A PC-Based Digital Storage Oscilloscope and Logic Analyzer

Build this handy multipurpose instrument for your test bench.

By Larry Cicchinelli, K3PTO

The device described in this article is a combination digital storage oscilloscope (DSO) and logic analyzer (LA). It interfaces to a PC through a parallel (printer) port as shown in Table 1. It is built using several printed-circuit boards. The system is designed to allow for six input cards. Each card can be either eight logic-analyzer channels or a single analog channel (8-bit resolution).

A previous article (*QEX*, Jan 2000) describes an eight-channel LA that I built and have used quite a bit. The knowledge and experience gained in its development were instrumental in the development of this project. The original intent was to build a DSO only; however, it soon became obvious that most of the design was also useful for implementing a logic analyzer (see Fig 1). I used many of the ideas from the original LA in the design of this combination device. My main goals for the DSO/LA were:

- Ease of construction—use printed circuit boards
- Two analog channels
- Minimum 20-MHz sample rate
- LA channels
- Trigger selection to allow any combination of analog, digital and LA signals
- An initialization file (referred to in the text as the INI file) that allows the user to define almost all of the operating parameters

The design consists of circuit boards that stack one upon another (see Fig 2). They are connected together via "board stacking" connectors (see Fig 3). This method allows me to design a system that is easy to modify, expand and customize. It allows the builder to implement up to 48 LA channels (in multiples of eight) or a maximum of six analog channels and any combination thereof. The resulting assembly is fairly close to a 4.5-inch cube with two analog cards and one LA card. Great care was taken in the design of the circuit boards in order to achieve this modularity. A template board having just the interboard connectors was designed first. It was then used as the basis for all the boards.

I wanted to achieve a 20-MHz sample rate so that the analog channels would be useful to 10 MHz. The

119 River Run Cir
Sacramento, CA 95833
k3pto@arrl.net

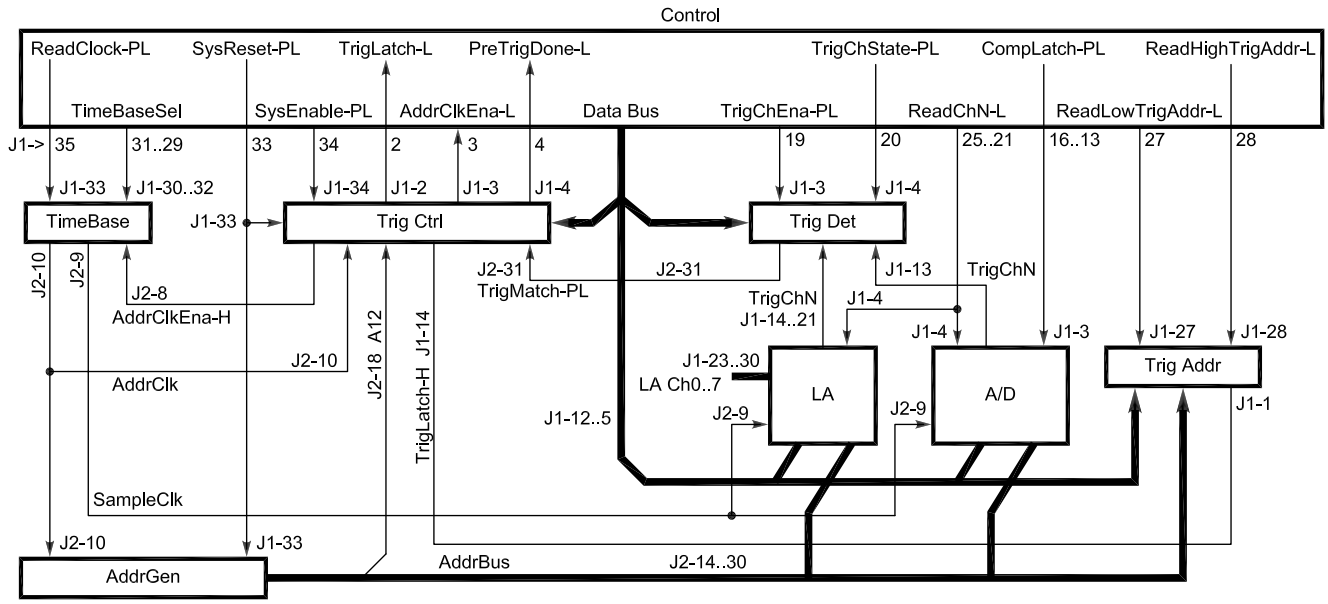

Fig 1—A block diagram of the digital-storage oscilloscope/logic analyzer project.

Fig 2—A picture of my "sandwich" construction before I converted several circuits to PLDs.

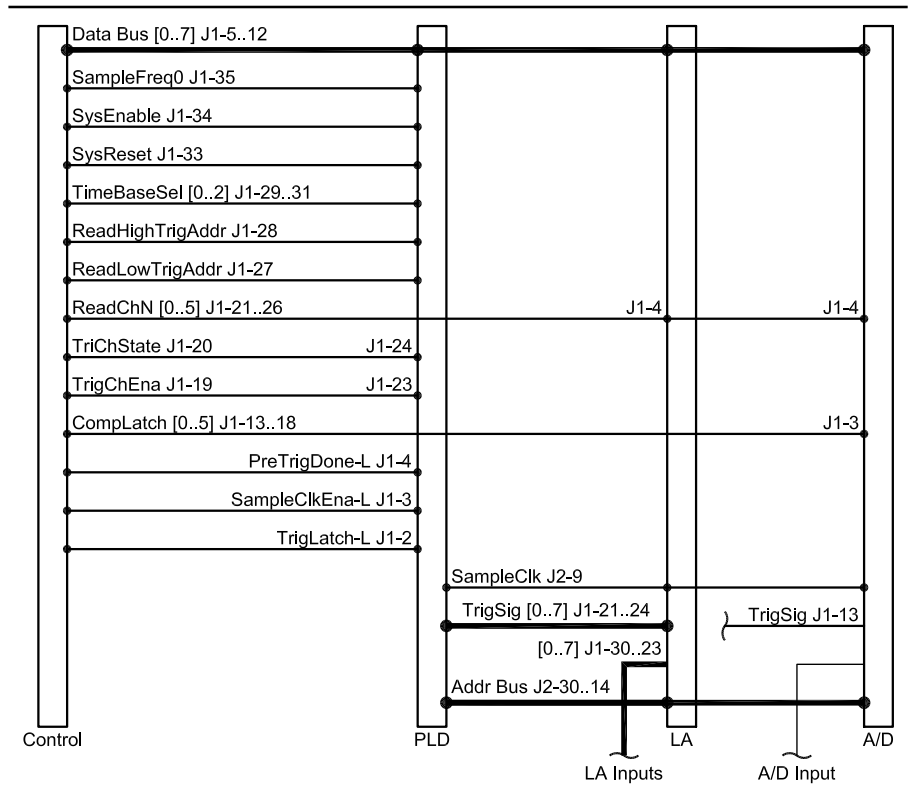

Fig 3—A wiring diagram of the connections between boards.

A/D converter selected (ADS820) is specified to a 20-MHz sample rate, a 60-MHz bandwidth and has 10-bit resolution. Since the circuit uses only the most significant eight bits, some errors of the A/D can be ignored. A higher-frequency A/D IC is available, and it is pin compatible with the unit I selected, but I doubt my ability to implement a circuit board that would allow the additional bandwidth. The circuit appears in Fig 4.

Major Circuits

- Control—interfaces to the printer port, source for all the static control signals.
- Trigger Address—stores the address at which the trigger occurs.
- Trigger Control—develops most of the dynamic control signals, pre/post trigger count.
- Trigger Detect—detects the trigger condition.
- Address Generator—17-bit address = 132 k.
- Time Base—generates the sample clock frequencies.
- LA—CMOS levels, eight channels, 132 k memory depth.
- A/D Converter—+0.5 to +4.5 V, 8-bit resolution using a 10-bit A/D and 132 k memory depth.

Design History

During the various phases of building and debugging the circuits, I developed several versions of some cir-

cuits. Initially, the above circuits were built on seven individual circuit boards. I managed to implement the Address-Generator and the Trigger-Address circuits on opposite sides of the same board. I used this implementation to get the initial hardware and software design working. The completed assembly was an approximately 4.5-inch cube with one LA and one A/D board. Since I do not have through-hole construction capability, there were quite a few hand-wired jumpers on the various boards.

I then decided to try my hand at using *programmable logic devices* (PLDs). Design number two combined the Time-Base and Address-Generator circuits into a single PLD and the Trigger-Control and Trigger-Detect circuits into another PLD. The PLDs are on opposite sides of the same circuit board. I had never used PLDs before, so this was a learning experience. Getting the software to work with this design was relatively easy, since I was able to keep the design essentially the same, with only some minor changes. This design has five circuit boards.

