```
--------------------------------------------------------------------------------
-- UART Package Declaration
--
-- by Weijun Zhang, 05/2001
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
package my_package is
    FUNCTION parity(inputs: std_logic_vector(7 downto 0)) RETURN std_logic;
end my_package;

PACKAGE body my_package is
    FUNCTION parity(inputs: std_logic_vector(7 downto 0)) RETURN std_logic is
    variable temp: std_logic;
    begin
        temp:='0';
        for i in 7 downto 0 loop
            temp:=temp xor inputs(i);
        end loop;
        return temp;
    end parity;
end my_package;

--------------------------------------------------------------------------------
-- Modeling UART: HD-6402
-- UART Receiver Model (behavior modeling)
--
-- by Weijun Zhang, 05/2001
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.my_package.all;
--------------------------------------------------------------------------------
entity recei
is

port(   RRC:            in std_logic;
        MR:             in std_logic;
        RRI:            in std_logic;
        SFD:            in std_logic;
        RRD:            in std_logic;
        DRR:            in std_logic;
        CRL:            in std_logic;
        CTRLWORD:       in std_logic_vector(4 downto 0);
        OE:             out std_logic;
        PE:             out std_logic;
        FE:             out std_logic;
        DR:             out std_logic;
        RBR:            out std_logic_vector(7 downto 0)
);
end recei;
--------------------------------------------------------------------------------
architecture behv of recei is

    constant reg_init: std_logic_vector(11 downto 0):="111111111111";

    signal ctrl_word: std_logic_vector(4 downto 0);
    signal recei_reg: std_logic_vector(11 downto 0);
    signal rri_temp: std_logic;
    signal dr_sig,pe_sig,oe_sig,fe_sig: std_logic;
```

```
    signal rx_empty: boolean;
    signal rbr_temp: std_logic_vector(7 downto 0);

begin

P1: process(CRL,CTRLWORD)
begin
    if CRL='1' then
        ctrl_word <= CTRLWORD;
    end if;
end process;

    rri_temp <= RRI;

P2: process
    variable cnt16: integer range 0 to 15;
    begin

        wait until RRC'event and
RRC='1';

        if MR='1' then                  -- initialize and wait to receive data...
            cnt16 := 0 ;                -- Original state...
            rx_empty <= true;
            oe_sig<='0';
            recei_reg <=
reg_init;
        elsif (rx_empty and rri_temp='0') then
            cnt16 := 0;                 -- start reading the bit come in...
            rx_empty <= false;
            if dr_sig/='0' then oe_sig<='1';
            elsif dr_sig='0' then oe_sig<='0';
            end if;
            recei_reg <= reg_init;
        elsif (cnt16=7 and (not rx_empty))then
            recei_reg <= rri_temp & recei_reg(11 downto 1);
            cnt16 := cnt16 +1;
        elsif cnt16 = 15 then           -- finish reading 1-bit of receipt
            cnt16 := 0;                 -- ready to read next..
        else
            cnt16 := cnt16 + 1;
        end if;

        case ctrl_word(4 downto 1) is

            when "0001" =>
                if (not rx_empty) and recei_reg(5)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(5)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
                    if DRR='0' then dr_sig <= '0';
                    end if;
                end if;

            when "0000"|"0011" =>
                if (not rx_empty) and recei_reg(4)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(4)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
```

```
        if DRR='0' then dr_sig <= '0';
        end if;
    end if;

when "0010" =>
    if (not rx_empty) and recei_reg(3)='0' then
        rx_empty <= true;
        if DRR='1' then dr_sig <= '1';
        elsif DRR='0' then dr_sig <='0';
        end if;
    elsif recei_reg(3)/='0' then dr_sig <= '0';
    elsif (rx_empty) then
        if DRR='0' then dr_sig <= '0';
        end if;
    end if;

when "0101" =>
    if (not rx_empty) and recei_reg(4)='0' then
        rx_empty <= true;
        if DRR='1' then dr_sig <= '1';
        elsif DRR='0' then dr_sig <='0';
        end if;
    elsif recei_reg(4)/='0' then dr_sig <= '0';
    elsif (rx_empty) then
        if DRR='0' then dr_sig <= '0';
        end if;
    end if;

when "0100"|"0111" =>
    if (not rx_empty) and recei_reg(3)='0' then
        rx_empty <= true;
        if DRR='1' then dr_sig <= '1';
        elsif DRR='0' then dr_sig <='0';
        end if;
    elsif recei_reg(3)/='0' then dr_sig <= '0';
    elsif (rx_empty) then
        if DRR='0' then dr_sig <= '0';
        end if;
    end if;

when "0110" =>
    if (not rx_empty) and recei_reg(2)='0' then
        rx_empty <= true;
