1. **Objective**: The objective of this project report is to design and implement a UART (Universal Asynchronous Receiver/Transmitter) interface using VHDL,Vivado,Basys-3 Board where the input is given in the basys-3 board in the ascii format and the character is visualized in the computer in the tera term terminal, and the received character's ascii code is visualized with the help of led's on board.

2. **Software hardware used:**

   Software apps:- Vivado,Tera Term,ModelSim
   Hardware components:- Digilent Basys 3

3. **Overview of the system:**
   This system implements a UART (Universal Asynchronous Receiver Transmitter) controller along with supporting modules such as UART transmitter and receiver, UART button debounce, and clock generators.

   1. UART Controller (UART_controller):
   - Acts as the central component integrating the UART transmitter and receiver.
   - Interfaces with external entities like the clock (`clk`), reset (`reset`), transmission enable (`tx_enable`), data input (`data_in`), data output (`data_out`), receive (`rx`), and transmit (`tx`) signals.

   2. UART Transmitter (UART_tx):
   - Generates serial data for transmission based on parallel input data (`tx_data_in`) and a start signal (`tx_start`).
   - Implements a finite state machine (`UART_tx_FSM`) to handle the transmission process, including start bit, data bits, and stop bit generation.

   3. UART Receiver (`UART_rx`):
   - Receives serial data and converts it into parallel data (`rx_data_out`).
   - Also utilizes a finite state machine (`UART_rx_FSM`) to synchronize with the incoming serial data stream.

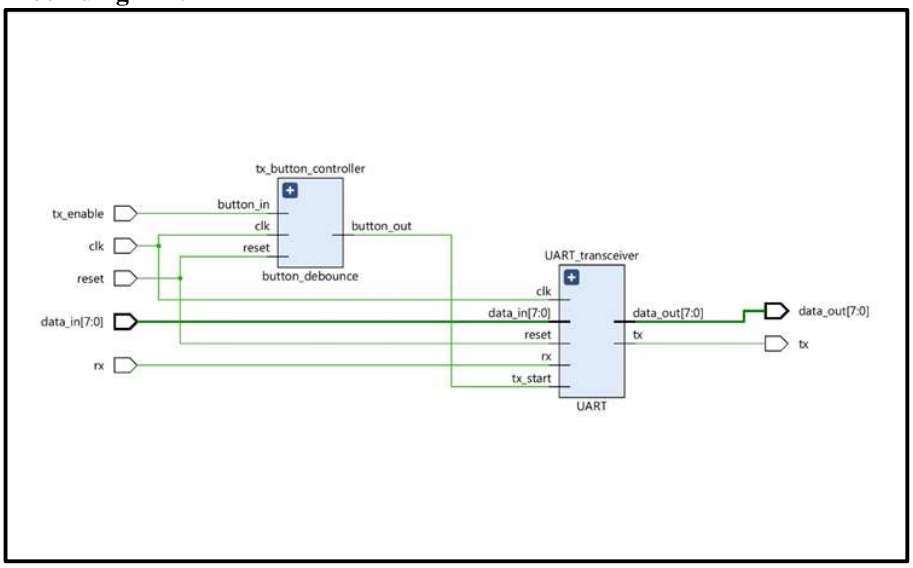
   4. UART Button Debounce (`button_debounce):
   - Debounces a button input (`button_in`) to prevent multiple signals due to mechanical noise or bouncing.
   - Generates a clean output signal (`button_out`) indicating a stable button press.

   5. Clock Generation:
   - Generates clocks (`clk`) for system synchronization and baud rate timing.
   - Implements clock dividers to derive baud rate clocks for UART transmitter and receiver.

   6. Testbench (`UART_controller_tb):
   - Provides a simulation environment to test the functionality of the UART controller and its associated modules.
   - Simulates clock generation, button presses, and data transmission/reception to verify the system's behavior.

Overall, the system enables communication between devices using UART protocol, ensuring reliable transmission and reception of data while handling debouncing for user input.

4. **Block diagram:**



5. **Introduction:** UART is a critical protocol for serial communication, widely used in embedded systems for data exchange between devices such as microcontrollers, computers, and peripheral devices. This design allows transmitting bits from the board to the computer terminal, and receiving bits from the terminal to the board. You can transmit data from the board to the terminal by pressing the push button on the board. Transmitting bits can be set by using the user switches, and receiving bits can be checked on the user LEDs.

6. **Complete Code with comments:**

--Transmitter Module

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

entity UART_tx is

   generic(
      BAUD_CLK_TICKS: integer := 868); -- clk/baud_rate (100 000 000 / 115 200 =
868.0555)
   port(
      clk           : in  std_logic;
      reset         : in  std_logic;
      tx_start      : in  std_logic;
      tx_data_in    : in  std_logic_vector (7 downto 0);
      tx_data_out   : out std_logic
      );
end UART_tx;

architecture Behavioral of UART_tx is
```

```
    type tx_states_t is (IDLE, START, DATA, STOP);
    signal tx_state  : tx_states_t := IDLE;
    signal baud_rate_clk     : std_logic:= '0';
    signal data_index        : integer range 0 to 7 := 0;
    signal data_index_reset  : std_logic := '1';
    signal stored_data       : std_logic_vector(7 downto 0) := (others=>'0');
    signal start_detected    : std_logic := '0';
    signal start_reset       : std_logic := '0';
begin
-- The baud_rate_clk_generator process generates the UART baud rate clock by
-- setting the baud_rate_clk signal when the counter counts BAUD_CLK_TICKS
-- ticks of the master clk. The BAUD_CLK_TICKS constant is specified in
-- the package and reflects the ratio between the master clk and the baud rate.

    baud_rate_clk_generator: process(clk)
    variable baud_count: integer range 0 to (BAUD_CLK_TICKS - 1) :=
(BAUD_CLK_TICKS - 1);
    begin
       if rising_edge(clk) then
          if (reset = '1') then
             baud_rate_clk <= '0';
             baud_count := (BAUD_CLK_TICKS - 1);
          else
             if (baud_count = 0) then
                baud_rate_clk <= '1';
                baud_count := (BAUD_CLK_TICKS - 1);
             else
                baud_rate_clk <= '0';
                baud_count := baud_count - 1;
             end if;
          end if;
       end if;
    end process baud_rate_clk_generator;


-- The tx_start_detector process works on the master clk frequency and catches
-- short (one clk cycle long) impulses in the tx_start signal and keeps it for
-- the UART_tx_FSM. tx_start_detector is needed because the UART_tx_FSM works on
-- the baud rate frequency, but the button_debounce module generates one master clk
-- cycle long impulse per one button push. start_detected keeps the information that
-- such event has occurred.
-- The second purpose of tx_start_detector is to secure the transmitting data.
-- stored_data keeps the transmitting data saved during the transmission.