Design three combines the following circuits into a single PLD: Trigger Address (Fig 5), Trigger Control (Fig 6), Trigger Detect (Fig 7), Address Generator and Time Base (Fig 8). This yields a much more compact system that requires only two circuit boards more than the desired input boards. Additional wiring is shown in Fig 9.[1]

I would like to encourage any of you who are designing logic circuits to consider PLDs. My experience was most satisfactory. The free software is very easy to use, especially since I had already debugged my initial circuitry. It was only necessary to translate my schematics to functions available in the PLD design software. The "larger" functions I needed were available as library elements. To give you a better idea of the ease of use of PLDs (Programmable Logic Devices), I will expand on the 17-bit counter example.

The address generator requires a 17-bit (131,072 count) synchronous counter. A synchronous counter is required so that all the address bits change at the same time, synchronously. The discrete circuit I initially implemented is shown in the Fig 10. Notice that it uses five ICs. Even though I use only one stage of the fifth device, it is required to maintain synchronous operation.

The PLD implementation is quite a bit simpler (see Fig 11). I needed to select the synchronous counter from a list of device models, select and define the needed parameters and place it in my schematic. A complete list of possible parameters appears as table B in the download package (see Note 3). The only inputs I needed were *aclr* and *clk*, the only output was *q[]* and the parameters I specified were direction and width.

Once the part was placed, I then needed to "wire" the I/O ports to the appropriate circuits or I/O pins. See the AddrClock circuit of the Time Base and Address Generator schematic.

When the schematic entry has been completed, the next step is to "compile" it. This basically assigns the schematic elements to the internal circuit blocks of the PLD and determines whether or not the circuit fits. If the circuit does not fit, there are several options available that can be used to reduce the amount of resources used. Some of these are:

1. Disabling JTAG (Joint Test Action Group) capability.

2. Disabling globally assigned signals such as clocks and clears.

Once this stage of compilation is complete, the pin assignments must be made. This can be done automatically or manually. I always chose manually, because I wanted the PLD I/O to allow the easiest printed circuit board layout. The manual method is really quite easy. I had previously determined the signal-to-pin assignments based on the PC board layout. The PLD software presents a list of signals as well as a "picture" of the PLD. All I had to do was "drag and drop" each signal onto the desired pin.

Now the final compilation phase can be executed. This phase attempts to "fit" the circuit into the PLD with the pin assignments. The software I used does several iterations to complete the fitting process. I have seen up to 20 iterations. If it is successful, a file is created that is used to program the PLD. If the process is not successful, some pin reassignment will be necessary.

Finally, it is time to program the PLD. A programming device or equivalent circuit is required. I was able to obtain a programmer from the PLD manufacturer; but, since the circuit consists of a common IC and a few resistors, it would not be difficult to build one. The PLD I selected is capable of in-system programming. This requires only that I implement the appropriate connector on the circuit board with four connections to specific PLD pins. These four pins can be used for I/O, but I chose to not do that. The programmer connects to the printer port of the PC and the connector on my board. Once the PC software programmed the PLD, I removed the programming cable from the board.

The circuit is now ready for the "smoke" test! Since it is impossible to probe the internal circuit points, I strongly suggest that test points be designed into the PLD circuit. At one point of my design cycle, I had three of them. These allowed me to 'scope some of the critical timing points. Because I had previously implemented identical circuits using discrete logic, my debugging was minimal, so I did not need many test points. Untested circuits will probably require more. I recommend that outputs be buffered so that external loading of the I/O pins does not adversely affect system operation.

This was my first attempt at using PLDs, and I found it to be much easier than expected. The free software, downloaded from the Web, is fairly intuitive and comes with a tutorial. I ran the tutorial through the first dozen or so steps; then, like a typical engineer and ham, went off on my own. I was able to go from starting with no previous experience to a finished product within a few weeks working an hour or so several evenings a week. In the download package (see Note 1), Table B shows a complete list of the possible I/O signals and parameters that may be used to define the operation of the counter; Table C lists the parts used in each board.

Table 1—Printer Port Signal Usage

Output Control Bits

Pin	*Printer Port Name*	*Usage*
1	/C0: Strobe	RegSel 0
14	/C1: Auto-Linefeed	RegSel 1
16	C2: Initialize	RegSel 2
17	/C3: Select-Printer	Strobe

Input Control Bits

Pin	*Printer Port Name*	*Usage*
10	S6: Ack	PreTrigDone-L
11	/S7: Busy	
12	S5: Paper-Out	AddrClockEna-L
13	S4: Select	Trigger Latch-L
15	S3: Error	

Data Bits

Pin	*Printer Port Name*	*Usage*
2 - 9	D0 - D7	Data
18	Ground	

Circuit Descriptions

I have followed a naming convention for the signal names to more easily determine their operation and make the schematics and other docu-

[1]Notes appear on page 00.

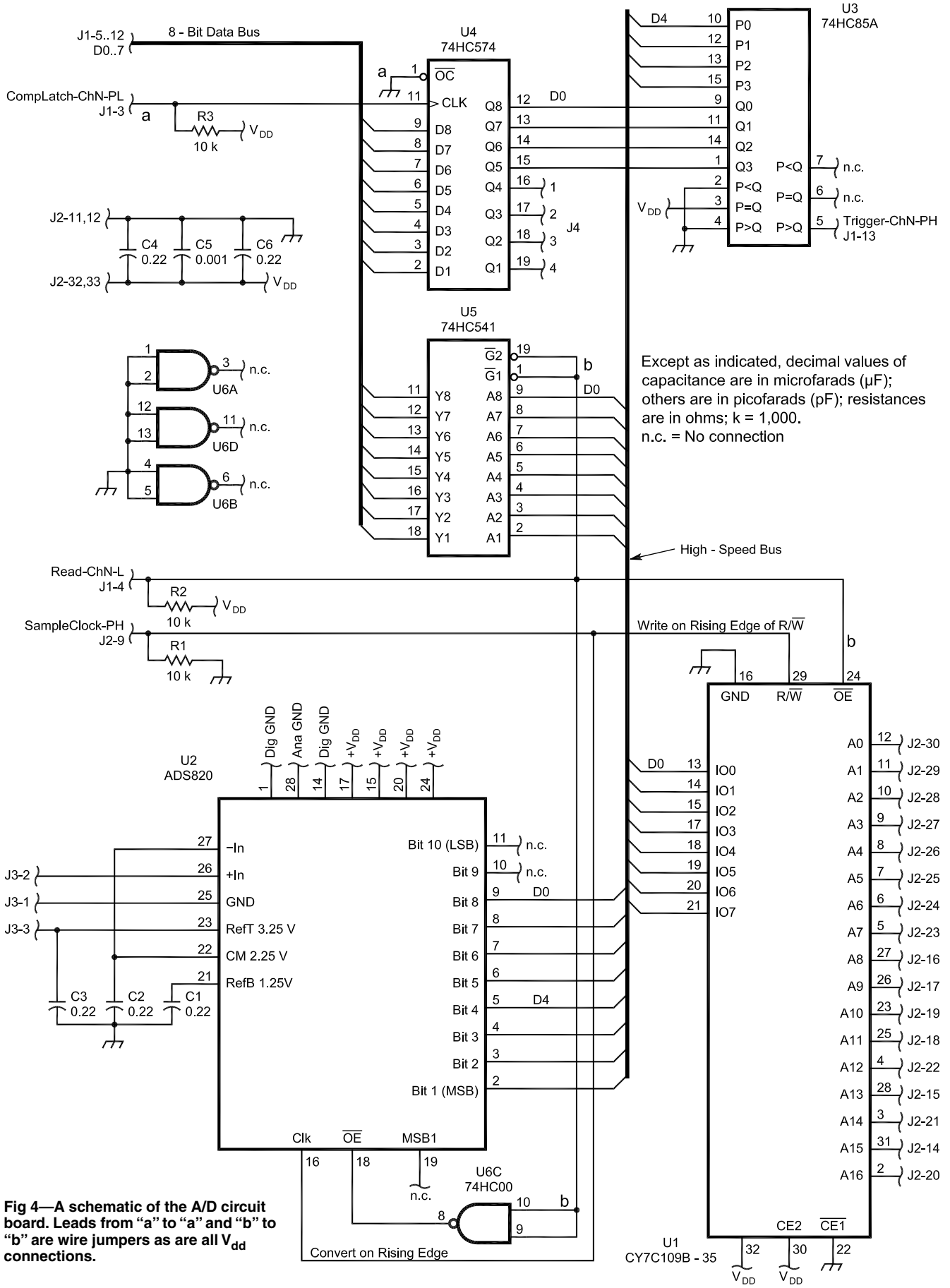

Fig 4—A schematic of the A/D circuit board. Leads from "a" to "a" and "b" to "b" are wire jumpers as are all V_{dd} connections.

mentation easier to follow:

1. Dynamic signals have a "PL" or "PH" suffix, indicating "pulse low" or "pulse high."

Example: *SysReset-PL*.

