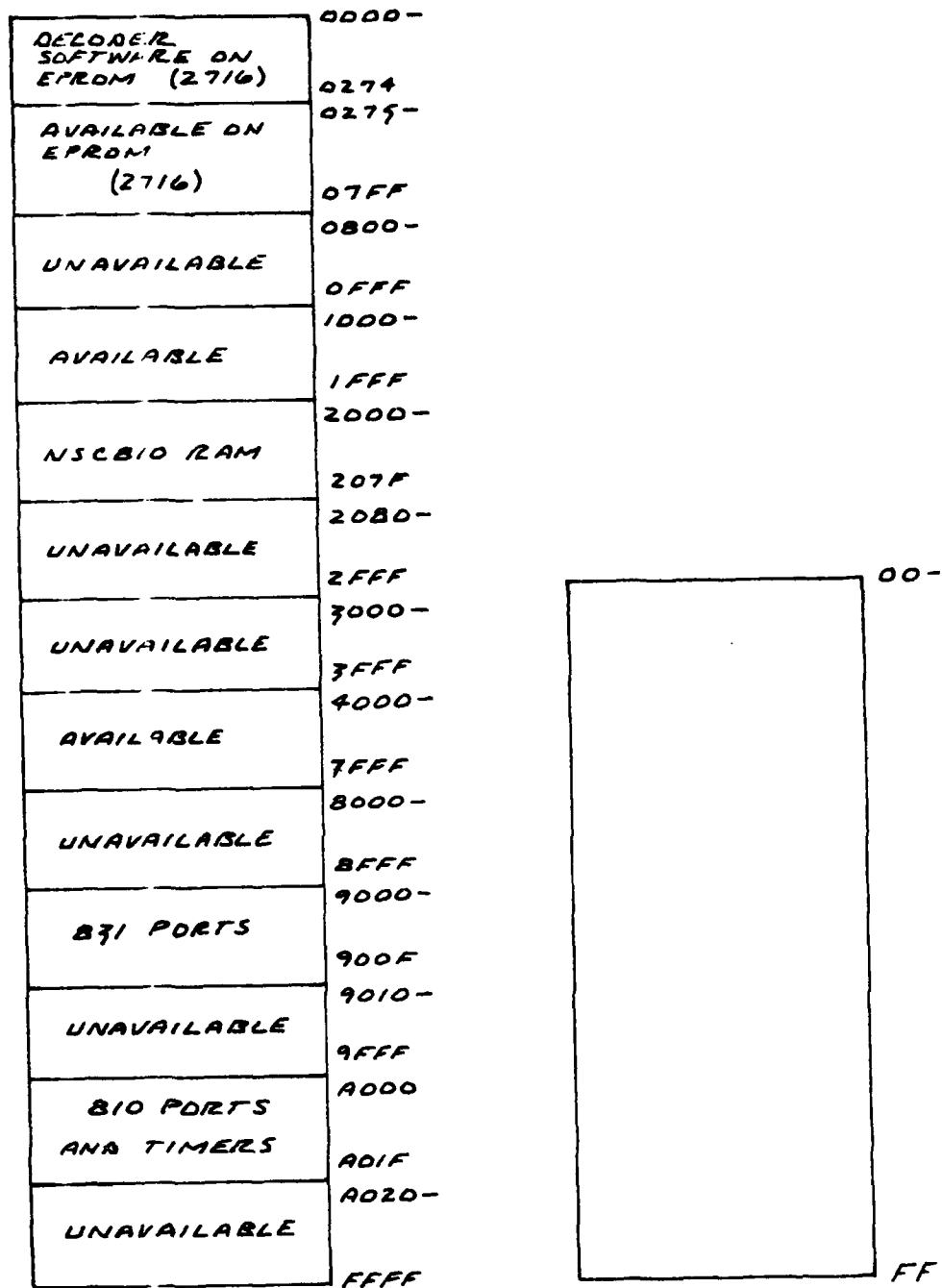


1. **TIME FRAME:** 1.0 second.
2. **CODE DIGIT WEIGHTING OPTIONS:** BCD, SB or both:
 - a. Binary Coded Decimal Time-of-Year CODE WORD — 30 binary digits.
 - (1) Seconds, minutes, hours and days. Recycles yearly.
 - b. Straight Binary Time-of-Day CODE WORD — 17 binary digits.
 - (1) Seconds only. Recycles each 24 hours. (06399)
3. **CODE WORD STRUCTURE:**
 - a. BCD: Word begins at INDEX COUNT 1. Binary-coded elements occur between POSITION IDENTIFIER ELEMENTS (seven for seconds, seven for minutes; six for hours; ten for days) until the CODE WORD is complete. A POSITION IDENTIFIER occurs between decimal digits in each group to provide separation for visual resolution.
 - b. SB: Word begins at INDEX COUNT 80. Seventeen binary-coded elements occur with a POSITION IDENTIFIER between the 0th and 10th binary-coded elements.
4. **LEAST SIGNIFICANT DIGIT:** occurs first.

5. ELEMENT RATES AVAILABLE:

- a. 100 per second (basic Element rate)
 - b. 10 per second (POSITIVE IDENTIFIER Rate)
 - c. 1 per second (Frame Rate)
- 6. ELEMENT IDENTIFICATION:**
- a. "On-Time" reference point for each Element is its leading edge.
 - b. INDEX MARKER duration: 2 milliseconds (Binary zero or uncoded Element)
 - c. CODE DIGIT duration: 5 milliseconds (Binary one)
 - d. POSITION IDENTIFIER duration: 8 milliseconds
 - e. REFERENCE MARKER — one per second; Two consecutive POSITION IDENTIFIERS.
(The "On-Time" point, to which the CODE WORD refers, is the leading edge of the second POSITION IDENTIFIER.)
- 7. RESOLUTION:** 10 milliseconds (unmodulated); 1 millisecond (modulated).
- 8. CARRIER FREQUENCY:** 1 kHz when modulated.

IRIQ STANDARD FORMAT B
SIGNAL BOX
100 PPS CODE
REFERENCE IRQ DOCUMENT 104-71



MEMORY MAP

I/O MAP

Figure 3.6 System Memory Map

4.0 ADJUSTMENTS

4.1 Accuracy of the Back-Up Timer

