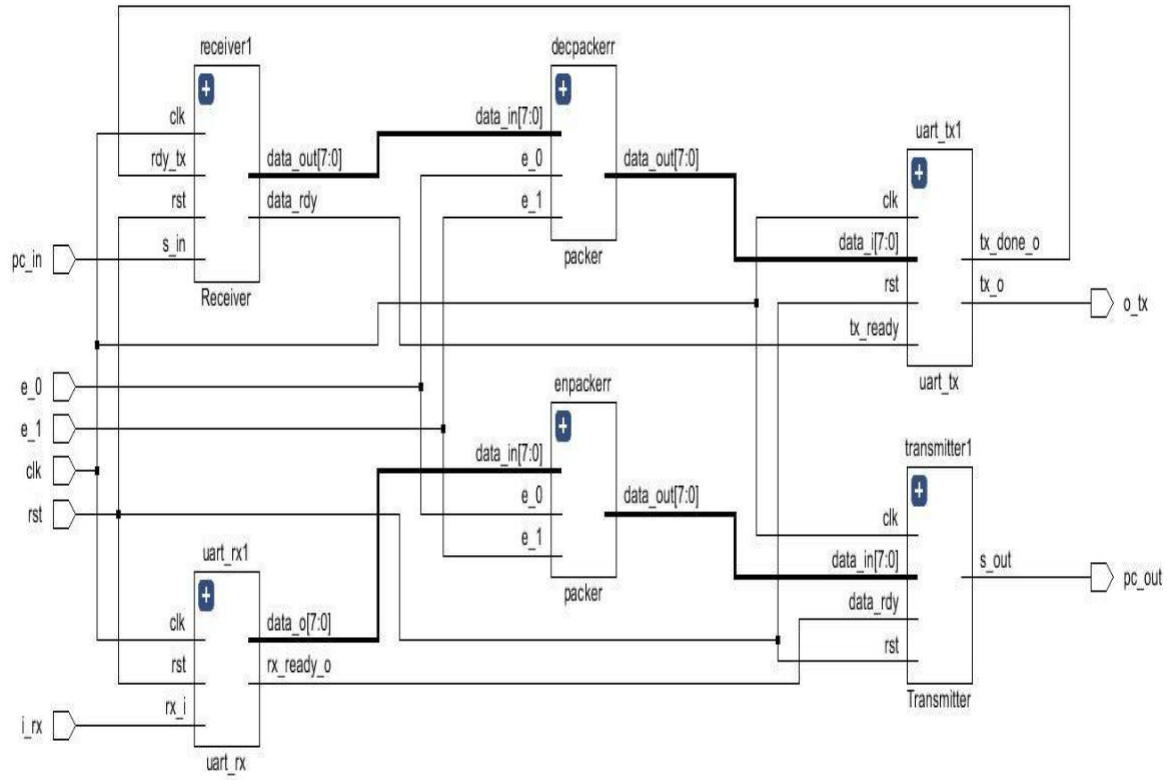
ELEC-457 FPGA PROJE 1

Sait Berk Varımlı 141024054

Volkan Tabanlı 141024034

Alper Kağan Korkmaz 141024053

RTL Schematic



Uart_rx

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

```
 entity uart_rx is
        generic(
                D : integer := 8;
                baudrate : integer := 1085
        );
        port(
                clk : in std_logic;
                rst : in std_logic;
                rx_i : in std_logic;
                data_o : out std_logic_vector( ( D - 1 ) downto 0);
                rx_ready_o : out std_logic
        );
end uart_rx;

architecture rtluart_rx of uart_rx is

        type state_rx is ( s0res, s1idle, s2start, s3receive );
        signal state_next : state_rx := s0res;

begin

        t1 : process( clk, state_next, rst, rx_i ) is
                variable clkcnt : integer := 0;
                variable tickcnt : integer := 0;
                variable tick : boolean := false;
                variable tick_even : boolean := true;
        begin
if( rising_edge( clk ) ) then
```

```
                case state_next is

when s0res =>

                data_o <= ( others => '0' );
                        rx_ready_o <= '0';
                        clkcnt := 0;
                        tickcnt := 0;
                        tick := false;
                        tick_even := true;

                        if( rst = '1' ) then
                        state_next <= s0res;
                        else
                        if( rx_i = '1' ) then
                                state_next <= s1idle;
                        else
                                state_next <= s2start;
                        end if;
                        end if;


when s1idle =>

                        clkcnt := clkcnt + 1;
                        if( clkcnt > baudrate ) then
                        rx_ready_o <= '0';
                        end if;
```

```
                    if( rst = '1' ) then
                    state_next <= s0res;
                    else
                    if( rx_i = '1' ) then
                    state_next <= s1idle;
                    else
                    state_next <= s2start;
                    end if;
                    end if;

when s2start =>

                    data_o <= ( others => '0' );
                    rx_ready_o <= '0';
                    clkcnt := 0;
                    tickcnt := 0;
                    tick := false;
                    tick_even := true;

                    if( rst = '1' ) then
                    state_next <= s0res;
                    else
                    state_next <= s3receive;
                    end if;

when s3receive =>
```

```
clkcnt := clkcnt + 1;
if( clkcnt = baudrate ) then
clkcnt := 0;
tick := true;
else
tick := false;
end if;

if( tick ) then
if( tick_even = false ) then
tick_even := true;
else
if( tickcnt = 0 ) then
elsif( tickcnt > 0 and tickcnt < D + 1 ) then
data_o( tickcnt - 1 ) <= rx_i;
elsif( tickcnt = D + 1 ) then
rx_ready_o <= '1';
end if;
tickcnt := tickcnt + 1;
tick_even := false;
end if;
end if;

if( rst = '1' ) then
state_next <= s0res;
else
if( tickcnt = D + 2 ) then
