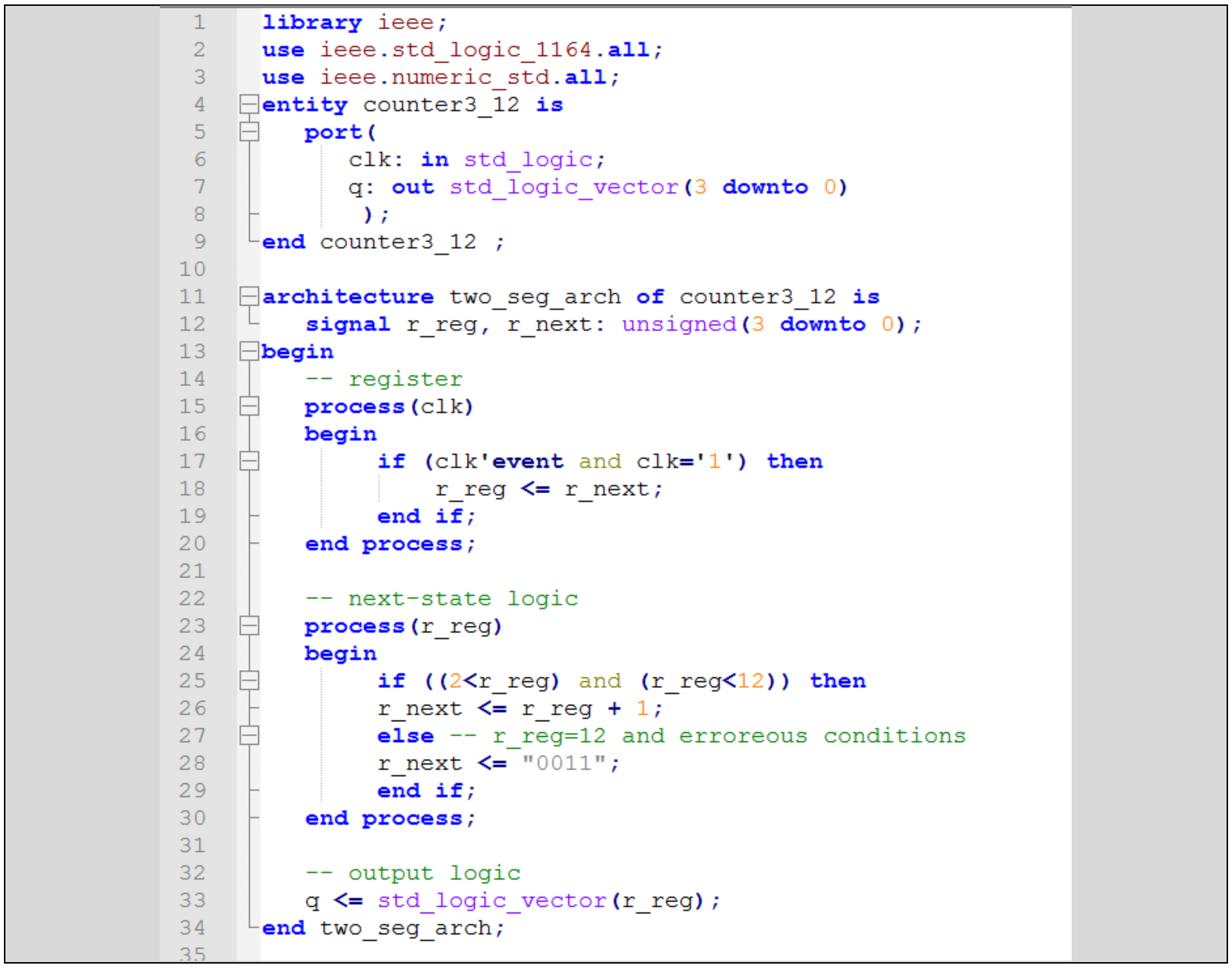
CEG3155: DGD 9

Question 1 (Chu 8.7)

Consider a 4-bit counter that counts from 3 (“0011”) to 12 (“1100”) and then wraps around. If the counter enters an unused state (such as “0000”) because of noise, it will restart from “0011” at the next rising edge of the clock. Derive the VHDL code for this circuit and draw the conceptual top-level diagram.

