What Next for PAL-DEVICES —
The Second Generation Challenge

By

David Winkel
Computer Science Department
Indiana University
Bloomington, IN 47405

TECHNICAL REPORT NO. 188

What Next for PAL-DEVICES —
The Second Generation Challenge

by

David Winkel
Indiana University

Revised: May, 1986

What Next For PAL-DEVICES — The Second Generation Challenge

Success in Silicon is the reward for designing parts that are produceable, match system needs, and are widely used.

First generation PAL-DEVICES meet these goals by replacing random logic while reducing parts count, board area, inventory, and system power. More important, programmability ensures easy field updates and concealment of proprietary logic. These unique attributes ensure their dominance in this area for the foreseeable future. What next?

The challenge facing designers of second generation PAL-DEVICES is finding something more than logic that needs customizing. How about programmable data paths and controllers? This is the domain of conventional bit slices widely used in CPUs, device controllers, and special purpose systems. PAL-DEVICES are faster, have more general output structures, and can do parallel operations, while retaining customizability and concealment of proprietary logic.

This technical report explores bit slice data paths in detail. PAL implementation of control logic demonstrates similar advantages and is treated in references 2 and 3. Bit slices are popular because they give the appearance of customizability although their internal architecture is fixed. Data is taken from two source registers, sent through an ALU, and stored in a destination register. Microcode is used to customize operations by issuing properly sequenced commands to build more complex operations. The success of bit slice data paths validates this approach but we have to learn to live with their limitations. For example, only one operation can be done per microcycle whereas traditional hardware achieves speed by overlapping operations. Another bothersome restriction results from the fixed data path. Often we would like to send out a register's contents to an external memory and this can't be done unless an external "shadow register" latches the value. Also, we waste an entire microcycle for the transfer. This irritating restriction shows up in the memory address structure of most computers; we want to send out the program counter (PC) register as a memory address (MA) at the same time we use the arithmetic unit to increment PC.

$$(PC \rightarrow MA, PC+1 \rightarrow PC)$$

Large registered PAL-DEVICES overcome many of these limitations by having customizable data paths between any combination of registers. In the above example we would dedicate one data path for $PC \rightarrow MA$ and a separate path to $PC+1 \rightarrow PC$ and execute them in parallel. For N registers, all $N*(N-1)/2$ data paths are in the fuse matrix waiting to be programmed! This flexibility is not exploited even in conventional architectures because it takes an overwhelming number of wires. Bus oriented systems, or in bit slices a single fixed data path, are used to cope with this complexity. Of course the same complexity is hidden inside the PAL but it is now the manufacturer's responsibility to produce it; the designer might as well use it. The user sees this large matrix in terms of logic equations which are converted to burn patterns by logic assemblers or compilers, thus hiding internal details.

When To Use a PAL Bit Slice

If you need speed, complex data paths, or overlapped operations a PAL is the answer. Program it as a generalized bit slice. Smaller PAL-DEVICES (32VX10) will hold one bit; larger devices can accomodate 4 to 8 bits. There is one restriction, arithmetic operations are best done with external arithmetic units to speed up carry propagation across a word. This may even be desirable since the PAL can easily be programmed to present arbitrary pairs of registers to the arithmetic unit while purely logic operations are handled in parallel inside the PAL.

The PAL must support common system functions and conventional protocols:

1) hold register contents until explicitly modified
2) have multiple outputs
3) have multiple inputs
4) have bidirectional I/O
5) be vertically microcoded (set up multiple data paths in response to a few command bits)

All the larger registered PAL-DEVICES support functions 2–5. However, as shown later only PAL-DEVICES with an exclusive OR in the register path support function 1 and this is the most important protocol in any general register architecture. It simply must be available. The reason for this protocol is not obvious unless you remember that most registers are seldom modified. It is much simpler to decode only those cases where registers are modified and leave unchanged data as a *default*. If you have to also decode for unchanged values you run out of resources.

This protocol is not unique to PAL-DEVICES. The register file of conventional bit slices (for example, the 2903) also use it. In MSI designs the enabled D flip-flop supports this "hold until I say otherwise" philosophy and is a favorite chip with experienced designers. 74LS398 and 74LS399 chips are common MSI incarnations of this idea. It is also used in shift registers such as the MMI LS498. The JK flip-flop can implement this function but at the expense of controlling J and K separately. All these devices feed the system clock directly to registers and thus avoid clock skew problems. In any event the user is prevented from gating clocks on individual flip-flops because of internal PAL structure. This enforced discipline guides a designer towards good engineering practice.

