Lab #7

ECE 4304 Spring 2021

Professor Aly

California State Polytechnic University, Pomona

Ahiezer Lopez	Joe Si	Sander Zuckerman
ahiezerlopez@cpp.edu	joesi@cpp.edu	sazuckerman@cpp.edu
Bronco ID:012715521	Bronco ID: 012636091	Bronco ID: 012644671

4/07/2021

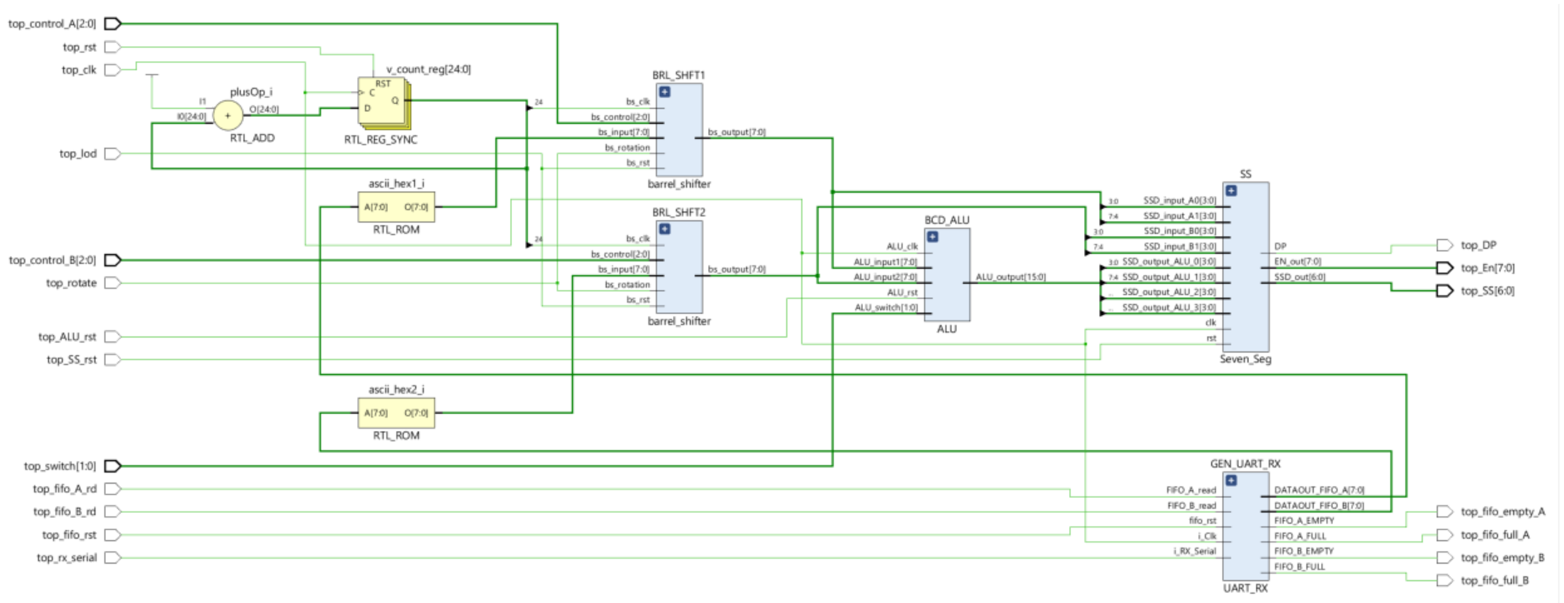
Objective:
Students will design a UART compatible machine utilizing FIFOs, ALUs, Barrel Shifters, and Seven Segments. The number of control bits will be 6 as it will be 3 per barrel shifter. We will have to read buttons each for the two FIFOs as well as a load button for the ALU. Along with the ALU, there are two switches that determine the math operation being executed. The outputs will have 4 LEDs representing whether the FIFOs are full or empty. Lastly, for the Seven Segments we will have 4 of them representing our ALU output and 4 representing our 8 bit inputs A and B.

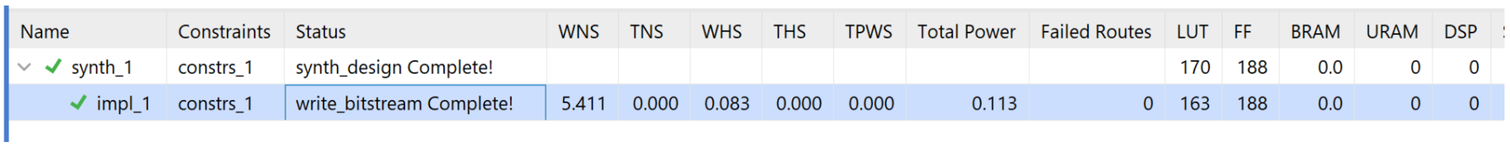
Materials:
- FPGA (Nexys A7-100T)
- Vivado Software
- Computer
- UART Pmod (optional)

Contributions:
For this lab, everyone had their own part in it. With all parts/ modules already created, the group itself focused on the connection of the UART module and the rest of the code. Having been exposed to how UART works in a previous lecture, this gave us a headstart in this project. Still, we had to research to smooth out any confusion we had regarding UART. We each worked together to connect all the modules accordingly. The most time consuming part of the project was debugging, but through everyone's individual effort in debugging we were able to get it working.

Design Process:
Using the flowchart from below, we implemented our idea on how to approach this lab. We were able to implement the codes from previous labs and the new code provided during class to create our lab. We had multiple issues when designing this lab, one issue was how the connection between the FPGA and the computer through UART was supposed to be done. We had to use the program "Tera Term" to easily send data directly through the USB. This brought on another issue regarding the ASCII data not sending in the correct format. Another issue was created for the transmission of data between the UART and the FIFOs as one FIFO may fill up however, the second FIFO would not. These issues were resolved after hours of debugging and the objective of this lab was completed.
Design:
Using some code from previous lab experiments such as the seven segment code, the Arithmetic Logic Unit, and the barrel shifter we were able to combine these components with what was shown in class. In class, we were provided with the FIFO and the UART_rx code, so all we had to do was wire up / connect these components together correctly. The main design was already created and we just had to correctly wire up these components with the right settings such as the baud rate and stop bit.

Circuit Diagram:

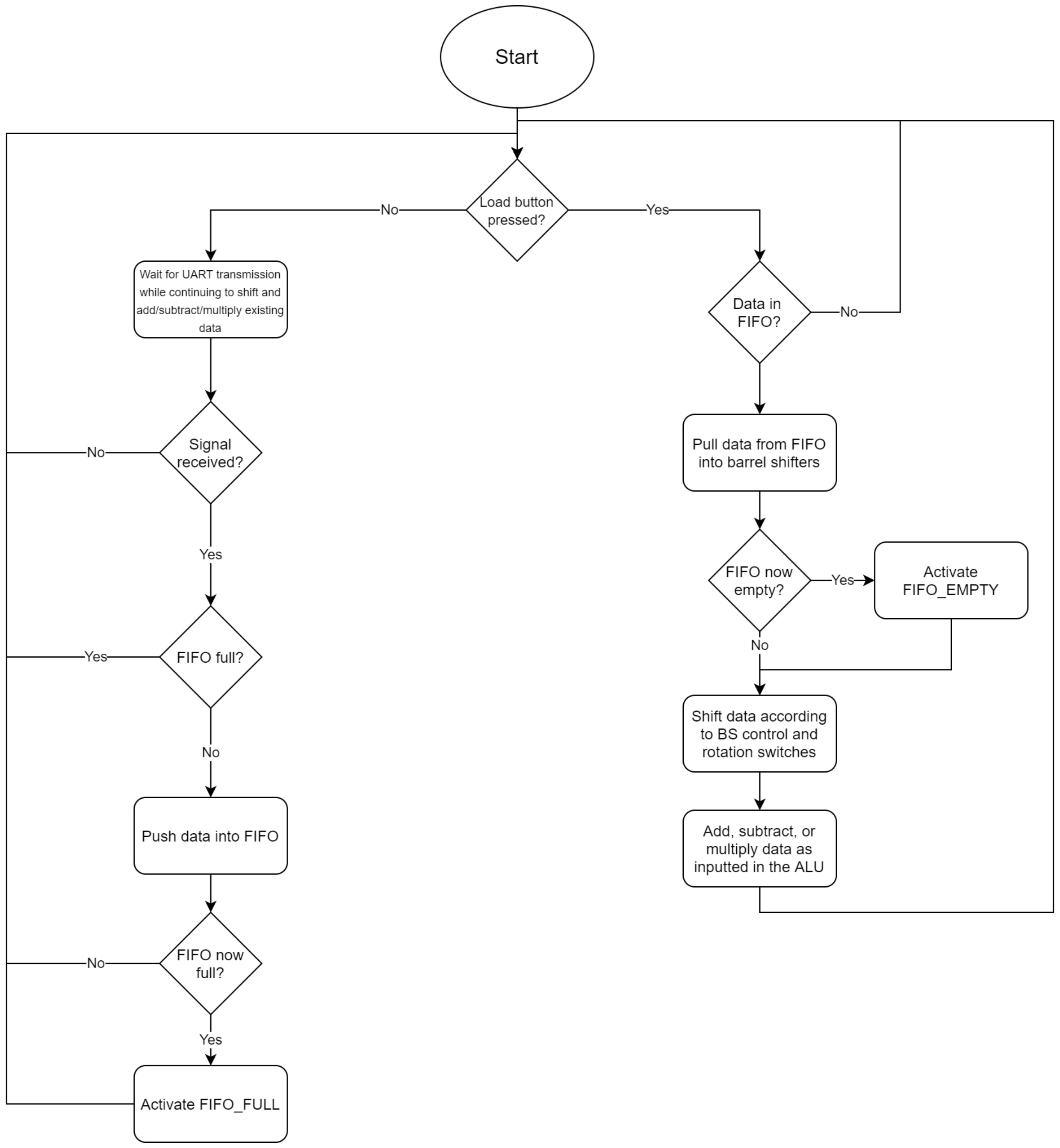




Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								170	188	0.0	0	0
✓ impl_1	constrs_1	write_bitstream Complete!	5.411	0.000	0.083	0.000	0.000	0.113	0	163	188	0.0	0	0

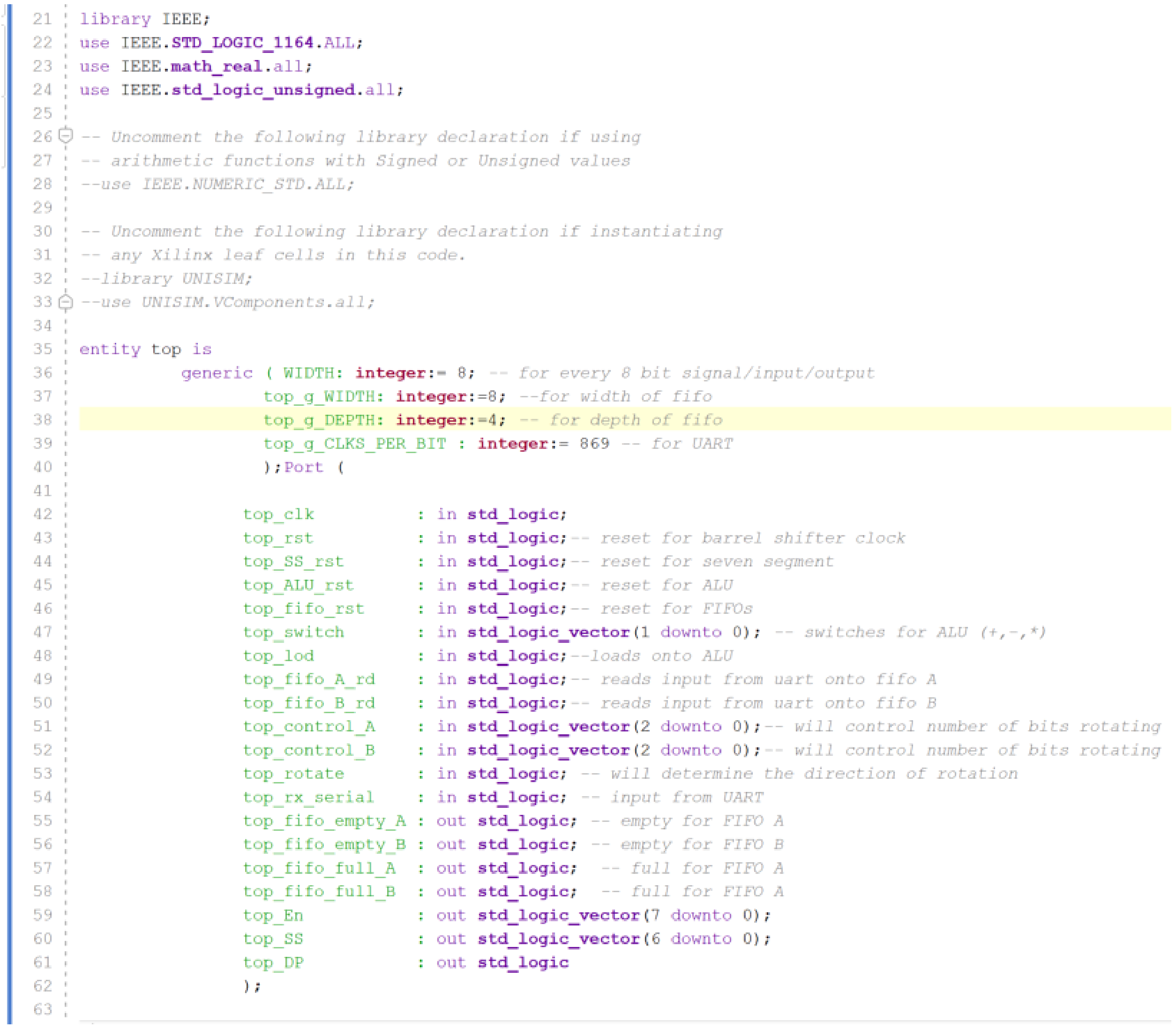
Implementation:

Logic Flow Diagram

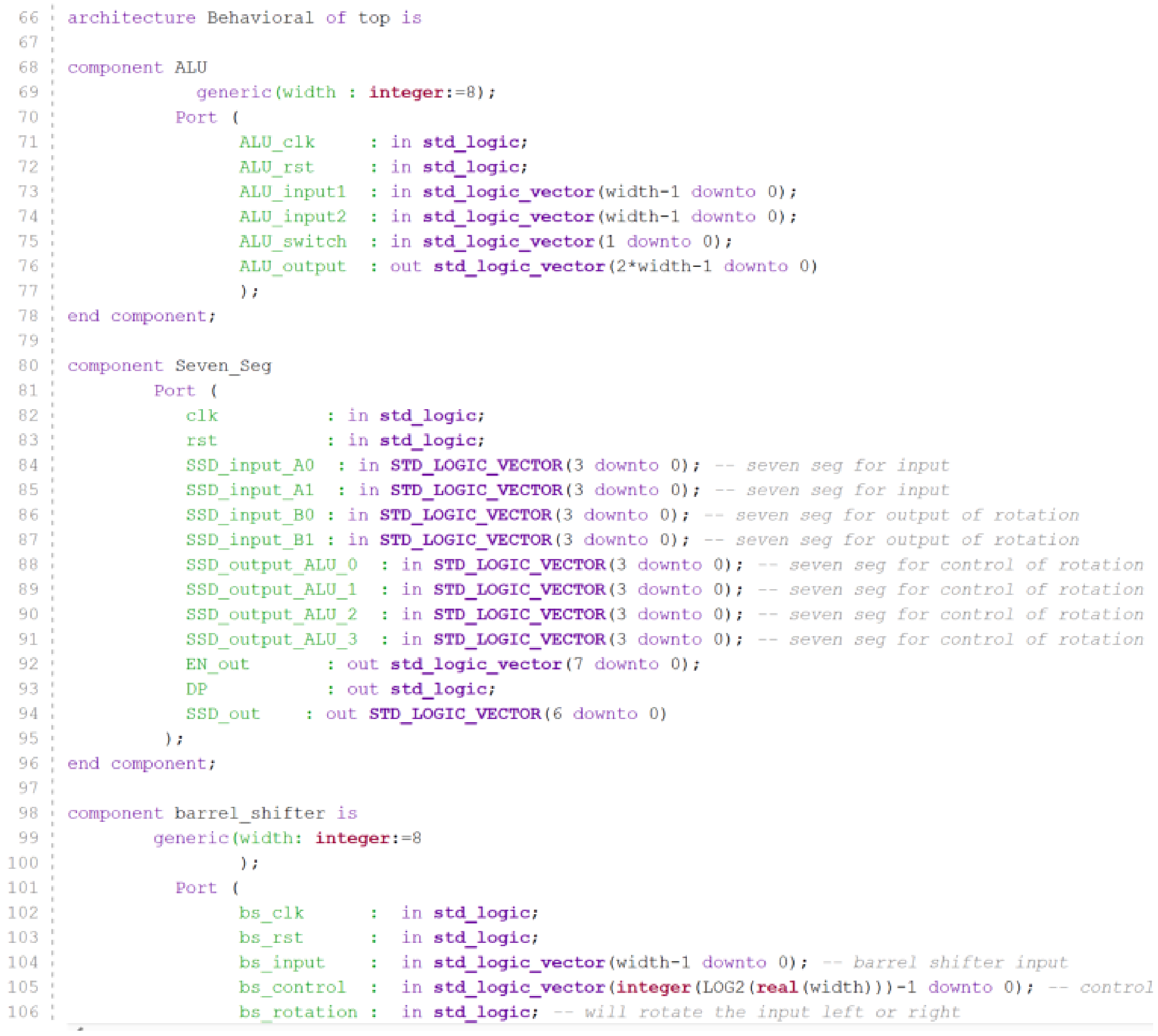

Start
Load button pressed?
No
Yes
Wait for UART transmission while continuing to shift and add/subtract/multiply existing data
Data in FIFO?
No
Signal received?
No
Yes
Pull data from FIFO into barrel shifters
FIFO now empty?
Yes
Activate FIFO_EMPTY
No
FIFO full?
Yes
No
Shift data according to BS control and rotation switches
Push data into FIFO
Add, subtract, or multiply data as inputted in the ALU
FIFO now full?
No
Yes
Activate FIFO_FULL

Design:

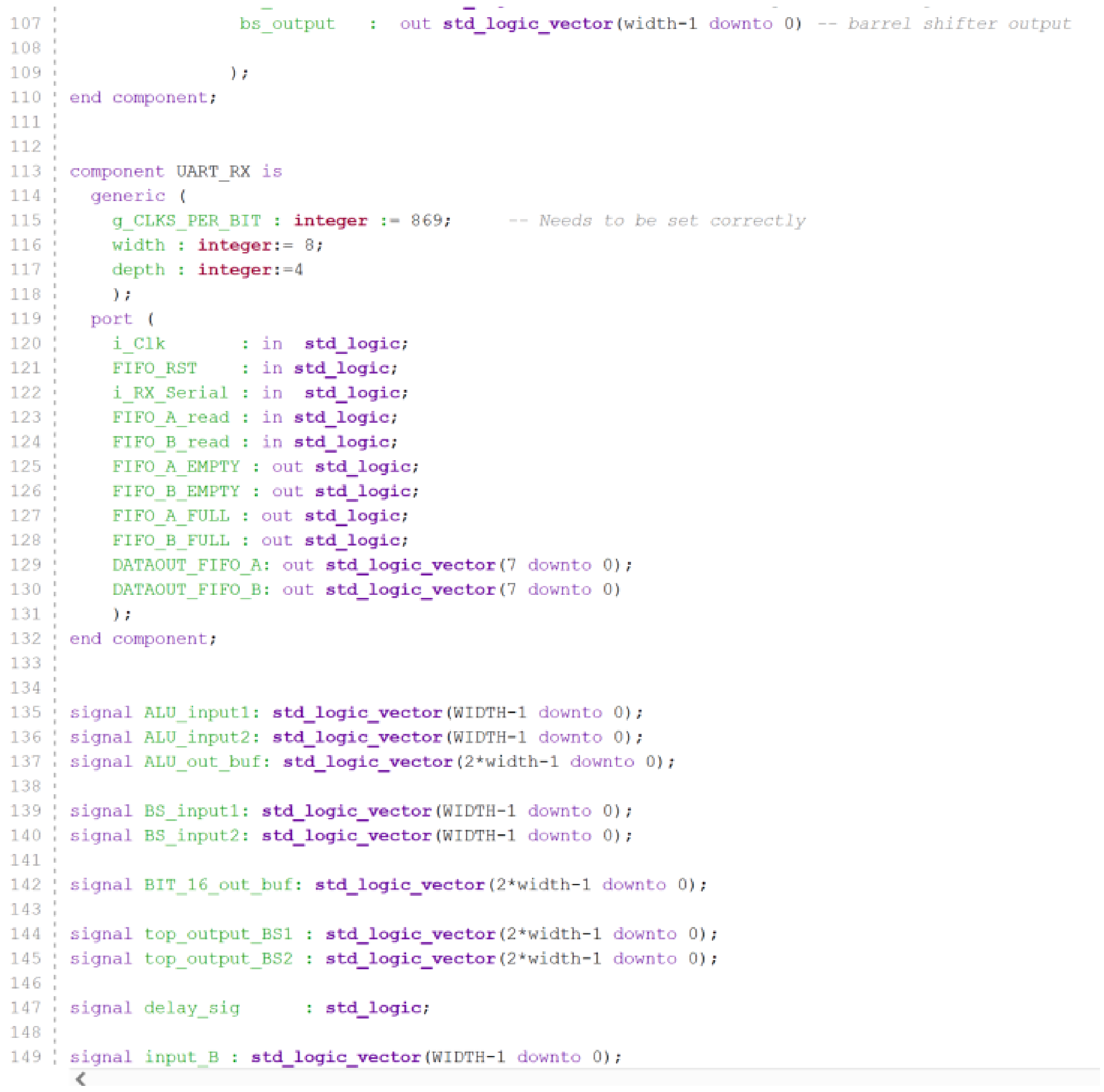
Top Module



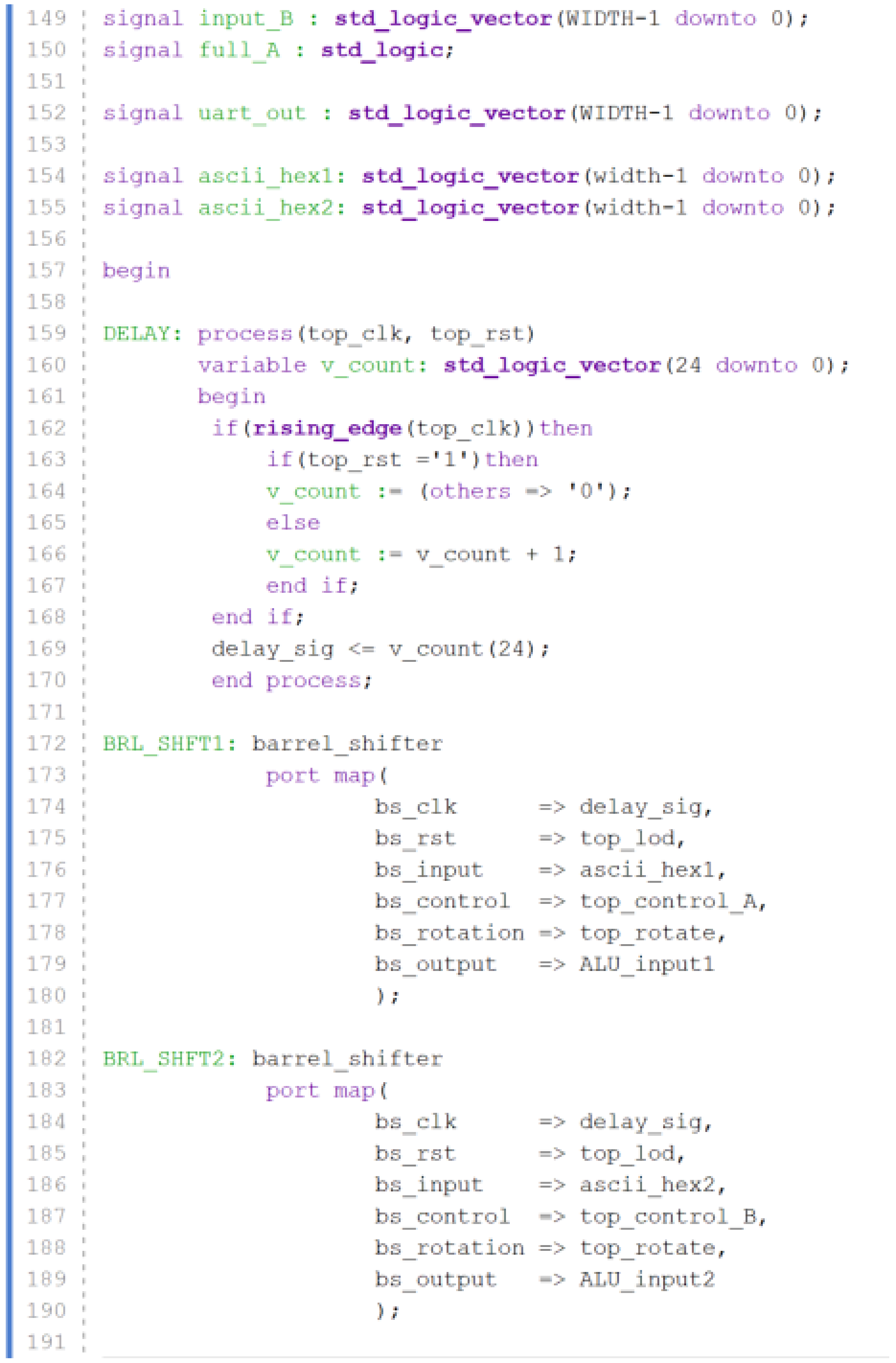
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.math_real.all;
use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top is
          generic ( WIDTH: integer:= 8; -- for every 8 bit signal/input/output
                  top_g_WIDTH: integer:=8; --for width of fifo
                  top_g_DEPTH: integer:=4; -- for depth of fifo
                  top_g_CLKS_PER_BIT : integer:= 869 -- for UART
                  );Port (

                top_clk          : in std_logic;
                top_rst          : in std_logic;-- reset for barrel shifter clock
                top_SS_rst       : in std_logic;-- reset for seven segment
                top_ALU_rst      : in std_logic;-- reset for ALU
                top_fifo_rst     : in std_logic;-- reset for FIFOs
                top_switch       : in std_logic_vector(1 downto 0); -- switches for ALU (+,-,*)
                top_lod          : in std_logic;--loads onto ALU
                top_fifo_A_rd    : in std_logic;-- reads input from uart onto fifo A
                top_fifo_B_rd    : in std_logic;-- reads input from uart onto fifo B
                top_control_A    : in std_logic_vector(2 downto 0);-- will control number of bits rotating
                top_control_B    : in std_logic_vector(2 downto 0);-- will control number of bits rotating
                top_rotate       : in std_logic; -- will determine the direction of rotation
                top_rx_serial    : in std_logic; -- input from UART
                top_fifo_empty_A : out std_logic; -- empty for FIFO A
                top_fifo_empty_B : out std_logic; -- empty for FIFO B
                top_fifo_full_A  : out std_logic;  -- full for FIFO A
                top_fifo_full_B  : out std_logic;  -- full for FIFO A
                top_En           : out std_logic_vector(7 downto 0);
                top_SS           : out std_logic_vector(6 downto 0);
                top_DP           : out std_logic
                );

```



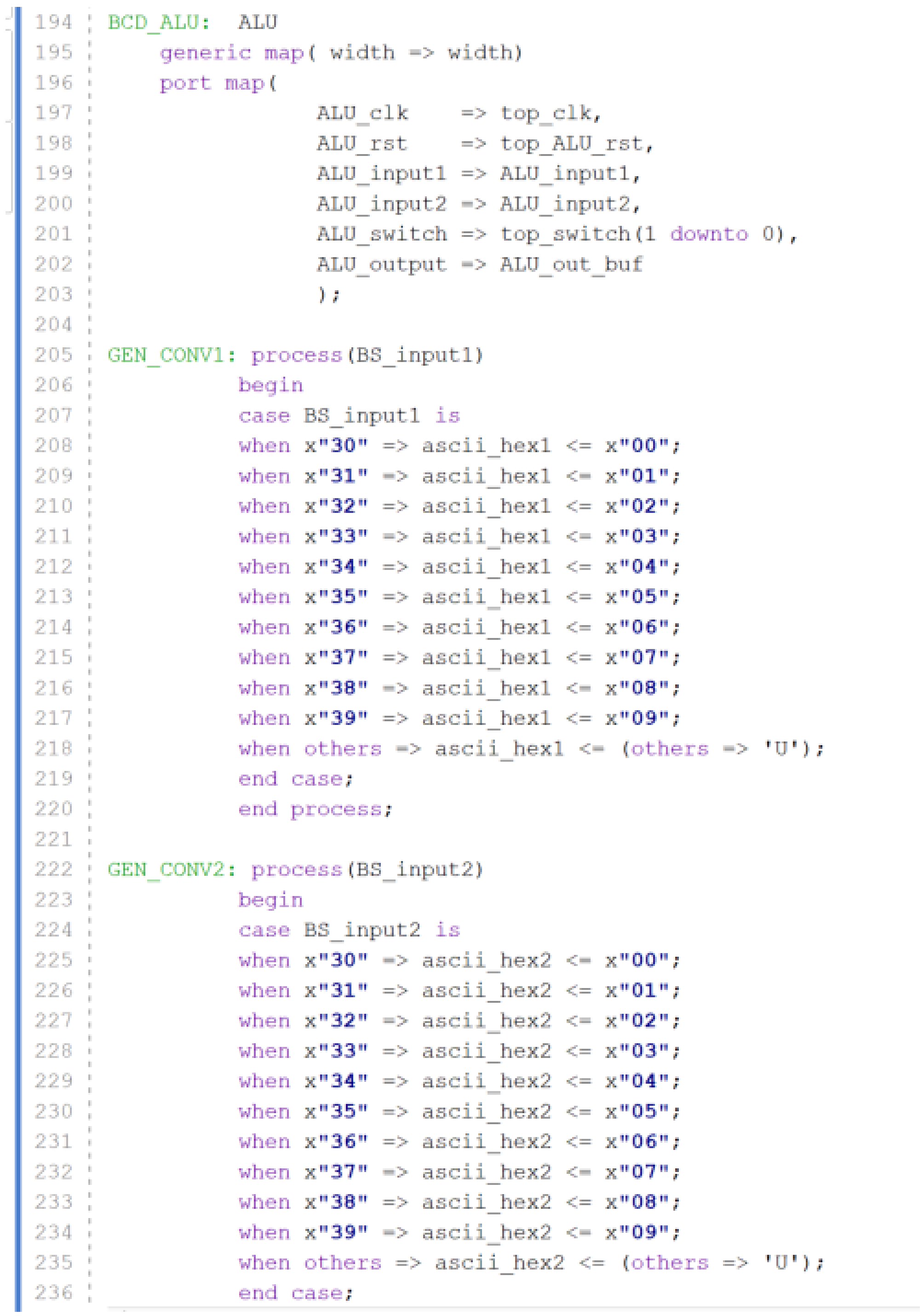
```
architecture Behavioral of top is