The back-up timer accuracy is directly dependent upon the accuracy of the system's timebase, since the 100 Hz pulse generator is produced by counting a predetermined number of clock periods from the CPU's clock output. The frequency of the back-up timer can easily be set to 100Hz and fine tuned with small changes in the CPU's oscillator input frequency. Presently, under software control, the tuning resolution is in steps of 5.0003 milliHz for every one-digit change in timer 1's modulus value.

4.2 Changing System Clock Frequency

Although the decoder relies totally on the CPU clock frequency of 2 MHz as a reference for all timing operations, a change in the system's clock frequency requires only minor changes in the decoder's software. The range of acceptable CPU speeds depends upon the mode of the NSC810's timers, and the initial value of the timer's modulus. For a more complete description of the operation of the timers, see National Semiconductor's NSC810 product description.

If the system clock frequency is increased or decreased, the modulus will decrement faster or slower, respectively. Therefore, it is necessary to make changes in the software routines which use this modulus value to define the incoming pulses, both primary and backup. The following general procedure can be followed.

If the frequency is increased, the numbers that the modulus is compared to (after the incoming IRIG pulse has caused the modulus to decrement to a certain value) will need to be changed since the modulus will be decrementing faster. (Refer to the 'BIT' subroutine in the software listing in appendix A). These numbers are easily calculated. For example:

Incoming position marker has a pulse width of 8ms
CPU Clock Period = 0.5us
Timer input PRESCALE = divide by 64 (defined by timer mode)
2 Byte modulus (also defined by timer mode)

Therefore, the modulus value will decrement every $(64 \times 0.5\text{us}) = 32\text{us}$.