XOR PAL-DEVICES (32VX10) provide "hold until load new" if you exploit some of the little known properties of the XOR, whereas the older OR structures (22V10) will give the function only with great difficulty.

The Problem With Or Structures — a Simple Example

Consider a subset of three 22V10 registers, R0–R2, and try to move data between them:

R2 → R1	when L=true, hold otherwise
R2	hold always

If we dedicate an input pin to L its input buffer will send L and /L to the fuse matrix so it is easy to implement the required equations:

D1= L*Q2 + /L*Q1 (recirculate R1 on /L, load Q2 on L)
D2= Q2 (recirculate always)

Since we often want to do as many as 30 unique types of data and logical transfers we can't dedicate an input pin for each of them without becoming pin limited. We are forced to send in commands in encoded form, where n bits encode 2**n different operations. For our example we explore the problems caused by 3-bit commands; they become even more unmanageble with 4 or 5 bits.

Suppose we encode the above load command with the pattern I2,I1,I0 = 101. Now we are forced to generate /L using DeMorgan's law and the AND-OR structure of the PAL:

/L = /I2 + I1 + /I0

and this uses precious OR inputs at a frightening rate. The equations become:

D1= I2*/I1*I0*Q2 + /I2*Q1 + I1*Q1 + /I0*Q1 Eq.1
D2= I2*/I1*I0*Q2 + (all other terms that leave R2 unchanged)*Q2 Eq.2

This is bad enough but it gets worse. Suppose command 101 remains the same and 010 transfers R0 → R1 then:

D0= (all terms that leave R0 unchanged)*Q0 Eq.3
D1= I2*/I1*/I0*Q2 + /I2*I1*/I0*Q0 + /I1*/I0*Q1 + /I2*I0*Q1 Eq.4
+ I1*I0*Q1 + I2*I1*Q1
D2= as in Eq. 2

We have used up 6 terms in the D1 OR structure with this trivial example. The last four terms in D1 are the expansion of /(L1 + L2) where L1 = 101 and L2 = 010. Reasonable command widths are out of the question because there is no automatic default for the hold case. If there were, the equation for D1 would simplify to the first 2 terms. As we see next, the XOR eliminates the decode for /L and gives a natural default hold.

Xtra Power From the XOR

So how does it work?

We are trying to synthesize the standard enabled D excitation table:

TABLE 1

LOAD	NEW DATA	Q	Q after clock
0	-	0	0
0	-	1	1
1	0	-	0
1	1	-	1

if N= NEW DATA

D= L*N + L*Q

This is usually synthesized with a mux in MSI devices because it implicitly treats hold (don't load) as the default case (2), Fig. 1a; but can also be implemented with an XOR, Fig. 1b.

Fig. 1a

Fig. 1b

Find the truth table for Y such that Y XOR Q will implement an enabled D excitation table. The answer is:

TABLE 2

L	N	Q	enabled D output		Y
0	0	0	0	hold	0
0	0	1	1	when	0
0	1	0	0	L=0	0
0	1	1	1		0
1	0	0	0	load N	0
1	0	1	0	when	1
1	1	0	1	L=1	1
1	1	1	1		0

where Y is chosen such that Y XOR Q will = enabled D output.

$$Y = L * /N * Q + L * N * /Q$$

SINCE THERE ARE TERMS IN L ONLY,
THIS SOLVES THE LOAD PROBLEM!

The default case is nicely handled since there are load terms only. If L is false, Y = 0, and Q XOR 0 = Q, which recirculates the old Q until L becomes true. Further, multiple load terms are easily implemented by a straightforward generalization of the Y equation. Let L1 be the load condition for new data N1 and L2 for N2 then:

Y= L1*/N1*Q + L1*N1*/Q + L2*/N2*Q + L2*N2*/Q Eq.5

Let us add a new command, 010, that transfers R0 to R1.

Load R1= command 101 for transfer from R2 (L1=I2*/I1*I0) Eq.6
or command 010 for transfer from R0 (L2=/I2*I1*/I0)

Load R0 = default hold
Load R2 = default hold

For the above example the equations are:

Y0 = 0 (default recirculate of Q0) Eq.7
Y1 = I2*/I1*I0*/Q2*Q1 + I2*/I1*I0*Q2*/Q1 Eq.8
/I2*I1*/I0*/Q0*Q1 + /I2*I1*/I0*Q0*/Q1
Y2 = 0 (default recirculate of Q2 Eq.9

But there is still more flexibility hidden in the XOR! We can implement any logic function, as shown in the theory section. Looking ahead we see that clearing a register can be done by feeding Q into the XOR input. Setting is done by sending /Q into the XOR.

