Sending Data from Basys3 to MATLAB using UART

VHDL Code:

There are three modules

1) Top Wrapper
2) Uart Module
3) Block Memory Generator IP

1) Top Wrapper

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

entity top_wrapper is
   generic (
      N : integer := 4  -- Default value for the constant
    );
   Port (
      clock_in    : in  std_logic;
      reset_in    : in  std_logic;
      start_in    : in  std_logic;
      txd_out     : out std_logic
   );
end top_wrapper;

architecture Behavioral of top_wrapper is
   component memory_data
    port (
      clka : IN STD_LOGIC;
      ena : IN STD_LOGIC;
      wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      addra : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0);
      dina : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      douta : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
    );
   end component;

   component uart_basys
   generic (
      N : integer
   );
```

```
    port (
      clock_in    : in  std_logic;
      reset_in    : in  std_logic;
      start_in    : in  std_logic;
      mem_data_in : in  std_logic_vector(31 downto 0);
      txd_out     : out std_logic := '0';
      ready_out   : out std_logic;
      mem_addr_out: out std_logic_vector(N-1 downto 0):= (others => '0')
    );
    end component;

    signal wea : std_logic_vector(0 downto 0) := (others => '0');
    signal mem_data_in : std_logic_vector(31 downto 0) := (others => '0');
    signal ready_out : std_logic := '0';
    signal mem_addr_out : std_logic_vector(N-1 downto 0) := (others => '0');
    signal data_in : std_logic_vector(31 downto 0) := (others => '0');
begin
       uut : memory_data
         PORT MAP (
           clka => clock_in,
           ena => '1',
           wea => wea,
           addra => mem_addr_out,
           dina => data_in,
           douta => mem_data_in
         );

         uut2 : uart_basys
         generic map(
           N  => N
         )
         port map (
           clock_in => clock_in,
           reset_in => reset_in,
           start_in => start_in,
           mem_data_in => mem_data_in,
           txd_out => txd_out,
           ready_out => ready_out,
           mem_addr_out => mem_addr_out
         );