component ALU
            generic(width : integer:=8);
          Port (
                ALU_clk     : in std_logic;
                ALU_rst     : in std_logic;
                ALU_input1  : in std_logic_vector(width-1 downto 0);
                ALU_input2  : in std_logic_vector(width-1 downto 0);
                ALU_switch  : in std_logic_vector(1 downto 0);
                ALU_output  : out std_logic_vector(2*width-1 downto 0)
                );
end component;

component Seven_Seg
        Port (
           clk          : in std_logic;
           rst          : in std_logic;
           SSD_input_A0  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
           SSD_input_A1  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
           SSD_input_B0 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
           SSD_input_B1 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
           SSD_output_ALU_0  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           SSD_output_ALU_1  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           SSD_output_ALU_2  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           SSD_output_ALU_3  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           EN_out       : out std_logic_vector(7 downto 0);
           DP           : out std_logic;
           SSD_out    : out STD_LOGIC_VECTOR(6 downto 0)
         );
end component;

component barrel_shifter is
        generic(width: integer:=8
                );
          Port (
                bs_clk      :  in std_logic;
                bs_rst      :  in std_logic;
                bs_input    :  in std_logic_vector(width-1 downto 0); -- barrel shifter input
                bs_control  :  in std_logic_vector(integer(LOG2(real(width)))-1 downto 0); -- control
                bs_rotation :  in std_logic; -- will rotate the input left or right
```



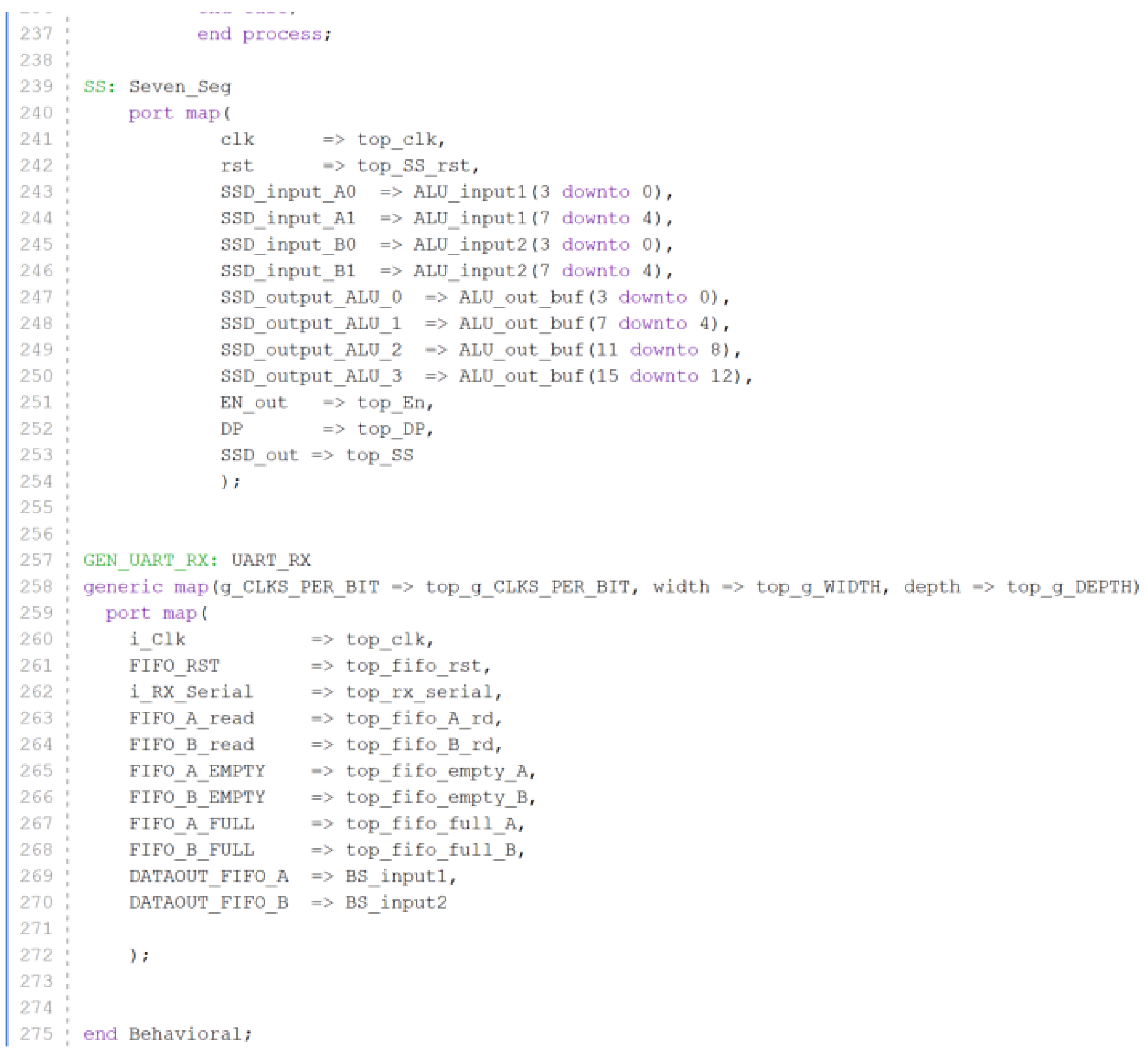
```vhdl
                bs_output   :  out std_logic_vector(width-1 downto 0) -- barrel shifter output

               );
end component;


component UART_RX is
  generic (
    g_CLKS_PER_BIT : integer := 869;     -- Needs to be set correctly
    width : integer:= 8;
    depth : integer:=4
    );
  port (
    i_Clk       : in  std_logic;
    FIFO_RST    : in std_logic;
    i_RX_Serial : in  std_logic;
    FIFO_A_read : in std_logic;
    FIFO_B_read : in std_logic;
    FIFO_A_EMPTY : out std_logic;
    FIFO_B_EMPTY : out std_logic;
    FIFO_A_FULL : out std_logic;
    FIFO_B_FULL : out std_logic;
    DATAOUT_FIFO_A: out std_logic_vector(7 downto 0);
    DATAOUT_FIFO_B: out std_logic_vector(7 downto 0)
    );
end component;


signal ALU_input1: std_logic_vector(WIDTH-1 downto 0);
signal ALU_input2: std_logic_vector(WIDTH-1 downto 0);
signal ALU_out_buf: std_logic_vector(2*width-1 downto 0);

signal BS_input1: std_logic_vector(WIDTH-1 downto 0);
signal BS_input2: std_logic_vector(WIDTH-1 downto 0);

signal BIT_16_out_buf: std_logic_vector(2*width-1 downto 0);

signal top_output_BS1 : std_logic_vector(2*width-1 downto 0);
signal top_output_BS2 : std_logic_vector(2*width-1 downto 0);

signal delay_sig      : std_logic;

signal input_B : std_logic_vector(WIDTH-1 downto 0);
```



```vhdl
signal input_B : std_logic_vector(WIDTH-1 downto 0);
signal full_A : std_logic;

signal uart_out : std_logic_vector(WIDTH-1 downto 0);

signal ascii_hex1: std_logic_vector(width-1 downto 0);
signal ascii_hex2: std_logic_vector(width-1 downto 0);

begin

DELAY: process(top_clk, top_rst)
       variable v_count: std_logic_vector(24 downto 0);
       begin
        if(rising_edge(top_clk))then
            if(top_rst ='1')then
            v_count := (others => '0');
            else
            v_count := v_count + 1;
            end if;
        end if;
        delay_sig <= v_count(24);
        end process;

BRL_SHFT1: barrel_shifter
            port map(
                    bs_clk      => delay_sig,
                    bs_rst      => top_lod,
                    bs_input    => ascii_hex1,
                    bs_control  => top_control_A,
                    bs_rotation => top_rotate,
                    bs_output   => ALU_input1
                    );

BRL_SHFT2: barrel_shifter
            port map(
                    bs_clk      => delay_sig,
                    bs_rst      => top_lod,
                    bs_input    => ascii_hex2,
                    bs_control  => top_control_B,
                    bs_rotation => top_rotate,
                    bs_output   => ALU_input2
                    );
```

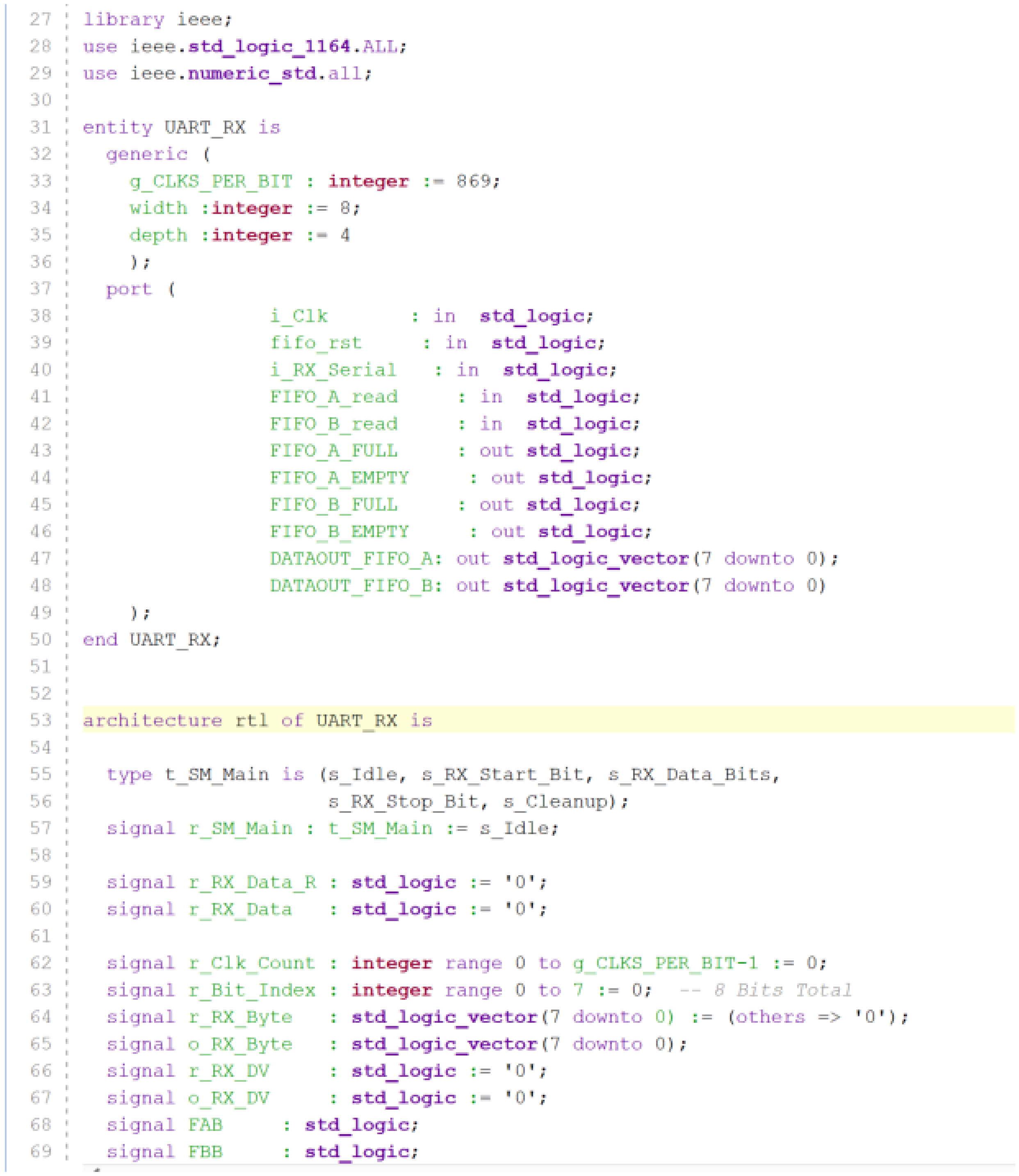


```
BCD_ALU:  ALU
    generic map( width => width)
    port map(
                ALU_clk    => top_clk,
                ALU_rst    => top_ALU_rst,
                ALU_input1 => ALU_input1,
                ALU_input2 => ALU_input2,
                ALU_switch => top_switch(1 downto 0),
                ALU_output => ALU_out_buf
                );

GEN_CONV1: process(BS_input1)
          begin
          case BS_input1 is
          when x"30" => ascii_hex1 <= x"00";
          when x"31" => ascii_hex1 <= x"01";
          when x"32" => ascii_hex1 <= x"02";
          when x"33" => ascii_hex1 <= x"03";
          when x"34" => ascii_hex1 <= x"04";
          when x"35" => ascii_hex1 <= x"05";
          when x"36" => ascii_hex1 <= x"06";
          when x"37" => ascii_hex1 <= x"07";
          when x"38" => ascii_hex1 <= x"08";
          when x"39" => ascii_hex1 <= x"09";
          when others => ascii_hex1 <= (others => 'U');
          end case;
          end process;

