Preamble

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  entity preamble is  port(  clk: in std\_logic;  start: in std\_logic;  data\_out: out std\_logic  );  end preamble;  architecture preamble\_arch of preamble is  type state\_type is  (idle,out1,out2,out3,out4,out5,out6,out7,out8);  signal state\_reg, state\_next: state\_type := idle;  begin  process(clk)  begin  if(rising\_edge(clk)) then  state\_reg <= state\_next;  end if;  end process;    process(state\_reg,start)  begin  state\_next <= state\_reg;  case state\_reg is  when idle=>  if start='1' then  state\_next <= out1;  end if;  when out1=>  state\_next <= out2;  data\_out <= '1';  when out2=>  state\_next <= out3;  data\_out <= '0';  when out3=>  state\_next <= out4;  data\_out <= '1';  when out4=>  state\_next <= out5;  data\_out <= '0';  when out5=>  state\_next <= out6;  data\_out <= '1';  when out6=>  state\_next <= out7;  data\_out <= '0';  when out7=>  state\_next <= out8;  data\_out <= '1';  when out8=>  state\_next <= idle;  data\_out <= '0';  end case;  end process;  end preamble\_arch; |

Preamble\_clever

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  entity preamble\_clever is  port(  clk: in std\_logic;  start: in std\_logic;  data\_out: out std\_logic  );  end preamble\_clever;  architecture clever\_arch of preamble\_clever is  constant idle: std\_logic\_vector(3 downto 0) := "0000";  constant out1: std\_logic\_vector(3 downto 0) := "0001";  constant out2: std\_logic\_vector(3 downto 0) := "0010";  constant out3: std\_logic\_vector(3 downto 0) := "0011";  constant out4: std\_logic\_vector(3 downto 0) := "0100";  constant out5: std\_logic\_vector(3 downto 0) := "0101";  constant out6: std\_logic\_vector(3 downto 0) := "0110";  constant out7: std\_logic\_vector(3 downto 0) := "0111";  constant out8: std\_logic\_vector(3 downto 0) := "1000";  signal state\_reg, state\_next: std\_logic\_vector(3 downto 0) := "0000";  begin  process(clk)  begin  if(rising\_edge(clk)) then  state\_reg <= state\_next;  end if;  end process;    process(state\_reg,start)  begin  state\_next <= state\_reg;  case state\_reg is  when idle=>  if start='1' then  state\_next <= out1;  end if;  when out1=>  state\_next <= out2;  data\_out <= state\_reg(0);  when out2=>  state\_next <= out3;  data\_out <= state\_reg(0);  when out3=>  state\_next <= out4;  data\_out <= state\_reg(0);  when out4=>  state\_next <= out5;  data\_out <= state\_reg(0);  when out5=>  state\_next <= out6;  data\_out <= state\_reg(0);  when out6=>  state\_next <= out7;  data\_out <= state\_reg(0);  when out7=>  state\_next <= out8;  data\_out <= state\_reg(0);  when others=>  state\_next <= idle;  data\_out <= state\_reg(0);  end case;  end process;  end clever\_arch; |

Look-ahead

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  entity preamble\_look is  port(  clk: in std\_logic;  start: in std\_logic;  data\_out: out std\_logic  );  end preamble\_look;  architecture look\_arch of preamble\_look is  type state\_type is  (idle,out1,out2,out3,out4,out5,out6,out7,out8);  signal state\_reg, state\_next: state\_type := idle;  signal data\_buffer: std\_logic := '0';  begin  process(clk)  begin  if(rising\_edge(clk)) then  state\_reg <= state\_next;  data\_out <= data\_buffer;  end if;  end process;    process(state\_reg,start)  begin  state\_next <= state\_reg;  case state\_reg is  when idle=>  if start='1' then  state\_next <= out1;  data\_buffer <= '1';  end if;  when out1=>  state\_next <= out2;  data\_buffer <= '0';  when out2=>  state\_next <= out3;  data\_buffer <= '1';  when out3=>  state\_next <= out4;  data\_buffer <= '0';  when out4=>  state\_next <= out5;  data\_buffer <= '1';  when out5=>  state\_next <= out6;  data\_buffer <= '0';  when out6=>  state\_next <= out7;  data\_buffer <= '1';  when out7=>  state\_next <= out8;  data\_buffer <= '0';  when out8=>  state\_next <= idle;  data\_buffer <= '0';  end case;  end process;    end look\_arch; |

