Introduction to VHDL
ECE 5146/4146

Project
UART Design

Group 2

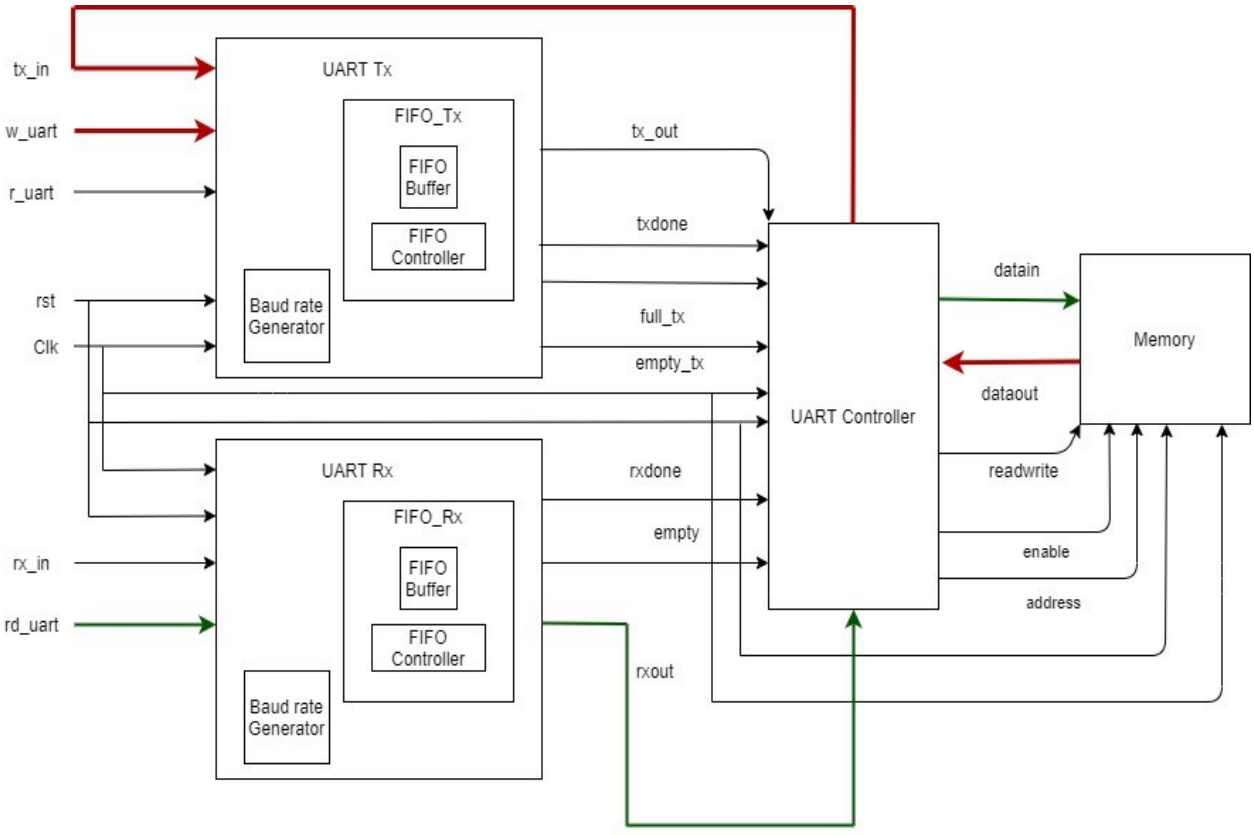
Dhanvin Athray – 801167002

Geraldine Shirley Nicholas – 801104352

Summary

Universal Asynchronous Receiver-Transmitter(UART) is a computer hardware device that is used for asynchronous serial communication and acts as an intermediary between parallel and serial interfaces. It consists of a Transmitter and Receiver unit where on the Transmitter side it creates the data packet, appending sync and parity bits, and send that packet out the Tx line with precise timing according to the set baud rate and on the Receiver end, the UART samples the Rx line at rates according to the expected baud rate, collect the sync bits, and displays the data. In the transmitter subsystem each byte starts with a start sequence followed by data and a stop sequence and parity bits are optional. All these are transmitted using a shift register clocked at the nominal data rate. The baud rate generator generates a sampling signal whose frequency is exactly 16 times the UART's designated baud rate. FIFO interface circuit is integrated with the UART subsystem to hold the data. BRAM is implemented where data from UART FIFO buffer Rx is sent to the RAM array on Mode 1 and data from RAM array is sent to FIFO buffer Tx on Mode 3. The overall system is shown below where it consists of:

1. UART Tx with Baud rate generator and FIFO Interface
2. UART Rx with Baud rate generator and FIFO Interface
3. UART Controller
4. BRAM



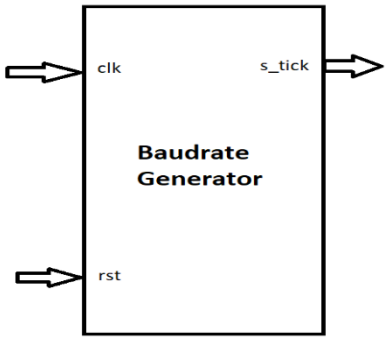
Module 1:

UART Subsystem:

The UART Subsystem consist of the Baud rate generator with UART Tx and UART Rx.

i) Baud Rate Generator:

Baud rate generator is a clock generator which is used to determine the rate at which data is transmitted by a UART. It takes a relatively high frequency and dividing it down to lower frequency. Common baud rates are 2400, 4800, 9600 and 19,200.A mod-163 counter is used to implement baud generator. The tick output will assert for one clock cycle every 163 clock cycles of the system clock.



VHDL Code:

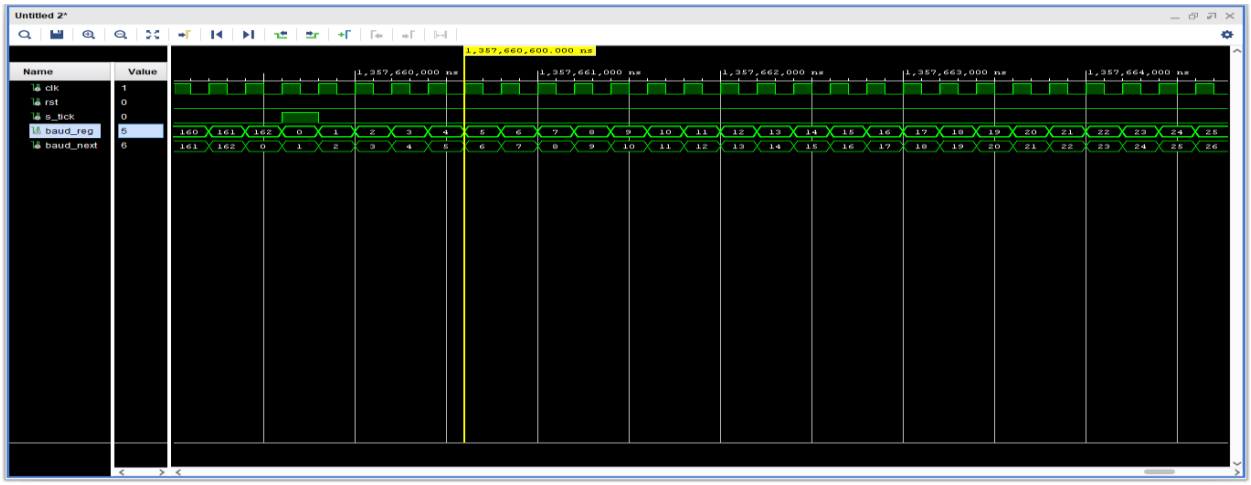
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.NUMERIC_STD.ALL;
entity Baudgen is
port(
    clk,rst: in std_logic;
    s_tick: out std_logic);
end Baudgen;
architecture Behavioral of Baudgen is
signal baud_reg,baud_next : integer range 0 to 162;
begin
   --process block
   process(clk,rst)
   begin
      if(rst = '1') then
         baud_reg <= 0;
      elsif(rising_edge(clk)) then
         baud_reg <= baud_next;
      end if;
   end process;
--next state logic
   baud_next <= 0 when baud_reg = 162 else
```

