IT 451: Computer Organization and Architecture (Sessional)

Name – Anupam Sanidhya

Enrolment number – 510817049

Roll number – 42 (Hy)

Assignment 4

1.) Design and simulate a 5-bit bidirectional shift register. Use 'DIRECTION' as a control input as STD_LOGIC with its value '1' for right shift and '0' for left shift.

a. Synthesize it for Spartan 6. Report device utilization.

b. Create appropriate test benches to reflect its functional behaviour for all possible cases.

Include the snapshots in your report.

VHDL MODULE:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity Register_Module is

   Port ( Cur_Val : inout  STD_LOGIC_VECTOR (4 downto 0);

                          In_Bit : in STD_LOGIC;

                          Clock : in STD_LOGIC;

        Direction : in  STD_LOGIC);

end Register_Module;


architecture Behavioral of Register_Module is

signal temp : STD_LOGIC_VECTOR(4 downto 0);

begin

        process(Cur_Val,Direction,temp,In_Bit,Clock)

        begin

                if(rising_edge(Clock))

                then

                        temp <= Cur_Val;

                                case Direction is

                                when '0' =>

                                        temp(1) <= temp(0);

                                        temp(2) <= temp(1);

                                        temp(3) <= temp(2);

                                        temp(4) <= temp(3);

                                        temp(0) <= In_Bit;

                                when '1' =>

                                        temp(0) <= temp(1);

                                        temp(1) <= temp(2);