end Behavioral;
```

2) Uart Module

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

entity uart_basys is
   generic (
      N : integer := 4  -- Default value for the constant
    );
   Port (
      clock_in    : in  std_logic;
      reset_in    : in  std_logic;
      start_in    : in  std_logic;
      mem_data_in : in  std_logic_vector(31 downto 0):= (others => '0');
      txd_out     : out std_logic;
      ready_out   : out std_logic;
      mem_addr_out: out std_logic_vector(N-1 downto 0):= (others => '0')
   );
end uart_basys;

architecture Behavioral of uart_basys is

   -- State definitions for the FSM
   type state_type is (IDLE, START, DATA, STOP, DONE);
   signal state, next_state : state_type;

   -- Signals for baud rate generation
   constant CLK_FREQ : integer := 100000000; -- 100 MHz
   constant BAUD_RATE : integer := 115200;
   constant BAUD_COUNT : integer := CLK_FREQ / BAUD_RATE;
   signal baud_counter : integer := 0;
   signal baud_tick : std_logic := '0';

   -- Signals for data transmission
   signal tx_data : std_logic_vector(7 downto 0) := (others => '0');
   signal tx_shift_reg : std_logic_vector(9 downto 0); -- 1 start, 8 data, 1 stop
   signal bit_counter : integer range 0 to 9 := 0;

   signal transmitting : std_logic := '0';
   signal start_flag : std_logic := '0';
   signal stop_flag : std_logic := '0';

   signal debug_counter : integer range 0 to 100 := 0;
   signal data_counter : integer range 0 to 16 := 0;
   signal wea : std_logic_vector(0 downto 0):= (others => '0');
   signal data_in : std_logic_vector(31 downto 0) := (others => '0');
--    signal mem_data_in : std_logic_vector(31 downto 0) := (others => '0');
```

```
   signal pipelinestart : std_logic := '0';
   signal index_counter : integer range 0 to 11 := 0;

   -- ROM address for memory read
   signal mem_addr : std_logic_vector(N-1 downto 0) := (others => '0');

   -- Signals for start and stop codes
   signal start_code : std_logic_vector(31 downto 0) := x"55AACC03";
   signal stop_code  : std_logic_vector(31 downto 0) := x"AA5503CC";

begin
   process(clock_in,baud_tick)
      begin
         if reset_in = '1' then
            state <= IDLE;
         else
            if rising_edge(clock_in) then
               state <= next_state;
               case state is
                  when IDLE =>
                     mem_addr_out <= mem_addr;
                     ready_out <= '1';
                     stop_flag <= '0';
                     ready_out <= '1';
                     if start_flag = '1' then
                        next_state <= START;  -- Load start sequence
                        index_counter <= 0;
                     else
                        next_state <= IDLE;
                     end if;
                  when START =>
                     mem_addr_out <= mem_addr;
                     ready_out <= '0';
--                     start_state_flag <= '1';
                     case index_counter is
                        when 0 =>
                           tx_data <= start_code(31 downto 24);
                           index_counter <= 1;
                        when 1 =>
                            tx_shift_reg <= '0' & tx_data & '1';
                            index_counter <= 2;
                        when 2 =>
                              if baud_tick = '1' then
                                 if bit_counter = 9 then
                                    bit_counter <= 0;
                                    index_counter <= 3;
                                 else
```

```
                        txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                        tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                        bit_counter <= bit_counter + 1;
                    end if;
                end if;
            when 3 =>
              tx_data <= start_code(23 downto 16);
              index_counter <= 4;
            when 4 =>
              tx_shift_reg <= '0' & tx_data & '1';
              index_counter <= 5;
            when 5 =>
--              debug_counter <= debug_counter + 1;
              if baud_tick = '1' then
                if bit_counter = 9 then
                  bit_counter <= 0;
                  index_counter <= 6;
                else
                  txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                  tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                  bit_counter <= bit_counter + 1;
                end if;
              end if;
            when 6 =>
                tx_data <= start_code(15 downto 8);
                index_counter <= 7;
            when 7 =>
                tx_shift_reg <= '0' & tx_data & '1';
                index_counter <= 8;
            when 8 =>
              if baud_tick = '1' then
                if bit_counter = 9 then
                  bit_counter <= 0;
                  index_counter <= 9;
                else
                  txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                  tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                  bit_counter <= bit_counter + 1;
                end if;
              end if;
            when 9 =>
                tx_data <= start_code(7 downto 0);
                index_counter <= 10;
            when 10 =>
                tx_shift_reg <= '0' & tx_data & '1';
                index_counter <= 11;
            when 11 =>
              if baud_tick = '1' then
```

```
                    if bit_counter = 9 then
                       bit_counter <= 0;
                       index_counter <= 0;
                       next_state <= DATA;
                    else
                       txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                       tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                       bit_counter <= bit_counter + 1;
                    end if;
                 end if;
               when others =>
                 index_counter <= 0;

         end case;

      when DATA =>
         mem_addr_out <= mem_addr;
         case data_counter is
            when 0 =>
               mem_addr <= (others => '0');
               data_counter <= 1;
            when 1 =>
               data_counter <= 15;
            when 15 =>
               tx_data <= mem_data_in(31 downto 24);
               data_counter <= 2;
            when 2 =>
               tx_shift_reg <= '0' & tx_data & '1';
               data_counter <= 3;
            when 3 =>
               if baud_tick = '1' then
                     if bit_counter = 9 then
                        bit_counter <= 0;
                        data_counter <= 4;
                     else
                        txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                        tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                        bit_counter <= bit_counter + 1;
                     end if;
                  end if;
               when 4 =>
                 tx_data <= mem_data_in(23 downto 16);
                 data_counter <= 5;
               when 5 =>
                 tx_shift_reg <= '0' & tx_data & '1';
                 data_counter <= 6;
               when 6 =>
                 if baud_tick = '1' then
```

```
            if bit_counter = 9 then
               bit_counter <= 0;
               data_counter <= 7;
            else
               txd_out <= tx_shift_reg(9);  -- Transmit LSB first
               tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
               bit_counter <= bit_counter + 1;
            end if;
         end if;
      when 7 =>
         tx_data <= mem_data_in(15 downto 8);
         data_counter <= 8;
      when 8 =>
         tx_shift_reg <= '0' & tx_data & '1';
         data_counter <= 9;
      when 9 =>
         if baud_tick = '1' then
            if bit_counter = 9 then
               bit_counter <= 0;
               data_counter <= 10;
            else
               txd_out <= tx_shift_reg(9);  -- Transmit LSB first
               tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
               bit_counter <= bit_counter + 1;
            end if;
         end if;
       when 10 =>
         tx_data <= mem_data_in(7 downto 0);
         data_counter <= 11;
       when 11 =>
         tx_shift_reg <= '0' & tx_data & '1';
         data_counter <= 12;
       when 12 =>
         if baud_tick = '1' then
            if bit_counter = 9 then
               bit_counter <= 0;
               data_counter <= 13;
            else
               txd_out <= tx_shift_reg(9);  -- Transmit LSB first
               tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
               bit_counter <= bit_counter + 1;
            end if;
         end if;
        when 13 =>
         if mem_addr = (2**N)-1 then
            next_state <= STOP;
            index_counter <= 0;
            mem_addr <= x"0";
```

```
                        else
                           mem_addr <= mem_addr + 1;
                           data_counter <= 14;
                        end if;
                     when 14 =>
                        data_counter <= 1;
                  when others =>
                     data_counter <= 0;
               end case;


--                next_state <= STOP;
--                index_counter <= 0;

            when STOP =>
--                start_state_flag <= '1';
               case index_counter is
                  when 0 =>
                     tx_data <= stop_code(31 downto 24);
                     index_counter <= 1;
                  when 1 =>
                      tx_shift_reg <= '0' & tx_data & '1';
                      index_counter <= 2;
                  when 2 =>
                        if baud_tick = '1' then
                           if bit_counter = 9 then
                              bit_counter <= 0;
                              index_counter <= 3;
                           else
                              txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                              tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                              bit_counter <= bit_counter + 1;
                           end if;
                        end if;
                  when 3 =>
                     tx_data <= stop_code(23 downto 16);
                     index_counter <= 4;
                  when 4 =>
                     tx_shift_reg <= '0' & tx_data & '1';
                     index_counter <= 5;
                  when 5 =>
--                      debug_counter <= debug_counter + 1;
                     if baud_tick = '1' then
                        if bit_counter = 9 then
                           bit_counter <= 0;
                           index_counter <= 6;
                        else
                           txd_out <= tx_shift_reg(9);  -- Transmit LSB first
```

```
                        tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                        bit_counter <= bit_counter + 1;
                    end if;
                end if;
            when 6 =>
                tx_data <= stop_code(15 downto 8);
                index_counter <= 7;
            when 7 =>
                tx_shift_reg <= '0' & tx_data & '1';
                index_counter <= 8;
            when 8 =>
             if baud_tick = '1' then
                if bit_counter = 9 then
                    bit_counter <= 0;
                    index_counter <= 9;
                else
                    txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                    tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                    bit_counter <= bit_counter + 1;
                end if;
             end if;
            when 9 =>
                tx_data <= stop_code(7 downto 0);
                index_counter <= 10;
            when 10 =>
                tx_shift_reg <= '0' & tx_data & '1';
                index_counter <= 11;
            when 11 =>
             if baud_tick = '1' then
                if bit_counter = 9 then
                    bit_counter <= 0;
                    index_counter <= 0;
                    next_state <= IDLE;
                    mem_addr <= x"0";
                    stop_flag <= '1';
                else
                    txd_out <= tx_shift_reg(9);  -- Transmit LSB first
                    tx_shift_reg <= tx_shift_reg(8 downto 0) & '0';  -- Shift data
                    bit_counter <= bit_counter + 1;
                end if;
             end if;
            when others =>
             index_counter <= 0;
        end case;

    when others =>
        state <= IDLE;
end case;
```

```
            end if;
        end if;

    end process;

  process(clock_in)
    begin
      if rising_edge(clock_in) and (start_flag = '1') then
        if baud_counter = BAUD_COUNT - 1 then
          baud_counter <= 0;
          baud_tick <= '1';
        else
          baud_counter <= baud_counter + 1;
          baud_tick <= '0';
        end if;
      end if;
    end process;

  -- done flag to reset start_flag is required to be implemented later
  process(clock_in)
    begin
      if reset_in = '1' then
        start_flag <= '0';
      else
        if start_in = '1' then
          start_flag <= '1';
        else
          if stop_flag = '1' then
            start_flag <= '0';
          else
            start_flag <= start_flag;
          end if;
        end if;
      end if;
    end process;

end Behavioral;
```

