Table 6-31 VHDL behavioral architecture of a 32-bit mode-dependent comparator.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Vmodecmp is
    port ( M: in STD_LOGIC_VECTOR (1 downto 0);     -- mode
           A, B: in STD_LOGIC_VECTOR (31 downto 0); -- unsigned integers
           EQ, GT: out STD_LOGIC );                 -- comparison results
end Vmodecmp;

architecture Vmodecmp_arch of Vmodecmp is
begin
  process (M, A, B)
  begin
    case M is
      when "00" =>
        if A = B then EQ <= '1'; else EQ <= '0'; end if;
        if A > B then GT <= '1'; else GT <= '0'; end if;
      when "01" =>
        if A(31 downto 1) = B(31 downto 1) then EQ <= '1'; else EQ <= '0'; end if;
        if A(31 downto 1) > B(31 downto 1) then GT <= '1'; else GT <= '0'; end if;
      when "10" =>
        if A(31 downto 2) = B(31 downto 2) then EQ <= '1'; else EQ <= '0'; end if;
        if A(31 downto 2) > B(31 downto 2) then GT <= '1'; else GT <= '0'; end if;
      when others => EQ <= '0';  GT <= '0';
    end case;
  end process;
end Vmodecmp_arch;
```

individual comparators may or may not be fast, as discussed in the previous subsection, but we won't worry about speed for this example.

A more efficient alternative is to perform just one comparison for the 30 high-order bits of the inputs, and use additional logic that is dependent on mode to give a final result using the low-order bits as necessary. This approach is shown in Table 6-32. Two variables `EQ30` and `GT30` are used within the process to hold the results of the comparison of the 30 high-order bits. A case statement similar to the previous architecture's is then used to obtain the final results as a function of the mode. If desired, the speed of the 30-bit comparison can be optimized using the methods discussed in the preceding subsection.

6.3.6 Ones Counter

Several important algorithms include the step of counting the number of "1" bits in a data word. In fact, some microprocessor instruction sets have been extended recently to include ones counting as a basic instruction. In this example, let us suppose that we have a requirement to design a combinational circuit that counts ones in a 32-bit word as part of the arithmetic and logic unit of a microprocessor.

Table 6-32 More efficient architecture for a 32-bit mode-dependent comparator.

```
architecture Vmodecpe_arch of Vmodecmp is
begin
  process (M, A, B)
  variable EQ30, GT30: STD_LOGIC; -- 30-bit comparison results
  begin
    if A(31 downto 2) = B(31 downto 2) then EQ30 := '1'; else EQ30 := '0'; end if;
    if A(31 downto 2) > B(31 downto 2) then GT30 := '1'; else GT30 := '0'; end if;
    case M is
      when "00" =>
        if EQ30='1' and A(1 downto 0) = B(1 downto 0) then
          EQ <= '1'; else EQ <= '0'; end if;
        if GT30='1' or (EQ30='1' and A(1 downto 0) > B(1 downto 0)) then
          GT <= '1'; else GT <= '0'; end if;
      when "01" =>
        if EQ30='1' and A(1) = B(1) then EQ <= '1'; else EQ <= '0'; end if;
        if GT30='1' or (EQ30='1' and A(1) > B(1)) then
          GT <= '1'; else GT <= '0'; end if;
      when "10" =>  EQ <= EQ30;  GT <= GT30;
      when others => EQ <= '0';  GT <= '0';
    end case;
  end process;
end Vmodecpe_arch;
```

Ones counting can be described very easily by a behavioral VHDL program, as shown in Table 6-33. This program is fully synthesizable, but it may generate a very slow, inefficient realization with 32 5-bit adders in series.

Table 6-33 Behavioral VHDL program for a 32-bit ones counter.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity Vcnt1s is
    port ( D: in STD_LOGIC_VECTOR (31 downto 0);
           SUM: out STD_LOGIC_VECTOR (4 downto 0) );
end Vcnt1s;

architecture Vcnt1s_arch of Vcnt1s is
begin
  process (D)
  variable S: STD_LOGIC_VECTOR(4 downto 0);
  begin
    S := "00000";
    for i in 0 to 31 loop
      if D(i) = '1' then S := S + "00001"; end if;
    end loop;
    SUM <= S;
  end process;
end Vcnt1s_arch;
```

