--UACRT (universal asynchronous receiver and transmitter)

--it's a serial communication using just two wires and it can also detect errors and correct them

--we can send and receive data at the same time as we have two processes runs concurrent (RX and TX)

--put to simulate this project we modified this module to this idea

--we receive a signal from outside and convert it to character using RX process then we transmit this character to signal on TX pin

--if the input signal equal the output signal this mean that our serial communication run will

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity UACRT is

Port ( serial\_in : in STD\_LOGIC; --serial\_in is RX pin

serial\_out : out STD\_LOGIC); --serial\_out is TX pin

end UACRT;

architecture Behavioral of UACRT is

--declaration signal here

signal exchange : character; --this signal carry character from RX process to TX process

signal completed : boolean; --this signal indicate if receiving process has ended or not

--Tx vector

signal TX\_vector : std\_logic\_vector(10 downto 0);

--RX vector

signal RX\_vector : std\_logic\_vector(10 downto 0);

--procedures

--definition of RX

procedure RX (

signal Rx\_data : in std\_logic ;

signal data\_vector : out std\_logic\_vector(0 to 10)) is

begin

--wait the start bit

--wait for 124 us;

--receive data

for i in 0 to 10 loop

data\_vector(i) <= Rx\_data ;

--wait for 104 us ;

end loop ;

--end for

end RX;

--definition of TX

procedure TX(

signal out\_vector : in std\_logic\_vector(10 downto 0);

signal out\_serial : out std\_logic) is

begin

-- Send Start Bit

out\_serial <= '0';

--wait for 104 us;

-- Send Data Byte

for j in 0 to 10 loop

out\_serial <= out\_vector(j);

--wait for 104 us;

end loop;

-- Send Stop Bit

out\_serial <= '1';

--wait for 208 us;

end TX;

--definition of correction code

procedure check(

signal d: in std\_logic\_vector(10 downto 0);

signal check\_out: out std\_logic\_VECTOR(10 downto 0)) is

variable c: std\_LOGIC\_VECTOR(3 downto 0);

begin

--checking code

c(0) := d(0) xor d(2) xor d(4) xor d(6) xor d(8) xor d(10) ;

c(1) := d(1) xor d(2) xor d(5) xor d(6) xor d(9) xor d(10) ;

c(2) := d(3) xor d(4) xor d(5) xor d(6) ;

c(3) := d(7) xor d(8) xor d(9) xor d(10) ;

--correction code

if c = "0001" then

check\_out(0) <= not d(0) ;

elsif c = "0010" then

check\_out(1) <= not d(1) ;

elsif c = "0011" then

check\_out(2) <= not d(2) ;

elsif c = "0100" then

check\_out(3) <= not d(3) ;

elsif c = "0101" then

check\_out(4) <= not d(4) ;

elsif c = "0110" then

check\_out(5) <= not d(5) ;

elsif c = "0111" then

check\_out(6) <= not d(6) ;

elsif c = "1000" then

check\_out(7) <= not d(7) ;

elsif c = "1001" then

check\_out(8) <= not d(8) ;

elsif c = "1010" then

check\_out(9) <= not d(9) ;

elsif c = "1011" then

check\_out(10) <= not d(10) ;

end if;

end check;

--binary to asci procedure definition

procedure Binary\_to\_Asci

( signal B : in STD\_LOGIC\_VECTOR (10 downto 0) ;

signal A : out character

)is

begin

if ( B = "10010011101" ) then

A<='a';

elsif (B = "01010011100") then

A <= 'b';

elsif (B = "11000011111") then

A<= 'c';

elsif (B = "00110011111") then

A<= 'd';

elsif (B = "10100011100") then

A<= 'e';

elsif (B = "01100011101") then

A<= 'f';

elsif (B = "11110011110") then

A<= 'g' ;

elsif (B = "00001010101") then

A<= 'h' ;

elsif (B = "10011010110") then

A<= 'i' ;

elsif (B = "01011010111") then

A<= 'j' ;

elsif (B = "11001010100") then

A<= 'k' ;

elsif (B = "00111010100") then

A<= 'l' ;

elsif (B = "10101010111" )then

A<= 'm' ;

elsif (B = "01101010110") then

A<= 'n' ;

elsif ( B = "11111010101")then

A<= 'o' ;

elsif (B = "00000110100") then

A<='p';

elsif (B = "10010110111") then

A<= 'q';

elsif (B = "01010110110") then

A<= 'r' ;

elsif (B = "11000110101") then

A<= 's' ;

elsif (B = "00110110101") then

A<= 't' ;

elsif (B = "10100110110") then

A<= 'u';

elsif (B = "01100110111") then

A<= 'v' ;

elsif (B = "11110110100") then

A<= 'w';

elsif (B = "00001111111") then

A<= 'x';

elsif (B = "10011111100") then

A<= 'y';

elsif (B = "01011111101") then

A<='z';

elsif (B = "10010000100") then

A<='A';

elsif (B = "01010000101") then

A<='B';