```
            baud_reg + 1;
--output logic
   s_tick <= '1' when baud_reg = 0 else
          '0';
end Behavioral;
```

Testbench:

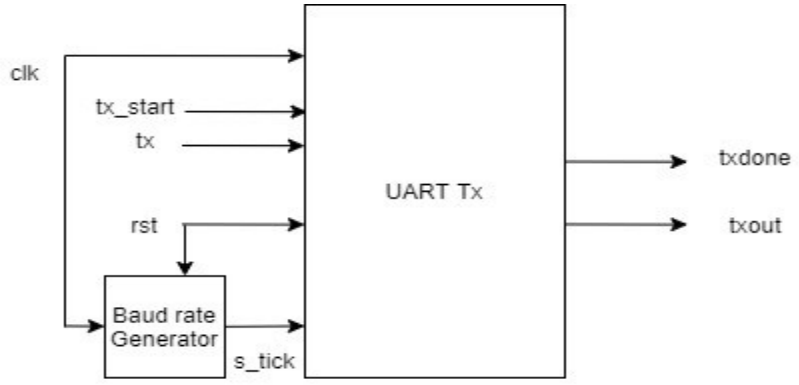
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Baudgen_Sim is
--  Port ( );
end Baudgen_Sim;
architecture Behavioral of Baudgen_Sim is
signal clk,rst,s_tick : std_logic;
begin
p0: entity work.Baudgen(Behavioral) port map(clk => clk, rst => rst, s_tick => s_tick);
clock:process
 begin
   clk <= '1';
          wait for 100 ns;
          clk <= '0';
          wait for 100 ns;
 end process;
 stimulus : process
 begin
   rst <= '1';
          wait for 200 ns;
          rst <= '0';
          wait;
          end process;
end Behavioral;
```

Wave Form:



ii) UART Tx

The UART Tx integrated with baud rate generator.



VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity UART_Tx is
--  Port ( );
port(
    clk,rst,tx_start       : in std_logic;
    tx                  : in std_logic_vector(7 downto 0);
    txdone,txout           : out std_logic);
end UART_Tx;

architecture Behavioral of UART_Tx is
--signal
type state_type is (idle, start, data, stop);   -- states
signal state_reg, state_next : state_type;
signal s_reg, s_next  : integer range 0 to 15;   -- ticks = 16
signal n_reg, n_next  : integer range 0 to 7;    -- data bits count = 8
signal b_reg, b_next  : std_logic_vector( 7 downto 0);    -- data byte buffer containing 8 bits of data
(input)
signal tx_reg, tx_next: std_logic;
signal s_tick        : std_logic;
signal txdonesig      : std_logic;
--component Baud Generator
component Baudgen is
--  Port ( );
port(
    clk,rst: in std_logic;
    s_tick: out std_logic);
end component;
```

```
begin
B: Baudgen port map (clk => clk, rst => rst, s_tick => s_tick);
--process FSMD state and data registers
   process(clk,rst)
   begin
      if(rst = '1') then
         state_reg <= idle;
         s_reg     <= 0;
         n_reg     <= 0;
         b_reg     <= (others => '0');

      elsif(rising_edge(clk)) then
         state_reg <= state_next;
         s_reg     <= s_next;
         n_reg     <= n_next;
         b_reg     <= b_next;
         tx_reg    <= tx_next;

      end if;
   end process;
   --next state logic
   process(state_reg, s_reg, n_reg, b_reg, s_tick, tx_reg, tx_start,tx)
   begin
   state_next <= state_reg;
   s_next <= s_reg;
   n_next <= n_reg;
   b_next <= b_reg;
   tx_next<= tx_reg;
   txdonesig <= '0';

   case state_reg is
     -- idle state
      when idle =>
         tx_next   <= '1';

         if(tx_start = '1') then
            state_next <= start;
            s_next     <= 0;
            b_next     <= tx;
         else
            state_next <= idle;
         end if;
      --start state
      when start =>
         tx_next   <= '0';

         if(s_tick = '0') then
            state_next <= start;
```

```
    else
      if(s_reg = 15) then
        state_next <= data;
        s_next     <= 0;
        n_next     <= 0;
      else
        state_next <= start;
        s_next     <= s_reg + 1;
      end if;
    end if;
-- data state
when data =>
    tx_next <= b_reg(0);

    if(s_tick = '0') then
      state_next <= data;
    else
      if(s_reg = 15) then
        s_next <= 0;
        b_next <= '0' & b_reg(7 downto 1);
        if(n_reg = 7) then
          state_next <= stop;
          n_next     <= 0;
        else
          state_next <= data;
          n_next     <= n_reg + 1;
        end if;
      else
        s_next <= s_reg + 1;
      end if;
    end if;
-- stop state
when stop =>
    tx_next     <= '1';

    if(s_tick = '0') then
      state_next <= stop;
    else
      if(s_reg = 15) then
        state_next <= idle;
        s_next     <= 0;
        txdonesig     <= '1';   -- Tx done bit

      else
        state_next <= stop;
        s_next     <= s_reg + 1;
        end if;
      end if;
```

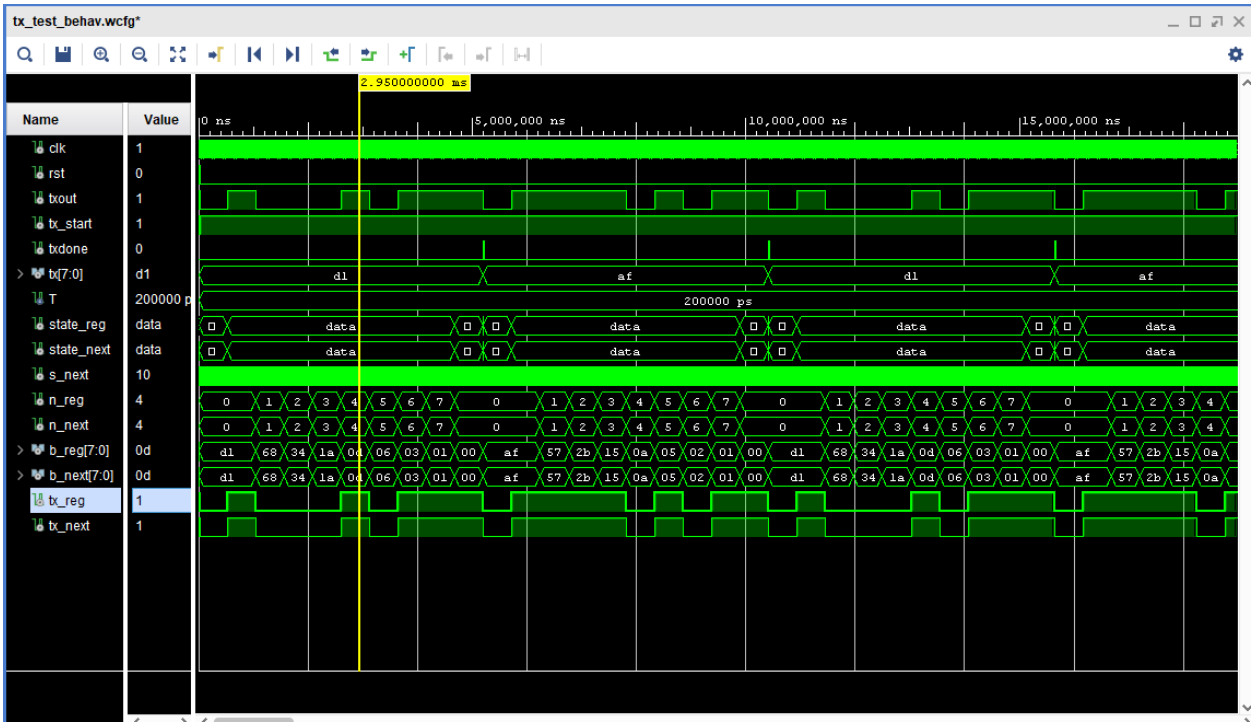
```
      end case;
   end process;