GEN_CONV2: process(BS_input2)
          begin
          case BS_input2 is
          when x"30" => ascii_hex2 <= x"00";
          when x"31" => ascii_hex2 <= x"01";
          when x"32" => ascii_hex2 <= x"02";
          when x"33" => ascii_hex2 <= x"03";
          when x"34" => ascii_hex2 <= x"04";
          when x"35" => ascii_hex2 <= x"05";
          when x"36" => ascii_hex2 <= x"06";
          when x"37" => ascii_hex2 <= x"07";
          when x"38" => ascii_hex2 <= x"08";
          when x"39" => ascii_hex2 <= x"09";
          when others => ascii_hex2 <= (others => 'U');
          end case;
```

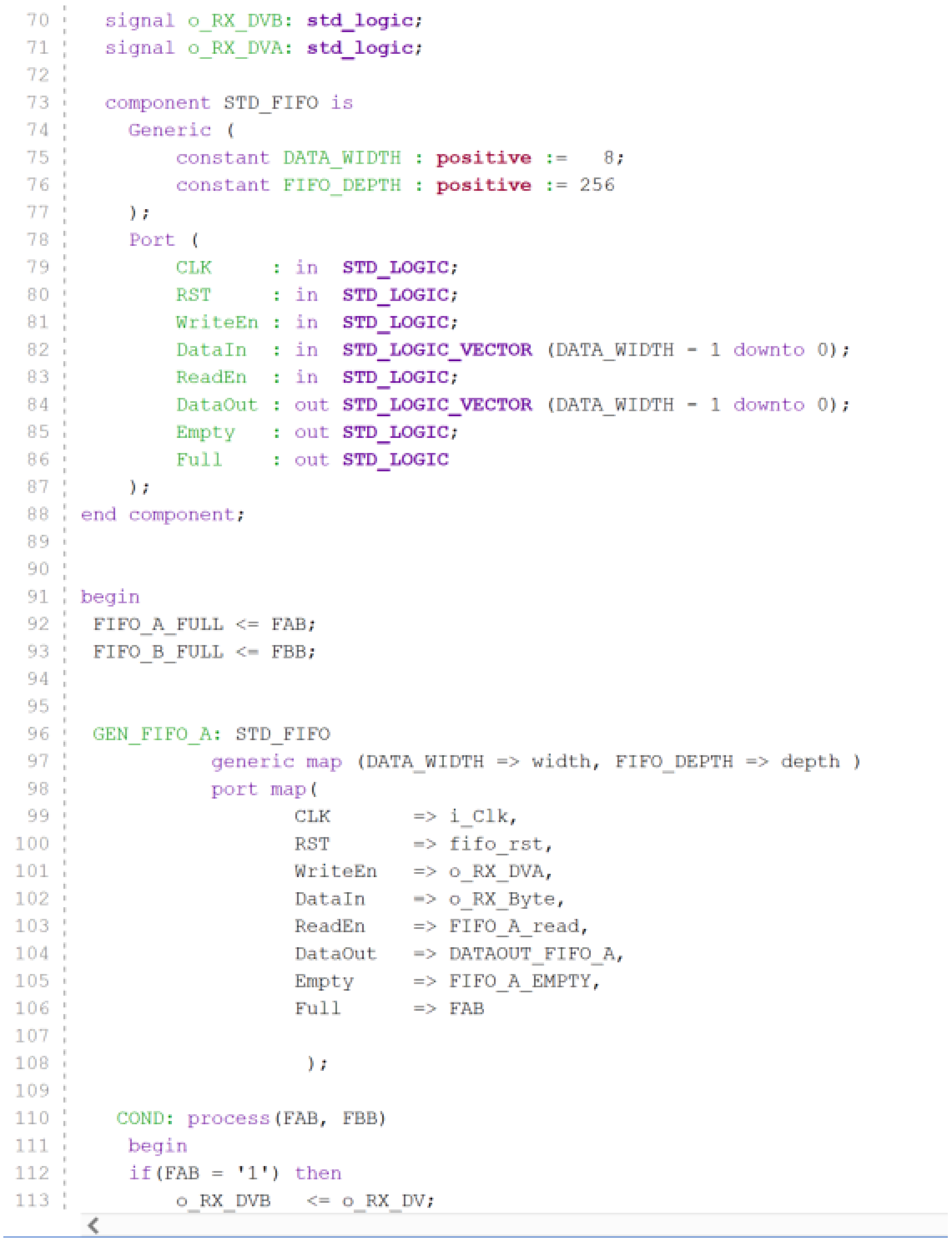


```vhdl
          end process;

SS: Seven_Seg
    port map(
            clk      => top_clk,
            rst      => top_SS_rst,
            SSD_input_A0  => ALU_input1(3 downto 0),
            SSD_input_A1  => ALU_input1(7 downto 4),
            SSD_input_B0  => ALU_input2(3 downto 0),
            SSD_input_B1  => ALU_input2(7 downto 4),
            SSD_output_ALU_0  => ALU_out_buf(3 downto 0),
            SSD_output_ALU_1  => ALU_out_buf(7 downto 4),
            SSD_output_ALU_2  => ALU_out_buf(11 downto 8),
            SSD_output_ALU_3  => ALU_out_buf(15 downto 12),
            EN_out   => top_En,
            DP       => top_DP,
            SSD_out => top_SS
            );


GEN_UART_RX: UART_RX
generic map(g_CLKS_PER_BIT => top_g_CLKS_PER_BIT, width => top_g_WIDTH, depth => top_g_DEPTH)
  port map(
    i_Clk           => top_clk,
    FIFO_RST        => top_fifo_rst,
    i_RX_Serial     => top_rx_serial,
    FIFO_A_read     => top_fifo_A_rd,
    FIFO_B_read     => top_fifo_B_rd,
    FIFO_A_EMPTY    => top_fifo_empty_A,
    FIFO_B_EMPTY    => top_fifo_empty_B,
    FIFO_A_FULL     => top_fifo_full_A,
    FIFO_B_FULL     => top_fifo_full_B,
    DATAOUT_FIFO_A  => BS_input1,
    DATAOUT_FIFO_B  => BS_input2

    );


end Behavioral;
```

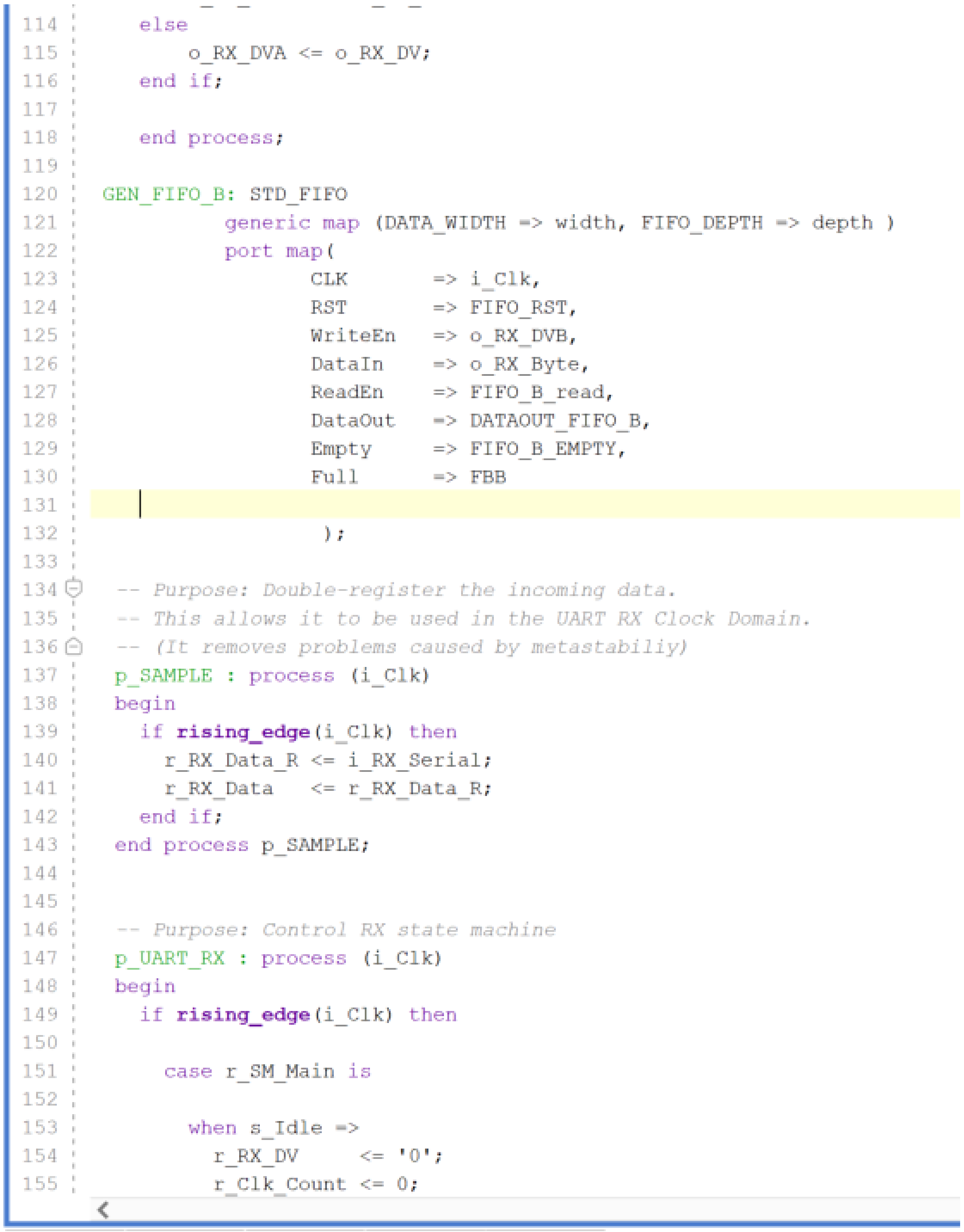
UART Rx Module:



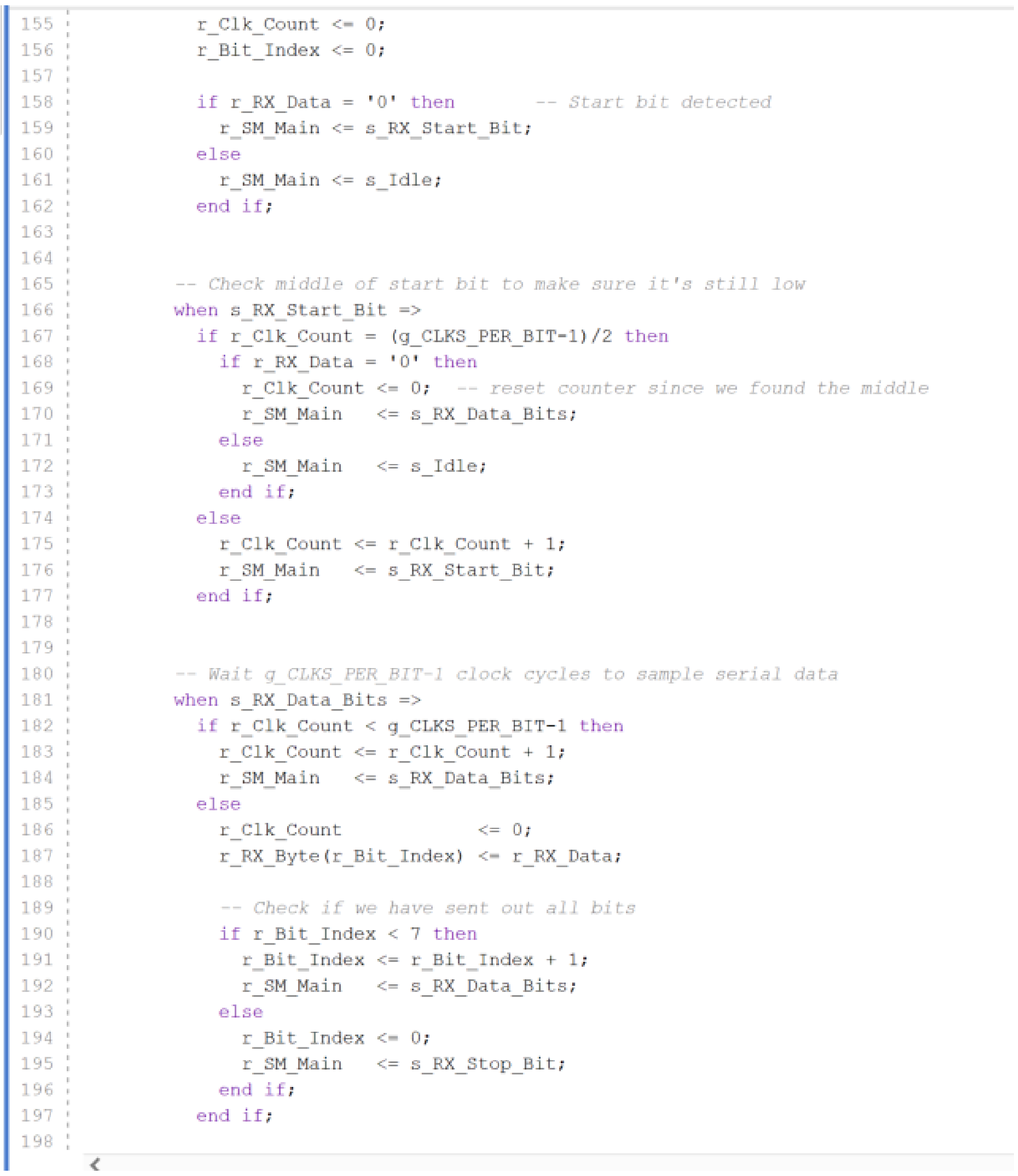
```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity UART_RX is
  generic (
    g_CLKS_PER_BIT : integer := 869;
    width :integer := 8;
    depth :integer := 4
    );
  port (
                i_Clk       : in  std_logic;
                fifo_rst     : in  std_logic;
                i_RX_Serial   : in  std_logic;
                FIFO_A_read     : in  std_logic;
                FIFO_B_read     : in  std_logic;
                FIFO_A_FULL     : out std_logic;
                FIFO_A_EMPTY     : out std_logic;
                FIFO_B_FULL     : out std_logic;
                FIFO_B_EMPTY     : out std_logic;
                DATAOUT_FIFO_A: out std_logic_vector(7 downto 0);
                DATAOUT_FIFO_B: out std_logic_vector(7 downto 0)
    );
end UART_RX;


architecture rtl of UART_RX is

  type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
                     s_RX_Stop_Bit, s_Cleanup);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_RX_Data_R : std_logic := '0';
  signal r_RX_Data   : std_logic := '0';

