In the preceding section we showed how ABEL provides an easy way to specify counter behavior. However, this section describes counter excitation equations in more depth. Understanding this material is necessary if you want to design large counters using X-series PLDs, described later in this section.

* 10.5.1 Counter Behavior

Like most MSI counters, a PLD-based counter is a *synchronous* counter, that is, all of its output bits change at the rising edge of the clock. The basic logic function for one output bit of a synchronous binary up counter is, in words, “Change state if counting is enabled and all the lower-order bits are 1.” (To get a binary *down* counter, just change 1 to 0 in this statement.) A corresponding ABEL equation is written below for bit 4 of a synchronous binary up counter:

```
Q4 := /Q4 * (CNTEN * Q3 * Q2 * Q1 * Q0)   " Complement
    +  Q4 * /(CNTEN * Q3 * Q2 * Q1 * Q0); " Preserve
```

The first line complements Q4 if all of the conditions for changing state are true. Otherwise, the second line preserves the state of Q4. This equation requires six product terms to realize in any of the sequential PLDs that we’ve studied so far, assuming that /Q4 is assigned to a registered output pin. Using ABEL’s XOR operator, :+:, we can write the equation in just one line:

```
Q4 := Q4 :+: (CNTEN * Q3 * Q2 * Q1 * Q0); " Toggle if enabled
```

However, after expansion, the equation still requires a total of six product terms.

Other counter features, like synchronous loading and clearing, can be added quite easily to the preceding equation:

```
Q4 := /CLR * /LD * (Q4 :+: (CNTEN * Q3 * Q2 * Q1 * Q0)) "Toggle
    + /CLR * LD * D4; " Load
```

The expanded and reduced equation has eight product terms, which just fits on a registered output of a 16V8R or 16Rx. Higher-order counter bits (Q5, Q6, . . .) require even more product terms, and therefore do not fit in the PLDs that we’ve discussed so far. So, before we look at other modifications to the basic counter equations, we’ll introduce a series of PLDs that were specifically designed to reduce the product-term requirements of counter logic.

* 10.5.2 X-Series PLDs

PAL20X8
PAL20X4
PAL20X10
PAL20L10

The “X series” of PLDs uses XOR gates to facilitate the design of binary counters. This series includes the *PAL20X8*, whose logic diagram is shown in Figure 10–13, and three similar devices. The other devices have a different allocation of registered and combinational outputs, as shown in Table 10–10 and Figure 10–14. The *PAL20L10* has *no* registered outputs or XOR gates, but is included in the se-

