PALs Simplify Complex Circuits

A hardware designer's experiences with PALs

Trevor G. Marshall

THE PAL (programmable array logic) is not some recent, gee-whiz technology. You'll find PALs on your personal computer motherboard and in almost every other personal computer peripheral you examine. They sneaked into hundreds of designs while you weren't looking.

I remember yawning when, in 1980, John Birkner (the father of the PAL) showed me how to turn an ordinary-looking 20-pin DIP into an equally ordinary-looking DIP that supposedly, after programming, would contain six fundamental basic gates. "So what?" I said. "I know all about 74LS00-series gates and what I can do with them. Why should I want to create six different gates inside a DIP and pay 20 times the price of an LS00 for it?"

Four years passed before I realized what a mistake I had made. The PAL is a software element, not a hardware device. It allows designers to alter the topology of their logic designs even after the circuit boards have been fabricated. With PALs you can commit to production much earlier in the design cycle because an algorithmic change can solve any design or debug problems that might arise.

Without PALs, the DSI-32 (see "The DSI-32 Coprocessor Board," August and September 1985 BYTE) and the DSI-020 (see "The Definicon 68020 Coprocessor," July and August 1986 BYTE) coprocessor boards would never have been possible. Some of the eight PALs on the DSI-32 went through dozens of design iterations before the final product was shipped. The PALs were reprogrammed to correct deficiencies in the CPU and memory management unit, incompatibilities with some of the host personal computers, and errors in the basic design.

So much for history. I want to look at how PALs can help you implement your latest design and at how easy they are to use. I will focus on the commodity PALs, particularly the 16L8 and 16R4 types. These are inexpensive, typically costing less than $2 in production quantities. They are, however, adequately powerful to act as a training ground for a budding designer and to implement most synchronous or asynchronous logic designs. They are available with as little as 10-nanosecond maximum propagation delay, and you can program them with low-cost hardware. Manufacturers include Monolithic Memories, Texas Instruments, National Semiconductor, and Advanced Micro Devices.

Combinatorial PALs

Figure 1 shows the logic for a combinatorial cell from the 16L8. The hardware consists first of a grid of fuses that feed into AND gates followed by an OR gate, then a tristate inverter with a feedback term. Not all the elements have exactly the same topology, but they are similar. [Editor's note: *For more discussion of the architecture of the 16L8, see Vincent J. Coli's "Introduction to Programmable Array Logic" page 207.*]

A simple example illustrates the 16L8's software structure. I have used the syntax of the PALASM version 1 development software. (Monolithic Memories put this software into the public domain.) I'll name the 10 inputs A, B, C, D, E, F, G, H, I, and J (active high) and the output /O1 (active low). Note that the outputs are always inverted from the inputs in the 16L8 due to the inverter between the OR gate and the output. This inversion is denoted with the slash. So /O1 = A means that the O1 output will contain an inverted copy of the input A. Each of the inputs can be used to control the tristate enable, so the equation IF (/B) /O1 = A means that the inverter output was tristate (disabled) until B was negated (pulled low).

An equation for this cell can be quite complex:

```
IF (/A*B*/C*D*E*/F*G*H*/I*J)/O1 =
   A*B*C*/D*/E*/F*/G*H*I*J
   + /A*/B*C*/D*/E*/F*/G*H*I*J
   + A*/B*C*/D*E*/F*/G*H*I*J
   + A*B*/C*/D*/E*/F*/G*H*I*J
   + A*B*/C*D*/E*/F*/G*H*I*J
   + A*B*C*/D*E*F*G*H*I*J
   + /A*B*C*D*/E*F*G*H*/I*J
```

Note that the symbol * means logical

continued

Trevor G. Marshall has published over 50 papers in fields ranging from electronic music to biomedical engineering. He is director of engineering at Definicon Systems and can be contacted via modem at the Thousand Oaks Technical Database at (805) 492-5472 or (805) 493-1495 or by mail at Definicon Systems Inc., 31324 Via Colinas #108/9, Westlake Village, CA 91362.

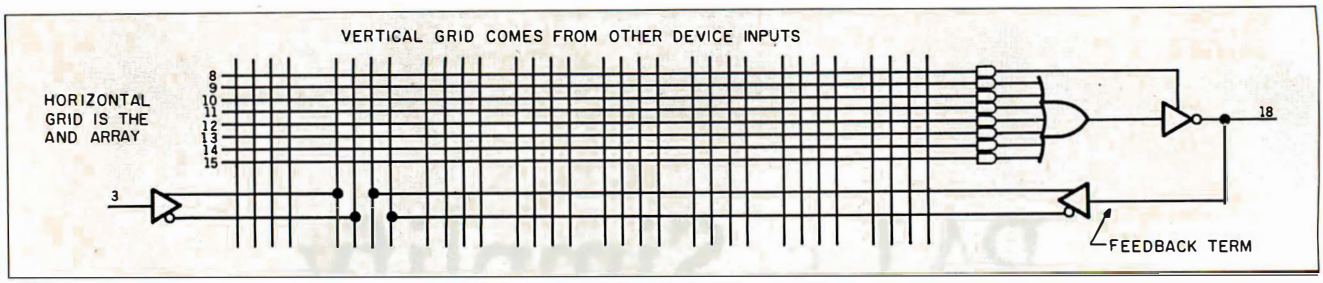

Figure 1: *A logic cell from a 16L8 PAL. Notice the inverter between the OR gate and the output.*

AND; the symbol + means logical OR.

At this point, I have used up the horizontal AND array for the tristate term and the seven OR terms for the actual output. I still have not used the feedback terms from /O1 and the other device outputs. They can be used in the tristate control term or in the actual logic equations.

In summary, this is the *software* structure of the combinational PALs:

IF (TRISTATE TERMS) OUTPUT =
INPUT TERMS
+ MORE INPUT TERMS

Any input can be used true or inverted. The inputs most easily form AND associations. OR associations of many AND subsets yield the device's remaining flexibility.