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
  signal o_RX_Byte   : std_logic_vector(7 downto 0);
  signal r_RX_DV     : std_logic := '0';
  signal o_RX_DV     : std_logic := '0';
  signal FAB     : std_logic;
  signal FBB     : std_logic;
```



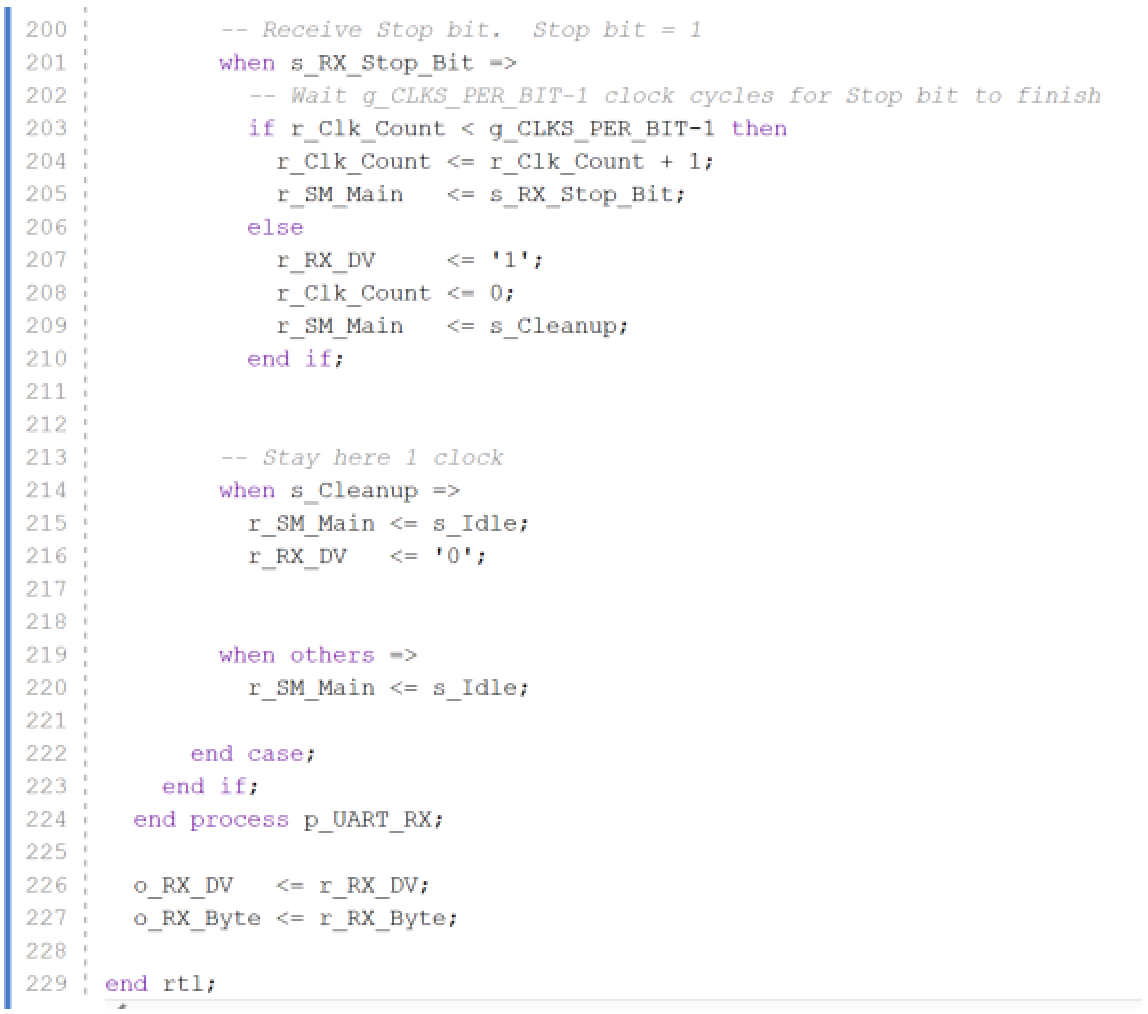
```
  signal o_RX_DVB: std_logic;
  signal o_RX_DVA: std_logic;

  component STD_FIFO is
    Generic (
        constant DATA_WIDTH : positive :=   8;
        constant FIFO_DEPTH : positive := 256
    );
    Port (
        CLK     : in  STD_LOGIC;
        RST     : in  STD_LOGIC;
        WriteEn : in  STD_LOGIC;
        DataIn  : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        ReadEn  : in  STD_LOGIC;
        DataOut : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        Empty   : out STD_LOGIC;
        Full    : out STD_LOGIC
    );
end component;


begin
 FIFO_A_FULL <= FAB;
 FIFO_B_FULL <= FBB;


 GEN_FIFO_A: STD_FIFO
           generic map (DATA_WIDTH => width, FIFO_DEPTH => depth )
           port map(
                  CLK       => i_Clk,
                  RST       => fifo_rst,
                  WriteEn   => o_RX_DVA,
                  DataIn    => o_RX_Byte,
                  ReadEn    => FIFO_A_read,
                  DataOut   => DATAOUT_FIFO_A,
                  Empty     => FIFO_A_EMPTY,
                  Full      => FAB

                   );

   COND: process(FAB, FBB)
    begin
    if(FAB = '1') then
        o_RX_DVB   <= o_RX_DV;
```



```
   else
       o_RX_DVA <= o_RX_DV;
   end if;

   end process;

GEN_FIFO_B: STD_FIFO
          generic map (DATA_WIDTH => width, FIFO_DEPTH => depth )
          port map(
                 CLK        => i_Clk,
                 RST        => FIFO_RST,
                 WriteEn    => o_RX_DVB,
                 DataIn     => o_RX_Byte,
                 ReadEn     => FIFO_B_read,
                 DataOut    => DATAOUT_FIFO_B,
                 Empty      => FIFO_B_EMPTY,
                 Full       => FBB

                  );

 -- Purpose: Double-register the incoming data.
 -- This allows it to be used in the UART RX Clock Domain.
 -- (It removes problems caused by metastabiliy)
 p_SAMPLE : process (i_Clk)
 begin
   if rising_edge(i_Clk) then
     r_RX_Data_R <= i_RX_Serial;
     r_RX_Data   <= r_RX_Data_R;
   end if;
 end process p_SAMPLE;


 -- Purpose: Control RX state machine
 p_UART_RX : process (i_Clk)
 begin
   if rising_edge(i_Clk) then

     case r_SM_Main is

       when s_Idle =>
         r_RX_DV     <= '0';
         r_Clk_Count <= 0;
```

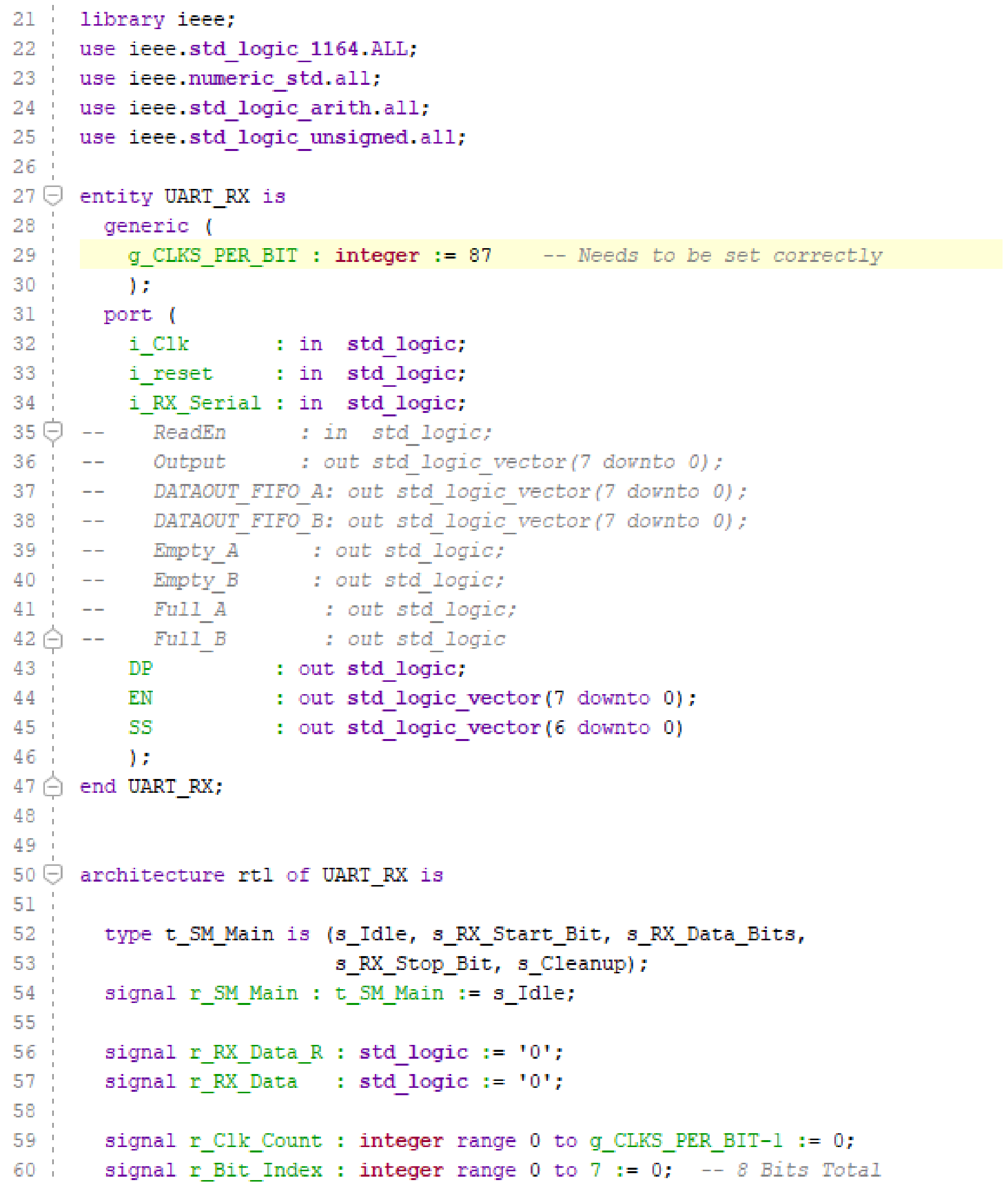


```
        r_Clk_Count <= 0;
        r_Bit_Index <= 0;

        if r_RX_Data = '0' then       -- Start bit detected
          r_SM_Main <= s_RX_Start_Bit;
        else
          r_SM_Main <= s_Idle;
        end if;


      -- Check middle of start bit to make sure it's still low
      when s_RX_Start_Bit =>
        if r_Clk_Count = (g_CLKS_PER_BIT-1)/2 then
          if r_RX_Data = '0' then
            r_Clk_Count <= 0;  -- reset counter since we found the middle
            r_SM_Main   <= s_RX_Data_Bits;
          else
            r_SM_Main   <= s_Idle;
          end if;
        else
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_RX_Start_Bit;
        end if;


      -- Wait g_CLKS_PER_BIT-1 clock cycles to sample serial data
      when s_RX_Data_Bits =>
        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_RX_Data_Bits;
        else
          r_Clk_Count            <= 0;
          r_RX_Byte(r_Bit_Index) <= r_RX_Data;

          -- Check if we have sent out all bits
          if r_Bit_Index < 7 then
            r_Bit_Index <= r_Bit_Index + 1;
            r_SM_Main   <= s_RX_Data_Bits;
          else
            r_Bit_Index <= 0;
            r_SM_Main   <= s_RX_Stop_Bit;
          end if;
        end if;

```

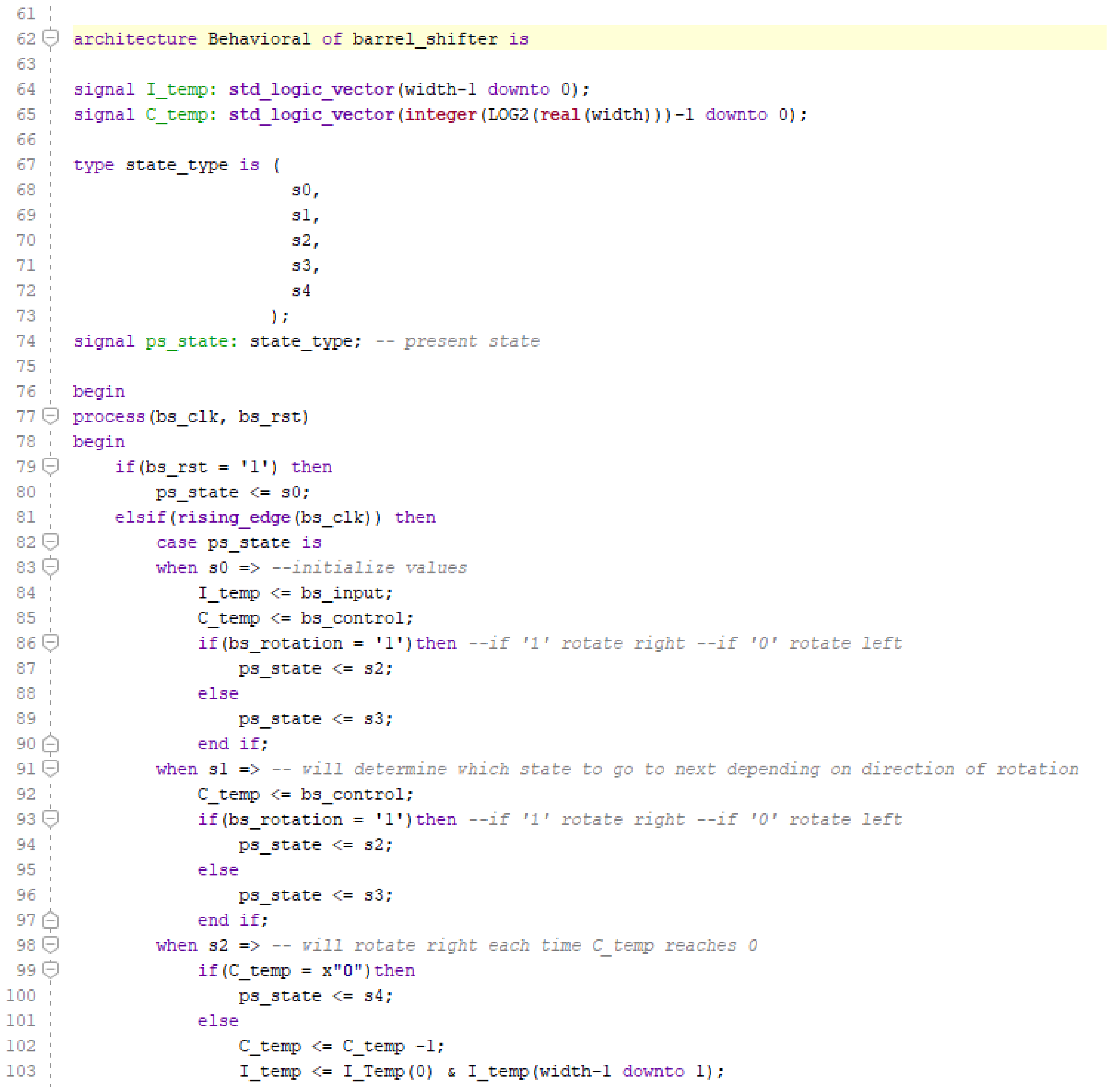