                                        temp(2) <= temp(3);
```

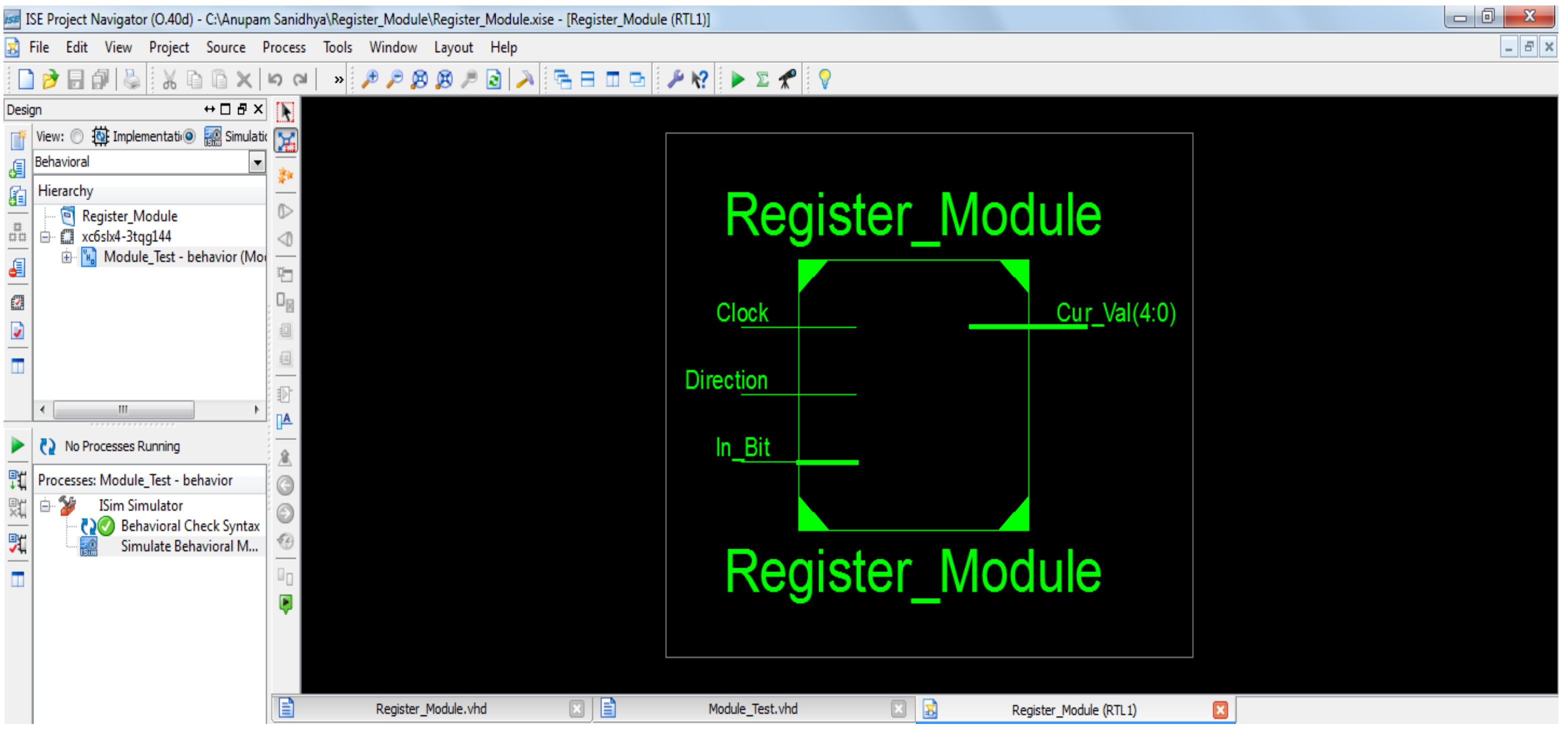
```
                    temp(3) <= temp(4);
                    temp(4) <= In_Bit;
                when others =>
                    temp <= "00000";
            end case;
        end if;
    end process;
    Cur_Val <= temp;
end Behavioral;
```

TEST BENCH :

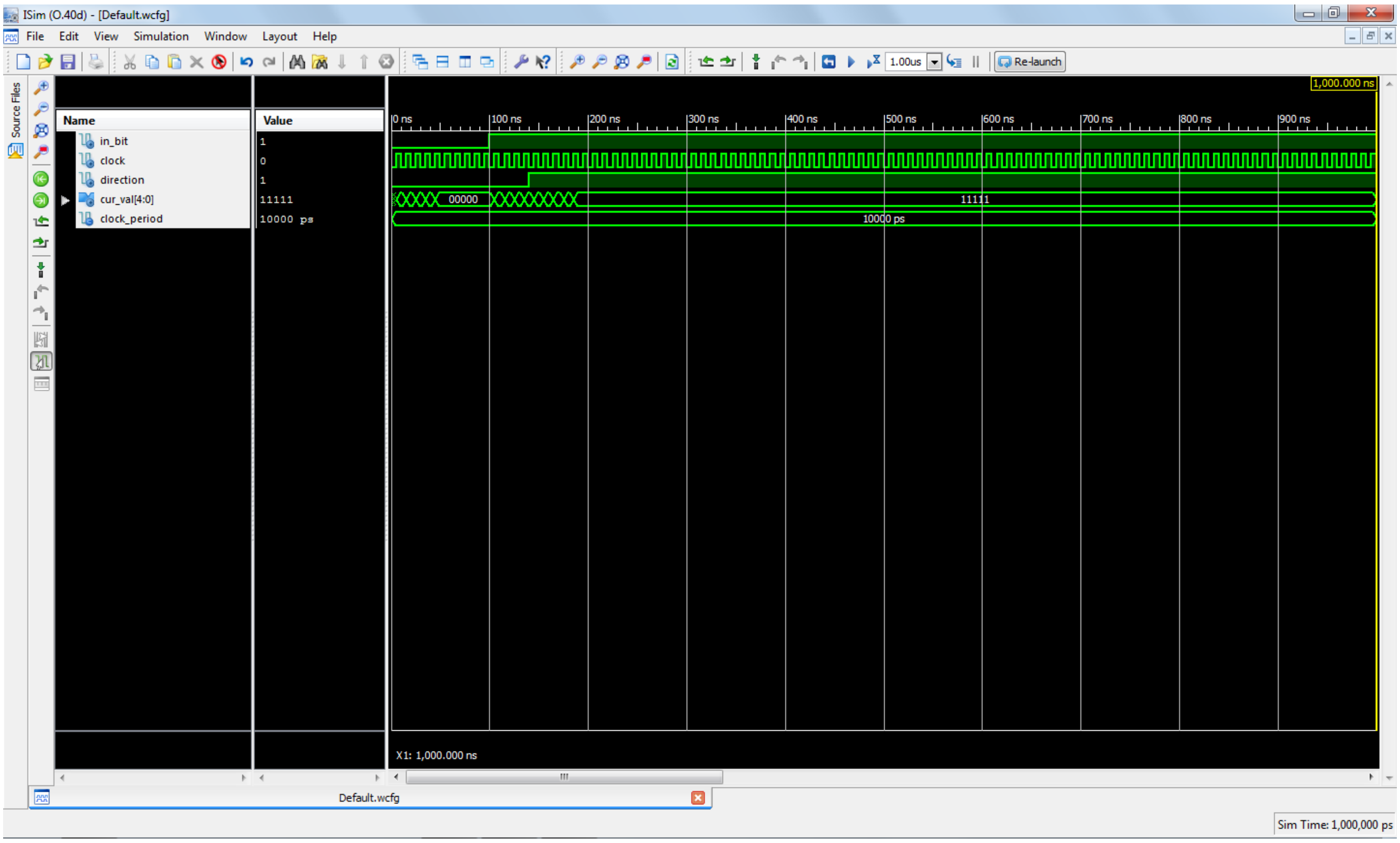
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Module_Test IS
END Module_Test;
ARCHITECTURE behavior OF Module_Test IS
   COMPONENT Register_Module
   PORT(
       Cur_Val : INOUT  std_logic_vector(4 downto 0);
       In_Bit : IN  std_logic;
       Clock : IN  std_logic;
       Direction : IN  std_logic
      );
   END COMPONENT;
  signal In_Bit : std_logic := '0';
```

```
   signal Clock : std_logic := '0';
   signal Direction : std_logic := '0';
   signal Cur_Val : std_logic_vector(4 downto 0) := "11111";
   constant Clock_period : time := 10 ns;
BEGIN
   uut: Register_Module PORT MAP (
          Cur_Val => Cur_Val,
          In_Bit => In_Bit,
          Clock => Clock,
          Direction => Direction
        );
   Clock_process :process
   begin
                Clock <= '0';
                wait for Clock_period/2;
                Clock <= '1';
                wait for Clock_period/2;
   end process;
   stim_proc: process
   begin
      wait for 100 ns;
                In_Bit <= '1';
                wait for 40 ns;
                Direction <= '1';
      wait;
   end process;
END;
```

