مباحث ویژه در الکترونیک

VHDL

Dr. Mohsen Shakiba

Practice4:

UART Controller:

UART_RX and TestBench

Mohammad Niknam

951843189

تمرین شماره ۴

طبق پروتکل UART که پیش از این به آن پرداختیم، یک کنترلر برای دریافت اطلاعات، طراحی و مبتنی بر VHDL(قابل سنتز) توصیف نمایید. Baudrate را 9600کیلوبیت بر ثانیه (اصلاح : 9600 بیت بر ثانیه) فرض کنید. حتما سیگنالهای کنترلی لازم را در نظر بگیرید (مثلا یک سیگنال خروجی که دریافت یک بسته داده جدید را اعلان نماید). (راهنمایی: این ماژول ورودی سریال UART را دریافت می کند و پس از دریافت هر بسته اطلاعات 8 بیتی، آن را در خروجی قرار می دهد)

-عملکرد کنترلر طراحی شده را با نوشتن یک تست بنچ ارزیابی نمایید. شبیه سازی در سطح پیاده سازی نهایی و یا همان Post-Route انجام شود. (راهنمایی: برای ارزیابی عملکرد کنترلر گیرنده، در واقع شما باید یک یا چند بسته داده UART را به عنوان ورودی کنترلر تولید نمایید که این کار در یک فرآیند با استفاده صحیح از عبارت های wait قابل انجام است)

نکته : به منظور کاهش زمان شبیه سازی میتوانید نرخ ارسال داده را از 9600 به 115200 بیت بر ثانیه افزایش دهید. به این ترتیب تعداد پالسهای ساعت برای انتقال هر بیت به 24000000/115200 = 208 پالس کاهش می یابد (با فرض فرکانس پالس ساعت 24 مگاهرتز

در طراحی برای سادگی در ارزیابی ماژول فرکانس و baudrate به نحو زیر در نظر گرفته شد

Freq: 100 MHZ

Baudrate: 115200

Pulse for each bit: 868

Universal Asynchronous Receiver/Transmitter

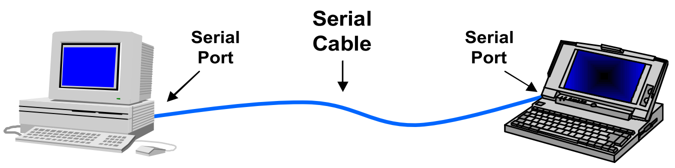
UART

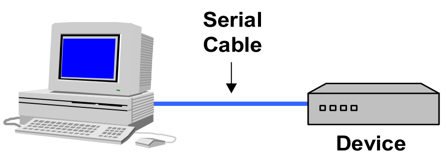
Why use a UART?

- A UART may be used when:
  - High speed is not required
  - A cheap communication line between **two** devices is required
- Asynchronous serial communication is very cheap
  - Requires a transmitter and/or receiver
  - Single wire for each direction (plus ground wire)
  - Relatively simple hardware
  - Asynchronous because the
- PC devices such as mice and modems used to often be asynchronous serial devices

UART Uses

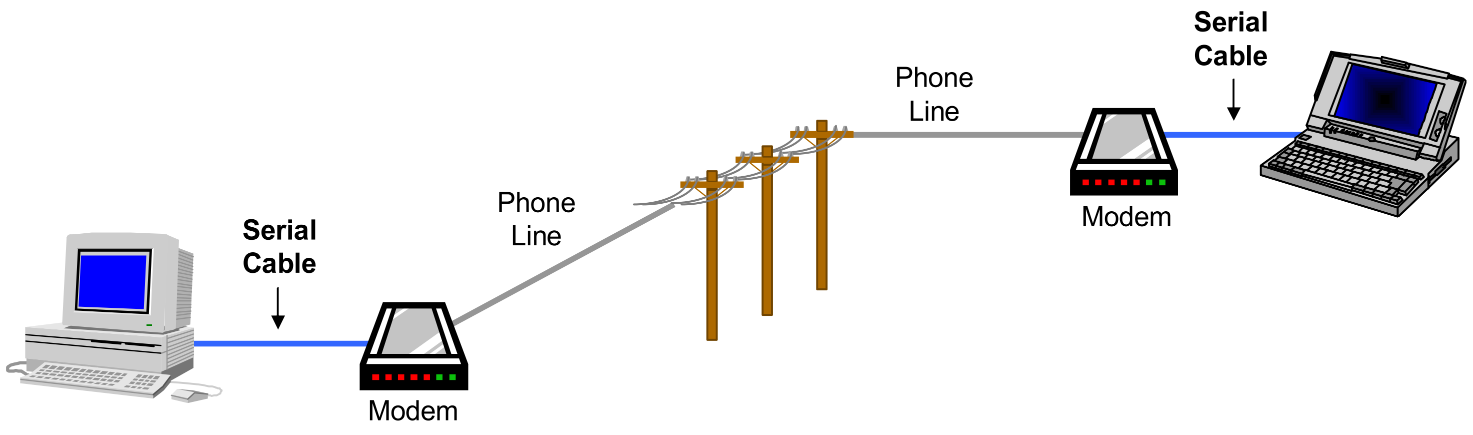
- PC serial port is a UART!
- Serializes data to be sent over serial cable
  - De-serializes received data



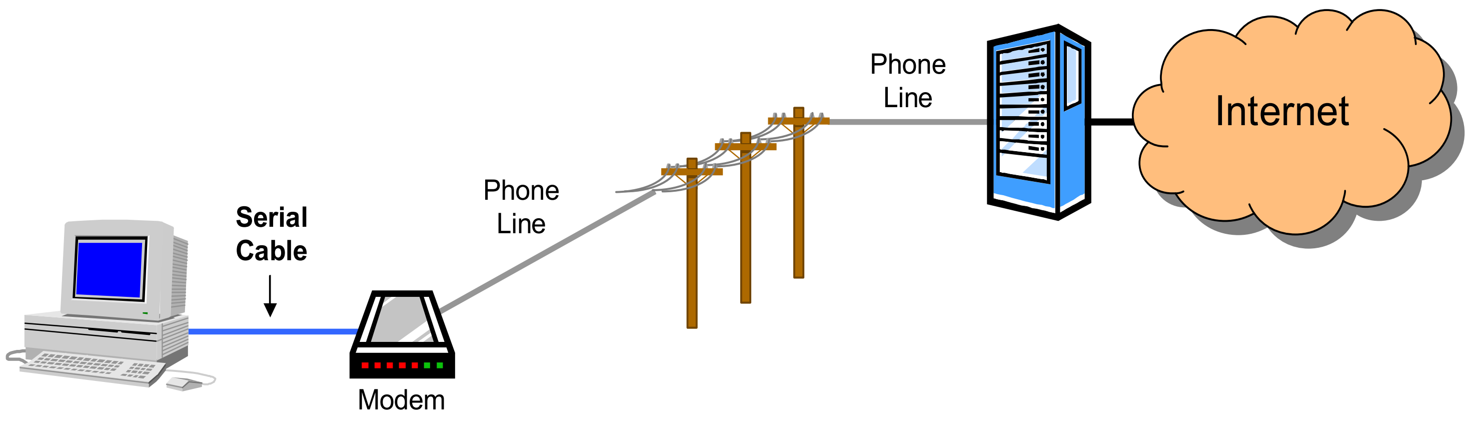


UART Uses

- Communication between distant computers
  - Serializes data to be sent to modem
  - De-serializes data received from modem