```
        -- Receive Stop bit.  Stop bit = 1
        when s_RX_Stop_Bit =>
          -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
          if r_Clk_Count < g_CLKS_PER_BIT-1 then
            r_Clk_Count <= r_Clk_Count + 1;
            r_SM_Main   <= s_RX_Stop_Bit;
          else
            r_RX_DV     <= '1';
            r_Clk_Count <= 0;
            r_SM_Main   <= s_Cleanup;
          end if;


        -- Stay here 1 clock
        when s_Cleanup =>
          r_SM_Main <= s_Idle;
          r_RX_DV   <= '0';


        when others =>
          r_SM_Main <= s_Idle;

      end case;
    end if;
  end process p_UART_RX;

  o_RX_DV   <= r_RX_DV;
  o_RX_Byte <= r_RX_Byte;

end rtl;
```

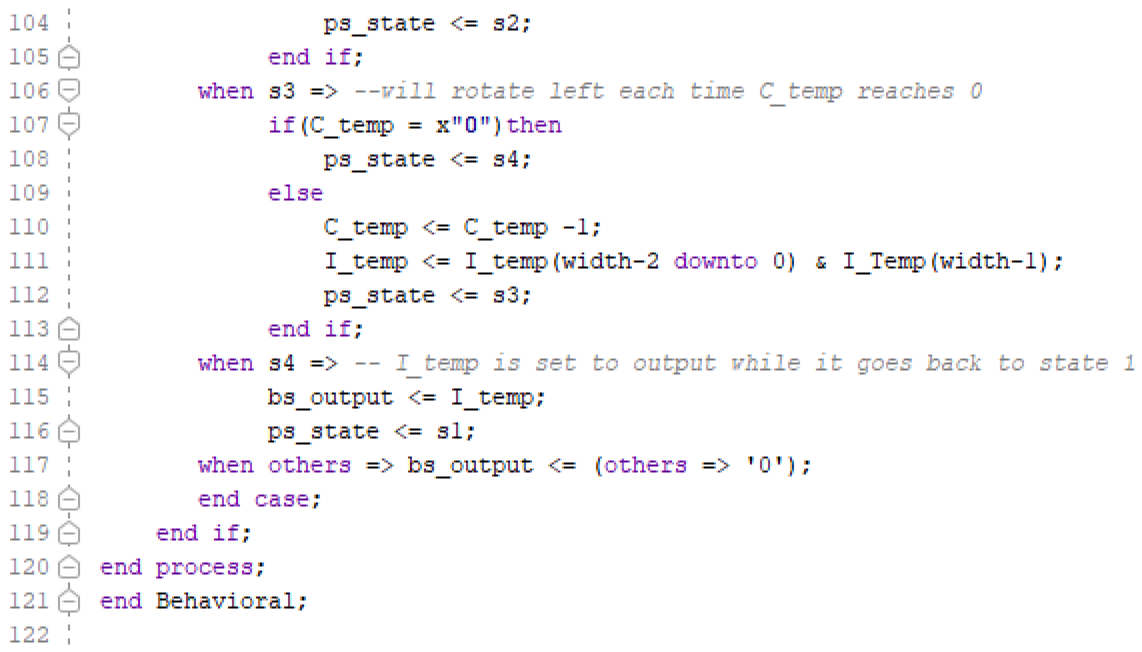
Barrel Shifter Module:



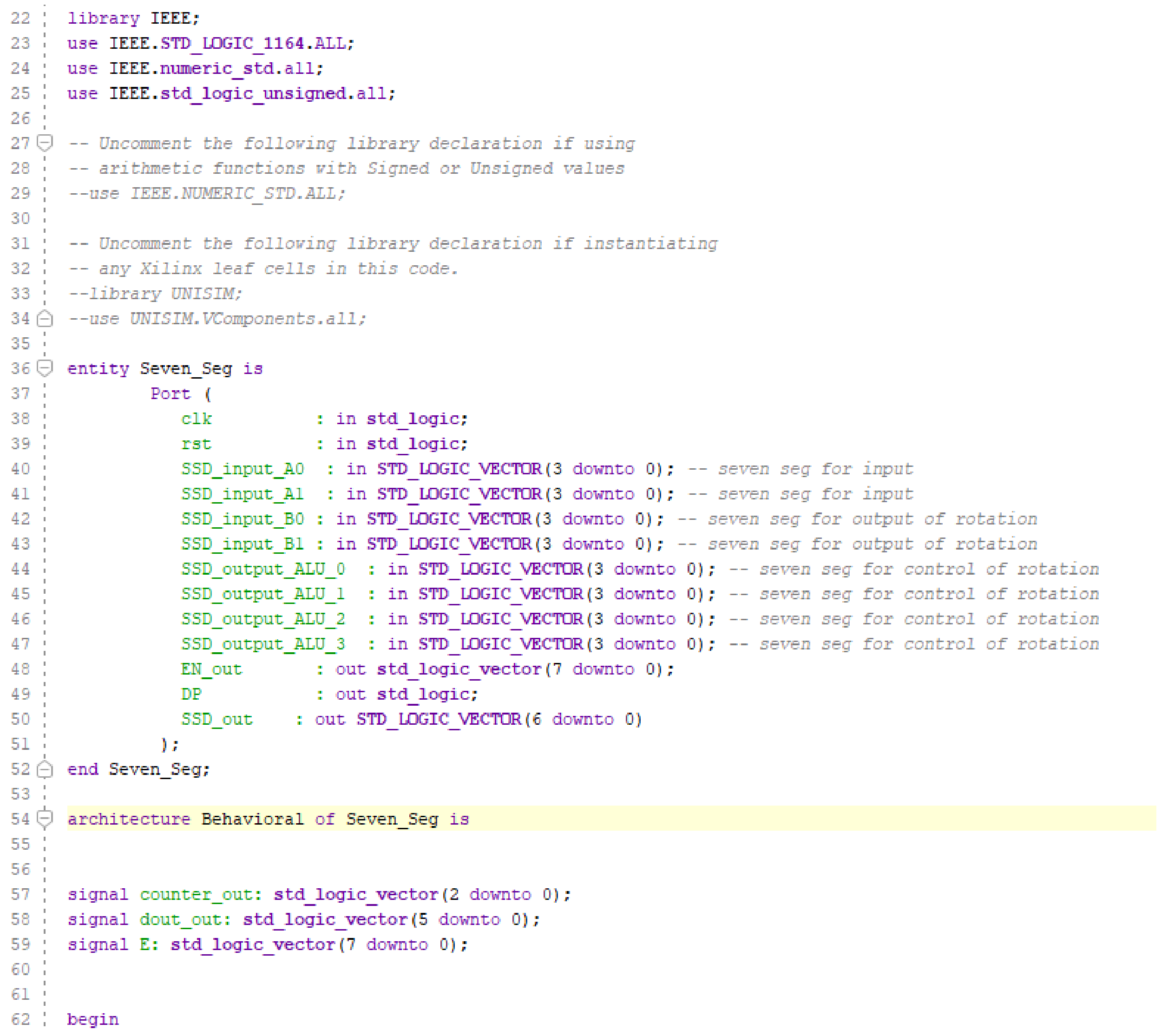
```vhdl
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity UART_RX is
  generic (
    g_CLKS_PER_BIT : integer := 87    -- Needs to be set correctly
    );
  port (
    i_Clk       : in  std_logic;
    i_reset     : in  std_logic;
    i_RX_Serial : in  std_logic;
--    ReadEn      : in  std_logic;
--    Output      : out std_logic_vector(7 downto 0);
--    DATAOUT_FIFO_A: out std_logic_vector(7 downto 0);
--    DATAOUT_FIFO_B: out std_logic_vector(7 downto 0);
--    Empty_A      : out std_logic;
--    Empty_B      : out std_logic;
--    Full_A        : out std_logic;
--    Full_B        : out std_logic
    DP          : out std_logic;
    EN          : out std_logic_vector(7 downto 0);
    SS          : out std_logic_vector(6 downto 0)
    );
end UART_RX;


architecture rtl of UART_RX is

  type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
                     s_RX_Stop_Bit, s_Cleanup);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_RX_Data_R : std_logic := '0';
  signal r_RX_Data   : std_logic := '0';

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
```



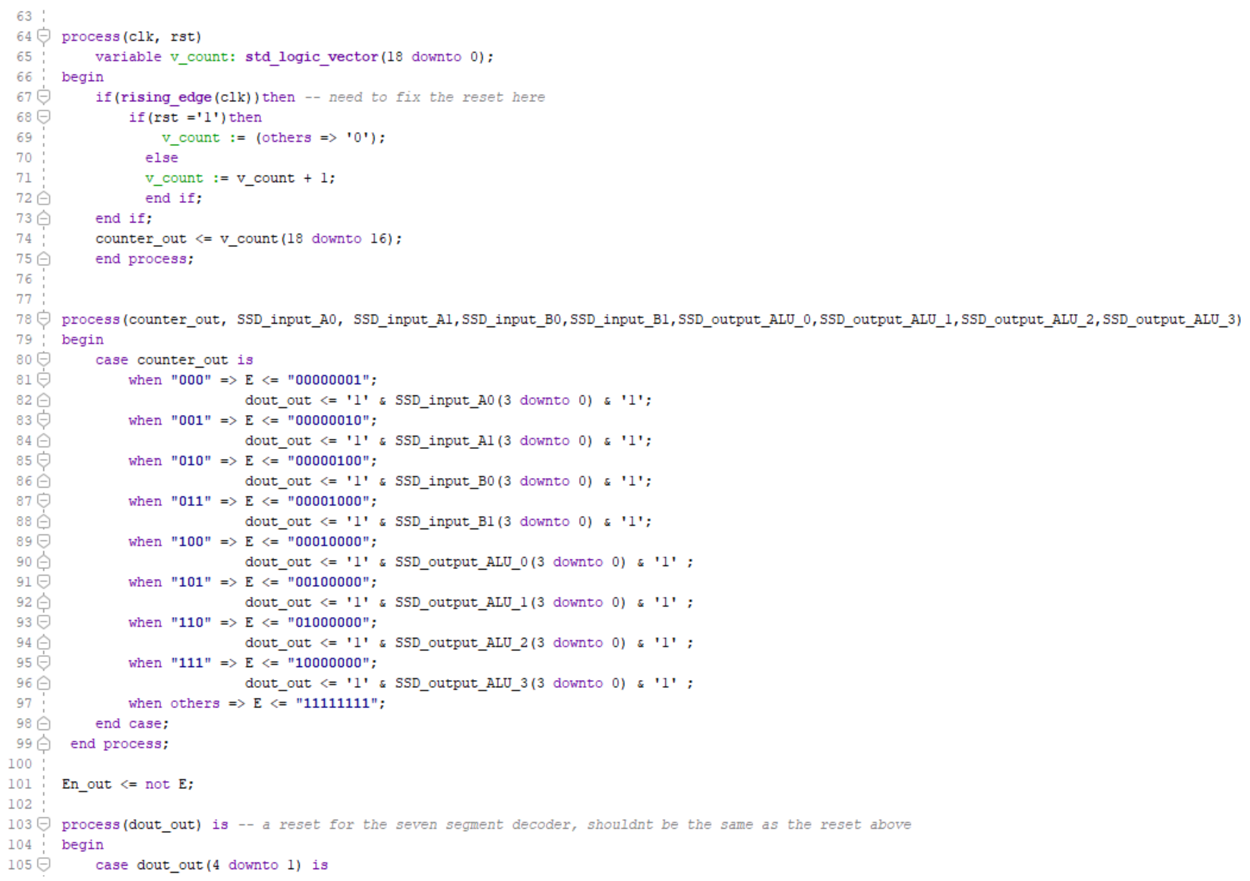
```vhdl

architecture Behavioral of barrel_shifter is

signal I_temp: std_logic_vector(width-1 downto 0);
signal C_temp: std_logic_vector(integer(LOG2(real(width)))-1 downto 0);

type state_type is (
                     s0,
                     s1,
                     s2,
                     s3,
                     s4
                   );
signal ps_state: state_type; -- present state

begin
process(bs_clk, bs_rst)
begin
    if(bs_rst = '1') then
        ps_state <= s0;
    elsif(rising_edge(bs_clk)) then
        case ps_state is
        when s0 => --initialize values
            I_temp <= bs_input;
            C_temp <= bs_control;
            if(bs_rotation = '1')then --if '1' rotate right --if '0' rotate left
                ps_state <= s2;
            else
                ps_state <= s3;
            end if;
        when s1 => -- will determine which state to go to next depending on direction of rotation
            C_temp <= bs_control;
            if(bs_rotation = '1')then --if '1' rotate right --if '0' rotate left
                ps_state <= s2;
            else
                ps_state <= s3;
            end if;
        when s2 => -- will rotate right each time C_temp reaches 0
            if(C_temp = x"0")then
                ps_state <= s4;
            else
                C_temp <= C_temp -1;
                I_temp <= I_Temp(0) & I_temp(width-1 downto 1);
```


```
                ps_state <= s2;
            end if;
       when s3 => --will rotate left each time C_temp reaches 0
            if(C_temp = x"0")then
                ps_state <= s4;
            else
                C_temp <= C_temp -1;
                I_temp <= I_temp(width-2 downto 0) & I_Temp(width-1);
                ps_state <= s3;
            end if;
       when s4 => -- I_temp is set to output while it goes back to state 1
            bs_output <= I_temp;
            ps_state <= s1;
       when others => bs_output <= (others => '0');
       end case;
    end if;
end process;
end Behavioral;
```

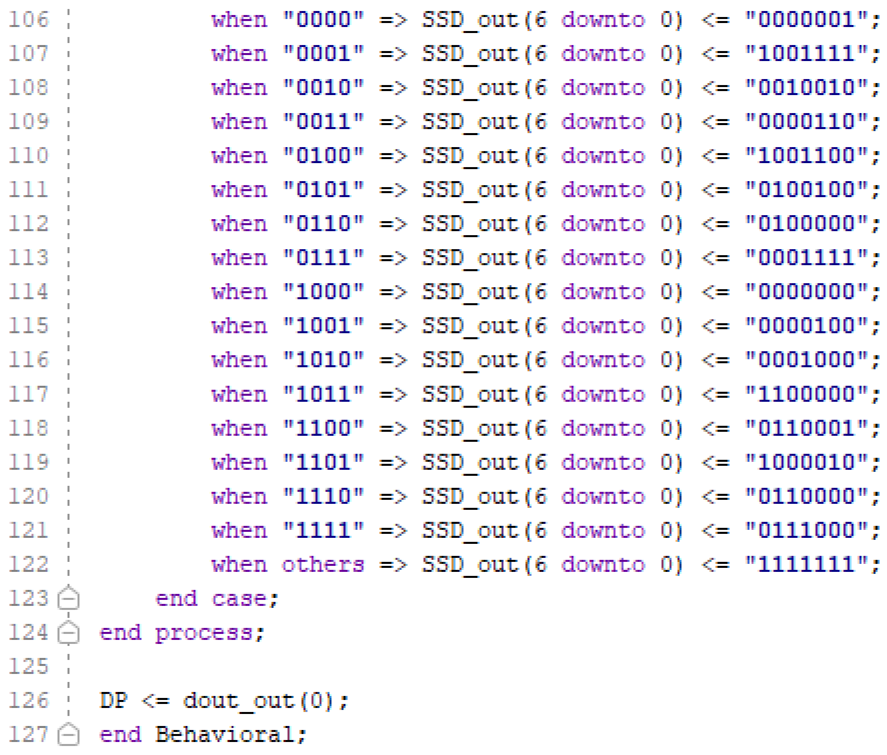
Seven-Segment Display Module:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Seven_Seg is
        Port (
           clk          : in std_logic;
           rst          : in std_logic;
           SSD_input_A0  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
           SSD_input_A1  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
           SSD_input_B0 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
           SSD_input_B1 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
           SSD_output_ALU_0  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           SSD_output_ALU_1  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           SSD_output_ALU_2  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           SSD_output_ALU_3  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
           EN_out       : out std_logic_vector(7 downto 0);
           DP           : out std_logic;
           SSD_out    : out STD_LOGIC_VECTOR(6 downto 0)
         );
end Seven_Seg;

architecture Behavioral of Seven_Seg is


signal counter_out: std_logic_vector(2 downto 0);
signal dout_out: std_logic_vector(5 downto 0);
signal E: std_logic_vector(7 downto 0);


begin
```