    tx_start_detector: process(clk)
    begin
       if rising_edge(clk) then
          if (reset ='1') or (start_reset = '1') then
             start_detected <= '0';
          else
```

```
            if (tx_start = '1') and (start_detected = '0') then
               start_detected <= '1';
               stored_data <= tx_data_in;
            end if;
         end if;
      end if;
   end process tx_start_detector;

-- The data_index_counter process is a simple counter from 0 to 7 working on the baud
-- rate frequency. It is used to perform transformation between the parallel
-- data (stored_data) and the serial output (tx_data_out).
-- The data_index signal is used in UART_tx_FSM to go over the stored_data vector
-- and send the bits one by one.

   data_index_counter: process(clk)
   begin
      if rising_edge(clk) then
         if (reset = '1') or (data_index_reset = '1') then
            data_index <= 0;
         elsif (baud_rate_clk = '1') then
            data_index <= data_index + 1;
         end if;
      end if;
   end process data_index_counter;
-- The UART_FSM_tx process represents a Finite State Machine which has
-- four states (IDLE, START, DATA, STOP). See inline comments for more details.

   UART_tx_FSM: process(clk)
   begin
      if rising_edge(clk) then
         if (reset = '1') then
            tx_state <= IDLE;
            data_index_reset <= '1';   -- keep data_index_counter on hold
            start_reset <= '1';        -- keep tx_start_detector on hold
            tx_data_out <= '1';        -- keep tx line set along the standard
         else
            if (baud_rate_clk = '1') then   -- the FSM works on the baud rate frequency
               case tx_state is

                  when IDLE =>
                     data_index_reset <= '1';    -- keep data_index_counter on hold
                     start_reset <= '0';         -- enable tx_start_detector to wait for starting
impulses
                     tx_data_out <= '1';         -- keep tx line set along the standard

                     if (start_detected = '1') then
                        tx_state <= START;
                     end if;

                  when START =>
                     data_index_reset <= '0';   -- enable data_index_counter for DATA state
```

```
                    tx_data_out <= '0';        -- send '0' as a start bit

                    tx_state <= DATA;

                when DATA =>
                    tx_data_out <= stored_data(data_index);   -- send one bit per one baud clock
cycle 8 times

                    if (data_index = 7) then
                        data_index_reset <= '1';              -- disable data_index_counter when it has
reached 8
                        tx_state <= STOP;
                    end if;

                when STOP =>
                    tx_data_out <= '1';     -- send '1' as a stop bit
                    start_reset <= '1';     -- prepare tx_start_detector to be ready detecting the
next impuls in IDLE

                    tx_state <= IDLE;

                when others =>
                    tx_state <= IDLE;
            end case;
        end if;
    end if;
  end if;
end process UART_tx_FSM;
end Behavioral;
```

--Receiver Module

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity UART_rx is

  generic(
    BAUD_X16_CLK_TICKS: integer := 54); -- (clk / baud_rate) / 16 => (100 000 000 /
115 200) / 16 = 54.25
  port(
    clk           : in  std_logic;
    reset         : in  std_logic;
    rx_data_in    : in  std_logic;
    rx_data_out   : out std_logic_vector (7 downto 0)
    );
end UART_rx;

architecture Behavioral of UART_rx is

  type rx_states_t is (IDLE, START, DATA, STOP);
  signal rx_state: rx_states_t := IDLE;
```

```
   signal baud_rate_x16_clk  : std_logic := '0';
   signal rx_stored_data     : std_logic_vector(7 downto 0) := (others => '0');
begin
-- The baud_rate_x16_clk_generator process generates an oversampled clock.
-- The baud_rate_x16_clk signal is 16 times faster than the baud rate clock.
-- Oversampling is needed to put the capture point at the middle of duration of
-- the receiving bit.
-- The BAUD_X16_CLK_TICKS constant reflects the ratio between the master clk
-- and the x16 baud rate.
   baud_rate_x16_clk_generator: process(clk)
   variable baud_x16_count: integer range 0 to (BAUD_X16_CLK_TICKS - 1) :=
(BAUD_X16_CLK_TICKS - 1);
   begin
      if rising_edge(clk) then
         if (reset = '1') then
            baud_rate_x16_clk <= '0';
            baud_x16_count := (BAUD_X16_CLK_TICKS - 1);
         else
            if (baud_x16_count = 0) then
               baud_rate_x16_clk <= '1';
               baud_x16_count := (BAUD_X16_CLK_TICKS - 1);
            else
               baud_rate_x16_clk <= '0';
               baud_x16_count := baud_x16_count - 1;
            end if;
         end if;
      end if;
   end process baud_rate_x16_clk_generator;

-- The UART_rx_FSM process represents a Finite State Machine which has
-- four states (IDLE, START, DATA, STOP). See inline comments for more details.

   UART_rx_FSM: process(clk)
      variable bit_duration_count : integer range 0 to 15 := 0;
      variable bit_count          : integer range 0 to 7  := 0;
   begin
      if rising_edge(clk) then
         if (reset = '1') then
            rx_state <= IDLE;
            rx_stored_data <= (others => '0');
            rx_data_out <= (others => '0');
            bit_duration_count := 0;
            bit_count := 0;
         else
            if (baud_rate_x16_clk = '1') then     -- the FSM works 16 times faster the baud rate
frequency
               case rx_state is

                  when IDLE =>

                     rx_stored_data <= (others => '0');    -- clean the received data register
```

```
                    bit_duration_count := 0;             -- reset counters
                    bit_count := 0;

                    if (rx_data_in = '0') then           -- if the start bit received
                       rx_state <= START;                 -- transit to the START state
                    end if;

                 when START =>

                    if (rx_data_in = '0') then           -- verify that the start bit is preset
                       if (bit_duration_count = 7) then   -- wait a half of the baud rate cycle
                          rx_state <= DATA;              -- (it puts the capture point at the middle of
duration of the receiving bit)
                          bit_duration_count := 0;
                       else
                          bit_duration_count := bit_duration_count + 1;
                       end if;
                    else
                       rx_state <= IDLE;                 -- the start bit is not preset (false alarm)
                    end if;

                 when DATA =>

                    if (bit_duration_count = 15) then              -- wait for "one" baud rate cycle
(not strictly one, about one)
                       rx_stored_data(bit_count) <= rx_data_in;     -- fill in the receiving register
one received bit.
                       bit_duration_count := 0;
                       if (bit_count = 7) then                     -- when all 8 bit received, go to the
STOP state
                          rx_state <= STOP;
                          bit_duration_count := 0;
                       else
                          bit_count := bit_count + 1;
                       end if;
                    else
                       bit_duration_count := bit_duration_count + 1;
                    end if;

                 when STOP =>

                    if (bit_duration_count = 15) then      -- wait for "one" baud rate cycle
                       rx_data_out <= rx_stored_data;     -- transer the received data to the
outside world
                       rx_state <= IDLE;
                    else
                       bit_duration_count := bit_duration_count + 1;
                    end if;

                 when others =>
                    rx_state <= IDLE;
```

```
                end case;
            end if;
        end if;
    end if;
end process UART_rx_FSM;

end Behavioral;
--UART MODULE
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity UART is

    port(
        clk           : in  std_logic;
        reset         : in  std_logic;
        tx_start      : in  std_logic;

        data_in       : in  std_logic_vector (7 downto 0);
        data_out      : out std_logic_vector (7 downto 0);

        rx            : in  std_logic;
        tx            : out std_logic
        );
end UART;
architecture Behavioral of UART is
    component UART_tx
        port(
            clk           : in  std_logic;
            reset         : in  std_logic;
            tx_start      : in  std_logic;
            tx_data_in    : in  std_logic_vector (7 downto 0);
            tx_data_out   : out std_logic
            );
    end component;

    component UART_rx
        port(
            clk           : in  std_logic;
            reset         : in  std_logic;
            rx_data_in    : in  std_logic;
            rx_data_out   : out std_logic_vector (7 downto 0)
            );
    end component;
begin
    transmitter: UART_tx
    port map(
            clk           => clk,
            reset         => reset,
            tx_start      => tx_start,
            tx_data_in    => data_in,
```

```
        tx_data_out    => tx
        );
    receiver: UART_rx
    port map(
        clk            => clk,
        reset          => reset,
        rx_data_in     => rx,
        rx_data_out    => data_out
        );
end Behavioral;
```

