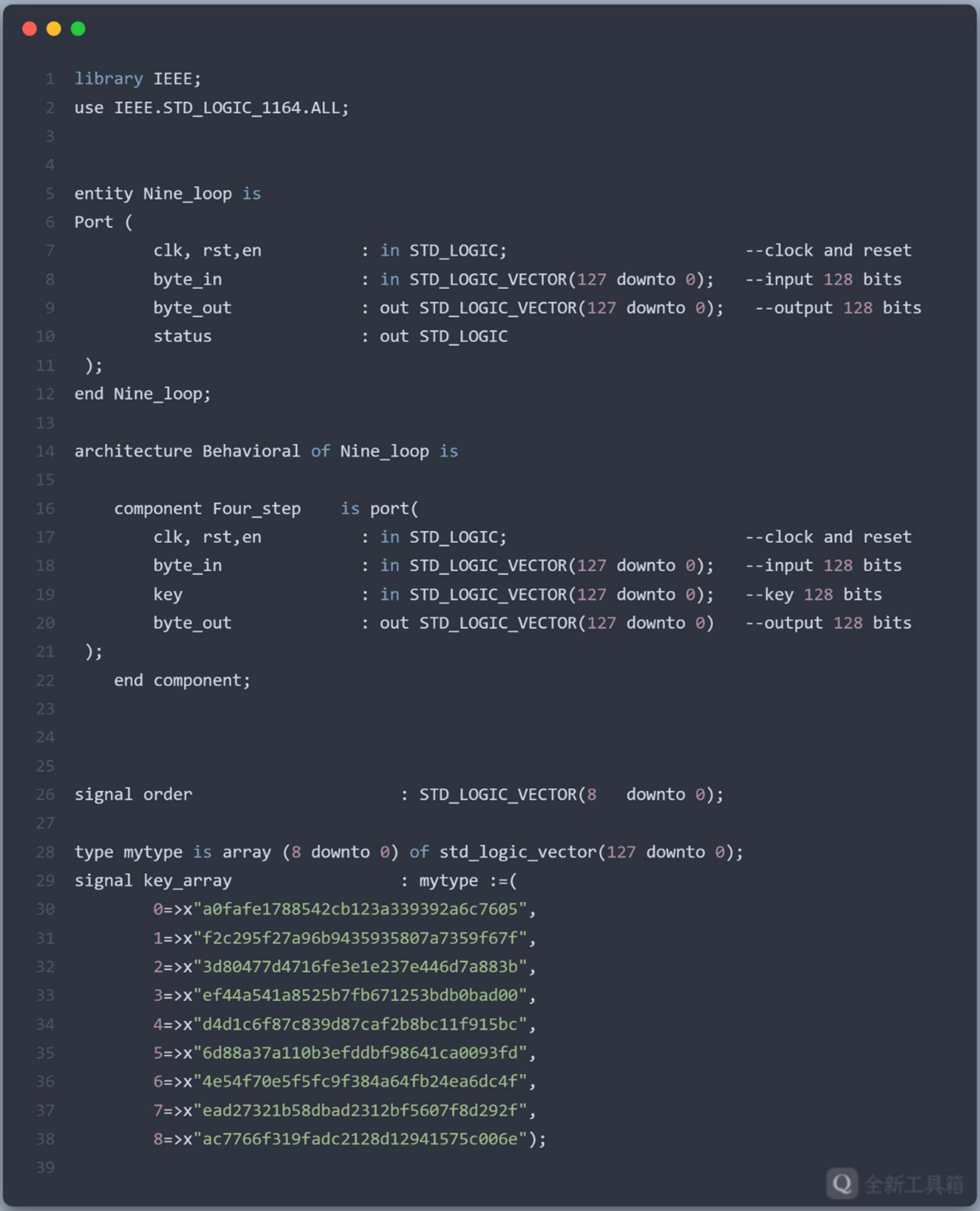
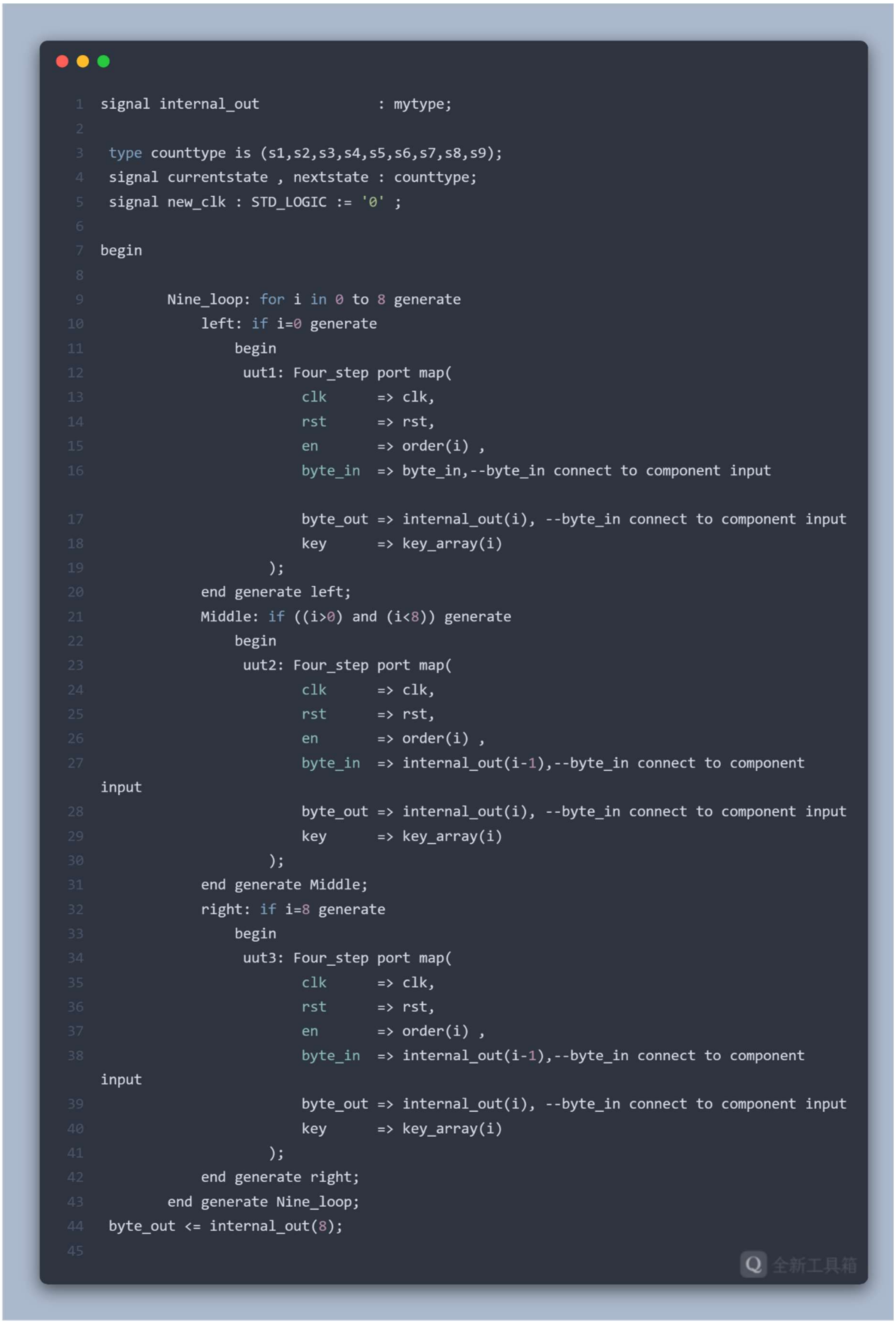