   -- output logic
   txout <= tx_reg;
   txdone <= txdonesig;
end Behavioral;
```

Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tx_test is
--  Port ( );
end tx_test;

architecture Behavioral of tx_test is
--signal
signal clk,rst,txout,tx_start,txdone : std_logic;
signal tx          : std_logic_vector(7 downto 0);
--constant
constant T : time := 200 ns;
begin
--port map
UUT: entity work.UART_Tx(behavioral) port map(clk => clk, rst => rst, tx => tx, tx_start => tx_start, txout
=> txout, txdone => txdone);
--clock
clock:process
  begin
    clk <= '1';
            wait for T/2;
            clk <= '0';
            wait for T/2;
  end process;
--reset
reset : process
  begin
    rst <= '1';
            wait for T;
            rst <= '0';
            tx_start <= '1';
            wait;
  end process;
--data bits
data  : process
  begin
     tx <= x"d1";
     wait for (10 * 163 * 16 * T);
```

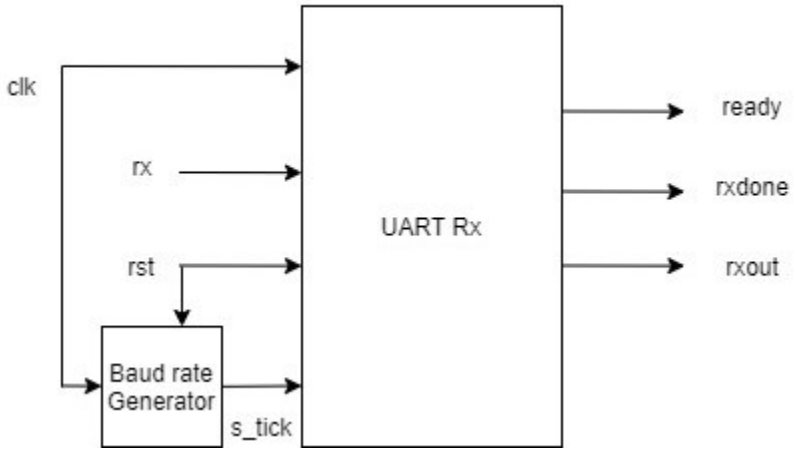
```
    tx <= x"af";
    wait for (10 * 163 * 16 * T);
  end process;
end Behavioral;
```

Waveform:



iii) <u>UART Rx</u>

The UART Rx integrated with baud rate generator.



VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity UART_Rx is
--  Port ( );
port(
    clk,rst,rx : in std_logic;
    ready      : out std_logic;
    rxdone     : out std_logic;
    rxout      : out std_logic_vector(7 downto 0));
end UART_Rx;

architecture Behavioral of UART_Rx is
--signal
type state_type is (idle, start, data, stop);   -- states
signal state_reg, state_next : state_type;
signal s_reg, s_next : integer range 0 to 15;   -- ticks = 16
signal n_reg, n_next : integer range 0 to 7;    -- data bits count = 8
signal b_reg, b_next : std_logic_vector( 7 downto 0);    -- data byte buffer containing 8 bits of data
signal s_tick       : std_logic;
--component Baud Generator
component Baudgen is
--  Port ( );
port(
    clk,rst: in std_logic;
    s_tick: out std_logic);
end component;
begin
B: Baudgen port map (clk => clk, rst => rst, s_tick => s_tick);
  process(clk,rst)
   begin
      if(rst = '1') then
         state_reg <= idle;
         s_reg    <= 0;
         n_reg    <= 0;
         b_reg    <= (others => '0');
      elsif(rising_edge(clk)) then
         state_reg <= state_next;
         s_reg    <= s_next;
         n_reg    <= n_next;
         b_reg    <= b_next;
      end if;
   end process;
--next state logic
   process(state_reg, s_reg, n_reg, b_reg, s_tick, rx)
   begin
```

```
s_next <= s_reg;
n_next <= n_reg;
b_next <= b_reg;
ready  <= '0';
rxdone <= '0';
case state_reg is
   -- idle state
   when idle =>
      if(rx = '0') then
         state_next <= start;
         s_next     <= 0;
      else
         state_next <= idle;
      end if;
      ready <= '1';
   -- start state
   when start =>
      if(s_tick = '0') then
         state_next <= start;
      else
         if(s_reg = 7) then
            state_next <= data;
            s_next     <= 0;
         else
            state_next <= start;
            s_next     <= s_reg + 1;
         end if;
      end if;
   -- data state
   when data =>
      if(s_tick = '0') then
         state_next <= data;
      else
         if(s_reg = 15) then
            s_next <= 0;
            b_next <= rx & b_reg(7 downto 1);
            if(n_reg = 7) then
               state_next <= stop;
               n_next     <= 0;
            else
               state_next <= data;
               n_next     <= n_reg + 1;
            end if;
         else
            state_next <= data;
            s_next <= s_reg + 1;
         end if;
      end if;
```

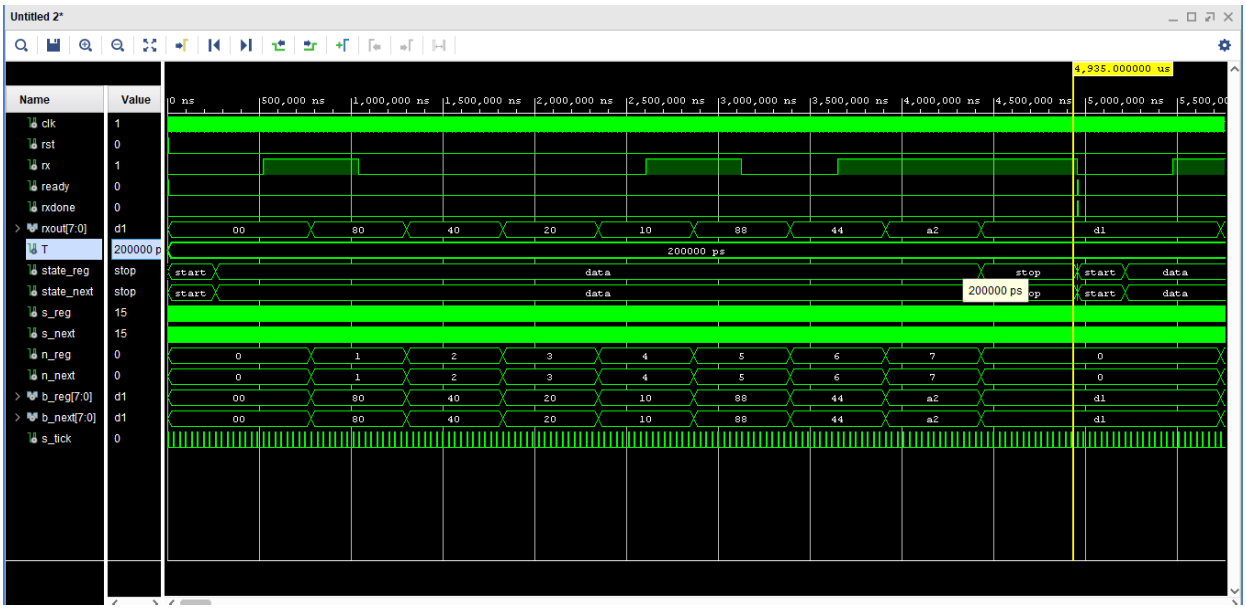
```
          when stop =>
             if(s_tick = '0') then
                state_next <= stop;
             else
                if(s_reg = 15) then
                   state_next <= idle;
                   s_next     <= 0;
                   rxdone     <= '1';   -- Rx done bit
                else
                   state_next <= stop;
                   s_next     <= s_reg + 1;
                end if;
             end if;
       end case;
    end process;
    rxout <= b_reg;
end Behavioral;
```

Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity UART_Rx_Sim is
--  Port ( );
end UART_Rx_Sim;

architecture Behavioral of UART_Rx_Sim is
--signal
signal clk,rst,rx,ready,rxdone : std_logic;
signal rxout           : std_logic_vector(7 downto 0);
--constant
constant T : time := 200 ns;
begin
--port map
UUT: entity work.UART_Rx(behavioral) port map(clk => clk, rst => rst, rx => rx, ready => ready, rxout =>
rxout, rxdone => rxdone);
--clock
clock:process
  begin
    clk <= '1';
            wait for T/2;
            clk <= '0';
            wait for T/2;
  end process;
--reset
reset : process
  begin
    rst <= '1';
```

```
            wait for T;
            rst <= '0';
            wait;
  end process;
--data bits
data  : process
   begin
      rx <= '0';
      wait for (163 * 16 * T);
      rx <= '1';
      wait for (163 * 16 * T);
      rx <= '0';
      wait for (163 * 16 * T);
      rx <= '0';
      wait for (163 * 16 * T);
      rx <= '0';
      wait for (163 * 16 * T);
      rx <= '1';
      wait for (163 * 16 * T);
      rx <= '0';
      wait for (163 * 16 * T);
      rx <= '1';
      wait for (163 * 16 * T);
      rx <= '1';
      wait for (163 * 16 * T);
      wait for (163 * 8 * T);
   end process;
end Behavioral;
```

