How to simulate an UART VHDL code with ghdl

René Doß
info@dossmatik.de

January 6, 2017

Contents

1 introduction UART transmission **2**

2 advanced TX Unit with FIFO **2**

3 advanced RX Unit with FIFO **8**

4 example application **13**

5 Makefile for workflow **19**

6 file in/out **19**

7 VHPI Interface to an other language **22**

8 closing words **28**

Abstract

This article was written to demonstrate on a practical example how ghdl works. The reader should have basic knowledge about VHDL. This example is fitable code into a FPGA. The code is written vendor independent. UART as example is generally understandable and also parctical on many applications. An asynchronous serial transmission has two parts Receiver and Transmitter. The handling is typical integrated on all microcontrollers today. Different sensors can communicate over UART. This makes this data transmission very interesting for a hardware developer. The Receiver and Transmitter code is included with an integrated FIFO. These all are demonstrated with ghdl simulation. You will get new spirit for VHDL code validation. The VHDL files are attached in this document.

1 introduction UART transmission

The signal is at silence condition high. A low on line manifests a start of transmission, followed by a number of data bits. The data bits are send from least significant bit (LSB) to most significant bit (MSB). Next bit is a high, witch is called stop bit and it is needed to mark the end of transmission. In this state a new start bit can be correct capture by the receiver. Note that no clock signal has sent through the serial line. The transmitter resynchronize on the falling edge in start bit condition. The data speed and the data length of sender an receiver have to be the same.

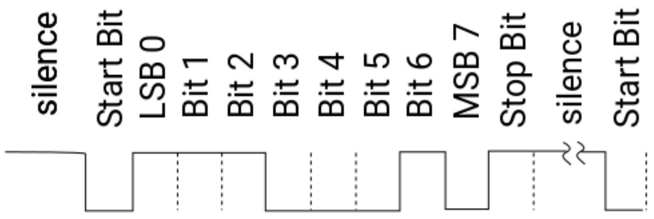


Figure 1: UART Timing

The baudrate specifies the transmission speed. The unit is baud bit per second. Common baudrates are established. Internally the hardware of UART runs in higher clock rate.

2 advanced TX Unit with FIFO

The UART interface is slow and needs a buffer for better synchronisation. Data is written in a FIFO. FIFO is the abrivation first in first out. This FIFO is realised in a ring buffer. Figure 3 explains how it works. Two pointers point at an address of internal RAM. The pointer nextwrite points to the address where the next incoming data will be written. When new data is incoming the FIFO the pointer is increased. Now valid data has to be read from the FIFO and the nextread is increased. If the two pointers in the FIFO are equal the FIFO is empty.