2. Static signals have an "H" or "L" suffix indicating a "High" true or "Low" true state.

Example: *Read-Ch0-L*.

3. Static signals, which are neither high nor low true, do not have a suffix.

Example: *TimeBaseSel0*

In general, I have labeled only signals that go between circuits.

Control Circuit

The main purposes of the Control Circuit (Fig 12 and 13) are to provide the interface to the PC (parallel port) and develop the static signals used to control the remaining circuits. The printer-port control bits are used to control the primary-decoder (U11) function. The functions performed by the primary decoder are:

- System Reset
- System Enable
- Secondary Decoder and Time Base Select Strobe—U12
- Read Clock and Secondary Decoder Read Strobe—U15
- Secondary Decoder Write Strobe—U14

Except for the Time Base Select signals, all of the signals generated by

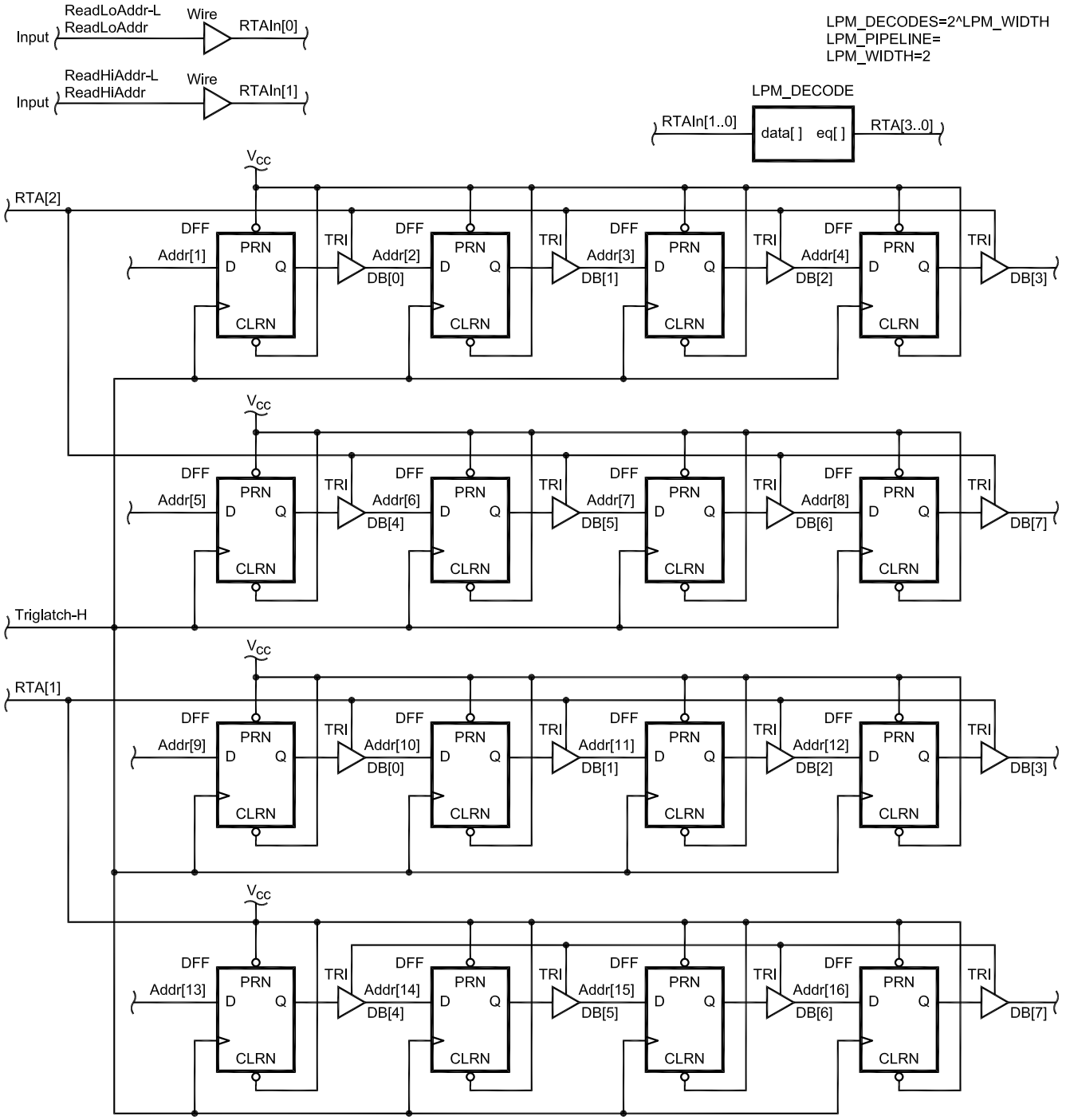

Fig 5—A schematic of the PLD Trigger Address Latch circuit.

this circuit are low-true pulses.

There is also a bus transceiver (U13) that controls the direction of data flow between the PC and the DSO/LA. It uses the *ReadClock* to determine the data direction. When *ReadClock* is low, read from the DSO/LA; when it's high, write to DSO/LA. Although many of the circuits allow for customization, this one does not, since the software is written to correspond with the functions it performs.

Time Base

The operation of the Time Base is rather straightforward (see Fig 8). There are three control bits (with possible expansion to four) that come from the Control Circuit (*TimeBaseSel*). These bits control a multiplexer that is used to select from among eight signals that can clock the Address Generator (*AddrClk-PH*). Seven of these are also the Sample Clock signal for both the A/D and LA circuits (*SampleClock-PH*). The eighth signal is for advancing the Address Generator when the program needs to read the data from the A/D and LA RAMs. The 20-MHz oscillator and Sample Clock are enabled by a signal from the Trigger Control Circuit (*SampleClockEna-H*).

Notice that the signal driving the Address Generator goes through three inverters, while the signal that drives the Sample Clock only goes through a single gate. There are two reasons for this configuration:

1. The Sample Clock should be disabled when reading from the A/D and LA Circuits.

2. The Address Clock is delayed slightly so that the input signals are sampled a short time (two gate delays) before the address bus is incremented. This allows the input signals and address bus to settle for almost a full clock period before the data is sampled and is really only significant at the highest sample rate but is necessary to meet the access time specification of the RAM.

I elected to make the sample rate/period circuit as simple as possible. I have used binary dividers to generate the various sample rates. Starting with a 20-MHz clock the next rate is 10 MHz, followed by 5 MHz and so on. The sample periods with this circuit are: 50 ns, 100 ns, 200 ns, 400 ns, 800 ns, 1.6 µs and 3.2 µs. These may seem like strange values, but it really does not make any practical difference in the interpretation of the displayed signals. Since the software allows you to measure the time between cursors, it accounts for the