```

```
                    state_next <= s1idle;
                    else
                    state_next <= s3receive;
                    end if;
                    end if;


            end case;

end if;

        end process;

end rtluart_rx;
```

Uart_tx

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

entity uart_tx is
        generic(
                D : integer := 8;
                baudrate : integer := 1085
        );
        port(
                clk : in std_logic;
                rst : in std_logic;
                tx_ready : in std_logic;
                data_i : in std_logic_vector( ( D - 1 ) downto 0);
```

```
                tx_o : out std_logic;
                tx_done_o : out std_logic
        );
end uart_tx;

architecture rtluart_tx of uart_tx is

        type state_tx is ( s0res, s1idle, s2start, s4_t );
        signal state_next_tx : state_tx := s0res;

begin

        t1 : process( clk, state_next_tx, rst, tx_ready, data_i ) is
                variable clkcnt : integer := 0;
                variable tickcnt : integer := 0;
                variable tick : boolean := false;
                variable tx_start : boolean := false;
                variable data_c_tx : std_logic_vector( ( D - 1 ) downto 0);
        begin
if( rising_edge( clk ) ) then

                case state_next_tx is

when s0res =>

                                        tx_o <= '1';
                                        tx_done_o <= '1';
                                        clkcnt := 0;
                                        tickcnt := 0;
```

```
                                    tick := false;

                                    if( rst = '1' ) then
                                    state_next_tx <= s0res;
                                    else
                                    if( tx_ready = '1' ) then
                                    state_next_tx <= s2start;
                                    else
                                    state_next_tx <= s1idle;
                                    end if;
                                    end if;

when s1idle =>

                                    tx_o <= '1';
                                    tx_done_o <= '1';
                                    clkcnt := 0;
                                    tickcnt := 0;
                                    tick := false;

                                    if( rst = '1' ) then
                                    state_next_tx <= s0res;
                                    else
                                    if( tx_ready = '1' ) then
                                    state_next_tx <= s2start;
                                    else
                                    state_next_tx <= s1idle;
                                    end if;
```

```
                                    end if;

when s2start =>

                                    data_c_tx := data_i;
                                    tx_o <= '0';
                                    tx_done_o <= '0';
                                    clkcnt := 0;
                                    tickcnt := 0;
                                    tick := false;