-- Uart Button Debounce Module

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

entity button_debounce is
    generic (
        COUNTER_SIZE : integer := 10_000
        );
    port ( clk        : in  std_logic;
           reset      : in  std_logic;
           button_in  : in  std_logic;
           button_out : out std_logic);
end button_debounce;
architecture Behavioral of button_debounce is

    signal flipflop_1       : std_logic := '0';    -- output of flip-flop 1
    signal flipflop_2       : std_logic := '0';    -- output of flip-flop 2
    signal flipflop_3       : std_logic := '0';    -- output of flip-flop 3
    signal flipflop_4       : std_logic := '0';    -- output of flip-flop 4
    signal count_start      : std_logic := '0';

begin

-- The input_flipflops process creates two serial flip-flops (flip-flop 1 and
-- flip-flop 2). The signal from button_in passes them one by one. If flip_flop_1
-- and flip_flop_2 are different, it means the button has been activated, and
-- count_start becomes '1' for one master clock cycle.

    input_flipflops: process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then
                flipflop_1 <= '0';
                flipflop_2 <= '0';
            else
                flipflop_1 <= button_in;
                flipflop_2 <= flipflop_1;
            end if;
        end if;
```

```
    end process input_flipflops;
-- The count_start signal triggers the pause_counter process to start counting
    count_start <= flipflop_1 xor flipflop_2;
-- The pause_counter process passes the button_in signal farther from flip-flop 2
-- to flip-flop 3, but after COUNTER_SIZE master clock cycles. This allows
-- the button_in signal to stabilize in a certain state before being passed to the output.

    pause_counter: process(clk)
        variable count: integer range 0 to COUNTER_SIZE := 0;
    begin
        if rising_edge(clk) then
            if (reset = '1') then
                count := 0;
                flipflop_3 <= '0';
            else
                if (count_start = '1') then
                    count := 0;
                elsif (count < COUNTER_SIZE) then
                    count := count + 1;
                else
                    flipflop_3 <= flipflop_2;
                end if;
            end if;
        end if;
    end process pause_counter;


-- the purpose of the output_flipflop process is creating another flip-flop (flip-flop 4),
-- which creates a delay between the flipflop_3 and flipflop_4 signals. The delay is
-- one master clock cycle long.

    output_flipflop: process(clk)
    begin
        if rising_edge(clk) then
            if (reset = '1') then
                flipflop_4 <= '0';
            else
                flipflop_4 <= flipflop_3;
            end if;
        end if;
    end process output_flipflop;
-- The delay is needed to create one short (one master clock cycle long) impuls
-- at the button_out output. When pause_counter has finished, the flipflop_3 signal gets
-- the button_in information. At the moment flipflop_4 hasn't changed yet.
-- This creates '1' at the button_out output for one master clock cycle, only if
-- flipflop_3 is '1' (The button has been pressed, not released).

    with flipflop_3 select
    button_out <= flipflop_3 xor flipflop_4 when '1',
                  '0'                       when others;
end Behavioral;
```

```
--Uart Controller Module
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity UART_controller is
   port(
      clk            : in  std_logic;
      reset          : in  std_logic;
      tx_enable      : in  std_logic;
      data_in        : in  std_logic_vector (7 downto 0);
      data_out       : out std_logic_vector (7 downto 0);
      rx             : in  std_logic;
      tx             : out std_logic
      );
end UART_controller;


architecture Behavioral of UART_controller is
   component button_debounce
      port(
         clk       : in  std_logic;
         reset     : in  std_logic;
         button_in  : in  std_logic;
         button_out : out std_logic
         );
   end component;
   component UART
      port(
         clk           : in  std_logic;
         reset         : in  std_logic;
         tx_start      : in  std_logic;

         data_in       : in  std_logic_vector (7 downto 0);
         data_out      : out std_logic_vector (7 downto 0);

         rx            : in  std_logic;
         tx            : out std_logic
         );
   end component;

   signal button_pressed : std_logic;

begin

   tx_button_controller: button_debounce
   port map(
         clk           => clk,
         reset         => reset,
```

```
        button_in      => tx_enable,
        button_out     => button_pressed
        );

    UART_transceiver: UART
    port map(
        clk            => clk,
        reset          => reset,
        tx_start       => button_pressed,
        data_in        => data_in,
        data_out       => data_out,
        rx             => rx,
        tx             => tx
        );

end Behavioral;
```

--XDC file(configuring pins)