Solution

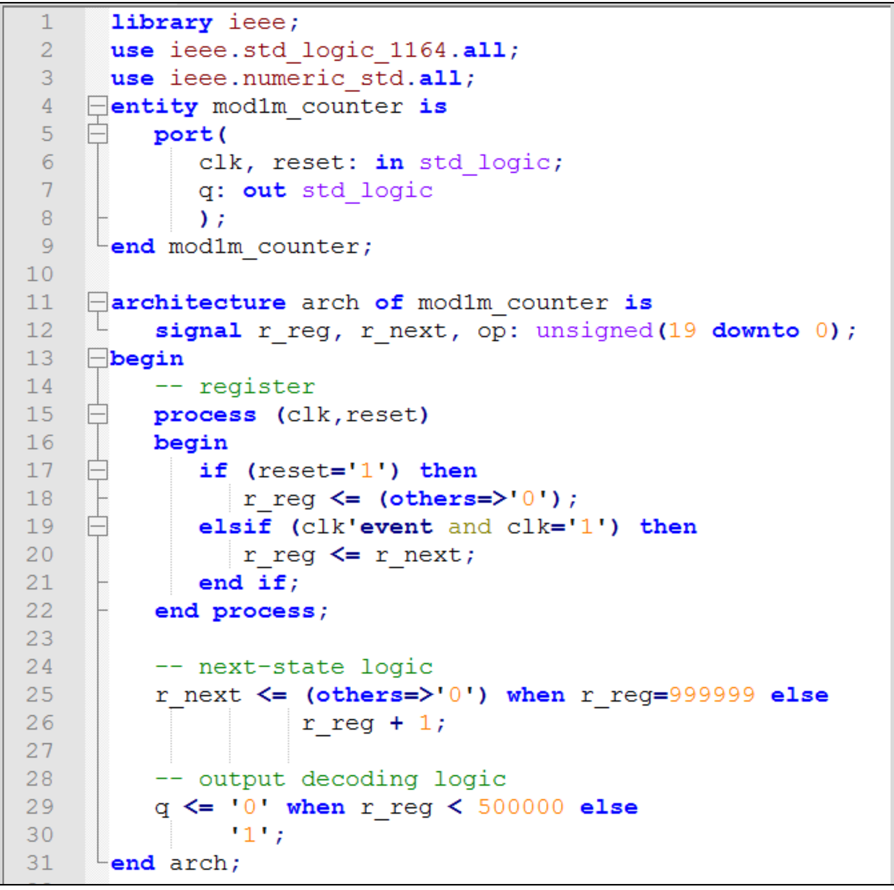


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity counter3_12 is
   port(
      clk: in std_logic;
      q: out std_logic_vector(3 downto 0)
       );
end counter3_12 ;

architecture two_seg_arch of counter3_12 is
   signal r_reg, r_next: unsigned(3 downto 0);
begin
   -- register
   process(clk)
   begin
        if (clk'event and clk='1') then
            r_reg <= r_next;
        end if;
   end process;

   -- next-state logic
   process(r_reg)
   begin
        if ((2<r_reg) and (r_reg<12)) then
        r_next <= r_reg + 1;
        else -- r_reg=12 and erroreous conditions
        r_next <= "0011";
        end if;
   end process;

   -- output logic
   q <= std_logic_vector(r_reg);
end two_seg_arch;
```

Question 2 (Chu 8.10)

Assume that we have a 1-MHz clock signal. Design a circuit that generates a 1-Hz output pulse with a 50% duty cycle (i.e., 50% of '1' and 50% of '0'). Derive the VHDL code for this circuit.

Solution



```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity mod1m_counter is
   port(
      clk, reset: in std_logic;
      q: out std_logic
      );
end mod1m_counter;

architecture arch of mod1m_counter is
   signal r_reg, r_next, op: unsigned(19 downto 0);
begin
   -- register
   process (clk,reset)
   begin
      if (reset='1') then
         r_reg <= (others=>'0');
      elsif (clk'event and clk='1') then
         r_reg <= r_next;
      end if;
   end process;

   -- next-state logic
   r_next <= (others=>'0') when r_reg=999999 else
             r_reg + 1;