The expected modulus value after an 8ms pulse has been read by the timer would be $\$FFFF - (8\text{ms}/32\text{us}) = \text{about } \$FF05$. A two byte modulus was used since a one byte

modulus can only measure a maximum width of $\$FFX32us = 8.16ms$. If the position marker was longer than 8.16ms, a one byte modulus would cause an error. If a two byte modulus was used, the high byte would simply decrement by one every time the low byte reached zero. Therefore, to determine what each pulse is, the high byte of the modulus is checked to see if its least significant bit is equal to zero since the high byte will decrement from $\$FF$ to $\$FE$. If so, we assume that the pulse is a position marker in excess of 8.16ms. If the LSB of the high byte is still a 'one', the pulse must have been less than 8.16ms. The low byte is then checked for greater than 145 which represents a time less than 3.5ms. A count greater than 145 indicates a logic zero and a count less than 145 indicates either a logic one, or a position marker less than 8.16ms long. If the count was less than 145, it is checked for greater than 52. If the count is greater than 52, the pulse was more than 3.5ms long and less than 6.5ms, indication that pulse was probably a 5ms logic 'one'. If the count was less than 52, it is assumed that the pulse probably was a position marker shorter than 8.16ms.

Since one of the above three pulses will occur every 10ms, the software timer buffers are incremented immediately after the pulse has been defined.

5.0 SYSTEM DEDICATION

The decoder was designed to be as non-dedicated as possible to the single task of translating IRIG code. The system uses only two of the available five interrupts to accomplish the decoding. Presently, after the program decodes each pulse it waits for the next one by halting the processor and enabling the interrupts. The processor waits anywhere from 3ms to 15ms before the next IRIG bit interrupts the CPU from its HALT state. Since execution of the primary program takes less than 500us, at least 2.5ms are available out of every 10ms for other tasks before the CPU is expected to be interrupted again. If the program is in the back-up mode due to IRIG failure, the 100 Hz interrupt sequence is self-generated and predictable. Therefore, in the back-up mode there is at least 9.5ms of CPU time available out of every 10ms. Any auxilliary program may be executed during this period in the primary or backup mode as long as it either returns to the decoder program prior to the next expected IRIG pulse or makes sure that the RSTA interrupt is enabled so that IRIG or the back-up timer may interrupt the auxilliary program when needed. It is also imperative that the CPU stack be restored to its original state prior to returning to the decoder program.

APPENDIX A
IRIG B DECODER SOFTWARE LISTING

```

ASEG
.280
;
; ***** IRIG FORMAT 'B' DECODER SOFTWARE LISTING *****
;----- DEFINITIONS -----
;PIO related definitions
;
MDR0 EQU 0A007H ;mode definition register address
TMR0 EQU 0A018H ;timer 0 mode register address
MDO EQU 00H ;mode definition register set for BASIC I/O
TMO EQU 11011011B ;timer 0 mode set for pulse width meas'mnt
TIMER0 EQU 0A010H ;modulus value address.... timer 0
TOSTAR EQU 0A015H ;write causes timer 0 to 'start'
TOSTOP EQU 0A014H ;a write causes T0 to 'stop'
;
TMR1 EQU 0A019H ;timer 1 mode register address
TMIDEL EQU 01011001B ;Timer 1 Mode set for a 17mS delay
TM1PULS EQU 01011001B ;Timer 1 Mode set for 100Hz pulse generator
TIMRONE EQU 0A012H ;T1 modulus address
T1STAR EQU 0A017H ;TIMER 1 start address
T1STOP EQU 0A016H ;TIMER 1 stop address
;
ICR EQU 0BBH ;INTERRUPT CONTROL REGISTER
;
DDRA EQU 0A004H ;ADDRESSES OF NSC810 DDR'S
DDRB EQU 0A005H
DDRC EQU 0A006H
PORTA EQU 0A000H ;ADDRESSES OF NSC810 PORTS
PORTB EQU 0A001H
PORTC EQU 0A002H
;
DDA EQU 9004H ;ADDRESSES OF NSC831 DDR'S
DDB EQU 9005H
DDC EQU 9006H
PRTA EQU 9000H ;ADDRESSES OF NSC831 PORTS
PRTB EQU 9001H
PRTC EQU 9002H
;
HUN EQU 2000H ;LOCATIONS OF SOFTWARE TIMER
TEN EQU 2001H ; BUFFERS
SECND EQU 2002H
MINUT EQU 2003H
HOUR EQU 2004H
DAYS EQU 2005H ;low 8 bits of DAYS
DH EQU 2006H ; high 2 bits of DAYS
BAKFLAG EQU 2007H ;backup 100hz pulse generator enable
OUTENBL EQU 2008H ;output enable for software timers
STACK EQU 2080H
AOKAY EQU 3000H
;
;
;
ORG 0000
JP START ;SKIP OVER INTERRUPT VECTORS
;----- VECTORS -----
;
ORG 3CH
CALL BACKUP ;RESTART [A] interrupt vector
POP AF ; simulate returns from
POP AF ; RSTA & 'GET'
JP FRAME1 ;continue checking for irig
;
ORG 66H ;non-maskable interrupt vector
EX AF,AF
LD A,(BAKFLAG)
CP 00H
JR Z,EXCH
JP START
EXCH: EX AF,AF ;since we have interrupted the execution
CALL BIT ;of RSTA, we will start back at square 1
RET ; since the stack is probably messed up
;RETURN FROM NMI