        if DRR='1' then dr_sig <= '1';
        elsif DRR='0' then dr_sig <='0';
        end if;
    elsif recei_reg(2)/='0' then dr_sig <= '0';
    elsif (rx_empty) then
        if DRR='0' then dr_sig <= '0';
        end if;
    end if;

when "1001" =>
    if (not rx_empty) and recei_reg(3)='0' then
        rx_empty <= true;
        if DRR='1' then dr_sig <= '1';
        elsif DRR='0' then dr_sig <='0';
        end if;
    elsif recei_reg(3)/='0' then dr_sig <= '0';
    elsif (rx_empty) then
        if DRR='0' then
            dr_sig <= '0';
        end if;
    end if;
```

```
            when "1000"|"1011" =>
                if (not rx_empty) and recei_reg(2)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(2)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
                    if DRR='0' then dr_sig <= '0';
                    end if;
                end if;

            when "1010" =>
                if (not rx_empty) and recei_reg(1)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(1)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
                    if DRR='0' then dr_sig <= '0';
                    end if;
                end if;

            when "1101" =>
                if (not rx_empty) and recei_reg(2)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(2)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
                    if DRR='0' then dr_sig <= '0';
                    end if;
                end if;

            when "1100"|"1111" =>
                if (not rx_empty) and recei_reg(1)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(1)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
                    if DRR='0' then dr_sig <= '0';
                    end if;
                end if;

            when others =>
                if (not rx_empty) and recei_reg(0)='0' then
                    rx_empty <= true;
                    if DRR='1' then dr_sig <= '1';
                    elsif DRR='0' then dr_sig <='0';
                    end if;
                elsif recei_reg(0)/='0' then dr_sig <= '0';
                elsif (rx_empty) then
                    if DRR='0' then dr_sig <= '0';
                    end if;
                end if;

        end
case;

end process;
```

```
P3: process(RRC,recei_reg,ctrl_word)
begin
    case ctrl_word(4 downto 1) is
        when "0001" =>
            rbr_temp <= "000" & recei_reg(10 downto 6);
        when "0000"|"0011" =>
            rbr_temp <= "000" & recei_reg(9 downto 5);
        when "0010" =>
            rbr_temp <= "000" & recei_reg(8 downto 4);
        when "0101" =>
            rbr_temp <= "00" & recei_reg(10 downto 5);
        when "0100"|"0111" =>
            rbr_temp <= "00" & recei_reg(9 downto 4);
        when "0110" =>
            rbr_temp <= "00" & recei_reg(8 downto 3);
        when "1001" =>
            rbr_temp <= '0' & recei_reg(10 downto 4);
        when "1000"|"1011" =>
            rbr_temp <= '0' & recei_reg(9 downto 3);
        when "1010" =>
            rbr_temp <= '0' & recei_reg(8 downto 2);
        when "1101" =>
            rbr_temp <= recei_reg(10 downto 3);
        when "1100"|"1111" =>
            rbr_temp <= recei_reg(9 downto 2);
        when others =>
            rbr_temp <= recei_reg(8 downto 1);
        end
case;

end process;

P4: process(dr_sig,ctrl_word,rbr_temp,recei_reg)
begin
    if dr_sig='1' then
        if ctrl_word(0)='1' then                -- even parity case
            if ctrl_word(1)='0' then
                if ctrl_word(2)='1' then
                    if (parity(rbr_temp) xor recei_reg(9))='0' then
                        pe_sig <='0';
                    else
                        pe_sig <= '1';
                    end if;
                elsif ctrl_word(2)='0' then
                    if (parity(rbr_temp) xor recei_reg(10))='0' then
                        pe_sig <='0';
                    else
                        pe_sig <= '1';
                    end if;
                end if;
            elsif ctrl_word(1)='1' then
                pe_sig <= '0';
            end if;
        elsif ctrl_word(0)='0' then             -- odd parity case
            if ctrl_word(1)='0' then
                if ctrl_word(2)='1' then
                    if (parity(rbr_temp) xor recei_reg(9))='1' then
                        pe_sig <='0';
                    else
                        pe_sig <= '1';
                    end if;
                elsif ctrl_word(2)='0' then
                    if (parity(rbr_temp) xor recei_reg(10))='1' then
                        pe_sig <='0';
                    else
```

```
                    pe_sig <= '1';
                end if;
            end if;
        elsif ctrl_word(1)='1' then
            pe_sig <= '0';
        end if;
    end if;

    if ctrl_word(2)='0' then