   -- output decoding logic
   q <= '0' when r_reg < 500000 else
        '1';
end arch;
```

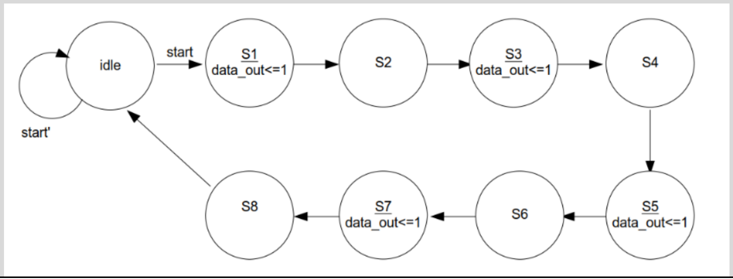
Question 3 (Chu 10.5)

In digital communication, a special synchronization pattern, known as a preamble, is used to indicate the beginning of a packet. For example, the Ethernet I1 preamble includes eight repeating octets of "10101010". We wish to design an FSM that generates the "10101010" pattern. The circuit has an input signal, start, and an output, data-out. When start is 'l', the "10101010" will be generated in the next eight clock cycles.

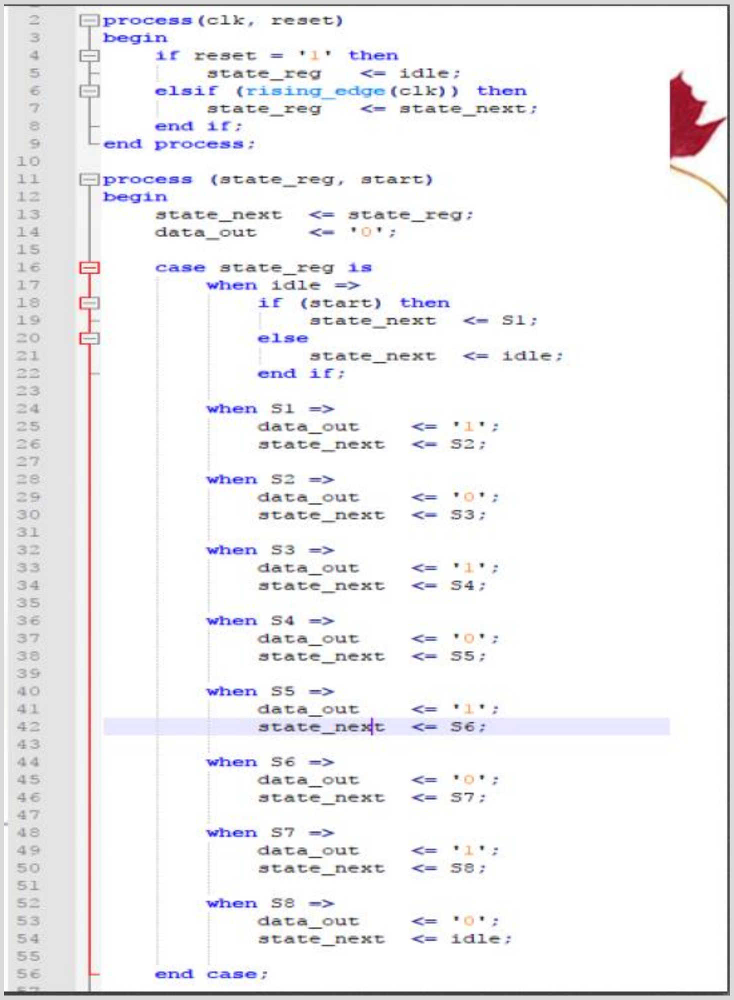
a) Derive the state diagram.
b) Derive VHDL code according to the state diagram.

Solution

a)



b)