Let's add two new commands to our example, 000 will clear R1 and 001 will set it. The Y1 equation has two new terms added to it:

$$Y1 = \text{Eq.8} + /I2*/I1*/I0*Q1 + /I2*/I1*I0*/Q1 \qquad \text{Eq.10}$$

Of course a given command need not affect all registers in the same way. In addition to the above actions of command 000 let it also simultaneously clear R0, R1 and set R2. The equation for Y1 is unchanged, Y0 and Y2 become:

$$Y0 = /I2*/I1*/I0*Q0 \qquad \text{Eq.11}$$
$$Y2 = /I2*/I1*/I0*/Q2 \qquad \text{Eq.12}$$

This is an illustration of our requirement that bit slices be vertically microcoded with given command patterns simultaneously affecting all data paths and register contents, but each individually programmable.

Underlying Theory Of The XOR Register Feedback Path

It is easy to show that there can be only 4 possible functions, F, that allow a general flip-flop excitation table to be implemented under the assumption that the innermost atomic storage element is a D flip-flop. We must be able to set or clear the flip-flop regardless of its initial state. This is clearly a minimum requirement for loading new data. Also, we must permit holding old data in order to meet the protocol for register storage. The 4 cases are:

TABLE 3

Q N	Din_1	Din_2	Din_3	Din_4
0 0	0	0	1	1
0 1	1	1	0	0
1 0	0	1	0	1
1 1	1	0	1	0
	$Din_1 = N$	$Din_2 = N\ XOR\ Q$	$Din_3 = N\ XNOR\ Q$	$Din_4 = NOT\ N$
	Fig. 3a	Fig. 3b	Fig. 3c	Fig. 3d

(Q, N are available in the fuse matrix feeding the OR)

The correspondence with commercial PAL structures is evident. The XNOR structure has the same functionality for generating logic outputs as the XOR so there is no need to consider it further. In any event you can implement it with the XOR structure by using the Boolean identity:

A XNOR B = A XOR (NOT B) = (NOT A) XOR B

The NOT N case is again similar to the first and can dismissed because it has the same functionality. Also, there are no commercial PAL-DEVICES using the NOR structure.

Fig. 3b is capable of generating any function Din of N and Q, Din(N,Q) so there is no penalty in using only PAL-DEVICES with the XOR structure. To show this we examine all possible logic functions of two variables, N and Q, and show how they can generated by generating the proper intermediate function Y of N and Q, Y(N,Q), such that Din = Q XOR Y

TABLE 4

col# NQ	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	00	00	00	00	00	00	00	00	11	11	11	11	11	11	11	11
01	01	01	01	01	10	10	10	10	01	01	01	01	10	10	10	10
10	00	00	11	11	00	00	11	11	00	00	11	11	00	00	11	11
11	01	10	01	10	01	10	01	10	01	10	01	10	01	10	01	10

1st column = Din (desired logic function)
2nd column = Y (such that Y XOR Q = Din)

We have already used several results from this table.

To clear a flip-flop we must feed it a zero	F(N,Q)=0 then	Y = Q	(col 0)
To set a flip-flop we must feed it a one	F(N,Q)=1 then	Y =/Q	(col 15)
To load a flip-flop we must feed it an N	F(N,Q)=N then	Y = Z	(col 6)

(where Z = N XOR Q)

If you want to load a flip-flop with some function of new data, N, and the current value of Q, look in the first column of the table until you find the desired logic function of N,Q and use the OR structure of the PAL to generate the corresponding intermediate function, Y. The PAL automatically forms the XOR of Y and Q, feeds it to the D input, and Q will have the desired value after the next clock edge.

For example suppose we take our previous set of three registers and add a new command 100 that:

loads R0 with the OR of R0 , R2	(col 7, Table 4)
loads R2 with the AND of R0 , R2	(col 1, Table 4)

The equation for Y1 remains unchanged while Y0, Y2 become:

Y0 = Eq.11 + I2*/I1*/I0*Q2*/Q0 Eq.13
Y2 = Eq.12 + I2*/I1*/I0*/Q0*Q2 Eq.14

Do you wish to make a toggle flip-flop? Find F = /Q (col 10) and see that Y = 1.

