

HTML AND FLAT DESIGN

Fitters for PLD's and CPLD's will attempt to do a FLAT DESIGN. That is, all functions in your system will be in Sum-of-Products form. While this is the fastest possible implementation, it may not be practical in terms of size. An example of this is shown in the notes on ADDERS.

Consider the example below which might be used as a parity detector:

```
entity parity is port
  (a: in bit_vector(5 downto 0); e: out bit);
end parity;
architecture behavioral of parity is
begin
  process (i, a, e)
  begin
    e <= a(0) xor a(1) xor a(2) xor
          a(3) xor a(4) xor a(5);
  end process;
end behavioral;
```

If you try to compile this and fit it into a 22V10 PAL, during the fitting process you will get an error message: "Too many product terms." Now, it should be possible to fit a 6-bit parity detector into a 22V10. The problem is that such a function, in SOP form, requires 32 product terms. A 22V10 has, at most, 16. The solution, you might guess, would be to break it down into two 3-bit parity detectors and feed them into a 2-bit one. You might be tempted to do the following.

```
entity parity is port
  (a: in bit_vector(5 downto 0); e: out bit);
end parity;
architecture behavioral of parity is
  signal t1, t2: bit;
begin
  process (a, e, t1, t2)
  begin
    e <= t1 xor t2;
    t1 <= a(0) xor a(1) xor a(2);
    t2 <= a(3) xor a(4) xor a(5);
  end process;
end behavioral;
```

You think, "That should do it." You think wrong! If you examine the report file in detail, you will find the following:

Substituting virtuals - pass 1:

Note: Virtual equation for 't1' has been expanded (cost = 8):

```
t1 <= ((not a_1 and not a_0 and a_2)
      OR (not a_2 and not a_0 and a_1)
      OR (not a_2 and not a_1 and a_0))
```

```
OR (a_2 and a_1 and a_0));
```

Note: Virtual equation for 't2' has been expanded (cost = 32):

```
t2 <= ((not a_4 and not a_3 and a_5)
OR (not a_5 and not a_3 and a_4)
OR (not a_5 and not a_4 and a_3)
OR (a_5 and a_4 and a_3));
```

Warp2 is saying, "Hey you don't need t1 and t2, I'll remove them for you." So, the design equation is the same as before.

If you are really tricky, you might figure out the following solution.

```
entity parity is port
  (a: in bit_vector(5 downto 0); t1, t2: inout bit; e: out bit);
end parity;
architecture behavioral of parity is
begin
  process (a, e, t1, t2)
  begin
    e<=t1 xor t2;
    t1 <= a(0) xor a(1) xor a(2);
    t2 <= a(3) xor a(4) xor a(5);
  end process;
end behavior;
```

By declaring t1 and t2 as pins, you will force Warp to compute the intermediate terms. You are almost right. The equations generated for this are shown below.

DESIGN EQUATIONS (19:20:30)

$$t1 = a_2 \cdot \overline{a_1} \cdot \overline{a_0} + \overline{a_2} \cdot a_1 \cdot \overline{a_0} + \overline{a_2} \cdot \overline{a_1} \cdot a_0 + a_2 \cdot a_1 \cdot a_0$$

$$t2 = a_5 \cdot \overline{a_4} \cdot \overline{a_3} + \overline{a_5} \cdot a_4 \cdot \overline{a_3} + \overline{a_5} \cdot \overline{a_4} \cdot a_3 + a_5 \cdot a_4 \cdot a_3$$

