Sequential Circuit Design using VHDL and FPGAs
Project 4
Serial Asynchronous Data Transmission (SADT)

Project Overview

This project describes how to implement a digital system in order to transfer serially data from the FPGA board to a PC or any other device that includes a UART module [1], [2].

The `sadt` module of this project consists of three submodules :

1. A 16x8 bit `prom`.
2. A serial asynchronous `transmitter`.
3. A `control` module.

Figure 1 shows the structure of the `sadt` module.

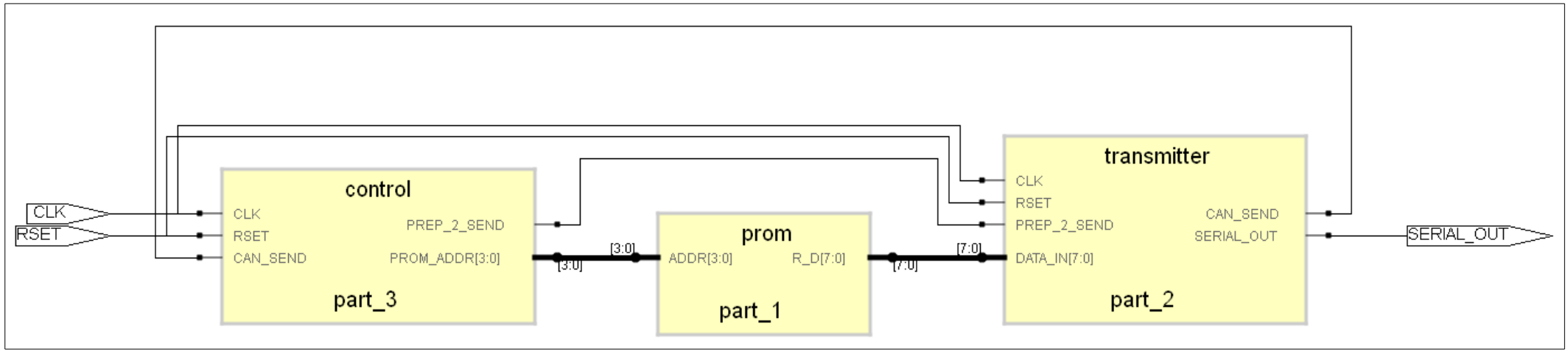

Figure 1

The serial asynchronous `transmitter` is a sequential circuit that transmits data bits serially. This circuit can send data to any device that includes a UART module. The `prom` holds the data to be sent. The function of the `sadt` module is simple and it is summarized in the simplified Moore state diagram (**Figure 2**) of the `control` module state machine.

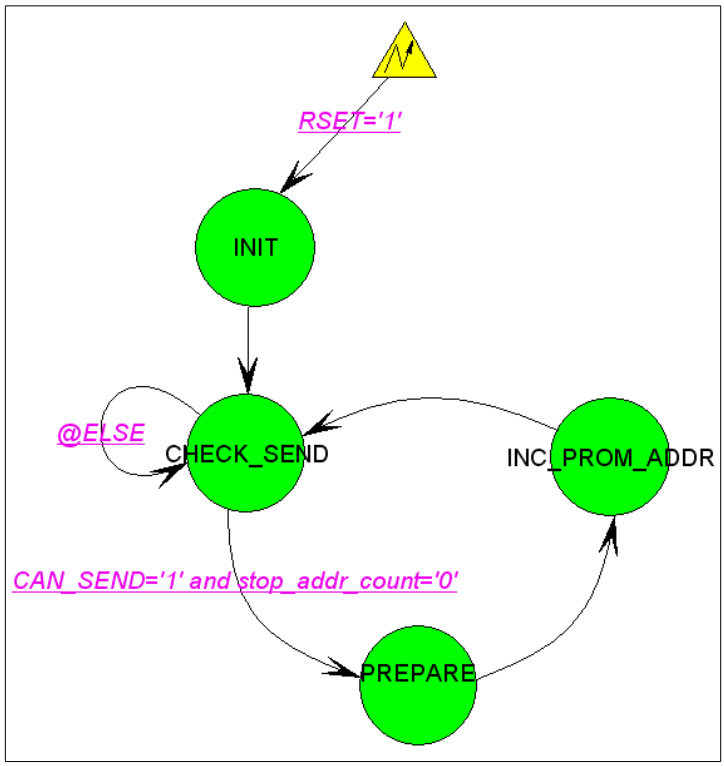

Figure 2

When the user presses the RSET button (`RSET='1'`), the state machine moves asynchronously to the `INIT` state. At the `INIT` state the `control` module initializes internal signals. At the next rising edge of the clock the state machine moves to the `CHECK_SEND` state. If the `transmitter` is not sending data and the address of the last `prom` word to be sent is not yet reached (`CAN_SEND='1' and stop_addr_count='0'`), the state machine moves to the `PREPARE` state, else it stays at the `CHECK_SEND` state. At the `PREPARE` state the `control` module informs the `transmitter` to prepare to send (using the `PREP_2_SEND` signal) the data that the `prom` holds at address 0. At the next rising edge of the clock the state machine moves to the `INC_PROM_ADDR` state where the `control` module increases by one the `prom` address (see **Figure 1**). Finally, at the next rising edge of the clock the state machine returns to the `CHECK_SEND` state.

Design of the Serial Asynchronous Data Transmission system

The following pages will demonstrate how to implement the Serial Asynchronous Data Transmission system using Xilinx ISE 13.3, VHDL and CORE Generator system as design entry, ChipScope Pro tool for virtual input (ICON and VIO cores), and Xilinx Evaluation boards.

1. Open Xilinx ISE :

Double click the ISE desktop icon. Alternatively open a new terminal and give the command "`./.bin/ise`" (without the quotes). For remote users, open a new terminal and give the following commands "`export DISPLAY=:1`" and "`./.remote/ise`" (without the quotes).

2. Create a new Xilinx ISE Project :

In the ISE Project Navigator window, select `File->New Project...`. In the New Project Wizard window, type `Project4src` in the Name field and `/home/fpga-user/Documents/Project4src` in the Location field and click Next (see **Figure 3**).

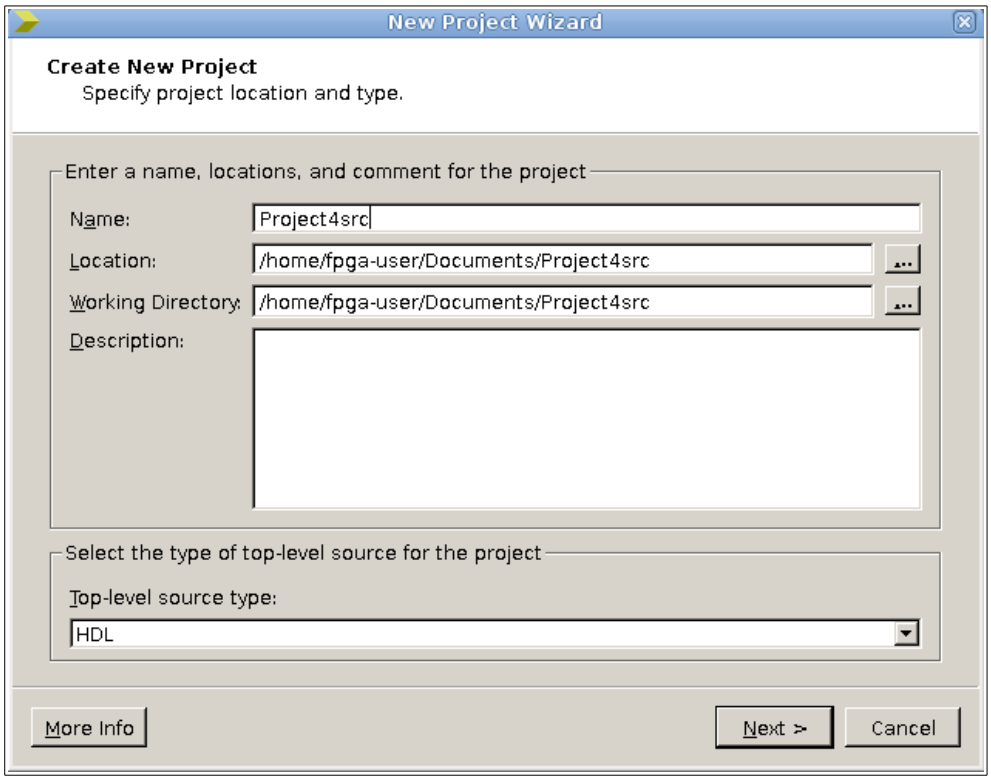

Figure 3

In the next New Project Wizard window, select the target Xilinx FPGA Evaluation board (e.g. the Spartan-6 SP605 Evaluation Platform) in the Evaluation Development Board field, select VHDL in the Preferred Language field and click Next. The last New Project Wizard window must be similar to the one shown in **Figure 4**. Click Finish to create the project `Project4src` or navigate to the previous windows using Back to make corrections.

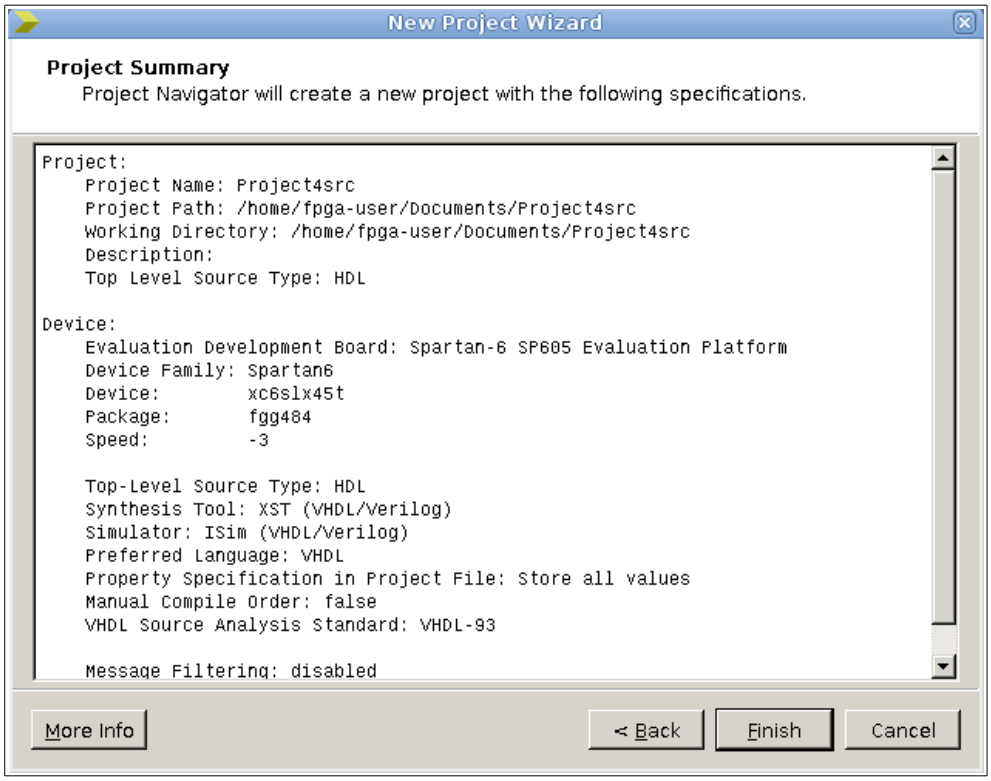

Figure 4

3. Add New Source (`transmitter`) in the Project4src project :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `transmitter` in the File Name field, select VHDL Module and click Next (see **Figure 5**). The next New Source Wizard window must be completed as shown in **Figure 6**.