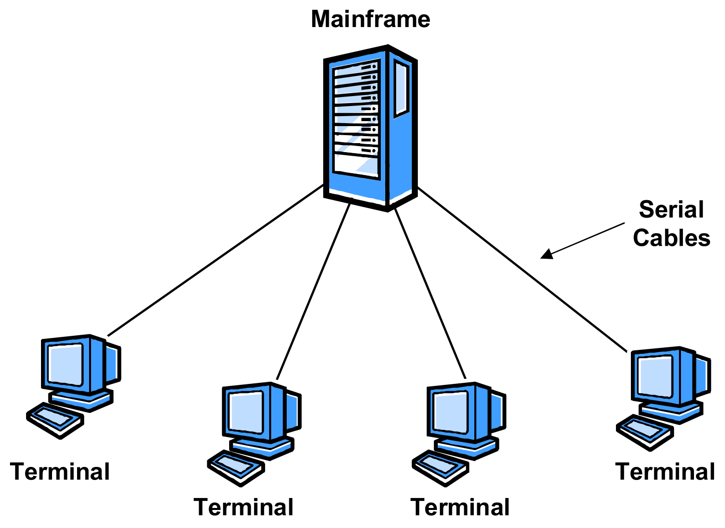


- Used to be commonly used for internet access



UART Uses

- Used to be used for mainframe access
  - A mainframe could have dozens of serial ports



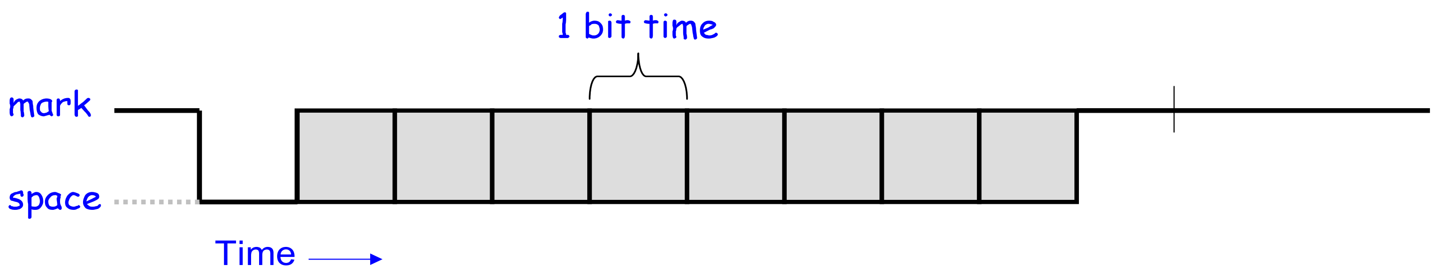
- Becoming much less common
- Largely been replaced by faster, more sophisticated interfaces
  - PCs: USB (peripherals), Ethernet (networking)
  - Chip to chip: I2C, SPI
- Still used today when simple low speed communication is needed

UART Functions

- Outbound data
  - Convert from parallel to serial
  - Add start and stop delineators (bits)
  - Add parity bit
- Inbound data
  - Convert from serial to parallel
  - Remove start and stop delineators (bits)
  - Check and remove parity bit

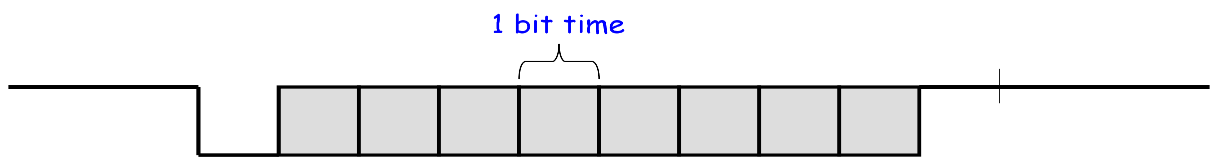
UART Character Transmission

- Below is a timing diagram for the transmission of a single byte
- Uses a single wire for transmission
- Each block represents a bit that can be a **mark** (logic '1', high) or **space** (logic '0', low)



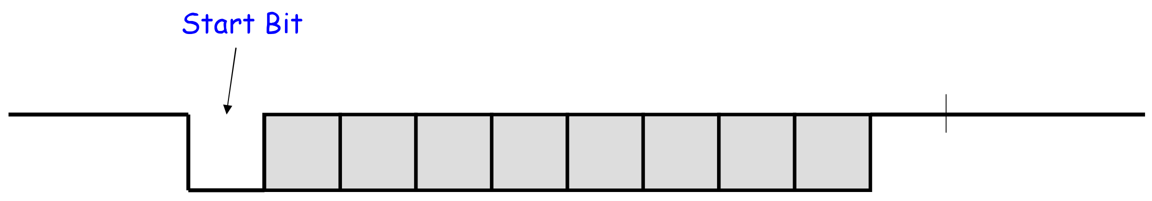
UART Character Transmission

- Each bit has a fixed time duration determined by the transmission rate
- Example: a 1200 bps (bits per second) UART will have a 1/1200 s or about 833.3 μs bit width



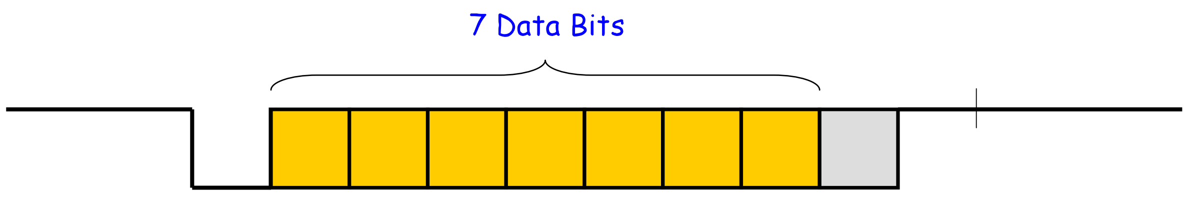
UART Character Transmission

- The **start bit** marks the beginning of a new word
- When detected, the receiver synchronizes with the new data stream



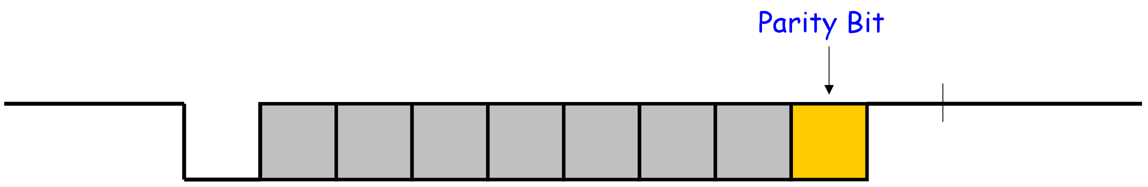
UART Character Transmission

- Next follows the **data bits** (7 or 8)
- The least significant bit is sent first



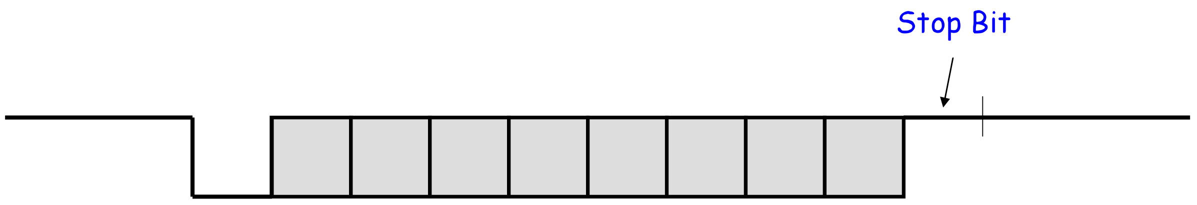
UART Character Transmission

- The **parity bit** is added to make the number of 1's even (even parity) or odd (odd parity)
- This bit can be used by the receiver to check for transmission errors



