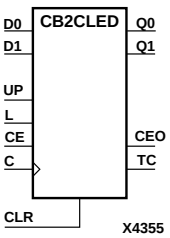
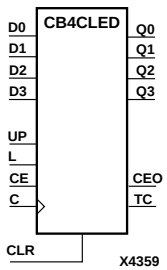
CB2CLED, CB4CLED, CB8CLED, CB16CLED

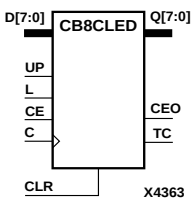
2-, 4-, 8-, 16-Bit Loadable Cascadable Bidirectional Binary Counters with Clock Enable and Asynchronous Clear

Architectures Supported

CBD2CLED, CBD4CLED, CBD8CLED, CBD16CLED	
Spartan-II, Spartan-IIE	No
Spartan-3	No
Virtex, Virtex-E	No
Virtex-II, Virtex-II Pro, Virtex-II Pro X	No
XC9500, XC9500XV, XC9500XL	No
CoolRunner XPLA3	No
CoolRunner-II	Macro
CoolRunner-IIS	No







CB2CLED, CB4CLED, CB8CLED, and CB16CLED are, respectively, 2-, 4-, 8- and 16-bit (stage), synchronously loadable, asynchronously clearable, cascadable, bidirectional binary counters. The asynchronous clear (CLR) is the highest priority input. When CLR is High, all other inputs are ignored; the Q outputs, terminal count (TC), and clock enable out (CEO) go to logic level zero, independent of clock transitions. The data on the D inputs is loaded into the counter when the load enable input (L) is High during the Low-to-High clock (C) transition, independent of the state of clock enable (CE). The Q outputs decrement when CE is High and UP is Low during the Low-to-High clock transition. The Q outputs increment when CE and UP are High. The counter ignores clock transitions when CE is Low.

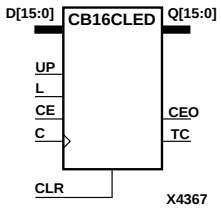
For counting up, the TC output is High when all Q outputs and UP are High. For counting down, the TC output is High when all Q outputs and UP are Low. To cascade counters, the CEO output of each counter is connected to the CE pin of the next stage. The clock, UP, L, and CLR inputs are connected in parallel. CEO is active (High) when TC and CE are High. The maximum length of the counter is determined by the accumulated CE-to-TC propagation delays versus the clock period. The clock period must be greater than $n(t_{CE\text{-}TC})$, where n is the number of stages and the time $t_{CE\text{-}TC}$ is the CE-to-TC propagation delay of each stage.

When cascading counters, use the CEO output if the counter uses the CE input; use the TC output if it does not. For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, see "CB2X1, CB4X1, CB8X1, CB16X1" for high-performance cascadable, bidirectional counters.

The counter is asynchronously cleared, output Low, when power is applied.

For XC9500/XV/XL, CoolRunner XPLA3, and CoolRunner-II, the power-on condition can be simulated by applying a High-level pulse on the PRLD global net.

Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, and Virtex-II Pro X simulate power-on when global set/reset (GSR) is active.