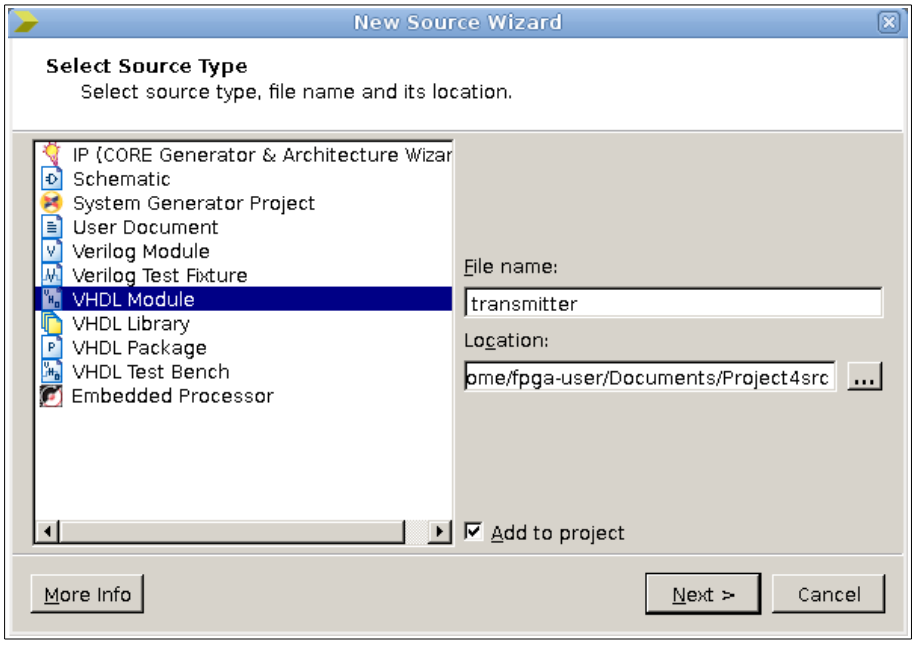

Figure 5

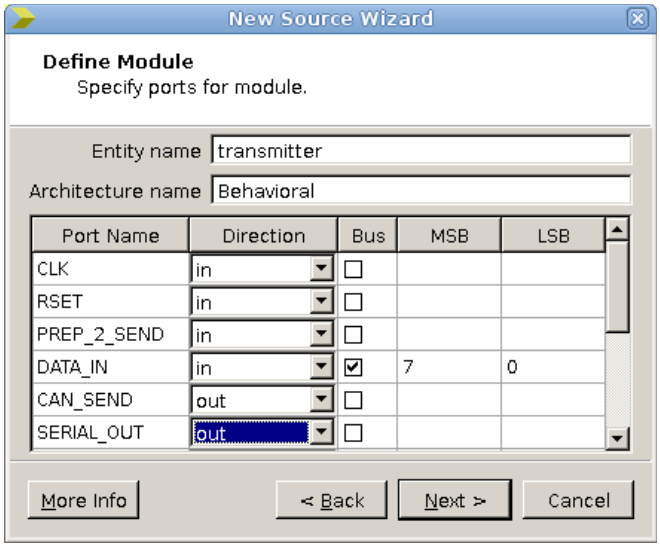

Figure 6

Clicking Next and Finish, the file `transmitter.vhd` is created.

4. Define the behavior of the `transmitter` VHDL module :

Remove the green highlighted lines (the comments) and edit the `transmitter.vhd` file to look like the one shown in **Figure 7**. The `transmitter` entity consists of two counters, a state machine and a shift register.

The counter described in lines `32-42` counts clock signal (`CLK`) cycles, from `0` up to the `CLK_DIV` value. This counter, tagged with `shift_clk_en :`, is used for frequency division (clock generation). When the `shift_clk_en` counter reaches the `CLK_DIV` value the `shift_en` signal becomes `'1'` and enables the shift register to shift data (line `44` – see also lines `130-131`). In this way the data of the shift register are shifted by one bit every `CLK_DIV + 1` cycles of the `CLK` signal. The above-mentioned method uses the clock enable input port of the shift register to enable the clock input port once every `CLK_DIV + 1` cycles of the `CLK` signal. Alternative methods used for clock generation that use FPGA logic resources to drive the clock input port of the shift register are considered as a bad design practice.

The counter tagged with the `counter :` label, described in lines `47-57`, is used to count the bits that are shifted out of the shift register.

The outputs of the above-mentioned counters (signals `count` and `counter_out`) are used as inputs to the Moore FSM described in lines `60-116` (see line `78`).

The state diagram that represents the `next_state_and_state_register` process (lines `60-85`) is shown in **Figure 8**.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity transmitter is
6      generic(CLK_DIV  : integer := 233);
7      Port(CLK,RSET    : in  STD_LOGIC;
8           PREP_2_SEND : in  STD_LOGIC;
9           DATA_IN     : in  STD_LOGIC_VECTOR(7 downto 0);
10          CAN_SEND    : out STD_LOGIC;
11          SERIAL_OUT  : out STD_LOGIC);
12  end transmitter;
13
14  architecture Behavioral of transmitter is
15     signal shift_clk_en_rset : std_logic;
16     signal count             : integer range 0 to CLK_DIV;
17     signal shift_en          : std_logic;
18
19     signal count_rset        : std_logic;
20     signal counter_out       : integer range 0 to 9;
21
22     type state_type is (INIT,STANDBY,LOAD_DATA,SENDING);
23     signal state             : state_type;
24
25     signal cansend           : std_logic;
26
27     signal shift_reg         : STD_LOGIC_VECTOR(8 downto 0);
28     signal shift_rset        : std_logic;
29     signal shift_load        : std_logic;
30  begin
31     -- shift register clock enable :
32     shift_clk_en :
33     process(CLK)
34     begin
35        if rising_edge(CLK) then
36           if shift_clk_en_rset = '1' or count = CLK_DIV then
37              count <= 0;
38           else
39              count <= count + 1;
40           end if;
41        end if;
42     end process shift_clk_en;
43
44     shift_en <= '1' when count = CLK_DIV else '0';
45
46     -- 0 to 9 counter :
47     counter :
48     process(CLK,shift_en)
49     begin
50        if rising_edge(CLK) and shift_en = '1'  then
51           if count_rset = '1' or counter_out = 9 then
52              counter_out <= 0;
53           else
54              counter_out <= counter_out + 1;
55           end if;
56        end if;
57     end process counter;
58
59     -- Moore type FSM :
60     next_state_and_state_register :
61     process(CLK,RSET)
62     begin
63        if RSET = '1' then
64           state <= INIT;
65        elsif rising_edge(CLK) then
66           case state is
67              when INIT =>
68                 state <= STANDBY;
69              when STANDBY =>
70                 if PREP_2_SEND = '1' then
71                    state <= LOAD_DATA;
72                 else
73                    state <= STANDBY;
74                 end if;
75              when LOAD_DATA =>
76                 state <= SENDING;
77              when SENDING =>
78                 if counter_out = 9 and count = CLK_DIV then
79                    state <= STANDBY;
80                 else
81                    state <= SENDING;
82                 end if;
83           end case;
84        end if;
85     end process next_state_and_state_register;
86
87     output_calculation :
88     process(state)
89     begin
90        case state is
91           when INIT =>
92              shift_clk_en_rset <= '1';
93              count_rset        <= '1';
94              shift_rset        <= '1';
94              shift_rset        <= '1';
95              shift_load        <= '0';
96              cansend           <= '0';
97           when STANDBY =>
98              shift_clk_en_rset <= '1';
99              count_rset        <= '1';
100             shift_rset        <= '1';
101             shift_load        <= '0';
102             cansend           <= '1';
103          when LOAD_DATA =>
104             shift_clk_en_rset <= '1';
105             count_rset        <= '1';
106             shift_rset        <= '0';
107             shift_load        <= '1';
108             cansend           <= '0';
109          when SENDING =>
110             shift_clk_en_rset <= '0';
111             count_rset        <= '0';
112             shift_rset        <= '0';
113             shift_load        <= '0';
114             cansend           <= '0';
115        end case;
116     end process output_calculation;
117
118     -- READY TO SEND NEXT WORD OUTPUT SIGNAL :
119     CAN_SEND <= cansend;
120
121     -- Shift Register :
122     shift_register :
123     process(CLK)
124     begin
125        if rising_edge(CLK) then
126           if shift_rset = '1' then
127              shift_reg <= (OTHERS => '1');
128           elsif shift_load = '1' then
129              shift_reg <= DATA_IN & '0';
130           elsif shift_en = '1' then
131              shift_reg <= '1' & shift_reg(8 downto 1);
132           end if;
133        end if;
134     end process shift_register;
135
136     SERIAL_OUT <= shift_reg(0);
137  end Behavioral;
```

Figure 7

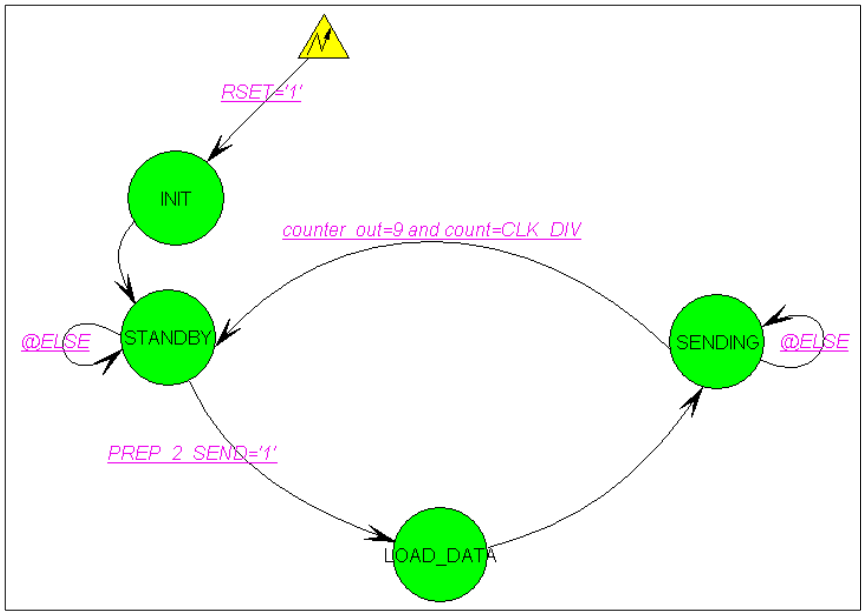

Figure 8

When the user presses the RSET button (`RSET='1'`), the Moore FSM moves asynchronously to the `INIT` state. At the `INIT` state the FSM initializes signals (see **Figure 7**, lines `92-96`). At the next rising edge of the clock the FSM moves to the `STANDBY` state. At this state the FSM informs the module that will control the `transmitter` module (the `control` module, see **Figure 1**) that the `transmitter` can send data – `cansend <= '1'` (see **Figure 7**, lines `98-102` and line `119`). At the rising edge of the clock when the `PREP_2_SEND` signal value is `'1'` the FSM moves to the `LOAD_DATA` state. At the next rising edge of the clock the 8-bit word to be send (`DATA_IN`) and a `'0'` are loaded parallely in the shift register – `shift_load <= '1'` (see **Figure 7**, lines `104-108` and lines `128-129`), and the FSM moves to the `SENDING` state. At this state the shift register shifts out 10 bits of data : 1 start bit – `'0'`, 8 data bits (`DATA_IN`) – from the lsb to msb, and 1 stop bit – `'1'` (see **Figure 7**, lines `77-82` and lines `110-114`). At the next rising edge of the clock when the `counter` counter output value is equal to `9` and the `shift_clk_en` counter output is equal to `CLK_DIV` the FSM returns to the `STANDBY` state (see **Figure 8**). So after the reset (`RSET='1'`) when the input port `PREP_2_SEND` of the transmitter becomes `'1'`, the transmitter loads the data from the `DATA_IN` port, frames that data with a `'0'` (start bit) and a `'1'` (stop bit) and sends the 10 bits bit by bit every `CLK_DIV+1` clock (`CLK`) cycles.

In the ISE Project Navigator window select Implementation (in the View subwindow), select the `transmitter` module, and double-click Synthesize - XST (in the Processes subwindow) to verify that the `transmitter`'s VHDL syntax is correct and synthesizable.

5. Behavioral simulation of the `transmitter` module :

In the ISE Project Navigator window select `Project->New Source...`. In the New Source Wizard window type `transmitter_tb` in the File Name field, select VHDL Test Bench and click Next. In the next New Source Wizard window associate the `transmitter_tb` file with the `transmitter` module, click Next and then click Finish to

create the `transmitter_tb.vhd` testbench file. Modify the `transmitter_tb.vhd` testbench file to look like the one in **Figure 9**.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY transmitter_tb IS
5  END transmitter_tb;
6
7  ARCHITECTURE behavior OF transmitter_tb IS
8      -- Component Declaration for the Unit Under Test (UUT)
9      COMPONENT transmitter
10     PORT(
11          CLK : IN  std_logic;
12          RSET : IN  std_logic;
13          PREP_2_SEND : IN  std_logic;
14          DATA_IN : IN  std_logic_vector(7 downto 0);
15          CAN_SEND : OUT  std_logic;
16          SERIAL_OUT : OUT  std_logic
17         );
18     END COMPONENT;
19    --Inputs
20    signal CLK : std_logic := '0';
21    signal RSET : std_logic := '0';
22    signal PREP_2_SEND : std_logic := '0';
23    signal DATA_IN : std_logic_vector(7 downto 0) := (others => '0');
24    --Outputs
25    signal CAN_SEND : std_logic;
26    signal SERIAL_OUT : std_logic;
27    -- Clock period definition
28    constant CLK_period : time := 37 ns;
29 BEGIN
30    -- Instantiate the Unit Under Test (UUT)
31    uut: transmitter PORT MAP (
32           CLK => CLK,
33           RSET => RSET,
34           PREP_2_SEND => PREP_2_SEND,
35           DATA_IN => DATA_IN,
36           CAN_SEND => CAN_SEND,
37           SERIAL_OUT => SERIAL_OUT
38          );
39    -- Clock process definitions
40    CLK_process :process
41    begin
42       CLK <= '0';
43       wait for CLK_period/2;
44       CLK <= '1';
45       wait for CLK_period/2;
46    end process;
47    -- Stimulus process
48    stim_proc: process
49    begin
50       RSET        <= '0';
51       PREP_2_SEND <= '0';
52       DATA_IN     <= X"61";
53       wait for CLK_period*5;
54       RSET        <= '1';
55       PREP_2_SEND <= '0';
56       wait for CLK_period*5;
57       RSET        <= '0';
58       PREP_2_SEND <= '0';
59       wait for CLK_period*5;
60       RSET        <= '0';
61       PREP_2_SEND <= '1';
62       wait for CLK_period*5;
63       RSET        <= '0';
64       PREP_2_SEND <= '0';
65       wait;
66    end process;
67 END;
```

Figure 9

The statement in line `28` (**Figure 9**) defines the `CLK_period`, a `constant` of type `time` that is equal to `37 ns`. The value of the `CLK_period` is defined as equal to `37 ns` because the real clock period that will be used during implementation is equal to 37,037 ns (27 MHz). The `CLK_process` process (lines `40-46`, **Figure 9**) is used for the `CLK` signal definition.