        if recei_reg(11)/='1' then
            fe_sig<='1';
        else
            fe_sig<='0';
        end if;
    elsif ctrl_word(2)='1' then
        if recei_reg(11)/='1' or recei_reg(10)/='1' then
            fe_sig<='1';
        else
            fe_sig<='0';
        end if;
    end if;
else
    pe_sig <= '0';
    fe_sig <= '0';
end if;
end process;

P5: process(SFD,rbr_temp,RRD,dr_sig,pe_sig,fe_sig,oe_sig)
begin
    if RRD='1' then
        RBR <= "ZZZZZZZZ";
    elsif RRD='0' then
        RBR <= rbr_temp;
    end if;

    if SFD='1' then                     -- tri-buffer for other outputs
        DR <= 'Z';
        PE <= 'Z';
        OE <= 'Z';
        FE <= 'Z';
    elsif SFD='0' then
        DR <= dr_sig;
        PE <= pe_sig;
        OE <= oe_sig;
        FE <= fe_sig;
    end if;
end process;
end behv;

--------------------------------------------------------------------------------
-- Modeling UART: HD-6402
-- UART Transmitter Model (behavior modeling)
--
-- by Weijun Zhang, 05/2001
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.my_package.all;
--------------------------------------------------------------------------------
entity trans is
port(   TRC:            in std_logic;
        MR:             in std_logic;
        TBRL:           in std_logic;
```

```
        SFD:            in std_logic;
        CRL:            in std_logic;
        CTRLWORD:       in std_logic_vector(4 downto 0);
        TBR:            in std_logic_vector(7 downto 0);
        TRE:            out std_logic;
        TBRE:           out std_logic;
        TRO:            out std_logic
);
end trans;
--------------------------------------------------------------------------
architecture behv of trans is

    constant C0: unsigned(4 downto 0):="00001";
    constant CI: unsigned(4 downto 0):="01000";
    constant CM: unsigned(4 downto 0):="10000";
    constant MM: unsigned(3 downto 0):="1111";
    signal trans_reg: std_logic_vector(11 downto 0);
    signal trans_reg_temp: std_logic_vector(11 downto 0);
    signal TBR_sig: std_logic_vector(7 downto 0);
    signal old_tbr_sig: std_logic_vector(7 downto 0);
    signal delay: std_logic_vector(0 downto 0);
    signal go: std_logic;
    signal t_pari: std_logic;
    signal clk: std_logic;
    signal i: unsigned(3 downto 0);
    signal ctrl_word: std_logic_vector(4 downto 0);

begin

P1: process(CRL,CTRLWORD)               -- load control word here
    begin
        if CRL='1' then
            ctrl_word<=CTRLWORD;
        end if;
    end process;

-- this process make out the transmit register according to control word.
P2: process(clk,ctrl_word,MR,TBR,TBRL,TBR_sig,t_pari)
begin
    if (MR='1') then
        TBR_sig<="11111111";
        t_pari <= '0';
        trans_reg_temp <= "1111" & TBR_sig;
    else
        if TBRL='0' then
            TBR_sig <= TBR;
            t_pari <= parity(TBR);      -- call function and compute the parity
        end if;

        case ctrl_word is
            when "00000"|"00001" =>
                if t_pari='0' then                              -- odd parity
                    trans_reg_temp <= "111" & "11" & '1' & TBR_sig(4 downto 0) &
'0';
                elsif t_pari='1' then                           -- even parity
                    trans_reg_temp <= "111" & "11" & '0' & TBR_sig(4 downto 0) & '0';
                end if;
            when "00010"|"00011" =>
                if t_pari='0' then                              -- odd parity
                    trans_reg_temp <= "111" & "11" & '0' & TBR_sig(4 downto 0) &
'0';
                elsif t_pari='1' then                           -- even parity
                    trans_reg_temp <= "111" & "11" & '1' & TBR_sig(4 downto 0) & '0';
                end if;
            when "00100"|"00110"|"00101"|"00111" =>             -- no parity
```

```
                trans_reg_temp <= "1111" & "11" & TBR_sig(4 downto 0) & '0';
            when "01000"|"01001" =>
                if t_pari='0' then                                  -- odd parity
                    trans_reg_temp <= "11" & "11" & '1' & TBR_sig(5 downto 0) &
'0';
                elsif t_pari='1' then                               -- even parity
                    trans_reg_temp <= "11" & "11" & '0' & TBR_sig(5 downto 0) & '0';
                end if;
            when "01010"|"01011" =>
                if t_pari='0' then                                  -- odd parity
                    trans_reg_temp <= "11" & "11" & '0' & TBR_sig(5 downto 0) &
'0';
                elsif t_pari='1' then                               -- even parity
                    trans_reg_temp <= "11" & "11" & '1' & TBR_sig(5 downto 0) & '0';
                end if;
            when "01100"|"01110"|"01101"|"01111" =>                 -- no parity