```
set_property PACKAGE_PIN V17 [get_ports {data_in[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[0]}]
set_property PACKAGE_PIN V16 [get_ports {data_in[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[1]}]
set_property PACKAGE_PIN W16 [get_ports {data_in[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[2]}]
set_property PACKAGE_PIN W17 [get_ports {data_in[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[3]}]
set_property PACKAGE_PIN W15 [get_ports {data_in[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[4]}]
set_property PACKAGE_PIN V15 [get_ports {data_in[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[5]}]
set_property PACKAGE_PIN W14 [get_ports {data_in[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[6]}]
set_property PACKAGE_PIN W13 [get_ports {data_in[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_in[7]}]
set_property PACKAGE_PIN U16 [get_ports {data_out[0]}]
set_property PACKAGE_PIN E19 [get_ports {data_out[1]}]
set_property PACKAGE_PIN U19 [get_ports {data_out[2]}]
set_property PACKAGE_PIN V19 [get_ports {data_out[3]}]
set_property PACKAGE_PIN W18 [get_ports {data_out[4]}]
set_property PACKAGE_PIN U15 [get_ports {data_out[5]}]
set_property PACKAGE_PIN U14 [get_ports {data_out[6]}]
set_property PACKAGE_PIN V14 [get_ports {data_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN A18 [get_ports tx]
```

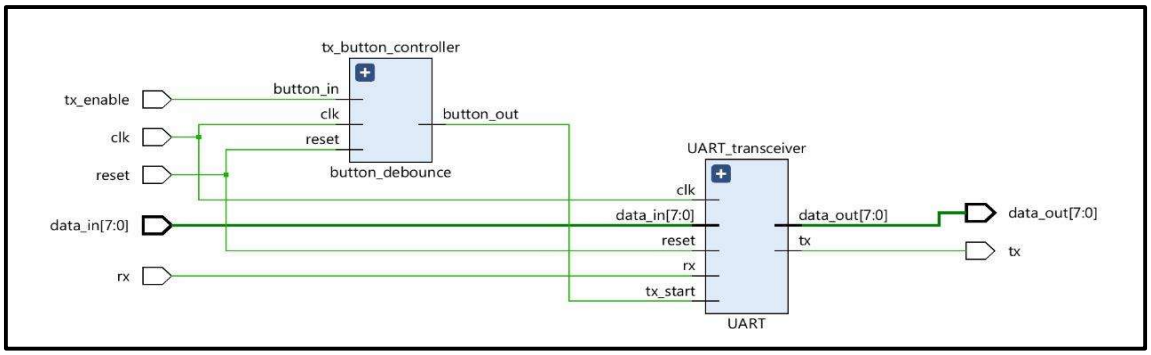
```
set_property IOSTANDARD LVCMOS33 [get_ports tx]
set_property PACKAGE_PIN B18 [get_ports rx]
set_property IOSTANDARD LVCMOS33 [get_ports rx]
set_property PACKAGE_PIN T18 [get_ports reset]
#set_property PACKAGE_PIN U18 [get_ports tx_start]
#set_property IOSTANDARD LVCMOS33 [get_ports tx_start]
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]


set_property PACKAGE_PIN U18 [get_ports tx_enable]
set_property IOSTANDARD LVCMOS33 [get_ports tx_enable]

create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports clk]
```

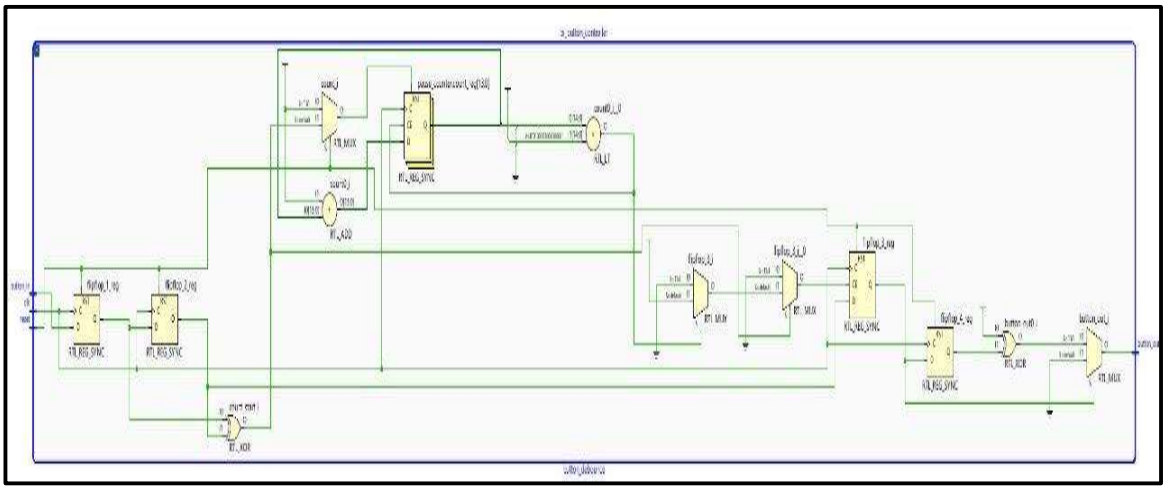
-- Uart test bench code for simulation

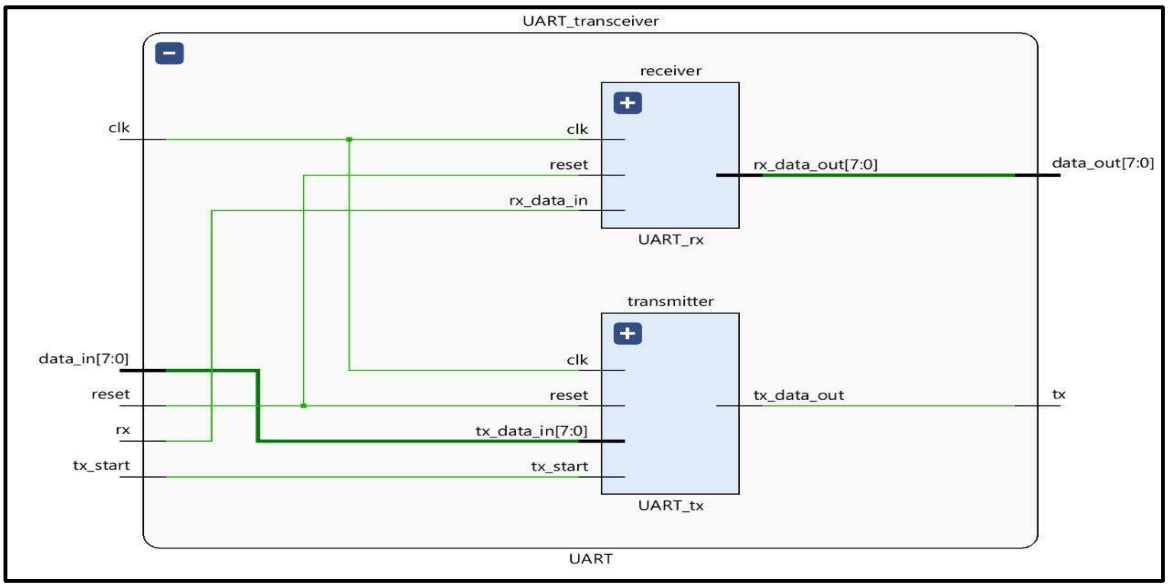
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UART_controller_tb is
--  Port ( );
end UART_controller_tb;

architecture Behavioral of UART_controller_tb is
   component UART_controller
      port(
         clk           : in  std_logic;
         reset         : in  std_logic;
         tx_enable     : in  std_logic;

         data_in       : in  std_logic_vector (7 downto 0);
         data_out      : out std_logic_vector (7 downto 0);

         rx            : in  std_logic;
         tx            : out std_logic
         );
   end component;

   signal clk           : std_logic := '0';
   signal reset         : std_logic := '0';
   signal tx_start      : std_logic := '0';

   signal tx_data_in    : std_logic_vector (7 downto 0) := (others => '0');
   signal rx_data_out   : std_logic_vector (7 downto 0) := (others => '0');
   signal tx_serial_out : std_logic := '1';
   signal rx_serial_in  : std_logic := '1';

   constant clock_cycle : time := 10 ns;    -- 1/clk (1 / 100 000 000)

begin
```

```
uut: UART_controller
port map(
        clk       => clk,
        reset     => reset,
        tx_enable => tx_start,

        data_in   => tx_data_in,
        data_out  => rx_data_out,

        tx        => tx_serial_out,
        rx        => rx_serial_in
        );

rx_serial_in <= tx_serial_out;
clk_generator: process
begin
    wait for clock_cycle/2;
        clk <= '1';
    wait for clock_cycle/2;
        clk <= '0';
end process clk_generator;


signals_generator: process
begin
    tx_data_in <= X"55";
    tx_start <= '0';

    wait for 25 us;
        tx_start <= '1';
    wait for 110 us;    -- test the button_debounce module
        tx_start <= '0';
    wait for 205 us;

    tx_data_in <= X"FF";

    wait for 28 us;
        tx_start <= '1';
    wait for 110 us;
        tx_start <= '0';
    wait for 200 us;

end process signals_generator;
         end Behavioral;
```