In the ISE Project Navigator window select Simulation (in the View subwindow), select the `transmitter_tb` testbench file, expand ISim Simulator (in the Processes subwindow) and double-click Behavioral Check Syntax to check `transmitter_tb` testbench's syntax.

Right-click Simulate Behavioral Model (in the Processes subwindow) and select Process Properties. Change the Simulation Run Time to 100000 ns and click OK. Now double-click Simulate Behavioral Model to run the simulation as specified by the `transmitter_tb` testbench file and the ISim Simulation Run Time parameter. Select `View->Zoom->To Full View` in the ISim simulator window. **Figure 10** shows the simulation waveform. Markers indicate the transmitted bits (every bit transmission lasts for 234x37 ns = 8658 ns). The transmission starts with the start bit (`'0'`), continues with the 8 data bits (from the lsb to the msb – the `data_in` value in **Figure 10** is `01100001`), and ends with the stop bit (`'1'`). So the bit sequence that is transmitted is `'0'`, `'1'`, `'0'`, `'0'`, `'0'`, `'0'`, `'1'`, `'1'`, `'0'`, `'1'`.

In the ISim simulator window, in the Instances and Processes subwindow expand the `transmitter_tb` (double-click on it or right-click on it and select `Expand`) and select the `uut`. In the Objects subwindow right-click on the `state` signal and select `Add To Wave Window`. To delete the markers right-click on the waveform and select `Markers->Delete All Markers`. In the Console window give the following commands (press the enter key after every command) : `restart` and `run 100 us` in order to rerun the simulation. Zoom at the beginning and at the end of the simulation to verify the correct operation of the `transmitter`'s state machine (see **Figure 11** and **Figure 12**).

Close the ISim simulator window.

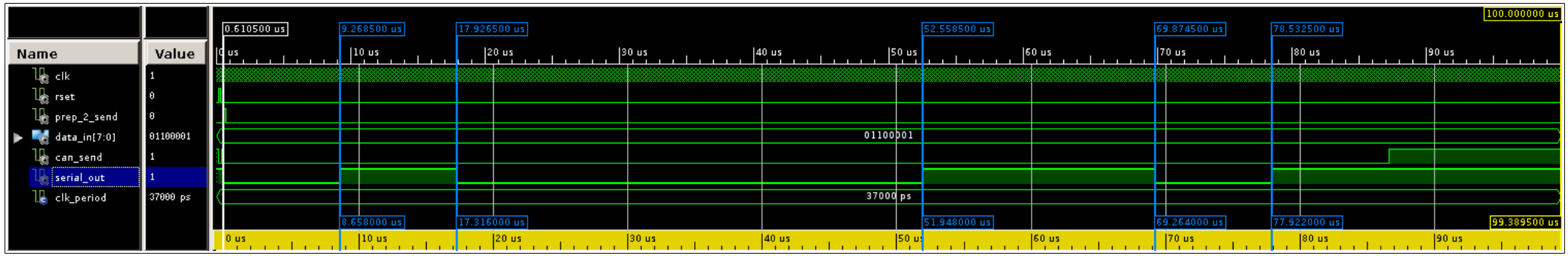

Figure 10

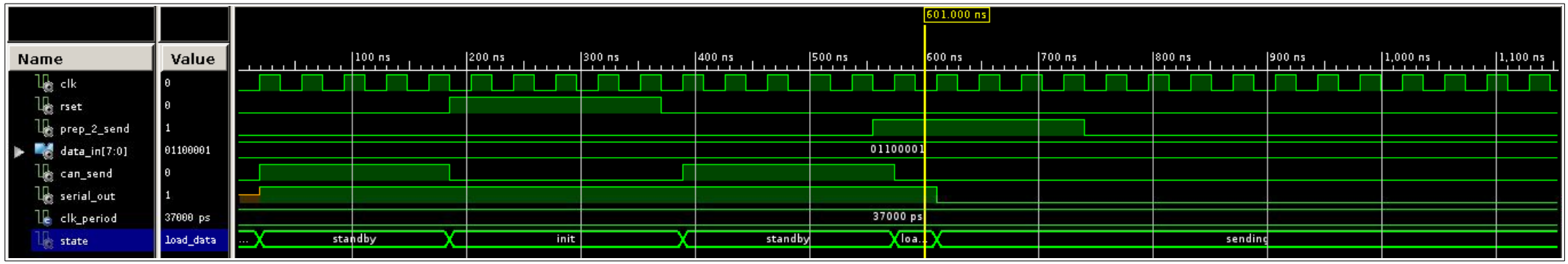

Figure 11

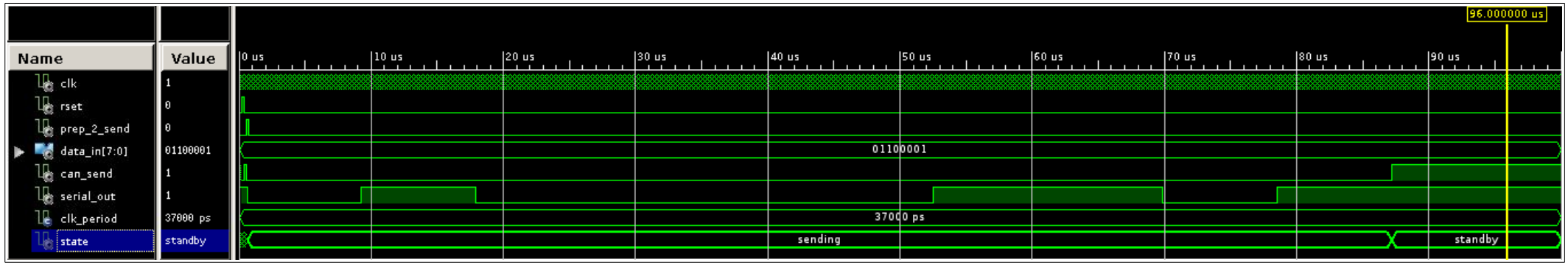

Figure 12

6. Add New Source (`prom`) in the Project4src project :

In the ISE Project Navigator window select Implementation (in the View subwindow) and select `Project->New Source...`. In the New Source Wizard window type `prom` in the File Name field, select VHDL Module and click Next. The next New Source Wizard window must be completed as shown in **Figure 13**.

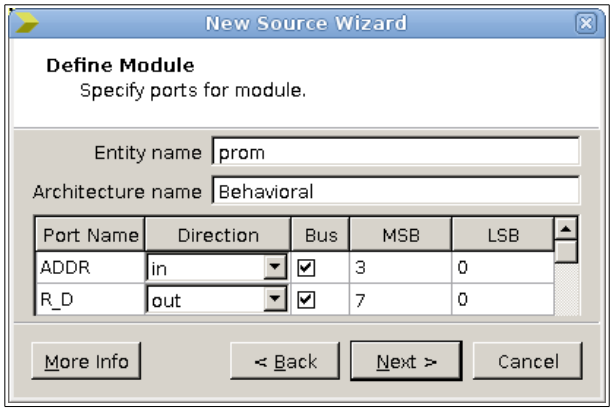

Figure 13

Clicking Next and Finish, the file `prom.vhd` is created.

7. Define the behavior of the `prom` VHDL module :

Remove the green highlighted lines (the comments) and edit the `prom.vhd` file to look like the one shown in **Figure 14**.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity prom is
6     generic(PROM_A_BITS : integer := 4);
7     port(ADDR : in std_logic_vector(PROM_A_BITS-1 downto 0);
8          R_D  : out std_logic_vector(7 downto 0));
9  end prom;
10
11 architecture behavioral of prom is
12    type RomType is array (0 to 2**PROM_A_BITS-1) of std_logic_vector(7 downto 0);
13
14    constant mem : RomType := (x"48",x"65",x"6c",x"6c",x"6f",x"20",x"57",x"6f",
15                               x"72",x"6c",x"64",x"21",x"00",x"00",x"00",x"00");
16 begin
17    R_D <= mem(to_integer(unsigned(ADDR)));
18 end behavioral;
```

Figure 14

In line `12` (**Figure 14**) the type `RomType` is defined as a linear array of $2^{PROM\_A\_BITS}$ x

`std_logic_vector(7 downto 0)`. The `2**PROM_A_BITS-1` expression means $2^{PROM\_A\_BITS-1}$. The `(0 to 2**PROM_A_BITS-1)` expression means that the first address of a RomType object is equal to 0 and the last address is equal to $2^{PROM\_A\_BITS-1}$. In lines `14-15` the constant `mem` of type `RomType` is defined. The `mem(0)` is of type `std_logic_vector(7 downto 0)` and it is equal to `X"48"`, the `mem(1)` is of type `std_logic_vector(7 downto 0)` and it is equal to `X"65"`, and so on.

The ASCII representation of the first 12 bytes stored in the `prom` is `Hello World!` (space is also considered as a character) [3].

In the ISE Project Navigator window select Implementation (in the View subwindow), select the `prom` module, right-click on the `prom` module and select `Set as Top Module` (click Yes at the Set as Top Module window) and double-click Synthesize - XST (in the Processes subwindow) to verify that the `prom`'s syntax is correct and synthesizable.

Behavioral simulation of the `prom` module is skipped because its description is very simple.

8. Add New Source (`control`) in the Project4src project :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `control` in the File Name field, select VHDL Module and click Next. The next New Source Wizard window must be completed as shown in **Figure 15**.

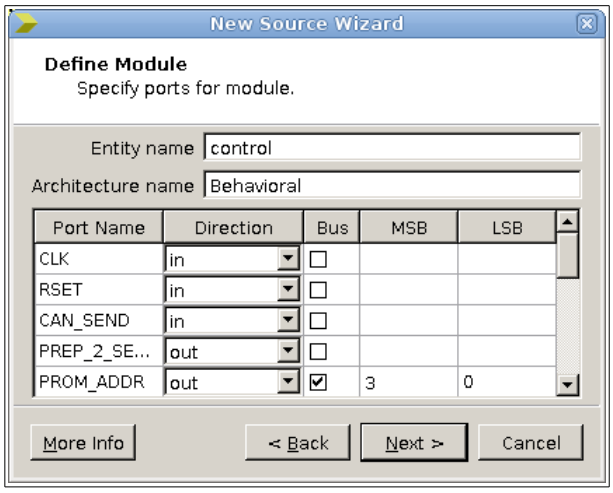

Figure 15

Clicking Next and Finish, the file `control.vhd` is created.

9. Define the behavior of the `control` VHDL module :

Remove the green highlighted lines (the comments) and edit the `control.vhd` file to look like the one shown in **Figure 16**. The `control` entity consists of a counter and a state machine.

In lines `23-33` (**Figure 16**) the `addr_counter` counter is described. The output of this counter

drives the `prom`'s `ADDR` input port (see line `83`, **Figure 16**, and **Figure 1**). This counter is used to increase the input address of the `prom` module.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity control is
6     generic(PROM_A_BITS    : integer := 4;
7             READ_UNTIL_CHR : integer := 12);
8     port(CLK,RSET     : in  STD_LOGIC;
9          CAN_SEND     : in  STD_LOGIC;
10         PREP_2_SEND  : out STD_LOGIC;
11         PROM_ADDR    : out  STD_LOGIC_VECTOR(PROM_A_BITS-1 downto 0));
12 end control;
13
14 architecture Behavioral of control is
15    type state_type is (INIT,CHECK_SEND,PREPARE,INC_PROM_ADDR);
16    signal state : state_type;
17
18    signal addr_count_rset : std_logic;
19    signal addr            : integer range 0 to 2**PROM_A_BITS-1;
20    signal addr_count_en   : std_logic;
21    signal stop_addr_count : std_logic;
22 begin
23    addr_counter :
24    process(CLK)
25    begin
26       if rising_edge(CLK) then
27          if addr_count_rset = '1' then
28             addr <= 0;
29          elsif addr_count_en = '1' then
30             addr <= addr + 1;
31          end if;
32       end if;
33    end process addr_counter;
34
35    stop_addr_count <= '1' when addr = READ_UNTIL_CHR else '0';
36
37    next_state_and_state_register :
38    process(CLK,RSET)
39    begin
40       if RSET = '1' then
41          state <= INIT;
42       elsif rising_edge(CLK) then
43          case state is
44             when INIT =>
45                state <= CHECK_SEND;
46             when CHECK_SEND =>
47                if CAN_SEND = '1' and stop_addr_count = '0' then
48                   state <= PREPARE;
49                else
50                   state <= CHECK_SEND;
51                end if;
52             when PREPARE =>
53                state <= INC_PROM_ADDR;
54             when INC_PROM_ADDR =>
55                state <= CHECK_SEND;
56          end case;
57       end if;
58    end process next_state_and_state_register;
59
60    output_calculation :
61    process(state)
62    begin
63       case state is
64          when INIT =>
65             PREP_2_SEND     <= '0';
66             addr_count_rset <= '1';
67             addr_count_en   <= '0';
68          when CHECK_SEND =>
69             PREP_2_SEND     <= '0';
70             addr_count_rset <= '0';
71             addr_count_en   <= '0';
72          when PREPARE =>
73             PREP_2_SEND     <= '1';
74             addr_count_rset <= '0';
75             addr_count_en   <= '0';
76          when INC_PROM_ADDR =>
77             PREP_2_SEND     <= '0';
78             addr_count_rset <= '0';
79             addr_count_en   <= '1';
80       end case;
81    end process output_calculation;
82
83    PROM_ADDR <= std_logic_vector(to_unsigned(addr,PROM_ADDR'LENGTH));
84 end Behavioral;
```

