Source Code:

Structural:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Register_Transfer is
	port (B : in std_logic_vector(3 downto 0);
			load, increment, clk, reset, Cin : in std_logic;
			Cout : out std_logic;
			A, Sum : inout std_logic_vector(3 downto 0));
end Register_Transfer;

architecture structure of Register_Transfer is
	component register_b is
		port (D : in std_logic_vector(3 downto 0);
				clk, reset : in std_logic;
				Q : out std_logic_vector(3 downto 0));
	end component;
	component para_adder is
		port (A, B : in std_logic_vector(3 downto 0);
				Cin : in std_logic;
				S : out std_logic_vector(3 downto 0);
				Cout : out std_logic);
	end component;
	component counter_a is
		port (D : in std_logic_vector(3 downto 0);
				load, increment, clk, reset : in std_logic;
				Q : out std_logic_vector(3 downto 0));
	end component;

	signal tmp: std_logic_vector(3 downto 0);

begin
	RB: register_b port map(B, clk, reset, tmp);
	PA: para_adder port map(tmp, A, Cin, Sum, Cout);
	RA: counter_a port map(Sum, load, increment, clk, reset, A);
end structure;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity register_b is
	port( D : in std_logic_vector(3 downto 0);
			clk : in std_logic;
			reset : in std_logic;
			Q : out std_logic_vector(3 downto 0));
end register_b;

architecture structure_ra of register_b is
	component d_flipflop is
		port( a, r, c : in std_logic;
				qq : out std_logic);
	end component;
begin
	DFF1: d_flipflop port map(D(0), reset, clk, Q(0));
	DFF2: d_flipflop port map(D(1), reset, clk, Q(1));
	DFF3: d_flipflop port map(D(2), reset, clk, Q(2));
	DFF4: d_flipflop port map(D(3), reset, clk, Q(3));
end structure_ra;
```

```
library ieee;
use ieee.std_logic_1164.all;

entity d_flipflop is
      port( a, r, c : in std_logic;
                  qq : out std_logic);
end d_flipflop;

architecture behave_d of d_flipflop is
      signal qtemp : std_logic :='0';
begin
      qq <= qtemp;
      process(c, r)
      begin
            if(r = '1') then
                  qtemp <= '0';
            elsif(rising_edge(c)) then
                  if(a = '0') then
                        qtemp <= '0';
                  else
                        qtemp <= '1';
                  end if;
            end if;
      end process;
end behave_d;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity para_adder is
    Port ( A : in  STD_LOGIC_VECTOR(3 downto 0);
           B : in  STD_LOGIC_VECTOR(3 downto 0);
                    Cin : in STD_LOGIC;
           S : out  STD_LOGIC_VECTOR(3 downto 0);
           Cout : out  STD_LOGIC);
end para_adder;

architecture structure_pa of para_adder is
      component fulladder
      port (x, y, z : in std_logic;
                  sum, carry : out std_logic);
      end component;

      signal C1, C2, C3 : std_logic;
begin
      FA1: fulladder port map (A(0), B(0), Cin, S(0), C1);
      FA2: fulladder port map (A(1), B(1), C1, S(1), C2);
      FA3: fulladder port map (A(2), B(2), C2, S(2), C3);
      FA4: fulladder port map (A(3), B(3), C3, S(3), Cout);
end structure_pa;

library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
      port (x, y, z : in std_logic;
                  sum, carry : out std_logic);
end fulladder;
architecture structure_f of fulladder is
      component halfadder
      port (a, b : in std_logic;
                  m, n : out std_logic);
      end component;
```

