Lecture 9 (Chapter 2 + more)

VHDL: Sequential design and FSM

Khaza Anuarul Hoque
ECE 4250/7250

VHDL Libraries

- Extend the functionality of VHDL.
- Define types, functions, components, and overloaded operators.
- Need a library statement and a use statement.

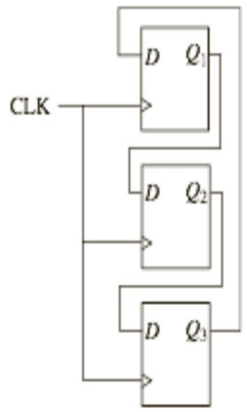
```
library IEEE;
use IEEE.numeric_bit.all;
```

- The **numeric_bit** package uses bit_vectors to represent unsigned and signed binary numbers.

Modeling Registers Using VHDL

- When several flip-flops change state on the same clock edge, statements representing these flip-flops can be placed in the same clocked processes.
- Example:

```
process(CLK)
   begin
   if CLK'event and CLK = '1' then
      Q1 <= Q3 after 5 ns;
      Q2 <= Q1 after 5 ns;
      Q3 <= Q2 after 5 ns;
   end if;
end process;
```



- These flip-flops all change state following the rising edge of the clock, and the three statements in the always statement execute in sequence with no delay.

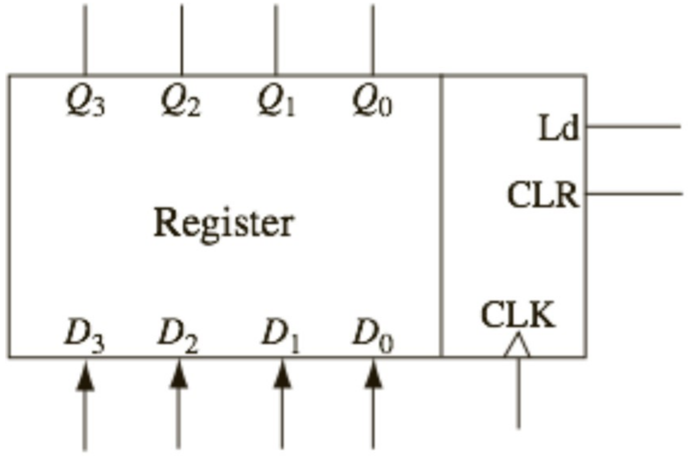
Modeling Registers VHDL (cont.)

- If we were to omit the delay in the preceding example and replace the sequential statements with:

```
Q1 <= Q3;  Q2 <= Q1;  Q3 <= Q2;
```

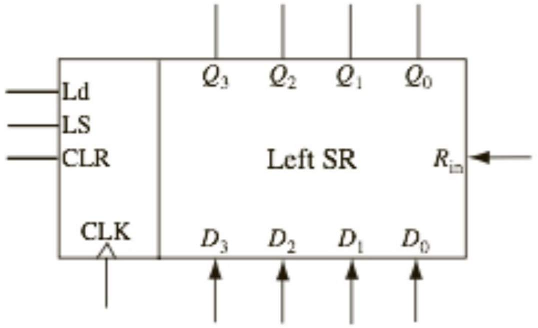
the operation would be essentially the same. The three statements execute in sequence in zero time, and then the Q’s values change after a delta delay.

Register with synchronous clear and load



```
process(CLK)
begin
  if CLK'event and CLK = '1' then
    if CLR = '1' then Q <= "0000";
    elsif Ld = '1' then Q <= D;
    end if;
  end if;
end process;
```

Left shift register with synchronous clear and load



```
process(CLK)
begin
  if CLK'event and CLK = '1' then
    if CLR = '1' then Q <= "0000";
    elsif Ld = '1' then Q <= D;
    elsif LS = '1' then Q <= Q(2 downto 0) & Rin;
    end if;
  end if;
end process;
```

Loops in VHDL

- Activity occurring in a repetitive way.
- Statements are sequential.
- Kinds of loop statements: **for**, **while**.
- Infinite loop:
  - General form:

```
[loop-label:] loop
  sequential statements
end loop [loop-label];
```

  - Can be terminated using exit statements:

```
exit; or exit when condition;
```

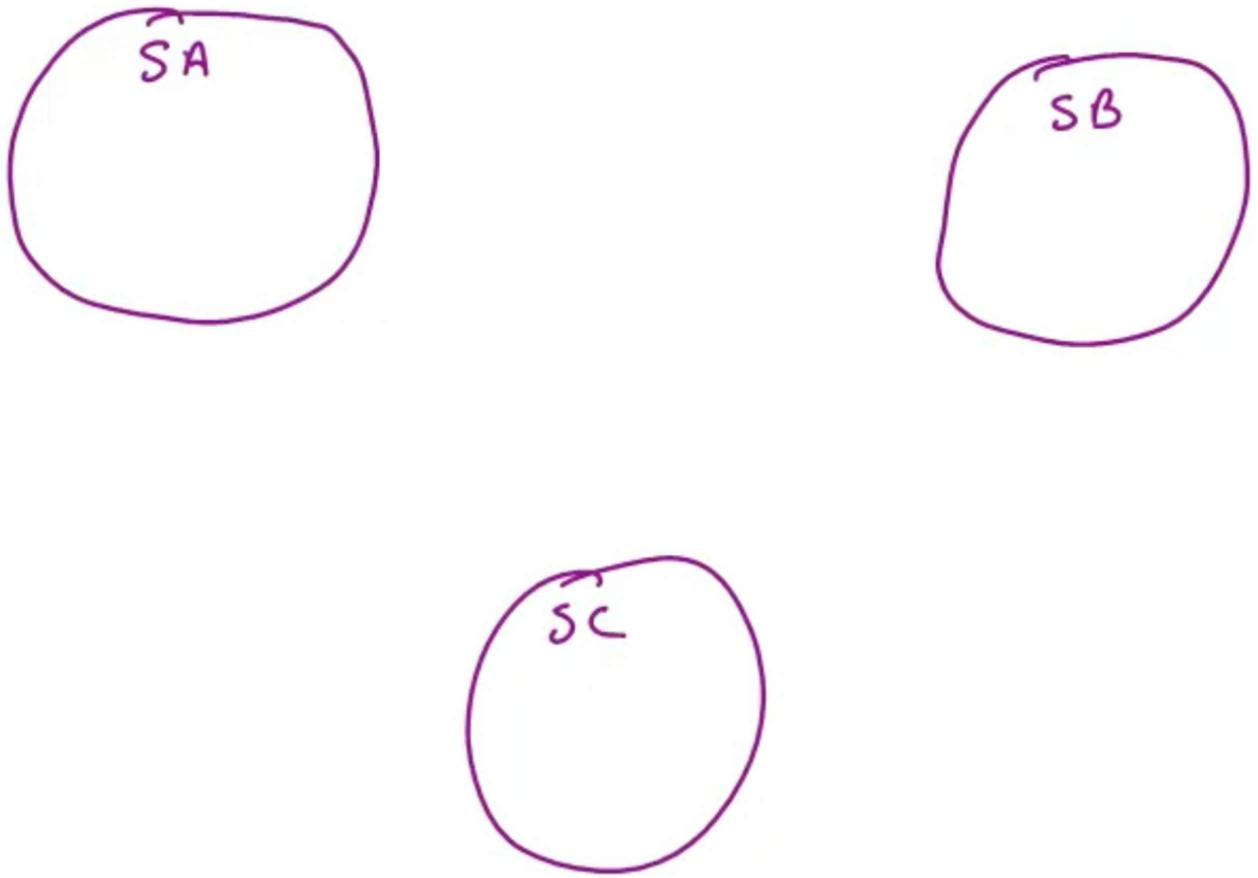
Loops in VHDL (continued)