A Basic Example

The DPORT20 PAL from the DSI-020 design is a simple example. Figure 2 shows the portion of the schematic containing this PAL and its associated counter. The DPORT20 PAL implements a simple DMA controller. After arbitration, this PAL takes control of the bus from the 68020 CPU and produces signals that, to the peripherals, appear identical to those the 68020 would have generated. Thus, the host PC can access the RAM, DUART, etc., using this PAL as the timing controller.

The 74F161A counter is normally held reset until the hold acknowledge 8086 (HLDA86) becomes true. At this time it begins counting to start the DMA cycle. The clock signal for the 74F161A comes from a 25- or 40-MHz oscillator, depending on whether you are using a 12.5- or 20-MHz 68020. As shown in figure 3, each output sequences within 1 to 2 nanoseconds of each other, yielding a synchronized binary count to the PAL's inputs.

Initially, I'll concentrate on generating the AS (address strobe) signal. AS must go true between counts 2 and 9 inclusive (figure 3). In addition, as this signal is normally generated by the 68020, the output pin can be driven only during the DMA cycle. Since the 16L8 has a built-in inverter on each output, the AS signal is active low, meaning it will go low when it's true. This is denoted by a slash (/) before the AS symbol in the pin list. In text and equations, I will deal with AS, as well as the other active-low designators, as an active-high designator. The PALASM software knows from the pin list whether a designator is actually active low and handles the inversion automatically. This saves you from having to think in negative logic. Listing 1 shows the pin list and equation defining the AS signal.

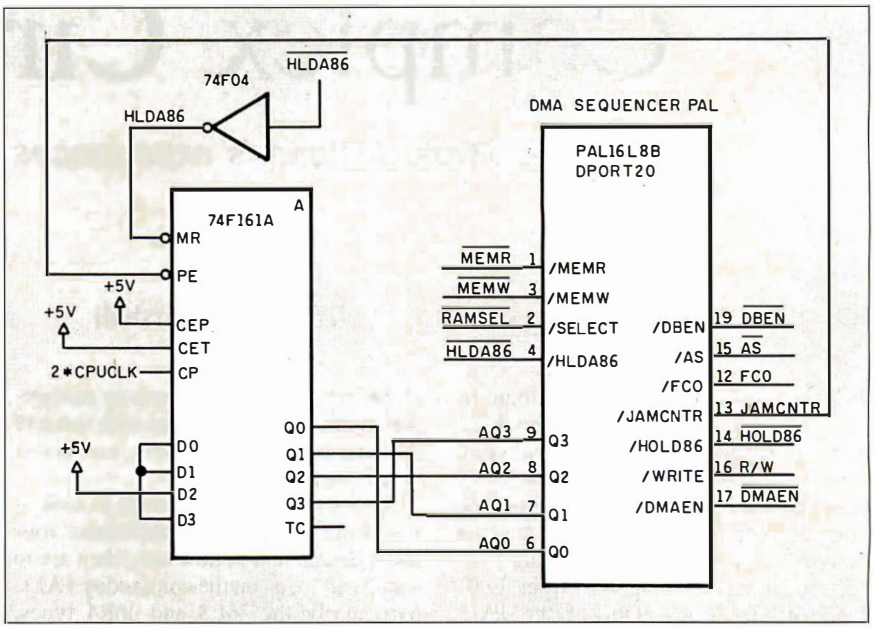

Figure 2: *The DPORT20 portion of the DSI-020 schematic.*

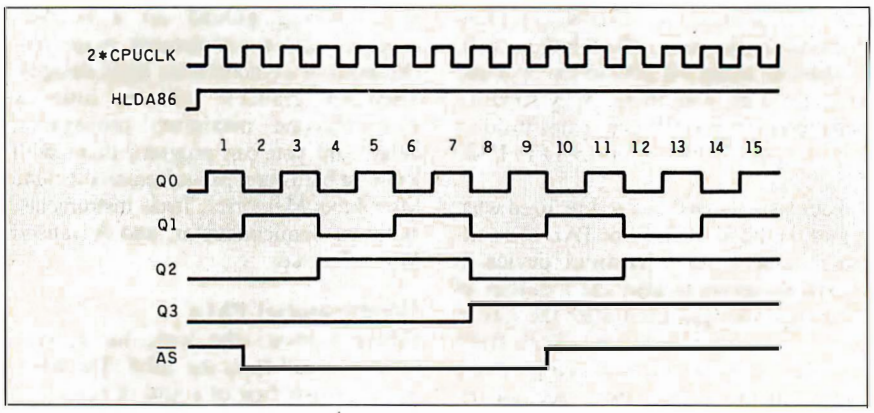

Figure 3: *Timing diagrams for the DPORT20.*

A 16L8 PAL has 20 pins. Pins 1 to 10 are named on the first line, pins 11 to 20 on the second. The symbol NC means no connection; this pin is not used. You should always leave some unconnected inputs and outputs in your PAL designs.

continued

They will be useful during debugging. The order in which you allocate the pins does not matter.

It is often convenient to reallocate the pins after you have begun routing of the circuit-board traces. Swapping PAL pins can significantly ease the routing task. On the 16L8, six of the pins are I/O pins that you can use as inputs instead of outputs. Be careful, however: Pins 12 and 19 are dedicated outputs and also have no feedback term.

