Development system facilitates programming of signal-processor chip

To program the 2920 signal-processor chip for a specific function, Intel's Signal Processor Development System (ELECTRONIC DESIGN, Sept. 1, 1979, p. 27) contains the hardware and software that takes a design from concept to implementation. The system completes an entire design—including testing and debugging—before the 2920 chip is plugged in. Unlike analog design procedures, the system lets the designer:

- Develop circuits faster by eliminating time spent in acquiring parts, in laying out and constructing boards and even in debugging, since the computer helps find the bugs.
- Eliminate bugs by editing and reassembling the program, not relaying out the breadboard.
- Add circuits by adding instructions to the program rather than rebuilding the breadboard.
- Document the design in detail at each step, because all the designing, testing and debugging is done with software.

The 2920 development system is a standard, Intellec Series II, Model 220/230 Microcomputer Development System (MDS). The MDS-220/230 uses an SP20 signal processing support package, comprising a 2920 assembler, a 2920 simulator and a 2920 EPROM programming personality card.

Both software and hardware versions of the simulator are available. The hardware version performs floating-point math routines in hardware via the iSBC-310 high-speed math unit; the software version handles its own floating-point math routines.

Since programs run on the 2920 are stored in 192 words of 24-bit-wide EPROM, software is easily modified, which facilitates real-time testing of prototypes. For program modifications, the previous program is erased from the EPROM; the EPROM programming personality card loads the new one onto the same chip.

The 2920 assembler translates symbolic assembly-language programs into machine-readable code. It accepts the designer's source file and generates a listing file and an object code file. The former lists the source code with corresponding machine code and memory locations. It also includes comments and titles (if any), error and warning diagnostics, the number of RAM and ROM locations used, and a table of user-defined symbols.

Charles Yager, Applications Engineer, Telecommunications Product Marketing, Intel Corp., 3065 Bowers Ave., Santa Clara, CA 95051.

Fig. 1 displays a sample 2920 assembly listing. The object-code file contains the machine-readable code used to program the 2920 for real-time testing or used to load the 2920 simulator for debugging.

The 2920 assembler contains controls and directives that permit the designer to modify the output files by changing the page length or width or both; by including a title which will appear on each page; by suppressing the symbol table; and by ejecting to the top of next page. Since all instructions are assembled at one time (batch assembly), controls and directives to the assembl er must appear either in the command that invokes the assembler or in the source code being translated. If no controls are specified, the assembler defaults to its own set of controls.

Error and warning diagnostics indicate when a mistake is made— not only a software mistake, but also a software-related hardware mistake. For example, timing is crucial when the 2920's Digital-to-Analog Register (DAR) is loaded with a digital value that must be converted to analog. The program must delay the output instruction sufficiently to accommodate the finite risetime of the amplifier in the 2920's d/a converter. If the delay given is insufficient, the assembler issues a warning.

The 2920 assembler is a one-pass design. Since the 2920 maintains a constant sample rate and since the rate depends on the number of instructions executed, no jumps are allowed except at the end of a program when the program counter is reset. Therefore, the assembler need not devote a separate pass to determine the address of the labels which would typically be found in jump instructions.

The assembler passes through the code once, using two location counters: one for the instructions and labels, and another for the user symbols representing the RAM locations. Even though the 2920 prohibits jumps, the assembler does allow labels for symbolic debugging in the simulator.

The assembler recognizes two types of symbols: reserved and user-defined. Reserved symbols are mnemonic 2920 instructions, addressable constants, assembler controls and assembler directives. User symbols are locations of RAM data and of labeled instructions.

When the assembler first encounters a user symbol, it assigns the value of the appropriate location counter to the symbol and then increments the counter. The assembler allows the designer to use the EQU statement to make two symbols equivalent. This use of EQU can be advantageous when a RAM value is used as a temporary memory location and has a different meaning in different program segments.

The assembler recognizes a control word called DEBUG, which generates additional object code containing the user-defined symbols. The EPROM programmer ignores this additional object code but the simulator recognizes it. Once the code is loaded into the simulator, a program can be debugged using the designer's symbols in the assembly listing.

The 2920 simulator (SM2920) simulates the program as it executes in the processor. Using the simulated program, the designer has access to the registers, inputs and outputs, clock, memory locations and other points of interest in the processor, to analyze the performance and problems of the program (Fig. 2).

Although it simulates the functions of the 2920, the SM2920 executes much more slowly than the 2920. The 2920 is a real-time signal processor, but the simulator is a logical-time signal processor.