                                    if( rst = '1' ) then
                                    state_next_tx <= s0res;
                                    else
                                    state_next_tx <= s4_t;
                                    end if;

when s4_t =>

                                    clkcnt := clkcnt + 1;
                                    if( clkcnt = baudrate ) then
                                    clkcnt := 0;
                                    tick := true;
                                    else
                                    tick := false;
                                    end if;

                                    if( tick ) then
```

```
                                    if( tickcnt >= 0 and tickcnt < D-1 ) then
                                    tx_o <= data_c_tx( tickcnt );
                                    tx_done_o <= '0';
                                    elsif( tickcnt = D-1 ) then
                                    tx_o <= (((((('0' xor data_c_tx(0)) xor data_c_tx(1)) xor
data_c_tx(2)) xor data_c_tx(3)) xor data_c_tx(4)) xor data_c_tx(5)) xor data_c_tx(6));
                                    tx_o <= '1';
                                    tx_done_o <= '0';
                                    elsif( tickcnt = D ) then
                                    tx_o <= '1';
                                    tx_done_o <= '1';
                                    end if;
                                    tickcnt := tickcnt + 1;
                                    end if;

                                    if( rst = '1' ) then
                                    state_next_tx <= s0res;
                                    else
                                    if( tickcnt = D + 2 ) then
                                    state_next_tx <= s1idle;
                                    else
                                    state_next_tx <= s4_t;
                                    end if;
                                    end if;


                end case;

end if;
```

```
        end process;

end rtluart_tx;
```

Uart_top

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

entity uart_top is
        port(
                e_0 : in std_logic;
                e_1 : in std_logic;
                clk : in std_logic;
                rst : in std_logic;
                i_rx : in std_logic;
                o_tx : out std_logic;
                pc_in : in std_logic;
                pc_out : out std_logic
        );
end uart_top;

architecture rtluart_top of uart_top is

        component uart_rx
                generic(
                        D : integer
                );
                port(
                        clk : in std_logic;
```

```
                rst : in std_logic;
                rx_i : in std_logic;
                data_o : out std_logic_vector( ( D - 1 ) downto 0 );
                rx_ready_o : out std_logic
        );
    end component uart_rx;

    component uart_tx is
        generic(
            D : integer
        );
        port(clk : in std_logic;
                rst : in std_logic;
                tx_ready : in std_logic;
                data_i : in std_logic_vector( ( D - 1 ) downto 0 );
                tx_o : out std_logic;
                tx_done_o : out std_logic
        );
    end component uart_tx;

component Transmitter is
        generic(
            D : integer
        );
        port(clk : in std_logic;
                rst : in std_logic;
                data_rdy : in std_logic;
                data_in : in std_logic_vector( ( D - 1 ) downto 0 );
                s_out : out std_logic
```

```
        );
end component Transmitter;

    component Receiver is
        generic(
            D : integer
        );
    port(
            clk : in std_logic;
            rst : in std_logic;
            rdy_tx : in std_logic;
            s_in : in std_logic;
            data_out : out std_logic_vector( ( D - 1 ) downto 0 );
            data_rdy : out std_logic
        );
    end component Receiver;

    component packer is
        generic(
            D : integer
        );
        port(
            e_0 : in std_logic;
            e_1 : in std_logic;
            data_in : in std_logic_vector( ( D - 1 ) downto 0 );
            data_out : out std_logic_vector( ( D - 1 ) downto 0 )
        );
    end component packer;
```

```
	signal data_buff : std_logic_vector( 7 downto 0 );
	signal data_buff_encry : std_logic_vector( 7 downto 0 );
	signal buff_ready : std_logic;

	signal r_data_buff : std_logic_vector( 7 downto 0 );
	signal r_data_buff_encry : std_logic_vector( 7 downto 0 );
	signal r_buff_ready : std_logic;

	signal rdy_tx : std_logic;

	signal s_tx_o : std_logic;
	signal pc_o : std_logic;



begin

	uart_rx1 : uart_rx
	generic map(
		D=>8
	)
	port map(clk => clk,rst => rst,rx_i => i_rx,data_o => data_buff,rx_ready_o
=> buff_ready
	);

	uart_tx1 : uart_tx
	generic map(
		D=>8
	)
	port map(
```

```
		clk => clk,rst => rst,tx_ready => r_buff_ready,data_i => r_data_buff,tx_o
=> s_tx_o,tx_done_o => rdy_tx
	);

	receiver1 : Receiver
	generic map(
		D=>8
	)
	port map(clk => clk,rst => rst,rdy_tx => rdy_tx,s_in => pc_in,data_out
=> r_data_buff_encry,data_rdy => r_buff_ready
	);

	transmitter1 : Transmitter
	generic map(
		D=>8
	)
	port map(
		clk => clk,rst => rst,data_rdy => buff_ready,data_in => data_buff_encry,s_out
=> pc_o
	);

	enpackerr : packer
	generic map(
		D=>8
	)
	port map(e_0 => e_0,e_1 => e_1,data_in => data_buff,data_out =>
	data_buff_encry );

	decpackerr : packer
	generic map(
```

```
            D=>8
        )
        port map(e_0 => e_0,e_1 => e_1,data_in => r_data_buff_encry,data_out
=> r_data_buff
        );



        o_tx <= s_tx_o;
        pc_out <= pc_o;

end rtluart_top;
```

