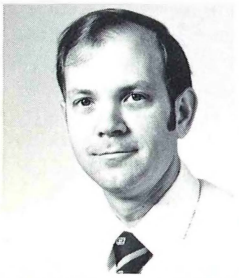
TECHNICAL INFORMATION
FOR TEKTRONIX CUSTOMERS

VOLUME 12 NUMBER 4
DECEMBER 1980

Tekscope

Tektronix®
COMMITTED TO EXCELLENCE

A User-Programmable Logic Analyzer for Microprocessor-Based Design



Mike Reiney, project manager for the 7D02, joined Tek in 1971 following receipt of his Bachelor's and Master's degrees in electrical engineering from Rice University. Mike was involved in the design of several TM 500 Series products before joining the Logic Analyzer design group. In his spare time Mike enjoys skiing, sailing, and motorcycling.

The logic analyzer is the basic tool for designing and debugging digital circuitry. The characteristics of a logic analyzer designed for working with random-logic systems differ considerably from those required for working with microprocessor-based systems. Random-logic analyzers usually emphasize sampling speed and depth of memory, while analyzers suitable for working with microprocessor systems feature a large number of input channels and extremely flexible triggering.

The new Tektronix 7D02 Logic Analyzer is primarily a tool for designing, debugging, and troubleshooting microprocessor systems. It can support both 8-bit and 16-bit microprocessors.

The 7D02 can acquire up to 28 channels (44 optionally) of synchronous data, and an additional eight channels of synchronous data or eight channels of asynchronous timing or state information are available through a timing option, for a total of 52 channels. With the timing option installed, the 7D02 is one of the most powerful, yet easy to use, tools available for working with microprocessor systems.

The 7D02 can be easily programmed (through a front-page keypad) to follow the complex sequences of events occurring in the system under test. Program displays are dynamic and interactive, with only necessary prompting on-screen at any given time.

As the programming cursor is moved, new prompting occurs. Menus default to reasonable values to simplify input, and the 7D02 assumes the most logical program and supplies intermediate program steps. Displayed data can be formatted in the basic radices and mnemonics pertinent to the microprocessor under test.

A series of personality modules adapt the 7D02 to the clock and bus characteristics of individual microprocessors. The basic 7D02 contains four word recognizers, two general-purpose counters, a user-configurable clock, data-qualification circuitry, programmable state machine, and three memories.

A programmable state machine

The programmable state machine is the key to the 7D02's flexibility. It provides two equally powerful capabilities — generation of a trigger algorithm that tracks the complex, convoluted program flow to trigger exactly where the user requires, and data qualification that determines precisely which data will be stored in the acquisition memory. In the 7D02, the qualify command works exactly like the trigger command. The user can employ these two commands to discard the bus transactions of no consequence and to store only that data which is of interest in solving the problem.

The user can program the 7D02's state machine for any one of four states, with each state representing a user-determined output that is the result of a user-determined input. The state-machine input consists of lines from the four word recognizers and two counters; the output consists of individual lines to the main trigger, timing option trigger, data qualifier, and four lines to control the two counters.

The inputs from the word recognizers and counters are called ***events,*** and the outputs are called ***commands.*** When programming the 7D02, the user can link any event or combination of events to any command or combination of commands. The state machine executes in real time with the system-under-test and can enter any of its states in any order, any number of times. These capabilities allow the user to program the 7D02 to follow the complex sequences encountered in microprocessor-based systems.

Block-structured programming language

The 7D02 programs with a block-structured language. The programming language uses four tests, each constructed using an IF-THEN-ELSE syntax. This syntax makes the execution of a command conditional on the occurrence of an event. Events are keyed in following an IF or an OR IF prompt, and commands are keyed in following a THEN DO or ELSE DO prompt. The user can have as many OR IF — THEN DO clause pairs as necessary (subject to memory limitations), but there can be only one ELSE clause in each test.

