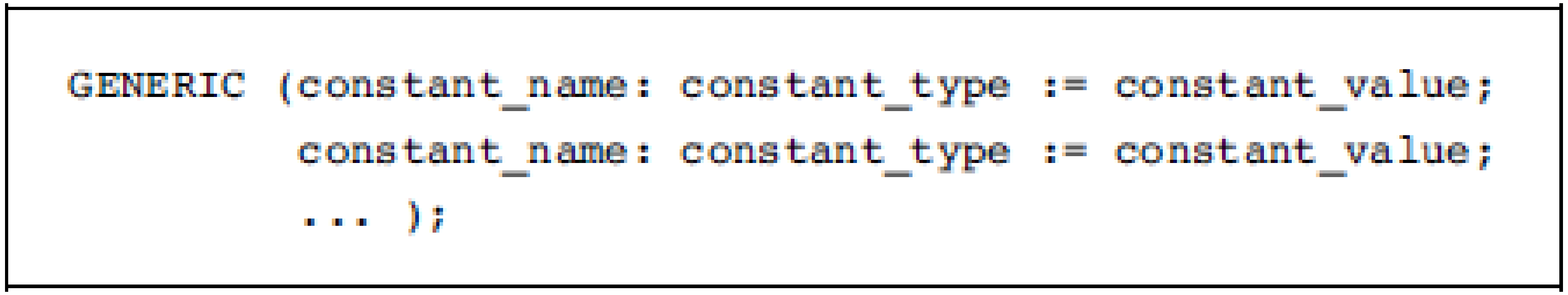