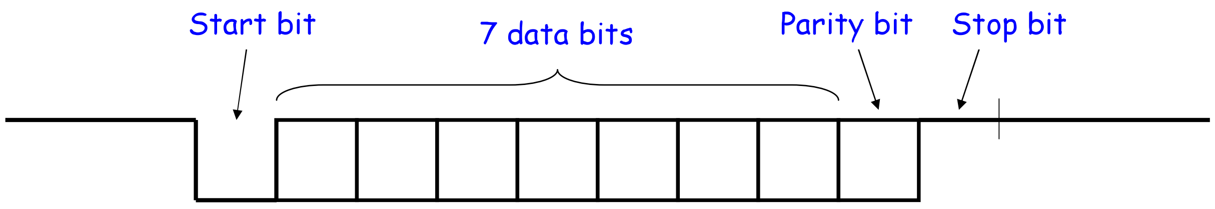
UART Character Transmission

- The **stop bit** marks the end of transmission
- Receiver checks to make sure it is '1'
- Separates one word from the start bit of the next word



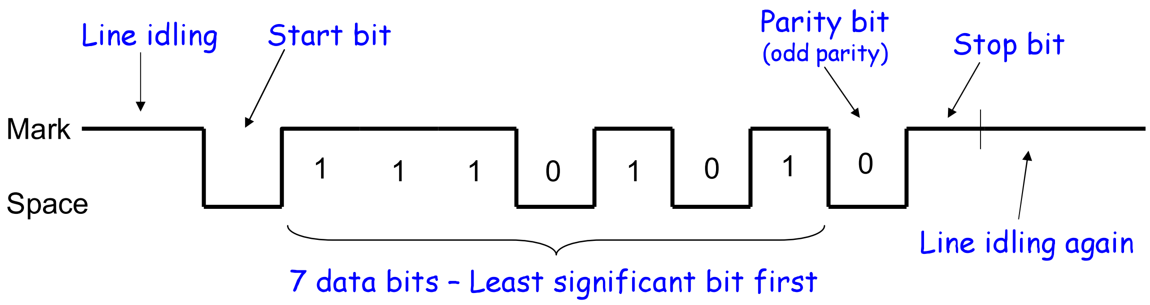
UART Character Transmission

- In the configuration shown, it takes 10 bits to send 7 bits of data

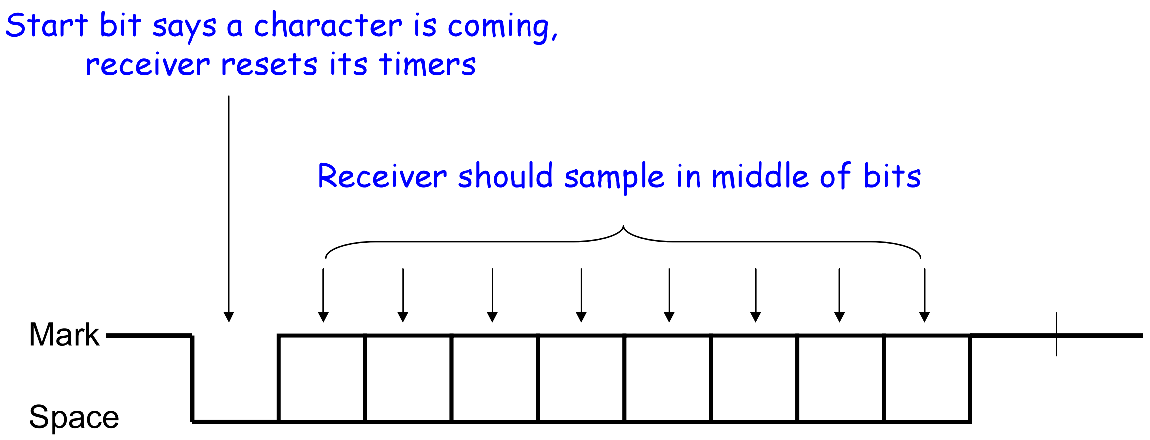


UART Transmission Example

- Send the ASCII letter 'W' (1010111)



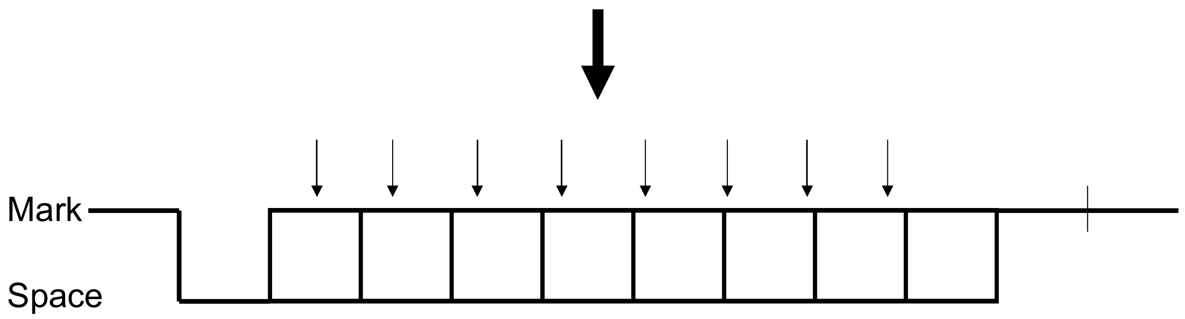
UART Character Reception


Start bit says a character is coming,
receiver resets its timers
Receiver should sample in middle of bits
Mark
Space

Receiver uses a timer (counter) to time when it samples.
Transmission rate (i.e., bit width) must be known!

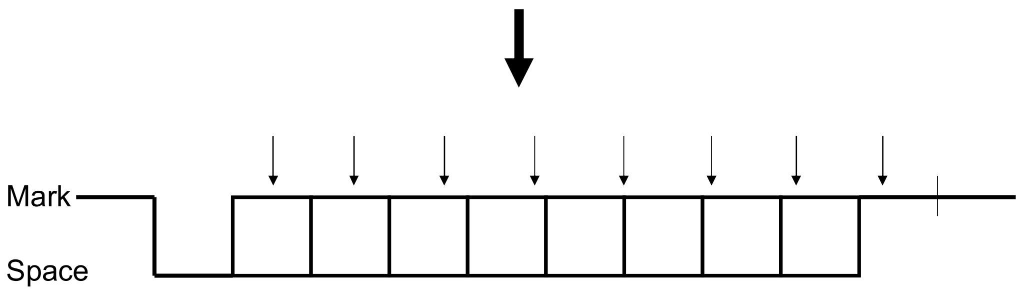
UART Character Reception

If receiver samples too quickly, see what happens…



UART Character Reception

If receiver samples too slowly, see what happens…



Receiver resynchronizes on every start bit.
Only has to be accurate enough to read 9 bits.