```

```

;
;----- INITIALIZATION -----
;
      ORG      80H
;
      LD      A,0
      LD      (OUTENBL),A
START: LD      A,0FFH      ;top of RAM
      LD      (DDRA),A    ;set up DDR's of ports
      LD      (DDRB),A    ;these ports has inputs
      LD      (DDA),A
      LD      (ddb),A
      LD      (DDC),A
      LD      A,27H
      LD      (DDRC),A    ;NSC810's PORT 'C' has some inputs
INIT:  LD      SP,STACK    ;set up stack in NSC810's top of RAM
      LD      A,0         ;clear the backup flag
      LD      (BAKFLAG),A
;
;----- MAIN PROGRAM -----
;
      The primary program checks for the presence of the 2 IRIG
; markers. When these are found, the program begins reading the
; incoming code. The code is input in serial, changed to parallel,
; and then sent to the parallel output ports. In addition to being sent
; to the ports, the time code is also sent to the software timers
; which update the ports every 10 milliseconds, and thus eliminate
; most of the INPUT-to-OUTPUT delay error.
;
FRAME1: CALL   GET          ;the program will fall through this loop
      LD      A,C          ; only when 2 consecutive pulses greater
      CP      0FFH         ; than 6.5 mS are detected
      JR      NZ,FRAME1    ; (MARKERS)
FRAME2: CALL   GET
      LD      A,C
      CP      0FFH
      JR      NZ,FRAME1
;
      LD      A,0          ;reset hundredths and tenths buffers
      LD      (HUN),A
      LD      (TEN),A
;
;NOW WE'RE IN SYNC, SO LETS START READING TIME CODE
;
SEC:   CALL   DIGITS        ;get the first 2 digits of seconds
      LD      (PORTA),A    ;send seconds to port
      LD      (SECND),A    ;send seconds to buffer
;
MIN:   CALL   CHECK        ;this bit should be an 8mS marker
      JR      NZ,FRAME1    ; if not, get re-framed
;
      CALL   DIGITS        ;get next two digits
      LD      (PORTB),A    ;send minutes to port
      LD      (MINUT),A    ;send minutes to buffer
;
HOURS: CALL   GET          ;ignore this bit
      CALL   CHECK        ;make sure this one is a position iden-
      JR      NZ,FRAME1    ; tifier (8ms marker)
      CALL   DIGITS
      LD      (PRTB),A     ;send hours
      LD      (HOUR),A
;
DYS:   CALL   GET          ;ignore this bit
      CALL   CHECK        ;check position again
      JR      NZ,FRAME1
      CALL   GET4
      CALL   SHIFT
      LD      B,A         ;save <A>
      CALL   GET          ;ignore this one
      CALL   GET4         ;get the next 4 bits
      OR      B
      LD      (PRTA),A     ;send 1st 8 bits to ports
      LD      (DAYS),A     ;send to buffer