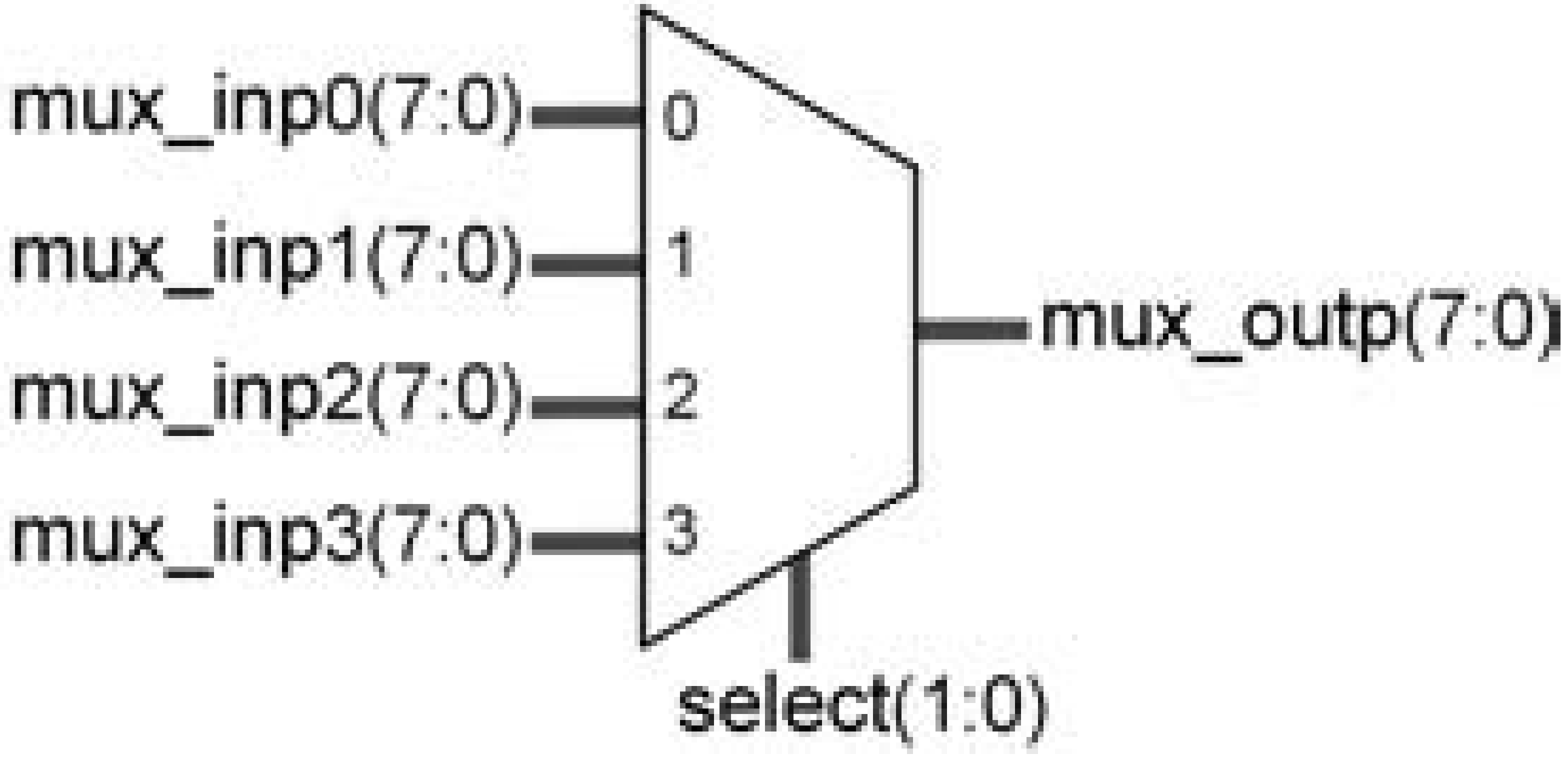
Generic statements in VHDL