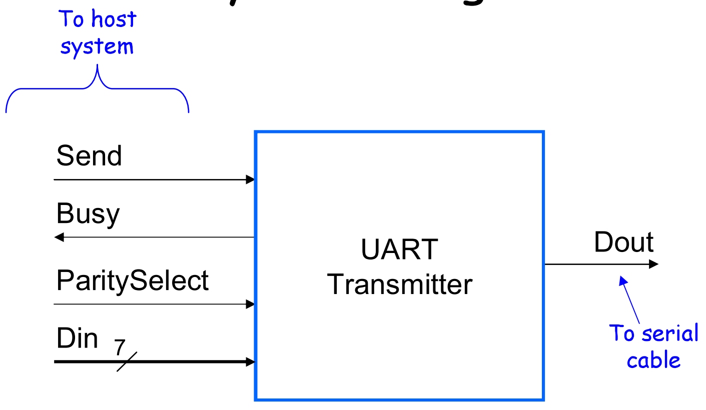
UART Character Reception

- Receiver also verifies that stop bit is '1'
  - If not, reports "framing error" to host system
- New start bit can appear immediately after stop bit
  - Receiver will resynchronize on each start bit

UAR T Options

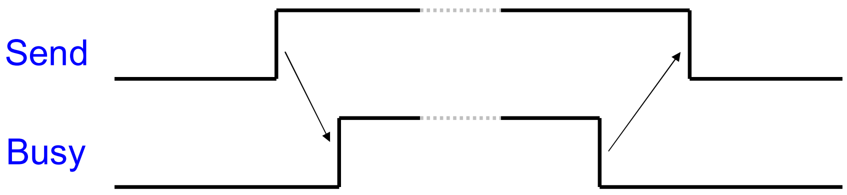
- UARTs usually have programmable options:
  - **Data:** 7 or 8 bits
  - **Parity:** even, odd, none, mark, space
  - **Stop bits:** 1, 1.5, 2
  - **Baud rate:** 300, 1200, 2400, 4800, 9600, 19.2K, 38.4k, 57.6k, 115.2k…

Design a UART Transmitter! System Diagram

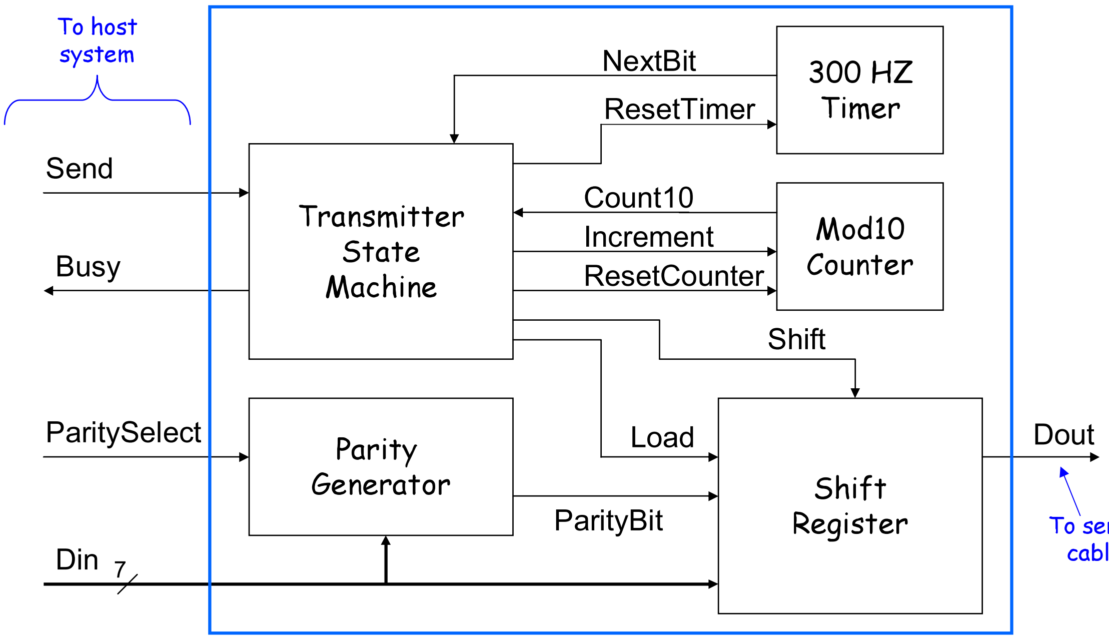


Transmitter/System Handshaking

- System asserts Send and holds it high when it wants to send a byte
- UART asserts Busy signal in response
- When UART has finished transfer, UART de-asserts Busy signal
- System de-asserts Send signal

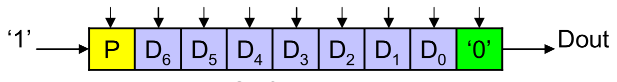


Transmitter Block Diagram



The Shift Register

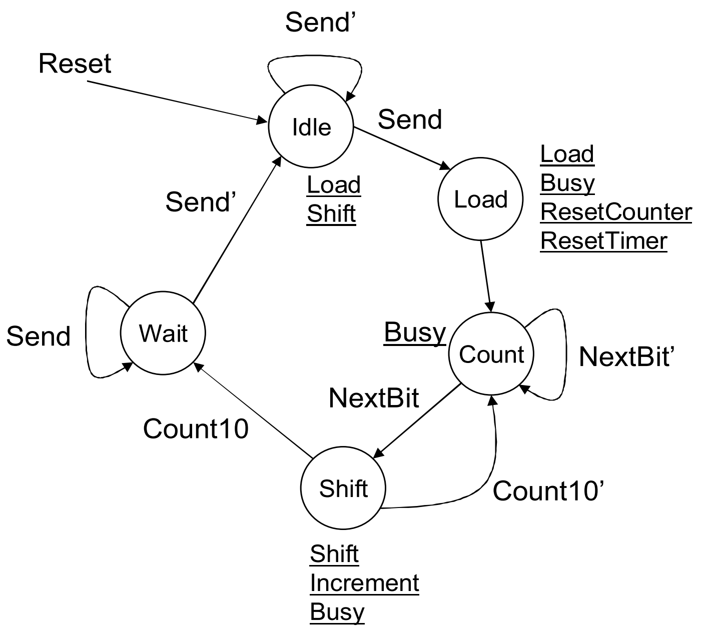
- Make it a 9-bit register
- When it loads:
  - Have it load '0' for the start bit on the right (LSB)
  - Have it load the parity bit on the left (MSB)
  - Have it load 7 data bits in the middle
- When it shifts:
  - Have it shift '1' into the left so a stop bit is sent at the end



Shift Register

- When it resets:
  - Have it load all 1's so that its default output is a '1' (line idle value)

Transmitter FSM



Be sure to choose state encodings and use logic minimization that ensures **Busy** signal will have no hazards…

UART_RX:

VHDL CODE

```vhdl
-- Engineer: Mohammad Niknam
-- Project Name:  UART_Controller
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity UART_RX is
    Generic(CLK_PULSE_NUM : integer := 868); --868 for freq=100Mhz / BUADRATE 115200
    port (  CLK : in std_logic;
            nRST :in std_logic;
            DATA_OUT : out std_logic_vector(7 downto 0);
            DATA_VLD : out std_logic; -- when DATA_VLD = 1, data reception is
            completed data on DATA_OUT are valid
            RX_Busy : out std_logic;  --data reception in progress
            Received_serial : in std_logic);
end UART_RX;

architecture Behavioral of UART_RX is
type FSMTYPE is (INIT_STATE, START_BIT_Receive, BIT_0, BIT_1, BIT_2, BIT_3, BIT_4,
BIT_5, BIT_6, BIT_7, STOP_BIT_Receive);

signal CSTATE, NSTATE : FSMTYPE;
signal CLK_CNT : unsigned(11 downto 0) ;
signal CLK_CNT_RST : std_logic;
signal DATA_REG : std_logic_vector(7 downto 0);
signal TMP : std_logic_vector(7 downto 0);
signal DATA_VALID : std_logic;

begin

    data_registration : process( CLK )
        variable VTMP : std_logic_vector(7 downto 0);
    begin
        if (CLK'event and CLK = '1') then
            TMP <= DATA_REG;
            if (DATA_VALID = '1') then
                DATA_OUT <= DATA_REG;
                DATA_VLD <= '1';
                VTMP := DATA_REG;
            else
                DATA_VLD <= '0';
                DATA_OUT <= VTMP;
            end if ;
        end if ;
    end process ; --data_registration


clk_counter : process( CLK )
begin
    if (CLK'event and CLK = '1') then
        if (CLK_CNT_RST = '1') then
            CLK_CNT <= (others => '0');
        else
            CLK_CNT <= CLK_CNT + 1;
        end if ;
    end if ;
end process ; -- clk_counter


state_registration : process( CLK )
begin
    if (CLK'event and CLK = '1') then
        if (nRST = '0') then
            CSTATE <= INIT_STATE;
        else
            CSTATE <= NSTATE;
        end if ;
    end if ;
end process ; -- state_registration
```

UART_RX:

VHDL CODE

```vhdl
next_state : process( CSTATE, CLK_CNT, DATA_REG, Received_serial, TMP)
begin
    NSTATE <= CSTATE;
    CLK_CNT_RST <= '0';
    DATA_VALID <= '0';
    DATA_REG <= TMP;

    case( CSTATE ) is
        when INIT_STATE =>
            DATA_REG <= (others => '0');
            RX_Busy <= '0';
            CLK_CNT_RST <= '1';
            if (Received_serial = '0') then
                NSTATE <= START_BIT_Receive;
            end if ;

        when START_BIT_Receive =>
            RX_Busy <= '1';
            if (TO_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_0 ;
            end if ;

        when BIT_0 =>
            DATA_REG(0) <= Received_serial;
            RX_Busy <= '1';
            if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_1;
            end if;

        when BIT_1 =>
            DATA_REG(1) <= Received_serial;
            RX_Busy <= '1';
            if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_2;
            end if;

        when BIT_2 =>
            DATA_REG(2) <= Received_serial;
            RX_Busy <= '1';
            if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_3;
            end if;

        when BIT_3 =>
            DATA_REG(3) <= Received_serial;
            RX_Busy <= '1';
            if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_4;
            end if;

        when BIT_4 =>
            DATA_REG(4) <= Received_serial;
            RX_Busy <= '1';
            if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_5;
            end if;

        when BIT_5 =>
            DATA_REG(5) <= Received_serial;
            RX_Busy <= '1';
            if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                CLK_CNT_RST <= '1';
                NSTATE <= BIT_6;
            end if;

```

UART_RX:

```vhdl
            when BIT_6 =>
                DATA_REG(6) <= Received_serial;
                RX_Busy <= '1';
                if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                    CLK_CNT_RST <= '1';
                    NSTATE <= BIT_7;
                end if;

            when BIT_7 =>
                DATA_REG(7) <= Received_serial;
                RX_Busy <= '1';
                if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                    CLK_CNT_RST <= '1';
                    NSTATE <= STOP_BIT_Receive;
                end if;

            when STOP_BIT_Receive =>
                RX_Busy <= '1';
                if (to_integer(CLK_CNT) = (CLK_PULSE_NUM - 1)) then
                    CLK_CNT_RST <= '1';
                    NSTATE <= INIT_STATE;
                    RX_Busy <= '0';
                    if (Received_serial = '1') then
                        DATA_VALID <= '1';
                    end if ;
                end if;

            when others =>
        end case ;
    end process ; -- next_state

    end Behavioral;
```

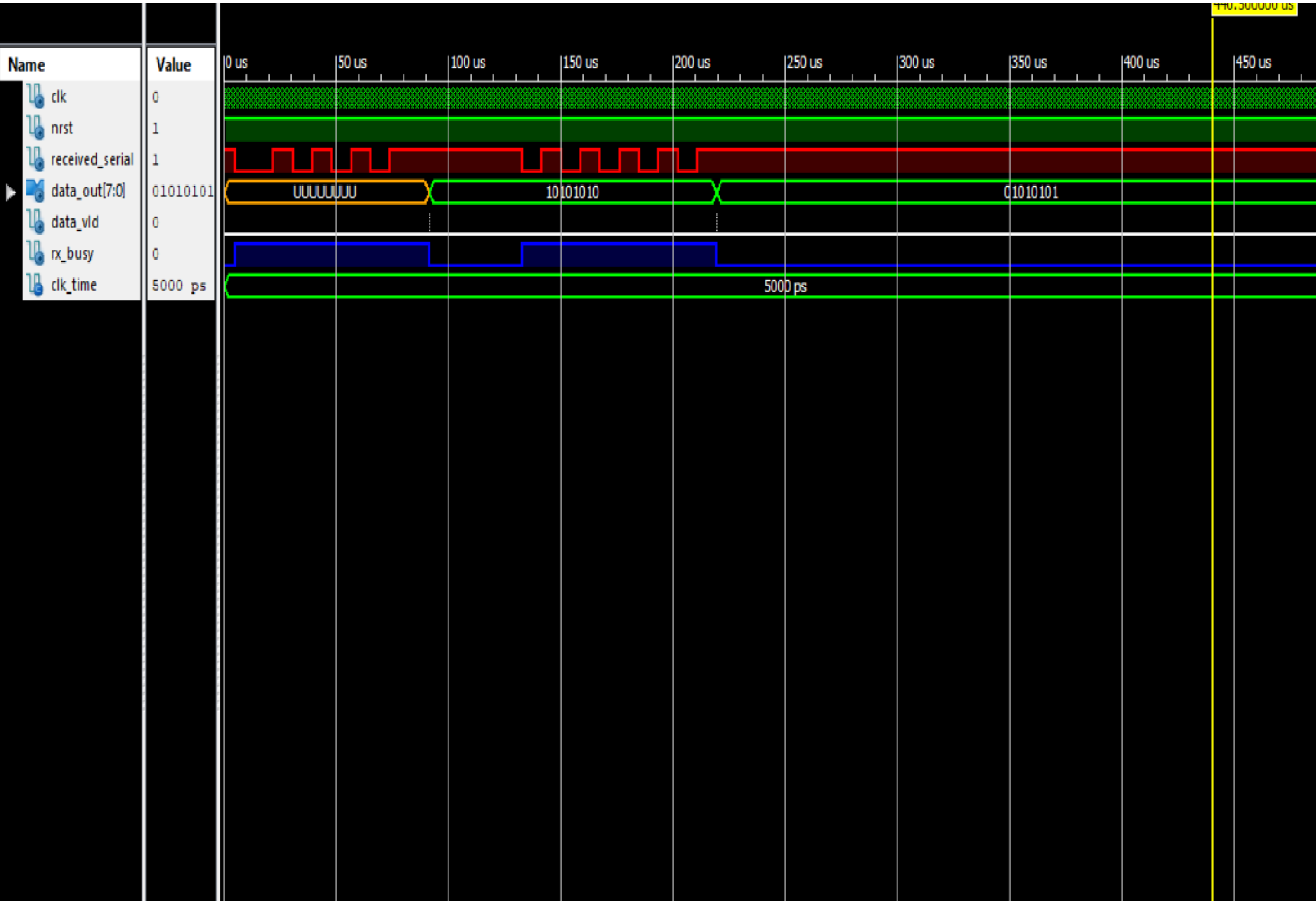
UART_RX_TB:

```vhdl
-- Engineer: Mohammad Niknam
-- Project Name:  UART_Controller
-- VHDL Test Bench Created by ISE for module: UART_RX
-- CLK_PULSE_NUM : 868 for freq=100Mhz / BUADRATE 115200
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY UART_RX_TB IS
END UART_RX_TB;

ARCHITECTURE behavior OF UART_RX_TB IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT UART_RX
    PORT(
         CLK : IN  std_logic;
         nRST : IN  std_logic;
         DATA_OUT : OUT  std_logic_vector(7 downto 0);
         DATA_VLD : OUT  std_logic;
         RX_Busy : OUT  std_logic;
         Received_serial : IN  std_logic
        );
    END COMPONENT;


   --Inputs
   signal CLK : std_logic := '0';
   signal nRST : std_logic := '0';
   signal Received_serial : std_logic := '0';

    --Outputs
   signal DATA_OUT : std_logic_vector(7 downto 0);
   signal DATA_VLD : std_logic;
   signal RX_Busy : std_logic;

   -- Clock period definitions
   constant CLK_TIME : time := 5 ns;      --DATA_Receive_period : ((868*2)+1) *
   CLK_TIME ns;

   BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: UART_RX PORT MAP (
          CLK => CLK,
          nRST => nRST,
          DATA_OUT => DATA_OUT,
          DATA_VLD => DATA_VLD,
          RX_Busy => RX_Busy,
          Received_serial => Received_serial
        );