RTL Schematic –



Simulation –



Name	Value
in_bit	1
clock	0
direction	1
cur_val[4:0]	11111
clock_period	10000 ps

2.) Design and simulate a UART transmitter.

a. Synthesize it for Spartan 6. Report device utilization.

b. Create appropriate test benches to reflect its functional behaviour for all possible cases. Include the snapshots in your report.

VHDL MODULE:

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

entity UART_TX is
  generic (
    g_CLKS_PER_BIT : integer := 115     -- Needs to be set correctly
    );
  port (
    i_Clk       : in  std_logic;
    i_TX_DV     : in  std_logic;
    i_TX_Byte   : in  std_logic_vector(7 downto 0);
    o_TX_Active : out std_logic;
    o_TX_Serial : out std_logic;
    o_TX_Done   : out std_logic
    );
end UART_TX;


architecture RTL of UART_TX is

  type t_SM_Main is (s_Idle, s_TX_Start_Bit, s_TX_Data_Bits,
                     s_TX_Stop_Bit, s_Cleanup);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal r_TX_Data   : std_logic_vector(7 downto 0) := (others => '0');
  signal r_TX_Done   : std_logic := '0';

begin
```

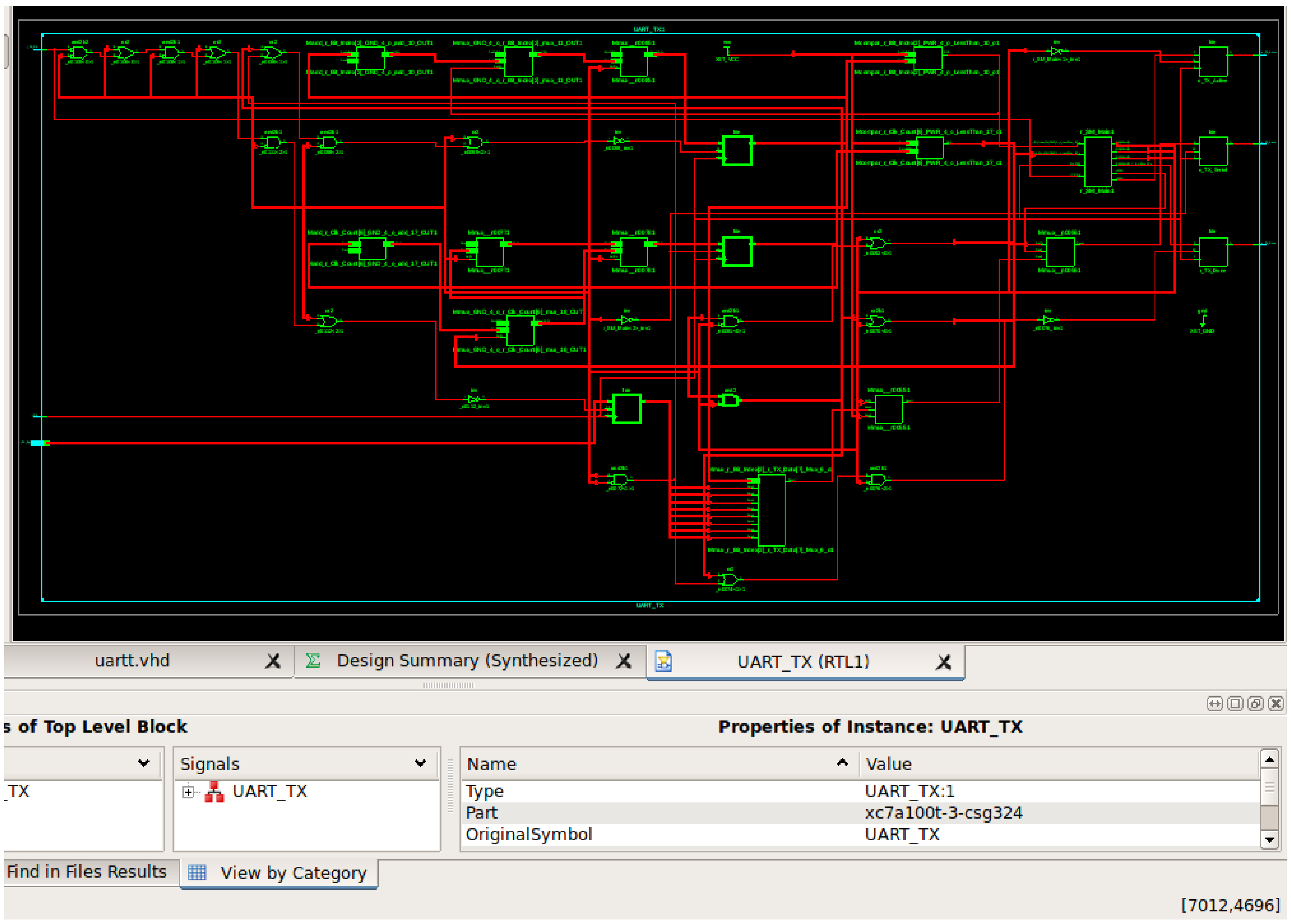
```
p_UART_TX : process (i_Clk)
begin
  if rising_edge(i_Clk) then

    case r_SM_Main is

      when s_Idle =>
        o_TX_Active <= '0';
        o_TX_Serial <= '1';         -- Drive Line High for Idle
        r_TX_Done   <= '0';
        r_Clk_Count <= 0;
        r_Bit_Index <= 0;

        if i_TX_DV = '1' then
          r_TX_Data <= i_TX_Byte;
          r_SM_Main <= s_TX_Start_Bit;
        else
          r_SM_Main <= s_Idle;
        end if;


      -- Send out Start Bit. Start bit = 0
      when s_TX_Start_Bit =>
        o_TX_Active <= '1';
        o_TX_Serial <= '0';

        -- Wait g_CLKS_PER_BIT-1 clock cycles for start bit to finish
        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_TX_Start_Bit;
        else
          r_Clk_Count <= 0;
          r_SM_Main   <= s_TX_Data_Bits;
        end if;


      -- Wait g_CLKS_PER_BIT-1 clock cycles for data bits to finish
      when s_TX_Data_Bits =>
        o_TX_Serial <= r_TX_Data(r_Bit_Index);

        if r_Clk_Count < g_CLKS_PER_BIT-1 then
          r_Clk_Count <= r_Clk_Count + 1;
          r_SM_Main   <= s_TX_Data_Bits;
        else
          r_Clk_Count <= 0;

          -- Check if we have sent out all bits
          if r_Bit_Index < 7 then
            r_Bit_Index <= r_Bit_Index + 1;
            r_SM_Main   <= s_TX_Data_Bits;
          else
            r_Bit_Index <= 0;
            r_SM_Main   <= s_TX_Stop_Bit;
          end if;
          end if;


        -- Send out Stop bit.  Stop bit = 1
        when s_TX_Stop_Bit =>
          o_TX_Serial <= '1';

          -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
          if r_Clk_Count < g_CLKS_PER_BIT-1 then
            r_Clk_Count <= r_Clk_Count + 1;
            r_SM_Main   <= s_TX_Stop_Bit;
          else
            r_TX_Done   <= '1';
            r_Clk_Count <= 0;
            r_SM_Main   <= s_Cleanup;
          end if;