Listing 1: *PALASM pin list and equation defining the AS signal for the DPORT20 PAL of the DSI-020 board.*

```
/MEMR /SELECT /MEMW /HLDA86 NC Q0 Q1 Q2 Q3 GND
NC /FC0 /JAMCNTR /HOLD86 /AS /WRITE /DMAEN NC /DBEN VCC

IF (HLDA86) AS = /Q3*/Q2*Q1*/Q0 ; 0010 is count=2
  + /Q3*/Q2*Q1*Q0               ; 0011 is count=3
  + /Q3*Q2*/Q1*/Q0              ; 0100 is count=4
  + /Q3*Q2*/Q1*Q0               ; 0101 is count=5
  + /Q3*Q2*Q1*/Q0               ; 0110 is count=6
  + /Q3*Q2*Q1*Q0                ; 0111 is count=7
  + Q3*/Q2*/Q1*/Q0              ; 1000 is count=8
  + Q3*/Q2*/Q1*Q0               ; 1001 is count=9
```

```
MONOLITHIC MEMORIES 20-PIN PALASM (tm) VERSION 1.7F
(C) COPYRIGHT 1983,1984 MONOLITHIC MEMORIES
PROGRAM LIMITS: 250 LINES 9999 CHARACTERS 150 VECTORS
WHAT IS THE SOURCE FILENAME (d:filename.ext) ?: testas1.pal
OUTPUT FILENAME - PRESS <ENTER> FOR NO OUTPUT FILE ?:
READING INPUT FILE

.............
PAL DESIGN FILE READ - 12 LINES 656 CHARS
ASSEMBLING INPUT FILE

.OUTPUT PIN NAME =  AS OUTPUT PIN NUMBER = 15
MINTERM IN LINE NUMBER 16
      + Q3 * /Q2 * /Q1 * Q0 ; 1001 IS COUNT=9
MAXIMUM OF 8 PRODUCT LINES ARE VALID FOR PAL16L8
 TOO MANY MINTERMS ARE SPECIFIED IN THIS EQUATION
```

Figure 4: *The file of listing 1 contains too many product terms to fit in a cell of a 16L8. PALASM flags this as an error.*

```
FUNCTION TABLE
Q3 Q2 Q1 Q0 /HLDA86 /AS  ;Pins to be tested, in this order
-------------------------------------------------------------
XXXX H Z  ;Check that the ouput goes tristate
LLLL L H  ;not asserted for 0
LLLH L H  ;or for 1
LLHL L L  ;but true from count of 2
LLHH L L  ;through
LHLL L L
LHLH L L
LHHL L L
LHHH L L
HLLL L L
HLLH L L  ;9
HLHL L H  ;deasserted at 10
HLHH L H  ;and should not come back
HHLL L H  ;through
HHLH L H
HHHL L H
HHHH L H  ;15
-------------------------------------------------------------
```

Figure 5a: *Function table for the DPORT20 PAL.*

The signals generated by the DPORT20 PAL of interest to this discussion are Q0, Q1, Q2, Q3, HLDA86, and AS. From the equation in listing 1, the AS output will be true during counts 2 through 9. To check this equation, I'll add a header and run listing 1 through the PALASM assembler as file TESTAS1.PAL. The assembler output in figure 4 shows a fatal error.

As I pointed out earlier, this cell of the 16L8 has a maximum of eight product lines (including the tristate control), and I have used nine. At this point, the theoreticians start mumbling magic words like "Karnaugh map" and "set theory," but you and I know the world isn't that complex. Look at the first two lines:

```
/Q3*/Q2*Q1*/Q0 ; count 2
 + /Q3*/Q2*Q1*Q0 ; count 3
```

What I am trying to say here is that if *either* Q0=0 (line 1) *or* Q0=1 (line 2), then, provided /Q3 *and* /Q2 *and* Q1 are true, the output should be true. Q0 is a "don't-care" factor and can be eliminated:

```
/Q3*/Q2*Q1 ; counts 2,3,          (1)
```

Similarly, lines 3 and 4 become

```
 + /Q3*Q2*/Q1 ; counts 4,5,       (2)
```

Lines 5 and 6 become

```
 + /Q3*Q2*Q1 ; counts 6,7,        (3)
```

Lines 6 and 7 become

```
 + Q3*/Q2*/Q1 ; counts 8,9,       (4)
```

Similarly, merging the new terms 1 and 3 results in

```
/Q3*Q1 ; counts 2,3,6,7
```

Thus, I can write

```
IF (HLDA86) AS =
   /Q3*Q1 ; counts 2,3,6,7
    + /Q3*Q2*/Q1 ; counts 4,5
    + Q3*/Q2*/Q1 ; counts 8,9
```

This time PALASM is happy and I now have four terms spare for later use during the debug phase.

Logic Simulation

PALASM software contains a logic simulator that lets you specify a series of conditions on the input pins and what you expect the outputs to do. You must construct a function table in the following format:

X on an input means to test with both a 1 and a 0.

continued

H means TRUE, HIGH, or 1; L means FALSE, LOW, or 0.
Z means high impedance (tristate).
Dashes (– signs) delineate the start and end of the test vectors.

Figure 5a shows the function table for the DPORT20 PAL. The output from PALASM's simulator is shown in figure 5b. The simulator prints a list of the logic states for all 20 pins. It places the values specified in the function table on the inputs and then checks that the outputs are correct. If it detects an error, the simulator tells you which output was in error.

Many PAL programmers use this simulation table to place voltages on the PAL pins during the test phase and ensure that the programmed part performs to your specifications. If the security fuse has been blown (to make it more difficult for somebody to copy your work), this method is the only way to verify correct operation of the programmed PAL.

You can also ask PALASM to produce a plot of the fuse map for you. Figure 6 is the plot of the AS output of the DPORT20 PAL.

When a term has been processed, it is shown alongside the fuse array that it represents. Masochists are thus able to check that the correct fuses have been blown. Note that the four unused terms have intact fuses, and it is possible to add extra terms to a PAL by reprogramming it. If you need to change a term in which the fuses have been blown, however, you can put your old PAL in the closest trash can.

This is probably a good time to mention the new ultraviolet erasable and electrically erasable CMOS programmable logic devices. [Editor's note: *The companies that make these parts refer to them as EPLDs and EEPLDs, but the author is talking about programmable-AND/fixed-OR architecture devices.*] The main advantage of these devices is that you can reprogram them many times and so save money in a development cycle. However, they are much more expensive than the bipolar types and usually require specialized programming hardware that far exceeds the total cost of all the bipolar PALs you will ever discard.