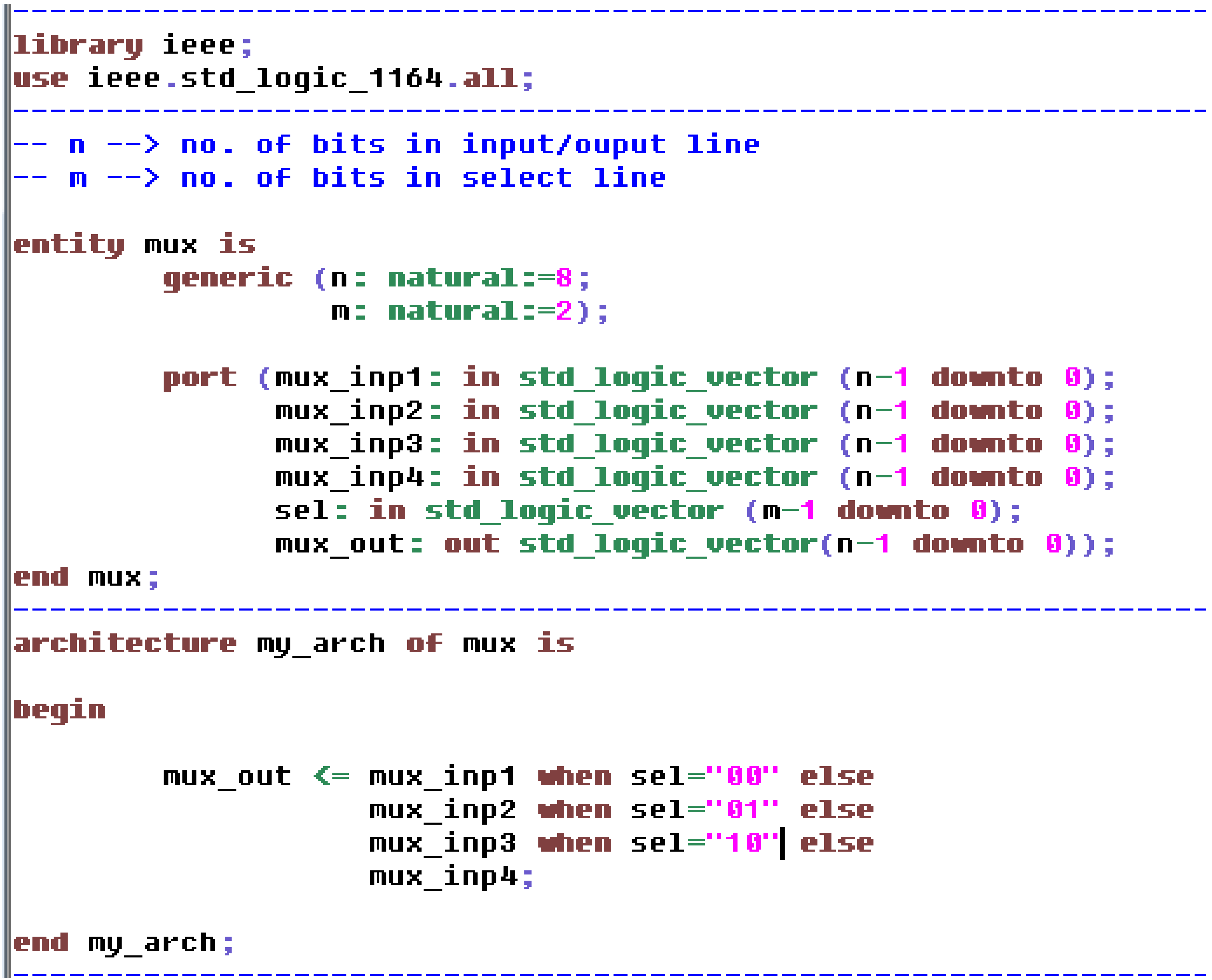
- GENERIC declarations allow the specification of generic parameters (that is, generic constants, which can be easily modified or adapted to different applications). Their purpose is to parameterize a design, conferring the code more flexibility and reusability.
- Generic declarations can be done within the ENTITY declaration. GENERIC is the only declaration allowed before the PORT clause, which causes such constants to be truly global because they can be used even in the PORT specifications.
- A simplified syntax for GENERIC declarations is shown below.



```
GENERIC (constant_name: constant_type := constant_value;
         constant_name: constant_type := constant_value;
         ... );
```