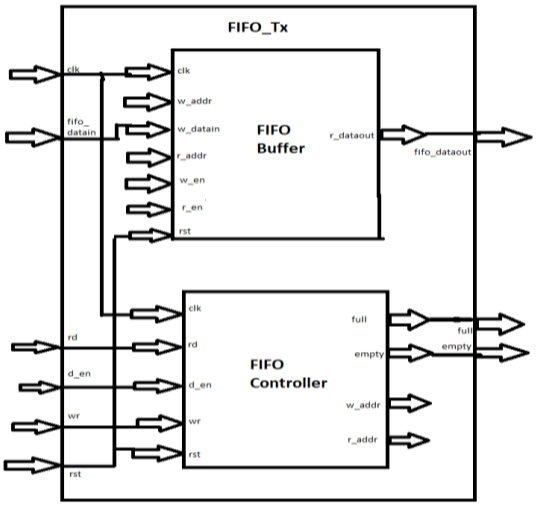
Waveform:

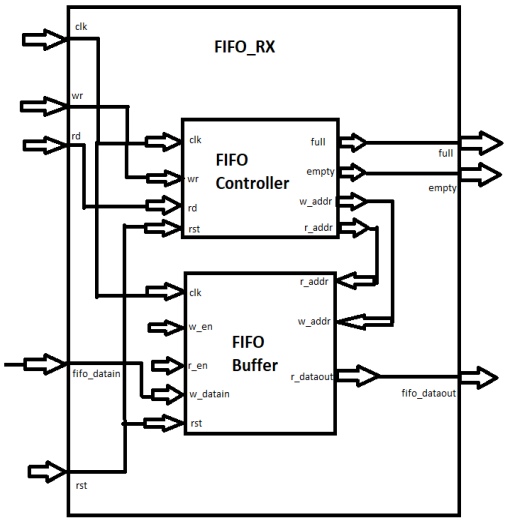


Module 2:

FIFO Interface circuit:

FIFO buffer includes a control circuit to a generic memory array, such as register file. FIFO buffer has a depth of 32 bytes. During the write enable the data is written into the buffer and when the read enable is set the data is read form the buffer. In the FIFO Controller the read and the write registers move to the next state when the empty and the full flags are zero, respectively.





VHDL Code:

i) FIFO Top

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Fifo is
--generic
generic (FIFO_DEPTH: integer := 5); --Fifo Depth is 32 bytes of data
--  Port ( );
port(
    clk,rst      : in std_logic;
    fifo_datain  : in std_logic_vector(7 downto 0);
    wr,rd        : in std_logic;
    full,empty   : out std_logic;
    fifo_dataout : out std_logic_vector(7 downto 0));
end Fifo;

architecture Behavioral of Fifo is
```

```
-- signals
signal w_addr_sig,r_addr_sig                              : std_logic_vector((FIFO_DEPTH - 1) downto 0);
signal w_en_sig, r_en_sig, full_sig, empty_sig, rd_sig, wr_sig     : std_logic;
-- components
-- Fifo controller
component Fifo_Controller is
--generic
generic (FIFO_DEPTH: integer := 5); --Fifo Depth is 32 bytes of data
--  Port ( );
port(
    clk,rst,rd,wr  : in std_logic;
    full,empty     : out std_logic;
    w_addr,r_addr  : out std_logic_vector((FIFO_DEPTH - 1) downto 0)
    );
end component;
-- fifo buffer
component FIFO_buffer is
--generic
generic (FIFO_DEPTH: integer := 5); --Fifo Depth is 32 bytes of data
--  Port ( );
port(
     clk,rst              : in std_logic;
     w_datain             : in std_logic_vector(7 downto 0);
     w_addr,r_addr        : in std_logic_vector((FIFO_DEPTH - 1) downto 0);
     w_en,r_en            : in std_logic;
     r_dataout            : out std_logic_vector(7 downto 0));
end component;

begin

   C : Fifo_Controller port map (clk => clk, rst => rst, rd => rd, wr => wr, full => full_sig, empty =>
empty_sig,
                          w_addr => w_addr_sig, r_addr => r_addr_sig);

   B : FIFO_buffer port map (clk => clk, rst => rst, w_datain => fifo_datain, w_addr => w_addr_sig,
                      r_addr => r_addr_sig, w_en => w_en_sig, r_en => r_en_sig, r_dataout => fifo_dataout);
   -- r_en
   r_en_sig <= '1' when (rd_sig = '1' and empty_sig = '0' ) else
            '0';
   -- w_en
   w_en_sig <= '1' when (wr_sig = '1' and full_sig = '0') else
            '0';
   -- output logic
   full  <= full_sig;
   empty <= empty_sig;
   rd_sig    <= rd;
   wr_sig    <= wr;
end Behavioral;
```

ii) FIFO Buffer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity FIFO_buffer is
--generic
generic (FIFO_DEPTH: integer := 5); --Fifo Depth is 32 bytes of data
--  Port ( );
port(
    clk,rst             : in std_logic;
    w_datain            : in std_logic_vector(7 downto 0);
    w_addr,r_addr       : in std_logic_vector((FIFO_DEPTH - 1) downto 0);
    w_en,r_en           : in std_logic;
    r_dataout           : out std_logic_vector(7 downto 0));
end FIFO_buffer;

architecture Behavioral of FIFO_buffer is
-- signals
type buffer_type is array ( 0 to (2 ** FIFO_DEPTH)-1) of std_logic_vector (7 downto 0); -- fifo buffer
contains 32 bytes of data
signal Fifo_buffer   : buffer_type;

begin
-- process register
   process(clk,rst)
   begin
      if(rst = '1') then
         Fifo_buffer <= (others => (others => '0'));
      elsif(rising_edge(clk))then
         if (w_en = '1') then
         Fifo_buffer(to_integer(unsigned(w_addr))) <= w_datain;
         end if;
      end if;
   end process;
      r_dataout <= Fifo_buffer(to_integer(unsigned(r_addr))) when r_en = '1' else
               (others => '0');
end Behavioral;
```

iii) FIFO controller

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity Fifo_Controller is
--generic
generic (FIFO_DEPTH: integer := 5); --Fifo Depth is 32 bytes of data
--  Port ( );
port(
    clk,rst,rd,wr  : in std_logic;
    full,empty     : out std_logic;
    w_addr,r_addr : out std_logic_vector((FIFO_DEPTH - 1) downto 0)
    );
end Fifo_Controller;