- for loop:
  - General form:
```
[loop-label:] for loop-index in range loop
  sequential statements
end loop [loop-label];
```
  - Loop-index is incremented at the end of each loop.
  - Continues for every value in the range.
- while loop:
  - General form:
```
[loop-label:] while condition loop
  sequential statements
end loop [loop-label];
```
  - Condition is tested at the beginning of each loop.
  - Loop is terminated if condition is false.

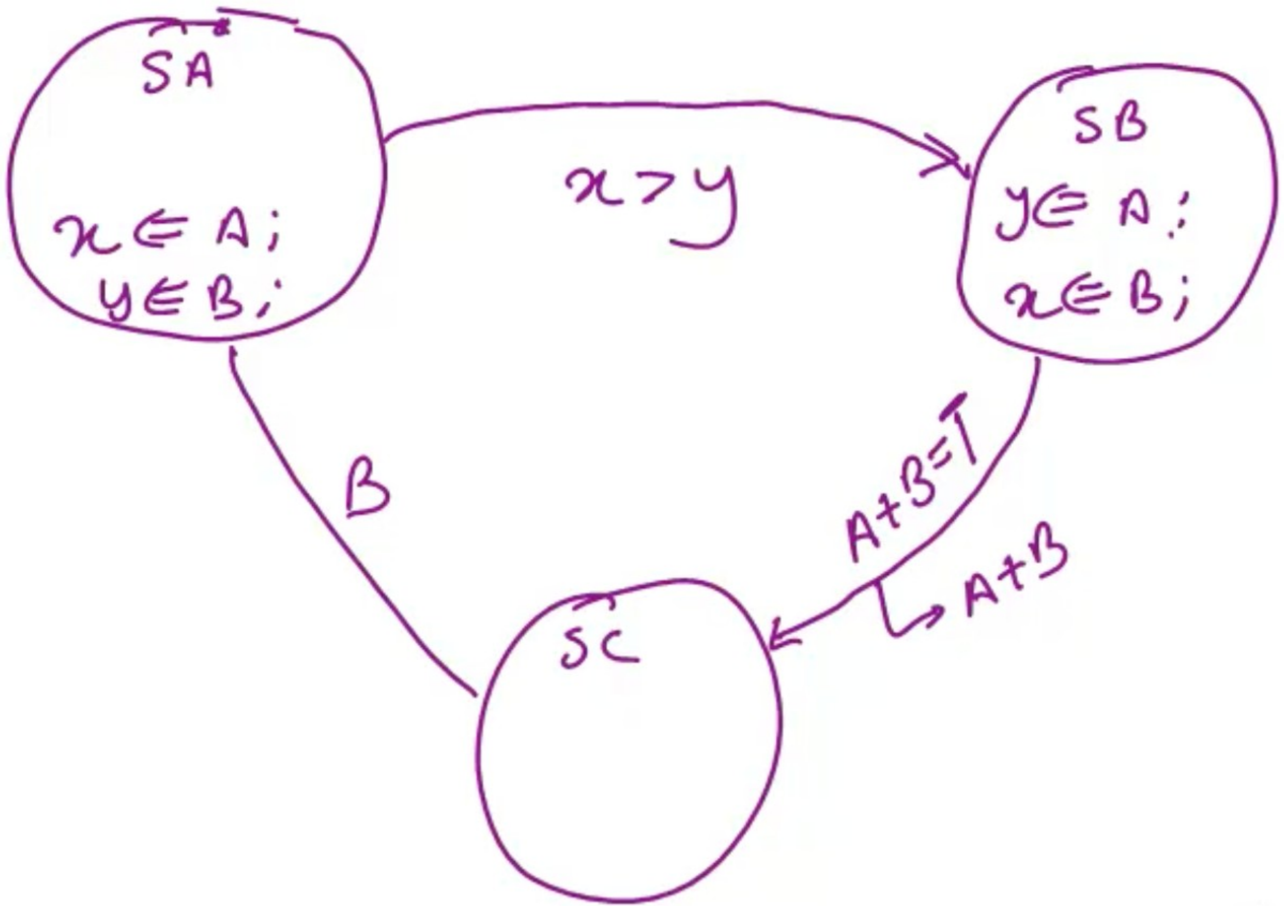
Finite State Machines (FSMs)

- A Circuit/System/Machine that
  - Goes through a fixed number of states
  - Has fixed number of Input/Output combinations
- Used typically to
  - Control
  - Monitor
  - Calculate
  - Communication protocols, where data may be sent asynchronously or different data required different responses
  - Control of simple machine
  - Implementing simple user interfaces