$$e = a_5 \cdot \overline{a_4} \cdot \overline{a_3} \cdot \overline{a_2} \cdot \overline{a_1} \cdot \overline{a_0} + \overline{a_5} \cdot a_4 \cdot \overline{a_3} \cdot \overline{a_2} \cdot \overline{a_1} \cdot \overline{a_0} + \overline{a_5} \cdot \overline{a_4} \cdot a_3 \cdot \overline{a_2} \cdot \overline{a_1} \cdot \overline{a_0} + a_5 \cdot a_4 \cdot a_3 \cdot \overline{a_2} \cdot \overline{a_1} \cdot \overline{a_0} + \overline{a_5} \cdot \overline{a_4} \cdot \overline{a_3} \cdot a_2 \cdot \overline{a_1} \cdot \overline{a_0} + a_5 \cdot a_4 \cdot \overline{a_3} \cdot a_2 \cdot \overline{a_1} \cdot \overline{a_0} + \overline{a_5} \cdot \overline{a_4} \cdot a_3 \cdot a_2 \cdot \overline{a_1} \cdot \overline{a_0} + \overline{a_5} \cdot a_4 \cdot a_3 \cdot a_2 \cdot \overline{a_1} \cdot \overline{a_0}$$

```

+ /a_5 * /a_4 * /a_3 * /a_2 * a_1 * /a_0
+ a_5 * a_4 * /a_3 * /a_2 * a_1 * /a_0
+ a_5 * /a_4 * a_3 * /a_2 * a_1 * /a_0
+ /a_5 * a_4 * a_3 * /a_2 * a_1 * /a_0
+ a_5 * /a_4 * /a_3 * a_2 * a_1 * /a_0
+ /a_5 * a_4 * /a_3 * a_2 * a_1 * /a_0
+ /a_5 * /a_4 * a_3 * a_2 * a_1 * /a_0
+ a_5 * a_4 * a_3 * a_2 * a_1 * /a_0
+ /a_5 * /a_4 * /a_3 * /a_2 * /a_1 * a_0
+ a_5 * a_4 * /a_3 * /a_2 * /a_1 * a_0
+ a_5 * /a_4 * a_3 * /a_2 * /a_1 * a_0
+ /a_5 * a_4 * a_3 * /a_2 * /a_1 * a_0
+ a_5 * /a_4 * /a_3 * a_2 * /a_1 * a_0
+ /a_5 * a_4 * /a_3 * a_2 * /a_1 * a_0
+ /a_5 * /a_4 * a_3 * a_2 * /a_1 * a_0
+ a_5 * a_4 * a_3 * a_2 * /a_1 * a_0
+ a_5 * /a_4 * /a_3 * /a_2 * a_1 * a_0
+ /a_5 * a_4 * /a_3 * /a_2 * a_1 * a_0
+ /a_5 * /a_4 * a_3 * /a_2 * a_1 * a_0
+ a_5 * a_4 * a_3 * /a_2 * a_1 * a_0
+ /a_5 * /a_4 * /a_3 * a_2 * a_1 * a_0
+ a_5 * a_4 * /a_3 * a_2 * a_1 * a_0
+ a_5 * /a_4 * a_3 * a_2 * a_1 * a_0
+ /a_5 * a_4 * a_3 * a_2 * a_1 * a_0

```

Sure enough, it provide t1 and t2 as outputs as expected. But is still implemented E as a SOP expression and the design still doesn't fit! The only solution to this problem is shown below.

```

entity parity is port
  (a: in bit_vector(5 downto 0); e: out bit);
end parity;
architecture behavioral of parity is
  signal t1, t2: bit;
  attribute synthesis_off of t1: signal is true;
  attribute synthesis_off of t2: signal is true;
begin
  process (a, e, t1, t2)
  begin
    e<=t1 xor t2;
    t1 <= a(0) xor a(1) xor a(2);
    t2 <= a(3) xor a(4) xor a(5);
  end process;
end behavioral;

```

The ATTRIBUTE expressions tell WARP2 that it is NOT to synthesize t1 and t2 out of the equation for e. The logic generated is then as below which does fit into a 22V10.

DESIGN EQUATIONS (19: 24: 34)

e =
t1 * /t2

$$+ /t1 * t2$$

$$\begin{aligned} t2 = & a_5 * /a_4 * /a_3 \\ & + /a_5 * a_4 * /a_3 \\ & + /a_5 * /a_4 * a_3 \\ & + a_5 * a_4 * a_3 \end{aligned}$$

$$\begin{aligned} t1 = & a_2 * /a_1 * /a_0 \\ & + /a_2 * a_1 * /a_0 \\ & + /a_2 * /a_1 * a_0 \\ & + a_2 * a_1 * a_0 \end{aligned}$$