```
	component orgate
	port (p, q : in std_logic;
			r : out std_logic);
	end component;

	signal hs, hc1, hc2 : std_logic;
begin
	HA1: halfadder port map (x, y, hs, hc1);
	HA2: halfadder port map (hs, z, sum, hc2);
	ORG: orgate port map (hc1, hc2, carry);
end structure_f;

library ieee;
use ieee.std_logic_1164.all;

entity halfadder is
	port (a, b : in std_logic;
			m, n : out std_logic);
end halfadder;
architecture dataflow_h of halfadder is
begin
	m <= a xor b;
	n <= a and b;
end dataflow_h;

library ieee;
use ieee.std_logic_1164.all;

entity orgate is
	port (p, q : in std_logic;
			r : out std_logic);
end orgate;
architecture dataflow_o of orgate is
begin
	r <= p or q;
end dataflow_o;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity counter_a is
	port (D : in std_logic_vector(3 downto 0);
			load, increment, clk, reset : in std_logic;
			Q : out std_logic_vector(3 downto 0));
end counter_a;
architecture structure_c of counter_a is
signal temp: std_logic_vector(3 downto 0) := x"0";
begin
	process(clk, reset)
	begin
		if(reset='1') then temp <=x"0";
		elsif(rising_edge(clk)) then
			if(load='1') then temp <= D;
			elsif(increment='1') then temp <= temp + x"1";
			end if;
		end if;
	end process;
	Q <= temp;
	end structure_c;
```

Behavioral:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity Register_Transfer_Behave is
      port (B : in std_logic_vector(3 downto 0);
                  load, increment, clk, reset, Cin : in std_logic;
                  Cout : out std_logic;
                  A, Sum : inout std_logic_vector(3 downto 0));
end Register_Transfer_Behave;

architecture Behavioral of Register_Transfer_Behave is
signal c : std_logic := '0';
signal tmp : std_logic_vector(3 downto 0) := x"0";
begin
      process(A, B, c)
      begin
            for i in 0 to 3 loop
                  if(c = '0') then
                        Sum(i) <= A(i) xor B(i);
                        c <= A(i) and B(i);
                  else
                        Sum(i) <= not (A(i) xor B(i));
                        c <= A(i) or B(i);
                  end if;
            end loop;
      end process;
      Cout <= c;

      process(clk, reset)
      begin
            if(reset='1') then tmp <= x"0";
            elsif(rising_edge(clk)) then
                  if(load='1') then tmp <= Sum;
                  elsif(increment='1') then tmp <= tmp + x"1";
                  end if;
            end if;
      end process;
      A <= tmp;
end Behavioral;
```

Test Bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Register_Transfer_Behave_tb IS
END Register_Transfer_Behave_tb;

ARCHITECTURE behavior OF Register_Transfer_Behave_tb IS

    COMPONENT Register_Transfer_Behave
    PORT(
         B : IN  std_logic_vector(3 downto 0);
         load : IN  std_logic;
         increment : IN  std_logic;
         clk : IN  std_logic;
         reset : IN  std_logic;
         Cin : IN  std_logic;
         Cout : OUT  std_logic;
         A : INOUT  std_logic_vector(3 downto 0);
         Sum : INOUT  std_logic_vector(3 downto 0)
        );
    END COMPONENT;
   signal B : std_logic_vector(3 downto 0) := (others => '0');
   signal load : std_logic := '0';
   signal increment : std_logic := '1';
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   signal Cin : std_logic := '0';

   signal A : std_logic_vector(3 downto 0);
   signal Sum : std_logic_vector(3 downto 0);

   signal Cout : std_logic;

   constant clk_period : time := 10 ns;
      shared variable simend : boolean := false;

BEGIN
   uut: Register_Transfer_Behave PORT MAP (
          B => B,
          load => load,
          increment => increment,
          clk => clk,
          reset => reset,
          Cin => Cin,
          Cout => Cout,
          A => A,
          Sum => Sum
        );

   clk_process :process
   begin
         while simend = false loop
               clk <= not clk;
               wait for clk_period/2;
         end loop;
   end process;

   stim_proc: process
   begin
         B <= "0110";
         wait for 400 ns;
         B <= "1001";
         wait for 300 ns;
         reset <= '1';
         wait for 200 ns;
         simend := true;
   end process;
      stim_proc1: process
      begin
         wait for 100 ns;
         load <= not load;
         increment <= not increment;
      end process;

END;
```