Example of a generic multiplexer





```vhdl
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------------------------------
-- n --> no. of bits in input/ouput line
-- m --> no. of bits in select line

entity mux is
        generic (n: natural:=8;
                 m: natural:=2);

        port (mux_inp1: in std_logic_vector (n-1 downto 0);
              mux_inp2: in std_logic_vector (n-1 downto 0);
              mux_inp3: in std_logic_vector (n-1 downto 0);
              mux_inp4: in std_logic_vector (n-1 downto 0);
              sel: in std_logic_vector (m-1 downto 0);
              mux_out: out std_logic_vector(n-1 downto 0));
end mux;
----------------------------------------------------------------
architecture my_arch of mux is

begin

        mux_out <= mux_inp1 when sel="00" else
                   mux_inp2 when sel="01" else
                   mux_inp3 when sel="10" else
                   mux_inp4;

end my_arch;
----------------------------------------------------------------
```

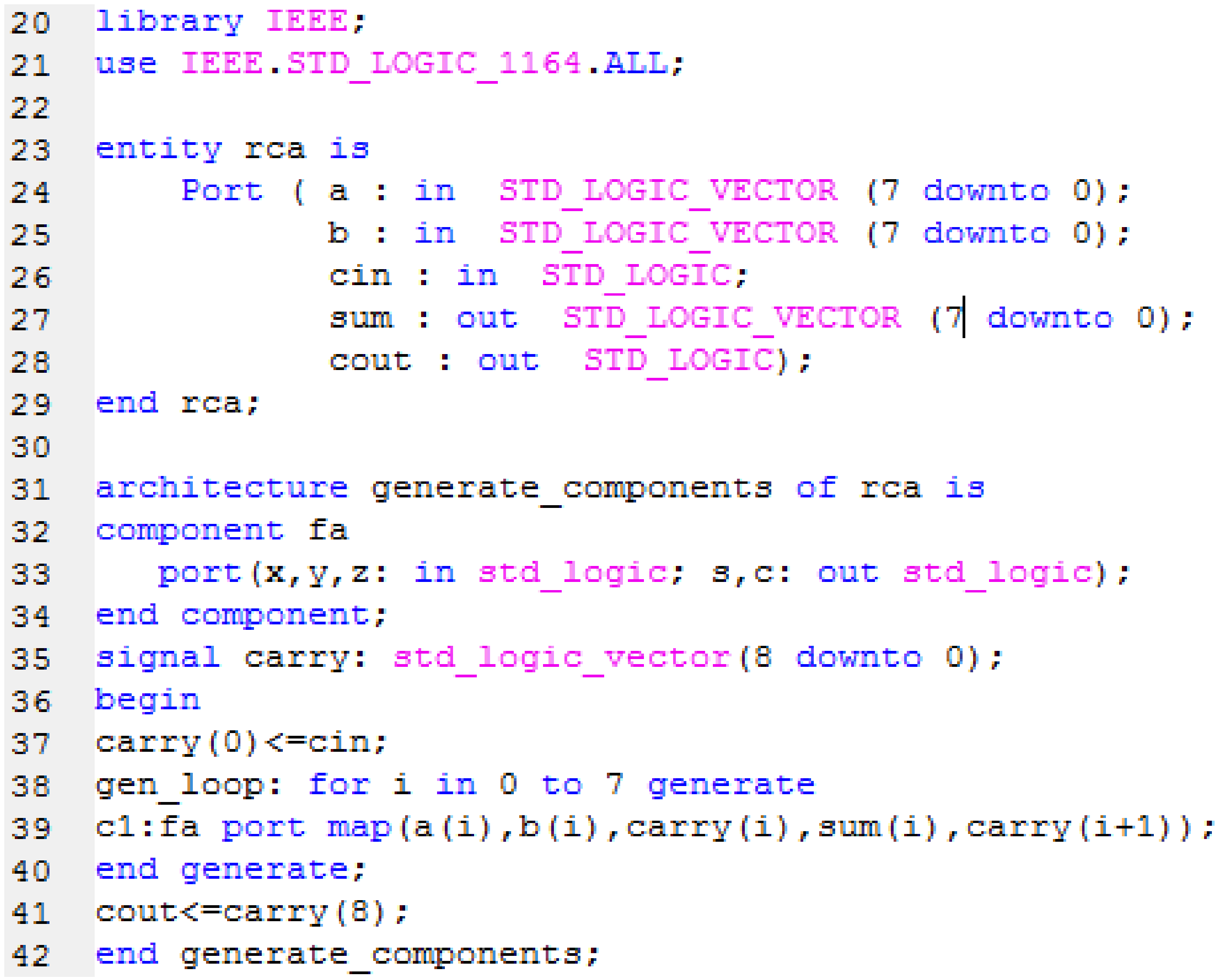
Generate Statement

- GENERATE is another concurrent statement. In its most popular form, it is equivalent to the sequential statement LOOP in the sense that it too is employed to have a section of code repeated a number of times.

- FOR-GENERATE statement is used to create multiple instances of a section of code. A simplified syntax for it is shown below. Notice that a label is required and that the word BEGIN is only needed when declarations are made.

```
label: FOR identifier IN range GENERATE
  [declarative_part
BEGIN]
  concurrent_statements_part
END GENERATE [label];
```