Copyright © 1994 by John F. Wakerly

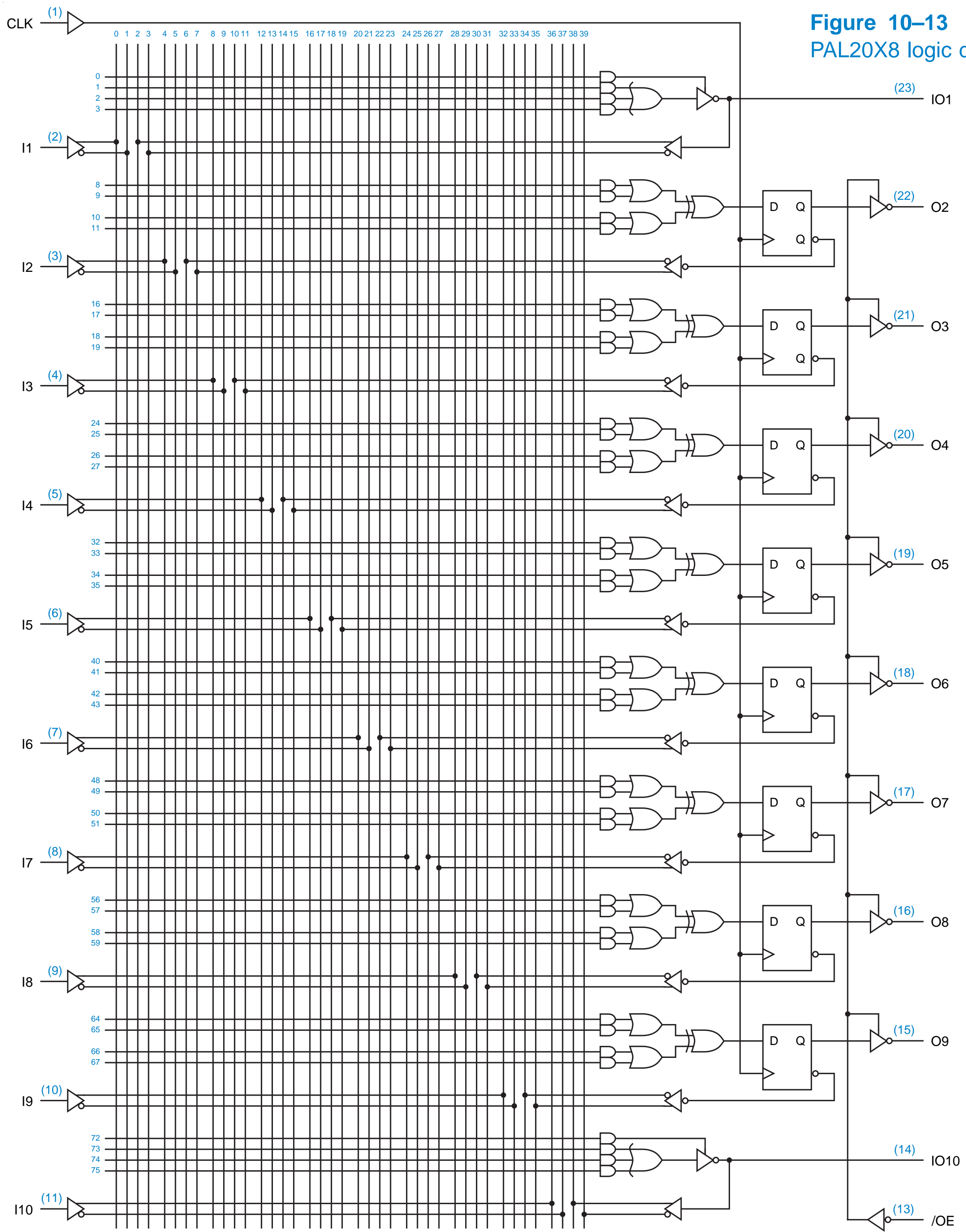

Figure 10–13 PAL20X8 logic diagram.

Copyright © 1994 by John F. Wakerly

Table 10–10 Characteristics of "X-series" PLDs.

Part number	Package pins	AND-gate inputs	Inputs to AND array: Primary inputs	Inputs to AND array: Bidirectional comb. outputs	Registered outputs	Combinational outputs
PAL20L10	24	20	12	8	0	2
PAL20X4	24	20	10	6	4	0
PAL20X8	24	20	10	2	8	0
PAL20X10	24	20	10	0	10	0

GAL20XV10

ries because it has the same AND-OR array as the other devices. Lattice Semiconductor also manufactures a GAL device, the *GAL20XV10*, that can be configured to emulate any device in the table, and more; each output can be individually configured to be combinational or registered, as in the 16V8 and 20V8.

Each X-series output has only four product terms available, which often leaves designers starving for more. The combinational outputs use one of the four terms to control the three-state enable, while the registered outputs perform the XOR of two ORed pairs of product terms. In ABEL, the generic equation for a registered output is

```
OUT := (P1 + P2) :+: (P3 + P4);
```

where P1–P4 are the four product terms.

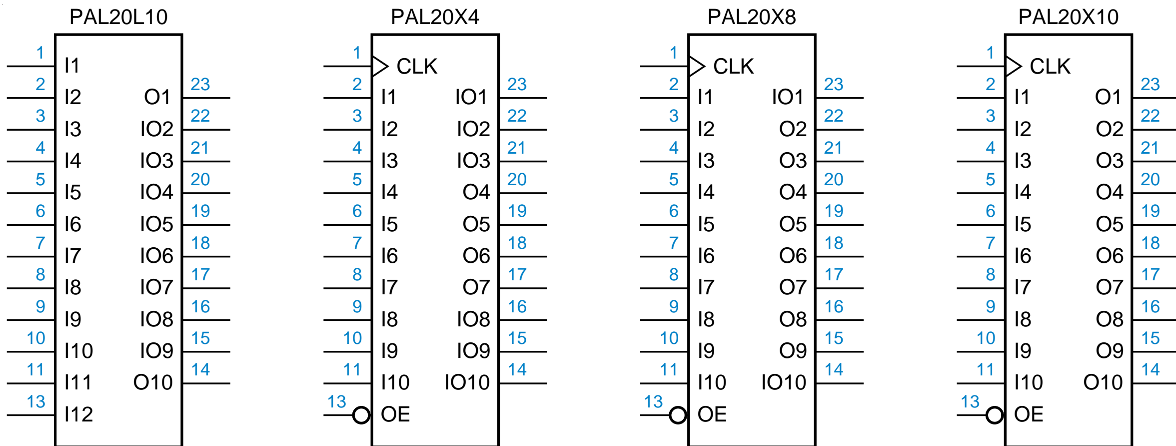

Figure 10–14 Traditional logic symbols for X-series PLDs.