In addition, programmable-AND/fixed-OR devices have only recently become available in even medium-speed grades. You also need to consider that the erasable parts are generally singly sourced components.

Remember that the main advantage of the bipolar commodity PALs is their speed. Two of the PALs on the DSI-020 board are B parts (16L8B, with 15-ns maximum delay), and erasable devices would not have been fast enough. When you are designing with 40-MHz clocks, the extra 10-ns delay in the EPLD and EEPLD parts becomes critical.

One other output of the DPORT20 PAL that deserves mention is the JAMCNTR output. Its function is to ensure that the counter "dead-ends" at 15 rather than sequencing through from 0 again. Its control equation is simplicity itself:

```
IF (VCC) JAMCNTR = Q3*Q2*Q1*Q0
```

The IF(VCC) is optional; it means that the output is always enabled. The reason it deserves mention is because it is an excellent example of how a PAL can help you correct design errors. The initial design of the DPORT20 controller did not dead-end the counter, and the DMA controller kept cycling round and round. A

continued

```
MONOLITHIC MEMORIES 20-PIN PALASM (tm) VERSION 1.7F
(C) COPYRIGHT 1983,1984 MONOLITHIC MEMORIES
PROGRAM LIMITS: 250 LINES 9999 CHARACTERS 150 VECTORS
WHAT IS THE SOURCE FILENAME (d:filename.ext) ?: testas2.pal
OUTPUT FILENAME - PRESS <ENTER> FOR NO OUTPUT FILE ?:
READING INPUT FILE

................................
PAL DESIGN FILE READ -    32 LINES   1001 CHARS
ASSEMBLING INPUT FILE

.
E=ECHO  O=PINOUT   S=SIMULATE   F=FAULT   TESTING
P=PLOT  B=BRIEF    I=INTEL HEX  J=JEDEC   H=HEX
D=DOC   C=CATALOG  Q=QUIT
ENTER OPERATION CODE: s
DMA SEQUENCER                   ; as you see fit

   1 XXX1XXXXXXXXXZXXXX1
   2 XXX0X0000XXXXXHXXXX1
   3 XXX0X1000XXXXXHXXXX1
   4 XXX0X0100XXXXXLXXXX1
   5 XXX0X1100XXXXXLXXXX1
   6 XXX0X0010XXXXXLXXXX1
   7 XXX0X1010XXXXXLXXXX1
   8 XXX0X0110XXXXXLXXXX1
   9 XXX0X1110XXXXXLXXXX1
  10 XXX0X0001XXXXXLXXXX1
  11 XXX0X1001XXXXXLXXXX1
  12 XXX0X0101XXXXXHXXXX1
  13 XXX0X1101XXXXXHXXXX1
  14 XXX0X0011XXXXXHXXXX1
  15 XXX0X1011XXXXXHXXXX1
  16 XXX0X0111XXXXXHXXXX1
  17 XXX0X1111XXXXXHXXXX1
PASS SIMULATION
```

Figure 5b: *Result from PALASM simulation of the AS output using the function table of figure 5a.*

```
                11 1111 1111 2222 2222 2233
 0123 4567 8901 2345 6789 0123 4567 8901

32 ---- ---- -X-- ---- ---- ---- ---- ---- HLDA86
33 ---- ---- ---- ---- ---- X--- ---- -X-- /Q3*Q1
34 ---- ---- ---- ---- ---- -X-- X--- -X-- /Q3*Q2*/Q1
35 ---- ---- ---- ---- ---- -X-- -X-- X--- Q3*/Q2*/Q1
36 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
37 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
38 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
39 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX

LEGEND: X : FUSE NOT BLOWN (L,N,0) - : FUSE BLOWN (H,P,1)
NUMBER OF FUSES BLOWN = 119
```

Figure 6: *Fuse plot of the AS output of the DPORT20 PAL.*

few cut traces and assignment of this spare output fixed the problem.

Clocked or Synchronous Devices

One of the clocked cells from a 16R4 is shown in figure 7. Note that all eight OR minterms are available to the programmer since the tristate control and the clock come from dedicated pins. This is an inflexible structure, and many new PAL designs allow more programmed control of the flip-flops. Nevertheless, with ingenuity the ubiquitous 16R4, 16R6, and 16R8 are adequate for most synchronous designs.

DSI-32 HOLD PAL

The DSI-32 design uses a 16R4 to arbitrate DMA priorities. Two distinct subsystems, the dynamic RAM refresh controller and the 8086 dual-port DMA circuitry, can request the bus from the 32032. Since the 8086 requests are asynchronous with the system clock, the priority arbitration circuitry involves two levels of latching: first of the asynchronous requests, then of the arbitrated acknowledge outputs.

Figure 8 is the schematic of the HOLD PAL from the DSI-32 article. Note that the 32032 CPU clock signal, CTTL, goes into the clock pin for the flip-flops (pin 1) and also into the AND array. The HOLD86 input signals that the 8086 wants the 32032 bus, the RFIO that the refresh circuit wants it. HOLD requests the CPU to tristate the address and data buses. The 32032 asserts HLDA (hold acknowledge) to signal that the buses are free. HLDA86 is the acknowledge signal to the 8086 that it has won the bus; RFSHACK tells the refresh controller it is in control. I will discuss T1 later; POWERON and ADSO are unimportant. Listing 2 shows the PALASM file for the HOLD PAL.

Note that RFIOI and HOLD86I are two internal nodes whose outputs are not connected to the external world but merely fed back into the array. The first task is to synchronize the asynchronous input requests with the 32032 CPU clock (CTTL). This is done using the first two equations of listing 2.

The := symbol means "clock this data after the low-to-high transition of the clock." The RFIOI and HOLD86I outputs will now contain copies of the RFIO and HOLD86 inputs, sampled on the preceding positive clock edge. The third equation of listing 2 shows that when one of them is active, the corresponding subsystem is requesting the 32032 CPU bus. A combinatorial output is ideal for this.