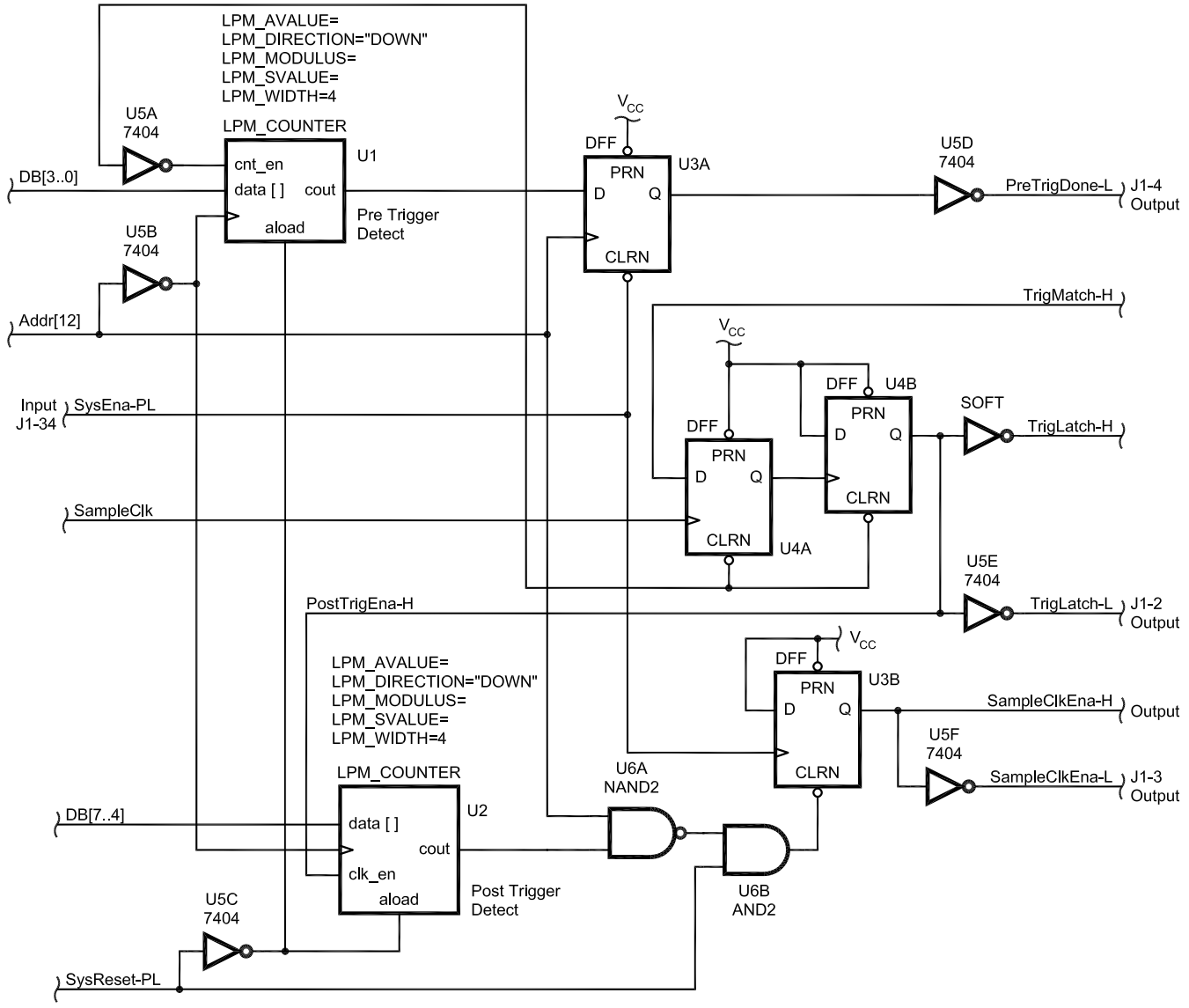

Fig 6—A schematic of the PLD Trigger Control circuit.

sample period. Also, the software allows you to set the time scale.

The design of the system is such that if you do not like the sample periods I have chosen, you can redesign the circuit to develop the ones that you like. You can specify the sample periods in the INI file. All you need to do is relate the 3 or 4-bit decoder input value to the sample period as follows: SELECT_<*decode value: 1 … 15*> = <*sample period*>. For example: SELECT_1 = 50ns. The program correctly interprets nanoseconds, microseconds and milliseconds as units of time.

Address Generator

This is a 17-bit synchronous binary counter that yields 2^{17} = 131,072 addresses (see Fig 8). It is driven by the Address Clock from the Time Base Circuit. The counter outputs are all set to zero by the *SysReset-PL* signal from the Control Circuit. Because of the way the Pre and Post Trigger circuits operate, the full count is not utilized. This will be explained further in the discussion of the Trigger Control circuit.

The original design of the circuit called for 74HC161 binary counters that are specified to 25 MHz with a 5-V supply. When more than two units are cascaded, however, there is a glitch. I had not thoroughly read the details of the specification and missed the information on the glitch. I thought that since they were rated to 25 MHz, I would not have a problem since I was only going up to 20 MHz. The glitch occurs because of internal delays when using the Ripple Carry Out. This limits the upper frequency to 17 MHz when cascading. The solution was relatively simple: use 74AC161 devices instead. They are rated at a much higher frequency but draw quite a bit more current. Since I have now converted the circuit to a PLD, the above situation is no longer applicable, but I thought it worth mentioning since it was part of my learning experience.

Trigger Detect

Those of you who have used commercial LAs will recognize that the triggering available with this design is very limited. These units will usually have quite a few possibilities for triggering such as:

- High level
- Low level
- Low to high transition
- High to low transition
- Either transition
- Various logic combinations of selected signals

This circuit compares the current state of the trigger inputs to those that have been selected by the operator. The circuit operates as an 8-bit **AND** gate (see Fig 7). Each of the eight input signals can be enabled or disabled and inverted or not inverted. The source for each of the eight inputs is up to you. I have mine set up so that the first six signals are channels 0 through 5 of the first LA board. The remaining two inputs are from the two A/D boards. This is one area where the system is not very flexible. You cannot dynamically select from among all the possible inputs. You must predetermine which eight signals you will use as possible triggers. If this is not flexible enough for your applications,

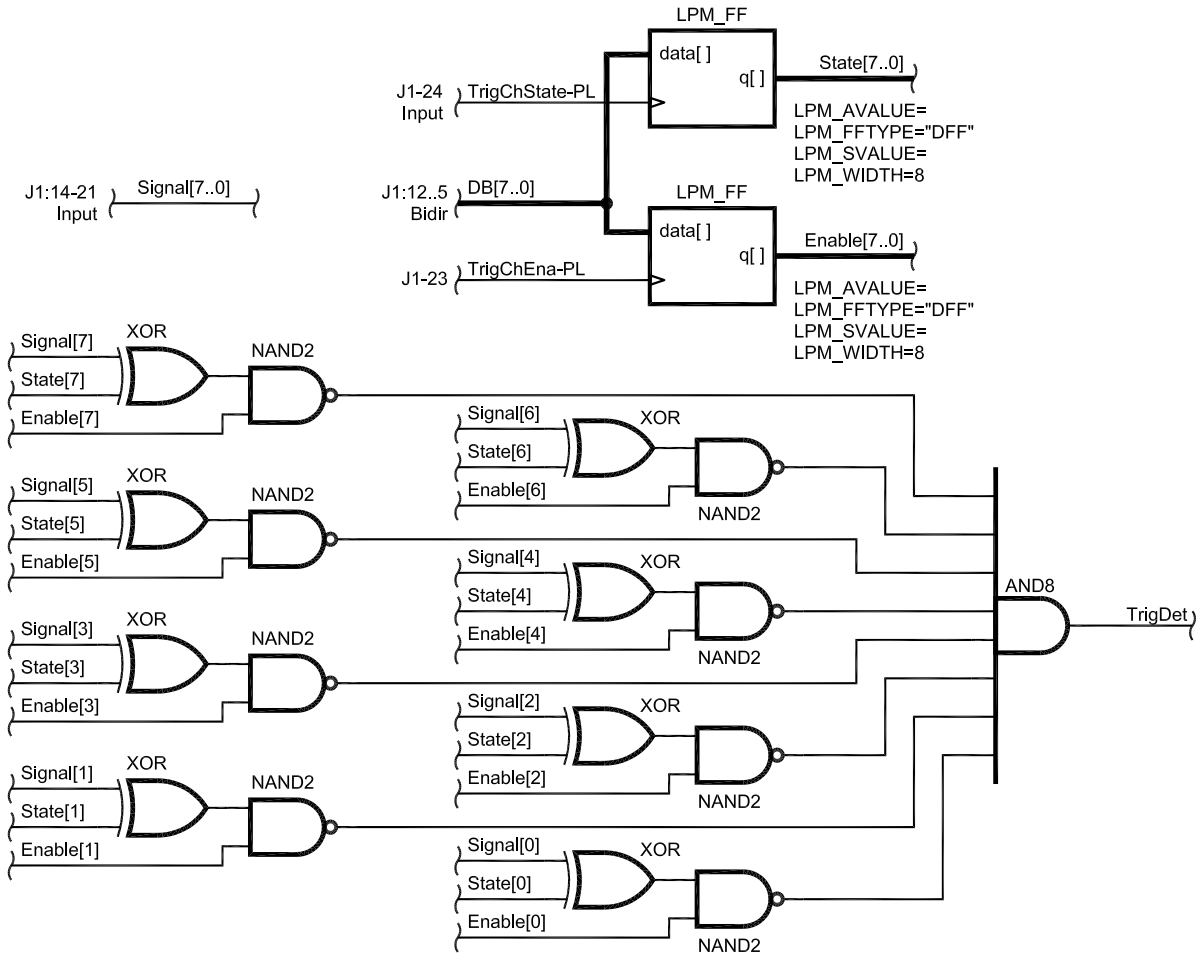

Fig 7—A schematic of the PLD Trigger Detect circuit.

you will need to implement a switch of some kind. I am certainly open to suggestions about how to make this function more flexible.

Each exclusive-or gate is used as programmable inverter. When its control input is low, the output follows the input. When the control input is high, the output will be the inversion of the input. The following **AND** gate is used to enable/disable the input signal. When its control signal is low, the output is forced high, independent of the input signal. This basically creates a match condition for the final eight-input **AND** gate. If all the signal inputs are disabled, there will be a continuous match condition.

The original, discrete-logic version of the circuit was quite different. The 8-bit latches were 74HC574s, very similar to the PLD implementation. However, the enable/disable circuit used tri-state buffers (74HC126) and the output **AND** gate was actually an 8-bit comparator (74HC688). The circuit required five ICs. If I were to implement the circuit I designed for the PLD as discrete ICs, it would require seven. Since the circuit is now in a PLD, I can probably modify it to meet some other requirements.