                trans_reg_temp <= "111" & "11" & TBR_sig(5 downto 0) &
'0';
            when "10000"|"10001" =>
                if t_pari='0' then                                  -- odd parity
                    trans_reg_temp <= '1' & "11" & '1' & TBR_sig(6 downto 0) &
'0';
                elsif t_pari='1' then                               -- even parity
                    trans_reg_temp <= '1' & "11" & '0' & TBR_sig(6 downto 0) & '0';
                end if;
            when "10010"|"10011" =>
                if t_pari='0' then                                  -- odd parity
                    trans_reg_temp <= '1' & "11" & '0' & TBR_sig(6 downto 0) &
'0';
                elsif t_pari='1' then                               -- even parity
                    trans_reg_temp <= '1' & "11" & '1' & TBR_sig(6 downto 0) & '0';
                end if;
            when "10100"|"10110"|"10101"|"10111" =>                 -- no parity
                trans_reg_temp <= "11" & "11" & TBR_sig(6 downto 0) &
'0';
            when "11000"|"11001" =>
                if t_pari='0' then                                  -- odd parity
                    trans_reg_temp <= "11" & '1' & TBR_sig(7 downto 0) &
'0';
                elsif t_pari='1' then                               -- even parity
                    trans_reg_temp <= "11" & '0' & TBR_sig(7 downto 0) & '0';
                end if;
            when "11010"|"11011" =>
                if t_pari='0' then                                  -- odd parity
                    trans_reg_temp <= "11" & '0' & TBR_sig(7 downto 0) &
'0';
                elsif t_pari='1' then                               -- even parity
                    trans_reg_temp <= "11" & '1' & TBR_sig(7 downto 0) & '0';
                end if;
            when others =>                                          -- no parity
                trans_reg_temp <= '1' & "11" & TBR_sig(7 downto 0) & '0';
        end case;
    end if;
end process;

-- P1 describes the whole transmission procedure
P3: process
        variable cnt: integer range 0 to 12;
        variable cnt_limit: integer range 0 to 12;
    begin

        wait until TRC'event and TRC='1';

        if MR='1' then
            old_tbr_sig <= "11111111";
```

```
    delay<="0";
    go<='0';
    i <= "0000";
else
    if (i=MM) then
        i<="0000";
        if(go='0') then
            delay<="0";
            cnt := 12;
            TRE <= '1';
            if (old_tbr_sig=TBR_sig) then
                go <= '0';
            elsif (old_tbr_sig/=TBR_sig) then
                go <='1';
            end if;
            trans_reg <= "111111111111";
        elsif (go='1' and delay="0") then
            go<='1';
            cnt:=0;
            delay<=delay+1;
            TRE <= '1';
            trans_reg <= "111111111111";
        elsif (go='1' and delay="1" and cnt=0) then
            go <= '1';
            cnt:=cnt+1;
            delay<=delay+0;
            old_tbr_sig<=TBR_sig;
            TRE<='0';
            trans_reg <= trans_reg_temp;
        elsif (go='1' and delay="1" and cnt/=0) then
            trans_reg <= '1' & trans_reg(11 downto 1);
            case ctrl_word(4 downto 2) is
                when "000" =>   if ctrl_word(0)='0' then
                                    cnt_limit := 8;
                                elsif ctrl_word(0)='1' then
                                    cnt_limit :=9;
                                end if;
                when "001" =>   if ctrl_word(0)='0' then
                                    cnt_limit := 7;
                                elsif ctrl_word(0)='1' then
                                    cnt_limit :=8;
                                end if;
                when "010" =>   if ctrl_word(0)='0' then
                                    cnt_limit := 9;
                                elsif ctrl_word(0)='1' then
                                    cnt_limit :=10;
                                end if;
                when "011" =>   if ctrl_word(0)='0' then
                                    cnt_limit := 8;
                                elsif ctrl_word(0)='1' then
                                    cnt_limit :=9;
                                end if;
                when "100" =>   if ctrl_word(0)='0' then
                                    cnt_limit := 10;
                                elsif ctrl_word(0)='1' then
                                    cnt_limit :=11;
                                end if;
                when "101" =>   if ctrl_word(0)='0' then
                                    cnt_limit := 9;
                                elsif ctrl_word(0)='1' then
                                    cnt_limit :=10;
                                end if;
                when "110" =>   if ctrl_word(0)='0' then