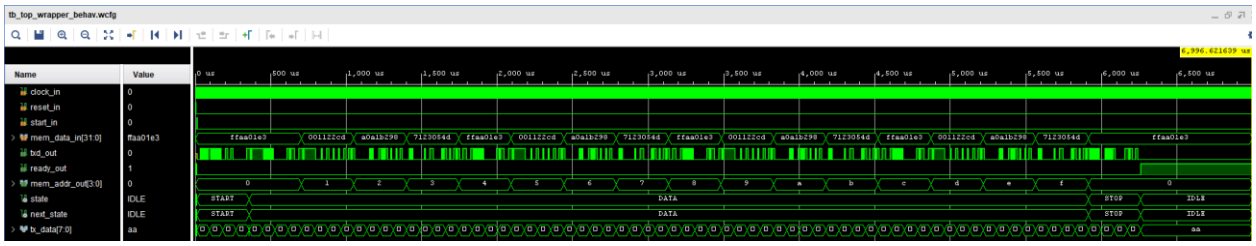
3) Bram Generator is an IP. Its coefficient file is as follows:

```
memory_initialization_radix = 16;
memory_initialization_vector = FFAA01E3,
001122CD,
A0A1B298,
7123054D,
FFAA01E3,
001122CD,
```

A0A1B298,
7123054D,
FFAA01E3,
001122CD,
A0A1B298,
7123054D,
FFAA01E3,
001122CD,
A0A1B298,
7123054D;

Simulation Results:

Full View



Zoomed in View