architecture Behavioral of Fifo_Controller is
--signals
signal r_addr_reg, r_addr_next : unsigned (FIFO_DEPTH downto 0); -- 0 to 32 (5 downto 0)
signal w_addr_reg, w_addr_next : unsigned (FIFO_DEPTH downto 0); -- 0 to 32 (5 downto 0)
signal full_flag, empty_flag : std_logic;

begin
--process register block
   process(clk, rst)
   begin
      if(rst = '1') then
         w_addr_reg <= (others =>'0');
         r_addr_reg <= (others =>'0');
      elsif(rising_edge(clk)) then
         w_addr_reg <= w_addr_next;
         r_addr_reg <= r_addr_next;
      end if;
   end process;
   -- write address next state logic
   w_addr_next <= w_addr_reg + 1 when (wr = '1' and full_flag = '0') else
               w_addr_reg;
  -- full flag logic
  full_flag <= '1' when (r_addr_reg(FIFO_DEPTH) /= w_addr_reg(FIFO_DEPTH)) and
(r_addr_reg(FIFO_DEPTH-1 downto 0) = w_addr_reg(FIFO_DEPTH-1 downto 0)) else
             '0';
  -- write output logic
  w_addr    <= std_logic_vector(w_addr_reg(FIFO_DEPTH - 1 downto 0));
  -- full flag output
  full <= full_flag;
  -- read address next state logic
  r_addr_next <= r_addr_reg + 1 when (rd = '1' and empty_flag = '0') else
              r_addr_reg;
  -- empty flag logic
  empty_flag <= '1' when (r_addr_reg = w_addr_reg) else
              '0';
  -- read port output
  r_addr     <= std_logic_vector(r_addr_reg(FIFO_DEPTH - 1 downto 0));
```

```
  -- empty flag output
  empty     <= empty_flag;
end Behavioral;
```

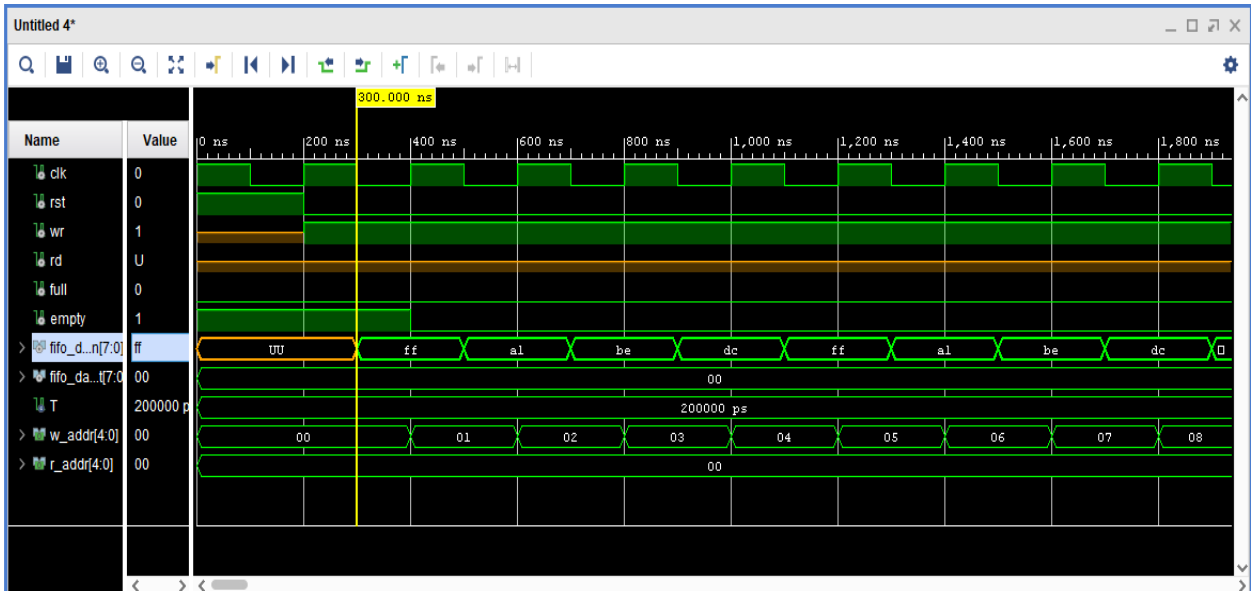
Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

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

architecture Behavioral of Fifo_Sim is
--signals
signal clk,rst,wr,rd,full,empty     :  std_logic;
signal fifo_datain,fifo_dataout     :  std_logic_vector(7 downto 0);
--constant
constant T : time := 200 ns;

begin
UUT : entity work.Fifo(behavioral) port map(clk => clk, rst => rst, wr => wr, rd => rd, full => full,
                                empty => empty, fifo_datain => fifo_datain, fifo_dataout => fifo_dataout);

--clock
clock : process
  begin
    clk <= '1';
           wait for T/2;
           clk <= '0';
           wait for T/2;
  end process;
--reset
  reset : process
    begin
      rst <= '1';
      wait for T;
```

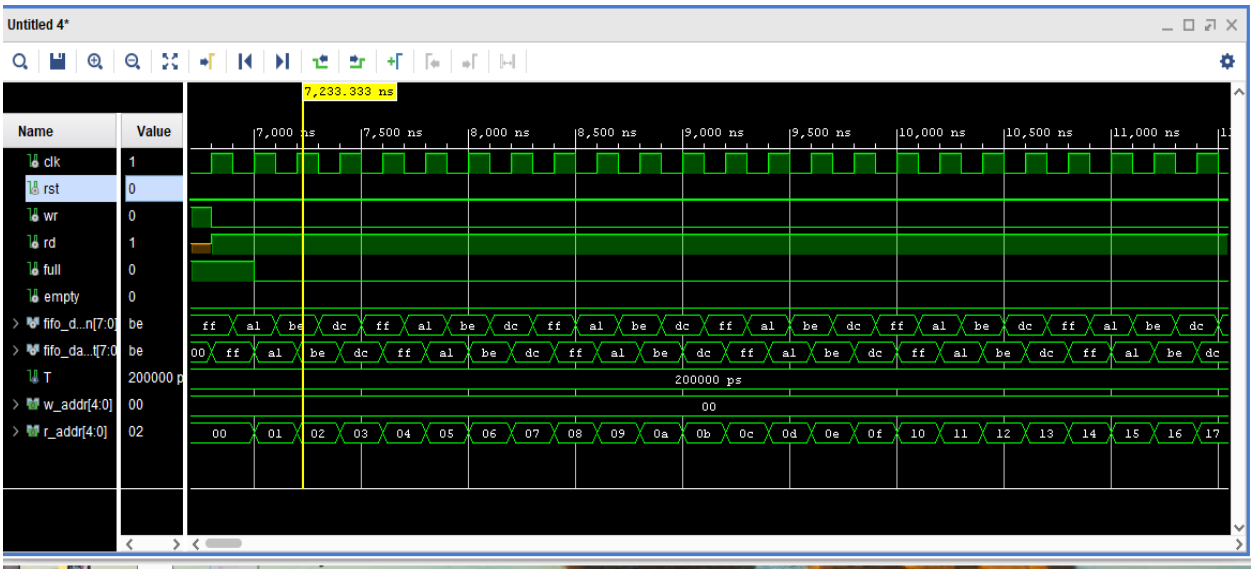
```
    rst <= '0';
    wait;
  end process;
-- wr_en
  w : process
     begin
        wait for T;
        wr  <= '1';
        wait for 33 * T;
        wr <= '0';
        wait;
     end process;
-- rd_en
  r : process
     begin
        wait for 34 * T;
        rd <= '1';
        wait for 33 * T;
        rd <= '0';
        wait;
     end process;
-- data
 data : process
    begin
       wait for T;
       wait until falling_edge(clk);
       for i in 0 to 32 loop
          fifo_datain <= x"ff";
          wait for T;
          fifo_datain <= x"a1";
          wait for T;
          fifo_datain <= x"be";
          wait for T;
          fifo_datain <= x"dc";
          wait for T;
       end loop;
    end process;
end Behavioral;
```

Waveform:

TX:



RX:



Module 3:

UART with Readback BRAM Module

It consists of UART controller, BRAM and the Top module integrating all the components(UART Subsystem with FIFO).