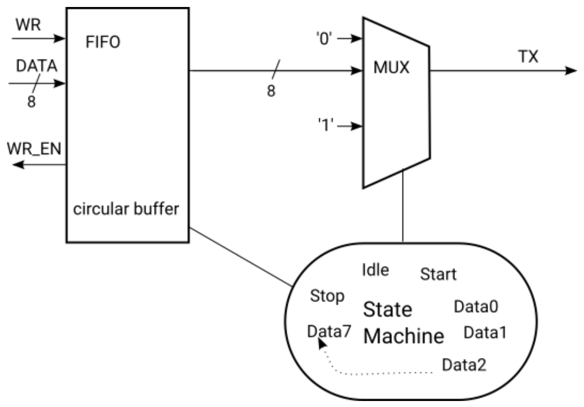


Figure 2: Blockdiagram TX

```
-- UART_TX_8N1.vhd
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

--UART transmiter without parity

entity UART_8N1_TX is
  generic(
    clk_freq    : integer;
    baudrate    : integer;
    addr_depth : integer:=5);
  port(
    clk         : in std_logic;
    reset       : in std_logic;
    --8bit  interface
    wdata       : in   std_logic_vector(7 downto 0);
    wr          : in   std_logic;
    wr_en       : out std_logic;

    --physical wire
    tx          : out std_logic);
end;


architecture Behavioral of UART_8N1_TX is
--FSM
  type   state_type is (idle, start, data0, data1, data2, data3, data4,
   data5, data6, data7, stop);

  signal state     : state_type := idle;
  signal nextstate : state_type := idle;

  --FIFO
  type   RAM is array (0 to (2**(addr_depth)-1)) of std_logic_vector (7 downto 0);
  signal fifo      : RAM;
  signal nextwrite : unsigned((addr_depth-1) downto 0);
  signal nextread  : unsigned((addr_depth-1) downto 0);

  signal send_empty: std_logic;
  --output
  signal   data_tx      : std_logic_vector (7 downto 0);
  constant tick         : integer := (clk_freq/baudrate);
  signal   tick_counter : integer range 0 to (tick+1);

begin

  wr_en   <= '0' when (nextwrite+1 = nextread) else '1';
  send_empty<='1' when nextwrite=nextread else '0';

  process(clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        nextread <= (others => '0');
      elsif state = stop and nextstate = idle then
        nextread <= nextread+1;
      end if;
    end if;
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
      state <= nextstate;
    end if;
  end process;

  process(clk)
  begin
    if rising_edge(clk) then
      if reset = '1' then
        nextwrite <= (others => '0');
      elsif wr = '1' then
        fifo(to_integer(nextwrite)) <= wdata;
        nextwrite                   <= nextwrite+1;

      end if;
    end if;
  end process;


  data_tx  <= fifo(to_integer(nextread));

  process(clk)
  begin
    if rising_edge(clk) then
      if state = idle then
        if nextwrite /= nextread then
          nextstate    <= start;
          tick_counter <= 0;
          tx           <= '0';
        else
          tx <= '1';
        end if;
      else
        if tick_counter = tick then
          tick_counter <= 0;
        else
          tick_counter <= tick_counter + 1;
        end if;
      end if;
      if tick_counter = tick then
        if state = start then
          nextstate <= data0;
          tx        <= data_tx(0);
        end if;

        if state = data0 then
          nextstate <= data1;
          tx        <= data_tx(1);
        end if;

        if state = data1 then
          nextstate <= data2;
          tx        <= data_tx(2);
        end if;

        if state = data2 then
          nextstate <= data3;
          tx        <= data_tx(3);
        end if;
        if state = data3 then
          nextstate <= data4;
          tx        <= data_tx(4);
        end if;
        if state = data4 then
          nextstate <= data5;
          tx        <= data_tx(5);
        end if;
        if state = data5 then
          nextstate <= data6;
          tx        <= data_tx(6);
        end if;
        if state = data6 then
          nextstate <= data7;
          tx        <= data_tx(7);
        end if;
        if state = data7 then
          nextstate <= stop;
          tx        <= '1';
        end if;
        if state = stop then
          nextstate <= idle;
          tx        <= '1';
        end if;
      end if;

      if reset = '1' then
        tick_counter <= 0;
        tx           <= '1';
        nextstate    <= idle;
      end if;
    end if;

  end process;

end Behavioral;
```