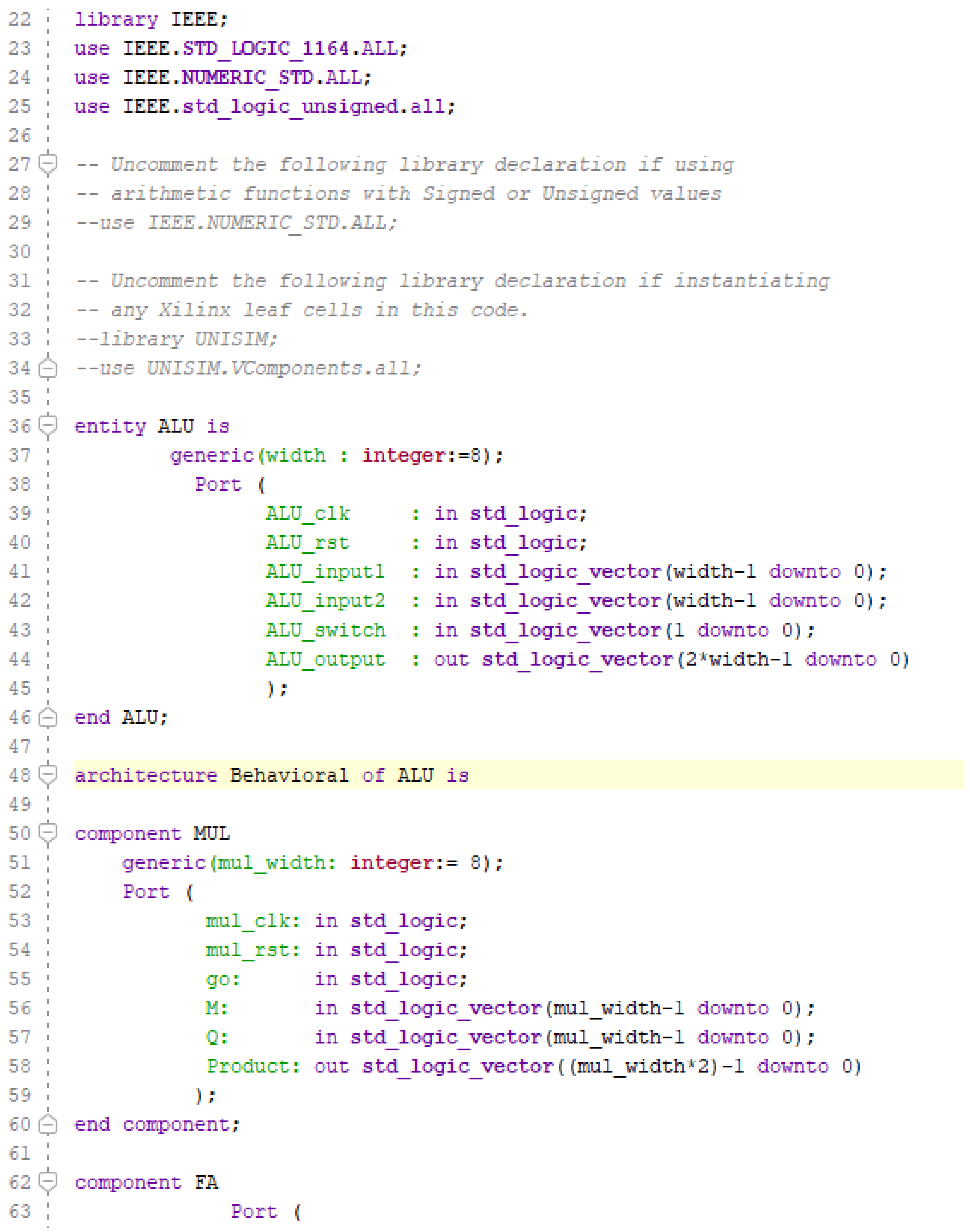


```vhdl

process(clk, rst)
    variable v_count: std_logic_vector(18 downto 0);
begin
    if(rising_edge(clk))then -- need to fix the reset here
        if(rst ='1')then
            v_count := (others => '0');
          else
          v_count := v_count + 1;
          end if;
    end if;
    counter_out <= v_count(18 downto 16);
    end process;


process(counter_out, SSD_input_A0, SSD_input_A1,SSD_input_B0,SSD_input_B1,SSD_output_ALU_0,SSD_output_ALU_1,SSD_output_ALU_2,SSD_output_ALU_3)
begin
    case counter_out is
        when "000" => E <= "00000001";
                      dout_out <= '1' & SSD_input_A0(3 downto 0) & '1';
        when "001" => E <= "00000010";
                      dout_out <= '1' & SSD_input_A1(3 downto 0) & '1';
        when "010" => E <= "00000100";
                      dout_out <= '1' & SSD_input_B0(3 downto 0) & '1';
        when "011" => E <= "00001000";
                      dout_out <= '1' & SSD_input_B1(3 downto 0) & '1';
        when "100" => E <= "00010000";
                      dout_out <= '1' & SSD_output_ALU_0(3 downto 0) & '1' ;
        when "101" => E <= "00100000";
                      dout_out <= '1' & SSD_output_ALU_1(3 downto 0) & '1' ;
        when "110" => E <= "01000000";
                      dout_out <= '1' & SSD_output_ALU_2(3 downto 0) & '1' ;
        when "111" => E <= "10000000";
                      dout_out <= '1' & SSD_output_ALU_3(3 downto 0) & '1' ;
        when others => E <= "11111111";
    end case;
 end process;

En_out <= not E;

process(dout_out) is -- a reset for the seven segment decoder, shouldnt be the same as the reset above
begin
    case dout_out(4 downto 1) is
```

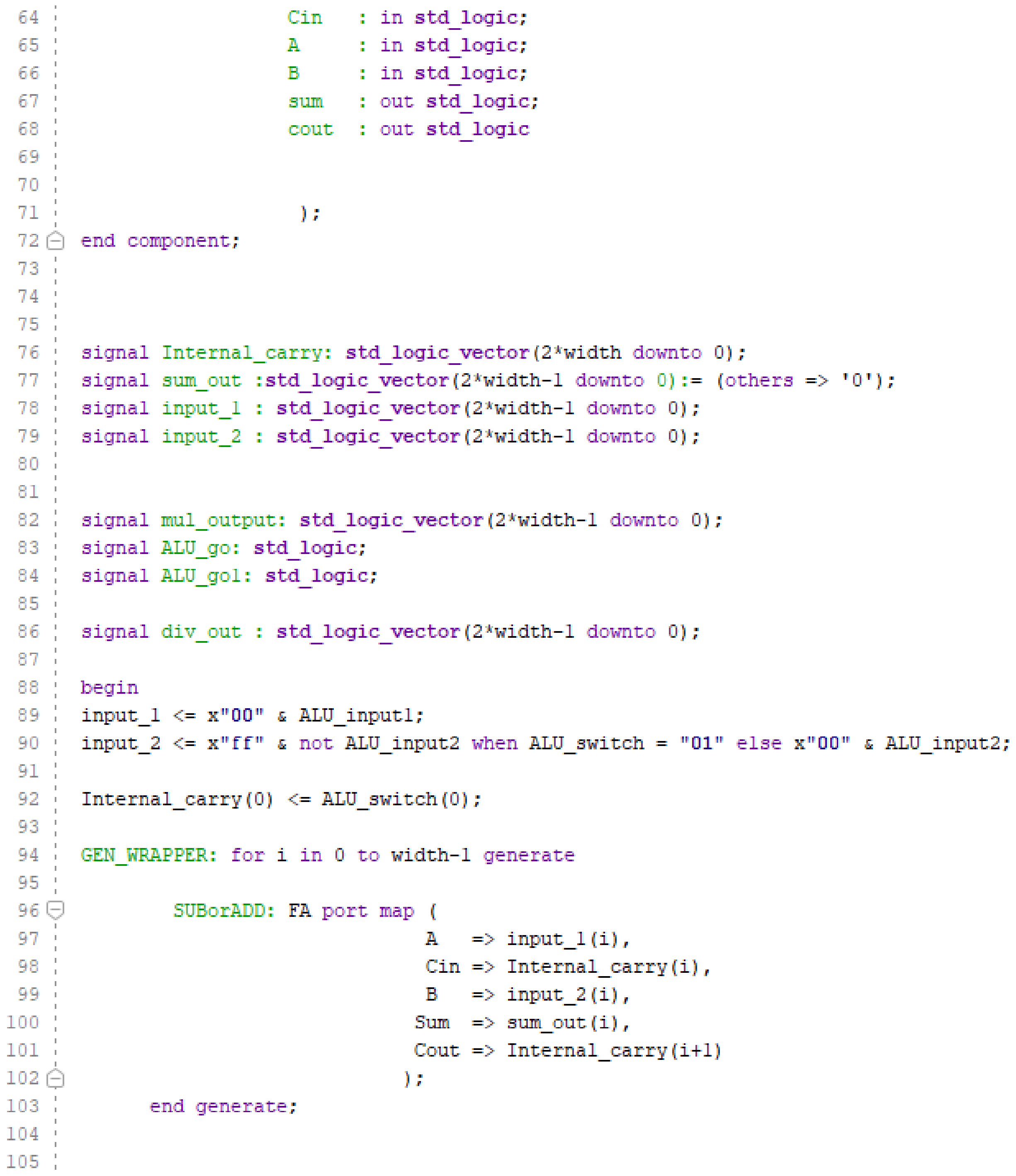


```vhdl
            when "0000" => SSD_out(6 downto 0) <= "0000001";
            when "0001" => SSD_out(6 downto 0) <= "1001111";
            when "0010" => SSD_out(6 downto 0) <= "0010010";
            when "0011" => SSD_out(6 downto 0) <= "0000110";
            when "0100" => SSD_out(6 downto 0) <= "1001100";
            when "0101" => SSD_out(6 downto 0) <= "0100100";
            when "0110" => SSD_out(6 downto 0) <= "0100000";
            when "0111" => SSD_out(6 downto 0) <= "0001111";
            when "1000" => SSD_out(6 downto 0) <= "0000000";
            when "1001" => SSD_out(6 downto 0) <= "0000100";
            when "1010" => SSD_out(6 downto 0) <= "0001000";
            when "1011" => SSD_out(6 downto 0) <= "1100000";
            when "1100" => SSD_out(6 downto 0) <= "0110001";
            when "1101" => SSD_out(6 downto 0) <= "1000010";
            when "1110" => SSD_out(6 downto 0) <= "0110000";
            when "1111" => SSD_out(6 downto 0) <= "0111000";
            when others => SSD_out(6 downto 0) <= "1111111";
        end case;
end process;

DP <= dout_out(0);
end Behavioral;
```

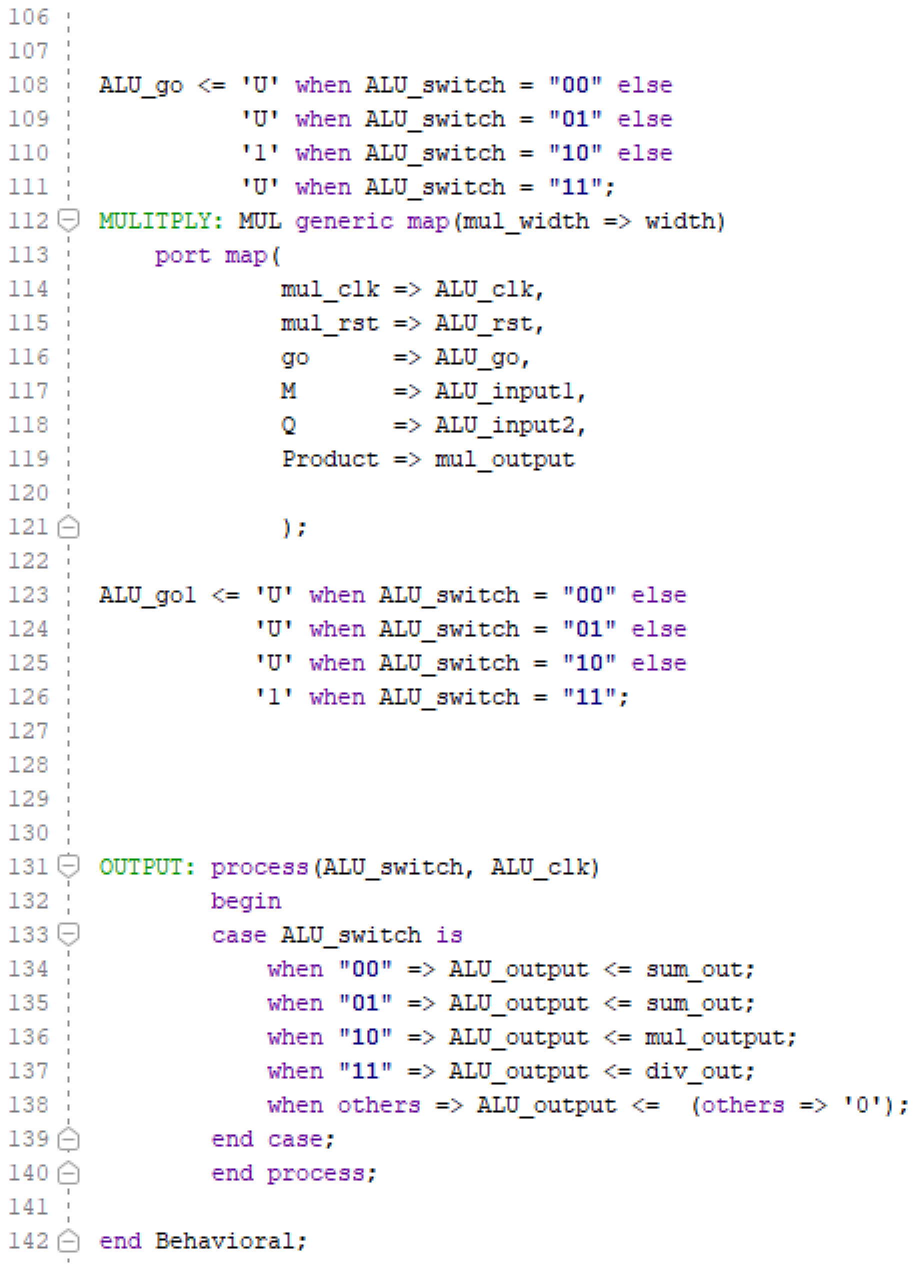
ALU Module:



```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ALU is
        generic(width : integer:=8);
          Port (
                ALU_clk     : in std_logic;
                ALU_rst     : in std_logic;
                ALU_input1  : in std_logic_vector(width-1 downto 0);
                ALU_input2  : in std_logic_vector(width-1 downto 0);
                ALU_switch  : in std_logic_vector(1 downto 0);
                ALU_output  : out std_logic_vector(2*width-1 downto 0)
                );
end ALU;

architecture Behavioral of ALU is

component MUL
    generic(mul_width: integer:= 8);
    Port (
           mul_clk: in std_logic;
           mul_rst: in std_logic;
           go:      in std_logic;
           M:       in std_logic_vector(mul_width-1 downto 0);
           Q:       in std_logic_vector(mul_width-1 downto 0);
           Product: out std_logic_vector((mul_width*2)-1 downto 0)
          );
end component;

component FA
             Port (
```