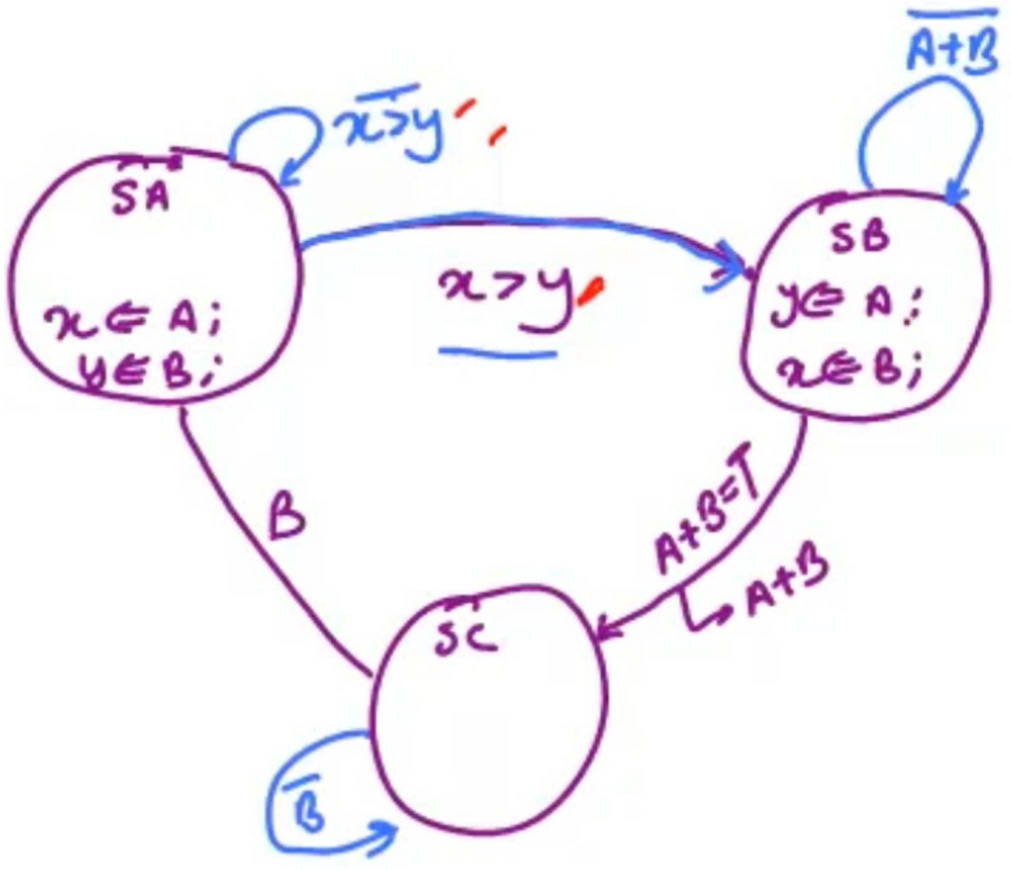
FSM diagrams



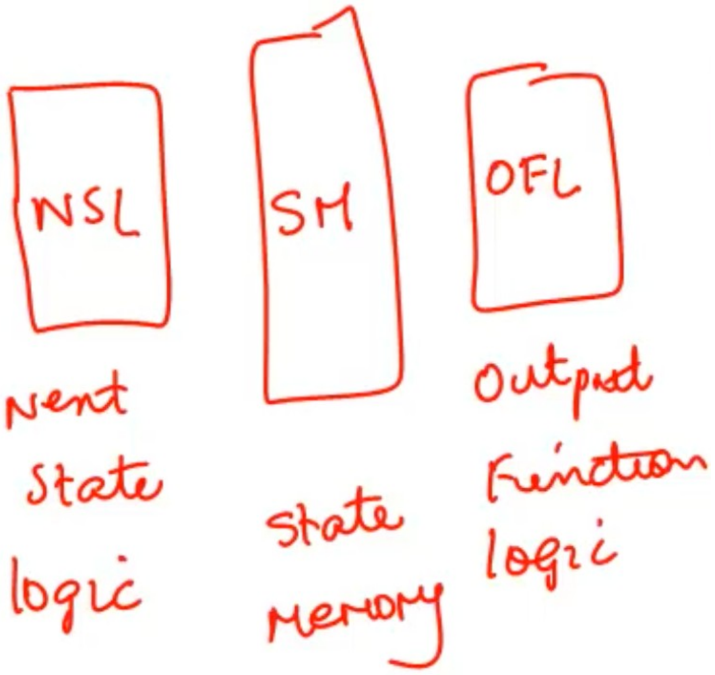
FSM diagrams (cont.)