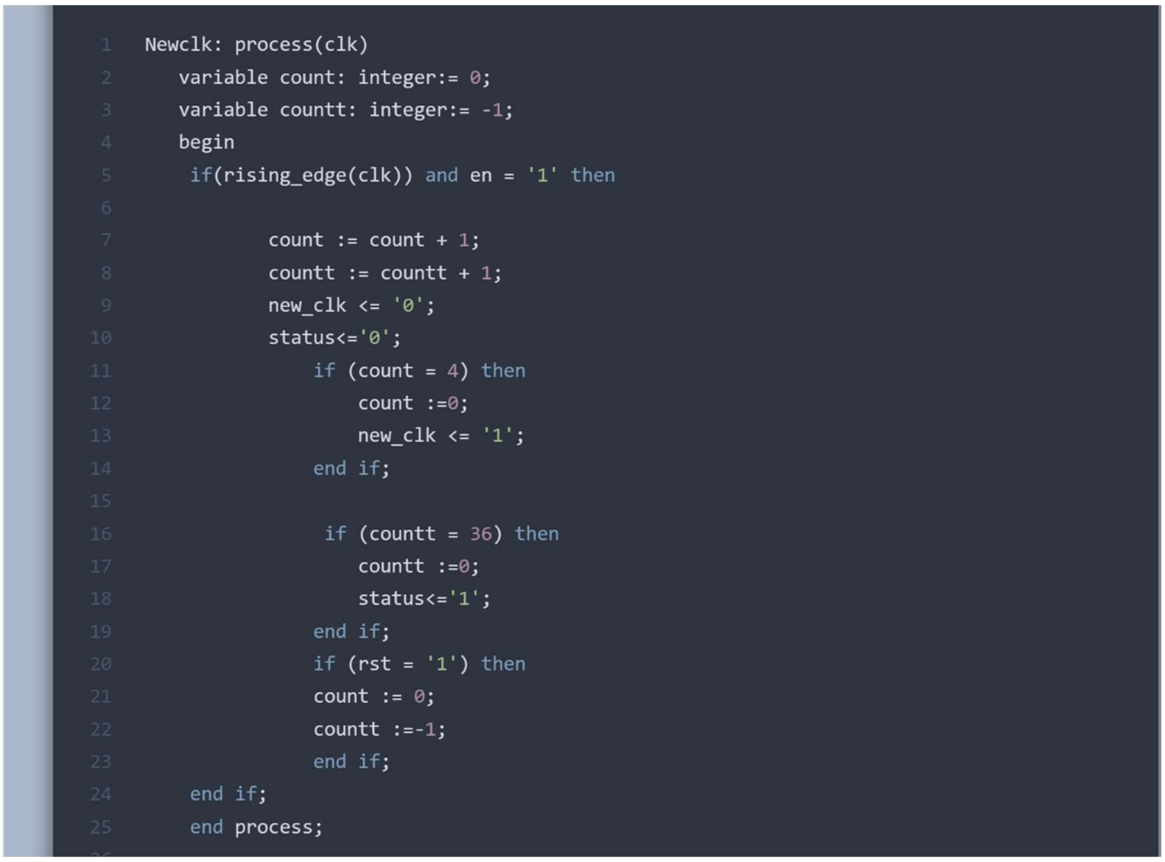