Figure 16

The statement in line 35 assigns to the stop_addr_count signal the '1' logic value when the counter's output (addr signal) reaches the READ_UNTIL_CHR value. The READ_UNTIL_CHR value specifies the number of prom words to be read and sent by the transmitter. The stop_addr_count signal is used as input to the Moore FSM described in lines 37-81 (see line 47).

The state diagram that represents the next_state_and_state_register process (lines 37-58) is shown in **Figure 17**.

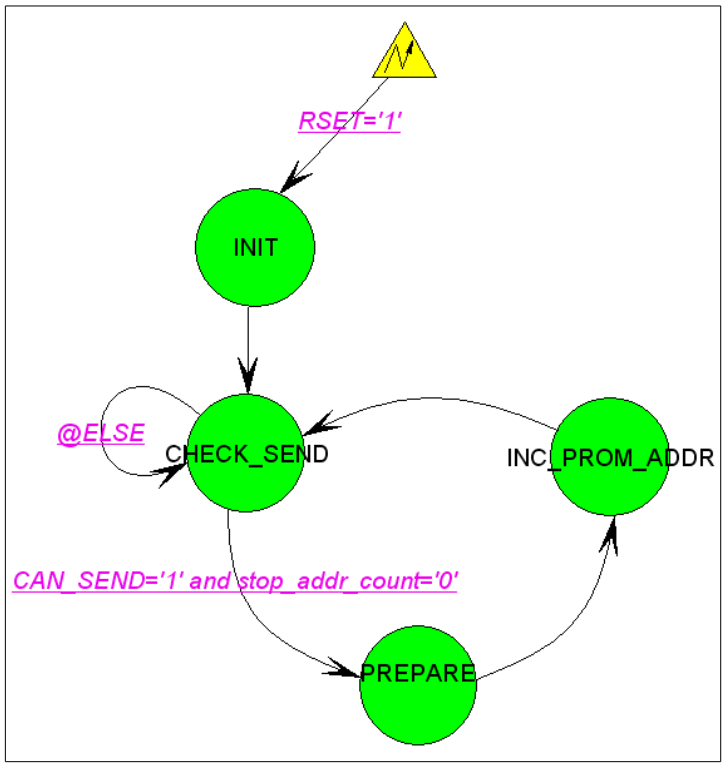

Figure 17

When the user presses the RSET button (RSET='1'), the state machine moves asynchronously to the INIT state. At the INIT state the control module initializes internal signals (see lines 65-67). At the next rising edge of the clock the state machine moves to the CHECK_SEND state. If the transmitter is not sending data and the address of the last prom word to be sent is not yet reached (CAN_SEND='1' and stop_addr_count='0'), the state machine moves to the PREPARE state, else it stays at the CHECK_SEND state. At the PREPARE state the control module informs the transmitter to prepare to send (see lines 73-75) the data that the prom holds at address 0. At the next rising edge of the clock the state machine moves to the INC_PROM_ADDR state where the control module increases by one the prom address (see lines 77-79). Finally, at the next rising edge of the clock the state machine returns to the CHECK_SEND state.

In the ISE Project Navigator window select Implementation (in the View subwindow), select the control module, right-click on the control module and select Set as Top Module (click Yes at the Set as Top Module window) and double-click Synthesize - XST (in

the Processes subwindow) to verify that the `control`'s syntax is correct and synthesizable.

10. Behavioral simulation of the `control` module :

In the ISE Project Navigator window select `Project->New Source...`. In the New Source Wizard window type `control_tb` in the File Name field, select VHDL Test Bench and click Next. In the next New Source Wizard window associate the `control_tb` file with the `control` module, click Next and then click Finish to create the `control_tb.vhd` testbench file. Modify the `control_tb.vhd` testbench file to look like the one in **Figure 18**.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY control_tb IS
5  END control_tb;
6
7  ARCHITECTURE behavior OF control_tb IS
8      -- Component Declaration for the Unit Under Test (UUT)
9      COMPONENT control
10     PORT(
11          CLK : IN  std_logic;
12          RSET : IN  std_logic;
13          CAN_SEND : IN  std_logic;
14          PREP_2_SEND : OUT  std_logic;
15          PROM_ADDR : OUT  std_logic_vector(3 downto 0)
16         );
17     END COMPONENT;
18    --Inputs
19    signal CLK : std_logic := '0';
20    signal RSET : std_logic := '0';
21    signal CAN_SEND : std_logic := '0';
22    --Outputs
23    signal PREP_2_SEND : std_logic;
24    signal PROM_ADDR : std_logic_vector(3 downto 0);
25    -- Clock period definitions
26    constant CLK_period : time := 37 ns;
27 BEGIN
28    -- Instantiate the Unit Under Test (UUT)
29    uut: control PORT MAP (
30           CLK => CLK,
31           RSET => RSET,
32           CAN_SEND => CAN_SEND,
33           PREP_2_SEND => PREP_2_SEND,
34           PROM_ADDR => PROM_ADDR
35          );
36    -- Clock process definitions
37    CLK_process :process
38    begin
39       CLK <= '0';
40       wait for CLK_period/2;
41       CLK <= '1';
42       wait for CLK_period/2;
43    end process;
44    -- Stimulus process
45    stim_proc: process
46    begin
47       RSET     <= '0';
48       CAN_SEND <= '0';
49       wait for CLK_period*5;
50       RSET     <= '1';
51       CAN_SEND <= '0';
52       wait for CLK_period*1;
53       RSET     <= '0';
54       CAN_SEND <= '0';
55       wait for CLK_period*3;
56       RSET     <= '0';
57       CAN_SEND <= '1';
58       wait for CLK_period*1;
59       RSET     <= '0';
60       CAN_SEND <= '0';
61       wait;
62    end process;
63 END;
```

Figure 18

In the ISE Project Navigator window select Simulation (in the View subwindow), select the `control_tb` testbench file, expand ISim Simulator (in the Processes subwindow) and double-click Behavioral Check Syntax to check `control_tb` testbench's syntax.

Right-click Simulate Behavioral Model (in the Processes subwindow) and select Process Properties. Set the Simulation Run Time to 1000 ns and click OK. Now double-click Simulate Behavioral Model to run the simulation as specified by the `transmitter_tb` testbench file and the ISim Simulation Run Time parameter. Select `View->Zoom->To Full View` in the ISim simulator window. In the Instances and Processes subwindow expand the `control_tb` (double-click on it or right-click on it and select `Expand`) and select the `uut`. In the Objects subwindow right-click on the `state` signal and select `Add To Wave`

`Window`. In the Console window give the following commands (press the enter key after every command) : `restart` and `run 1000 ns` in order to rerun the simulation and verify the correct operation of the `control`'s state machine (see **Figure 19**).

Close the ISim simulator window.

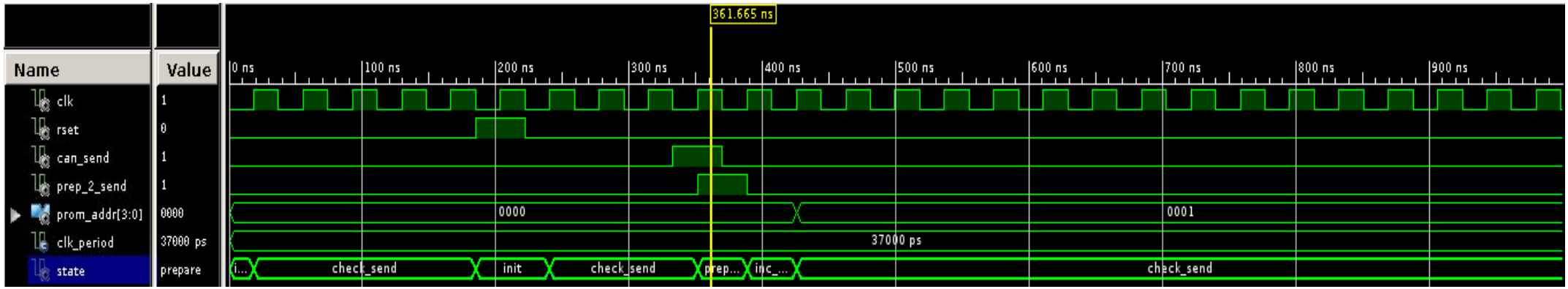
361.665 ns
Name
Value
clk
1
rset
0
can_send
1
prep_2_send
1
prom_addr[3:0]
0000
clk_period
37000 ps
state
prepare
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
0000
0001
37000 ps
i...
check_send
init
check_send
prep...
inc_...
check_send

Figure 19

11. Add New Source (`sadt_pck`) in the Project4src project :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `sadt_pck` in the File Name field, select VHDL Package, click Next and then click Finish to create the `sadt_pck.vhd` file.

12. Define the content of the `sadt_pck` VHDL package :

Remove the green highlighted lines (the comments) and edit the `sadt_pck.vhd` file to look like the one shown in **Figure 20**.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3
4  package sadt_pck is
5     component prom is
6        generic(PROM_A_BITS  : integer := 4);
7        port(ADDR : in std_logic_vector(PROM_A_BITS-1 downto 0);
8             R_D  : out std_logic_vector(7 downto 0));
9     end component;
10
11    component transmitter is
12       generic(CLK_DIV : integer := 233);
13       Port(CLK,RSET    : in  STD_LOGIC;
14            PREP_2_SEND : in  STD_LOGIC;
15            DATA_IN     : in  STD_LOGIC_VECTOR(7 downto 0);
16            CAN_SEND    : out STD_LOGIC;
17            SERIAL_OUT  : out STD_LOGIC);
18    end component;
19
20    component control is
21       generic(PROM_A_BITS    : integer := 4;
22               READ_UNTIL_CHR : integer := 12);
23       port(CLK,RSET    : in  STD_LOGIC;
24            CAN_SEND    : in  STD_LOGIC;
25            PREP_2_SEND : out STD_LOGIC;
26            PROM_ADDR   : out  STD_LOGIC_VECTOR(PROM_A_BITS-1 downto 0));
27    end component;
28 end sadt_pck;
```

Figure 20

The package `sadt_pck` (**Figure 20**) is used for component declaration. This package will be used in the `sadt` VHDL module to avoid the declaration of the components in the architecture declarative region. This way of component declaration is used when components are used often in many different modules in a big project. This kind of component declaration is mainly used in the current project for demonstration purposes – there is not actual need to use this kind of component declaration.

