

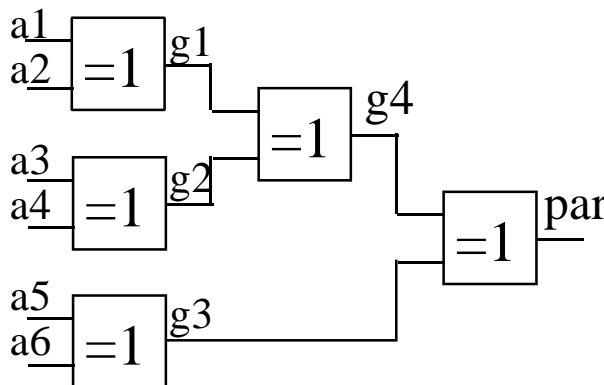
NOTES 7 CONTINUED
MORE ON THE PARITY GENERATOR

Before the days of the PLD, one might have synthesized a parity generator using exclusive-or gates as below.

For the 6-input parity generator, five gates would be required and would typically be connected as shown.

This device would have been quite slow as exclusive-or gates are, typically, slower than NAND and NOR gates. In addition, there is no way to avoid using three levels of them for a 6- to 8-input circuit.

One might describe precisely such a design in VHDL using a structural description. First, the exclusive-or gate itself must be described and placed in the “work” library.



```

library ieee;
use ieee.std_logic_1164.all;
entity exor is port(a,b: in std_logic;    o: out std_logic);
end exor;
architecture exor_gate of exor is
begin
    o<=a xor b;
end exor_gate;

```

Then the parity unit may be described as below.

```

library ieee;
use ieee.std_logic_1164.all;
use work.exor;
entity parity is port(a1, a2, a3, a4, a5, a6: in std_logic;
                    par: out std_logic);
end parity;
architecture parity_check of parity is
    signal g1, g2, g3, g4: std_logic;
begin
    gate1: exor port map (a1, a2, g1);
    gate2: exor port map (a3, a4, g2);
    gate3: exor port map (a5, a6, g3);
    gate4: exor port map (g1, g2, g4);
    gate5: exor port map (g3, g4, par);
end parity_check;

```

The above VHDL describes, precisely, the circuit described at the beginning. This may be seen better by “graphically annotating” the VHDL file as below.

```

library ieee;
use ieee.std_logic_1164.all;
use work.exor;
entity parity is port(a1, a2, a3, a4, a5, a6: in std_logic;
par: out std_logic);
end parity;
architecture parity_check of parity is
signal g1, g2, g3, g4:std_logic;
begin
gate1: exor port map (a1, a2, g1);
gate2: exor port map (a3, a4, g2);
gate3: exor port map (a5, a6, g3);
gate4: exor port map (g1, g2, g4);
gate5: exor port map (g3, g4, par);
end parity_check;

```

The lines in the diagram represent the “wires” that connect the inputs to the gates, that connect one gate to another, and connect the output gate to the output. Note the SIGNAL definition in the file which declares the connections which are internal to the circuit.

If this is compiled, the exact same “flat design” will be obtained in part one of this note. It can be unflattened with the “synthesis_off” attribute just as in the previous examples. Here we would add:

```
attribute synthesis_off of g1, g2, g3, g4: signal is true;
```

The resulting logic in a 22V10 would be as follows:

```

par = g3 * /g4 + /g3 * g4
g2 = a3 * /a4 + /a3 * a4
g1 = a1 * /a2 + /a1 * a2
g4 = g1 * /g2 + /g1 * g2
g3 = a5 * /a6 + /a5 * a6

```

While this may look neat and very simple, it actually implements the logic shown in the logic diagram - including the propagation through three levels of exclusive or gates (here implemented with AND/OR logic). If the design is compiled to a CYC371i instead, this delay factor can be found in the report file.

Signal Name	Delay Type	tmax	Path Description
cmb: : par[9]			
inp: : a1			
---->g1			
---->g4	tPD	17.5 ns	3 passes

There is no need to reduce the logic to such simple terms in the 22V10, especially since it results in a longer delay (three passes rather than two). This simply illustrates the “structural” approach to design in VHDL.