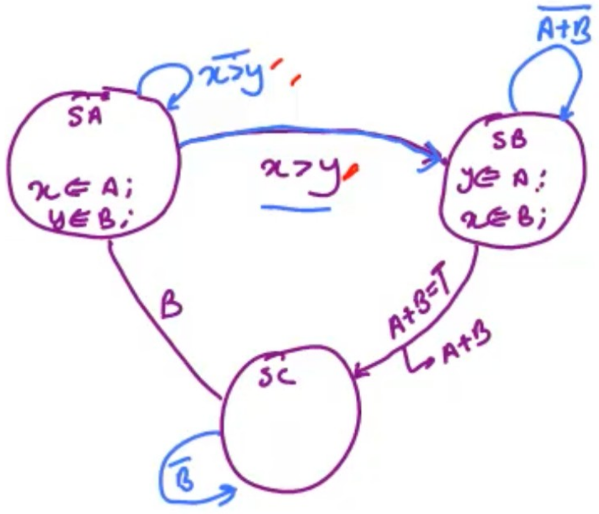


FSM diagrams fundamentals



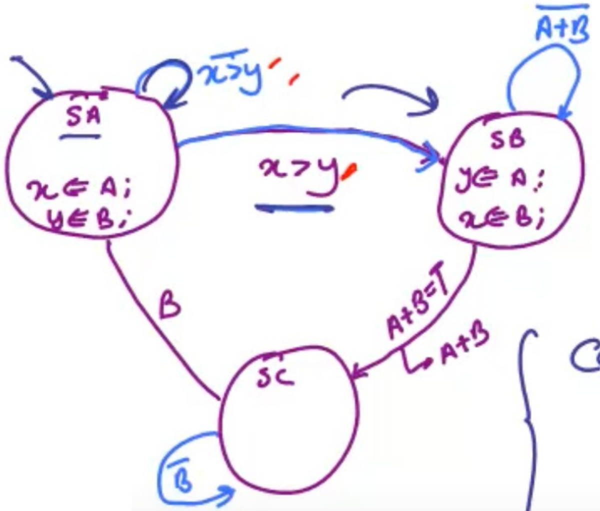
FSM diagrams fundamentals





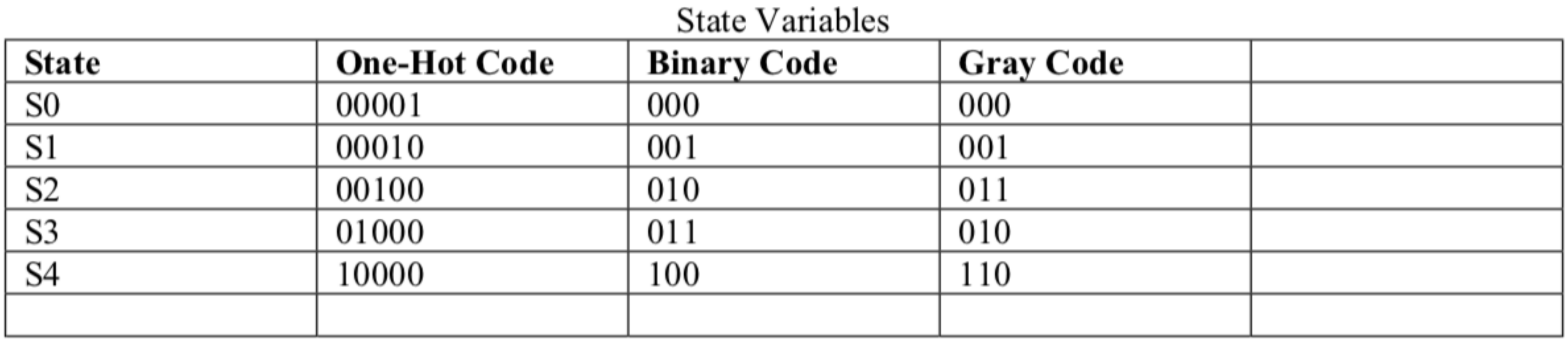
FSMs in VHDL

- NSL is modeled using if-else statements
- Each state is modeled using Case-statements
- State variables are defined using constants
- OFL is modeled using conditional assignments



```
{ If (x > y)
  State <= SB
  else
  state <= SA;

{ Case state is
    when SA =>
      x <= A;
      y <= B;
```

One-hot Encoding



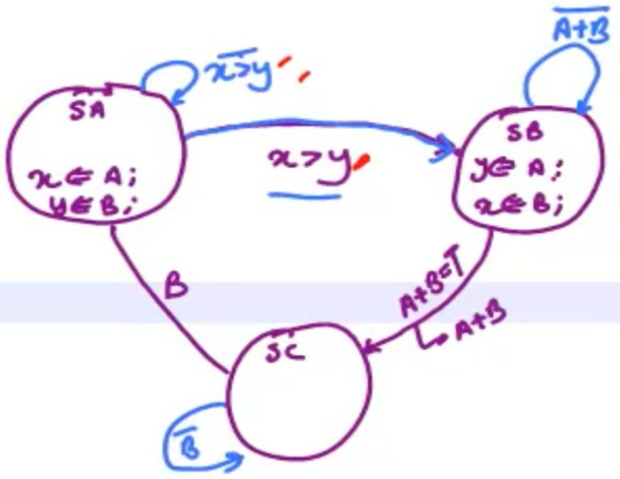
State Variables

State	One-Hot Code	Binary Code	Gray Code	
S0	00001	000	000	
S1	00010	001	001	
S2	00100	010	011	
S3	01000	011	010	
S4	10000	100	110	

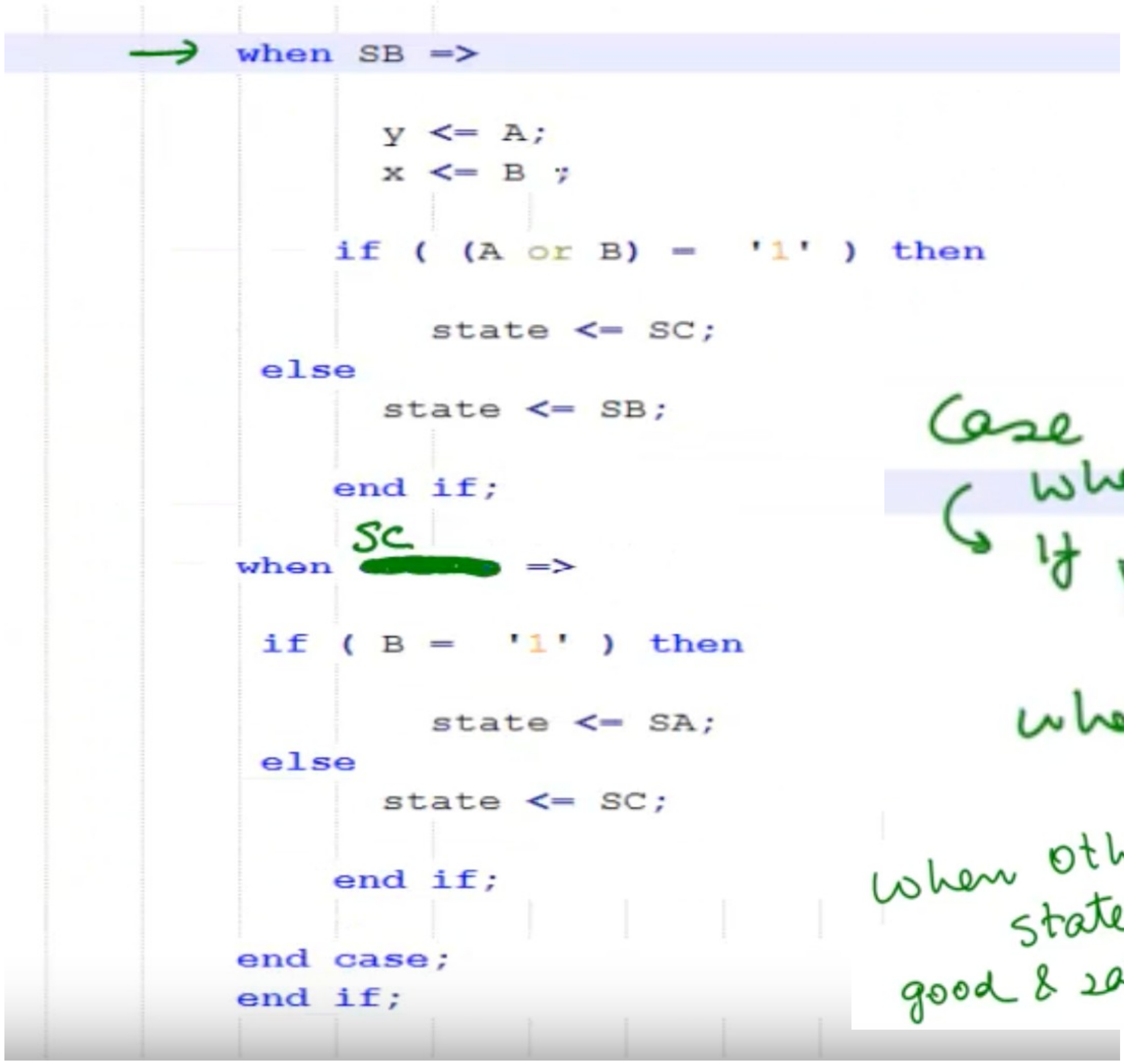
Table 1:An example of state Encoding for a 4 state Machine