Transmitter

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

entity Transmitter is
        generic(
                D : integer := 8;
                all_data : integer := 20;
                message : integer := 16;
                baudrate : integer := 1085
        );
        port(
                clk : in std_logic;
                rst : in std_logic;
                data_rdy : in std_logic;
                data_in : in std_logic_vector( ( D - 1 ) downto 0 );
                s_out : out std_logic
```

```
        );
end Transmitter;

architecture rtltransmitter of Transmitter is

        type transmitstate is ( s0res, s1idle, s2start, s3, s4_t );
        signal state_transmt : transmitstate := s0res;

begin

        t1 : process( clk, rst, data_rdy, data_in ) is
                variable datardyp : std_logic;
                variable alldata : std_logic_vector( ( D * all_data - 1 ) downto 0);
                variable datacnt : integer;
                variable s_cnt : integer;

                variable clkcnt : integer := 0;
                variable tickcnt : integer := 0;
                variable tick : boolean := false;

                variable endtransmission : boolean := false;
        begin
                if( rising_edge( clk ) ) then


                        case state_transmt is

when s0res =>
                                        -- actions
```

```
                    s_out <= '1';
                    datardyp := '0';
                    alldata := ( others => '0' );
                    datacnt := 0;
                    s_cnt := 0;
                    clkcnt := 0;
                    tickcnt := 0;
                    tick := false;
                    endtransmission := false;

                    -- state change
                    if( rst = '1' ) then
                    state_transmt <= s0res;
                    else
                    if( data_rdy = '1' ) then
                    state_transmt <= s2start;
                    else
                    state_transmt <= s1idle;
                    end if;
                    end if;

when s1idle =>
                    s_out <= '1';
                    datardyp := '0';
                    alldata := ( others => '0' );
                    datacnt := 0;
                    s_cnt := 0;
                    clkcnt := 0;
                    tickcnt := 0;
```

```
                                        tick := false;
                                        endtransmission := false;

                                        if( rst = '1' ) then
                                                state_transmt <= s0res;
                                        else
                                                if( data_rdy = '1' ) then
                                                        state_transmt <= s2start;
                                                else
                                                        state_transmt <= s1idle;
                                                end if;
                                        end if;

when s2start =>
                                        s_out <= '1';
                                        datardyp := '0';
                                        datacnt := 0;
                                        s_cnt := 0;
                                        clkcnt := 0;
                                        tickcnt := 0;
                                        tick := false;
                                        endtransmission := false;

                                        alldata := ( others => '0' );
                                        alldata( ( D * all_data - 1 ) downto ( D * all_data - 16 ) )
:= ( others => '1' );