Without the simulator, many points of interest in the 2920 would be inaccessible during debugging. For example, three pins on the 2920 are used for hardware debugging: End of Program (EOP), Overflow (OF) and the instruction cycle ($\overline{\text{CCLK}}$). With these pins, a designer can tell the instruction at which a register overflows, an output occurs or the end-of-program occurs. However, typical bugs—forgetting to clear a register or inadvertently clearing a register—would be invisible to the user without the simulator. Also, it would be impossible with the pins alone to examine registers and RAM locations at any point in time.

The simulator operates in two modes: interrogation and simulation. The interrogation mode occurs whenever the SM2920 is waiting for a command and not simulating the program execution. It can be used to prepare the system for a new operation or to investigate the results of the last operation.

In the second mode, the 2920 program residing in memory is simulated and trace data are collected and displayed. Simulation can start from time equals zero and program counter equals zero; or from the point at which the last simulation ended; or from some user-specified initial time and counter conditions. During program execution, a break in simulation may be specified according to the conditions of simulation. When a breakpoint is encountered, the SM2920 module reverts to the interrogation mode.

Any simulation with the SM2920 follows much the same sequence of steps illustrated in Fig. 3. On entering the interrogation mode for the first time, the user prepares the system for simulation by setting the sample rate, defining symbols, setting simulation breakpoints and trace qualifiers, and specifying what items are to be traced. Thereafter, using the input facility, the designer specifies the input signals to be simulated as functions of t ime or other variables. Once simulation begins, the trace data are displayed as they are collected.

When simulation halts (entering the interrogation mode), the designer can redisplay the trace data into a trace buffer. SM2920 commands position the trace buffer pointer to the desired information and display

```
ISIS-II 2920 ASSEMBLER X102
ASSEMBLER INVOKED BY: AS2920 SRG
SWEEP RATE GENERATOR
LINE LOC  OBJECT   SOURCE      STATEMENT
 1                 $DEBUG        TITLE ('SWEEP RATE GENERATOR')
 2
 3
 4    0   488AEF   CONSTANT:   LDA  S1,   KP5,  R00         ;DEVELOP A CONSTANT
 5    1   408A8C               ADD  S1,   KP1,  R05         ;FOR A 1Hz SWEEP RATE
 6    2   40008F               LDA  S1,   S1,   R13
 7    3   4000F8        SRG:   SUB  H1,   S1,   R00         ;BEGIN THE SAWTOOTH
 8    4   404CEF               LDA  DAR,  H1,   R00         ;WAVEFORM GENERATOR
 9    5   7882DD               ADD  H1,   KP4,  L01,  CNDS  ;IF H1 IS < 0 RESET
10                                                          ;TO ONE
SYMBOL:                  VALUE:
CONSTANT                    0
S1                          0
SRG                         3
H1                          1
ASSEMBLY COMPLETE
ERRORS    = 0
WARNINGS  = 0
RAMSIZE   = 2
ROMSIZE   = 6
```

1. In the assembly-language source code for the 2920, the six statements shown configure the 2920 as a sawtooth waveform generator with a 1-Hz sweep rate. To aid the designer, the code includes a list of symbols along with their absolute locations and the number of errors and of warnings found by the assembler during its translation of the program.

A demonstration of the 2920 simulator

Simulation of a sweep-rate generator (SRG) program demonstrates some of the testing capabilities of the 2920 development systems. The sweep-rate generator, a sawtooth wave oscillator, is one of the functional blocks in the 2920 spectrum analyzer (ELECTRONIC DESIGN, Nov. 8, 1979, p. 70).

In the spectrum analyzer, the SRG provides the horizontal sweep output for an oscilloscope, and creates an input to a voltage controlled oscillator. The oscillator produces a linear frequency sweep as a function of time.

To demonstrate the simplicity in changing oscillator frequency, the program listing has been modified slightly: The period of the SRG has been changed from 1 s (a length that could generate too many traces) to 488 μs (which corresponds to 2048 Hz). The right shift in instruction 2 of the SRG program is changed to RO2. This multiples the original constant by 2^{11}, which in turn multiples the frequency by 2^{11} or 2048. With this change, the designer tests the program as follows:

```
-SM2920
*LOAD SRG.HEX   ;

*ROM 0 TO 5
ROM 000 = LDA .S1,KP5,R00,NOP
ROM 001 = ADD .S1,KP1,R05,NOP
ROM 002 = LDA .S1,.S1,R13,NOP
ROM 003 = SUB .H1,.S1,R00,NOP
ROM 004 = LDA DAR,.H1,R00,NOP
ROM 005 = ADD .H1,KP4,L01,CNDS

*SYMBOL
.S1= 0.00000000
.H1= 1.00000000
.CONSTANT= 0.00000000
.SRG= 3.0000000E+0
```

This command loads the object code and the symbol table. The symbol table and the object code are dumped to verify that they have been loaded correctly.