When the 32032 indicates that the bus is free (HLDA signal is asserted), the PAL must resolve whether one or both DMA circuits are requesting the bus and arbitrate which one gains control. This is best done by using a "feed-forward" or "look-ahead" algorithm, as shown in the fourth equation of listing 2.

The first term of this equation says "if the 8086 is requesting the bus (HOLD86I is asserted) and the 32032 has released it (HLDA is asserted), then, provided both arbitration outputs (HLDA86 and RFSHACK) are currently inactive, give the bus to the 8086 on the next clock cycle by asserting HLDA86 to acknowledge that the 8086 has the bus."

The second term covers the case where the 8086 hold request, HOLD86I, arrives after the 32032 has released the bus to the refresh controller (HLDA and RFSHACK are true). In this case, the arbitrator will wait until the refresh request signal has gone away (RFIOI is inactive). The 8086 acknowledge, HLDA86, will then be asserted in the next clock cycle. Assume that RFSHACK will be removed on the same clock edge as its request (RFIOI).

After the 8086 has been acknowledged (HLDA86 is true), the third term will keep it true until the first clock cycle after the hold request (HOLD86I) has been removed. The refresh acknowledge arbitrator is shown in the fifth equation of listing 2.

Once again the first term checks to see if both acknowledge outputs are inactive. In this case, however, as the refresh is the lower-priority task, the HOLD86I signal is also checked to make sure that there is no simultaneous 8086 hold request until after the 8086 cycle has ended. This is necessary to ensure that the two acknowledge outputs are never asserted simultaneously. The second term again detects when the alternative DMA cycle is about to end, while the third term ensures that the output will be asserted for as long as the refresh request persists.

In case you are wondering, I did not dream these equations up in a flash of inspiration. It took many hours of doodling with pen and paper and some ideas from an application note called "An 8-bit Priority Interrupt Encoder with Registers" by Vincent Coli, *PAL Handbook* (3rd ed.), published by Monolithic Memories (MMI). This book is invaluable.

PALs to the Rescue

The DSI-32 prototypes did not work. The early 32032 parts that were shipped back in 1984 seemed to intermittently cease operation when the HOLD input was asserted at random. National Semiconductor suggested that if we synchronized the HOLD requests with the 32032 CPU cycles we might find a CPU state at which the 32032 would continue to operate correctly. What a task—the HOLD PAL now needed to keep track of which T state the CPU was executing and only assert HOLD selectively. The CPU has five types of T states: Tidle, T1, T2, T3, and T4 (see figure 9). The only way to distinguish them is by examining the bus signals and synchronizing them with CTTL, which is in phase with the 32032 CPU synchronization clock.

We felt that two of the signals in figure 9 could help us. The first was ADS, the

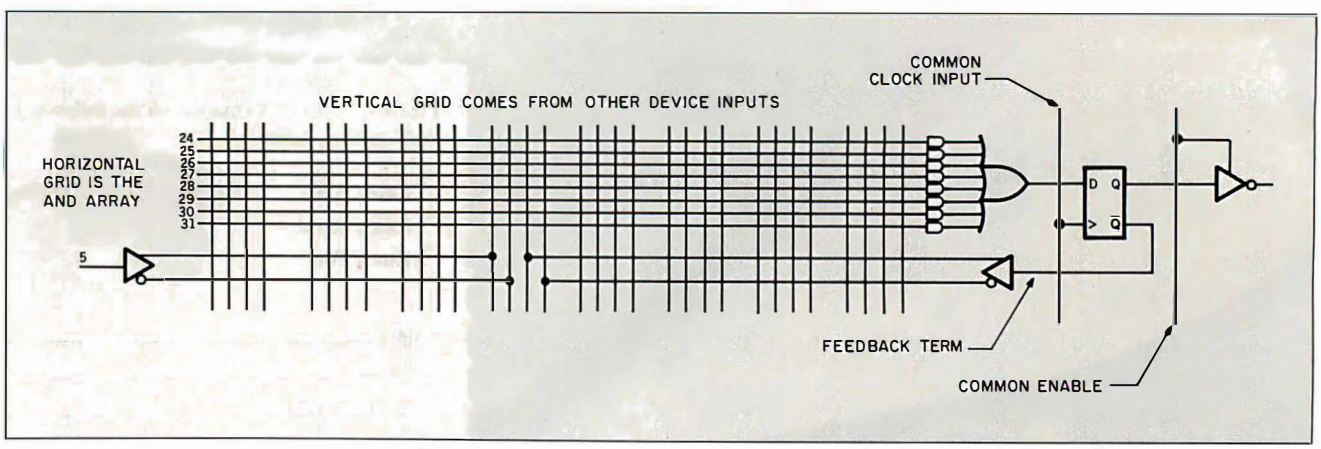

Figure 7: *Clocked logic cell from a 16R4 PAL.*

address strobe, and the second was TSO, the timing strobe output. Because ADS was always asserted in T1, if we could somehow latch it until the following clock edge we would know when T2 occurred. T2 turned out to be one of the two usable T states; T4 was the other, during which a HOLD request would be serviced correctly. ADS, however, is a very short signal. It must be stretched to the next positive edge of CTTL for the 16R4's flip-flops to recognize it.

The flip-flops in the 16R4 could not be used, as their clock is hard-wired from CTTL. A 74LS74 would have been an eminently good solution; however, the printed circuit board was already fabricated and we didn't have room for another chip. The final realization used one of the spare combinatorial outputs of the 16R4, named T1. It used the following equation, which says that when ADS pulses and CTTL is low, the T1 will be asserted and latched until CTTL goes high and tristates the T1 output:

```
IF(/CTTL) T1 = ADS + T1
```

Thus, the T1 output is a stretched copy of ADS, delayed sufficiently so that the 16R4's flip-flops can use it to latch bus requests during the CPU T state, T2. The T1 output needed a 2200-ohm pull-up resistor to work effectively at 10 MHz. We revised the equations that latch the asynchronous inputs to latch requests only when T1 is true or, if HOLD is already asserted, to keep holding and ignore synchronization with T2.

```
RFIOI := RFIO*T1 + RFIO*HOLD

HOLD86I := HOLD86*T1 +
    HOLD86*HOLD
```

The following equation causes a bus request to be sent to the 32032:

```
IF(VCC) HOLD = HOLD86I + RFIOI
```

The priority resolver was unchanged.