elsif (B = "11000000110") then

A<= 'C';

elsif (B = "00110000110" )then

A<='D';

elsif (B = "10100000101") then

A<='E';

elsif (B = "01100000100") then

A<='F';

elsif (B = "11110000111") then

A<='G';

elsif (B = "00001001100") then

A<='H' ;

elsif (B = "10011001111") then

A<='I';

elsif (B = "01011001110") then

A<='J';

elsif (B = "11001001101") then

A<='K';

elsif (B = "00111001101") then

A<='L' ;

elsif (B = "10101001110") then

A<='M';

elsif (B = "01101001111") then

A<='N';

elsif (B = "11111001100") then

A<='O';

elsif (B = "00000101101") then

A<='P';

elsif (B = "10010101110") then

A<='Q';

elsif (B = "01010101111") then

A<='R';

elsif (B = "11000101100") then

A<='S';

elsif (B = "00110101100") then

A<='T';

elsif ( B = "10100101111")then

A<='U';

elsif (B = "01100101110") then

A<='V';

elsif (B = "11110101101") then

A<='W';

elsif ( B = "00001100110")then

A<= 'X' ;

elsif (B = "10011100101") then

A<='Y';

elsif (B= "01011100100") then

A<='Z';

else

A<='-';

end if ;

end Binary\_to\_Asci;

--asci to binary definition

procedure Asci\_to\_binary

( signal A : in character ;

signal B : out STD\_LOGIC\_VECTOR (10 downto 0)

)is

begin

if(A = 'a') then

B <= "10010011101";

elsif (A = 'b') then

B <= "01010011100";

elsif(A = 'c') then

B<="11000011111";

elsif (A = 'd' ) then

B<="00110011111";

elsif (A = 'e') then

B<= "10100011100";

elsif(A = 'f') then

B<="01100011101";

elsif(A = 'g') then

B<="11110011110";

elsif(A = 'h') then

B<="00001010101";

elsif (A = 'i' ) then

B<="10011010110";

elsif (A = 'j') then

B<= "01011010111";

elsif(A = 'k') then

B<="11001010100";

elsif(A = 'l') then

B<="00111010100";

elsif(A = 'm') then

B<="10101010111";

elsif (A = 'n') then

B <= "01101010110";

elsif(A = 'o') then

B<="11111010101";

elsif (A = 'p' ) then

B<="00000110100";

elsif (A = 'q') then

B<= "10010110111";

elsif(A = 'r') then

B<="01010110110";

elsif(A = 's') then

B<="11000110101";

elsif(A = 't') then

B<="00110110101";

elsif (A = 'u') then

B <= "10100110110";

elsif(A = 'v') then

B<="01100110111";

elsif (A = 'w' ) then

B<="11110110100";

elsif (A = 'x') then

B<= "00001111111";

elsif(A = 'y') then

B<="10011111100";

elsif(A = 'z') then

B<="01011111101";

elsif(A = 'A') then

B<="00100001001";

elsif(A = 'B') then

B<="10100001010";

elsif (A = 'C') then

B <= "01100000011";

elsif(A = 'D') then

B<="01100001100";

elsif (A = 'E' ) then

B<="10100000101";

elsif (A = 'F') then

B<= "00100000110";

elsif(A = 'G') then

B<="11100001111";

elsif(A = 'H') then

B<="00110010000";

elsif(A = 'I') then

B<="11110011001";

elsif(A = 'J') then

B<="01110011010" ;

elsif (A = 'K') then

B <= "10110010011";

elsif(A = 'L') then

B<="10110011100";

elsif (A = 'M' ) then

B<="01110010101";

elsif (A = 'N') then

B<= "11110010110";

elsif(A = 'O') then

B<="00110011111";

elsif(A = 'P') then

B<="10110100000";

elsif(A = 'Q') then

B<="01110101001";

elsif(A = 'R') then

B<="11110101010" ;

elsif (A = 'S') then

B <= "00110100011";

elsif(A = 'T') then

B<="00110101100";

elsif (A = 'U' ) then

B<="11110100101";

elsif (A = 'V') then

B<= "01110100110";

elsif(A = 'W') then

B<="10110101111";

elsif(A = 'X') then

B<="01100110000";

elsif(A = 'Y') then

B<="10100111001";

elsif(A = 'Z') then

B<="00100111010";

else

B<="00000000000" ;

end if;

end Asci\_to\_binary;

begin

--RX process

process (serial\_in)

begin

RX(serial\_in,RX\_vector);

check(RX\_vector,Rx\_vector);

Binary\_to\_Asci(RX\_vector,exchange);

completed<= false; --this step triger TX process

completed <= true; --this step reset completed signal

--we write this step here not in TX process because we can't change the same signal in two process, \*\*\*\*it will not effect the TX because TX process will be busy in delay functions\*\*\*\*

end process;

--TX process

process (completed)

begin

Asci\_to\_binary(exchange,TX\_vector);

TX(TX\_vector,serial\_out);

end process;

end Behavioral;