                                        if( rst = '1' ) then
                                        state_transmt <= s0res;
                                        else
```

```
                                        state_transmt <= s3;
                                        end if;


when s3 =>
                                        if( data_rdy = '1' ) then
                                                if( datardyp = '0' ) then
                                                        datardyp := '1';
                                                        alldata( D * ( 3 + datacnt ) - 1 downto D *
( 2 + datacnt ) ) := data_in;
                                                        datacnt := datacnt + 1;
                                                else
                                                end if;
                                        else
                                                if( datardyp = '1' ) then
                                                        datardyp := '0';
                                                else
                                                end if;
                                        end if;

                                        if( datacnt = message ) then
                                                s_out <= '0';
                                        end if;

                                        if( rst = '1' ) then
                                                state_transmt <= s0res;
                                        else
                                                if( datacnt = message ) then
                                                        state_transmt <= s4_t;
                                                else
```

```
                    state_transmt <= s3;
                end if;
            end if;

when s4_t =>
            clkcnt := clkcnt + 1;
            if( clkcnt = baudrate ) then
            clkcnt := 0;
            tick := true;
            else
            tick := false;
            end if;

            if( tick ) then
            if( tickcnt >= 0 and tickcnt < D ) then
            s_out <= alldata( tickcnt + s_cnt * D );
            elsif( tickcnt = D ) then
            s_out <= '1';
            elsif( tickcnt = D + 1 ) then
            s_out <= '1';
            end if;
            tickcnt := tickcnt + 1;
            end if;

            if( tickcnt = D + 2 ) then
            s_out <= '0';
            clkcnt := 0;
```

```
tickcnt := 0;
tick := false;
s_cnt := s_cnt + 1;
end if;

if( s_cnt = all_data ) then
s_out <= '1';
s_cnt := 0;
endtransmission := true;
end if;

if( rst = '1' ) then
state_transmt <= s0res;
else
if( endtransmission ) then
state_transmt <= s1idle;
else
state_transmt <= s4_t;
end if;
end if;


    end case;
    end if;
    end process;