Four different modes :

Mode 1: When rd_uart_rx is set to 1, the data from FIFO buffer receiver is sent to the BRAM array.

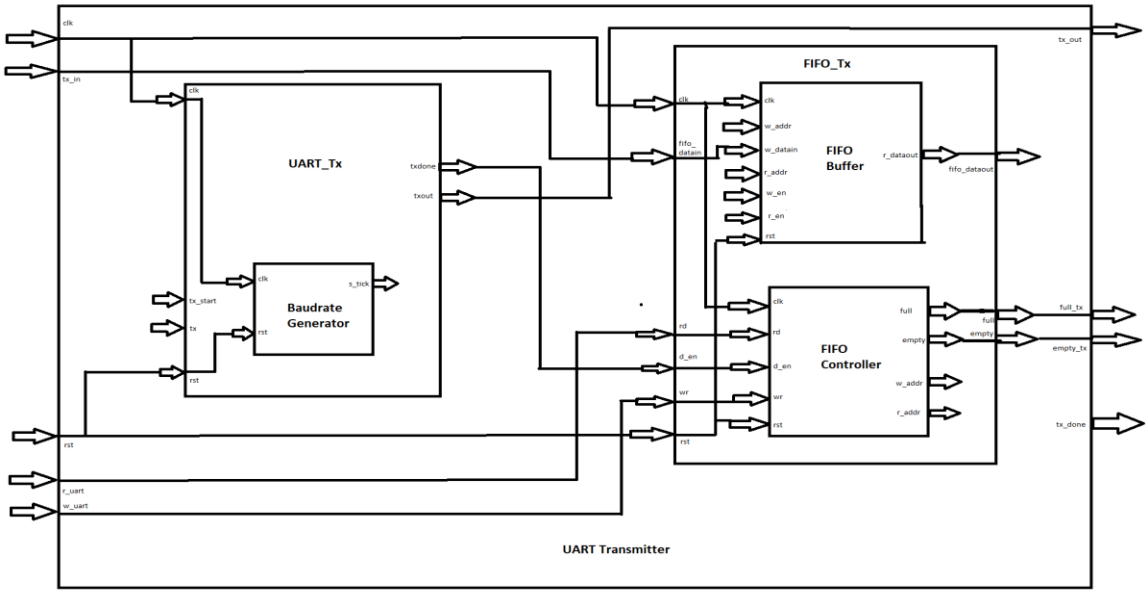
Mode 2: Reset all the flags in order to send the data from BRAM to UART.

Mode 3: When wr_uart_tx is set to 1 , the data from BRAM array is sent to FIFO buffer Transmitter.

Mode 4: Reset all the flags to carry out other operations.

UART Tx Subsystem with FIFO Interface:

Here the UART Tx is integrated with the FIFO Interface. The Baud rate generator provides the signals to transmitter which will determine the next state. A byte of data is loaded in parallel.
When w_uart= 1 the data is written in the buffer.
When r_uart =1: It checks for the empty and full flags.
When full=1 the data is read in tx_in and fifo_datain. The Fifo buffer holds the data and when the full flag is asserted it stops reading the data.

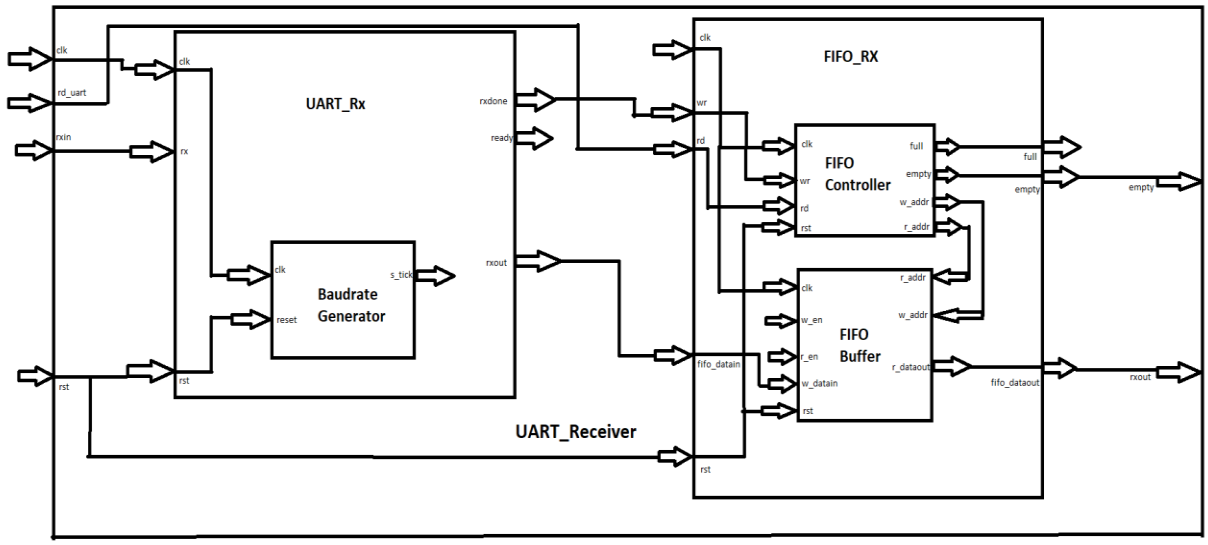


UART Subsystem with FIFO Interface:

Here the UART Rx is integrated with the FIFO Interface.The Baud rate generator provides the signals to receiver which will determine the next state.

When rd_uart =1:

- It checks for the empty and full flags.
- When empty=1 the data is available in rx_out and fifo_dataout.
- The Fifo buffer holds the data and when the full flag is asserted it stops reading the data.



Memory:

Vhdl Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Memory is
--  Port ( );
   port(
      address: in std_logic_vector(7 downto 0);
      datain: in std_logic_vector(7 downto 0);
      readwrite, en: in std_logic;
```

```
      clk: in std_logic;
      rst: in std_logic;
      dataout: out std_logic_vector(7 downto 0));
end Memory;

architecture RAM of Memory is
  type ram_type is array (0 to (2**address'length)-1) of std_logic_vector(dataout'range);
  signal ramArray : ram_type;