Copyright © 1994 by John F. Wakerly

* 10.5.3 Counter Design with X-Series PLDs

With a little factoring, the equation for output Qi of a binary synchronous up counter maps nicely into the structure of X-series PLDs:

```
Qi := (  (/CLR * /LD * Qi)                       " Hold
       +   0 )   :+:                             " Unused
      (  (/CLR * /LD * CNTEN * Qi-1 * ... * Q0)  " Toggle
       + (/CLR *  LD * Di) );                    " Load
```

The same basic equation works for high-order bits as well as low-order ones, with no increase in the number of product terms. In fact, one product term is not even needed (the explicit 0 on the second line).

The Qi signal in the preceding equation is active high at the flip-flop input and output, and therefore active low at the device pin. To obtain an active-high pin output, the flip-flop output must be active-low, /Qi, so the equations must be modified:

```
/Qi := (  (/PR * /LD * /Qi)                      " Hold
        +   0 )   :+:                            " Unused
       (  (/PR * /LD * CNTEN * Qi-1 * ... * Q0)  " Toggle
        + (/PR *  LD * /Di) );                   " Load
```

Note that the functionality of what was the CLR input is now different, since clearing the flip-flop *presets* the active-high output pin to 1. The unused product term can be employed to obtain a true CLR function (see Exercise 10.22). The equations for down counters are identical, except that the toggle term must detect the state `/Qi-1*...*/Q0`.

If we try to add more features, such as multiple load and clear inputs and nonbinary modulus counting, we quickly run out of product terms in an X-series PLD. Also, PLD compilers aren't smart enough to rearrange an arbitrary expression into the form required by the X-series structure. Let's look at some modifications that *are* possible using at most one extra product term.

In order to obtain a counter that counts from 0 to state N and then returns to state 0 and repeats, we must detect state N and force the counter to return to state 0 on the next count. Suppose that STATEN is a product term that detects state N. Then the expression STATEN·CNTEN is 1 when the counter should be forced into state 0 (unless the /LD input overrides). We can use a combinational output of the PLD to realize this expression:

```
FORCE0 = STATEN * CNTEN;
```

Using FORCE0 as a "helper" term, we can now easily obtain the desired behavior:

```
Qi := (  (/CLR * /LD * Qi * /FORCE0 )                  " Hold
       +   0 )  :+:                                    " Unused
      (  (/CLR * /LD * CNTEN * Qi-1 *...* Q0 * /FORCE0)" Toggle
       + (/CLR *  LD * Di) );                          " Load
```

Copyright © 1994 by John F. Wakerly

If we modified the equation above to use STATEN·CNTEN directly instead of FORCE0, we would run out of product terms—STATEN, a product term, blows up into a sum of many terms when complemented.

Another approach to forcing state 0 does not use a combinational "helper" output, but instead uses at most one extra product term for each Qi output, depending on i and N:

(1) If Qi would be 0 in state $N+1$, do not modify the equation for Qi.

(2) If Qi would normally change from 0 to 1 between states N and $N+1$, add a product term to cancel this change:

```
Qi := (  (/CLR * /LD * Qi)                        " Hold
       + (/CLR * /LD * CNTEN * STATEN) )  :+:     " Cancel
      (  (/CLR * /LD * CNTEN * Qi-1 * ... * Q0)  " Toggle
       + (/CLR *  LD * Di) );                     " Load
```

(3) If Qi would be 1 in both states N and $N+1$, add a product term to toggle it to 0:

```
Qi := (  (/CLR * /LD * Qi)                        " Hold
       + (/CLR *  LD * Di) )  :+:                 " Load
      (  (/CLR * /LD * CNTEN * Qi-1 * ... * Q0)  " Toggle
       + (/CLR * /LD * CNTEN * STATEN) );         " Extra
```

These rules can be extended to force the counter to any desired starting state, M. If Qi has the same value in states M and $N+1$, do nothing. If the values are different, add a "cancel" term if Qi has different values in states N and $N+1$, and an "extra" toggle term otherwise.

In general, if you get stuck with an equation that has more than two product terms on either side of the XOR, you can't perform it in a X-series PLD, unless you use an extra combinational output and two-pass logic. However, sometimes our simple rule of thumb about XOR gates can help:

- Any two signals (inputs or output) of an XOR or XNOR gate may be complemented without changing the resulting logic function.

That is, you can complement both inputs or the output and one input without changing the result. For example, suppose an equation has the form

```
OUT := S1 :+: ((P1) + (P2));
```

where S1 is a sum term and P1 and P2 are product terms. If S1 has more than two variables, the equation can't be realized in one level. However, if you can live with a complemented output signal, you can write

```
/OUT := /S1 :+: ((P1) + (P2));
```

Under DeMorgan's theorem, /S1 is a single product term, so there's no problem.

Copyright © 1994 by John F. Wakerly