FSMs in VHDL

```
architecture Behavioral of f is

   signal x: std_logic;
   signal y: std_logic;

   constant SA : std_logic_vector(2 downto 0):= "001";
   constant SB : std_logic_vector(2 downto 0):= "010";
   constant SC : std_logic_vector(2 downto 0):= "100";
   signal state: std_logic_vector(2 downto 0):= "001";
   begin
    process (clk )
    begin
          if rising_edge(clk) then

             case state is

              when SA =>

                 x <= A;
                  y <= B;

                 if ( x <= y) then   -- NSL

                     state <= SA;
              else
                  state <= SB;
                 end if; -- end of NSL
```



FSMs in VHDL



```
→ when SB =>

         y <= A;
         x <= B ;

      if ( (A or B) = '1' ) then

            state <= SC;
    else
         state <= SB;

      end if;
        SC
 when ~~[illegible]~~ =>

    if ( B = '1' ) then

            state <= SA;
    else
         state <= SC;

      end if;

 end case;
 end if;
```

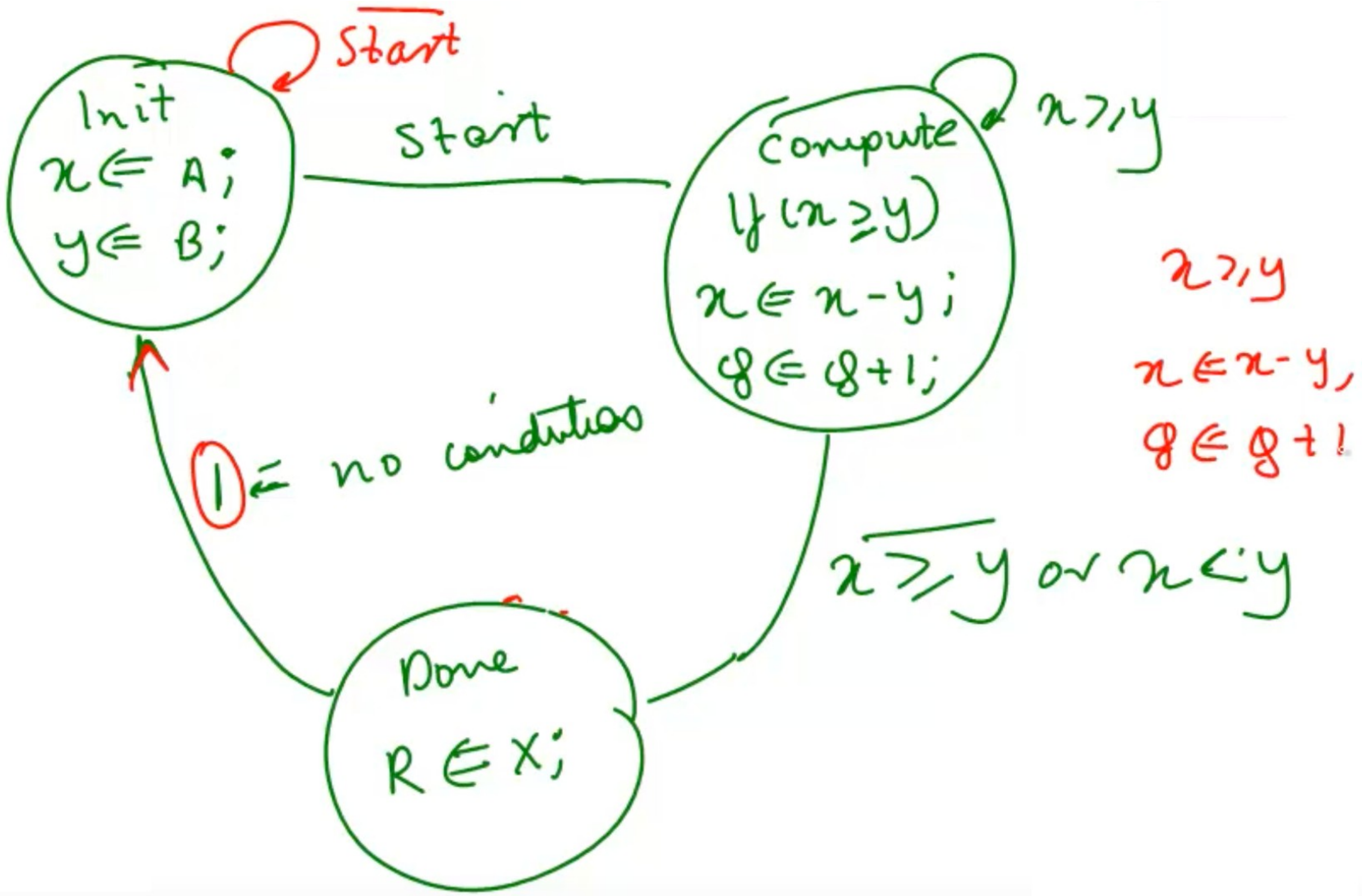
Case
when SA
if (← NSL
OFL
when SB

when others ⇒
state <= SA;
good & safe design

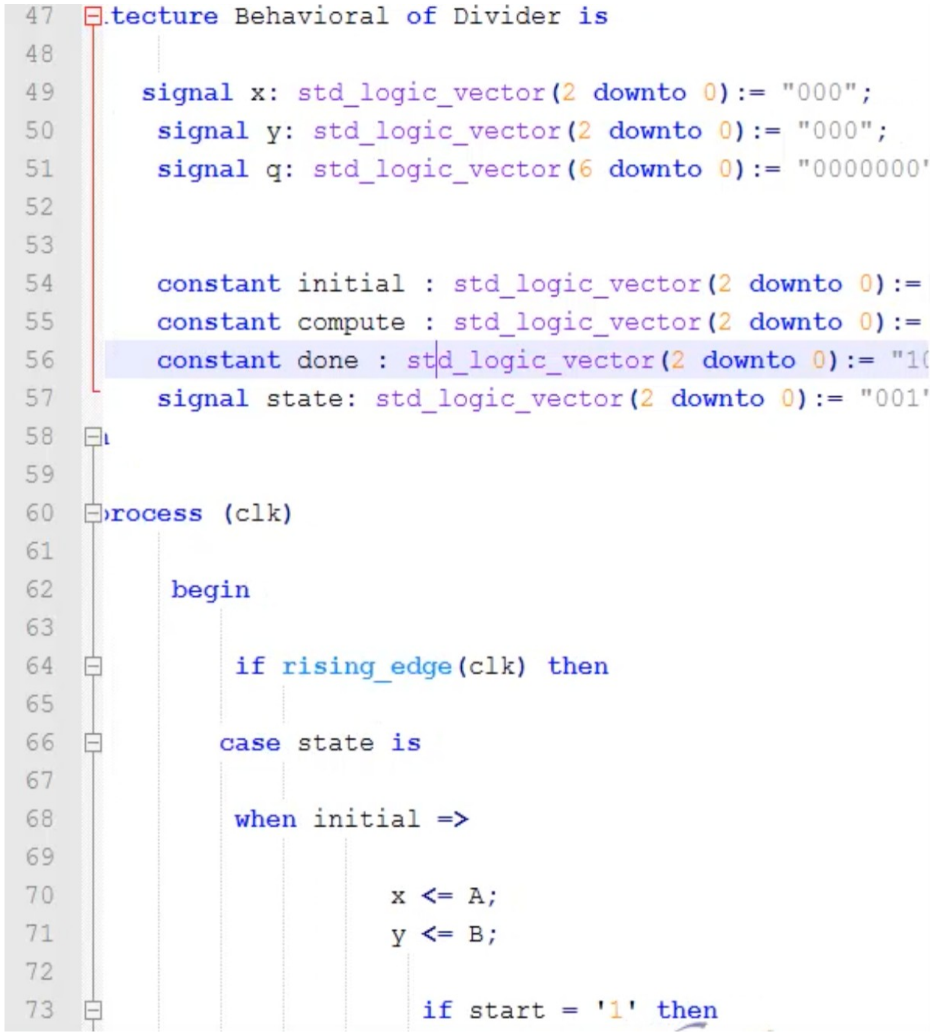
Divider example

$$\begin{tikzcd}
8-3=5 & Q=1 \\
5-3=2 \arrow[ddr, bend left] & Q=2 \\
2-3=-ve & Q=2 \\
 & R=2
\end{tikzcd}$$