In the ISE Project Navigator window select Implementation (in the View subwindow), select the `sadt_pck` package and double-click Check Syntax (in the Processes subwindow)

to verify that the `sadt_pck`'s syntax is correct.

13. Add New Source (`sadt`) in the Project4src project :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `sadt` in the File Name field, select VHDL Module and click Next. The next New Source Wizard window must be completed as shown in **Figure 21**.

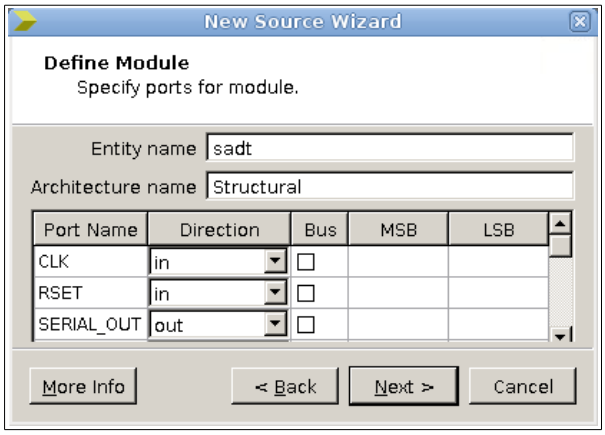

Figure 21

Clicking Next and Finish, the file `sadt.vhd` is created.

14. Define the `structure` of the `sadt` VHDL module :

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  library work;
4  use work.sadt_pck.all;
5
6  entity sadt is
7     generic(CLOCK_FREQ     : integer := 27000000;
8             BAUD_RATE      : integer := 115200;
9             PROM_A_BITS    : integer := 4;
10            READ_UNTIL_CHR : integer := 12);
11    port(CLK,RSET   : in  STD_LOGIC;
12         SERIAL_OUT : out STD_LOGIC);
13 end sadt;
14
15 architecture Structural of sadt is
16    signal addr       : std_logic_vector(PROM_A_BITS-1 downto 0);
17    signal r_d        : std_logic_vector(7 downto 0);
18
19    signal req_2_send : std_logic;
20    signal can_send   : std_logic;
21 begin
22    part_1 : prom generic map(PROM_A_BITS => PROM_A_BITS)
23       port map(addr,r_d);
24    part_2 : transmitter
25       generic map(CLK_DIV => (CLOCK_FREQ / BAUD_RATE) - 1)
26       port map(CLK,RSET,req_2_send,r_d,can_send,SERIAL_OUT);
27    part_3 : control
28       generic map(PROM_A_BITS => PROM_A_BITS,READ_UNTIL_CHR => READ_UNTIL_CHR)
29       port map(CLK,RSET,can_send,req_2_send,addr);
30 end Structural;
```

Figure 22

Remove the green highlighted lines (the comments) and edit the `sadt.vhd` file to look like the one shown in **Figure 22**.

In lines `3-4` the use of the `sadt_pck` package in the `sadt` module is shown. The `work` library indicates the working directory (`/home/fpga-user/Documents/Project4src`) where all the project VHDL source files are saved. The statement `use work.sadt_pck.all;` means that all the components declared in the `sadt_pck` package can be used in the `sadt` module.

In the ISE Project Navigator window select Implementation (in the View subwindow), right-click on the `sadt` module and select `Set as Top Module` (click Yes at the Set as Top Module window), expand Synthesize - XST (in the Processes subwindow) and double-click View RTL Schematic to verify that the `sadt`'s syntax is correct and synthesizable, and to view the RTL schematic of the `sadt` module (select Start with a schematic of the top-level block at the Set RTL/Tech Viewer Startup Mode window, click OK and double-click on the schematic to see **Figure 23**).

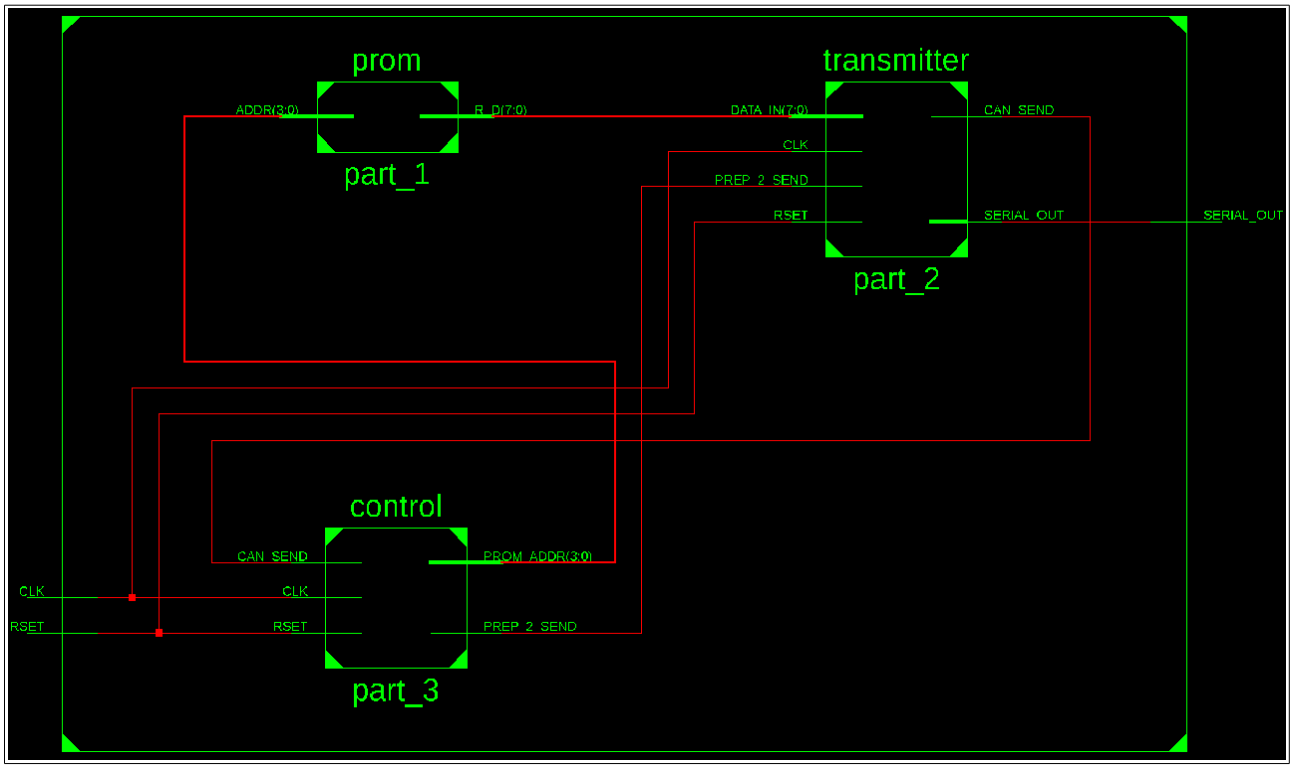

Figure 23

15. Behavioral simulation of the `sadt` module :

In the ISE Project Navigator window select `Project->New Source...`. In the New Source Wizard window type `sadt_tb` in the File Name field, select VHDL Test Bench and click Next. In the next New Source Wizard window associate the `sadt_tb` file with the `sadt` module, click Next and then click Finish to create the `sadt_tb.vhd` testbench file. Modify the `sadt_tb.vhd` testbench file to look like the one in **Figure 24**.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY sadt_tb IS
5  END sadt_tb;
6
7  ARCHITECTURE behavior OF sadt_tb IS
8      -- Component Declaration for the
9      -- Unit Under Test (UUT)
10     COMPONENT sadt
11     PORT(
12          CLK : IN  std_logic;
13          RSET : IN  std_logic;
14          SERIAL_OUT : OUT  std_logic
15         );
16     END COMPONENT;
17    --Inputs
18    signal CLK : std_logic := '0';
19    signal RSET : std_logic := '0';
20    --Outputs
21    signal SERIAL_OUT : std_logic;
22    -- Clock period definitions
23    constant CLK_period : time := 37 ns;
24 BEGIN
25    -- Instantiate the Unit Under Test (UUT)
26    uut: sadt PORT MAP (
27           CLK => CLK,
28           RSET => RSET,
29           SERIAL_OUT => SERIAL_OUT
30         );
31    -- Clock process definitions
32    CLK_process :process
33    begin
34       CLK <= '0';
35       wait for CLK_period/2;
36       CLK <= '1';
37       wait for CLK_period/2;
38    end process;
39    -- Stimulus process
40    stim_proc: process
41    begin
42       RSET <= '0';
43       wait for CLK_period*2;
44       RSET <= '1';
45       wait for CLK_period*2;
46       RSET <= '0';
47       wait;
48    end process;
49 END;
```

Figure 24

In the ISE Project Navigator window select Simulation (in the View subwindow), select the `sadt_tb` testbench file, expand ISim Simulator (in the Processes subwindow) and double-click Behavioral Check Syntax to check `sadt_tb` testbench's syntax.

Right-click Simulate Behavioral Model (in the Processes subwindow) and select Process Properties. Set the Simulation Run Time to 1100 us and click OK. Now double-click Simulate Behavioral Model to run the simulation as specified by the `transmitter_tb` testbench file and the ISim Simulation Run Time parameter. Select `View->Zoom->To Full View` in the ISim simulator window. In the Instances and Processes subwindow expand the `sadt_tb` (double-click on it or right-click on it and select `Expand`), expand the `uut` and select the `part_3`. In the Objects subwindow right-click on the `state` signal and select `Add To Wave Window`. In the Console window give the following commands (press the enter key after every command) : `restart` and `run 1100 us` in order to rerun the simulation and verify the correct operation of the `sadt` module (see **Figure 25**).

In **Figure 25** blue markers indicate the start of a character transmission. Compare the character transmissions to the prom's contents (also shown in **Figure 25**).

Close the ISim simulator window.

```
constant mem : RomType := (x"48",x"65",x"6c",x"6c",x"6f",x"20",x"57",x"6f",
                           x"72",x"6c",x"64",x"21",x"00",x"00",x"00",x"00");
```

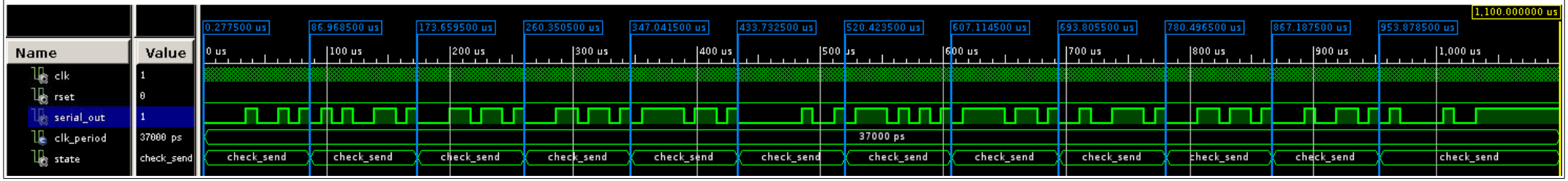

Figure 25

16. Generate ICON and VIO cores :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window, type `icon_mod` in the File Name field, select IP (CORE Generator & Architecture Wizard) and click Next. In the New Source Wizard window (Select IP), check Only IP compatible with chosen part, expand Debug & Verification, expand ChipScope Pro, select ICON (ChipScope Pro – Integrated Controller) version 1.06.a, click Next and Finish.

In the ICON (ChipScope Pro - Integrated Controller) window click Generate.

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `vio_mod` in the File Name field, select IP (CORE Generator & Architecture Wizard) and click Next. In the New Source Wizard window (Select IP), check Only IP compatible with chosen part, expand Debug & Verification, expand ChipScope Pro, select VIO (ChipScope Pro – Virtual Input/Output) version 1.05.a, click Next and Finish.

In the VIO (ChipScope Pro - Virtual Input/Output) window, check Enable Asynchronous Output Port (Width 1) and click Generate (see **Figure 26**). The VIO output port will feed the `sadt` module's `RSET` input port.