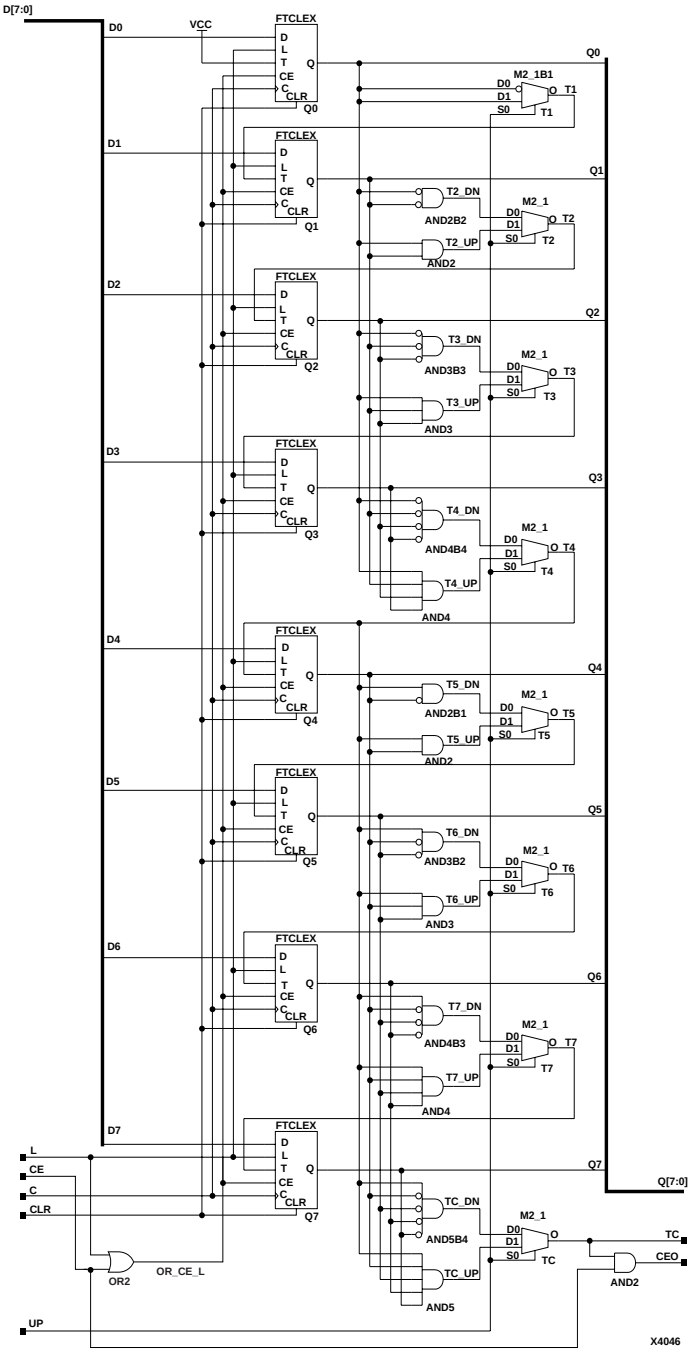
GSR defaults to active-High but can be inverted with an inverter in front of the GSR input of STARTUP_SPARTAN2, STARTUP_SPARTAN3, STARTUP_VIRTEX, or STARTUP_VIRTEX2.

Inputs						Outputs		
CLR	L	CE	C	UP	Dz – D0	Qz – Q0	TC	CEO
1	X	X	X	X	X	0	↑	↑*CE
0	1	X	↑	X	Dn	Dn	TC	CEO
0	0	0	X	X	X	No Chg	No Chg	0
0	0	1	↑	1	X	Inc	TC	CEO
0	0	1	↑	0	X	Dec	TC	CEO