A JK flip-flop is slightly harder since we have to provide for all 4 lines in its excitation table.

TABLE 5

J	K	Q after clock		
0	0	Q	hold	Y=0
0	1	0	clear	Y=Q
1	0	1	set	Y=/Q
1	1	/Q	toggle	Y=1

If you plot this on a K-MAP it simplifies to

Y = J*/Q + K*Q

So it takes two OR inputs, one for J, one for K, which is not surprising.

The possibilities are endless. Consider the set dominant flip-flop favored by early Xerox computer designers. It is like the JK except when J=K=1, then J overrides K. Better names are JS (J strong), KW (K weak), and the excitation table is:

TABLE 6

JS	KW	Q after clock
0	0	hold
0	1	0
1	0	1
1	1	1

Y = JS* + /JS*/KW*Q

(easily implemented using two OR inputs)

Occasionally one finds the XNOR function useful, especially if you are perverse enough to consider NAND, NOR logic more natural than the straightforward AND, OR logic provided by standard PAL-DEVICES. Proceed by making a table like Table 4 except you look for the value of Y such that Y XNOR Q provides the desired function F. You gain no functionality but the Y functions for NAND, NOR are simpler than for AND, OR. The reverse is true in Table 4.

Proof Of Minimality Of XOR Feedback Path

At first sight it appears wasteful to have two feedback terms as input to the OR gate developing the intermediate Y term to load new data into the register, (Table 2). The width of OR's is a limited PAL resource and it would be nice to conserve it. It is an easy mathematical exercise to prove that at least two terms are required regardless of the type of storage flip-flop or the feedback function. This is shown in a separate technical report (4). This proves the XOR is as efficient as any possible feedback element. Since it does all desired operations with two OR terms we need look no further.

Although not a proof, we can see that these structures also require two terms:

1) MUX	Fig. 1a	separate LOAD and N terms
2) JK	Table 5	separate J and K terms
3) SET DOMINANT	Table 6	separate JS and KW terms

Practical Application Rules

1) Dedicate 2 registers as output ports to feed an arithmetic unit. The AND-OR-XOR logic can select any of the remaining registers or input pins as sources for the arithmetic unit.
2) Dedicate one input to the output of the arithmetic unit which can then be distributed to any register.
3) Choose input/output pins to match system requirements.
4) Dedicate sufficient pins to send commands, L, into the bit slice.
5) Generate proper Y outputs using AND-OR logic to feed the XOR gate. (N is new data, either an input or another register).

a) clearing a register	Y= L*Q
b) setting a register	Y= L*/Q
c) loading a register	Y= L*N*/Q + L*/N*Q
d) ANDing a register with another register or input	Y= L*/N*Q
e) ORing a register with another	Y= L*N*/Q
f) XORing a register with another	Y= L*N
g) XNORing a register with another	Y= L*/N
h) NOTing a register	Y= L
i) holding a register	Y= 0

Summary

We have seen that:

A) The XOR feedback path enhances PAL structures
   1) Normal register "hold until load" protocols are easily implemented.
   2) Full logic capability is preserved.
   3) Normal or exotic flip-flops are easily synthesized.
   4) Set and Clear functions can be done naturally without the special hardware, provided on commercial PAL-DEVICES.

B) The newer registered PAL-DEVICES make excellent bit slices
   1) Fully parallel data transfers are not only possible but natural.
   2) The availability of three-state outputs supports normal system protocols for I/O
   3) They are faster than normal bit slices because of their flexible architecture.

C) To fully exploit the system power of the XOR PAL-DEVICES requires only simple extensions of traditional PAL techniques.

References

1) PAL Handbook, 3rd ed., Monolithic Memories, 2175 Mission College Blvd., Santa Clara Cal., 95050.
2) "The Art of Digital Design", F. Prosser and D.Winkel, 2nd ed, Prentice Hall, Englewood Cliffs, N.J., 1987. See chapter 3 for enabled D flip-flop.
3) I.U. Computer Science Tech Report (proposed), "PAL-DEVICES for Efficient State Machine Controllers" , David Winkel, 1986
4) I.U. Computer Science Tech Report (proposed), "Efficient PAL Storage Structures", D. Winkel, 1986
5) I.U. Computer Science Tech Report (proposed), "Comparison of Different Benchmark Implementations of the PDP8", D.Winkel, 1986
6) The present technical report also appears as a Monolithic Memories Application Note.