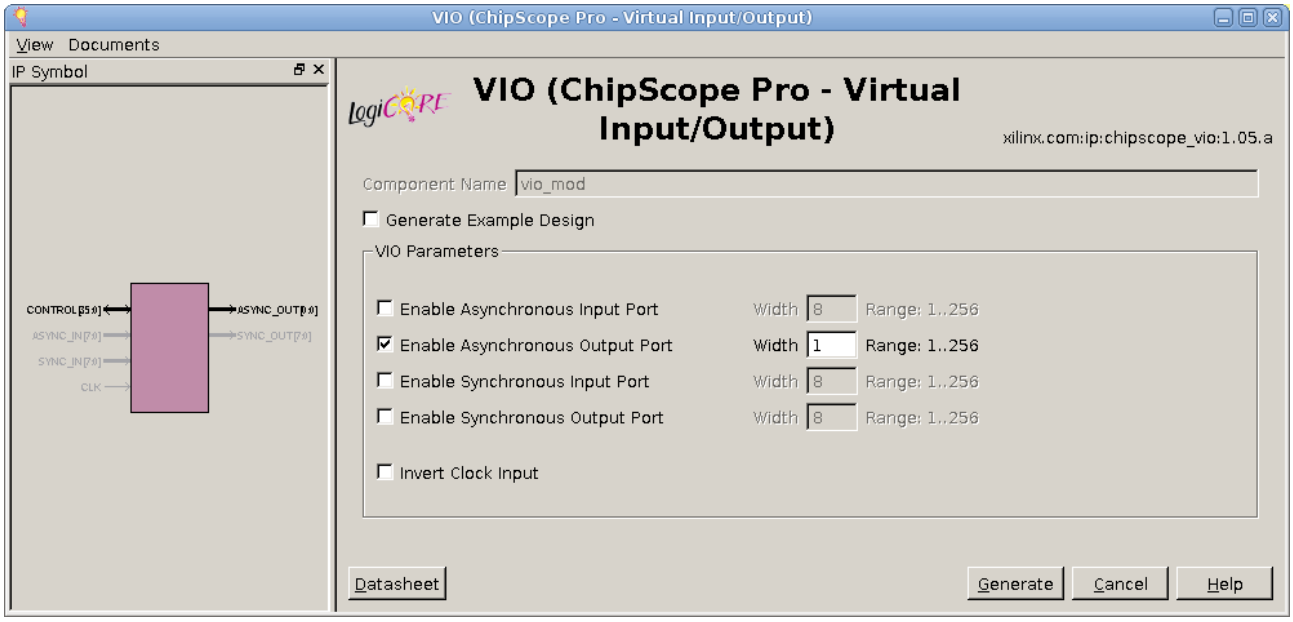

Figure 26

The `sadt` module and the two cores (ICON and VIO) will be used as components in a single VHDL module – the `top` module. The ICON and VIO cores are used to provide virtual input to the `sadt` module.

17. Create and Define the structure of the `top` VHDL module :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `top` in the File Name field, select VHDL Module and click Next. The next New Source Wizard window must be completed as shown in **Figure 27**.

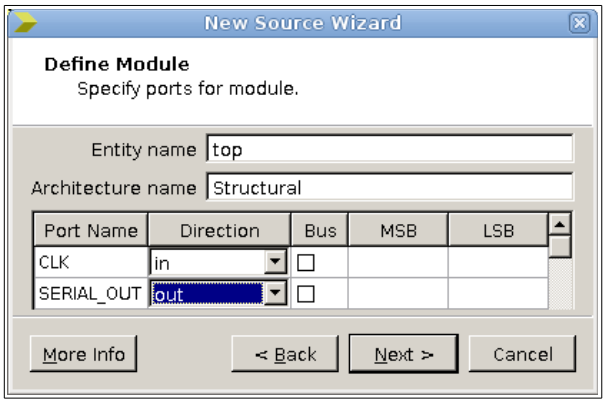

Figure 27

Clicking Next and Finish, the file `top.vhd` is created.

In the ISE Project Navigator window, select Implementation, select the `icon_mod` core (in the View subwindow), expand CORE Generator and double-click View HDL Instantiation Template (in the Processes subwindow). Copy the component declaration of the `icon_mod` core in the `top` module's architecture declarative region. Copy the instance declaration of the `icon_mod` core in the `top` module's architecture body and name the instance `part1`. Follow the above-mentioned steps for `vio_mod` core instantiation in the `top` module and name the instance `part2`.

Copy the entity declaration of the `sadt` module to the `top` module's architecture declarative region and edit it (in the `top` module) to form the component declaration of the `sadt` module. **Figure 28** shows how to complete `top` module's description.

In the ISE Project Navigator window, select Implementation, right-click on the `top` module (in the View subwindow) and select `Select as Top Module`, select the `top` module, expand Synthesize – XST (in the Processes subwindow) and double-click Check Syntax to verify that the `top` module's syntax is correct.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity top is
5     generic(CLOCK_FREQ     : integer := 27000000;
6             BAUD_RATE      : integer := 115200;
7             PROM_A_BITS    : integer := 4;
8             READ_UNTIL_CHR : integer := 12);
9      Port(CLK        : in  STD_LOGIC;
10          SERIAL_OUT : out  STD_LOGIC);
11 end top;
12
13 architecture Structural of top is
14    component icon_mod
15       PORT (
16          CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
17    end component;
18    component vio_mod
19       PORT (
20          CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
21          ASYNC_OUT : OUT STD_LOGIC_VECTOR(0 TO 0));
22    end component;
23    component sadt is
24    generic(CLOCK_FREQ     : integer := 27000000;
25            BAUD_RATE      : integer := 115200;
26            PROM_A_BITS    : integer := 4;
27            READ_UNTIL_CHR : integer := 12);
28    port(CLK,RSET   : in  STD_LOGIC;
29         SERIAL_OUT : out STD_LOGIC);
30    end component;
31
32    signal CONTROL   : STD_LOGIC_VECTOR(35 DOWNTO 0);
33    signal ASYNC_OUT : STD_LOGIC_VECTOR(0 TO 0);
34 begin
35    part1 : icon_mod
36       port map (CONTROL0 => CONTROL);
37    part2 : vio_mod
38       port map (CONTROL => CONTROL,
39                 ASYNC_OUT => ASYNC_OUT);
40    part3 : sadt
41       generic map(CLOCK_FREQ     => CLOCK_FREQ,
42                   BAUD_RATE      => BAUD_RATE,
43                   PROM_A_BITS    => PROM_A_BITS,
44                   READ_UNTIL_CHR => READ_UNTIL_CHR)
45       port map(CLK,ASYNC_OUT(0),SERIAL_OUT);
46
47 end Structural;
```

Figure 28

18. Enter Synthesis Constraints and Synthesize the `top` module :

In the ISE Project Navigator window select Implementation, select the `top` module (in the View subwindow), right-click Synthesize – XST (in the Processes subwindow) and select `Process Properties...` to modify options that affect the synthesis process (enter synthesis constraints). In the Process Properties window select HDL Options under Category, modify the FSM Encoding Algorithm value from Auto to One-Hot and click OK to close the window. Double-click Synthesize – XST to synthesize the top module and then right-click on

Synthesize – XST and select `View Text Report` to view the synthesis report. Under the Low Level Synthesis (in the synthesis report) the one-hot encoding of `top` module's state machines is presented (see **Figure 29**).

```
=========================================================================
*                          Low Level Synthesis                          *
=========================================================================
Optimizing FSM <FSM_1> on signal <state[1:4]> with One-Hot encoding.
---------------------------
 State         | Encoding
---------------------------
 init          | 0001
 check_send    | 0010
 prepare       | 0100
 inc_prom_addr | 1000
---------------------------
Optimizing FSM <part3/part_2/FSM_0> on signal <state[1:4]> with One-Hot encoding.
-----------------------
 State     | Encoding
-----------------------
 init      | 0001
 standby   | 0010
 load_data | 0100
 sending   | 1000
-----------------------
```

Figure 29

Under the Design Summary (in the synthesis report) the `Primitive and Black Box Usage`, the `Device utilization summary` and the `Timing Summary` can be found (estimated values). `Timing Summary` is shown in **Figure 30**.

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 7.069ns (Maximum Frequency: 141.470MHz)
   Minimum input arrival time before clock: 2.108ns
   Maximum output required time after clock: 3.597ns
   Maximum combinational path delay: No path found
```

Figure 30

The **Figure 30** shows that the 27MHz clock can be used in this design without the need of a timing constraint. However, for demonstration purposes, we will try to achieve a minimum period of 6.9 ns.

In the ISE Project Navigator window select Implementation, select the `top` module, expand Synthesize – XST and double-click View RTL Schematic to view the RTL schematic of the synthesized `top` module. In the Set RTL/Tech Viewer Startup Mode select Start with a Schematic of the top-level block and click OK. Double-click on the synthesized `top` module (black-box-like) graphic to view the schematic shown in **Figure 31**.

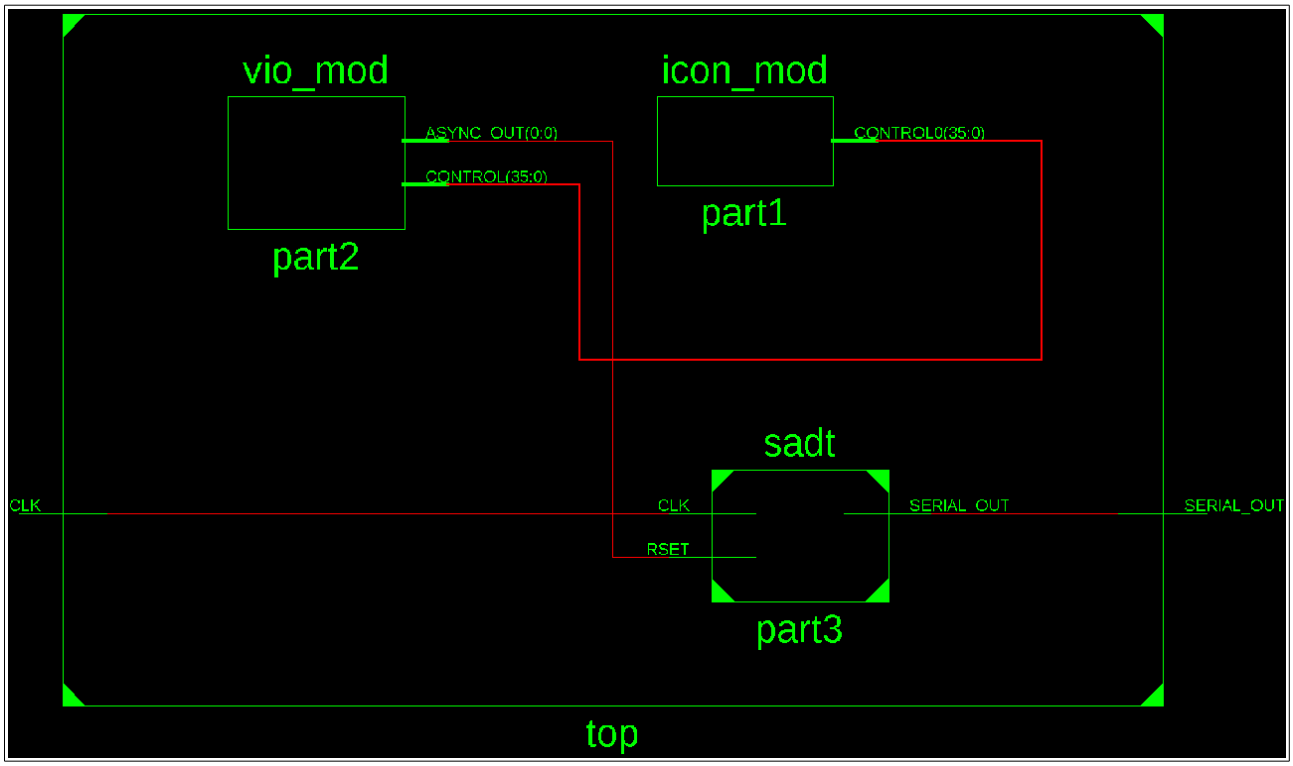

Figure 31

19. Create Timing Constraints,do Post-Synthesis I/O Pin Planning and Implement the `top` module:

In the ISE Project Navigator window select Implementation, select the `top` module (in the View subwindow), expand User Constraints (in the Processes subwindow) and double-click on Create Timing Constrains. Click Yes to the ISE Project Navigator window to create a UCF file. In the Timing Constraints tabular window, double-click `CLK` under the Unconstrained Clocks and complete the Clock Period window as shown in **Figure 32**.

In the Timing Constraints tabular window, click on the newly created entry under the Create Timing Constraints for Clock Domains (Period) table and then click on the Validate Constraints button. Then select `File->Save` to save the timing constraint for the `CLK` signal in the `top.ucf` constraint file and close the Timing Constraints tabular window.