The following code modifies the program in the simulator so that the frequency is 2048 Hz instead of 1 Hz; it also adds an End of Program (EOP) command.

```
*ROM 2=LDA .S1,.S1,R2
*ROM 0 TO 5
ROM 000 = LDA .S1,KP5,R00,NOP
ROM 001 = ADD .S1,KP1,R05,NOP
ROM 002 = LDA .S1,.S1,R02,NOP
ROM 003 = SUB .H1,.S1,R00,NOP
ROM 004 = LDA DAR,.H1,R00,NOP
ROM 005 = ADD .H1,KP4,L01,CNDS
*ROM 188=EOP
```

The next command, TPROG sets the sample period. The period set here corresponds to a sampling frequency of 13,020 Hz. This frequency equals the 2920 clock frequency of 10 MHz plus a full program of 192 instructions.

```
*TPROG=1/13020
```

The TRACE command specifies the items to be traced. For the first part of the SRG test, the TRACE allows the designer to determine if the constant generated in the program is the one desired. The program counter and the RAM location containing the constant are traced.

```
*TRACE=PC,RAM  .S1
```

Next, the QUALIFIER parameter indicates the conditions to be met to collect a trace. ALWAYS means "collect a trace every program instruction." The SIZE command gives the number of simulated instructions.

```
*QUALIFIER
QUALIFIER = ALWAYS
*SIZE=192
```

The simulation to test the constant begins. Because the constant is completely generated after the first three instructions, a BREAKPOINT is set to stop simulation at that point.

```
*BREAKPOINT=COUNT=3
*SIMULATE FROM 0

       PC                   RAM 0
SIMULATION BEGUN
 1.00000000            0.62500000
 2.0000000E+0          0.62890625
 3.0000000E+0          0.15722656
SIMULATION TERMINATED
*EVALUATE RAM .S1
     0.0010100001000000000000000B
                          0.15722656D
                            0.284000H
```

The simulator has verified that the 2920-generated constant is correct. The next test determines the period, frequency and waveshape of the sawtooth wave oscillator. TRACE is changed so that the time, DAR and the oscillator's RAM location can be observed. QUALIFIER is changed so that a trace is generated once per program pass. A BREAKPOINT is set to stop the simulation after two cycles of the wave are traced. Also the oscillator's RAM location is initialized to one. The simulation begins at time equals zero, when the program counter equals zero.

```
      *TRACE
      TRACE = PC,RAM 0
      *TRACE=T,DAR,RAM .H1
      *Q
      QUALIFIER = ALWAYS
      *Q=PC=0
      *RAM .H1=ONE
      *B
      BREAKPOINT = COUNT=3
      *B=T>.001

*SIMULATE FROM 0
    T                      DAR              RAM 1
SIMULATION BEGUN
  0.00007680          0.83984375       0.84277334
  0.00015361          0.68359375       0.68554683
  0.00023041          0.52734375       0.52832026
  0.00030722          0.36718750       0.37109370
  0.00038402          0.21093750       0.21386714
  0.00046083          0.05468750       0.05664056
  0.00053763         -0.10156250       0.89941396
  0.00061444          0.73828125       0.74218745
  0.00069124          0.58203125       0.58496089
  0.00076805          0.42578125       0.42773433
  0.00084485          0.26953125       0.27050776
  0.00092166          0.10937500       0.11328119
  0.00099846         -0.04687500       0.95605459
SIMULATION TERMINATED
```

Using linear interpolation, the frequency is determined from the time to be 2048 Hz, as expected. Judging from RAM location 1, the waveform is a sawtooth. The results of the complete simulation test show that the sawtooth oscillator is working correctly.

one, several or all the entries in the buffer.

Also, the designer can examine and change 2920 memory locations or registers, I/O ports or SM2920 psuedo-registers, to acquire valuable information on program operation at termination. Altering data or register values reveals their affect on the next simulation; changes can be patched into the program code itself. The designer can display and change symbolic values in the symbol table and set breakpoint and trace qualifier conditions.