```
process(clk, reset)
begin
    if reset = '1' then
        state_reg   <= idle;
    elsif (rising_edge(clk)) then
        state_reg   <= state_next;
    end if;
end process;

process (state_reg, start)
begin
    state_next    <= state_reg;
    data_out      <= '0';

    case state_reg is
        when idle =>
            if (start) then
                state_next   <= S1;
            else
                state_next   <= idle;
            end if;

        when S1 =>
            data_out     <= '1';
            state_next   <= S2;

        when S2 =>
            data_out     <= '0';
            state_next   <= S3;

        when S3 =>
            data_out     <= '1';
            state_next   <= S4;

        when S4 =>
            data_out     <= '0';
            state_next   <= S5;

        when S5 =>
            data_out     <= '1';
            state_next   <= S6;

        when S6 =>
            data_out     <= '0';
            state_next   <= S7;

        when S7 =>
            data_out     <= '1';
            state_next   <= S8;

        when S8 =>
            data_out     <= '0';
            state_next   <= idle;

    end case;
```

Question 4

A different approach for implementing integer division is to perform repeated subtraction. Let y and d be the dividend and divisor, respectively. Let q and r be the quotient and remainder, respectively.

a) Derive a pseudo algorithm.
b) Convert the algorithm to ASMD chart.
c) Derive the conceptual diagram.
d) Write the VHDL code.

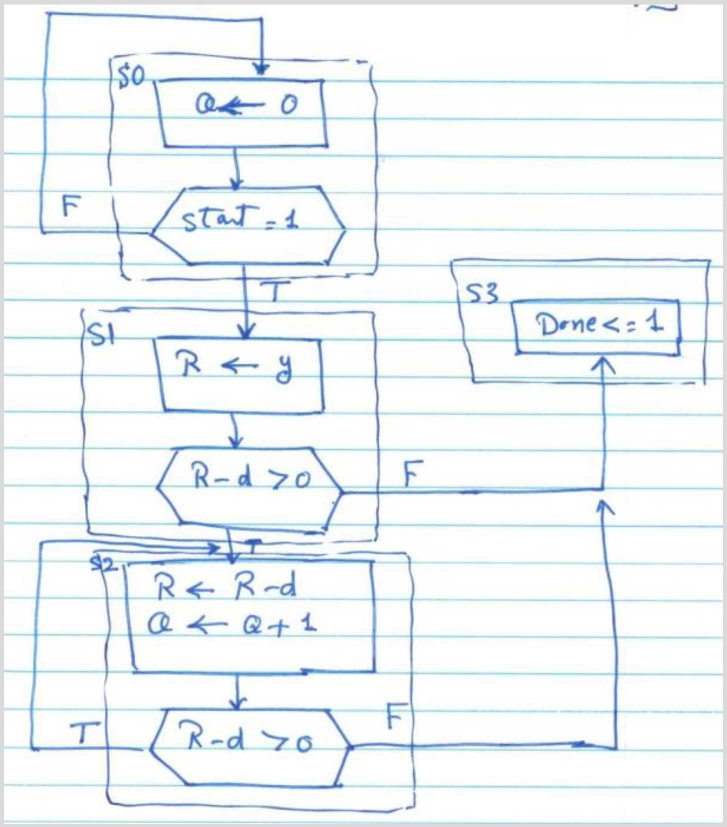
Solution

a) Pseudo algorithm:

```
$q = 0$
$r = y$
```

```
$while\ (r - d) > 0\ do$
    $r = r - d$
    $q = q + 1$
$end\ while$
```

b)

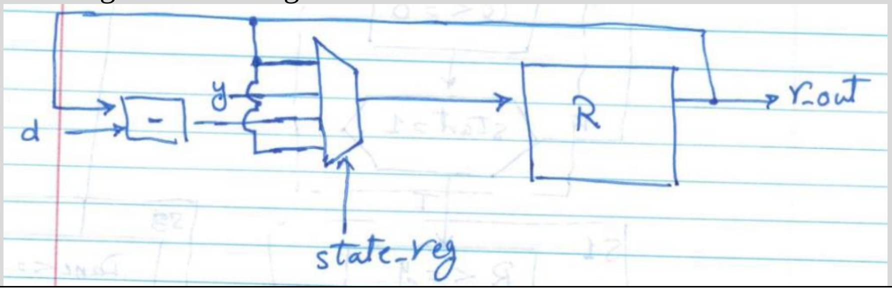


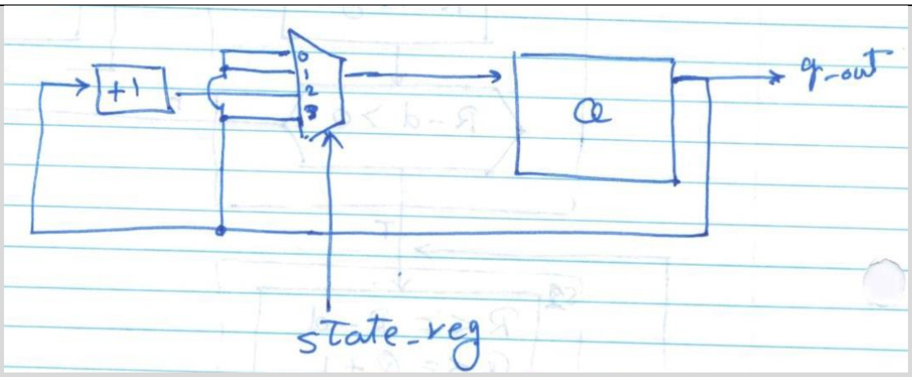
c)

- Write all RTL operations:

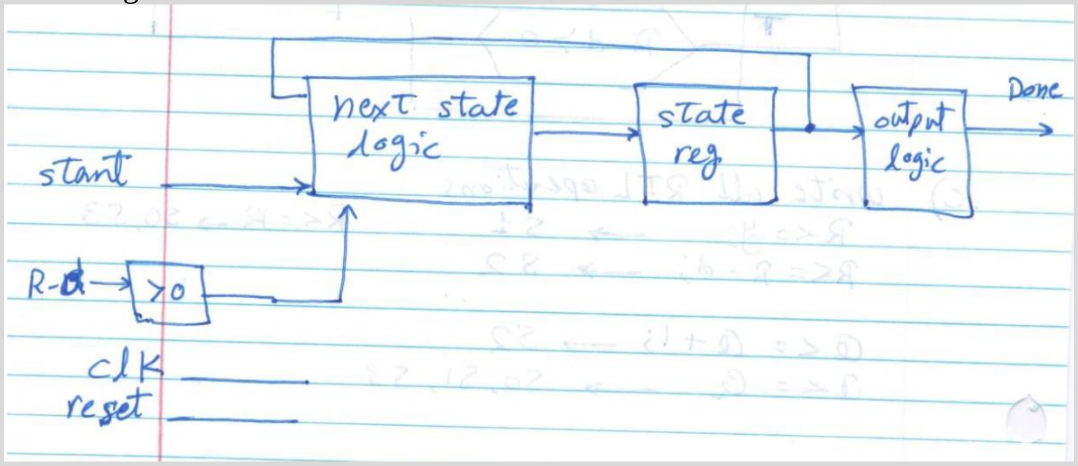
$r \leftarrow y : S1$
$r \leftarrow r - d : S2$
$r \leftarrow r : S0, S3$
$q \leftarrow q + 1 : S2$
$q \leftarrow q : S0, S1, S3$

- Derive the logic for each register:

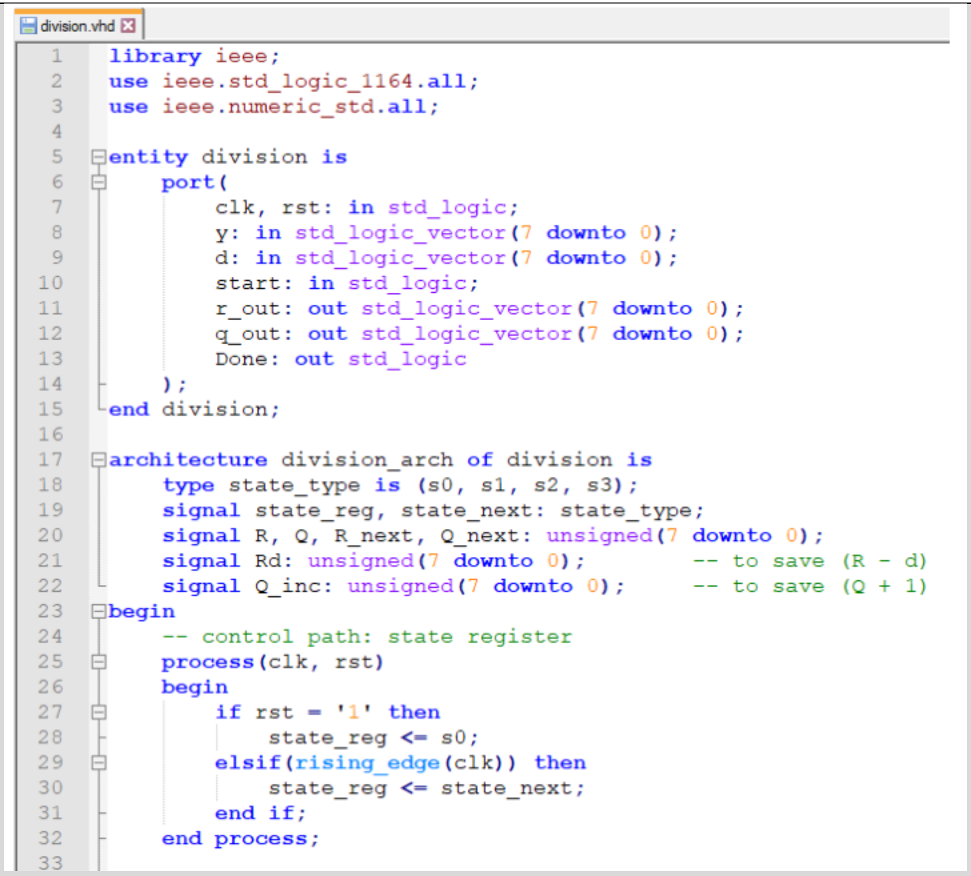



+1
0
1
2
3
Q
q_out
state_reg

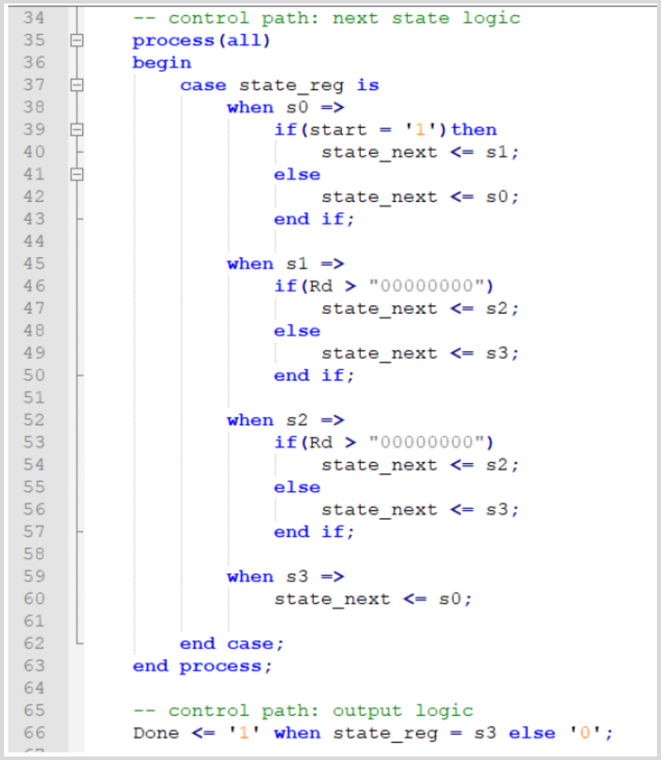
- FSM logic:


next state logic
state reg
output logic
Done
start
R-Q
>0
clk
reset

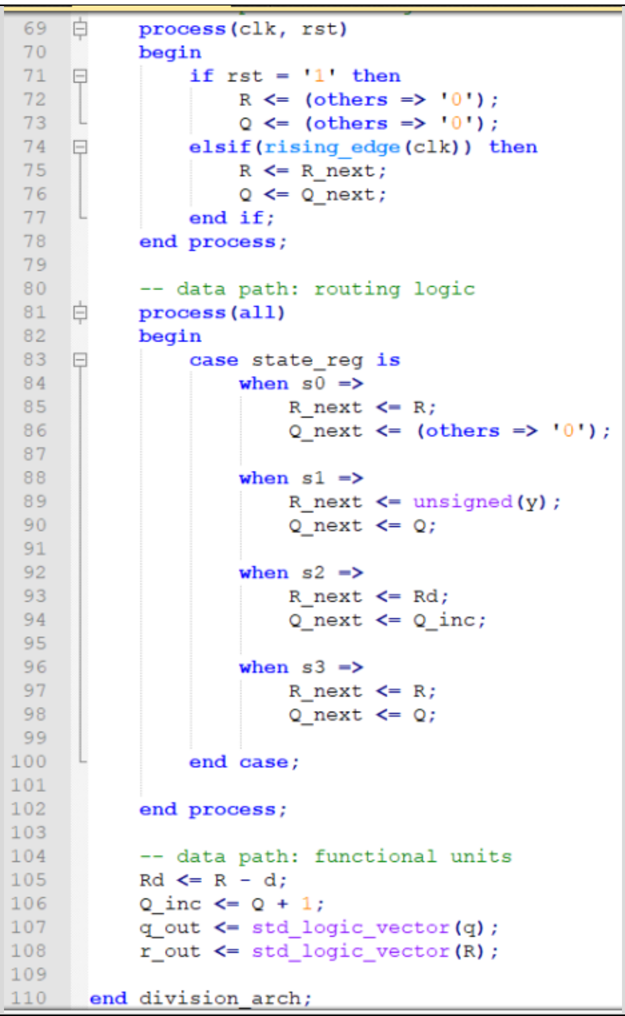
d)


division.vhd

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

entity division is
    port(
        clk, rst: in std_logic;
        y: in std_logic_vector(7 downto 0);
        d: in std_logic_vector(7 downto 0);
        start: in std_logic;
        r_out: out std_logic_vector(7 downto 0);
        q_out: out std_logic_vector(7 downto 0);
        Done: out std_logic
    );
end division;

architecture division_arch of division is
    type state_type is (s0, s1, s2, s3);
    signal state_reg, state_next: state_type;
    signal R, Q, R_next, Q_next: unsigned(7 downto 0);
    signal Rd: unsigned(7 downto 0);          -- to save (R - d)
    signal Q_inc: unsigned(7 downto 0);       -- to save (Q + 1)
begin
    -- control path: state register
    process(clk, rst)
    begin
        if rst = '1' then
            state_reg <= s0;
        elsif(rising_edge(clk)) then
            state_reg <= state_next;
        end if;
    end process;