```

```

THERE: CALL    CHECK                ;check position
        JR      NZ,FRAME1
        CALL    GET
        LD      B,C
        SRL     B                    ;now, get the last 2 bits of DAYS
        CALL    GET                  ; to determine the MSD
        LD      A,C
        OR      B
        RLC     A
        RLC     A
        LD      (PORTC),A            ;send final digit to port
        LD      (DH),A              ;send to buffer
        LD      A,OFFH              ;enable the buffer outputs
        LD      (OUTENBL),A
        JP      FRAME1

;
;..... MAIN PROGRAM SUBROUTINES .....
;
DIGITS: PUSH    BC                    ; this routine expects a certain
        CALL    GET4                  ; sequence of 8 incoming bits
        CALL    SHIFT                 ; to represent two IRIG digits
        LD      B,A
        CALL    GET4                  ; The two BCD digits are returned
        SRL     A                    ; in the accumulator - the 1st
        OR      B                    ; in LSB position.
        POP     BC
        RET

;
CHECK:  CALL    GET                    ;this routine checks to see
        LD      A,C                    ; if the next bit is an 8mS
        CP      OFFH                  ; position identifier as expected.
        RET                                     ; If it is not, an error has
                                           ; occured and the program must
                                           ; re-frame

;
SHIFT:  SRL     A                    ;This routine shifts the
        SRL     A                    ; upper 4 bits of the accum.
        SRL     A                    ; into the lower 4 bits
        SRL     A
        RET

;
;
;
GET4:   PUSH    BC                    ; This routine gets the next 4 bits from
        CALL    GET                    ;the incoming code string and returns them in
        LD      A,C                    ;the upper nibble of <A>.
        SRL     A
        CALL    GET
        OR      C
        SRL     A
        CALL    GET
        OR      C
        SRL     A
        CALL    GET
        OR      C
        POP     BC
        RET

;
;
;
GET:
;
; This subroutine waits for the next incoming pulse (or bit) and
; returns in the <C> register a unique value depending on the
; computed definition of the pulse.
; The incoming pulse causes an interrupt through $0066 interrupt
; vector, the interrupt service routine is executed, and then returns
; to this routine. The program then returns to the calling program
; with the calculated bit definition in register <C>.
; While waiting for the next expected IRIG pulse, a 17ms timer
; is started. If this timer reaches zero before the next IRIG pulse
; is sensed, the program will assume IRIG has failed, and will begin
; execution of the BACKUP routine.

```

```

;
PUSH    AF
LD      A,7
LD      (TMR0),A      ;reset T0
LD      A,TM0          ;load timer 0 with the desired
LD      (TMR0),A      ; timer mode
LD      A,MDO          ;set mode definition register
LD      (MDR0),A
LD      HL,0FFFFH      ;load timer 0 MODULUS with $FFFF
LD      (TIMER0),HL
LD      (TOSTAR),A     ;enable the timer
;
; If IRIG is still on, we should not make it all the way through the
; following 17ms delay caused by T1. If we do, it indicates IRIG failure,
; and we will jump to the backup timer routine.
;
LD      A,(BAKFLAG)     ;if BAKFLAG=0, we will execute
CP      0FFH           ; the code immediately below
JR      Z,WAIT          ; (this starts our 17ms timer)
;
LD      A,7
LD      (TMR1),A       ;reset T1
LD      A,TM1DEL        ;set-up timer to create delay
LD      (TMR1),A
LD      A,42H
LD      HL,4268H        ;this 2-byte modulus, ($4268), will produce a
LD      (TIMRONE),HL    ; 17ms count-down delay
LD      (T1STAR),A      ;start the delay
WAIT:   LD      A,08H    ;un-mask RSTA in the I.C.R
OUT     (ICR),A
LD      A,(AOKAY)       ;send acknowledge pulse via PAGE 30 select
POP     AF              ;enable both NMI and RSTA and wait
EI                      ; to see which is first
HALT                    ;(any other program may be executed here!)
;Return to here from interrupt service routine
RET                      ;return from 'CALL GET'
;
BIT:
;----- NMI interrupt service routine -----
;
;.....INTERRUPTED BY IRIG CODE ONLY .....
;
PUSH    AF
LD      (T0STOP),A     ; stop timer 0
LD      HL,(TIMER0)    ;get count from timer
BIT     0,H            ;test bit 0 of high byte
JR      Z,MARKER       ;a marker if >8.16ms
LD      A,L
CP      145
JR      NC,ZERO
CP      52
JR      NC,ONE
MARKER: LD      C,0FFH  ;long count is position identifier
CALL    UPDATE
JR      SKIP
;
ZERO:   LD      C,0     ;pulse is a logic zero
CALL    UPDATE
JR      SKIP
;
ONE:    LD      C,80H   ;medium count is a logic one
CALL    UPDATE
;
SKIP:   POP     AF      ;return from 'BIT'
RET
;
BACKUP:
;
;----- RSTA interrupt service routine -----
;
;..... INTERRUPTED BY TIMER 1 ONLY .....