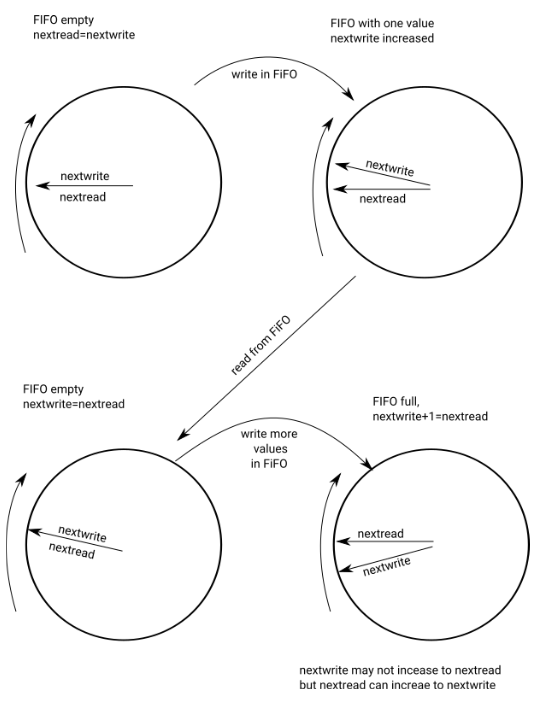


Figure 3: FIFO pointer

The FIFO is a circular buffer and works with two address pointers. One pointer is used to point at the write address. The other pointer points at the read address. Both pointers are increased with one step. The interface has two control signals for handshakes. send_busy and send_empty these are the FIFO states full and empty. The distance between nextwrite and nextread generates the signal busy because even one more wirte action would cause an overflow in the buffer. The testbench is required.

```
-- tb_UART_TX_8N1.vhd
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_UART_TX_8N1 is
end tb_UART_TX_8N1;

architecture behavior of tb_UART_TX_8N1 is

  --Inputs
  signal board_clk : std_logic := '0';
  signal reset     : std_logic := '0';
  signal send_data : std_logic_vector (7 downto 0);
  signal wr        : std_logic := '0';
  signal tx        : std_logic;

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

```vhdl
  component UART_8N1_TX is
  generic(
    clk_freq   : integer;
    baudrate   : integer;
    addr_depth : integer:=5);
  port(
    clk         : in std_logic;
    reset       : in std_logic;
    --8bit  interface
    wdata       : in  std_logic_vector(7 downto 0);
    wr          : in  std_logic;
    wr_en       : out std_logic;

    --physical wire
    tx          : out std_logic);
end component;

  constant board_clk_period : time := 10 ns;
  constant board_clk_freq: integer :=100E6;   --100MHz

begin

  -- Instantiate the Unit Under Test (UUT)
UART_TX: UART_8N1_TX
  generic map(
    clk_freq   => board_clk_freq,
    baudrate   => 115200
  --  addr_depth => use predefined
  )
  port map(
    clk         => board_clk,
    reset       => reset,

    wdata       => send_data,
    wr          => wr,
    wr_en       => open,

    tx          => tx
    );

    -- Clock process definitions
    board_clk_process : process
      begin
          board_clk <= '0';
        wait for board_clk_period/2;
          board_clk <= '1';
        wait for board_clk_period/2;
      end process;


   -- Stimulus process
   stim_proc : process
     begin
       reset <= '1';

      wait for 15 us;
        reset <='0';
        send_data<=X"A0";
      wait until rising_edge(board_clk);
        wr<= '1';

      wait until rising_edge(board_clk);
```

```
        wr<= '0';

      wait for 5 us;

      wait until rising_edge(board_clk);
        wr<= '1';
        send_data<=X"B1";

      wait until rising_edge(board_clk);
        wr<= '0';
      wait;
    end process;

  end;
```

Put both files in the same directory and simulate this testcase and check the timing requirements.

```
ghdl -i *.vhd
ghdl -m tb_UART_TX_8N1
ghdl -r tb_UART_TX_8N1 --stop-time=200us --wave=TX.ghw
gtkwave TX.ghw
```

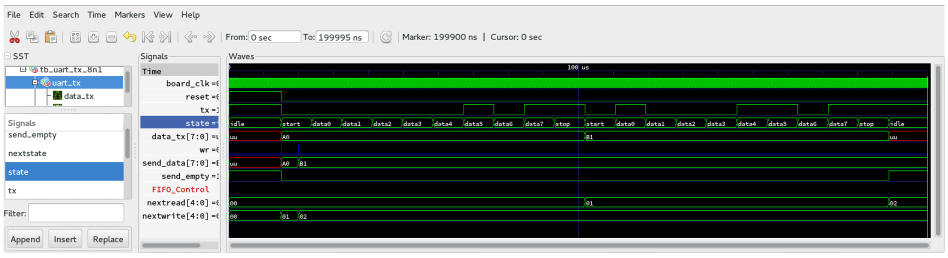


Figure 4: UART TX output

3 advanced RX Unit with FIFO

The receiver is the opposite component of the transmitter. The code is mapped also in this article. The generic is used to set the right timing. The tick_counter is the timer of one bit. The counter range of tick is calculated by the generic clk_freq and baudrate. These parameters make the code portable into different designs. There is nothing to change inside the code when another frequency or baudrate are used. Such parameter makes source code simpiler reusable. When the transmission is off, the RX has a high on line. The state maschine is in idle and waits for RX to turn low. This condition is checked of the falling edge on the start bit. All other bits are cought in the middle of the bit time.