Only one test may be active at a time. The user moves from one test to another by executing a GO TO command. The following example shows a program containing two tests and illustrates moving from one test to another. This operation represents a two-level sequential trigger.

```
  TEST 1
1 IF
1  WORD RECOGNIZER #1
1  DATA = XX
1  ADDRESS = 4325
1  /NMI = X    /IRQ = X    FETCH = X    R/W = X
1  BA = X    INVAL OP = X    EXT TRIG IN = X
1  TIMING WR = X
1 THEN DO
1  GOTO 2
  END TEST1
  TEST 2
2 IF
2  WORD RECOGNIZER #2
2  DATA = XX
2  ADDRESS = 694F
2  /NMI = X    IRQ = X    FETCH = X    R/W = X
2  BA = X    INVAL OP = X    EXT TRIG IN = X
2  TIMING WR = X
2 THEN DO
2  TRIGGER 0-MAIN
2     0-BEFORE DATA
2     0-SYSTEM UNDER TEST CONT.
2     0-STANDARD CLOCK QUAL.
  END TEST 2
```

Pressing the 7D02 START button initiates TEST 1. When address 4325 is detected, the event in TEST 1 becomes TRUE and the GO TO (to TEST 2) is executed, deactivating TEST 1 and activating TEST 2. If address 694F is detected, the main acquisition memory will be triggered and data will be stored. Note that if address 4325 occurs again it will be ignored because TEST 1 is inactive. Entering this program required only three keystrokes plus filling in the field values.

Solving a practical problem

Now, let's consider a practical problem. Assume there is a location, OUTBUF, that is the character buffer for a computer line-printer. This location is only written-to from subroutine OUTCHAR. Unfortunately, data on the printer is not always what we expect. The problem is to determine whether location OUTBUF is being written-to from some other section of the code.

With a traditional logic analyzer, we could trigger on OUTBUF and examine the data result to see if the access was legitimate. We may have to examine many legitimate accesses before finding the problem access.

Using the 7D02 we can locate the illegitimate access quickly and easily. The program is shown in figure 2. The two events in TEST 1 are the detection of the addresses for OUTBUF and OUTCHAR. All events in a test are evaluated simultaneously. If the location OUTBUF is written to, the 7D02 will trigger. If the beginning of a subroutine OUTCHAR is detected, the 7D02 will progress to TEST 2. As there is no trigger in TEST 2, the 7D02 cannot trigger during TEST 2. When the end of subroutine OUTCHAR occurs, TEST 2 ends, and the 7D02 goes back to TEST 1. Thus, we see that the 7D02 will trigger and display data only when location OUTBUF is written to, and only if subroutine OUTCHAR is not running.

Now, let's assume that OUTBUF was never written-to from an improper location. If the error occurs at least once every several minutes (a reasonable time to wait for a trigger), the qualify command can be used to verify the accuracy of the data being written to OUTBUF. Figure 3 shows the addition to the program required to instruct the 7D02 to acquire only data written to OUTBUF. If the printer malfunctions, the user can manually stop this program. The contents of the trace memory can then be compared to the printed text by putting the 7D02 in the ASCII format mode.

If the error occurs very infrequently, or if the result of the previous exercise shows that correct data was being written to OUTBUF, we must employ a less direct means to solve the problem.

The two counters in the 7D02 can be used either to count discrete occurrences, or as timers. We can employ the counters to insert a fixed time period as a part of the

```
    TEST 1
1   IF
1    WORD RECOGNIZER # 1
1    DATA=XX
1    ADDRESS=051B
1    IO/M=X   INRQ=X   FETCH=X   R/W=0
1    INACK=X   HOLD=X   EXT TRIG   IN=X
1    TIMING WR=X
1   THEN DO
1    TRIGGER 0-MAIN
1       0-BEFORE DATA
1       0-SYSTEM UNDER TEST CONT
1       0-STANDARD CLOCK QUAL
1   OR IF
1    WORD RECOGNIZER # 2
1    DATA=XX
1    ADDRESS=9721
1    IO/M=X   INRQ=X   FETCH=1   R/W=X
1    INACK=X   HOLD=X   EXT TRIG IN=X
1    TIMING WR=X
1   THEN DO
1    GO TO 2
    END TEST 1
    TEST 2
2   IF
2    WORD RECOGNIZER # 3
2    DATA=C9
2    ADDRESS=XXXX
2    IO/M=X   INRQ=X   FETCH=1   R/W=X
2    INACK=X   HOLD=X   EXT TRIG IN=X
2    TIMING WR=X
2   THEN DO
2    GO TO 1
    END TEST 2
```

Fig. 2. This 7D02 program will trigger when location OUTBUF is written to, unless subroutine OUTCHAR is running at the same time. The PM104 (8085) personality module is being used. Word recognizer 1 is TRUE when OUTBUF (015B) is written to. Word recognizer 2 is TRUE on the first instruction of OUTCHAR (9721). Word recognizer 3 is TRUE when a RET instruction is executed indicating the end of OUTCHAR. Entering this extensive program required only eight keystrokes plus filling in the field values.

trigger, a feature which can be very useful in some instances, as the following example shows.