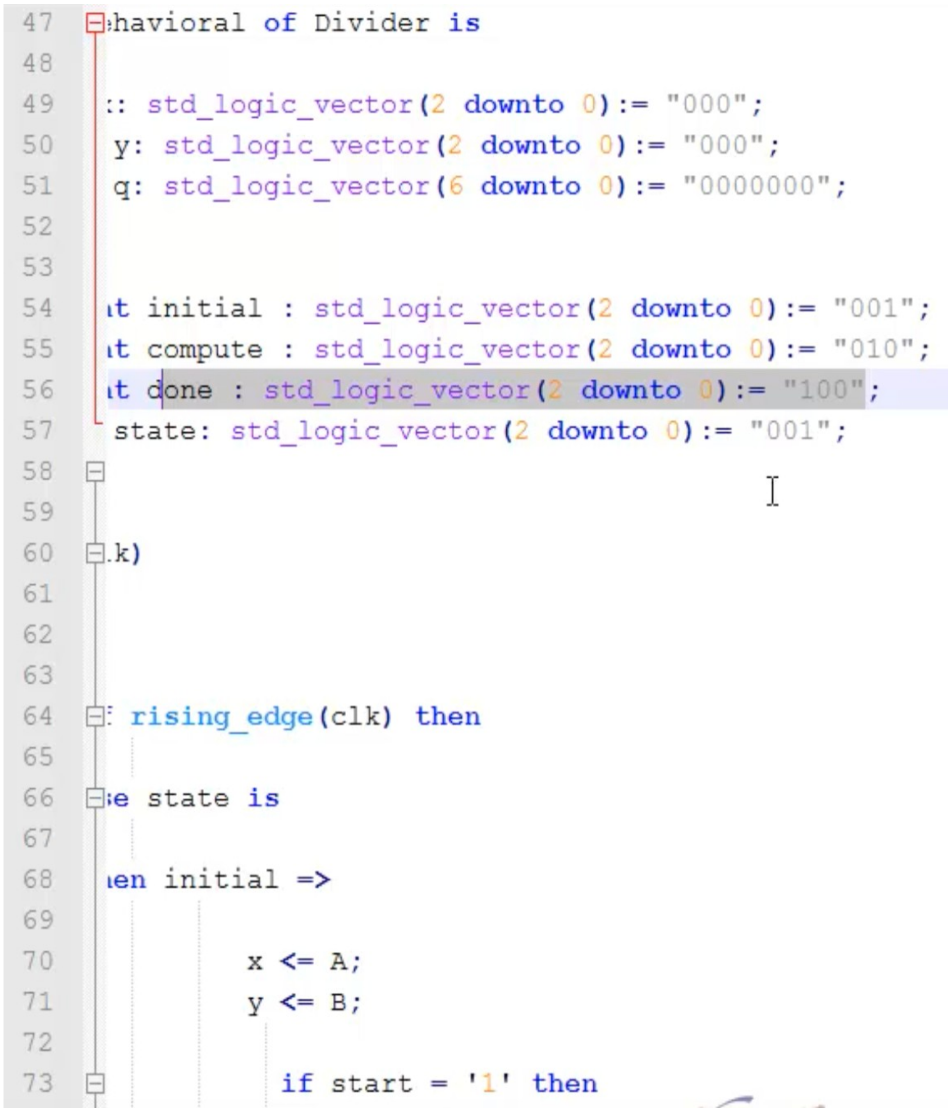
Divider example



Divider example

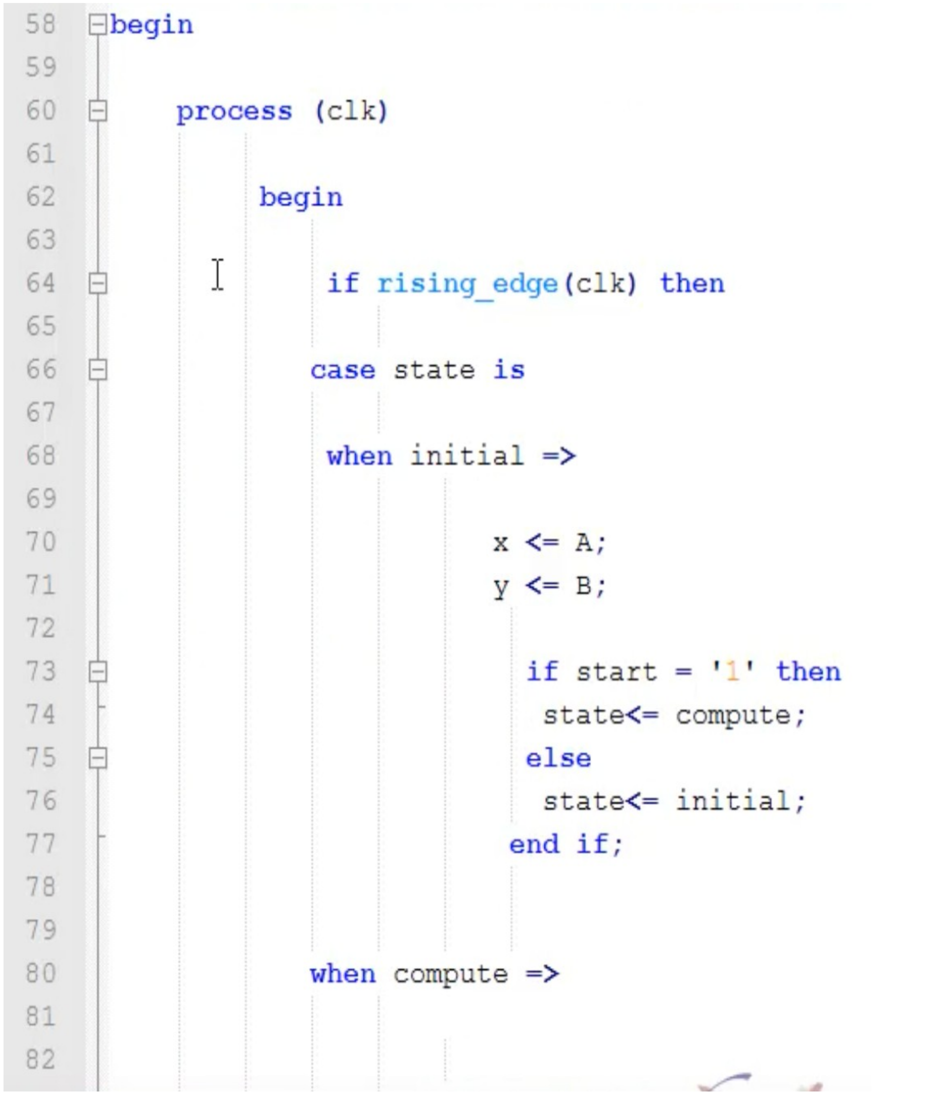


```
.tecture Behavioral of Divider is

   signal x: std_logic_vector(2 downto 0):= "000";
    signal y: std_logic_vector(2 downto 0):= "000";
    signal q: std_logic_vector(6 downto 0):= "0000000'


    constant initial : std_logic_vector(2 downto 0):=
    constant compute : std_logic_vector(2 downto 0):=
    constant done : std_logic_vector(2 downto 0):= "1(
    signal state: std_logic_vector(2 downto 0):= "001'
ı

>rocess (clk)

     begin

         if rising_edge(clk) then

        case state is

         when initial =>

