

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity Nine_loop is
6  Port (
7      clk, rst,en      : in STD_LOGIC;           --clock and reset
8      byte_in          : in STD_LOGIC_VECTOR(127 downto 0); --input 128 bits
9      byte_out          : out STD_LOGIC_VECTOR(127 downto 0); --output 128 bits
10     status            : out STD_LOGIC
11 );
12 end Nine_loop;
13
14 architecture Behavioral of Nine_loop is
15
16     component Four_step    is port(
17         clk, rst,en        : in STD_LOGIC;           --clock and reset
18         byte_in            : in STD_LOGIC_VECTOR(127 downto 0); --input 128 bits
19         key                : in STD_LOGIC_VECTOR(127 downto 0); --key 128 bits
20         byte_out           : out STD_LOGIC_VECTOR(127 downto 0) --output 128 bits
21 );
22     end component;
23
24
25
26     signal order          : STD_LOGIC_VECTOR(8   downto 0);
27
28     type mytype is array (8 downto 0) of std_logic_vector(127 downto 0);
29     signal key_array      : mytype :=(
30         0=>x"a0fafa1788542cb123a339392a6c7605",
31         1=>x"f2c295f27a96b9435935807a7359f67f",
32         2=>x"3d80477d4716fe3e1e237e446d7a883b",
33         3=>x"ef44a541a8525b7fb671253bdb0bad00",
34         4=>x"d4d1c6f87c839d87caf2b8bc11f915bc",
35         5=>x"6d88a37a110b3efddbf98641ca0093fd",
36         6=>x"4e54f70e5f5fc9f384a64fb24ea6dc4f",
37         7=>x"ead27321b58dbad2312bf5607f8d292f",
38         8=>x"ac7766f319fadc2128d12941575c006e");
39

```

```

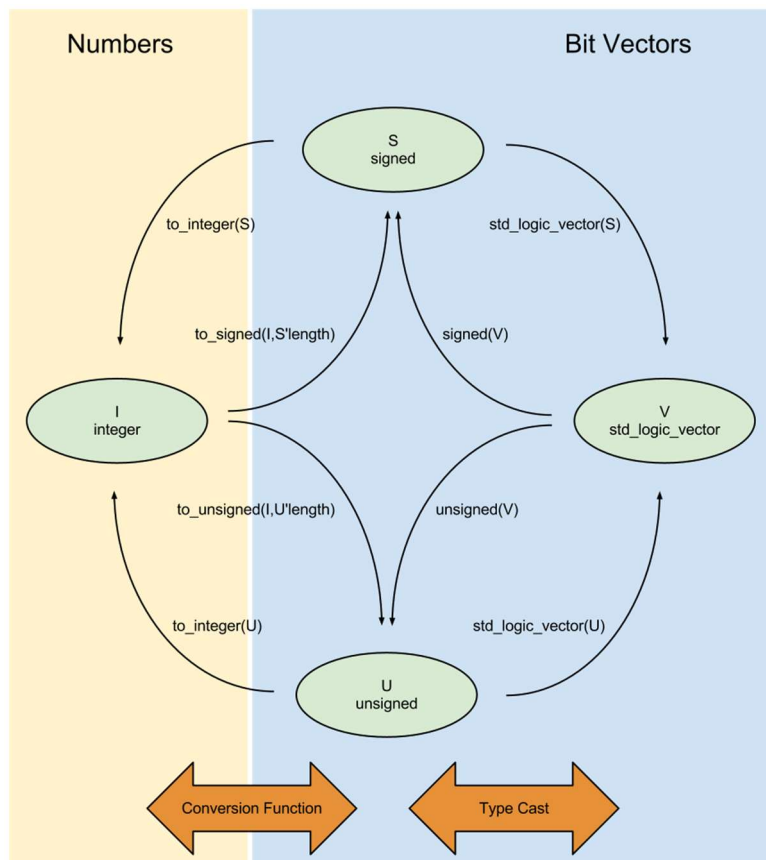
1  signal internal_out          : mytype;
2
3  type counttype is (s1,s2,s3,s4,s5,s6,s7,s8,s9);
4  signal currentstate , nextstate : counttype;
5  signal new_clk : STD_LOGIC := '0' ;
6
7  begin
8
9      Nine_loop: for i in 0 to 8 generate
10         left: if i=0 generate
11             begin
12                 uut1: Four_step port map(
13                     clk      => clk,
14                     rst      => rst,
15                     en       => order(i) ,
16                     byte_in  => byte_in,--byte_in connect to component input
17
18                     byte_out => internal_out(i), --byte_in connect to component input
19                     key      => key_array(i)
20                 );
21             end generate left;
22         Middle: if ((i>0) and (i<8)) generate
23             begin
24                 uut2: Four_step port map(
25                     clk      => clk,
26                     rst      => rst,
27                     en       => order(i) ,
28                     byte_in  => internal_out(i-1),--byte_in connect to component
29                     input
30                     byte_out => internal_out(i), --byte_in connect to component input
31                     key      => key_array(i)
32                 );
33             end generate Middle;
34         right: if i=8 generate
35             begin
36                 uut3: Four_step port map(
37                     clk      => clk,
38                     rst      => rst,
39                     en       => order(i) ,
40                     byte_in  => internal_out(i-1),--byte_in connect to component
41                     input
42                     byte_out => internal_out(i), --byte_in connect to component input
43                     key      => key_array(i)
44                 );
45             end generate right;
46         end generate Nine_loop;
47     byte_out <= internal_out(8);
48

```

```

1  Newclk: process(clk)
2      variable count: integer:= 0;
3      variable countt: integer:= -1;
4      begin
5          if(rising_edge(clk)) and en = '1' then
6
7              count := count + 1;
8              countt := countt + 1;
9              new_clk <= '0';
10             status<='0';
11             if (count = 4) then
12                 count :=0;
13                 new_clk <= '1';
14             end if;
15
16             if (countt = 36) then
17                 countt :=0;
18                 status<='1';
19             end if;
20             if (rst = '1') then
21                 count := 0;
22                 countt :=-1;
23             end if;
24         end if;
25     end process;
26

```



```

27
28     fsm1: process (en , currentstate)
29     begin
30         case currentstate is
31             when s1 => order <= (0 => '1', others => '0');
32                 nextstate <= s2 ;
33             when s2 => order <= (1 => '1', others => '0');
34                 nextstate <= s3 ;
35             when s3 => order <= (2 => '1', others => '0');
36                 nextstate <= s4 ;
37
38             when s4 => order <= (3 => '1', others => '0');
39                 nextstate <= s5 ;
40
41             when s5 => order <= (4 => '1', others => '0');
42                 nextstate <= s6 ;
43
44             when s6 => order <= (5 => '1', others => '0');
45                 nextstate <= s7 ;
46
47             when s7 => order <= (6 => '1', others => '0');
48                 nextstate <= s8 ;
49
50             when s8 => order <= (7 => '1', others => '0');
51                 nextstate <= s9 ;
52
53             when s9 => order <= (8 => '1', others => '0');
54                 nextstate <= s1 ;
55
56         end case;
57     end process;
58     -----
59
60     fsm2 : process (rst, new_clk)
61     begin
62         if (rst = '1') then
63             currentstate <= s1;
64             elsif (rising_edge(new_clk)) and (en = '1') then
65                 currentstate <= nextstate;
66         end if;
67     end process;
68 end Behavioral;

```