Hypothesizing that there is a timing error in the printer, we set up an experiment to determine if the setup time specification on the print head is being met. We start a counter when the character is written to OUTBUF. If the print hammer is actuated within 10 milliseconds, the timing specification is being violated and the 7D02 will trigger. The 7D02 program is shown in figure 4.

```
    QUALIFY
Q    STORE ONLY ON
Q    WORD RECOGNIZER # 1
Q    DATA=XX
Q    ADDRESS=051B
Q    IO/M=XX   INRQ=X   FETCH=1   R/W=0
Q    INACK=X   HOLD=X   EXT TRIG IN=X
Q    TIMING WR=X
Q   END QUALIFY
```

Fig. 3. Program additions required to qualify any write to 051B.

In this example, the external trigger line is connected to the hammer-driver signal and is defined in word recognizer 2 as a TRUE. Word recognizer 1 detects the write to OUTBUF.

TEST 1 waits for the write to OUTBUF. When this happens, the counter is started and TEST 2 is activated. In TEST 2 there is a race. If the hammer driver is actuated first,

```
   TEST 1
1  IF
1   WORD RECOGNIZER # 1
1   DATA=XX
1   ADDRESS=051B
1   IO/M=X   INRQ=X   FETCH=X   R/W=0
1   INACK=X   HOLD=X   EXT TRIG IN=X
1   TIMING WR=X
1  THEN DO
1   COUNTER # 1 2 MS
1     2 RESET AND RUN
1   GO TO 2
   END TEST 1
   TEST 2
2  IF
2   WORD RECOGNIZER # 2
2   DATA=XX
2   ADDRESS=XXXX
2   IO/M=X   INRQ=X   FETCH=X   R/W=X
2   INACK=X   HOLD=X   EXT TRIG IN=1
2   TIMING WR=X
2  THEN DO
2   TRIGGER 0 MAIN
2   2 AFTER DATA
2   0 SYSTEM UNDER TEST CONT 1
2   0 STANDARD CLOCK QUAL
2  OR IF
2   COUNTER # 1=00010 2 MS
2  THEN DO
2   GO TO 1
   END TEST 2
```

Fig. 4. Program to check timing margins. If OUTBUF (051B) is written to and the hammer driver is actuated (as indicated by the TRUE and EXT TRIG IN) before COUNTER #2 reaches 10 milliseconds, we know the timing specification has been violated. This program can be set up using only seven keystrokes plus filling in the field values.

the 7D02 triggers. If the timer runs out, sufficient setup time has elapsed and TEST 1 is actuated again.

It is possible for the 7D02 to loop between TEST 1 and TEST 2 millions of times without triggering. The 7D02 will trigger only when the timing constraint has been violated.

The 7D02 also could be easily programmed to check that every write to OUTBUF is followed by only one actuation of the print hammer.

The timing option

The triggering capability of the 7D02 makes it an ideal tool for integrating the hardware and software in a microprocessor-based system. However, conditions often require us to analyze the random-logic circuits associated with the microprocessor. The timing option available for the 7D02 provides this capability. With the timing option installed, the 7D02 is essentially two logic analyzers in one — a 52-channel synchronous analyzer (with expansion option), or a 44-channel synchronous analyzer plus an 8-channel asynchronous logic analyzer.

The timing option uses the 8-channel P6451 Logic Probe to acquire data. The timing option has its own 255 x 8-bit acquisition memory, 255 x 8-bit glitch memory, 8-channel word recognizer (the external trigger input provides a ninth nonstored channel), and an internal clock. The sampling rate is programmable over a range of 20 nanoseconds to 5 milliseconds. A programmable 0 to 300 nanosecond filter is provided for the word recognizer output.

You can establish the trigger relationship of the main and timing option sections in any manner you choose. For example, either or both sections can be triggered or armed from either or both sections. This extreme trigger versatility is useful for debugging the interaction between a microprocessor and its peripheral hardware.