```



```vhdl
    -- control path: next state logic
    process(all)
    begin
        case state_reg is
            when s0 =>
                if(start = '1')then
                    state_next <= s1;
                else
                    state_next <= s0;
                end if;

            when s1 =>
                if(Rd > "00000000")
                    state_next <= s2;
                else
                    state_next <= s3;
                end if;

            when s2 =>
                if(Rd > "00000000")
                    state_next <= s2;
                else
                    state_next <= s3;
                end if;

            when s3 =>
                state_next <= s0;

        end case;
    end process;

    -- control path: output logic
    Done <= '1' when state_reg = s3 else '0';
```



```vhdl
    process(clk, rst)
    begin
        if rst = '1' then
            R <= (others => '0');
            Q <= (others => '0');
        elsif(rising_edge(clk)) then
            R <= R_next;
            Q <= Q_next;
        end if;
    end process;

    -- data path: routing logic
    process(all)
    begin
        case state_reg is
            when s0 =>
                R_next <= R;
                Q_next <= (others => '0');

            when s1 =>
                R_next <= unsigned(y);
                Q_next <= Q;

            when s2 =>
                R_next <= Rd;
                Q_next <= Q_inc;

            when s3 =>
                R_next <= R;
                Q_next <= Q;

        end case;

    end process;

    -- data path: functional units
    Rd <= R - d;
    Q_inc <= Q + 1;
    q_out <= std_logic_vector(q);
    r_out <= std_logic_vector(R);

end division_arch;
```

Question 5 (Chu 11.1)

The ASMD chart in Figure 11.6 uses the n register to keep track of the number of iterations. It is initialized with b-in and counts down to 0. Alternatively, it can be initialized with 0 and counts up to b-in. From the implementation point of view, which method is better? Explain.