We added the ability for the aforementioned equations to synchronize with the T4 state by using the TSO signal:

```
IF(/CTTL) T1 = ADS + T1

RFIOI := RFIO*T1 + RFIOI*TSO
  + RFIO*HOLD

HOLD86I := HOLD86*T1 +
  HOLD86*TSO + HOLD86*HOLD

IF(VCC) HOLD =
  HOLD86I*/TSO*/T1
  + RFIOI*/TSO*/T1
  + HOLD86I*HOLD
  + RFIOI*HOLD
```

The HOLD PAL on the DSI-32 went through two more major changes. Terms were added to prevent HOLD requests, while the 32032 MMU was accessing the bus and the refresh acknowledge cycles were stretched to improve the RAS (row address strobe) precharge dynamic RAM timing parameter.

It is not exaggerating to say that the 16R4 HOLD PAL allowed Definicon to ship the DSI-32 several months earlier

continued

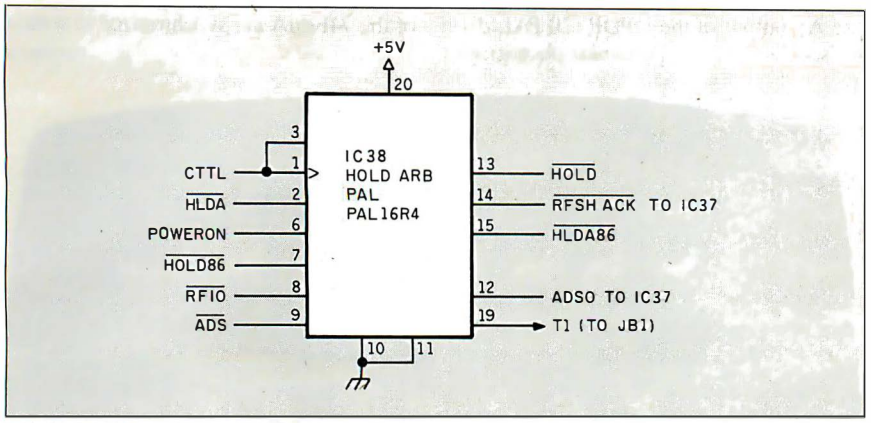

Figure 8: *The HOLD PAL portion of the DSI-32 schematic.*

Listing 2: *The PALASM file for the DSI-32 HOLD PAL.*

```
PAL16R4
IC2
(C) Copyright 1984,1985 Definicon Systems Inc.
Hold arbitration PAL for DSI-32 Rev B, TM, 12/5/84, first try
CLK /HLDA CTTL NC NC NC /HOLD86 /RFIO /ADS GND
/EN /ADSO /HOLD /RFSHACK /HLDA86 /RFIOI /HOLD86I NC NC VCC

RFIOI := RFIO
HOLD86I := HOLD86

IF(VCC) HOLD = HOLD86I + RFIOI

;First resolve the higher priority, the 8086
HLDA86 :=  HOLD86I * HLDA * /RFSHACK * /HLDA86
        + HOLD86I * HLDA * RFSHACK * /RFIOI
        + HOLD86I * HLDA86

;Resolve the refresh acknowledge
RFSHACK := RFIOI * HLDA * /RFSHACK * /HLDA86 * /HOLD86I
        + RFIOI * HLDA * HLDA86 * /HOLD86I
        + RFIOI * RFSHACK
```

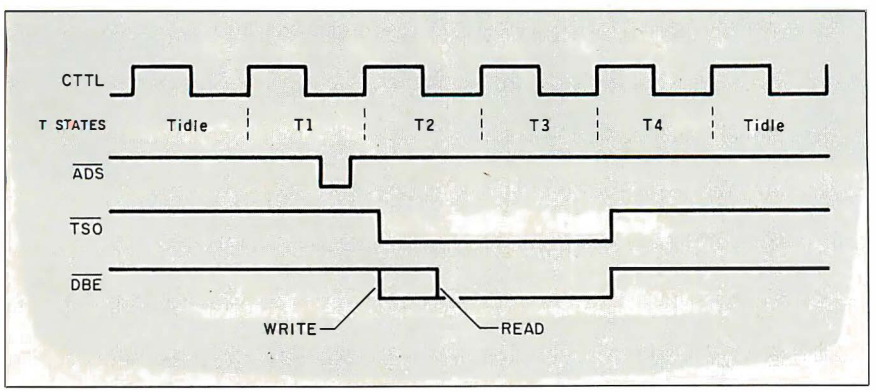

Figure 9: *Key 32032 CPU signals. The ADS and TSO signals were used to synchronize the HOLD PAL to CPU states T2 and T4.*

than a conventional 74LS00-series logic design would have.

What Can Go Wrong

Everybody tells you that PALs are designed so that all internal delays are matched and output glitches can't occur. Photo 1 is an oscilloscope photograph of the AS output of the DPORT20 PAL discussed earlier. This particular photograph was taken with a 20L10 PAL (the 24-pin equivalent of the 16L8).