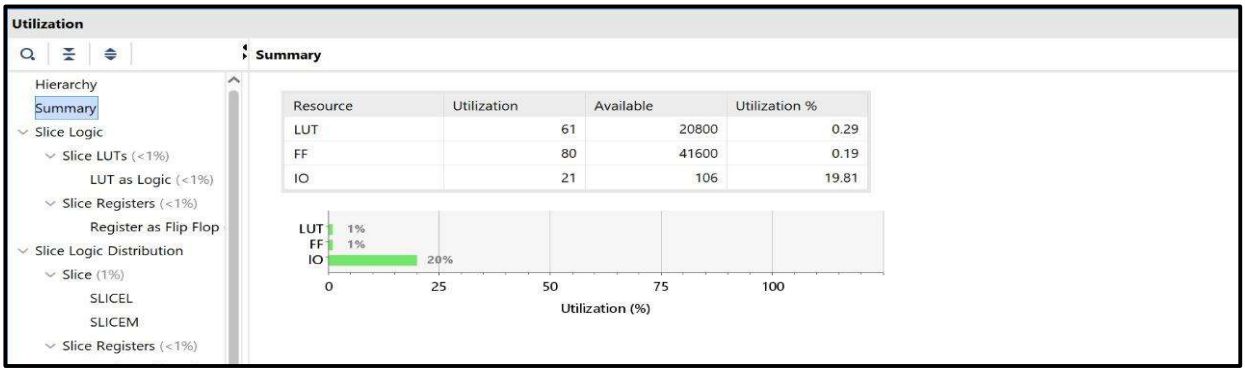
7. RTL:

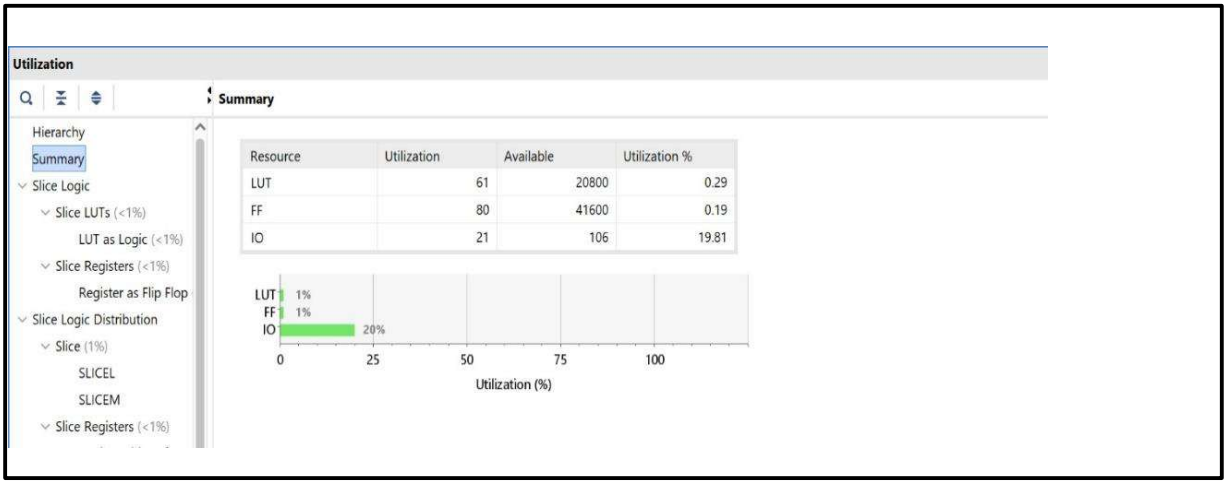

tx_button_controller
tx_enable
button_in
clk
button_out
clk
reset
reset
button_debounce
UART_transceiver
clk
data_in[7:0]
data_in[7:0]
data_out[7:0]
data_out[7:0]
reset
tx
tx
rx
rx
tx_start
UART