end rtltransmitter;
```

Receiver

```
library ieee;
```

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

entity Transmitter is
	generic(
		D : integer := 8;
		all_data : integer := 20;
		message : integer := 16;
		baudrate : integer := 1085
	);
	port(
		clk : in std_logic;
		rst : in std_logic;
		data_rdy : in std_logic;
		data_in : in std_logic_vector( ( D - 1 ) downto 0 );
		s_out : out std_logic
	);
end Transmitter;

architecture rtltransmitter of Transmitter is

	type transmitstate is ( s0res, s1idle, s2start, s3, s4_t );
	signal state_transmt : transmitstate := s0res;

begin

	t1 : process( clk, rst, data_rdy, data_in ) is
		variable datardyp : std_logic;
		  variable alldata : std_logic_vector( ( D * all_data - 1 ) downto 0);
```

```
        variable datacnt : integer;
        variable s_cnt : integer;

        variable clkcnt : integer := 0;
        variable tickcnt : integer := 0;
        variable tick : boolean := false;

        variable endtransmission : boolean := false;
    begin
        if( rising_edge( clk ) ) then


            case state_transmt is

when s0res =>
                                    -- actions
                                    s_out <= '1';
                                    datardyp := '0';
                                    alldata := ( others => '0' );
                                    datacnt := 0;
                                    s_cnt := 0;
                                    clkcnt := 0;
                                    tickcnt := 0;
                                    tick := false;
                                    endtransmission := false;

                                    -- state change
                                    if( rst = '1' ) then
                                    state_transmt <= s0res;
```

```
                    else
                    if( data_rdy = '1' ) then
                    state_transmt <= s2start;
                    else
                    state_transmt <= s1idle;
                    end if;
                    end if;

when s1idle =>
                    s_out <= '1';
                    datardyp := '0';
                    alldata := ( others => '0' );
                    datacnt := 0;
                    s_cnt := 0;
                    clkcnt := 0;
                    tickcnt := 0;
                    tick := false;
                    endtransmission := false;

                    if( rst = '1' ) then
                              state_transmt <= s0res;
                    else
                              if( data_rdy = '1' ) then
                                        state_transmt <= s2start;
                              else
                                        state_transmt <= s1idle;
                              end if;
                    end if;
```

```
when s2start =>
                                        s_out <= '1';
                                        datardyp := '0';
                                        datacnt := 0;
                                        s_cnt := 0;
                                        clkcnt := 0;
                                        tickcnt := 0;
                                        tick := false;
                                        endtransmission := false;

                                        alldata := ( others => '0' );
                                        alldata( ( D * all_data - 1 ) downto ( D * all_data - 16 ) )
:= ( others => '1' );

                                        if( rst = '1' ) then
                                        state_transmt <= s0res;
                                        else
                                        state_transmt <= s3;
                                        end if;



when s3 =>
                                        if( data_rdy = '1' ) then
                                                if( datardyp = '0' ) then
                                                        datardyp := '1';
                                                        alldata( D * ( 3 + datacnt ) - 1 downto D *
( 2 + datacnt ) ) := data_in;

                                                        datacnt := datacnt + 1;
                                                else
                                                end if;
```

```
                else
                        if( datardyp = '1' ) then
                                datardyp := '0';
                        else
                        end if;
                end if;

                if( datacnt = message ) then
                        s_out <= '0';
                end if;

                if( rst = '1' ) then
                        state_transmt <= s0res;
                else
                        if( datacnt = message ) then
                                state_transmt <= s4_t;
                        else
                                state_transmt <= s3;
                        end if;
                end if;

when s4_t =>
                clkcnt := clkcnt + 1;
                if( clkcnt = baudrate ) then
                clkcnt := 0;
                tick := true;
                else
```

```
tick := false;
end if;

if( tick ) then
if( tickcnt >= 0 and tickcnt < D ) then
s_out <= alldata( tickcnt + s_cnt * D );
elsif( tickcnt = D ) then
s_out <= '1';
elsif( tickcnt = D + 1 ) then
s_out <= '1';
end if;
tickcnt := tickcnt + 1;
end if;

if( tickcnt = D + 2 ) then
s_out <= '0';
clkcnt := 0;
tickcnt := 0;
tick := false;
s_cnt := s_cnt + 1;
end if;

if( s_cnt = all_data ) then
s_out <= '1';
s_cnt := 0;
endtransmission := true;
end if;

 if( rst = '1' ) then
```

```
                                        state_transmt <= s0res;
                                        else
                                        if( endtransmission ) then
                                        state_transmt <= s1idle;
                                        else
                                        state_transmt <= s4_t;
                                        end if;
                                        end if;


        end case;
        end if;
        end process;

end rtltransmitter;
```

