**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
4. **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; |

1. **Uart Module**

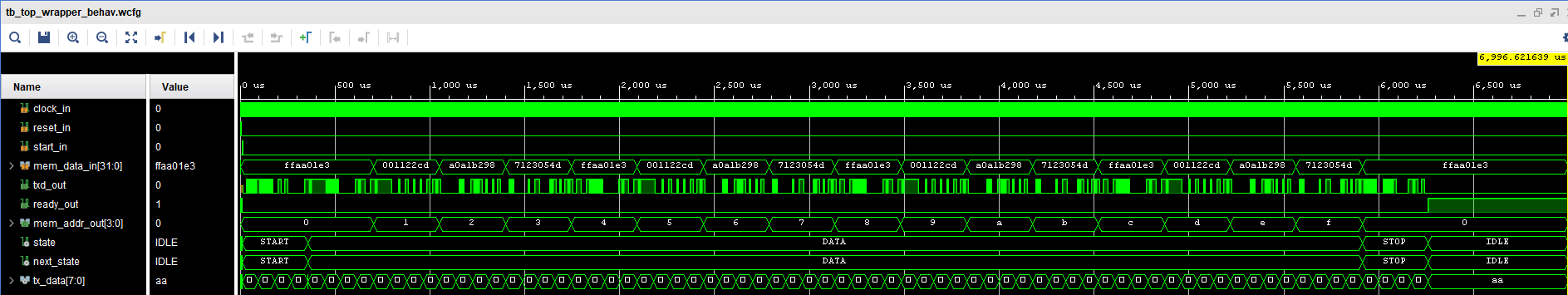
|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.STD\_LOGIC\_ARITH.ALL;  use IEEE.STD\_LOGIC\_UNSIGNED.ALL;  entity uart\_basys is  generic (  N : integer := 4 -- Default value for the constant  );  Port (  clock\_in : in std\_logic;  reset\_in : in std\_logic;  start\_in : in std\_logic;  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; |

1. **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