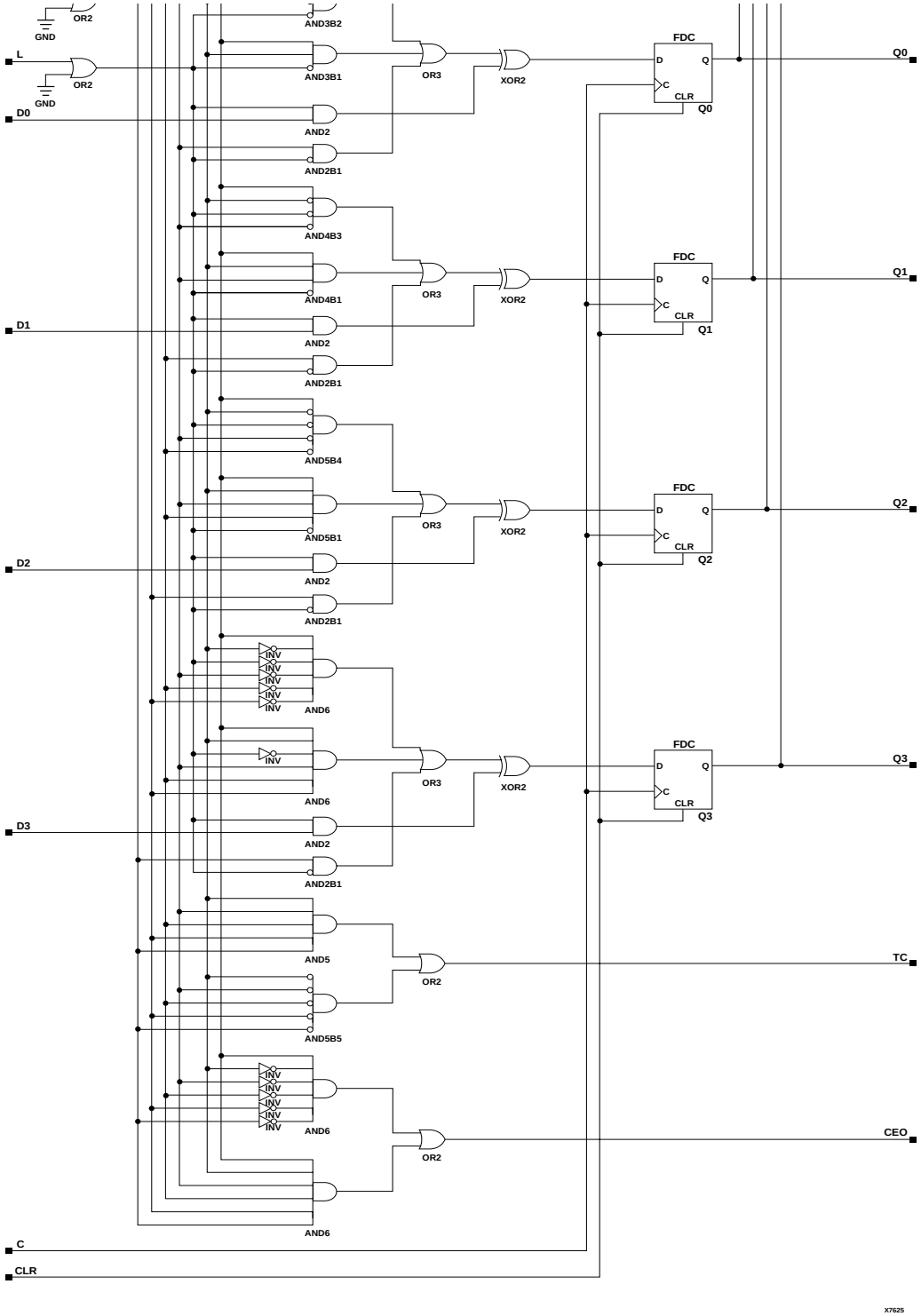
z = 1 for CB2CLED; z = 3 for CB4CLED; z = 7 for CB8CLED; z = 15 for CB16CLED

TC = (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP) + (Qz • Q(z-1) • Q(z-2) • ... • Q0 • UP)

CEO = TC • CE



CB8CLED Implementation Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-II Pro X



CB4CLED Implementation XC9500/XV/XL, CoolRunner XPLA3, CoolRunner-II

Usage

For HDL, these design elements are inferred rather than instantiated.

VHDL Inference Code

```
architecture Behavioral of cb2cled is

 constant TERMINAL_COUNT_UP : std_logic_vector(WIDTH-1 downto 0) :=
   (others => '1');
 constant TERMINAL_COUNT_DOWN : std_logic_vector(WIDTH-1 downto 0) :=
   (others => '0');

begin

process(C, CLR)
begin
 if (CLR='1') then
  Q <= (others => '0');
 elsif C'event and C='1' then
  if (L = '1') then
   Q <= D;
  elsif (CE='1') then
   if (UP='1') then
    Q <= Q+1;
   elsif (UP='0') then
    Q <= Q-1;
   end if;
  end if;
 end if;
end process;

process(Q, UP)
begin
 if (((Q = TERMINAL_COUNT_UP) and (UP = '1')) or
  ((Q = TERMINAL_COUNT_DOWN) and (UP = '0'))) then
  TC <='1';
 else
  TC <='0';
 end if;
end process;

CEO<=TC and CE;

end Behavioral;
```

Verilog Inference Code

```
always @ (posedge C or posedge CLR)
begin
 if (CLR)
  Q <= 0;
 else if (L)
  Q <= D;
 else if (CE)
  begin
   if (UP)
      Q <= Q + 1;
   else if (!UP)
      Q <= Q - 1;
  end
end
```

```
always @ (Q or UP)
begin
 if ((Q == TERMINAL_COUNT_UP && UP) || (Q == TERMINAL_COUNT_DOWN &&
   !UP))
  TC <= 1;
 else
  TC <= 0;
end

always @ (TC or CE)
begin
 CEO <= TC & CE;
end
```