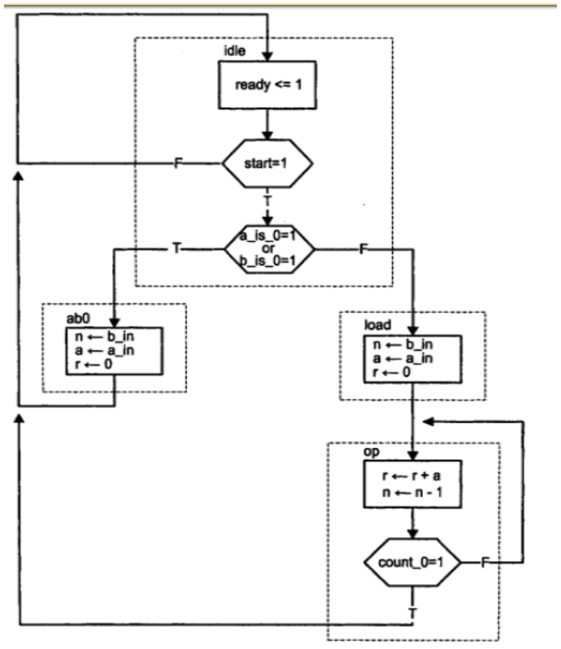


Figure 11.6 ASMD chart of a repetitive-addition multiplier.

Solution

- If the n register counts up, it has to be compared with b_in and a full comparator is needed.
- On the other hand, if the n register counts down, it is always compared with 0, which is a much simpler circuit (8-input nor gate).

Question 6 (Chu 11.2)

The ASMD chart in Figure 11.6 must return to the idle state after completion even when the main system is ready with a new set of inputs. An alternative is to allow the circuit to perform back-to-back operation in which the FSMD jumps to the ab0 or load state if the start signal is asserted while the current operation is completed.

(a) Modify the ASMD chart to reflect the change.

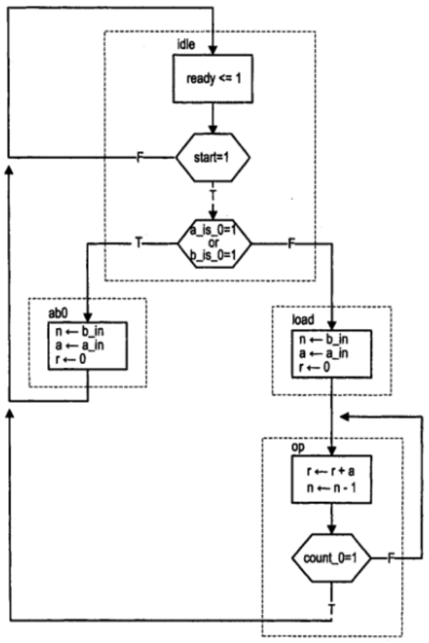
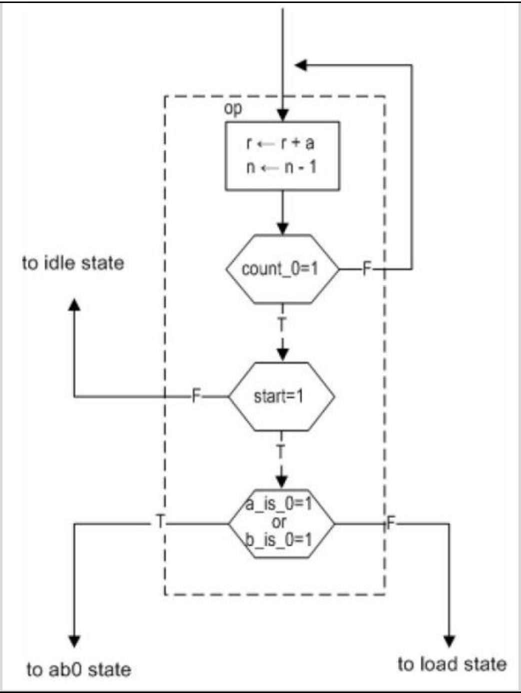


Figure 11.6 ASMD chart of a repetitive-addition multiplier.

Solution


op
r ← r + a
n ← n - 1
count_0=1
F
T
start=1
F
T
a_is_0=1 or b_is_0=1
T
F
to idle state
to ab0 state
to load state