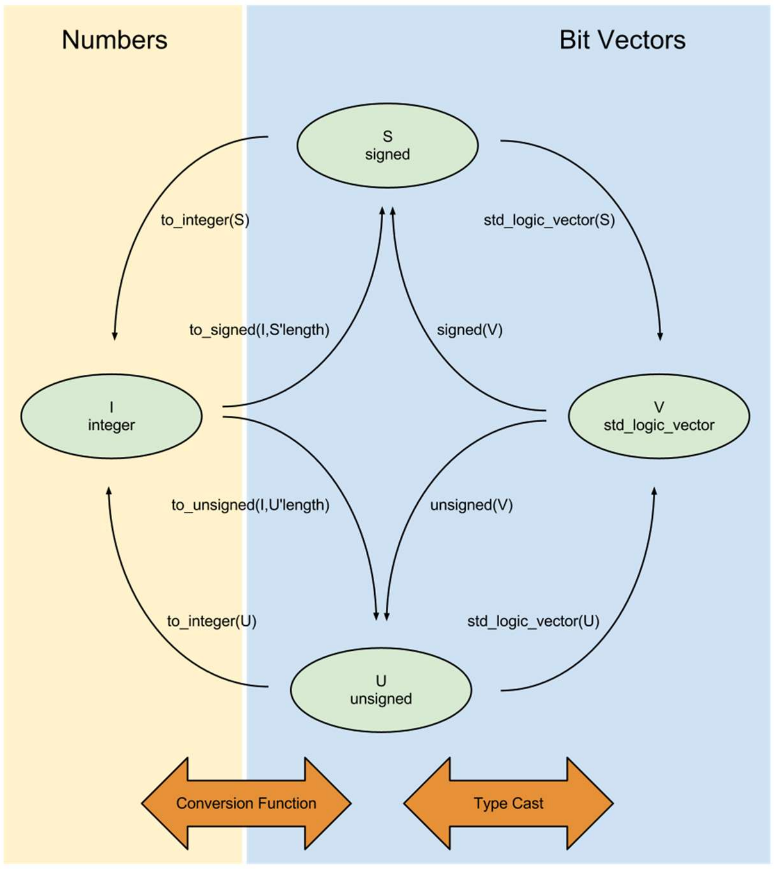
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity Nine_loop is
Port (
        clk, rst,en          : in STD_LOGIC;                        --clock and reset
        byte_in              : in STD_LOGIC_VECTOR(127 downto 0);   --input 128 bits
        byte_out             : out STD_LOGIC_VECTOR(127 downto 0);   --output 128 bits
        status               : out STD_LOGIC
 );
end Nine_loop;