By alternating between interrogation and simulation, the designer can debug and check every aspect of a program's operation before loading it into the EPROM of the 2920. At the end of the simulation session, the debuggedcode can be saved on an ISIS-II diskette file, using the SAVE command; part or all of the simulation session can be saved on a diskette file for future reference.

One example of a simulation command involves the designer issuing the command statement:

```
SIMULATE FROM 0 TILL RAM X> .3
            OR OVF = 1
```

This command simulates the program starting at time zero, and with the program counter at zero, until the RAM location symbolically represented by X is greater than 0.3 or until a register overflow occurs. Unlimited logical operands are allowed. The hierarchy of logical and arithmetic operations follows Fortran rules.

Variables in the simulator are specified as read or read-write, as shown in Fig. 2, during interaction between the designer and the simulator. To display or replace a variable like DAR, the designer types DAR (CR), and the simulator replies DAR-.00390625. To replace the variable's value, the designer types in DAR-5; DAR now equals 0.5. This operation can be completed with any read-write register in the simulator.

One of the more important and useful commands in the simulator is EVALUATE. It evaluates an expression and displays the result in decimal, hexadecimal and binary.For example, solving the equation SIN (20 * LOG (2.3))) requires the input:

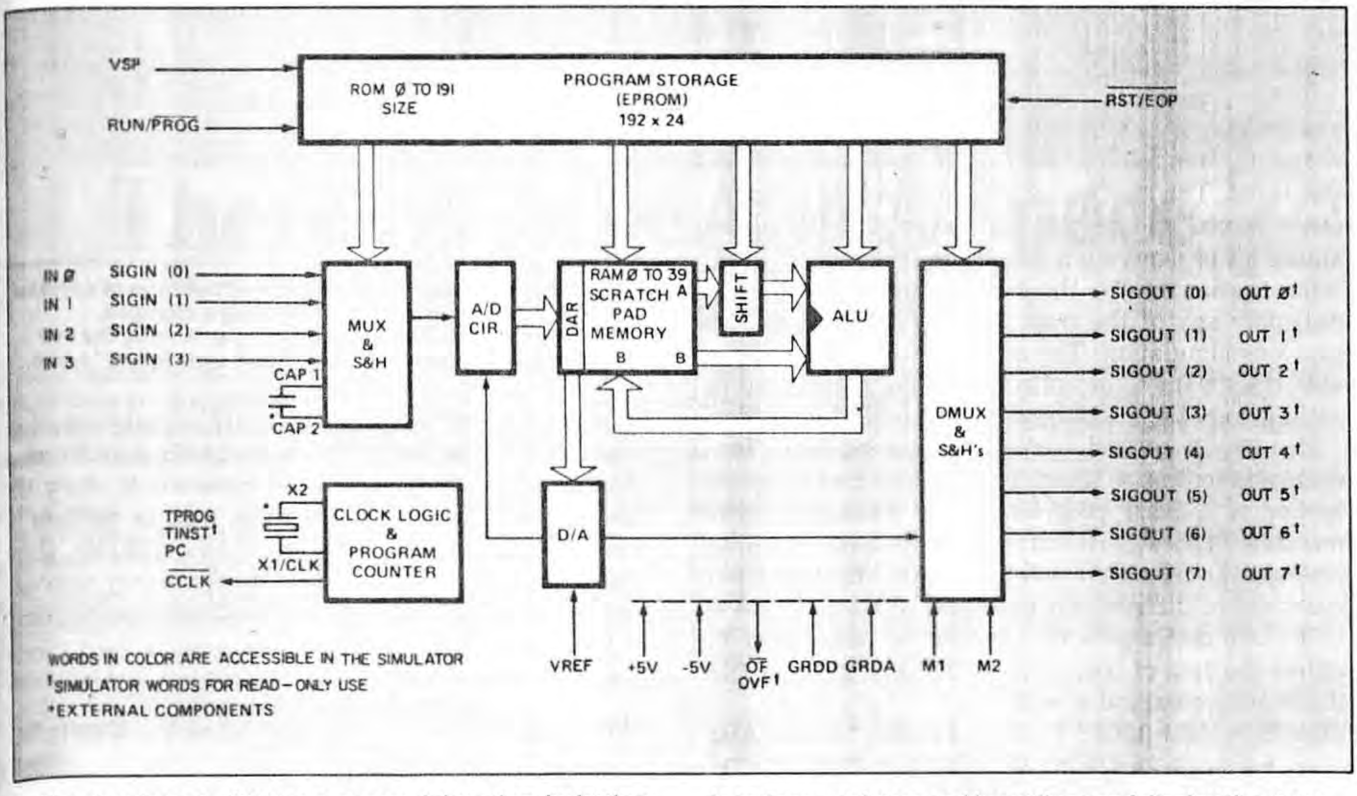
```
EVALUATE SIN (20 * (LOG (2.3)))
```

On the next line, the simulator displays the answer in the three different bases, which is an advantage in converting a number or result to another base. The syntax of the input expression is the same as that for a Fortran arithmetic expression.