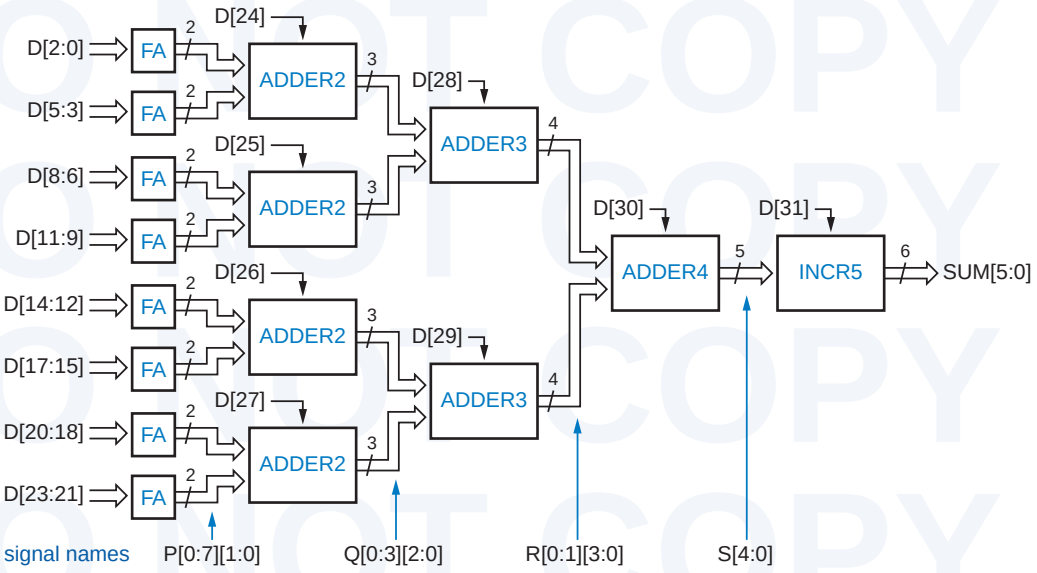

Figure 6-15 Structure of 32-bit ones counter.

To synthesize a more efficient realization of the ones counter, we must come up with an efficient structure and then write an architecture that describes it. Such a structure is the adder tree shown in Figure 6-15. A full adder (FA) adds three input bits to produce a 2-bit sum. Pairs of 2-bit numbers are added by 2-bit adders (ADDER2), each of which also has a carry input that can include add another 1-bit input to its sum. The resulting 3-bit sums are combined by 3-bit adders (ADDER3), and the final pair of 4-bit sums are combined in a 4-bit adder (ADDER4). By making use of the available carry inputs, this tree structure can combine 31 bits. A separate 5-bit incrementer is used at the end to handle the one remaining input bit.

The structure of Figure 6-15 can be created nicely by a structural VHDL architecture, as shown in Table 6-34. The program begins by declaring all of the components that will be used in the design, corresponding to the blocks in the figure.

The letter under each column of signals in Figure 6-15 corresponds to the name used for that signal in the program. Each of signals `P`, `Q`, and `R` is an array with one `STD_LOGIC_VECTOR` per connection in the corresponding column. The program defines a corresponding type for each of these, followed by the actual signal declaration.

The program in Table 6-34 makes good use of `generate` statements to create the multiple adder components on the left-hand side of the figure—eight FAs, four ADDER2s, and two ADDER3s. Finally, it instantiates one each of ADDER4 and INCR5.

Table 6-34 VHDL structural architecture for a 32-bit ones counter.

```
architecture Vcnt1str_arch of Vcnt1str is

component FA port ( A, B, CI: in  STD_LOGIC;
                    S, CO:    out STD_LOGIC );
end component;

component ADDER2 port ( A, B: in  STD_LOGIC_VECTOR(1 downto 0);
                        CI:   in  STD_LOGIC;
                        S:    out STD_LOGIC_VECTOR(2 downto 0) );
end component;

component ADDER3 port ( A, B: in  STD_LOGIC_VECTOR(2 downto 0);
                        CI:   in  STD_LOGIC;
                        S:    out STD_LOGIC_VECTOR(3 downto 0) );
end component;

component ADDER4 port ( A, B: in  STD_LOGIC_VECTOR(3 downto 0);
                        CI:   in  STD_LOGIC;
                        S:    out STD_LOGIC_VECTOR(4 downto 0) );
end component;

component INCR5 port ( A:  in  STD_LOGIC_VECTOR(4 downto 0);
                       CI: in  STD_LOGIC;
                       S:  out STD_LOGIC_VECTOR(5 downto 0) );
end component;

type Ptype is array (0 to 7) of STD_LOGIC_VECTOR(1 downto 0);
type Qtype is array (0 to 3) of STD_LOGIC_VECTOR(2 downto 0);
type Rtype is array (0 to 1) of STD_LOGIC_VECTOR(3 downto 0);
signal P: Ptype;  signal Q: Qtype;  signal R: Rtype;
signal S: STD_LOGIC_VECTOR(4 downto 0);

begin
  U1: for i in 0 to 7 generate
    U1C: FA port map (D(3*i), D(3*i+1), D(3*i+2), P(i)(0), P(i)(1));
  end generate;
  U2: for i in 0 to 3 generate
    U2C: ADDER2 port map (P(2*i), P(2*i+1), D(24+i), Q(i));
  end generate;
  U3: for i in 0 to 1 generate
    U3C: ADDER3 port map (Q(2*i), Q(2*i+1), D(28+i), R(i));
  end generate;
  U4: ADDER4 port map (R(0), R(1), D(30), S);
  U5: INCR5 port map (S, D(31), SUM);
end Vcnt1str_arch;
```

The definitions of the ones counter's individual component entities and architectures, from FA to INCR, can be made in separate structural or behavioral programs. For example, Table 6-35 is a structural program for FA. The rest of the components are left as exercises (6.20–6.22).