In the ISE Project Navigator window select Implementation, select the `top` module, expand User Constraints and double-click I/O Pin Planning (PlanAhead) - Post-Synthesis to assign the `SERIAL_OUT` port of the `top` module to a specific FPGA pin. In the PlanAhead window, expand the Scalar Ports located in the I/O Ports subwindow, drag-and-drop the `CLK` port to the AB13 FPGA pin and the `SERIAL_OUT` port to the B21 FPGA pin (for the spartan-6 FPGA) [4]. In the PlanAhead window select `File->Save Design` and then close the window.

In the ISE Project Navigator window select Implementation, select the `top` module (in the View subwindow) and double-click on Implement Design (in the Processes subwindow) to implement the `top` module (ignore the warnings that arise from the use of the ICON and VIO

cores).

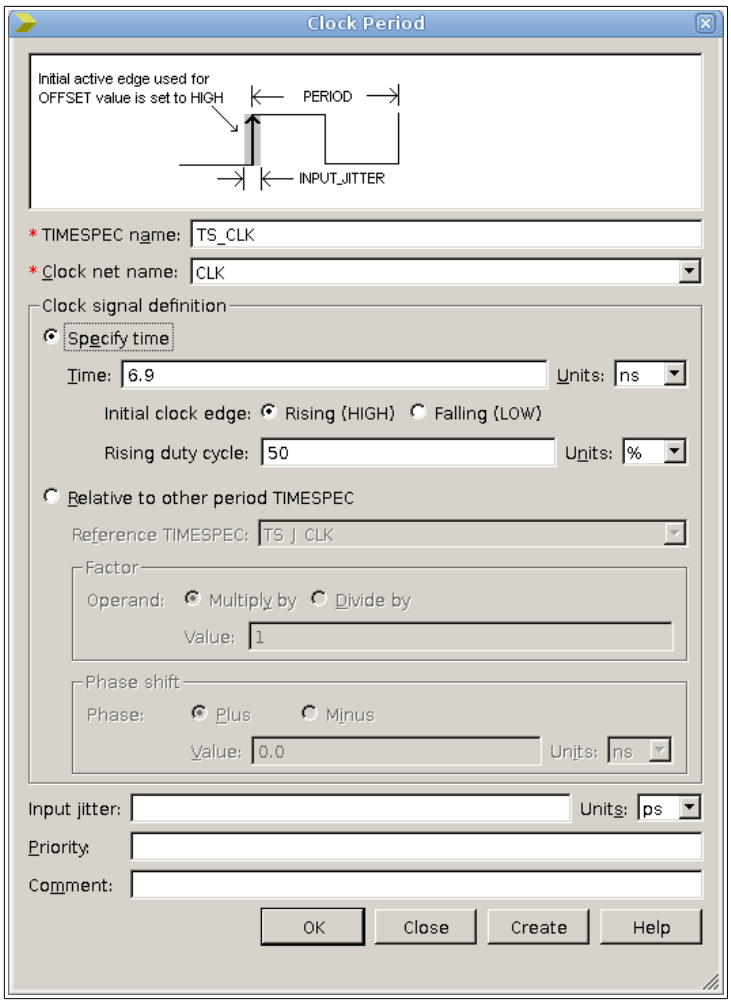

Figure 32

Double-click the Design Summary/Reports to view the exact Device Utilization Summary and the Performance Summary. Click on the All Constraints Met in the Performance Summary table to view the information shown in **Figure 33**.

		Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	TS_CLK = PERIOD TIMEGRP "CLK" 6.9 ns HIGH 50%	SETUP HOLD	2.657ns 0.439ns	4.243ns	0 0	0 0
2	Yes	TS_U_TO_I = MAXDELAY FROM TIMEGRP "U_CLK" TO TIMEGRP "I_CLK" 15 ns	SETUP HOLD	11.380ns 1.554ns	3.620ns	0 0	0 0
3	Yes	TS_U_TO_U = MAXDELAY FROM TIMEGRP "U_CLK" TO TIMEGRP "U_CLK" 15 ns	SETUP HOLD	14.102ns 0.456ns	0.898ns	0 0	0 0
4	Yes	TS_I_CLK = PERIOD TIMEGRP "I_CLK" 30 ns HIGH 50%	MINPERIOD	28.270ns	1.730ns	0	0
5	Yes	PATH "TS_I_TO_D_path" TIG	MAXDELAY		1.230ns		0
6	Yes	PATH "TS_D_TO_I_path" TIG	SETUP		4.064ns		0

Figure 33

Click on the first (`TS_CLK`) constraint and then right-click on the Maximum Data Path at Slow Process Corner and select Show in Technology Viewer to view the slowest

combinational data path of the top module (**Figure 34**).

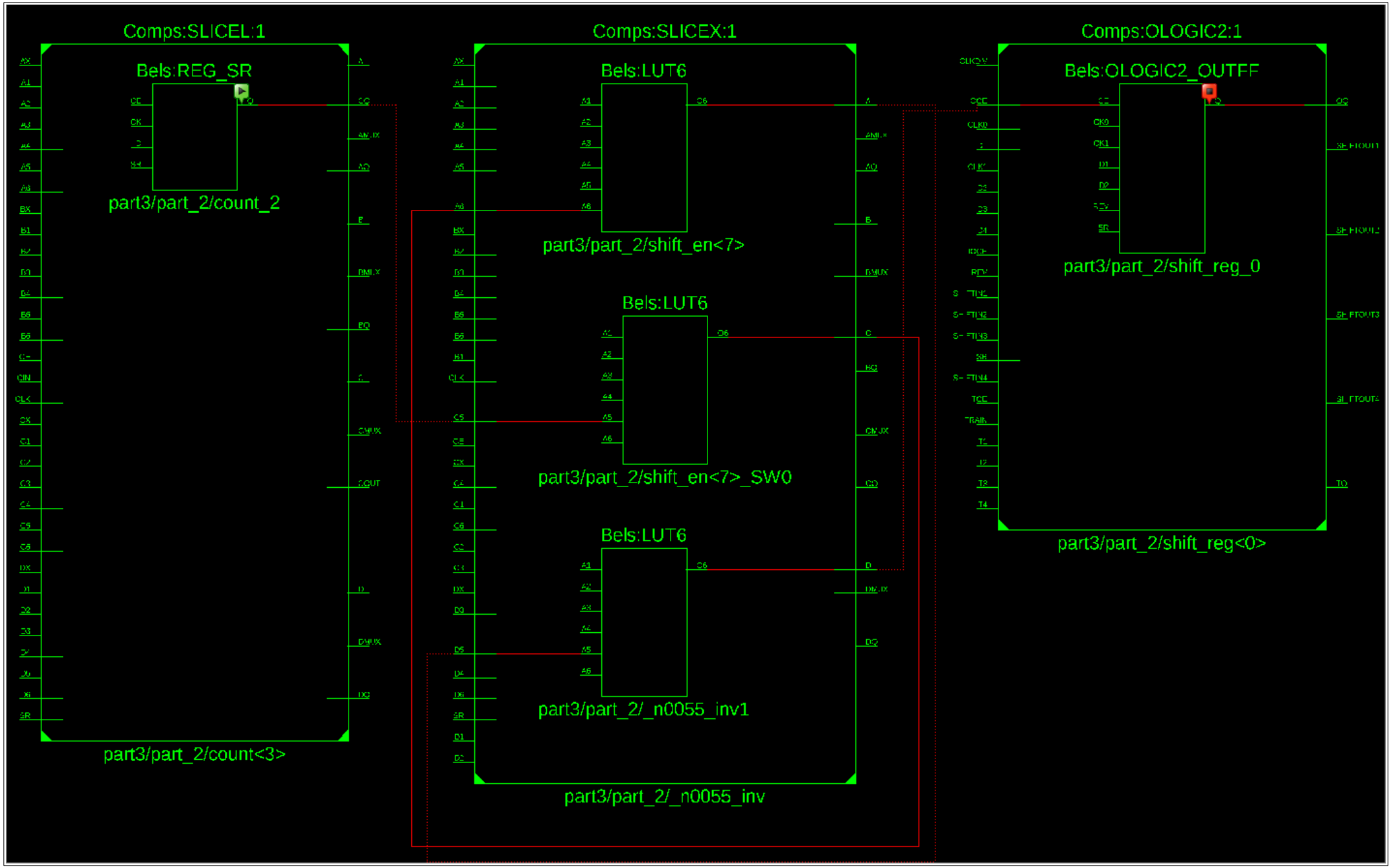
Comps:SLICEL:1
Bels:REG_SR
part3/part_2/count_2
part3/part_2/count<3>
Comps:SLICEX:1
Bels:LUT6
part3/part_2/shift_en<7>
Bels:LUT6
part3/part_2/shift_en<7>_SW0
Bels:LUT6
part3/part_2/_n0055_inv1
part3/part_2/_n0055_inv
Comps:OLOGIC2:1
Bels:OLOGIC2_OUTFF
part3/part_2/shift_reg_0
part3/part_2/shift_reg<0>

Figure 34

20. Generate Programming (Configuration) File :

In the ISE Project Navigator window select Implementation, select the `top` module (in the View subwindow), right-click Generate Programming File (in the Processes subwindow) and select Process Properties. In the Process Properties – Startup Options window select Startup Options category and change FPGA Start-Up Clock property to JTAG Clock, then select Readback Options category and check Create Readback Data files and Create Mask File properties (see **Figure 35**). Click OK to close the Process Properties – Startup Options window. The above changes have been made in order to program the FPGA and verify FPGA programming using JTAG.

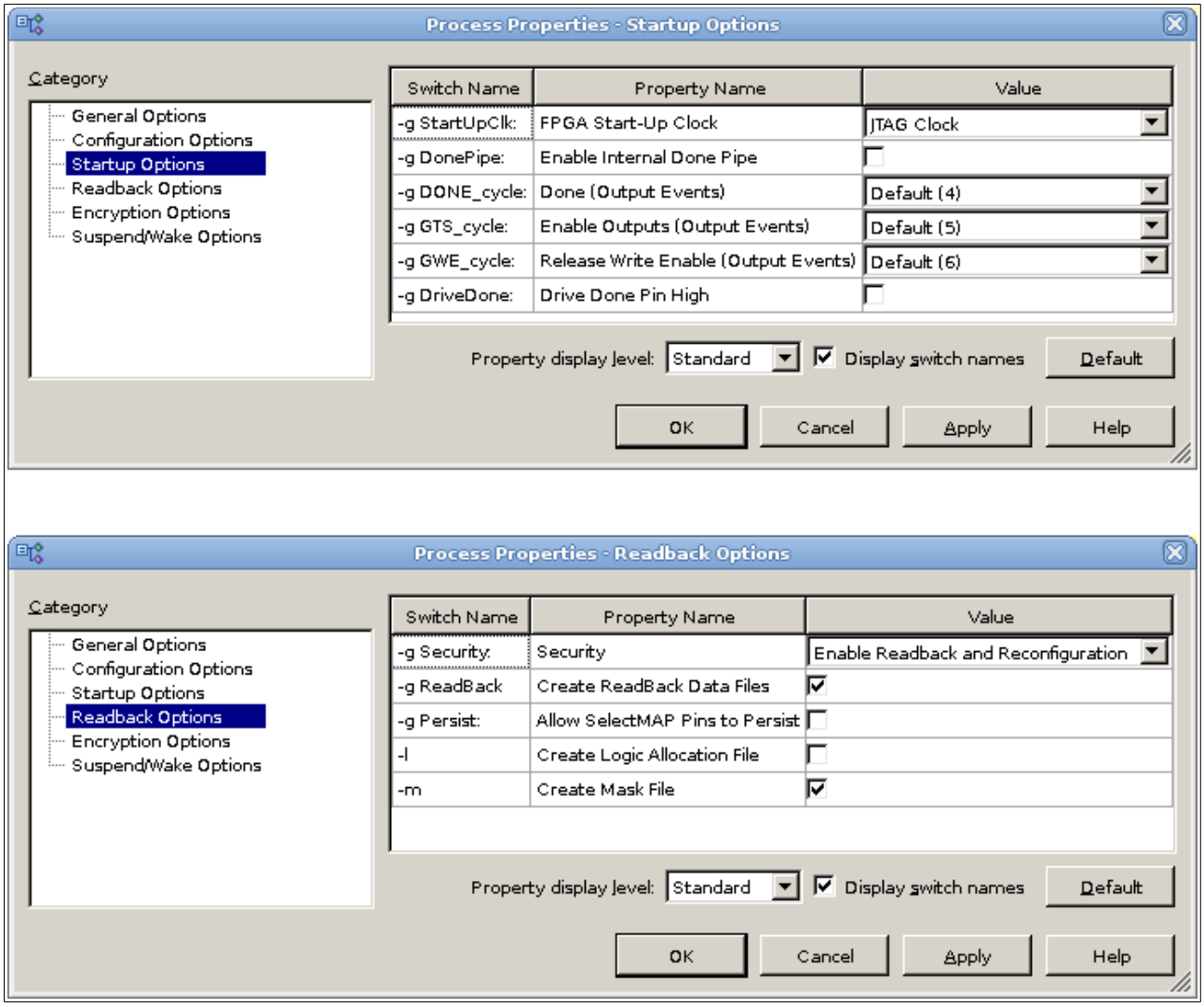

Figure 35

Double-click Generate Programming File to complete programming file generation.