Trigger Control

The Trigger Control (Fig 6) was perhaps the most difficult circuit to design and get working as I wanted. My goal was to have a programmable pre-trigger capability—unlike my original LA, which has a fixed amount of pre-trigger. I have added "U" numbers to the schematic of the PLD

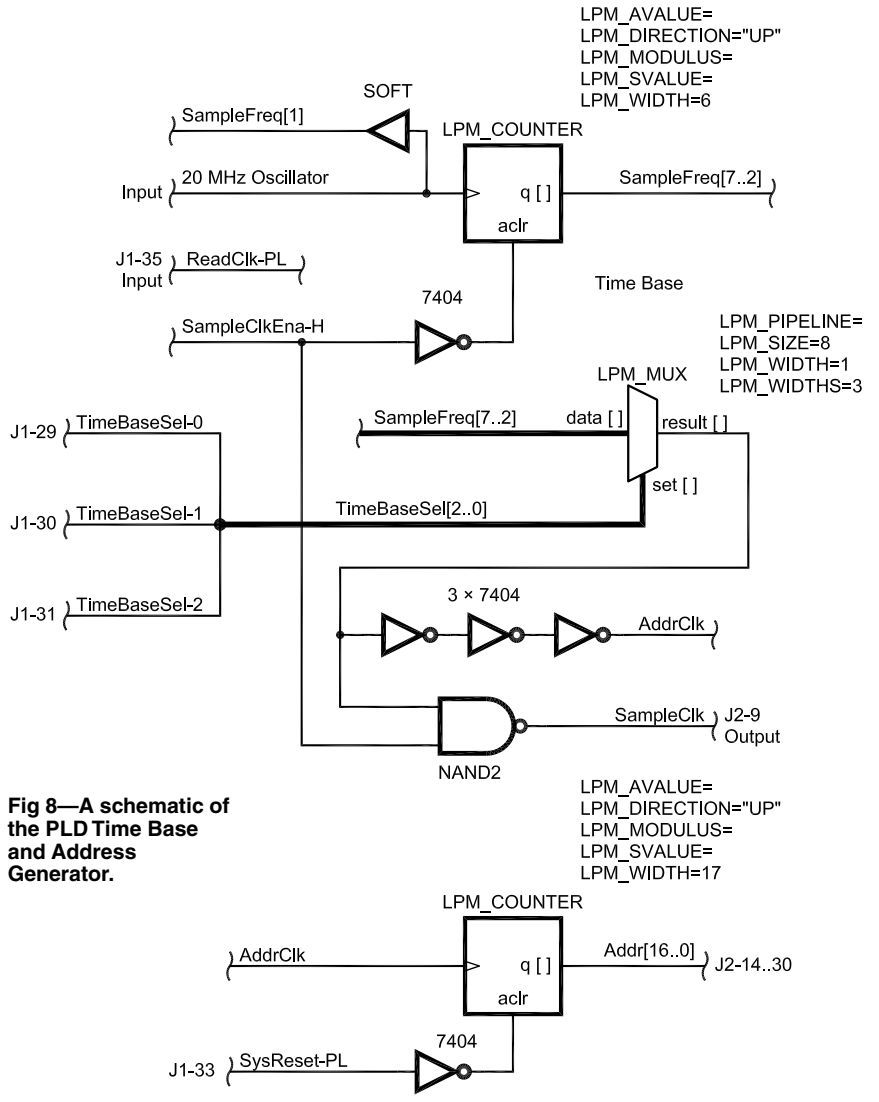

Fig 8—A schematic of the PLD Time Base and Address Generator.

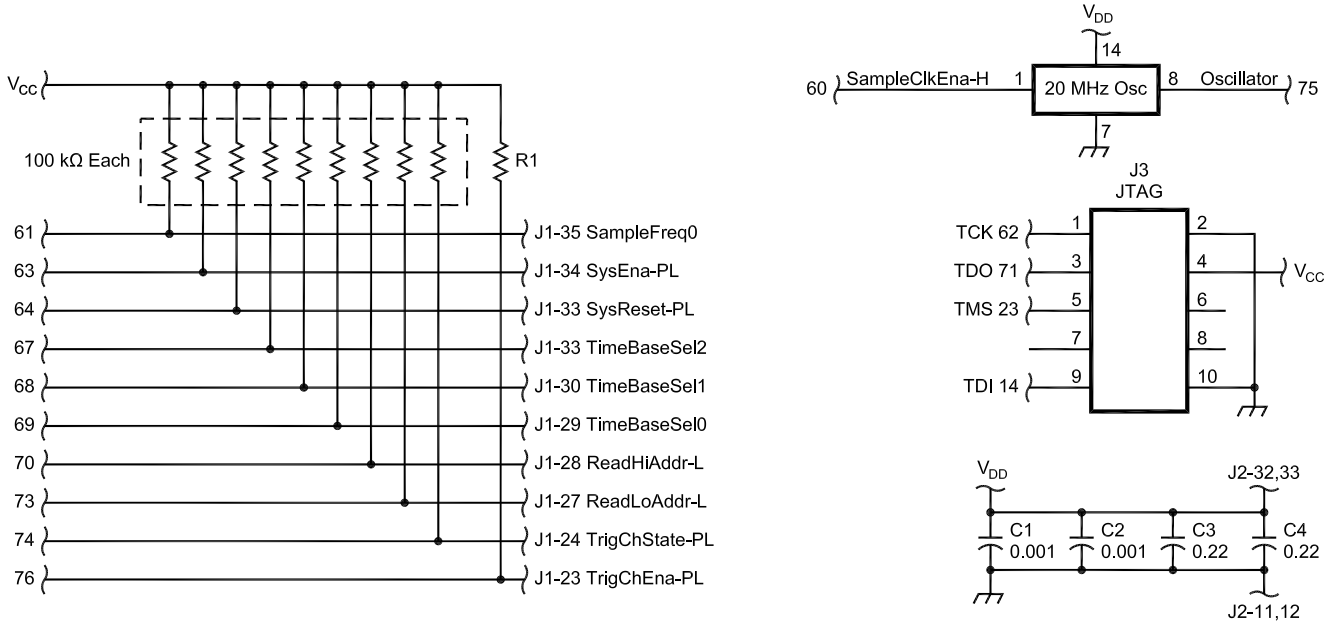

Fig 9—A schematic of the PLD board. Numbers without connector designations (J1 or J2) are on the PLD (EPM7128SLC84). See Table 2 for most signal connections.

circuit in order to make it easier to reference the different parts of the circuit. Both U1 (the pre-trigger counter) and U2 (the post-trigger counter) are 4-bit down counters. They are clocked by the inverted A12 signal. As such, they decrement every 8192 samples.

You can set the pre-trigger count to any value between 0 and 14. The number of pre-trigger samples can be determined by this formula: pre-trigger samples = (pre-trigger count × 8192) + 4096. The post-trigger count is set by the system as follows: post-trigger count = 15 – pre-trigger count. The total number of samples is (15 × 8192) + 4096 = 126976 = 2^{17} – 4096.

Once the system is "armed," the pre-trigger counter starts counting down to 0. When it reaches 0, the output of U3A will go high on the next positive transition of A12 (4096 samples later). This enables a Trigger Match to be detected. A Trigger Match will then cause the *Q* outputs of U4A and U4B to go high, thus enabling the post-trigger counter. When it reaches 0, the sample clock will be disabled and the system will stop sampling.

Three status signals are monitored by the program that allow it to determine the state of the system:

- *PreTrigDone-L*
- *TrigLatch-L*
- *SampleClockEna-L*

A message is displayed as the system waits for each of these signals. Please notice that each of these signals is derived from signals used internally by the Trigger Circuit. However, I have buffered them via sections of U5 so that the wiring associated with getting them to the PC does not interfere with the circuit operation.

The PLD version of the circuit is almost identical to the discrete version. I would like to detail some of the differences for those of you who wish to convert a circuit to a PLD:

The carry out of PLD counters is true when high and becomes active as soon as the count reaches 0 or 15, depending on the count direction. The discrete device I used is a 74HC191. Its carry out is true when low and is gated with the clock so that it goes true during the second half of the end count.

Both counter-control inputs, Count Enable and Load, are true when high for the PLD but are true when low for the '191.

The discrete version required an **AND** function so I used a 74HC00 as a **NAND** and then followed it with another section to invert it. This is certainly not necessary in a PLD, since it has built-in **AND** blocks.

