6.2.2 Simple Floating-Point Encoder

We defined a simple floating-point number format on page 467, and posed the design problem of converting a number from fixed-point to this floating point format. The I/O-pin requirements of this design are limited—11 inputs and 7 outputs—so we can potentially use a single PLD to replace the four parts that were used in the MSI solution.

An ABEL program for the fixed-to-floating-point converter is given in Table 6-4. The `WHEN` statement expresses the operation of determining the exponent value `E` in a very natural way. Then `E` is used to select the appropriate bits of `B` to use as the mantissa `M`.

Despite the deep nesting of the `WHEN` statement, only four product terms are needed in the minimal sum for each bit of `E`. The equations for the M bits are not too bad either, requiring only eight product terms each. Unfortunately, the

Table 6-4 An ABEL program for the fixed-point to floating-point PLD.

```
module fpenc
title 'Fixed-point to Floating-point Encoder'
FPENC device 'P20L8';

" Input and output pins
B10..B0              pin 1..11;
E2..E0, M3..M0       pin 21..15 istype 'com';

" Constant expressions
B = [B10..B0];
E = [E2..E0];
M = [M3..M0];

equations

WHEN B < 16 THEN E = 0;
ELSE WHEN B < 32 THEN E = 1;
ELSE WHEN B < 64 THEN E = 2;
ELSE WHEN B < 128 THEN E = 3;
ELSE WHEN B < 256 THEN E = 4;
ELSE WHEN B < 512 THEN E = 5;
ELSE WHEN B < 1024 THEN E = 6;
ELSE E = 7;

M = (E==0) & [B3..B0]
  # (E==1) & [B4..B1]
  # (E==2) & [B5..B2]
  # (E==3) & [B6..B3]
  # (E==4) & [B7..B4]
  # (E==5) & [B8..B5]
  # (E==6) & [B9..B6]
  # (E==7) & [B10..B7];

end fpenc
```

GAL20V8 has available only seven product terms per output. However, the GAL22V10 (Figure 8-22 on page 684) has more product terms available, so we can use that if we like.

One drawback of the design in Table 6-4 is that the `[M3..M0]` outputs are slow; since they use `[E2..E0]`, they take two passes through the PLD. A faster approach, if it fits, would be to rewrite the "select" terms (`E==0`, etc.) as intermediate equations before the equations section, and let ABEL expand the resulting `M` equations in a single level of logic. Unfortunately, ABEL does not allow `WHEN` statements outside of the `equations` section, so we'll have to roll up our sleeves and write our own logic expressions in the intermediate equations.

Table 6-5 shows the modified approach. The expressions for `S7`–`S0` are just mutually-exclusive AND-terms that indicate exponent values of 7–0 depending on the location of the most significant 1 bit in the fixed-point input number. The exponent `[E2..E0]` is a binary encoding of the select terms, and the mantissa bits `[M3..M0]` are generated using a select term for each case. It turns out that these `M` equations still require 8 product terms per output bit, but at least they're a lot faster since they use just one level of logic.

Table 6-5 Alternative ABEL program for the fixed-point to floating-point PLD.

```
module fpence
title 'Fixed-point to Floating-point Encoder'
FPENCE device 'P20L8';

" Input and output pins
B10..B0                     pin 1..11;
E2..E0, M3..M0              pin 21..15 istype 'com';

" Intermediate equations
S7 = B10;
S6 = !B10 & B9;
S5 = !B10 & !B9 & B8;
S4 = !B10 & !B9 & !B8 & B7;
S3 = !B10 & !B9 & !B8 & !B7 & B6;
S2 = !B10 & !B9 & !B8 & !B7 & !B6 & B5;
S1 = !B10 & !B9 & !B8 & !B7 & !B6 & !B5 & B4;
S0 = !B10 & !B9 & !B8 & !B7 & !B6 & !B5 & !B4;

equations

E2 = S7 # S6 # S5 # S4;
E1 = S7 # S6 # S3 # S2;
E0 = S7 # S5 # S3 # S1;

[M3..M0] = S0 & [B3..B0] # S1 & [B4..B1] # S2 & [B5..B2]
         # S3 & [B6..B3] # S4 & [B7..B4] # S5 & [B8..B5]
         # S6 & [B9..B6] # S7 & [B10..B7];

end fpenc
```