                                    cnt_limit :=11;
                                elsif ctrl_word(0)='1' then
```

```
                                        cnt_limit :=12;
                                    end if;
                            when "111" =>   if ctrl_word(0)='0' then
                                        cnt_limit :=10;
                                    elsif ctrl_word(0)='1' then
                                        cnt_limit :=11;
                                    end if;
                            when others =>
                        end case;

                        if cnt/=cnt_limit then
                            go <= '1';
                            delay<=delay+0;
                            cnt:=cnt+1;
                            TRE<='0';
                        elsif cnt=cnt_limit then
                            go <= '0';
                            delay<="0";
                            TRE<='1';
                        end if;
                    end if;

                    if SFD='1' then
                        TBRE <= 'Z';
                    elsif SFD='0' then
                        if (cnt=0 or cnt=1) then
                            TBRE <= '0';
                        else
                            TBRE <= '1';
                        end if;
                    end if;
                else
                    i<=i+1;
                end if;
            end if;
    end process;

        -- this cocurrent statement handle the serial output of Transmitter
        TRO <= trans_reg(0);

    end behv;

    ------------------------------------------------------------------------------
    -- Modeling UART: HD-6402
    -- Top Level Design
    --
    -- by Weijun Zhang, 05/2001
    ------------------------------------------------------------------------------
    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.std_logic_arith.all;
    use ieee.std_logic_unsigned.all;
    use work.my_package.all;
    ------------------------------------------------------------------------------
    entity UART_6402 is
    port(   TRC:            in std_logic;
            MR:             in std_logic;
            TBRL:           in std_logic;
            SFD:            in std_logic;
            CRL:            in std_logic;
            RRC:            in std_logic;
            RRI:            in std_logic;
            RRD:            in std_logic;
            DRR:            in std_logic;
            CLS2:           in std_logic;
```

```
        CLS1:           in std_logic;
        PI:             in std_logic;
        EPE:            in std_logic;
        SBS:            in std_logic;
        TBR:            in std_logic_vector(7 downto 0);
        TRE:            out std_logic;
        TBRE:           out std_logic;
        TRO:            out std_logic;
        OE:             out std_logic;
        PE:             out std_logic;
        FE:             out std_logic;
        DR:             out std_logic;
        RBR:            out std_logic_vector(7 downto 0)
);
end UART_6402;
--------------------------------------------------------------------------------
architecture struct of UART_6402 is

component trans
port(   TRC:            in std_logic;
        MR:             in std_logic;
        TBRL:           in std_logic;
        SFD:            in std_logic;
        CRL:            in std_logic;
        CTRLWORD:       in std_logic_vector(4 downto 0);
        TBR:            in std_logic_vector(7 downto 0);
        TRE:            out std_logic;
        TBRE:           out std_logic;
        TRO:            out std_logic
);
end component;

component recei
port(   RRC:            in std_logic;
        MR:             in std_logic;
        RRI:            in std_logic;
        SFD:            in std_logic;
        RRD:            in std_logic;
        DRR:            in std_logic;
        CRL:            in std_logic;
        CTRLWORD:       in std_logic_vector(4 downto 0);
        OE:             out std_logic;
        PE:             out std_logic;
        FE:             out std_logic;
        DR:             out std_logic;
        RBR:            out std_logic_vector(7 downto 0)
);
end component;

signal ctrl_w1, ctrl_w2: std_logic_vector(4 downto 0);

begin
    process(CLS2,CLS1,PI,EPE,SBS)
    begin
        ctrl_w1(4) <= CLS2;
        ctrl_w1(3) <= CLS1;
        ctrl_w1(2) <= PI;
        ctrl_w1(1) <= EPE;
        ctrl_w1(0) <= SBS;

        ctrl_w2(4) <= CLS2;
        ctrl_w2(3) <= CLS1;
        ctrl_w2(2) <= EPE;
        ctrl_w2(1) <= SBS;
        ctrl_w2(0) <= PI;
```

```
    end process;

    U1: trans port map(TRC,MR,TBRL,SFD,CRL,ctrl_w1,TBR,TRE,TBRE,TRO);
    U2: recei port map(RRC,MR,RRI,SFD,RRD,DRR,CRL,ctrl_w2,OE,PE,FE,DR,RBR);

end struct;
```