                 x <= A;
                 y <= B;

                  if start = '1' then
```

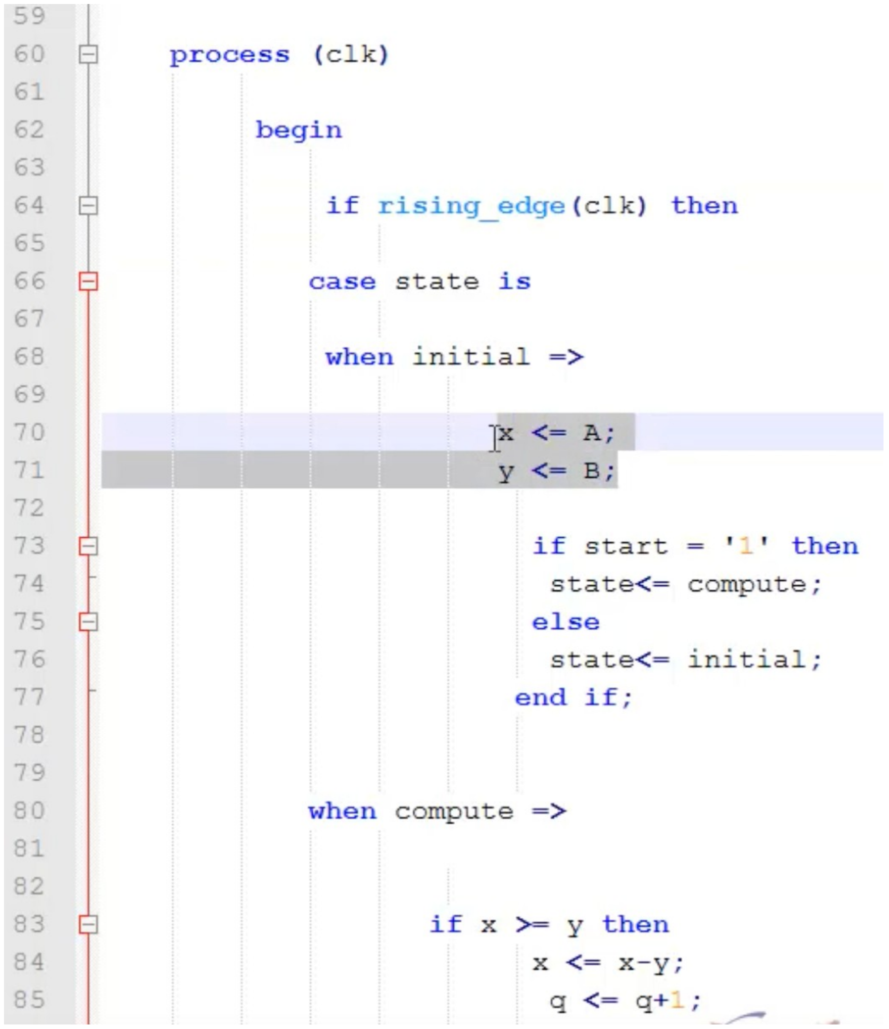


```
ehavioral of Divider is

:: std_logic_vector(2 downto 0):= "000";
 y: std_logic_vector(2 downto 0):= "000";
 q: std_logic_vector(6 downto 0):= "0000000";


ıt initial : std_logic_vector(2 downto 0):= "001";
ıt compute : std_logic_vector(2 downto 0):= "010";
ıt done : std_logic_vector(2 downto 0):= "100";
 state: std_logic_vector(2 downto 0):= "001";


.k)



: rising_edge(clk) then

:e state is

ıen initial =>

                x <= A;
                y <= B;

                  if start = '1' then
```