        -- Stay here 1 clock
        when s_Cleanup =>
          o_TX_Active <= '0';
          r_TX_Done   <= '1';
          r_SM_Main   <= s_Idle;


        when others =>
          r_SM_Main <= s_Idle;

      end case;
    end if;
  end process p_UART_TX;

  o_TX_Done <= r_TX_Done;

end RTL;
```

TEST BENCH :

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

entity uart_tb is
end uart_tb;

architecture behave of uart_tb is

  component uart_tx is
    generic (
      g_CLKS_PER_BIT : integer := 115   -- Needs to be set correctly
      );
  port (
    i_clk       : in  std_logic;
    i_tx_dv     : in  std_logic;
    i_tx_byte   : in  std_logic_vector(7 downto 0);
    o_tx_active : out std_logic;
    o_tx_serial : out std_logic;
    o_tx_done   : out std_logic
    );
end component uart_tx;

component uart_rx is
  generic (
    g_CLKS_PER_BIT : integer := 115   -- Needs to be set correctly
    );
  port (
    i_clk       : in  std_logic;
    i_rx_serial : in  std_logic;
    o_rx_dv     : out std_logic;
    o_rx_byte   : out std_logic_vector(7 downto 0)
    );
end component uart_rx;


-- Test Bench uses a 10 MHz Clock
-- Want to interface to 115200 baud UART
-- 10000000 / 115200 = 87 Clocks Per Bit.
constant c_CLKS_PER_BIT : integer := 87;

constant c_BIT_PERIOD : time := 8680 ns;

signal r_CLOCK     : std_logic                    := '0';
signal r_TX_DV     : std_logic                    := '0';
signal r_TX_BYTE   : std_logic_vector(7 downto 0) := (others => '0');
signal w_TX_SERIAL : std_logic;
signal w_TX_DONE   : std_logic;
signal w_RX_DV     : std_logic;
signal w_RX_BYTE   : std_logic_vector(7 downto 0);
signal r_RX_SERIAL : std_logic := '1';


-- Low-level byte-write
procedure UART_WRITE_BYTE (
  i_data_in       : in  std_logic_vector(7 downto 0);
  signal o_serial : out std_logic) is
begin

  -- Send Start Bit
  o_serial <= '0';
  wait for c_BIT_PERIOD;

  -- Send Data Byte
  for ii in 0 to 7 loop
    o_serial <= i_data_in(ii);
    wait for c_BIT_PERIOD;
  end loop;  -- ii
```

```
    -- Send Stop Bit
    o_serial <= '1';
    wait for c_BIT_PERIOD;
  end UART_WRITE_BYTE;


begin

  -- Instantiate UART transmitter
  UART_TX_INST : uart_tx
    generic map (
      g_CLKS_PER_BIT => c_CLKS_PER_BIT
      )
    port map (
      i_clk       => r_CLOCK,
      i_tx_dv     => r_TX_DV,
      i_tx_byte   => r_TX_BYTE,
      o_tx_active => open,
      o_tx_serial => w_TX_SERIAL,
      o_tx_done   => w_TX_DONE
      );

  -- Instantiate UART Receiver
  UART_RX_INST : uart_rx
    generic map (
      g_CLKS_PER_BIT => c_CLKS_PER_BIT
      )
    port map (
      i_clk       => r_CLOCK,
      i_rx_serial => r_RX_SERIAL,
      o_rx_dv     => w_RX_DV,
      o_rx_byte   => w_RX_BYTE
      );

  r_CLOCK <= not r_CLOCK after 50 ns;

  process is
  begin

    -- Tell the UART to send a command.
    wait until rising_edge(r_CLOCK);
    wait until rising_edge(r_CLOCK);
    r_TX_DV   <= '1';
    r_TX_BYTE <= X"AB";
    wait until rising_edge(r_CLOCK);
    r_TX_DV   <= '0';
    wait until w_TX_DONE = '1';


    -- Send a command to the UART
    wait until rising_edge(r_CLOCK);
    UART_WRITE_BYTE(X"3F", r_RX_SERIAL);
    wait until rising_edge(r_CLOCK);
    -- Check that the correct command was received
    if w_RX_BYTE = X"3F" then
      report "Test Passed - Correct Byte Received" severity note;
    else
      report "Test Failed - Incorrect Byte Received" severity note;
    end if;

    assert false report "Tests Complete" severity failure;

  end process;

end behave;
```

