3e8.54 XSvhd.4Please see the VHDL program below or in the accompanying `.zip` file (if published by your instructor). This program was kindly written and contributed by Xilinx application engineering, but it has not been further checked for correctness and coding style

```
--**********************************
-- PROBLEM : WAKERLY - 8.54
-- FILES :
--   8_54_top.vhd : top level file
--   8_54_par2ser.vhd : parallel to serial converter
--   8_54_control.vhd : control module
--   8_54_shift_synch.vhd : 8 bit shift register
--
-- DESCRIPTION :
--   Creates a parallel to serial converter.
--   Data in is described as 8 x 8bit modules,
--   with a single 8 bit data bus that carries
--   data of the format given in Figure 8-55.
--   Each serial link has its own SYNCH(i) line;
--   the pulses should be staggered so SYNCH(i+1)
--   occurs 1 clock cycle after SYNCH(i).
--
--   Because of this, the load_synch line should
--   also be staggered so the data transmitted
--   over the serial link will correspond to its
--   associated SYNCH line.
--**********************************

-- library declarations
library IEEE;
use IEEE.std_logic_1164.all;

-- top level entity declaration
entity wak_8_54_top is
    port (
        data: in STD_LOGIC_VECTOR (63 downto 0);
        clock: in STD_LOGIC;
        synch: buffer STD_LOGIC_VECTOR (7 downto 0);
        sdata: out STD_LOGIC_VECTOR (7 downto 0)
    );
end wak_8_54_top;

architecture wak_8_54_arch of wak_8_54_top is

signal load_shift_master: std_logic;
signal synch_master: std_logic;
signal load_shift: std_logic_vector (7 downto 0);

--component declarations
component par2ser is
    port (
        clock: in STD_LOGIC;
        data: in STD_LOGIC_VECTOR (7 downto 0);
        load_shift: in STD_LOGIC;
        sdata: out STD_LOGIC
    );
end component;
```

```
component control is
    port (
        clock: in STD_LOGIC;
        load_shift: out STD_LOGIC;
        synch: out STD_LOGIC
    );
end component;

component shift_synch is
    port (
        clock: in STD_LOGIC;
        synch_in: in STD_LOGIC;
        synch: buffer STD_LOGIC_VECTOR (7 downto 0)
    );
end component;

begin

--component instantiations
S1: shift_synch port map (clock=>clock, synch_in=>synch_master, synch=>synch);
S2: shift_synch port map (clock=>clock, synch_in=>load_shift_master,
synch=>load_shift);

U1: par2ser port map (clock=>clock, data=>data(7 downto 0), load_shift=>load_shift(0),
sdata=>sdata(0));
U2: par2ser port map (clock=>clock, data=>data(15 downto 8),
load_shift=>load_shift(1), sdata=>sdata(1));
U3: par2ser port map (clock=>clock, data=>data(23 downto 16),
load_shift=>load_shift(2), sdata=>sdata(2));
U4: par2ser port map (clock=>clock, data=>data(31 downto 24),
load_shift=>load_shift(3), sdata=>sdata(3));
U5: par2ser port map (clock=>clock, data=>data(39 downto 32),
load_shift=>load_shift(4), sdata=>sdata(4));
U6: par2ser port map (clock=>clock, data=>data(47 downto 40),
load_shift=>load_shift(5), sdata=>sdata(5));
U7: par2ser port map (clock=>clock, data=>data(55 downto 48),
load_shift=>load_shift(6), sdata=>sdata(6));
U8: par2ser port map (clock=>clock, data=>data(63 downto 56),
load_shift=>load_shift(7), sdata=>sdata(7));

U9: control port map (clock=>clock, load_shift=>load_shift_master,
synch=>synch_master);

end wak_8_54_arch;
```

```
--***********************************
-- Basically an 8-bit shift register
--  takes the synch_in signal as an
--  input, and outputs an 8 bit signal,
--  each consecutive bit delayed by one
--  from the previous bit.

-- library declaration
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-- top level entity declaration
entity shift_synch is
    port (
        clock: in STD_LOGIC;
        synch_in: in STD_LOGIC;
        synch: buffer STD_LOGIC_VECTOR (7 downto 0)
    );
end shift_synch;

architecture shift_synch_arch of shift_synch is

begin

-- low order synch signal is simply passed through
-- to output.  all others are delayed.
  synch(0) <= synch_in;

  process(clock)
  begin
    if clock'event and clock='1' then

      for I in 0 to 6 loop
        synch(I+1) <= synch(I);
      end loop;
    end if;
  end process;

end shift_synch_arch;
```

```
--************************************
-- Parallel to serial converter
--  Data is entered through 8 bit DATA bus
--  It is loaded into the register when
--  load_shift is low.  If load_shift is
--  high, shift data serially out through sdata

-- library declarations
library IEEE;
use IEEE.std_logic_1164.all;

-- top level entity declaration
entity par2ser is
    port (
        clock: in STD_LOGIC;
        data: in STD_LOGIC_VECTOR (7 downto 0);
        load_shift: in STD_LOGIC;
        sdata: out STD_LOGIC
    );
end par2ser;

architecture par2ser_arch of par2ser is

-- internal signal declaration
signal REG: STD_LOGIC_VECTOR(7 downto 0);
signal DIN: std_logic;

begin

-- DIN <= 0 will set the high order bit to be
-- zero once data is loaded in.
DIN <= '0';

-- process to create shift register
--accomplished by simply taking the DIN signal
--and concatenating on the end the previous
--6 high order bits.
process (clock)
  begin
    if clock'event and clock='1' then
      if load_shift = '0' then
       REG <= data;
      else
        REG <= DIN & REG(7 downto 1);
      end if;
    end if;
  sdata <= REG(0);
end process;

end par2ser_arch;
```

```
--*****************************
-- Control logic
--  controls the loading of the
--  parallel to serial shift register
--  through the load_shift signal.
--  also, controls the synch word.
--  this occurs every 256 clock cycles.

--library declaration
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

--top level entity declaration
entity control is
    port (
        clock: in STD_LOGIC;
        load_shift: out STD_LOGIC;
        synch: out STD_LOGIC
    );
end control;

architecture control_arch of control is
--internal signal declaration
signal COUNT: STD_LOGIC_VECTOR(7 downto 0);
signal load: STD_LOGIC;
begin
load <= '0';  --define constant

process (clock)
  begin
    if clock'event and clock='1' then

      count <= count + 1;

      if count(2 downto 0) = "110" then
        load_shift <= load;
      else
        load_shift <= not load;
      end if;

      if count = 254 then
        synch <= '1';
      else
        synch <= '0';
      end if;

    end if;
end process;

end control_arch;
```