Divider\_multi\_segments

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity divider\_multi is  port(  clk: in std\_logic;  y\_in: in std\_logic\_vector(7 downto 0);  d: in std\_logic\_vector(7 downto 0);  r\_out: out std\_logic\_vector(7 downto 0);  q\_out: out std\_logic\_vector(7 downto 0)  );    end divider\_multi;  architecture divider\_arch of divider\_multi is  type state\_type is  (idle,y\_less\_d,load,op,stop);  signal state\_reg, state\_next: state\_type := idle;  signal y\_reg, y\_next: unsigned(7 downto 0);  signal q\_reg, q\_next: unsigned(7 downto 0);  -- control path: state register  begin  process(clk)  begin  if(rising\_edge(clk)) then  state\_reg <= state\_next;  end if;  end process;    -- control path: next-state logic  process(state\_reg)  begin  state\_next <= state\_reg;  case state\_reg is  when idle=>  q\_out <= (others=>'0');  r\_out <= (others=>'0');  if y\_in <= d then  state\_next <= y\_less\_d;  else  state\_next <= load;  end if;  when y\_less\_d=>  state\_next <= idle;  q\_out <= (others=>'0');  r\_out <= y\_in;  when load=>  state\_next <= op;  when op=>  if(y\_reg <= unsigned(d)) then  state\_next <= stop;  else  state\_next <= state\_reg;  end if;  when stop=>  state\_next <= idle;  end case;  end process;    -- control path: output logic  r\_out <= y\_in when state\_reg=y\_less\_d else  std\_logic\_vector(y\_reg) when state\_reg=stop else  (others=>'0');  q\_out <= std\_logic\_vector(q\_reg) when state\_reg=stop else (others=>'0');  -- data path: data register  process(clk)  begin  if(rising\_edge(clk)) then  y\_reg <= y\_next;  q\_reg <= q\_next;  end if;  end process;    -- data path: routing multiplexer  process(state\_reg,y\_reg,q\_reg)  begin  case state\_reg is  when idle=>  y\_next <= y\_reg;  q\_next <= q\_reg;  when y\_less\_d=>  y\_next <= y\_reg;  q\_next <= q\_reg;  when load=>  y\_next <= unsigned(y\_in);  q\_next <= (others=>'0');  when op=>  y\_next <= y\_reg-unsigned(d);  q\_next <= q\_reg+1;  when stop=>  y\_next <= y\_reg;  q\_next <= q\_reg;  end case;  end process;  end divider\_arch; |

Divider\_ two\_segment

|  |
| --- |
| library ieee;  use ieee.std\_logic\_1164.all;  use ieee.numeric\_std.all;  entity divider\_two is  port(  clk: in std\_logic;  y\_in: in std\_logic\_vector(7 downto 0);  d: in std\_logic\_vector(7 downto 0);  r\_out: out std\_logic\_vector(7 downto 0);  q\_out: out std\_logic\_vector(7 downto 0)  );    end divider\_two;  architecture divider\_arch of divider\_two is  type state\_type is  (idle,y\_less\_d,load,op,stop);  signal state\_reg, state\_next: state\_type := idle;  signal y\_reg, y\_next: unsigned(7 downto 0);  signal q\_reg, q\_next: unsigned(7 downto 0);  begin  -- state and data registers  process(clk)  begin  if(rising\_edge(clk)) then  state\_reg <= state\_next;  y\_reg <= y\_next;  q\_reg <= q\_next;  q\_reg <= q\_next;    end if;  end process;    -- combinational circuit  process(state\_reg,y\_reg,q\_reg)  begin  state\_next <= state\_reg;  y\_next <= y\_reg;  q\_next <= q\_reg;    case state\_reg is  when idle=>  q\_out <= (others=>'0');  r\_out <= (others=>'0');  if y\_in <= d then  state\_next <= y\_less\_d;  else  state\_next <= load;  end if;  when y\_less\_d=>  state\_next <= idle;  q\_out <= (others=>'0');  r\_out <= y\_in;  when load=>  state\_next <= op;  y\_next <= unsigned(y\_in);  q\_next <= (others=>'0');  when op=>  y\_next <= y\_reg-unsigned(d);  q\_next <= q\_reg+1;  if(y\_reg <= unsigned(d)) then  state\_next <= stop;  else  state\_next <= state\_reg;  end if;  when stop=>  state\_next <= idle;  end case;  end process;    -- control path: output logic  r\_out <= y\_in when state\_reg=y\_less\_d else  std\_logic\_vector(y\_reg) when state\_reg=stop else  (others=>'0');  q\_out <= std\_logic\_vector(q\_reg) when state\_reg=stop else (others=>'0');    end divider\_arch; |