begin

process(clk,rst,readwrite) is
           begin
             if rst = '1' then
            -- Clear Memory on Reset
      ramArray <= (others => (others => '0'));
             else
             ramArray(128) <= x"02";
             ramArray(129) <= x"01";
             ramArray(130) <= x"00";
             ramArray(64)  <= x"11";
                       if rising_edge(Clk) then
                  if (readwrite = '1' or en ='1')then
                     ramArray(to_integer(unsigned(address))) <= datain;
                  end if;
               end if;
             end if;
   end process;
  dataout <= ramArray(to_integer(unsigned(address)));
end RAM;
```

UART Controller:

VHDL Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UART_Controller is
--  Port ( );
port(
    clk                                     : in std_logic;
    M1,M2,M3,M4,rxdone,txdone,empty_tx,empty_rx,full_tx,program_done       : in std_logic;
    fifo_data_in,mem_data_in                          : in std_logic_vector(7 downto 0);
    fifo_data_out,mem_data_out                          : out std_logic_vector(7 downto 0);
    mem_en,r_en,rd_uart_rx,wr_uart_tx,rd_uart_tx,mem_wr_en , start_prog_tx     : out std_logic);
end UART_Controller;
```

```
architecture Behavioral of UART_Controller is

begin
  --process
  process(M1,M2,M3,M4,empty_rx,full_tx,empty_tx,fifo_data_in,mem_data_in)
  begin
       if(M1 = '1') then
          mem_en <= '1';
          if(empty_rx = '1') then
            rd_uart_rx <= '0';
          else
            rd_uart_rx <= '1';
          end if;
          r_en    <= '0';
          wr_uart_tx <= '0';
          rd_uart_tx <= '0';
          mem_wr_en  <= '1';
          start_prog_tx <= '0';
          mem_data_out <= fifo_data_in;
      elsif(M2 = '1') then
           mem_en <= '0';
           rd_uart_rx <= '0';
            r_en    <= '0';
           wr_uart_tx <= '0';
           rd_uart_tx <= '0';
           mem_wr_en  <= '0';
           start_prog_tx <= '0';
          mem_data_out <= fifo_data_in;
           elsif(M3 = '1') then
          start_prog_tx <= '1';
          r_en    <= '0';
          mem_en    <= '1';
          mem_wr_en  <= '0';
         if(full_tx = '0' and empty_tx = '0') then
            rd_uart_tx <= '1';
            wr_uart_tx <= '1';
          elsif(empty_tx = '1') then
            wr_uart_tx <= '1';
            rd_uart_tx <= '0';
            elsif(full_tx = '1') then
            rd_uart_tx <= '1';
            wr_uart_tx <= '0';
           end if;
          start_prog_tx <= '1';
          fifo_data_out   <= mem_data_in ;

        elsif(M4 = '1') then
```

```
            r_en      <= '1';
            mem_en    <= '1';
             mem_wr_en  <= '1';
            if(full_tx = '1' ) then
            rd_uart_tx <= '1';
            wr_uart_tx <= '0';
            r_en <= '0';
            elsif(empty_tx = '1') then
            wr_uart_tx <= '1';
            rd_uart_tx <= '0';
            elsif(full_tx = '0' and empty_tx = '0' and txdone = '1') then
            rd_uart_tx <= '0';
            wr_uart_tx <= '0';
            end if;
            fifo_data_out   <= mem_data_in ;

      end if;
   end process;

end Behavioral;
```

Top Module:

Vhdl Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UART is
--  Port ( );
port(
    clk,rst,data_in,M1,M2,M3,M4   : in std_logic;
    data_out,txdone,rxdone        : out std_logic);
end UART;

architecture Behavioral of UART is
--component
-- Transmitter component along with baud generator and Transmitter FIFO
component UART_Transmitter is
--  Port ( );
port(
     clk,rst,w_uart,r_uart      : in std_logic;
     tx_in                      : in std_logic_vector(7 downto 0);
     tx_out,txdone,full_tx,empty_tx    : out std_logic);
end component;

-- Receiver component along with baud generator and Receiver FIFO
component UART_Receiver is
```

```
--  Port ( );
port(
    clk,rst,rxin,rd_uart      : in std_logic;
    empty,rxdone                : out std_logic;
    rxout                 : out std_logic_vector(7 downto 0));
end component;

--memory
component Memory is
--  Port ( );
port(
     clk, rst: in std_logic;
     readwrite , mem_wr_en: in std_logic;
     address : in std_logic_vector(7 downto 0);
     dataout       : in std_logic_vector(7 downto 0);
     datain  : out std_logic_vector(7 downto 0));
end component;

-- UART Controller
component UART_Controller is
--  Port ( );
port(
    clk                                      : in std_logic;
    M1,M2,M3,M4,rxdone,txdone,empty_tx,empty_rx,full_tx ,program_done  : in std_logic;
    fifo_data_in,mem_data_in                          : in std_logic_vector(7 downto 0);
    fifo_data_out,mem_data_out                          : out std_logic_vector(7 downto 0);
    mem_en,r_en,rd_uart_rx,wr_uart_tx,rd_uart_tx,mem_wr_en,start_prog_tx    : out std_logic);
end component;

--signals
signal done_sig,
mem_en,r_en,rd_uart_rx,wr_uart_tx,rd_uart_tx,mem_wr_en,rxdone_sig,tx_donesig,full_tx,empty_tx,tx
donesig : std_logic;
signal rxout_sig, fifo_in   : std_logic_vector(7 downto 0);
signal empty_rx,start_prog_tx ,readwritesig ,program_done : std_logic;
signal data_in_sig, data, addr  : std_logic_vector(7 downto 0);
begin

R   : UART_Receiver port map(clk => clk, rst => rst, rxin => data_in, rd_uart => rd_uart_rx,empty =>
empty_rx, rxout => rxout_sig,rxdone => rxdone_sig);
Con : UART_Controller port map(clk=>clk, M1 => M1, M2 => M2, M3 => M3, M4 => M4, rxdone =>
rxdone_sig,txdone => txdonesig,empty_tx => empty_tx, empty_rx => empty_rx,full_tx =>
full_tx,start_prog_tx =>start_prog_tx,program_done=>program_done,
                        fifo_data_in => rxout_sig, mem_data_in => data_in_sig,
                        fifo_data_out => fifo_in, mem_data_out => data, mem_en => mem_en, r_en => r_en,
                        rd_uart_rx => rd_uart_rx, wr_uart_tx => wr_uart_tx, rd_uart_tx =>
rd_uart_tx,mem_wr_en => mem_wr_en);
T   : UART_Transmitter port map (clk => clk, rst => rst, w_uart => wr_uart_tx, r_uart => rd_uart_tx,
```

```
                    tx_in => fifo_in, tx_out =>data_out, txdone => txdone,full_tx => full_tx, empty_tx =>
empty_tx);
M   : Memory port map (clk => clk, rst => rst, mem_wr_en=>mem_wr_en, address =>addr,
readwrite=>readwritesig, datain=>data_in_sig , dataout=>data
                );

txdone <= txdonesig;

end Behavioral;
```

Testbench:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture Behavioral of UART_Sim is
signal clk,rst,data_in : std_logic;
signal M1,M2,M3,M4 : std_logic:= '0';
signal     data_out,txdone            :  std_logic;
--constant
constant T : time := 200 ns;
begin
UUT: entity work.UART(Behavioral) port map (clk => clk, rst => rst, data_in => data_in, M1 => M1, M2 =>
M2, M3 => M3, M4 => M4, data_out => data_out,txdone => txdone);

--clock
clock:process
  begin
    clk <= '1';
          wait for T/2;
          clk <= '0';
          wait for T/2;
  end process;
--reset
reset : process
  begin
    rst <= '1';
          wait for T;
          rst <= '0';
          wait;
  end process;
```

```
 --mode
 mode: process
 begin
 wait for T;
 wait for (153 * 163 * 16 * T);
 M1 <= '1';
 M2 <= '0';
 M3 <= '0';
 M4 <= '0';
wait for (151 * 163 * 16 * T);
M2 <= '1';
M1 <= '0';
M3 <= '0';
M4 <= '0';
 wait for (70*T);
 M3 <= '1';
 M2 <= '0';
 M1 <= '0';
 M4 <= '0';
 wait for (1669050* T);
 M1 <= '0';
  M2 <= '0';
  M3 <= '0';
  M4 <= '1';
  wait for (26080 * T);
  end process;
--data bits
data  : process
  begin
       wait for T;
       wait until falling_edge(clk);
                                    --dc
       data_in <= '0';
       wait for (163 * 16 * T);
       data_in <= '0';
       wait for (163 * 16 * T);
       data_in <= '0';
       wait for (163 * 16 * T);
       data_in <= '1';
       wait for (163 * 16 * T);
       data_in <= '1';
       wait for (163 * 16 * T);
       data_in <= '1';
       wait for (163 * 16 * T);
       data_in <= '0';
       wait for (163 * 16 * T);
       data_in <= '1';
       wait for (163 * 16 * T);
```

```
        data_in <= '1';
        wait for (163 * 16 * T);
        wait for (163 * 8 * T);
                        --01
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '1';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        data_in <= '0';
        wait for (163 * 16 * T);
        wait for (163 * 8 * T);
                        --c2
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --80
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
```

```
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --48
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --88
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
```

```
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --77
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --24
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --f5
```

```
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --02
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --24
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
```