21. Configure the FPGA :

In the ISE Project Navigator window double-click Configure Target Device (in the Processes subwindow) and click OK to open Impact. Impact software tool is used for FPGA

configuration. In the ISE iMPACT window double-click Boundary Scan. Right-click on Right click to Add Device or Initialize JTAG chain and select Initialize Chain. Click Yes to the Auto Assign Configuration Files Query Dialog, select Bypass for the xccace device, navigate to `/home/fpga-user/Documents/Project4src`, select the `top.bit` file and click Open for the xc6slx45t Spartan-6 FPGA. Click No to the Attach SPI or BPI PROM window. Select Device 2 in the Device Programming Properties window and check the Verify property. Right-click on the xc6slx45t Spartan-6 FPGA and select Program (see **Figure 36**).

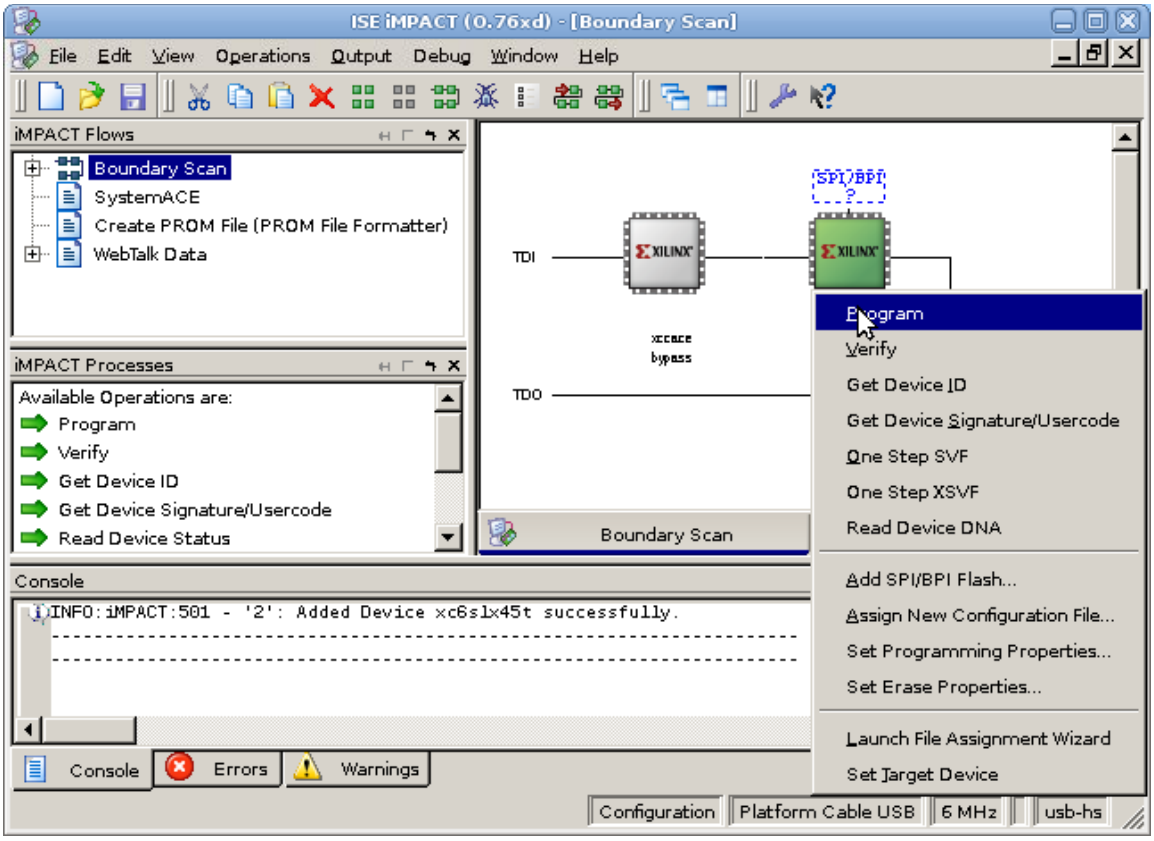

Figure 36

After the Program Succeeded message close the ISE iMPACT window (click No to iMPACT-Save Project window).

22. Test the implemented design using ChipScope and the CuteCom terminal program:

Double-click the CuteCom icon found on the Desktop of the operating system. In the CuteCom window select the options shown in **Figure 37** and then click the Open Device button.

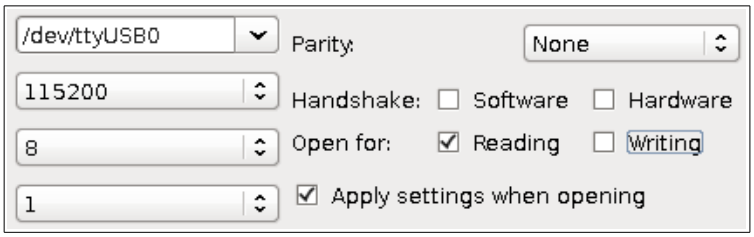

Figure 37

In the ISE Project Navigator double-click Analyze Design Using ChipScope (in the Processes subwindow). Click the Open Cable/Search JTAG Chain icon under the File menu in ChipScope Pro Analyzer window (see **Figure 38**) and click OK to the ChipScope Pro Analyzer (JTAG Chain Device Order) window.

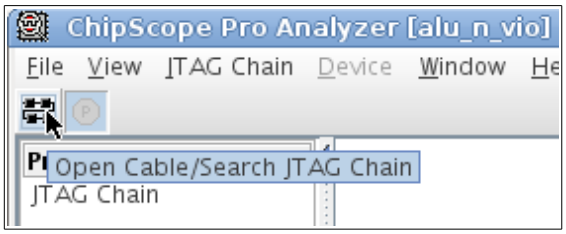

Figure 38

Double-click the VIO Console (see **Figure 39**) to open it.

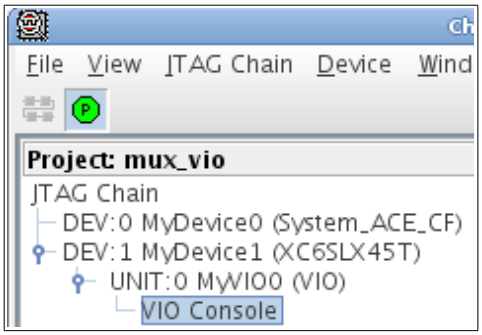

Figure 39

Right-click on the `AsyncOut[0]`, select Rename... and type RSET in the Input window. Change the RSET input to logic 1 (and press enter) and then back to logic 0 (and press enter) to begin the character transmission from the FPGA board to the PC (see the CuteCom window). The end result is shown in **Figure 40**.

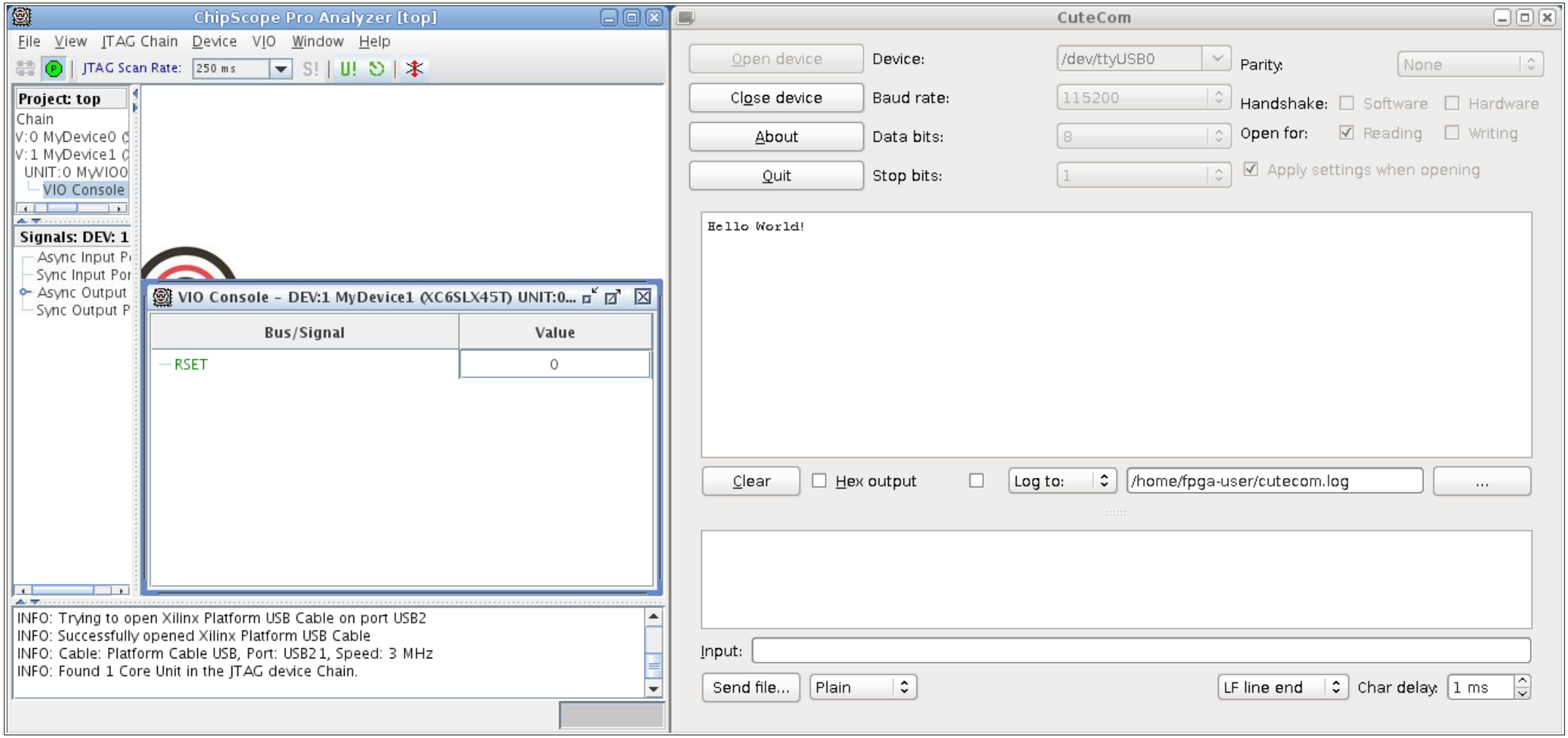
ChipScope Pro Analyzer [top]
File View JTAG Chain Device VIO Window Help
JTAG Scan Rate: 250 ms
Project: top
Chain
V:0 MyDevice0
V:1 MyDevice1
UNIT:0 MyVIO0
VIO Console
Signals: DEV: 1
Async Input P
Sync Input Por
Async Output
Sync Output P
VIO Console - DEV:1 MyDevice1 (XC6SLX45T) UNIT:0...
Bus/Signal
Value
RSET
0
INFO: Trying to open Xilinx Platform USB Cable on port USB2
INFO: Successfully opened Xilinx Platform USB Cable
INFO: Cable: Platform Cable USB, Port: USB21, Speed: 3 MHz
INFO: Found 1 Core Unit in the JTAG device Chain.
CuteCom
Open device
Close device
About
Quit
Device:
/dev/ttyUSB0
Baud rate:
115200
Data bits:
8
Stop bits:
1
Parity:
None
Handshake:
Software
Hardware
Open for:
Reading
Writing
Apply settings when opening
Hello World!
Clear
Hex output
Log to:
/home/fpga-user/cutecom.log
...
Input:
Send file...
Plain
LF line end
Char delay:
1 ms

Figure 40

References

[1] WIKIPEDIA, Universal asynchronous receiver/transmitter, http://en.wikipedia.org/wiki/UART

[2] Serial and UART Tutorial, http://www.freebsd.org

[3] WIKIPEDIA, ASCII, http://en.wikipedia.org/wiki/ASCII

[4] Xilinx Corporation, SP605 Hardware User Guide, 2011, http://www.xilinx.com