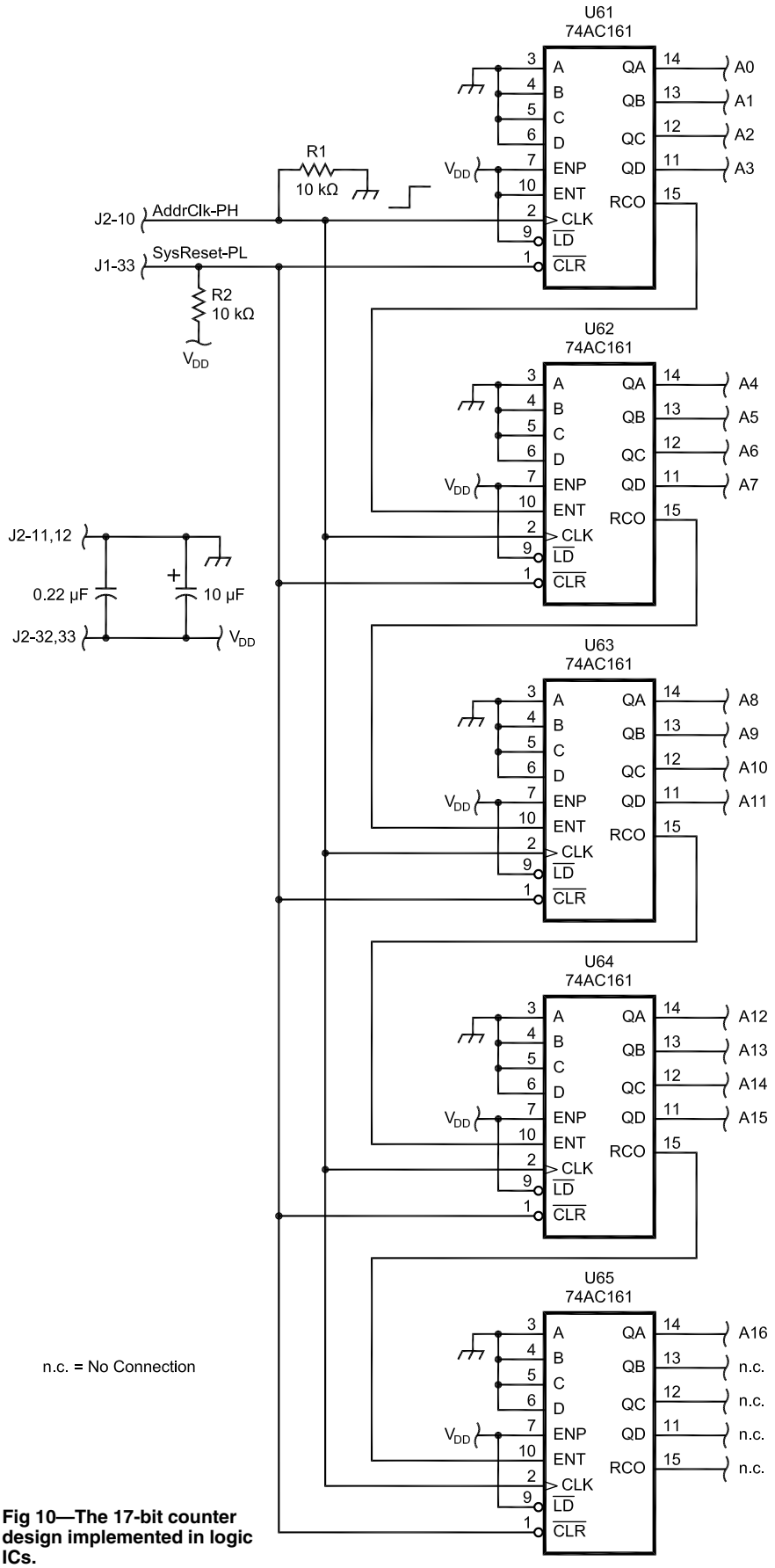

Fig 10—The 17-bit counter design implemented in logic ICs.

LA

This is another relatively simple circuit (see Fig 14). All it must do is apply eight input signals to the data bus of the RAM and allow the data bus to be read back by the PC. There are two resistor networks in the input circuit. One insures that the input-switch device has no floating inputs. The other limits current in case of excessive applied signal voltage. As can be seen on the schematic, U42 and U43 create an eight-pole, two-position switch controlled by the *Read-ChN-L* signal. When *Read-ChN-L* is high, the RAM data bus is connected to the input signals, and the *SampleClock-PH*

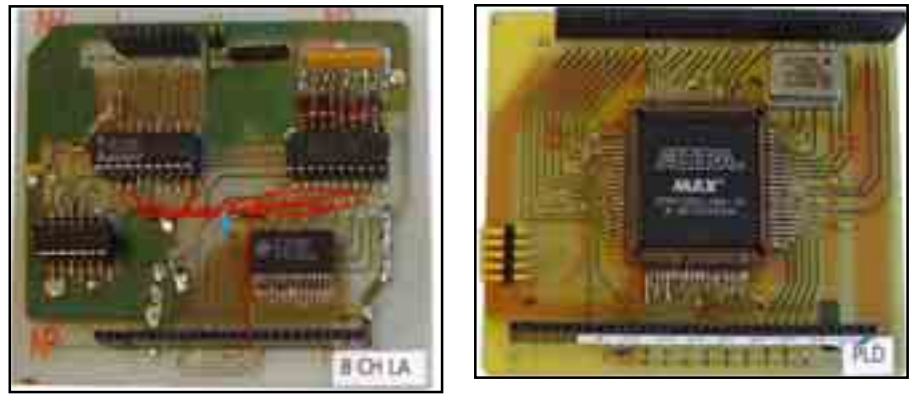

Fig 11—The circuitry of the eight-channel logic analyzer achieved with ICs (left) or a PLD (right).

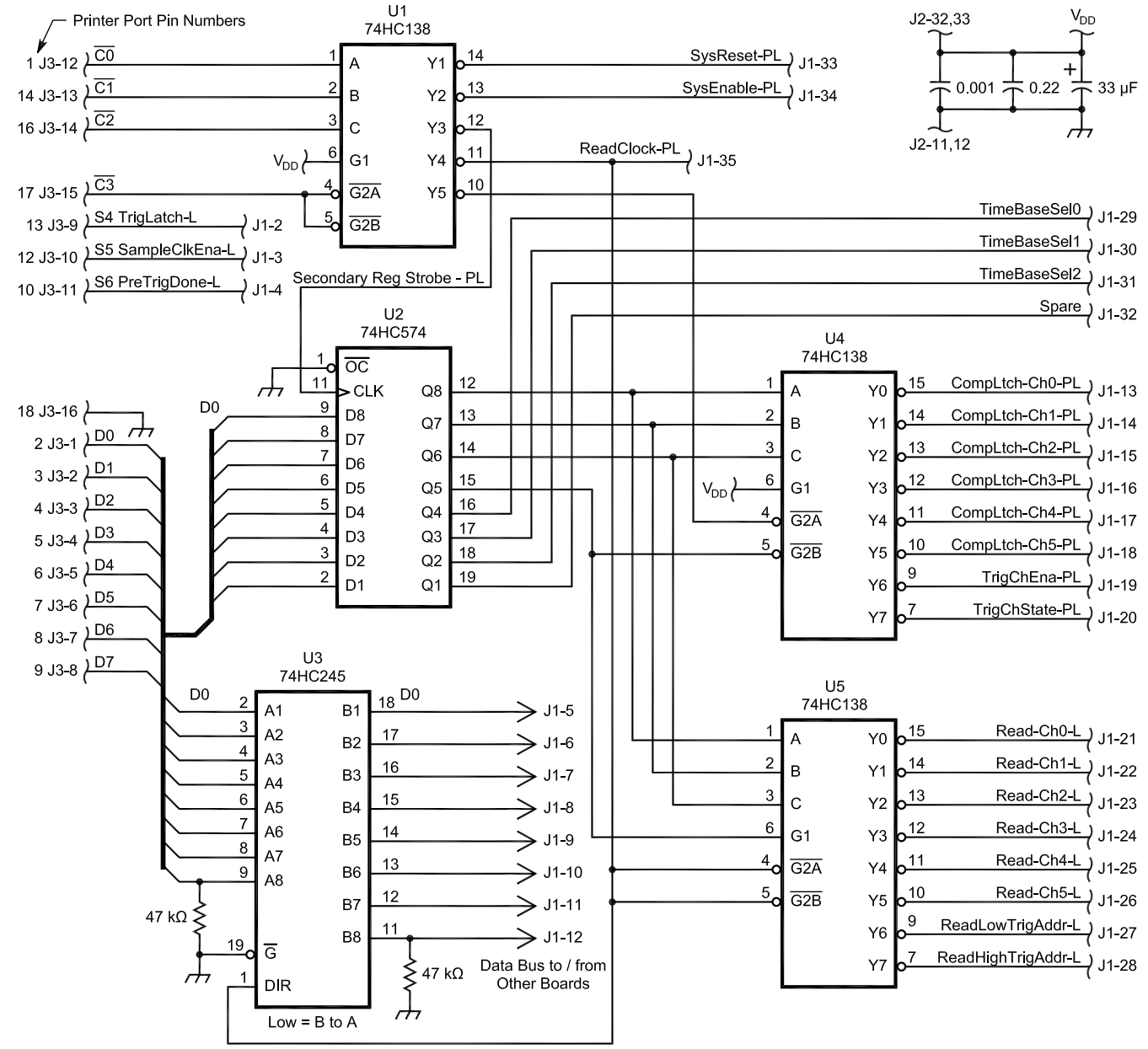

Fig 12—A schematic of the DSO/LA control board.

signal writes the data to the RAM on its rising edges. The *SampleClock* signal is high during the read operation. When *Read-ChN-L* is low, the RAM data bus is connected to the system data bus, which is eventually read by the PC. Each LA board has its own *Read-ChN-L* from the Control circuit.