With the EVALUATE command, the simulator becomes an on-the-spot calculator. Although the EVALUATE command is implicitly incorporated into many SM2920 instructions, it can be explicity incorporated into user's instructions.

Another important capability of the 2920 simulator is its input facility, which enables the designer to specify input functions on any or all of the four, 2920-multiplexed, input pins. Therefore, the designer can input test signals just like an analog designer would, to measure the system's performance.

Whenever the simulator encounters the command code for an input function (e.g., IN2 or IN0, which indicate which channel should be sampled), it evaluates the function using the current values for the variables used. In addition, the following symbolic constants can appear in the function: PI specifies 3.1415928; HPI means half of pi; TPI means twice pi; and ONE means 1. Certain built-in functions are also



2. The simulator facilitates program debugging, by letting the designer examine and change the contents of memory locations, registers and input lines and display the state of output lines from the demultiplexer.

available: ABS means absolute value, SQR means the square root of a number or variable, SAW means a sawtooth wave, SQW means a square wave and LOG is the natural log. For example:

INO – SIN (TPI * T * 500)

which indicates a sine wave at 500 Hz;

IN1 = SAW (1000 * T * TPI) * EXP (–T)

which indicates a descreasing, 1-kHz, sawtooth wave;

IN2 = SIN (TPI * T) * (10 + 1500 * T)

which indicates a sweeping sine wave; and

IN3 = IF 0.1 <T AND T<0.15 THEN ONE ELSE 0

which gives a pulse having a duration of 0.05 s.

The IF/THEN/ELSE construction specifies the conditions to be evaluated. If the conditions between the word IF and the word THEN are met, the value after the ELSE is used.

Finally, as previously described, SIMULATE initiates operation of the SM2920, and BREAKPOINT stops simulation. These two commands change the simulator's mode of operation: On executing SIMULATE, the simulator changes from the interrogation to the simulation mode; when the BREAKPOINT condition occurs, the simulator changes back to the interrogation mode. Halting conditions maybe specified in several ways, depending on whether simulation is ONCE or condition AND/OR condition..., or FOREVER.

Simulating ONCE is single-step simulation. The simulator executes one instruction, halts, and reverts back to the interrogation mode. Condition AND/OR condition..., simulates until certain conditions (logical and/or arithmetic) are met. For example:

SIMULATE TILL SQR (LOG(OUTO)) >.3 AND T>.001

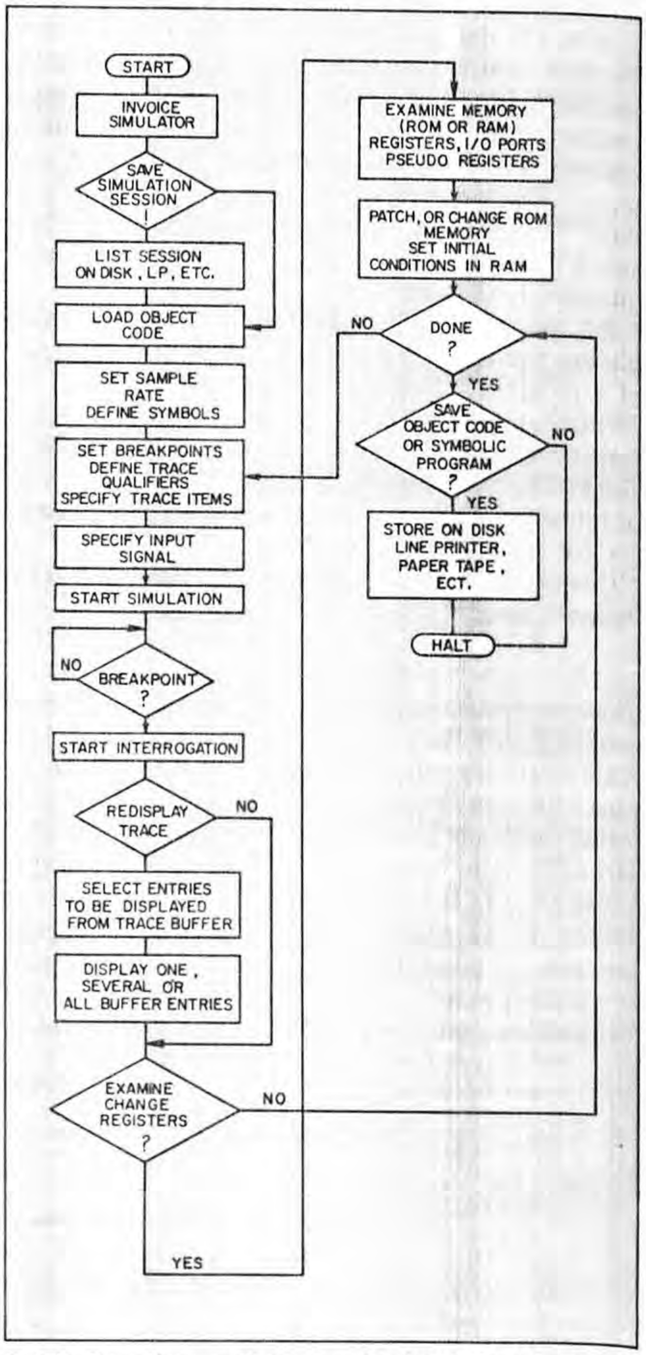
means halt simulation when the square root of the natural log of OUTO is greater than 0.3 and time is greater than 1 ms. Finally, the SM2920 can simulate FOREVER, i.e., until the system is turned off or the escape key, which aborts simulation, is hit.

During simulation, trace data are collected and displayed. The TRACE control commands simultaneously record and display the value of any register, status bit or expression in a trace data buffer. In the interrogation mode, the TRACE control commands redisplay any of the trace data collected during the previous simulation. The designer can enable or disable trace data to be collected during simulation by setting conditions which qualify or inhibit collection.

The TRACE control commands can be used, for example, to test a filter. To find the frequency response of a filter program in the simulated ROM memory, the designer first specifies an input sweeping sine wave. This wave sweeps across the frequency spectrum of interest; for example, 10 Hz at a rate of 1500 Hz/s and a gain of unity in the pass band. To collect the data of frequency (in Hz) and gain (in dB), the TRACE command would be:

TRACE = (10 + 1500 * T), 20 * (.4342945 * (LOG (ABS (OUTO/ONE)))).

Multiplication by the constant 0.4342945 converts the natural log into log base 10.


START
INVOICE SIMULATOR
SAVE SIMULATION SESSION ?
LIST SESSION ON DISK, LP, ETC.
LOAD OBJECT CODE
SET SAMPLE RATE DEFINE SYMBOLS
SET BREAKPOINTS DEFINE TRACE QUALIFIERS SPECIFY TRACE ITEMS
SPECIFY INPUT SIGNAL
START SIMULATION
NO
BREAKPOINT ?
START INTERROGATION
REDISPLAY TRACE
NO
SELECT ENTRIES TO BE DISPLAYED FROM TRACE BUFFER
DISPLAY ONE, SEVERAL OR ALL BUFFER ENTRIES
EXAMINE CHANGE REGISTERS ?
NO
YES
EXAMINE MEMORY (ROM OR RAM) REGISTERS, I/O PORTS PSEUDO REGISTERS
PATCH, OR CHANGE ROM MEMORY SET INITIAL CONDITIONS IN RAM
NO
DONE ?
YES
SAVE OBJECT CODE OR SYMBOLIC PROGRAM ?
NO
YES
STORE ON DISK LINE PRINTER, PAPER TAPE, ECT.
HALT

3. The flow diagram illustrates the simulation of a typical, assembled, 2920 program. After design changes, simulation is repeated, as implied by arranging the last decision block to return to the "start-simulation" block.

If the simulation began at time (T) equals zero, then the input signal would begin at 10 Hz and increase linearly. The displayed traced data would show the frequency in Hz in column 1 and gain in decibels in column 2. If the data were saved on a diskette, there would be enough points to plot the frequency response of the filter and to determine if the frequency response were correct for the system. ■■

How useful?	**Circle No.**
Immediate design application	**556**
Within the next year	**557**
Not applicable	**558**