If you examine the cell schematic for the 20L10, it's identical (except for fewer minterms) to that of the 16L8. Its performance, however, differs. On the top trace, you can see the Q0 output of the counter. Note the 9-ns-wide glitches. The outputs of the 74F161A are synchronized to within

continued

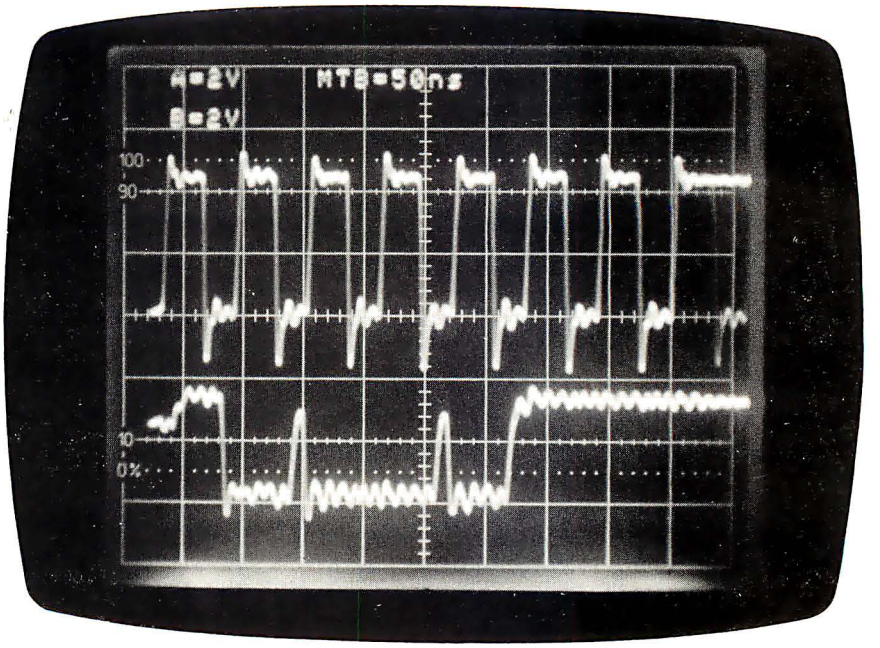

Photo 1: *The AS output of the DPORT20 PAL. The top trace is the Q0 input; the bottom trace is the AS output. Notice the 9-ns-wide glitches in the AS output between Q0 counts 4 and 5 and counts 8 and 9.*

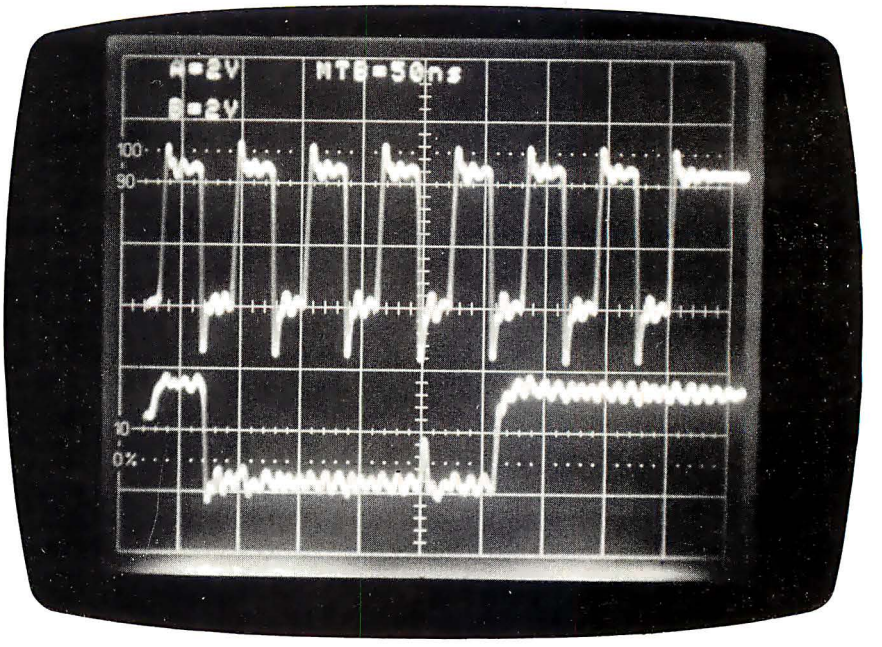

Photo 2: *Glitching of the AS output of the DPORT20 PAL implemented in a 16L8B device occurs only between counts 7 and 8.*

Listing 3: *The conversion of the PALASM version 1 file for the HOLD PAL into the format used by PALASM version 2.*

```
; created by PDSCNVT V2.21 - MARKET RELEASE (07-24-86)
; (C) - COPYRIGHT MONOLITHIC MEMORIES INC, 1986
 TITLE PDS CONVERSION FILE
 PATTERN EXAMPLE
 REVISION 1.00
 AUTHOR JOHN DOE
 COMPANY MONOLITHIC MEMORIES
 DATE 11/19/84
;PAL16R4
;IC2
;(C) Copyright 1984,1985 Definicon Systems Inc.
;Hold arbitration PAL for DSI-32 Rev B, TM, 12/5/84, first try

CHIP zzz PAL16R4 CLK /HLDA CTTL NC NC NC /HOLD86 /RFIO NC GND
     /EN NC /HOLD /RFSHACK /HLDA86 /RFIOI /HOLD86I NC NC VCC
EQUATIONS
RFIOI := RFIO          ;Latch the asynchronous inputs, first refresh request
HOLD86I := HOLD86      ;and now the access request from the 8086

HOLD = HOLD86I   ;immediately we get a request to tell the CPU
       + RFIOI
;Resolve the priorities, waiting for the HLDA before acknowledging
;First resolve the higher priority, the 8086
HOLD.TRST = VCC
HLDA86 :=  HOLD86I * HLDA * /RFSHACK * /HLDA86
          + HOLD86I * HLDA * RFSHACK * /RFIOI
          + HOLD86I * HLDA86

 ;Then resolve the refresh acknowledge
 RFSHACK := RFIOI * HLDA * /RFSHACK * /HLDA86 * /HOLD86I
           + RFIOI * HLDA * HLDA86 * /HOLD86I
           + RFIOI * RFSHACK

 ; FUNCTION
;SIMULATION
;-----------------------------------------------------------------------------
;SETF EN /HOLD86 /RFIO /RFIOI /HOLD86I /HOLD /HLDA
;CLOCKF CLK
;clock everything inactive

;SETF EN /HOLD86 /RFIO /RFIOI /HOLD86I /HOLD /HLDA /HLDA86 /RFSHACK
;CLOCKF CLK
;clock everything inactive

;SETF EN /HOLD86 RFIO RFIOI /HOLD86I HOLD /HLDA /HLDA86 /RFSHACK
;CLOCKF CLK
;RFIO recognized

;SETF EN /HOLD86 RFIO RFIOI /HOLD86I HOLD HLDA /HLDA86 RFSHACK
;CLOCKF CLK
;and acknowledged
```