A/D

The circuit is somewhat similar in operation to the LA except that the eight data bits come from an A/D channel instead of individual digital sources (see Fig 4). The *ReadChN-L* signal is used to determine whether the circuit is sampling A/D data or transferring it to the PC. U55 is used to connect the RAM data to the system data bus when *ReadChN-L* is low. When this signal is high it allows the *SampleClock-PH* signal to clock the A/D and write data to the RAM.

The A/D I selected (ADS820) requires a single 5-V power supply and has a maximum 20-MHz sample rate and a full power bandwidth of 65 MHz. There are faster pin-compatible versions available, but I did not feel that I could adequately implement the circuits for higher frequencies. There is nothing in the system design, however, that would prohibit you from implementing a faster A/D if you desire. The RAM (CY7C109B-35) is good up to about 28 MHz but has faster versions. The -20 is a 50-MHz part.

When sampling the input signal, the rising edge of *SampleClock-PH* initiates an A/D conversion and writes A/D data to the RAM. These types of A/D converters have a pipeline architecture. In the ADS820, this causes a six-cycle latency between when a signal is sampled and when the converted data is available on its output bus. In my design, this means that I need to account for the latency in the software so that I store the data in the correct place.

There are two additional functions on this card:

- Digital comparator
- Four-bit latch for controlling the A/D buffer

The digital comparator allows the user to set a trigger level that can then be used to trigger the system. On the schematic, notice that I have labeled the RAM data bus as "High Speed Bus." I tried to be as careful as possible in routing these signals in order to maintain signal integrity. The comparator is operating on the A/D samples at the sample rate, as high as 20 MHz. When the digitized signal value is greater than the comparison value stored in the upper four bits of U54, the **P>Q** output of U53 will go high. This signal can be used to trigger the system. I have implemented a four-bit comparator so the resolution is only 1 part in 16 (6.25%). I felt this was a reasonable compromise so that I could use the remaining four bits to control the A/D buffer.

The timing requirements for writing A/D data to the RAM can cause problems if the circuit design is not carefully considered. A careful inspection of the A/D timing shows a "dead" time on its data bus immediately following the positive transition of the clock. Note that writing to the RAM also occurs on the rising edge. At first I thought this might cause a problem. However, the A/D specifications show that the data is good for a minimum of 3.9 ns after the rising edge, while the RAM specifications show a 0 ns input hold time for the data. Although 3.9 ns does not seem like very much, it is sufficient for the circuit to operate properly.

A/D Buffer

This circuit (see Fig 15) is built on a board that is not part of the "stack" forming the main body of the instrument. I wanted it as close as possible to the signal source, so I designed a small board that mounts directly to the front of the LA/DSO housing and has the input (BNC) connector mounted to the board. This keeps the shortest possible signal path to the buffering op-amp. This is a simple dual op-amp circuit that has three basic functions:

- Buffer the signal, which may be up to 10 MHz.
- Adjust the signal to the 4-V range of the A/D converter.
- Offset adjustment so that it is centered about 2.5 V.
- The circuit I designed has the following voltage ranges:
- –10 V to +10 V
  –5 V to +5 V
  –2 V to +2 V
  –1 V to +1 V

This particular op amp has relatively high input and offset voltages and currents so its dc characteristics are not the best. Nonetheless, it is fast enough to handle 10-MHz signals as

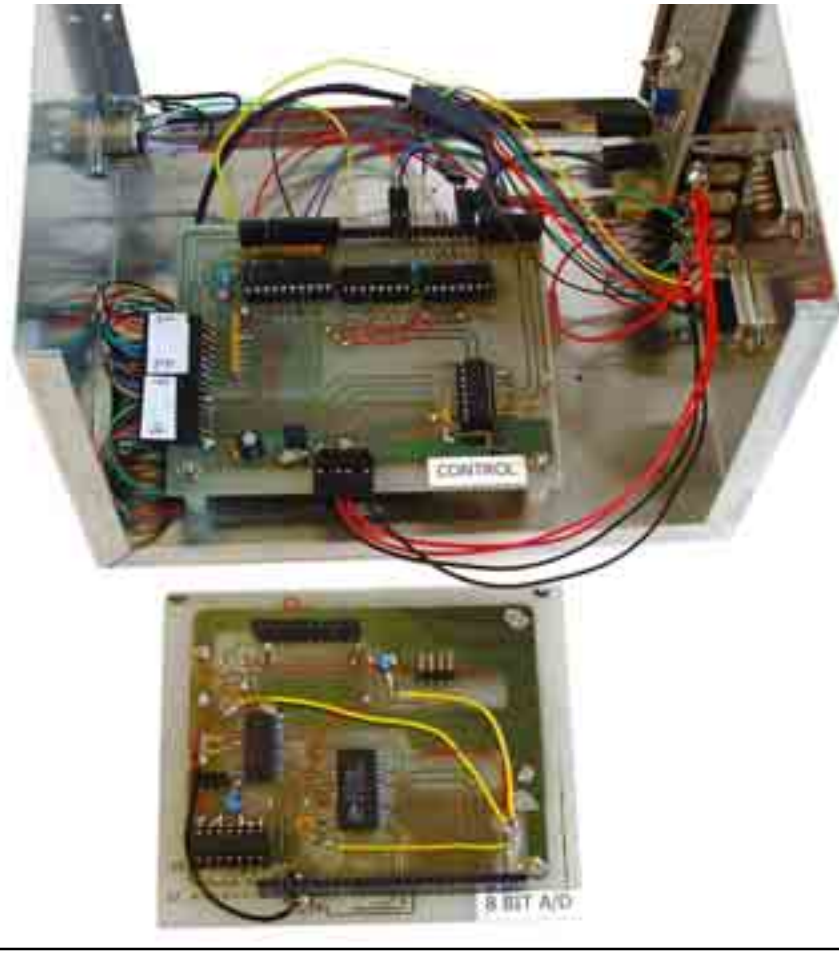

Fig 13—A photo of the Control circuit board and one 8-bit A/D board. Two of the five jumpers are there to "fix" broken traces. The other three are designed-in because I could not route them. There is one IC mounted on the underside.

long as you do not require the full 8-bit accuracy. There are op amps available that would be better, but they are probably more expensive and this great expense was not one of my primary design goals.

The input ranges are completely flexible in that you can make them whatever you want. The INI file allows you to specify the ranges associated with the 16 control values coming from the A/D circuit.

Trigger Address

This is perhaps the simplest of the circuits. It is simply a 16-bit latch that is triggered by the output of the Trigger Detect Circuit via the Trigger Control Circuit. This was done so that only a single trigger event will cause the address to be latched. I used a 16-bit latch, implemented via two 8-bit latches, because I did not want to build the necessary additional circuitry for the 17th bit. The program will read the upper 16 bits and then determine if the data actually meets the trigger criteria. If it does not, then it must be the next higher address. This little bit of extra programming saved me several ICs in the circuit design. However, now that I am using a PLD, I may go back and revisit this as I work on improving the system.

System Operation

The operation of the DSO/LA is fairly intuitive. You can get the basics simply by seeing the pictures that follow. There will be a complete Help file available, which explains each of the controls as well as the contents of the INI file. Here is a brief explanation of each of the menu items:

• File—typical file operations for read-

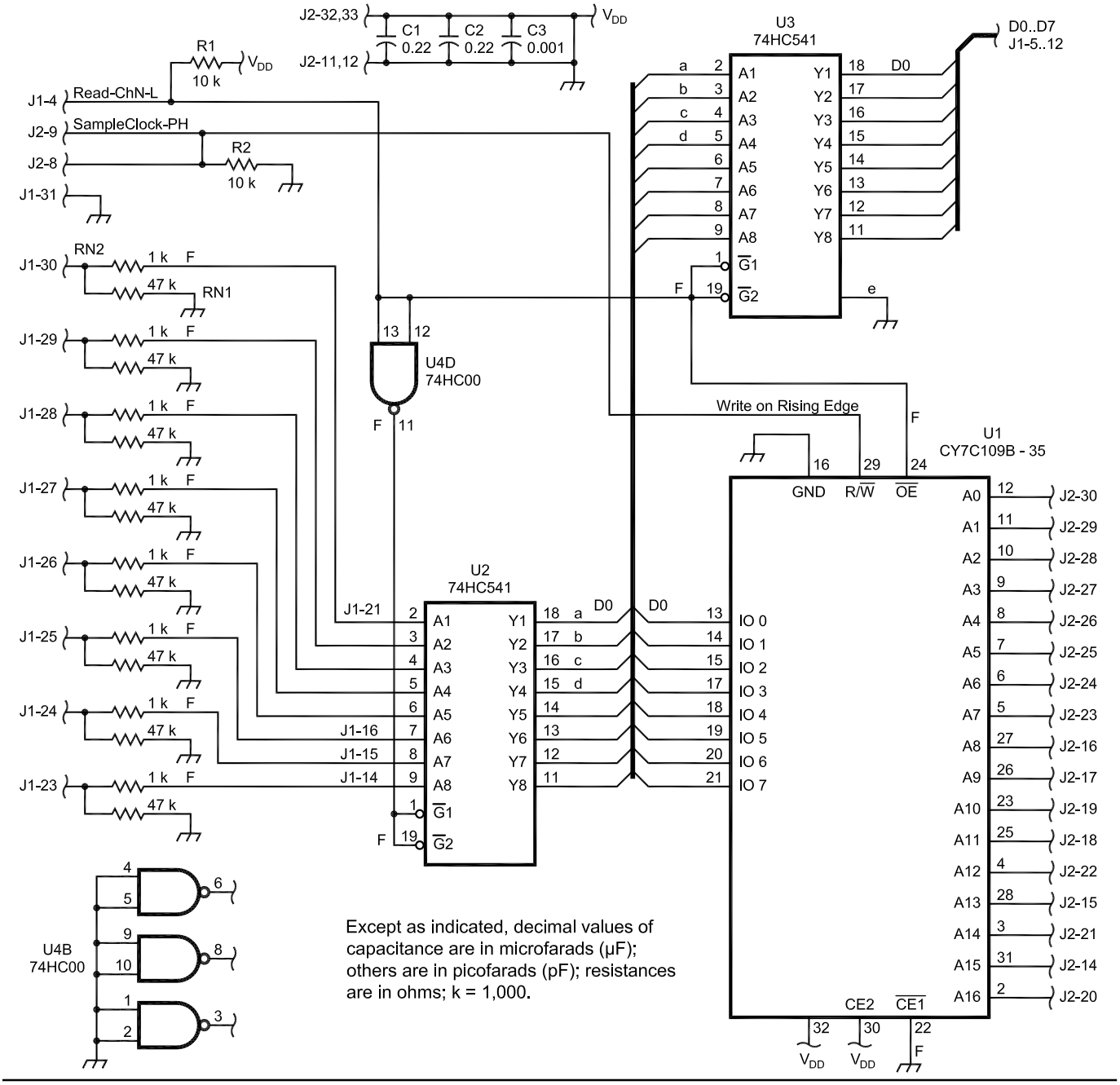

Fig 14—A schematic of a logic-analyzer circuit board. Wires labeled "a," "b," "c," "d," "e" and "f" are jumpers. Letters "F" indicate a connection between the two copper layers of the circuit board.

ing and writing both INI and data files.
- Boards—define the board for each of the six channels.
- Trigger—define trigger conditions, including the pre-trigger/post-trigger values.
- Display—select Time Base, Zoom factor, Slide factor and so on.
- Cursors—enable the four time-measurement cursors.
- Function Keys—list the applicable function keys.
- Debug—some hardware-debugging aids as well as some dummy data so you can "play" with the display.

Some General Comments and Suggestions for Making Circuit Boards

Make the traces as wide as you can. Most of the boards I developed started out with 20-mil traces. After completing the design, I would then go back and make each trace as wide as possible. Where necessary, I narrow the trace to fit between IC or connector pins. The main reason for doing this is to prevent the undercutting of traces during etching.

Make the power and ground traces even wider. This is done to lower their resistance. One of the guidelines I followed was to implement as many of the signal traces as possible ignoring the power. Since this was my first effort at double-sided circuit boards, and I can't implement feedthrough connections, I feel that wire jumpers for power and ground leads is better than running jumpers for signals. This allowed me to use relatively heavy wires for the power.

Implement a ground plane wherever possible. This reduces the amount of copper to be etched as well as giving more ground area.

Use a piece of scrap perforated board as a drilling template for the connector holes. Drill positioning by eye does not work well when for a 35-pin connector! I had some unused circuit boards from RadioShack with 0.1-inch-spaced holes. I lined up the holes with the positions to be drilled and then taped the "perf" board to my circuit board. Looking through the board with a 50-W lamp behind it makes this relatively easy.

Some General Guidelines

Drill the fewest holes possible. Looking closely at the photos, you may notice that the IC sockets look a little odd. I designed the boards as if using SMD ICs. However, I did use mostly normal DIP devices. I found IC sockets that have the flat side of their pins parallel to the long edge. This allowed me to bend them to the side at a right angle to the socket. I laid out the IC traces to ensure at least 0.1" of trace beyond the sides of the socket so that the socket pins could be soldered in place. I had to trim the leads on a few sockets, but this is quite easy.

Position ICs on both sides of the boards where necessary. This was easily implemented due mainly to the wide traces.

Keep the main circuit functions on separate boards. This makes for a very modular system that can be easily customized. It also makes testing the individual circuits easier. This was very important in the early designs so that I could make changes. It is not quite as important now that most of the circuitry is in PLDs.

Install "load" resistors on selected inputs. This, again, makes it easier to test the boards.

Install a few test points on each board—especially at least one ground. This makes it much easier for debugging! I even put a few test circuits into

Fig 16—Four address bus signals as captured by an LA board as well as the A12 signal as captured by an A/D board.

Fig 17—The same signals as Fig 16 but zoomed in.

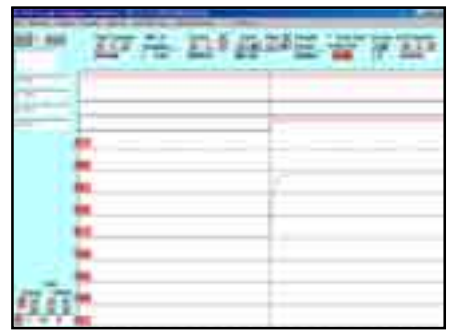

Fig 18—A view of A4 instead of A12. Also the time base has been changed from 1.6 µs per sample to 100 ns per sample. Notice that the signal fall-time can now be seen.

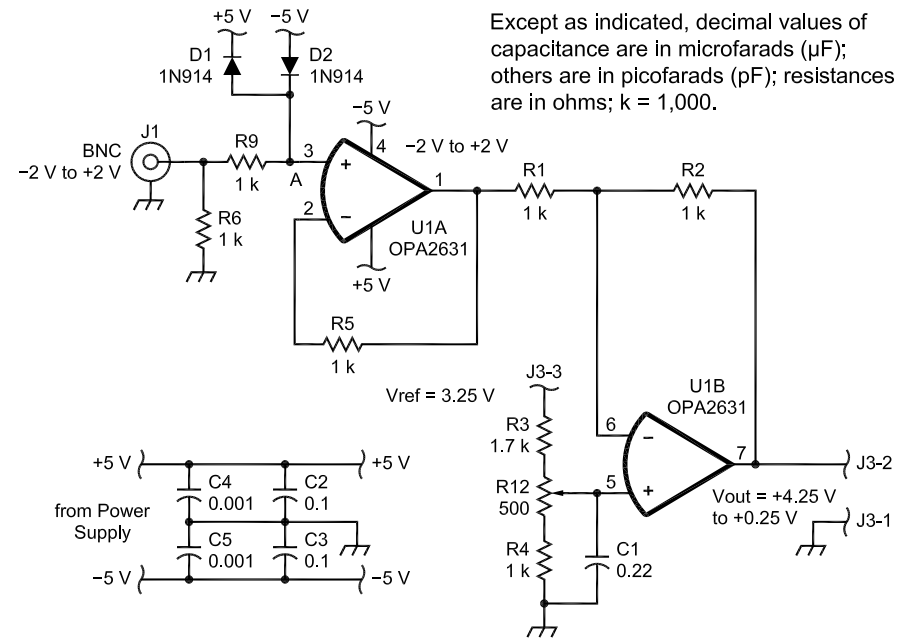

Fig 15—A schematic of the A/D input buffer and conditioning circuits.

the PLD in the first designs so I could monitor some internal signals.

Figs screens 16 through 18 are screen shots showing various features of the DSO/LA. Fig 16 shows four address bus signals as captured by an LA board as well as the A12 signal as captured by an A/D board. Fig 17 shows the same signals as the previous screen shot but zoomed in. Fig 18 shows A4 instead of A12. Also the time base has been changed from 1.6 µs per sample to 100 ns per sample. Notice that the signal fall time can now be seen. Several more features appear in on screen, but some don't show in the B/W figures here:

- A different color for the A/D trace
- The analog voltage for the A/D channel is in the Status box near the upper left corner. The background color is the same as the one selected for the trace.
- There is another cursor on the screen. The system allows up to four more cursors in addition to the main one.
- Horizontal grid lines for the A/D signal along with the Volts/Div next to the Sample Period.
- The A/D Scale and Offset features

Notes

[1]A package containing additional details of the PLD connections, possible PLD-counter I/O signals and parameters, a detailed parts list and PC-board etching patterns is available from *ARRLWeb.* You can download this package at **http://www.arrl.org/qexfiles/**. Look for 0307CICC.ZIP.