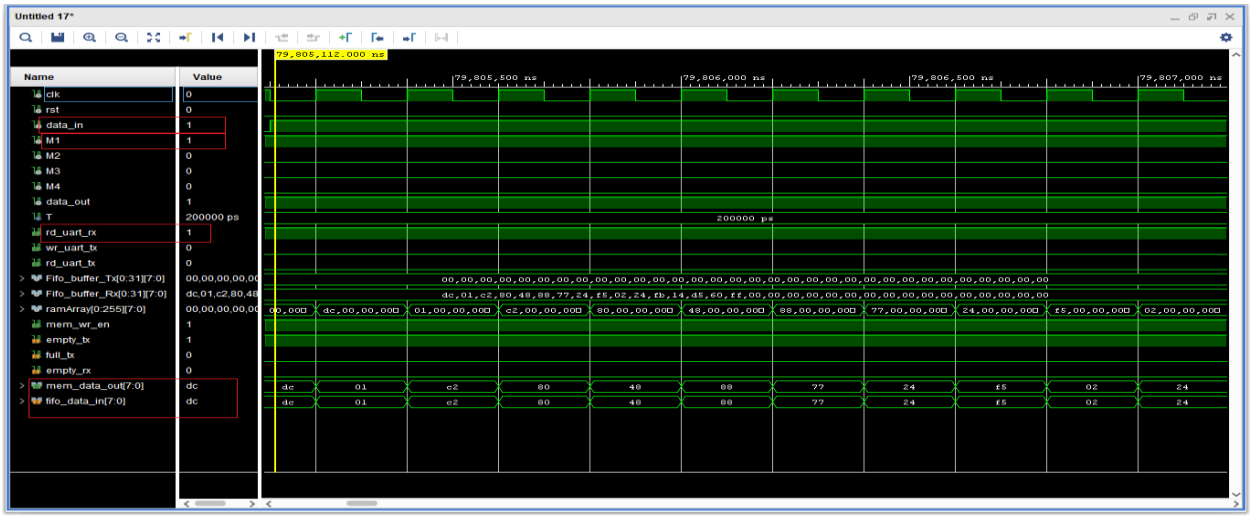
```
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --fb
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --14
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
```

```
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --d5
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --60
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --ff
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
```

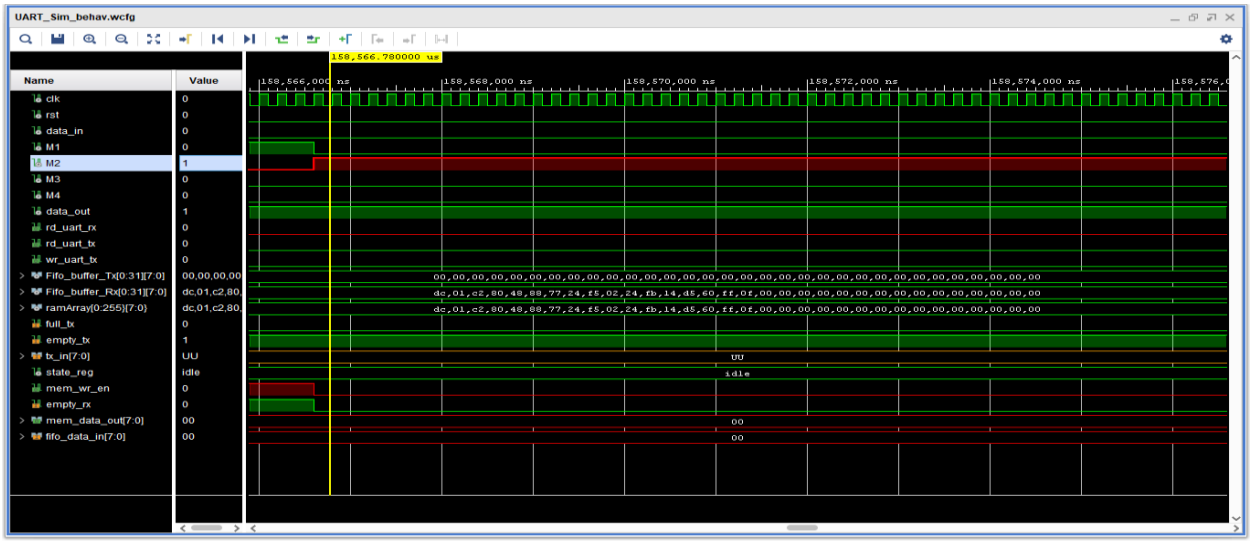
```
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
wait for (163 * 8 * T);
                        --0f
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '1';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait for (163 * 16 * T);
data_in <= '0';
wait;
  end process;
end Behavioral;
```

Waveform

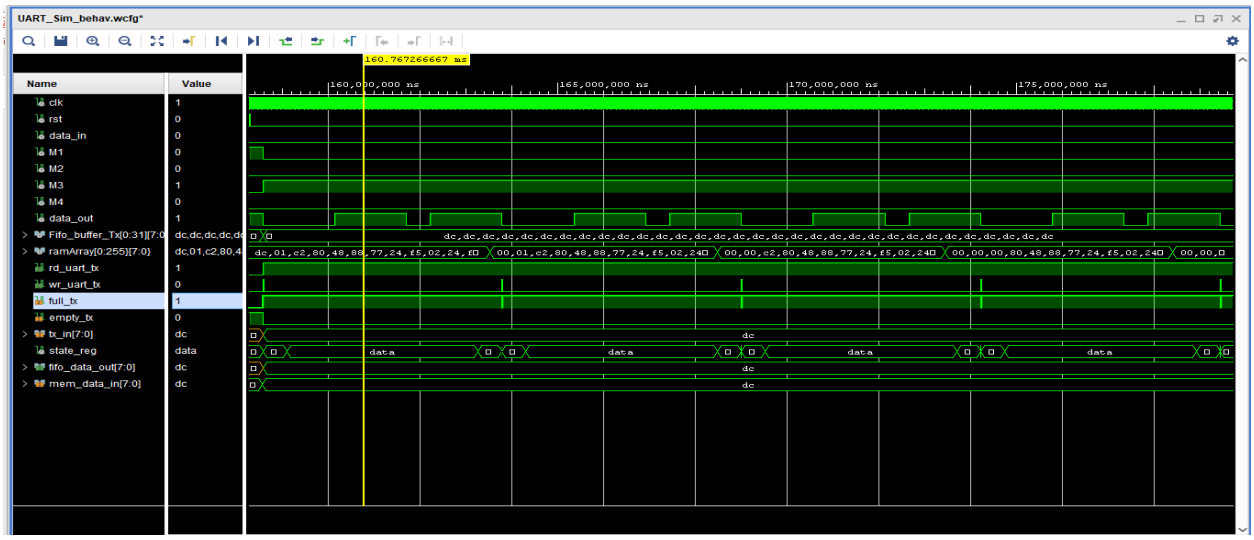
Mode 1:



Mode 2:



Mode 3:


UART_Sim_behav.wcfg*
160.767266667 ns
160,000,000 ns
165,000,000 ns
170,000,000 ns
175,000,000 ns
Name
Value
clk
rst
data_in
M1
M2
M3
M4
data_out
Fifo_buffer_Tx[0:31][7:0]
ramArray[0:255][7:0]
rd_uart_tx
wr_uart_tx
full_tx
empty_tx
tx_in[7:0]
state_reg
fifo_data_out[7:0]
mem_data_in[7:0]
data

Mode 4:


UART_Sim_behav.wcfg*
1,002.240677532 ns
1,000,000,000 ns
1,005,000,000 ns
1,010,000,000 ns
1,015,000,000 ns
Name
Value
clk
rst
data_in
M1
M2
M3
M4
data_out
Fifo_buffer_Tx[0:31][7:0]
ramArray[0:255][7:0]
rd_uart_tx
wr_uart_tx
full_tx
empty_tx
tx_in[7:0]
state_reg
fifo_data_out[7:0]
mem_data_in[7:0]
data