the resolution of the scope (1 to 2 ns). And those glitches are being generated by the difference between the low-to-high and high-to-low propagation delays of the logic internal to the PAL.

Photo 2 is a scope photograph of the output of a 16L8B, showing that the glitch in this case is a lot faster but still a problem. You can remove the remaining glitch, between counts 7 and 8, by allocating an unused output, say CNT7:

```
IF (VCC) CNT7  =  /Q3*Q2*Q1*Q0

IF (HLDA86) AS  =
   /Q3*Q1 ; counts 2,3,6,7
 + /Q3*Q2*/Q1 ; counts 4,5
 + Q3*/Q2*/Q1 ; counts 8,9
 + CNT7
```

The glitch occurring at the input transition from 7 to 8 is masked by the delay in the output buffer for the CNT7 term.

So be warned. When you have outgrown the capabilities of the 16L8 and 16R4, be sure to evaluate the advantages and disadvantages of the PAL families you choose.

Obtaining PALASM

I have been using the syntax of PALASM version 1. MMI has released PALASM version 2.21, which contains many enhancements and support for a range of PALs with advanced architectures. Unfortunately, it's much more tedious to write code using its new syntax. Listing 3 shows the PALASM 2.21 representation of the HOLD PAL file we discussed earlier in listing 2. This .PDS file was created by running the .PAL file from PALASM 1 through a conversion utility, PDSCNVT. I can probably put up with the representation for the logic, but the tedium of keying in all those simulation vectors is something I can do without.

PALASM 1 is in the public domain; for

```
;SETF EN /HOLD86 RFIO RFIOI /HOLD86I HOLD HLDA /HLDA86 RFSHACK
;Check DIAGON function

;SETF EN /HOLD86 /RFIO /RFIOI /HOLD86I /HOLD /HLDA
;CLOCKF CLK
;clock everything inactive

;SETF EN /HOLD86 /RFIO /RFIOI /HOLD86I /HOLD /HLDA /HLDA86 /RFSHACK
;CLOCKF CLK
;clock everything inactive

;SETF EN HOLD86 /RFIO /RFIOI HOLD86I HOLD /HLDA /HLDA86 /RFSHACK
;CLOCKF CLK
;HOLD86 recognized

;SETF EN HOLD86 /RFIO /RFIOI HOLD86I HOLD HLDA HLDA86 /RFSHACK
;CLOCKF CLK
;and acknowledged

;SETF EN HOLD86 /RFIO /RFIOI HOLD86I HOLD HLDA HLDA86 /RFSHACK
;Check DIAGON function

;SETF EN /HOLD86 /RFIO /RFIOI /HOLD86I /HOLD /HLDA
;CLOCKF CLK
;clock everything inactive

;SETF EN /HOLD86 /RFIO /RFIOI /HOLD86I /HOLD /HLDA /HLDA86 /RFSHACK
;CLOCKF CLK
;clock everything inactive

;SETF EN HOLD86 RFIO RFIOI HOLD86I HOLD /HLDA /HLDA86 /RFSHACK
;CLOCKF CLK
;both arrive at once

;SETF EN HOLD86 RFIO RFIOI HOLD86I HOLD HLDA HLDA86 /RFSHACK
;CLOCKF CLK
;8086 wins

;SETF EN /HOLD86 RFIO RFIOI /HOLD86I HOLD HLDA HLDA86 /RFSHACK
;CLOCKF CLK
;8086 goes away, hold active

;SETF EN /HOLD86 RFIO RFIOI /HOLD86I HOLD HLDA /HLDA86 RFSHACK
;CLOCKF CLK
;rfsh wins now
;-------------------------------------------------------------------------------
;DESCRIPTION
;The HOLD PAL arbitrates between two possible sources of bus requests to
;the 32032, refresh and PC bus access.
```

information on how to obtain version 2.21, you can contact MMI. I hope that somebody will take the source code and write a good simulator. Note that the fourth edition of the *Programmable Logic Handbook* is written for PALASM 2.21; the third edition is in PALASM 1 syntax.

I have obtained a copy of the old PALASM source code 1.3 written in FORTRAN-77. The compiled executable copies of later versions (1.7f) are available for the IBM PC; the FORTRAN source will be of most value to those readers interested in how PALASM works and those without access to IBM PCs. [Editor's note: *The programs are available from Trevor Marshall's Thousand Oaks Technical Database, (805) 492-5472 or (805) 493-1495, in the C:\PALASM subdirectory. They are also available on disk, in print, and on BIX (see the insert card following page 424 for details), or on BYTEnet (see page 4).*]

Summary

PALs offer a circuit designer the chance to overcome the inflexibility of hardware designs. This results in fewer changes to the circuit board during the debug phase and easier field upgrade during the operational phase of a product's life. As the variety of PAL configurations proliferates and the cost drops, it becomes increasingly difficult to justify the continued use of discrete logic devices. ■

ACKNOWLEDGMENTS

I wish to thank Definicon Systems Inc. for creating the environment in which it was possible to develop these devices and for permission to use examples of the proprietary PAL codes from our coprocessor products. George Scolaro worked with me on the DSI-32 PALs, and Chris Jones on the DSI-020 PALs.