```

UART_RX_TB::

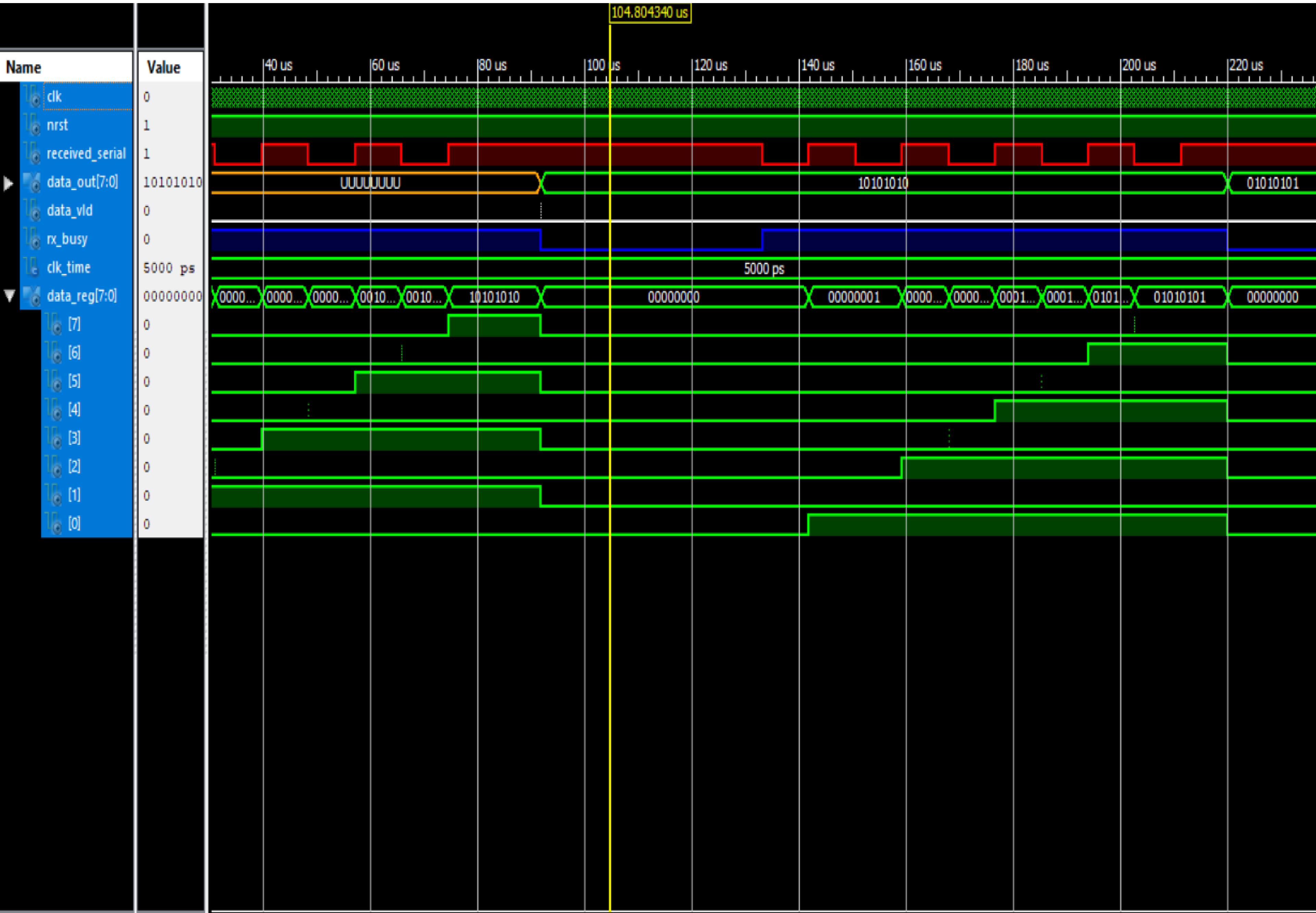
```
    CLK <= not(CLK) after CLK_TIME;
    nRST <= '0', '1' after 400 ns;  -- hold reset state for 400 ns.
    pro: process
    begin
       Received_serial <= '1';
       wait for 5000 ns;
       Received_serial <= '0'; --START_BIT
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_0
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_1
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_2
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_3
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_4
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_5
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_6
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_7
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --STOP_BIT
       wait for 50000 ns;
       Received_serial <= '0'; --START_BIT
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_0
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_1
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_2
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_3
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_4
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_5
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --BIT_6
       wait for 1737 * CLK_TIME;
       Received_serial <= '0'; --BIT_7
       wait for 1737 * CLK_TIME;
       Received_serial <= '1'; --STOP_BIT
       wait;
    end process;
 END;
```

Simulation:

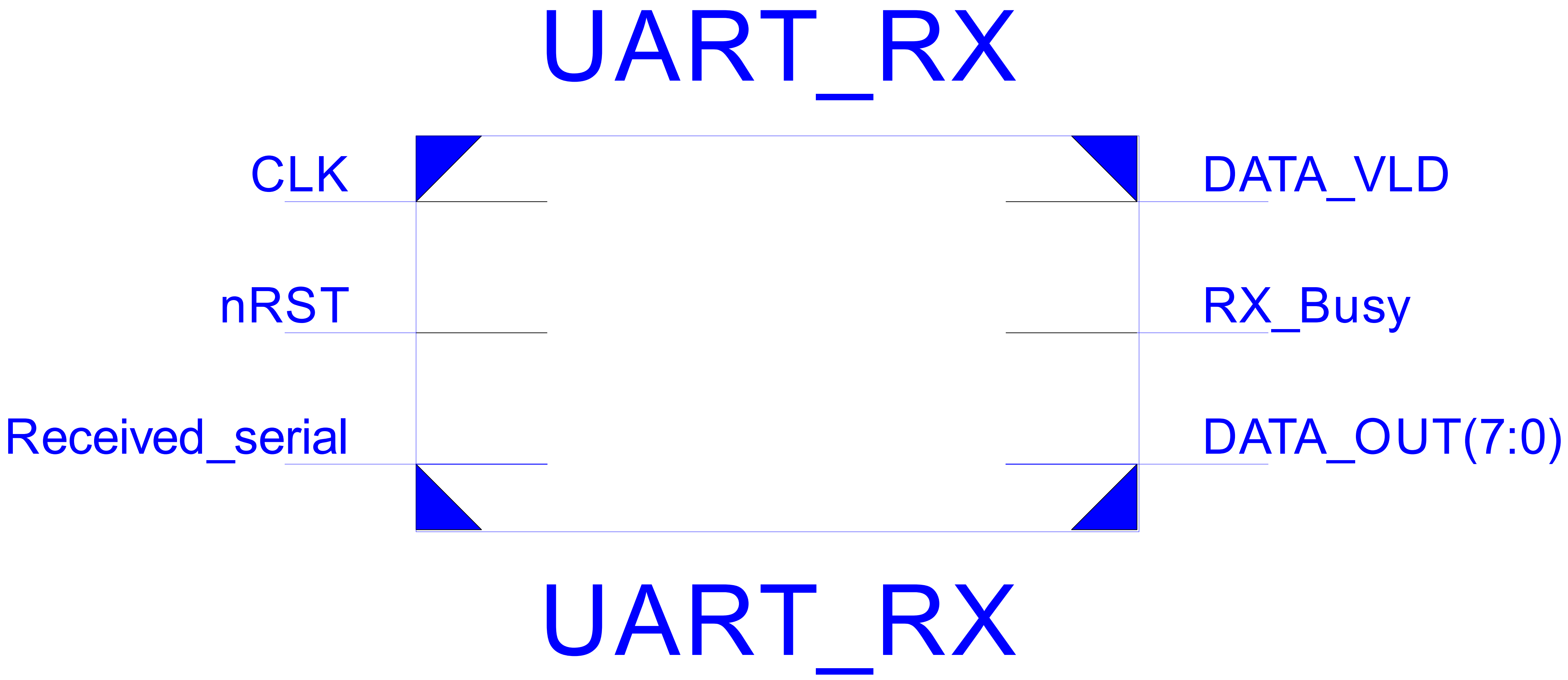


Name
Value
clk
0
nrst
1
received_serial
1
data_out[7:0]
01010101
data_vld
0
rx_busy
0
clk_time
5000 ps
0 us
50 us
100 us
150 us
200 us
250 us
300 us
350 us
400 us
450 us
UUUUUUUU
10101010
01010101
5000 ps

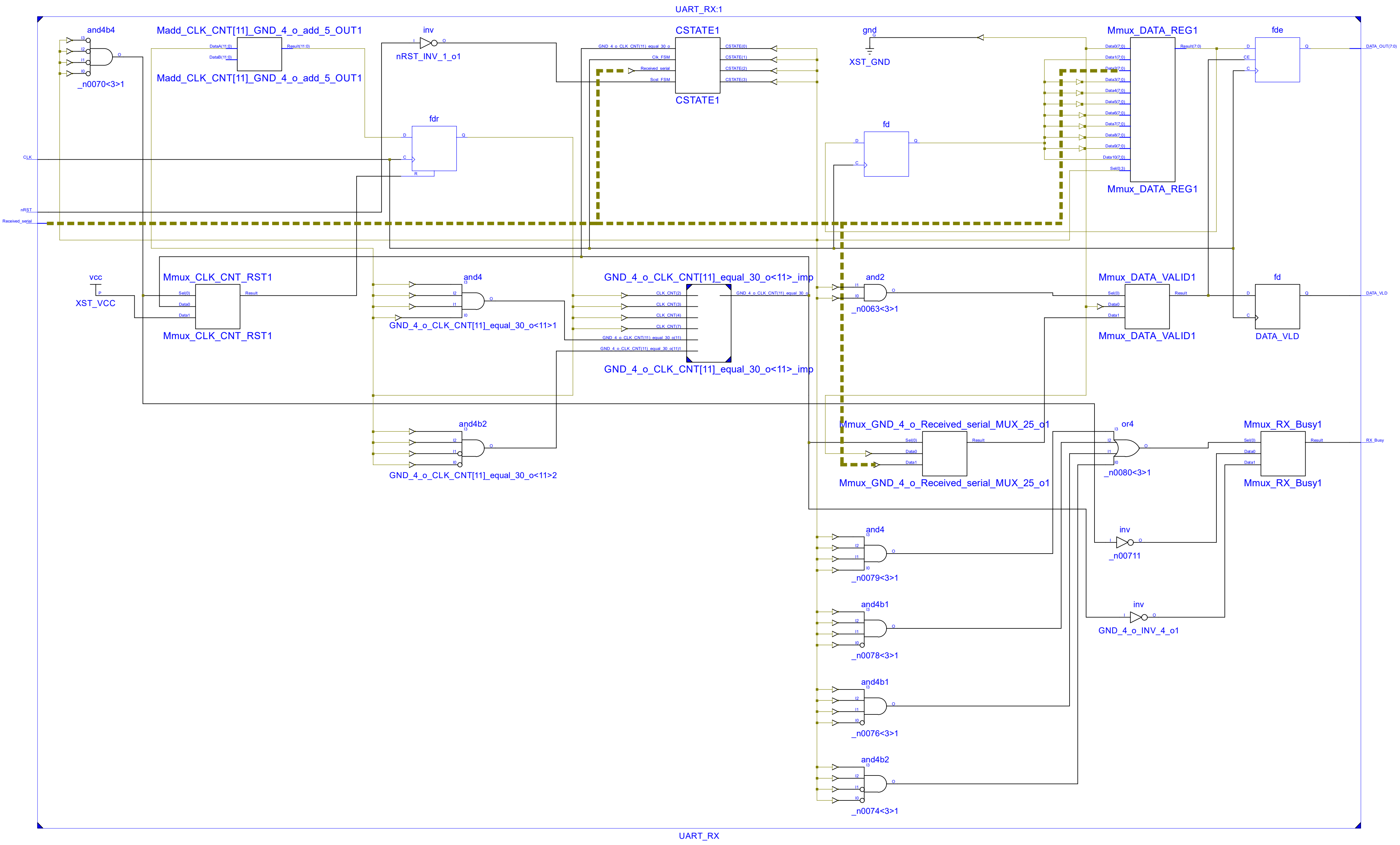
Simulation:


104.804340 us
Name
Value
clk
nrst
received_serial
data_out[7:0]
data_vld
rx_busy
clk_time
data_reg[7:0]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
0
1
1
10101010
0
0
5000 ps
00000000
40 us
60 us
80 us
100 us
120 us
140 us
160 us
180 us
200 us
220 us
UUUUUUUU
10101010
01010101
5000 ps
00000000
00000001
01010101