RTL Schematic –



uartt.vhd | Design Summary (Synthesized) | UART_TX (RTL1)

s of Top Level Block

Properties of Instance: UART_TX

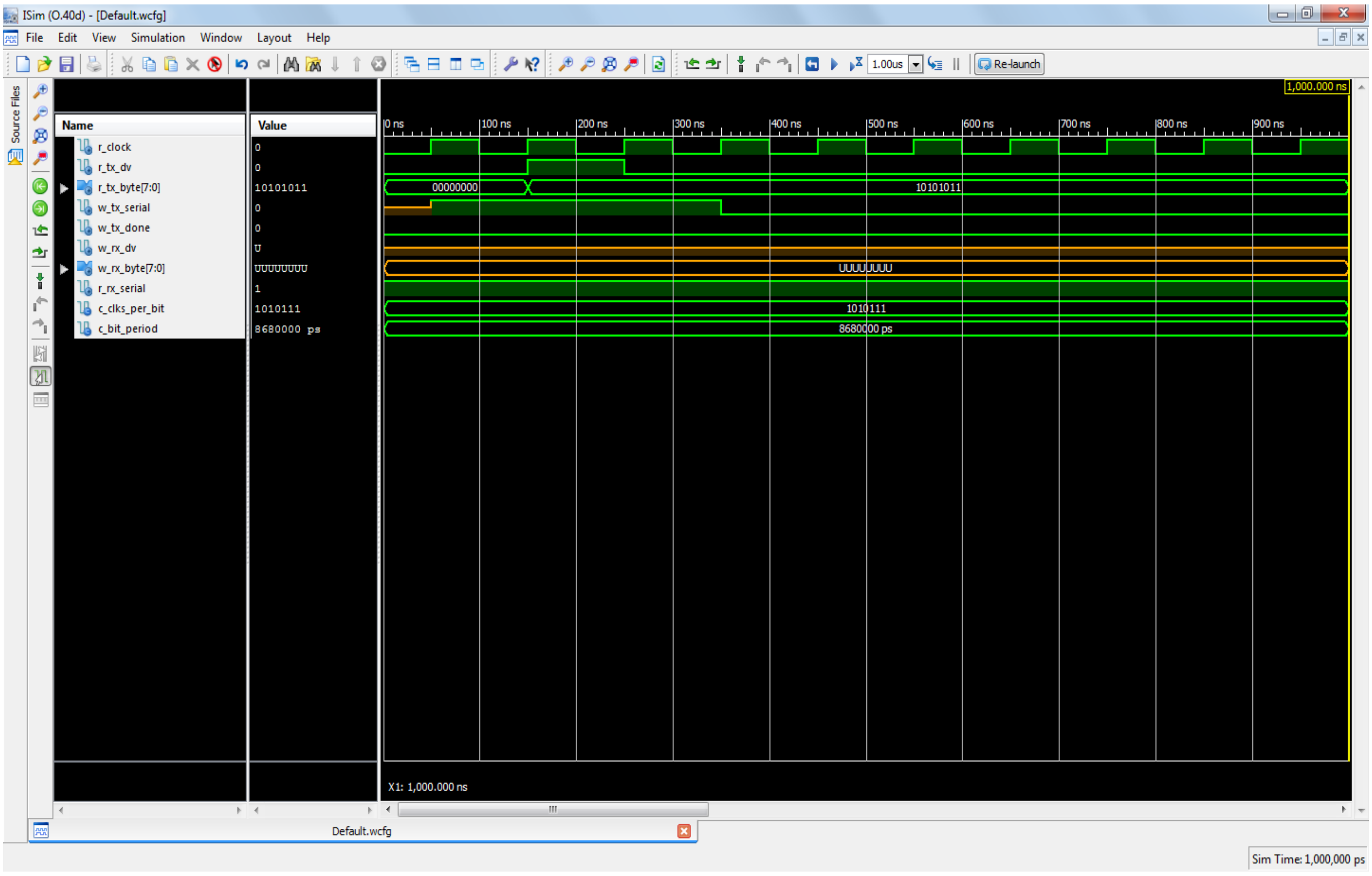
	Signals
_TX	UART_TX

Name	Value
Type	UART_TX:1
Part	xc7a100t-3-csg324
OriginalSymbol	UART_TX

Find in Files Results | View by Category

[7012,4696]

Simulation –



Name	Value
r_clock	0
r_tx_dv	0
r_tx_byte[7:0]	10101011
w_tx_serial	0
w_tx_done	0
w_rx_dv	U
w_rx_byte[7:0]	UUUUUUUU
r_rx_serial	1
c_clks_per_bit	1010111
c_bit_period	8680000 ps