Some design considerations

A simplified block diagram of the 7D02 is shown in figure 5. The ability to completely program the triggering and qualification algorithms requires the decision blocks (such as word recognizers and the state machine) to be implemented in random-access-memory (RAM), which places considerable time constraints on the real-time acquisition system. At the maximum rate of 10 megahertz, the word-recognizer RAMs require almost a full clock cycle for storage. Likewise, the counter subsystem and state machine require another clock cycle. The 7D02 uses a pipeline decision process for delaying data flow to allow time for producing complex signals such as triggers and qualifiers. The pipeline consists of two sets of data latches and the 256 x 44-bit acquisition memory. Words are consecutively clocked into the pipeline latches by the state clock signal; the same clock edge writes data into the acquisition RAM. If the qualify signal from the state machine is TRUE, then the RAM address counter increments. Otherwise, the next state clock overwrites the same memory cell and the word is, effectively, not stored.

The state machine is RAM-based also. The state-machine latch stores the signals from the word recognizers, the state-feedback bits from the RAM, and the feedback bits from the two counters. The latched data (which represents events in the user language) addresses a location in RAM that contains the data appropriate for the next operation. The data outputs from the state machine are the logic analyzer control lines and represent the commands issued by the user language.

The dual-counter subsystem is implemented with direct-memory-access controller ICs to conserve space and power. Under state-machine control, the glitchless start/stop allows resolution to be increased using time-interval-averaging techniques. One counter may be used as the loop counter for the averaged measurement, which is accumulated in the second counter.

Designing plug-in personality modules — to allow the 7D02 to accommodate many different microprocessors without dismantling the instrument to change personalities — presented an interesting challenge. To achieve the necessary flexibility, a programmable clock synthesizer is used and appropriate firmware is included in the personality module.

The 7D02 clock synthesizer can shift or divide the input clock by up to four clock cycles or times to accommodate multiphase clocks. A programmable external synchronizer (Esync) locks the 7D02 to the system under test. A programmable wait-state generator tracks microprocessor wait states.

The programmable clock shifter is a universal shift register (see figure 6). It is loaded by the Esync signal. The wait signal asserts the hold line to suspend shifting. The clock divider adds feedback around the shifter. The Esync and wait signals are generated from the information provided by the hardware and by the firmware in the personality module.

The personality module

The personality module contains input buffers, bus demultiplexers, special control generators, and firmware. The personality module firmware provides information to program the 7D02 clock synthesizer and clock qualifier. It also provides information