Generate statement with component instantiations

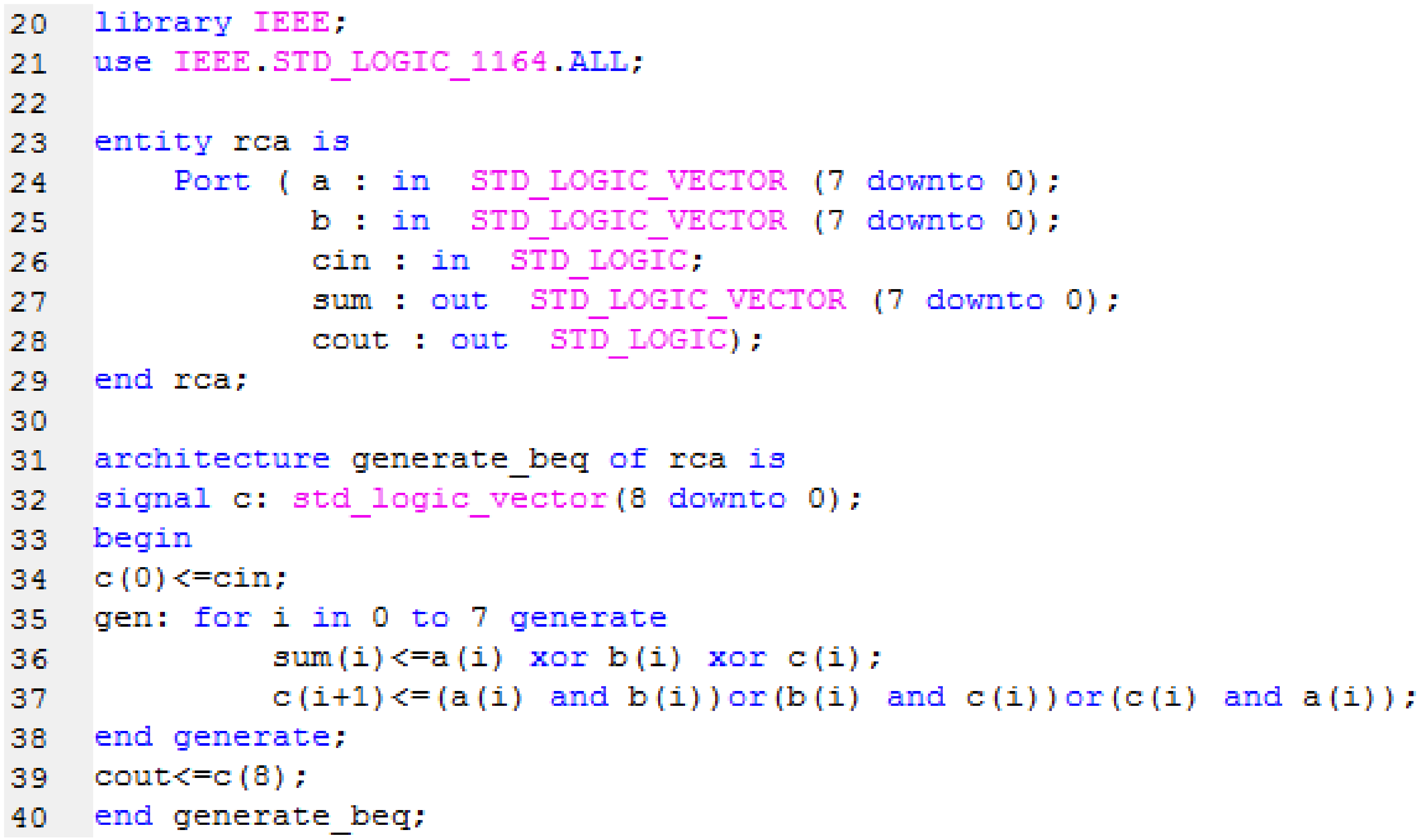


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rca is
    Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC_VECTOR (7 downto 0);
           cout : out  STD_LOGIC);
end rca;

architecture generate_components of rca is
component fa
   port(x,y,z: in std_logic; s,c: out std_logic);
end component;
signal carry: std_logic_vector(8 downto 0);
begin
carry(0)<=cin;
gen_loop: for i in 0 to 7 generate
c1:fa port map(a(i),b(i),carry(i),sum(i),carry(i+1));
end generate;
cout<=carry(8);
end generate_components;
```

Generate statement with Boolean expressions



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

entity rca is
    Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           cin : in  STD_LOGIC;
           sum : out  STD_LOGIC_VECTOR (7 downto 0);
           cout : out  STD_LOGIC);
end rca;

architecture generate_beq of rca is
signal c: std_logic_vector(8 downto 0);
begin
c(0)<=cin;
gen: for i in 0 to 7 generate
         sum(i)<=a(i) xor b(i) xor c(i);
         c(i+1)<=(a(i) and b(i))or(b(i) and c(i))or(c(i) and a(i));
end generate;
cout<=c(8);
end generate_beq;
```