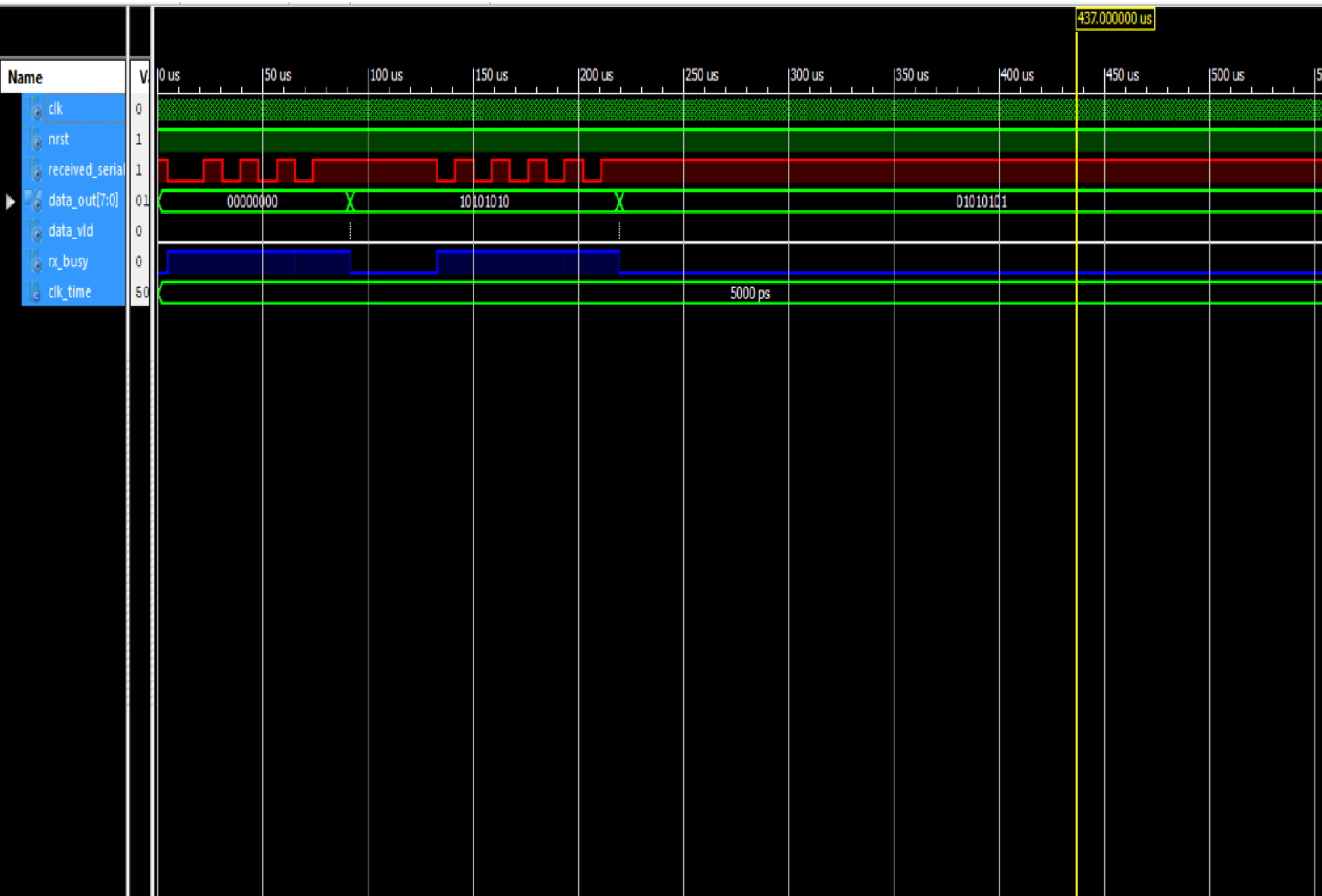
Synthesize & RTL Schematic:



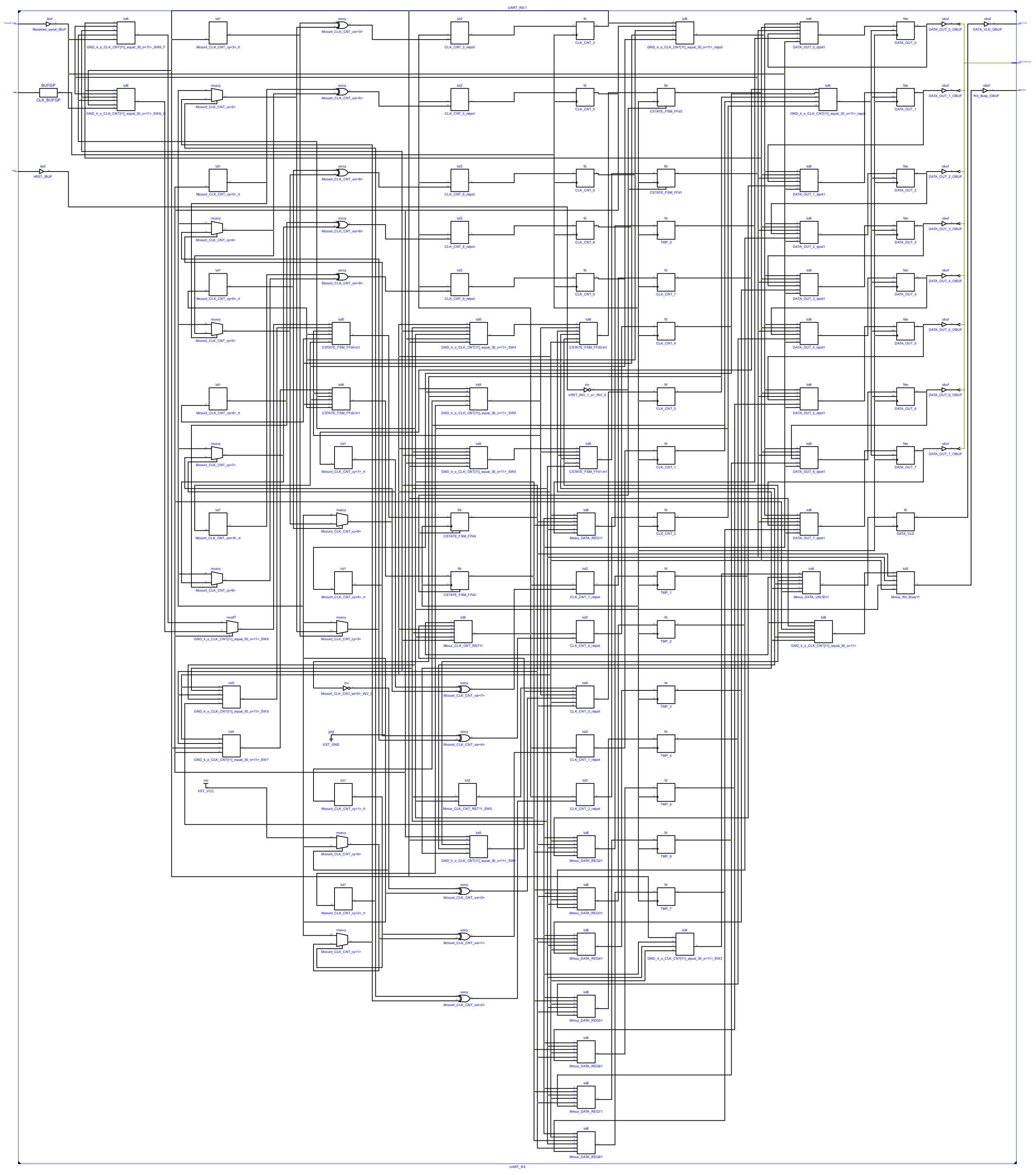
Synthesize & RTL Schematic:



Post-Route Simulation:



Technology Schematic:



UART_RX:1
UART_RX