Divider example

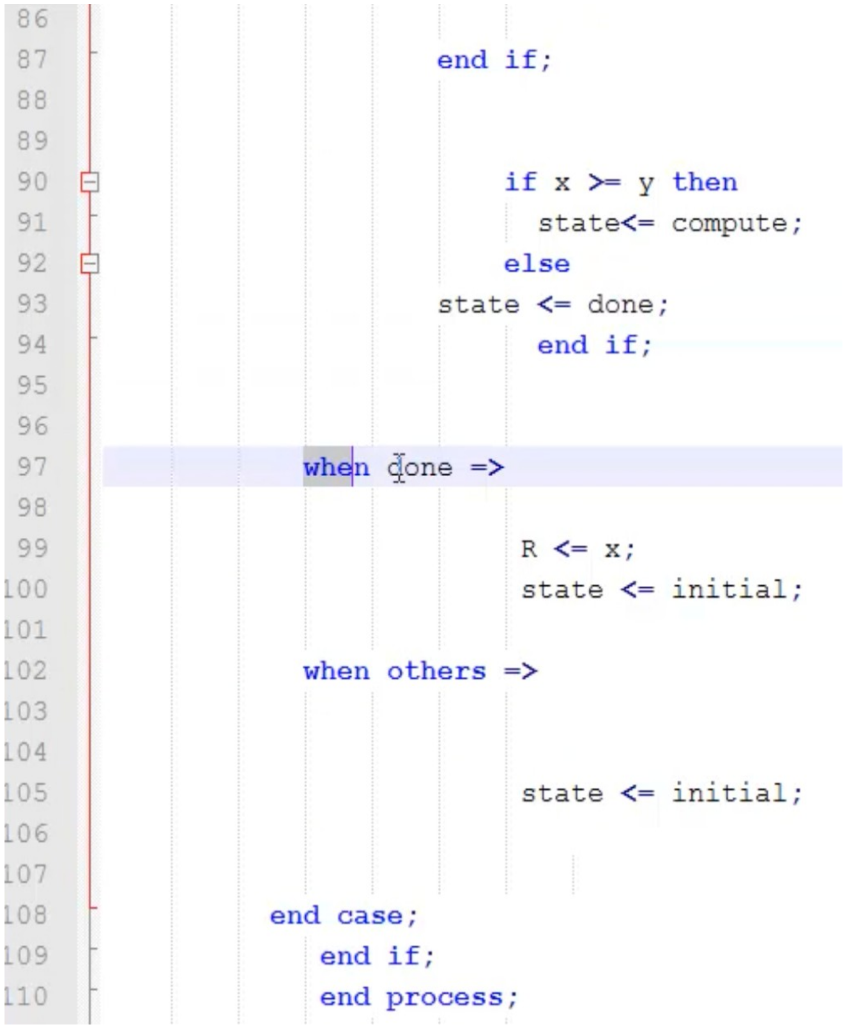


```
begin

    process (clk)

        begin

          if rising_edge(clk) then

         case state is

          when initial =>

                     x <= A;
                     y <= B;

                       if start = '1' then
                        state<= compute;
                       else
                        state<= initial;
                      end if;


         when compute =>

```

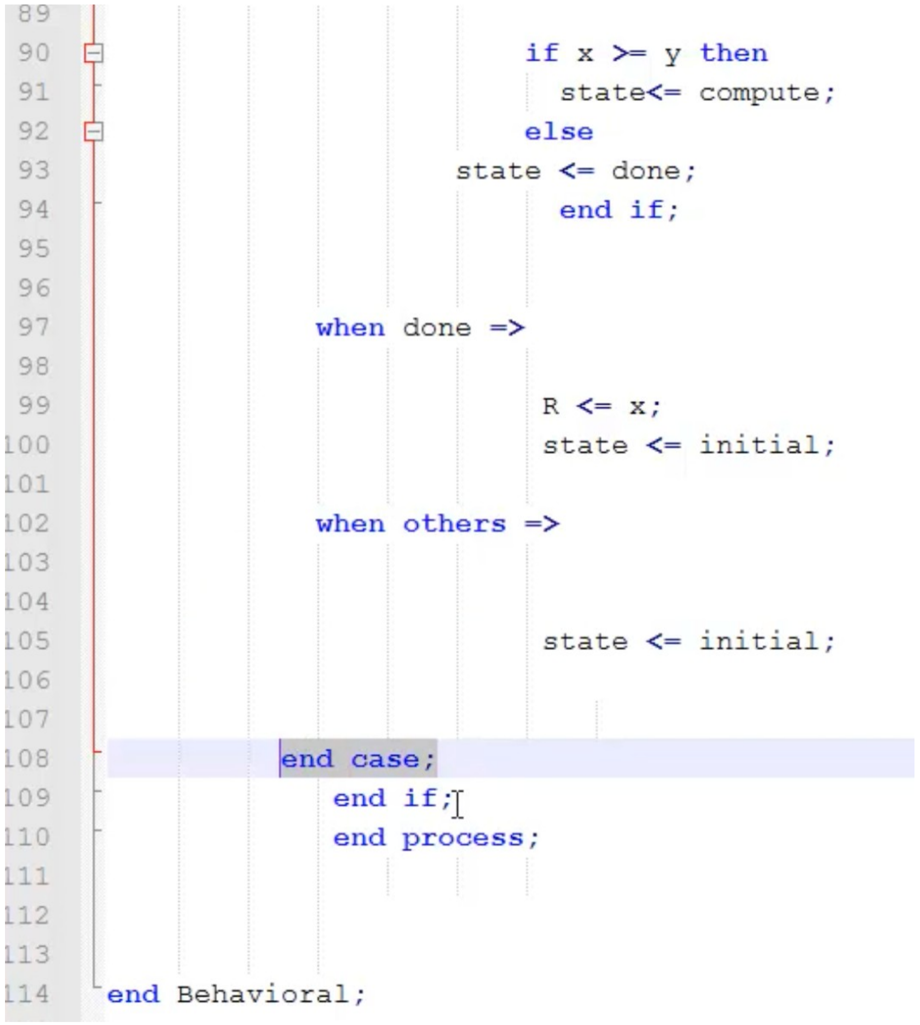


```

    process (clk)

        begin

          if rising_edge(clk) then

         case state is

          when initial =>

                     x <= A;
                     y <= B;

                       if start = '1' then
                        state<= compute;
                       else
                        state<= initial;
                      end if;


         when compute =>


                if x >= y then
                   x <= x-y;
                    q <= q+1;
```

Divider example



```
                    end if;


                        if x >= y then
                          state<= compute;
                        else
                    state <= done;
                          end if;


            when done =>

                         R <= x;
                         state <= initial;

            when others =>


                         state <= initial;


        end case;
           end if;
           end process;
```



```
                        if x >= y then
                          state<= compute;
                        else
                    state <= done;
                          end if;


            when done =>

                         R <= x;
                         state <= initial;

            when others =>


                         state <= initial;


        end case;
           end if;
           end process;



end Behavioral;
```