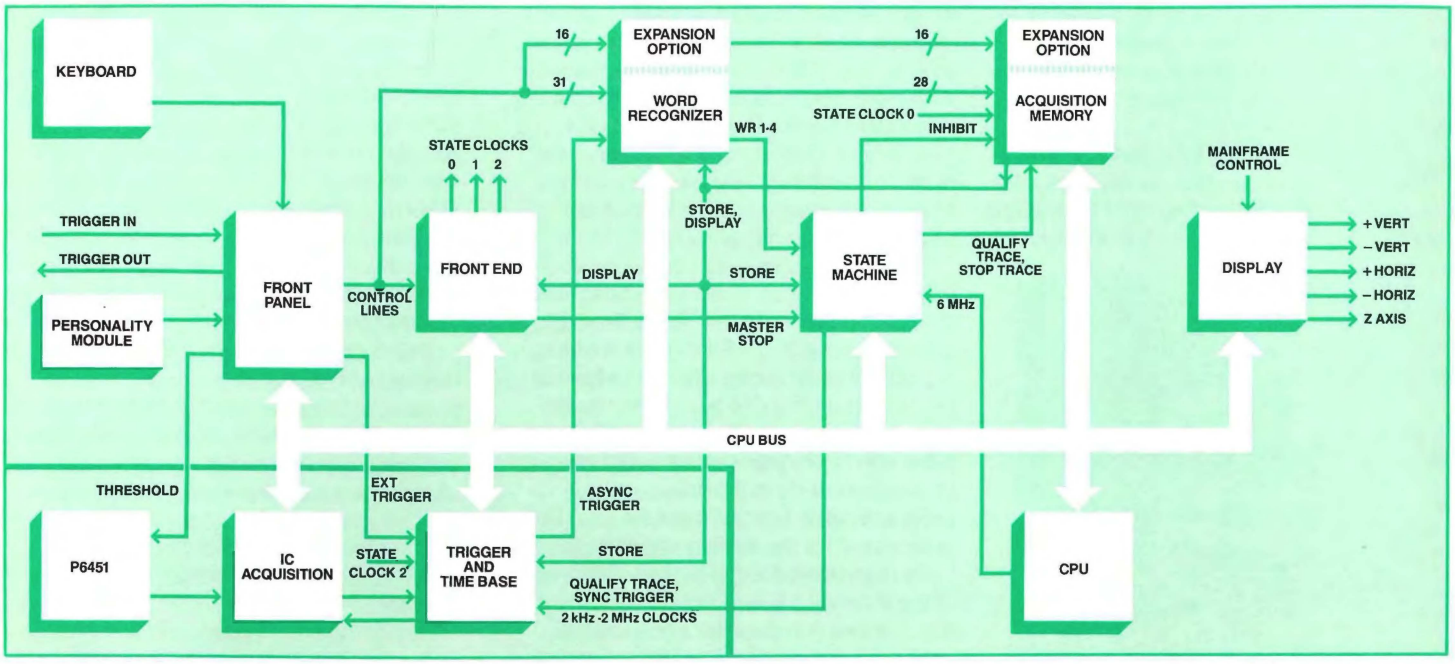


Fig. 5. Simplified block diagram of the 7D02 system. The expansion option allows the 7D02 to operate with 16-bit microprocessors and other systems, and extends the address lines to 24 and the data lines to 16. The timing option consists of the P6451, IC Acquisition, and Trigger and Time Base blocks.

to format the 7D02 input and output displays into the radices and mnemonics of the microprocessor under test.

To perform this format function, a special interpreter was developed. In display mode, data from the system is inhibited and the personality ROM is accessed via the acquisition bus. Commands are fetched from the personality module and executed by the interpreter. This approach allows extreme flexibility in designing future personality modules, saves coding space, and simplifies coding and debugging.

Self-test and diagnostic capabilities

An important consideration in using complex instrumentation is how to determine if it is working properly. The 7D02 has three levels of diagnostics to assist in this test.

At power-up, the 7D02 checks internal subsystems to the extent possible without having known data input. If problems exist, descriptive messages are displayed. These are keyed to troubleshooting trees in the 7D02 service manual.

The user can call up the Diagnostic Monitor — Module Test, which employs service test-generators contained in the personality modules. To verify all data paths through the system, the user plugs the acquisition probe into the service test socket, and the 7D02 acquires known input data and performs a checksum.

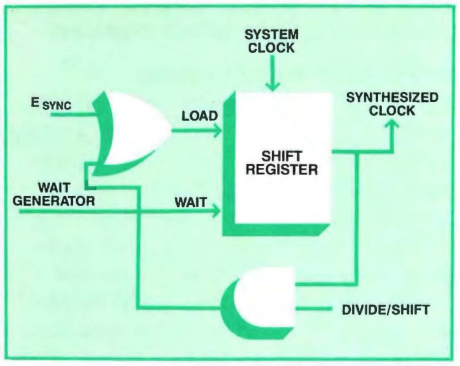


Fig. 6. Simplified block diagram of the 7D02's synthesized clock generator. Personality module firmware provides information to program the synthesizer and qualifier.

Suspect subsystems can be analyzed by using the Signature Exerciser Mode which generates test patterns that can be verified using a signature analyzer.

The diagnostic modes consume eight kilobytes of ROM (16% of the total firmware) and provide excellent diagnostic coverage of the 7D02 system.

Acknowledgements

Many people are involved in a project as extensive as the 7D02. While it isn't feasible to acknowledge each of them, I would like to express my thanks to all who worked so diligently on the project. Special thanks go to Dennis Glasby for the original 7D02 concept. Robin Teitzel was hardware project leader and Dave Moser performed a similar function for firmware. Paul Dittman, Steven Den Beste, Chris Benenati, and Bruce Ableidinger designed and implemented the firmware. The hardware team consisted of Vicky Tuite, Doug Boyce and Keith Taylor. Diagnostics are the work of Bob Heath, and the personality modules were the responsibility of Richard Jones. ■