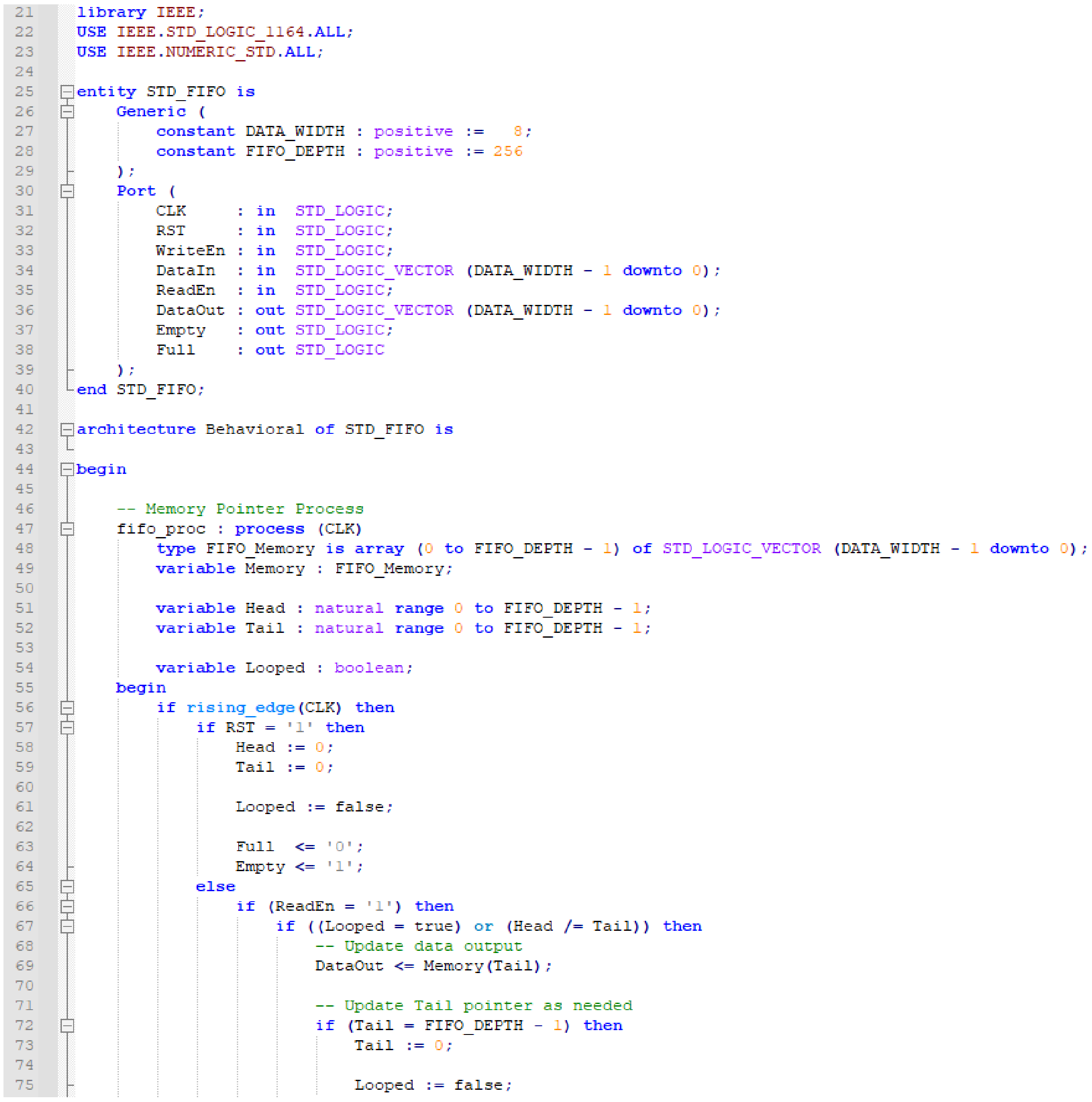


```vhdl
                    Cin   : in std_logic;
                    A     : in std_logic;
                    B     : in std_logic;
                    sum   : out std_logic;
                    cout  : out std_logic


                     );
end component;



signal Internal_carry: std_logic_vector(2*width downto 0);
signal sum_out :std_logic_vector(2*width-1 downto 0):= (others => '0');
signal input_1 : std_logic_vector(2*width-1 downto 0);
signal input_2 : std_logic_vector(2*width-1 downto 0);


signal mul_output: std_logic_vector(2*width-1 downto 0);
signal ALU_go: std_logic;
signal ALU_go1: std_logic;

signal div_out : std_logic_vector(2*width-1 downto 0);

begin
input_1 <= x"00" & ALU_input1;
input_2 <= x"ff" & not ALU_input2 when ALU_switch = "01" else x"00" & ALU_input2;

Internal_carry(0) <= ALU_switch(0);

GEN_WRAPPER: for i in 0 to width-1 generate

        SUBorADD: FA port map (
                              A   => input_1(i),
                              Cin => Internal_carry(i),
                              B   => input_2(i),
                             Sum  => sum_out(i),
                             Cout => Internal_carry(i+1)
                            );
      end generate;


```

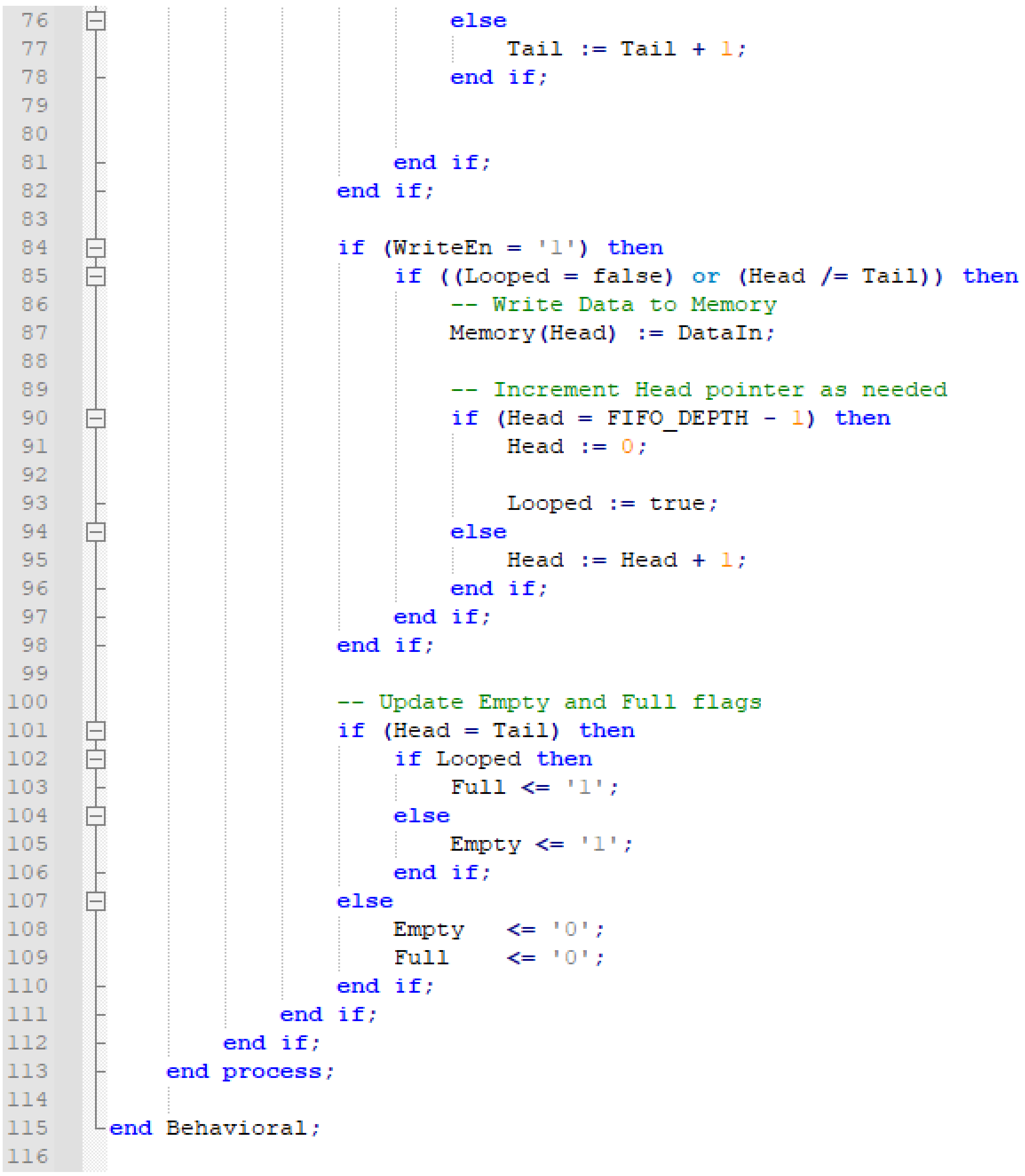


```vhdl


ALU_go <= 'U' when ALU_switch = "00" else
          'U' when ALU_switch = "01" else
          '1' when ALU_switch = "10" else
          'U' when ALU_switch = "11";
MULITPLY: MUL generic map(mul_width => width)
    port map(
             mul_clk => ALU_clk,
             mul_rst => ALU_rst,
             go      => ALU_go,
             M       => ALU_input1,
             Q       => ALU_input2,
             Product => mul_output

             );

ALU_go1 <= 'U' when ALU_switch = "00" else
           'U' when ALU_switch = "01" else
           'U' when ALU_switch = "10" else
           '1' when ALU_switch = "11";




OUTPUT: process(ALU_switch, ALU_clk)
        begin
        case ALU_switch is
            when "00" => ALU_output <= sum_out;
            when "01" => ALU_output <= sum_out;
            when "10" => ALU_output <= mul_output;
            when "11" => ALU_output <= div_out;
            when others => ALU_output <=  (others => '0');
        end case;
        end process;

end Behavioral;
```

FIFO Module:



```vhdl
library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

entity STD_FIFO is
    Generic (
        constant DATA_WIDTH : positive :=   8;
        constant FIFO_DEPTH : positive := 256
    );
    Port (
        CLK     : in  STD_LOGIC;
        RST     : in  STD_LOGIC;
        WriteEn : in  STD_LOGIC;
        DataIn  : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        ReadEn  : in  STD_LOGIC;
        DataOut : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        Empty   : out STD_LOGIC;
        Full    : out STD_LOGIC
    );
end STD_FIFO;

architecture Behavioral of STD_FIFO is

begin

    -- Memory Pointer Process
    fifo_proc : process (CLK)
        type FIFO_Memory is array (0 to FIFO_DEPTH - 1) of STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
        variable Memory : FIFO_Memory;

        variable Head : natural range 0 to FIFO_DEPTH - 1;
        variable Tail : natural range 0 to FIFO_DEPTH - 1;

        variable Looped : boolean;
    begin
        if rising_edge(CLK) then
            if RST = '1' then
                Head := 0;
                Tail := 0;

                Looped := false;

                Full  <= '0';
                Empty <= '1';
            else
                if (ReadEn = '1') then
                    if ((Looped = true) or (Head /= Tail)) then
                        -- Update data output
                        DataOut <= Memory(Tail);

                        -- Update Tail pointer as needed
                        if (Tail = FIFO_DEPTH - 1) then
                            Tail := 0;

                            Looped := false;
```



```vhdl
                        else
                            Tail := Tail + 1;
                        end if;


                    end if;
                end if;

                if (WriteEn = '1') then
                    if ((Looped = false) or (Head /= Tail)) then
                        -- Write Data to Memory
                        Memory(Head) := DataIn;

                        -- Increment Head pointer as needed
                        if (Head = FIFO_DEPTH - 1) then
                            Head := 0;

                            Looped := true;
                        else
                            Head := Head + 1;
                        end if;
                    end if;
                end if;

                -- Update Empty and Full flags
                if (Head = Tail) then
                    if Looped then
                        Full <= '1';
                    else
                        Empty <= '1';
                    end if;
                else
                    Empty   <= '0';
                    Full    <= '0';
                end if;
            end if;
        end if;
    end process;

end Behavioral;
```

Analysis:
Through a large amount of problems, we were finally able to create our design compatible with UART. The main problem that occurred was the transferring of data between UART and the FPGA. Initially we had used other code found online to help with the connection, however it was difficult to do so. To solve the problem we used Tera Term which automatically connects the computer inputs to the FPGA. Our connections were simple as all code was either created from previous labs or provided from class. We have small edits such as the 7 segment display and a conversion between ASCII to binary.

Conclusion:
In this lab, we were successfully able to create a UART compatible machine utilizing FIFOs, ALUs, Barrel Shifters, and Seven Segments. Using previous code such as the ALU, Barrel Shifter and seven segments from previous labs, and the code provided during class, we were able to connect these modules with ease. Our main issue was UART as we did have trouble connecting and transferring the data correctly into the FPGA, but with endless debugging we were able to get the right ASCII input we needed to fully say our code is working.