```

```

;
PUSH    HL
LD      HL,(TIMRONE)    ;read T1 modulus to reset T1 output
POP     HL
PUSH    AF
LD      (TOSTOP),A      ;the first time through the routine,
LD      A,(BAKFLAG)     ; BAKFLAG=$00
CP      OFFH            ; if BAKFLAG=$FF, we will skip
JR      Z,CALLUP         ;code immediately below to save time
FIRST:  LD      A,OFFH    ;set backup flag
LD      (BAKFLAG),A
;
LD      A,TM1PULS        ;configure T1 for 100Hz pulse generator
LD      (TMR1),A
LD      HL,9971          ;load MODULUS with 9971 to create a
LD      (TIMRONE),HL     ;100Hz timer that produces one active
                        ;low output pulse every 10ms to cause
                        ;an RSTA every 10ms.
                        ; Note: this MODULUS value is determined
                        ;somewhat by trial and error since the
                        ;execution of the program also adds
                        ;to the delay.
LD      (T1STAR),A      ;start the pulse generator
CALLUP: CALL    UPDATE   ;increment the software timer
POP     AF
RET     ;go back and check for IRIG again
;
UPDATE:
;
;-----SOFTWARE TIMERS-----
;
;.....WE SHOULD PASS THROUGH THIS LOOP EVERY 10ms .....
;
LD      A,(HUN)          ;increment the hundredth's count
INC     A                ; on every pass ----
LD      (HUN),A          ;---then see if anything else
CP      10               ; needs to be incremented also
RET     NZ
XOR     A
LD      (HUN),A
;
TENTHS: LD      A,(TEN)
INC     A
LD      (TEN),A
CP      10
JR      NZ,OUTPUT
XOR     A
LD      (TEN),A
;
SECONDS: LD      A,(SECND)    ;read buffer
INC     A                  ;increment seconds count
DAA     ;change to BCD
LD      (SECND),A          ;replace incremented seconds count
CP      60H                ;was count=60?
JR      NZ,OUTPUT          ;if not, just print all values again
XOR     A                  ;if so, reset to zero and then increment
LD      (SECND),A          ; the minutes counter
;
MINUTES: LD      A,(MINUT)
INC     A
DAA
LD      (MINUT),A
CP      60H
JR      NZ,OUTPUT
XOR     A
LD      (MINUT),A
;
HRS:    LD      A,(HOUR)
INC     A
DAA
LD      (HOUR),A
CP      24H
JR      NZ,OUTPUT
XOR     A
LD      (HOUR),A

```

```

;
DAZE:  LD      A,(DAYS)
      INC      A
      DAA
      LD      (DAYS),A
      CP      0                      ;compare to zero since 99+1=00 & <C>=1
      JR      NZ,OUTPUT
      LD      A,(DH)
      INC      A
      LD      (DH),A
;
OUTPUT: LD      A,(OUTENBL)          ;check the flag
      CP      OFFH                  ;output is enabled if FLAG=$FF
      JP      NZ,RETURN
;
      LD      A,(TEN)
      LD      (PRTC),A
      LD      A,(SECND)
      LD      (PORTA),A
      LD      A,(MINUT)
      LD      (PORTB),A
      LD      A,(HOUR)
      LD      (PRTB),A
      LD      A,(DAYS)
      LD      (PRTA),A
      LD      A,(DH)
      LD      (PORTC),A
RETURN: RET
;
      END

```

END

1-87

DTIC