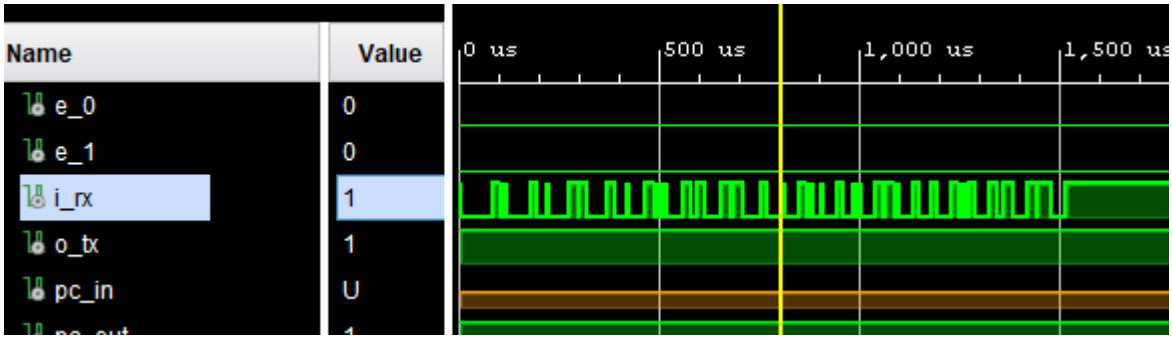
Packer

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

 entity packer is
        generic(
                D : integer := 8
        );
        port(
                e_0 : in std_logic;
                e_1 : in std_logic;
                data_in : in std_logic_vector( ( D - 1 ) downto 0);
                data_out : out std_logic_vector( ( D - 1 ) downto 0)
```

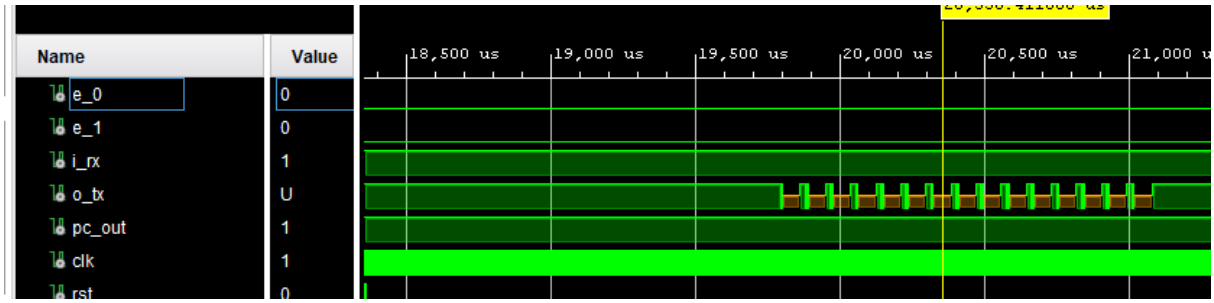
```
        );
end packer;

architecture pckr of packer is
begin
        data_out(0) <= data_in(0) xor e_0;
        data_out(1) <= data_in(1) xor '1';
        data_out(2) <= data_in(2) xor '1';
        data_out(3) <= data_in(3) xor '1';
        data_out(4) <= data_in(4) xor e_1;
        data_out(5) <= data_in(5) xor '1';
        data_out(6) <= data_in(6) xor '1';
        data_out(7) <= data_in(7) xor '1';

end pckr;
```

Simulation

İ_rx



Out



Constrains File

```
## Clock Signal
set_property -dict { PACKAGE_PIN H16    IOSTANDARD LVCMOS33 }
[get_ports { clk }]; #IO_L13P_T2_MRCC_35 Sch=SYSCLK
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4}
[get_ports { clk }];#set

## Switches
set_property -dict { PACKAGE_PIN M20    IOSTANDARD LVCMOS33 }
[get_ports { e_0 }]; #IO_L7N_T1_AD2N_35 Sch=e0
set_property -dict { PACKAGE_PIN M19  IOSTANDARD LVCMOS33 }
[get_ports { e_1 }]; #IO_L7P_T1_AD2P_35 Sch=e1

## Buttons
set_property -dict { PACKAGE_PIN D19    IOSTANDARD LVCMOS33 }
[get_ports { rst }]; #IO_L4P_T0_35 Sch=BTN0
#set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 }
[get_ports { ck_io13 }]; #IO_L23P_T3_34          Sch=CK_IO13

## ChipKit Inner Digital Header
set_property -dict { PACKAGE_PIN U5    IOSTANDARD LVCMOS33 }
[get_ports { pc_out }]; #IO_L19N_T3_VREF_13  Sch=CK_IO26
set_property -dict { PACKAGE_PIN V5    IOSTANDARD LVCMOS33 }
[get_ports { pc_in }]; #IO_L6N_T0_VREF_13   Sch=CK_IO27
set_property -dict { PACKAGE_PIN W10   IOSTANDARD LVCMOS33 }
[get_ports { o_tx }]; #IO_L16P_T2_13       Sch=CK_IO34
set_property -dict { PACKAGE_PIN W6    IOSTANDARD LVCMOS33 }
[get_ports { i_rx }]; #IO_L22N_T3_13       Sch=CK_IO35
```