architecture Behavioral of Nine_loop is

    component Four_step    is port(
        clk, rst,en          : in STD_LOGIC;                        --clock and reset
        byte_in              : in STD_LOGIC_VECTOR(127 downto 0);   --input 128 bits
        key                  : in STD_LOGIC_VECTOR(127 downto 0);   --key 128 bits
        byte_out             : out STD_LOGIC_VECTOR(127 downto 0)   --output 128 bits
 );
    end component;



signal order                     : STD_LOGIC_VECTOR(8   downto 0);

type mytype is array (8 downto 0) of std_logic_vector(127 downto 0);
signal key_array                 : mytype :=(
        0=>x"a0fafe1788542cb123a339392a6c7605",
        1=>x"f2c295f27a96b9435935807a7359f67f",
        2=>x"3d80477d4716fe3e1e237e446d7a883b",
        3=>x"ef44a541a8525b7fb671253bdb0bad00",
        4=>x"d4d1c6f87c839d87caf2b8bc11f915bc",
        5=>x"6d88a37a110b3efddbf98641ca0093fd",
        6=>x"4e54f70e5f5fc9f384a64fb24ea6dc4f",
        7=>x"ead27321b58dbad2312bf5607f8d292f",
        8=>x"ac7766f319fadc2128d12941575c006e");

```

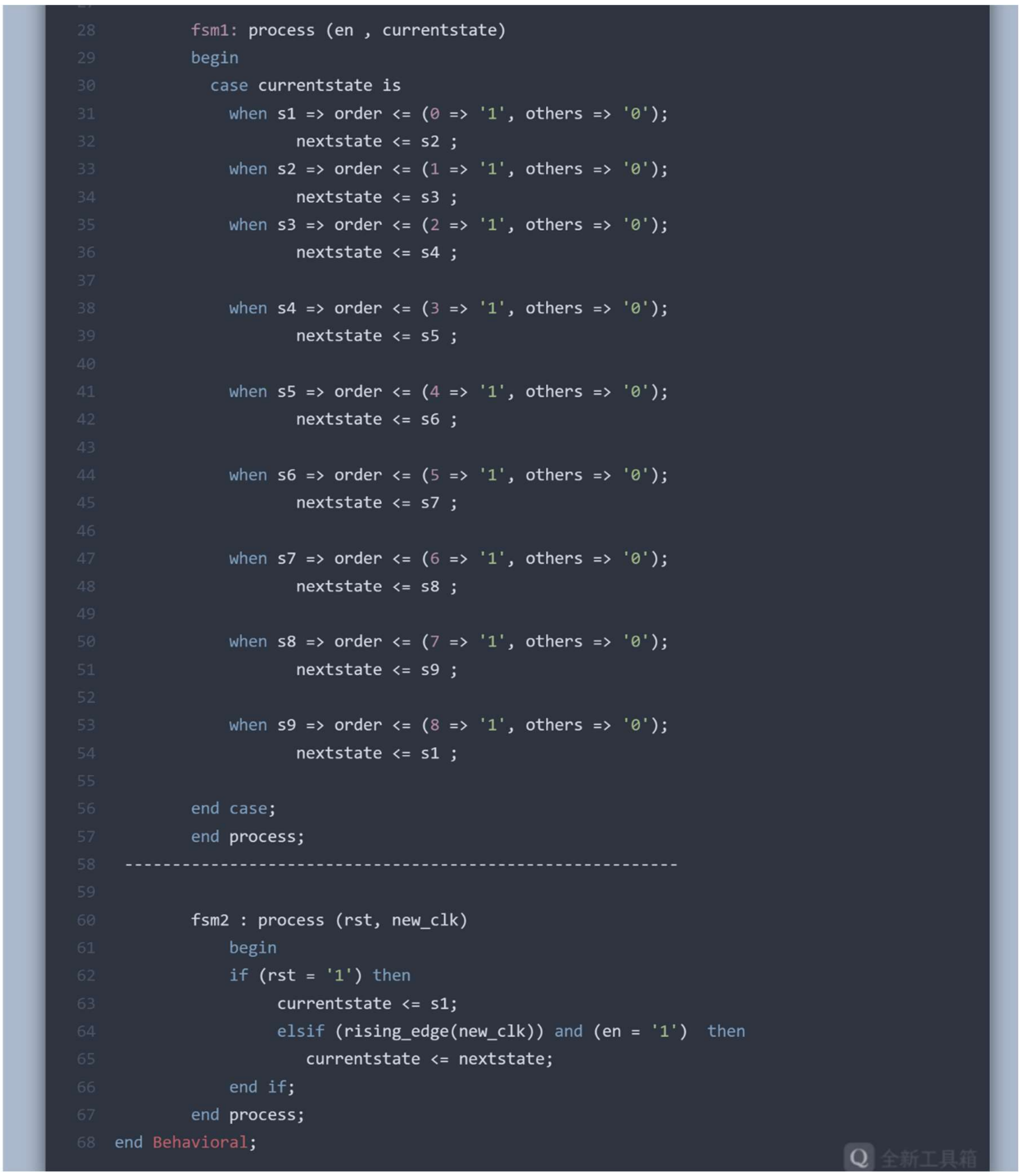