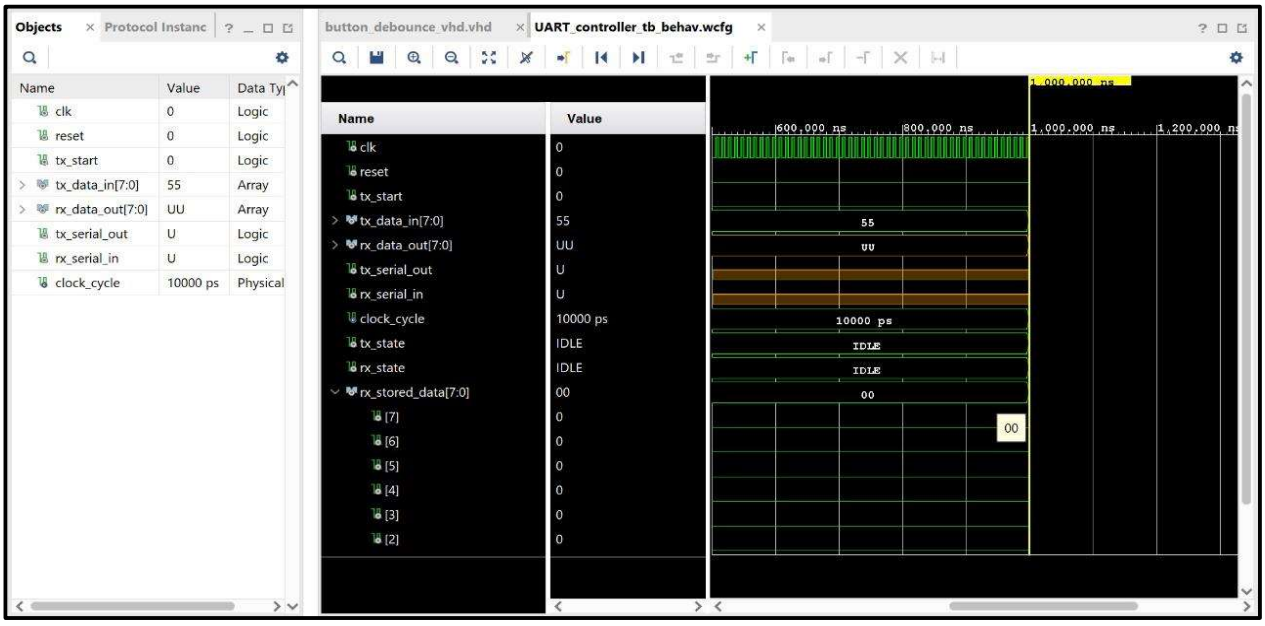

UART_transceiver
receiver
clk
clk
reset
rx_data_out[7:0]
data_out[7:0]
rx_data_in
UART_rx
transmitter
data_in[7:0]
clk
reset
reset
tx_data_out
tx
rx
tx_data_in[7:0]
tx_start
tx_start
UART_tx
UART

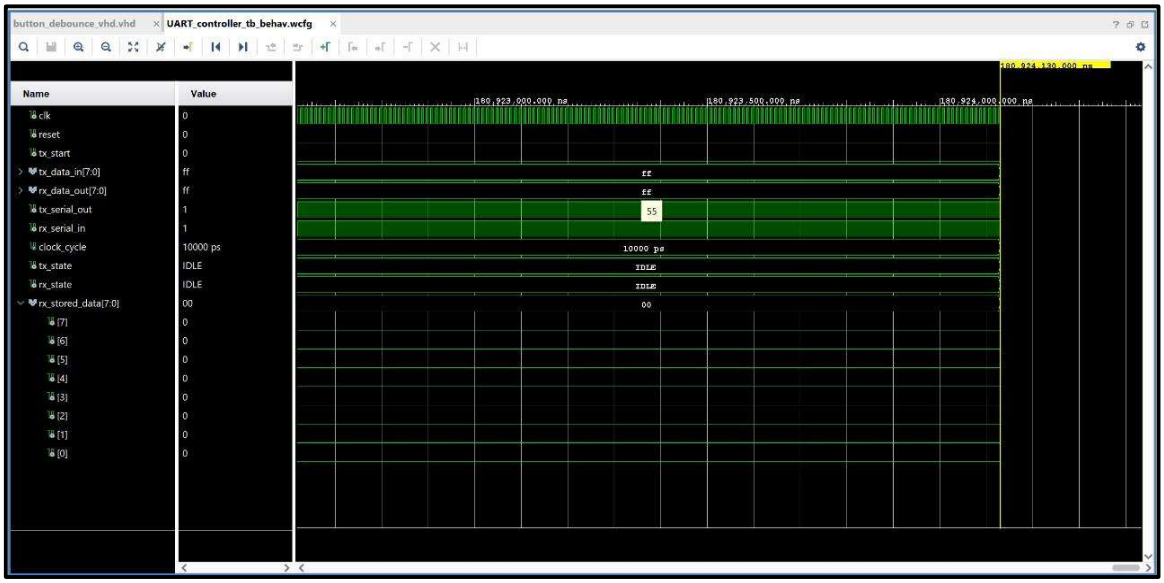
8. Resource Utilization summary table:

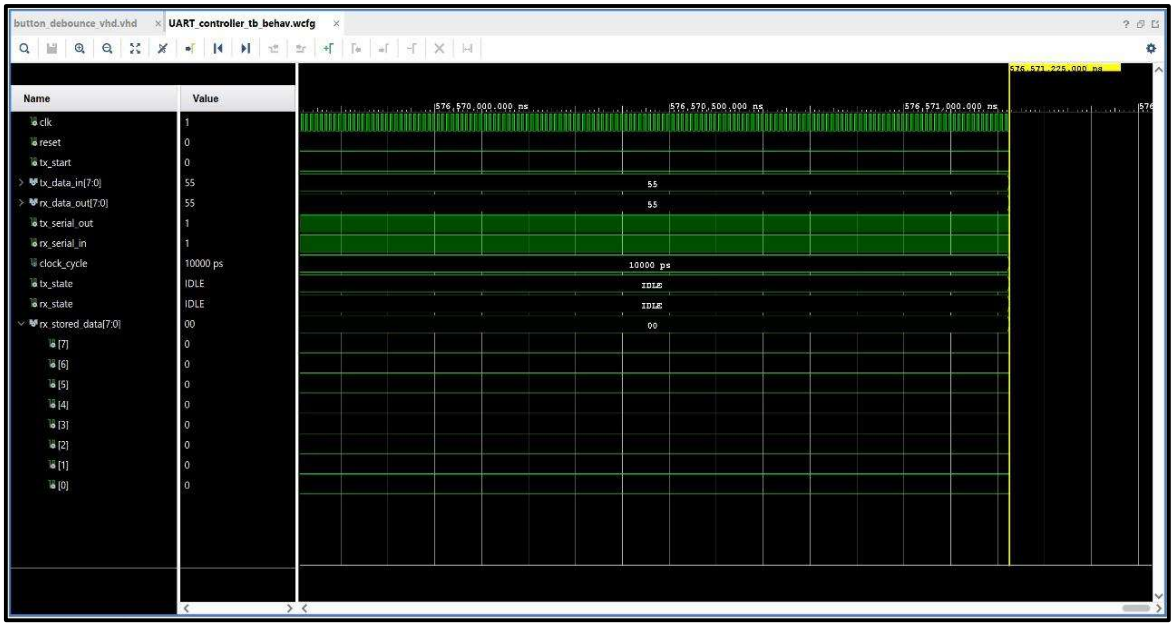




9. Output Waveform:

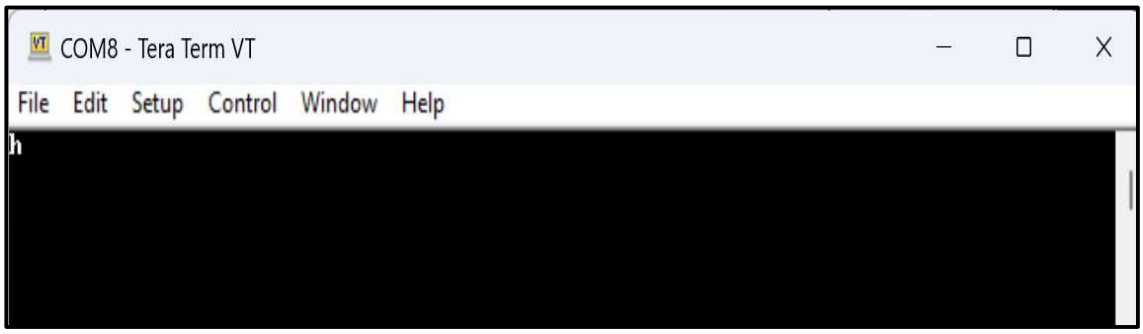


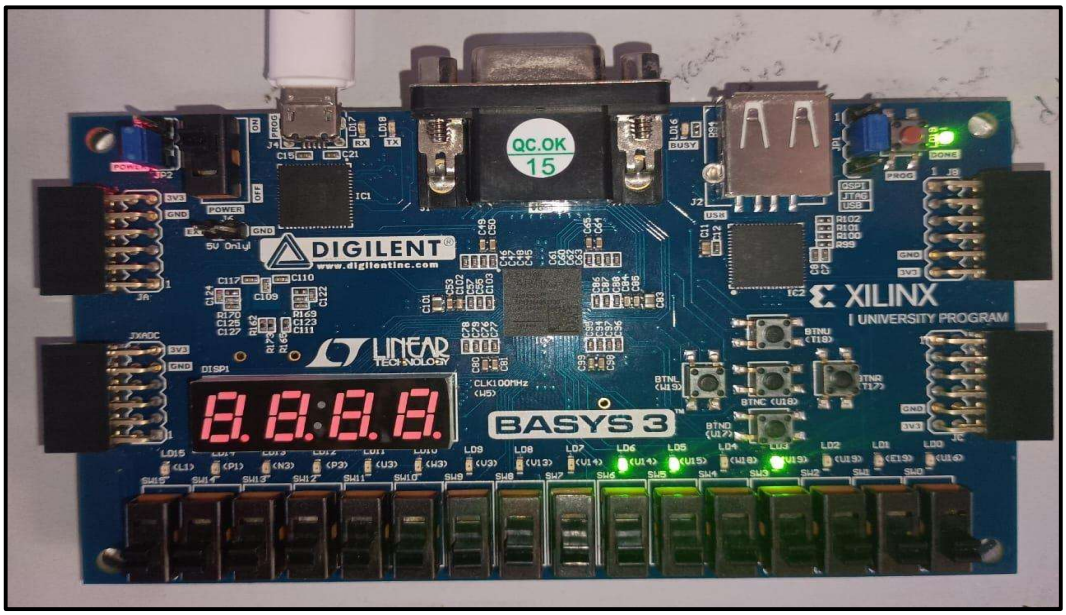




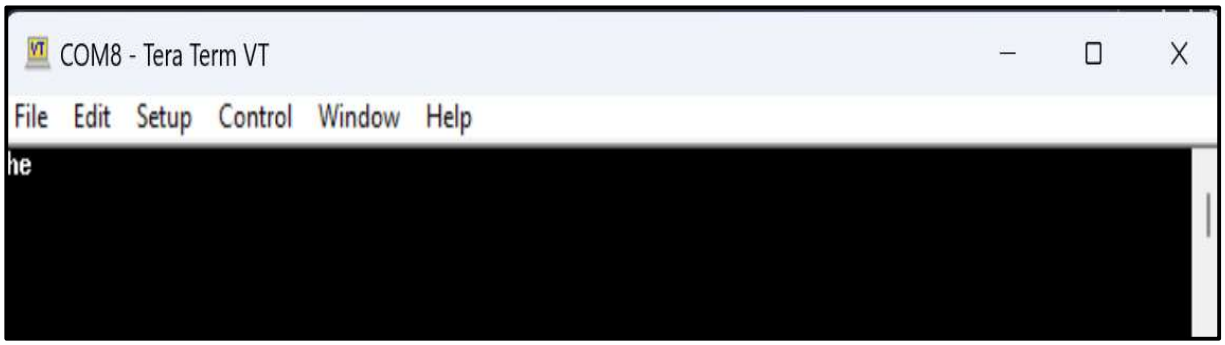
10. Hardware Images:

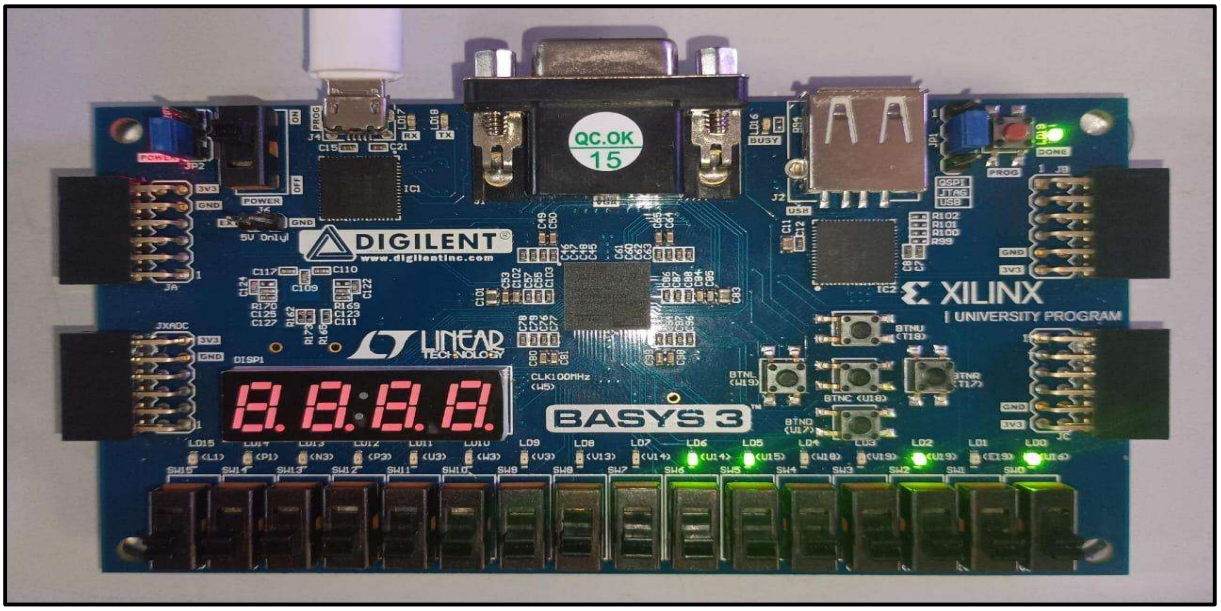
(i) Transmitting ‘h’ from Tera term and receiving ASCII of ‘h’-‘01101000’ on Basys3 board



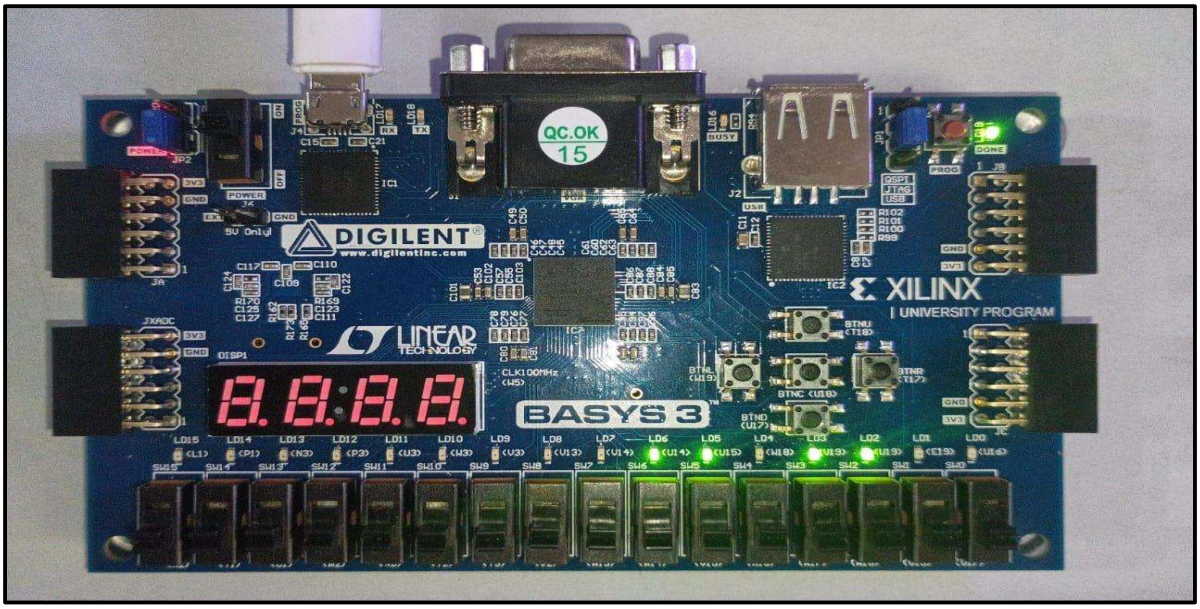


(ii) Transmitting ‘e’ from Tera term and receiving ASCII of ‘e’-‘01100101’ on Basys3 board

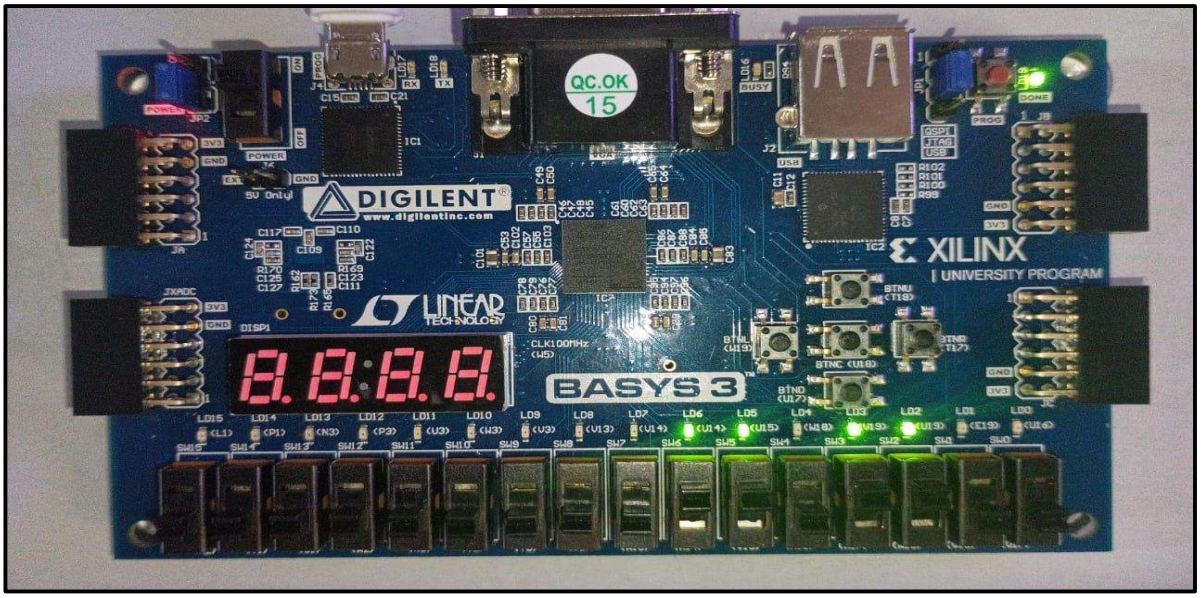




(iii) Transmitting 'l' from Tera term and receiving ASCII of 'l'-'01101100' on Basys3 board



(iv) Transmitting ASCII of 'l'-'01101100' from Basys3 board and pressing BTNC(U18) receives 'l' in Tera Term



(v) Transmitting ASCII of 'o'-'01101111' from Basys3 board and receiving 'o' in Tera Term