```vhdl
signal internal_out              : mytype;

 type counttype is (s1,s2,s3,s4,s5,s6,s7,s8,s9);
 signal currentstate , nextstate : counttype;
 signal new_clk : STD_LOGIC := '0' ;

begin

        Nine_loop: for i in 0 to 8 generate
            left: if i=0 generate
                begin
                 uut1: Four_step port map(
                        clk      => clk,
                        rst      => rst,
                        en       => order(i) ,
                        byte_in  => byte_in,--byte_in connect to component input

                        byte_out => internal_out(i), --byte_in connect to component input
                        key      => key_array(i)
                    );
            end generate left;
            Middle: if ((i>0) and (i<8)) generate
                begin
                 uut2: Four_step port map(
                        clk      => clk,
                        rst      => rst,
                        en       => order(i) ,
                        byte_in  => internal_out(i-1),--byte_in connect to component input
                        byte_out => internal_out(i), --byte_in connect to component input
                        key      => key_array(i)
                    );
            end generate Middle;
            right: if i=8 generate
                begin
                 uut3: Four_step port map(
                        clk      => clk,
                        rst      => rst,
                        en       => order(i) ,
                        byte_in  => internal_out(i-1),--byte_in connect to component input
                        byte_out => internal_out(i), --byte_in connect to component input
                        key      => key_array(i)
                    );
            end generate right;
        end generate Nine_loop;
 byte_out <= internal_out(8);
```



```
Newclk: process(clk)
   variable count: integer:= 0;
   variable countt: integer:= -1;
   begin
    if(rising_edge(clk)) and en = '1' then

           count := count + 1;
           countt := countt + 1;
           new_clk <= '0';
           status<='0';
               if (count = 4) then
                   count :=0;
                   new_clk <= '1';
               end if;

                if (countt = 36) then
                   countt :=0;
                   status<='1';
               end if;
               if (rst = '1') then
               count := 0;
               countt :=-1;
               end if;
    end if;
    end process;
```





```
        fsm1: process (en , currentstate)
        begin
          case currentstate is
            when s1 => order <= (0 => '1', others => '0');
                   nextstate <= s2 ;
            when s2 => order <= (1 => '1', others => '0');
                   nextstate <= s3 ;
            when s3 => order <= (2 => '1', others => '0');
                   nextstate <= s4 ;

            when s4 => order <= (3 => '1', others => '0');
                   nextstate <= s5 ;

            when s5 => order <= (4 => '1', others => '0');
                   nextstate <= s6 ;

            when s6 => order <= (5 => '1', others => '0');
                   nextstate <= s7 ;

            when s7 => order <= (6 => '1', others => '0');
                   nextstate <= s8 ;

            when s8 => order <= (7 => '1', others => '0');
                   nextstate <= s9 ;

            when s9 => order <= (8 => '1', others => '0');
                   nextstate <= s1 ;

        end case;
        end process;
 ----------------------------------------------------------

        fsm2 : process (rst, new_clk)
            begin
            if (rst = '1') then
                 currentstate <= s1;
                 elsif (rising_edge(new_clk)) and (en = '1')  then
                    currentstate <= nextstate;
            end if;
        end process;
end Behavioral;
```