```vhdl
-- UART_RX_8N1.vhd
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;


entity UART_8N1_RX is

  generic (clk_freq : integer;
           baudrate : integer);
  port(
    clk           : in  std_logic;
    reset         : in  std_logic;
    --8bit interface
    rdata         : out std_logic_vector(7 downto 0);
    rd            : in  std_logic;
    rd_en         : out std_logic;
    --physical wire RX
    rx            : in  std_logic
    );
end UART_8N1_RX;


architecture Behavioral of UART_8N1_RX is

  type   state_type is (idle, start, data0, data1, data2, data3, data4,
   data5, data6, data7, stop);
  signal state : state_type;

  signal puffer : std_logic_vector (7 downto 0);

--FIFO
  type RAM is array (0 to 63) of std_logic_vector (7 downto 0);

  signal fifo : RAM ;

  signal nextwrite    : unsigned(5 downto 0);
  signal nextread     : unsigned(5 downto 0);

  constant tick       : integer := clk_freq/baudrate;
  signal tick_counter : integer range 0 to (tick+1);


begin

  rdata  <= fifo(to_integer(nextread));

  process (clk)
  begin
    if rising_edge(clk) then
      if rd = '1'  then
        nextread <= nextread+1;
      end if;
      if reset = '1' then
        nextread <= (others => '0');
      end if;
    end if;
  end process;

 rd_en<= '0' when  nextread=nextwrite else '1';
```

```vhdl
process(clk)
begin

  if (clk'event and clk = '1') then
    tick_counter <= tick_counter + 1;

    case state is

      when idle =>
        tick_counter <= 0;
        if (rx = '0') then  --check start condtion
          state <= start;
        else
          state <= idle;
        end if;

      when start =>
        if (tick_counter = tick/2) then  --capture in the middle
          tick_counter <= 0;
          state        <= data0;
        end if;

      when data0 =>
        if (tick_counter = tick) then
          puffer (0)   <= rx;
          tick_counter <= 0;
          state        <= data1;
        end if;
      when data1 =>
        if (tick_counter = tick) then
          puffer (1)   <= rx;
          tick_counter <= 0;
          state        <= data2;
        end if;
      when data2 =>
        if (tick_counter = tick) then
          puffer (2)   <= rx;
          tick_counter <= 0;
          state        <= data3;
        end if;
      when data3 =>
        if (tick_counter = tick) then
          puffer(3)    <= rx;
          tick_counter <= 0;
          state        <= data4;
        end if;
      when data4 =>
        if (tick_counter = tick) then
          puffer (4)   <= rx;
          tick_counter <= 0;
          state        <= data5;
        end if;
      when data5 =>
        if (tick_counter = tick) then
          puffer (5)   <= rx;
          tick_counter <= 0;
          state        <= data6;
        end if;
      when data6 =>
        if (tick_counter = tick) then
          puffer (6)   <= rx;
```

```
              tick_counter <= 0;
              state        <= data7;
            end if;
          when data7 =>
            if (tick_counter = tick) then
              puffer (7)   <= rx;
              tick_counter <= 0;
              state        <= stop;
            end if;
          when stop =>
            if (tick_counter = tick) then
              fifo(to_integer(nextwrite)) <= puffer;
              nextwrite                   <= nextwrite+1;
              tick_counter <= 0;
              state        <= idle;
            end if;
        end case;
        if reset='1' then
          state <=idle;
          nextwrite <= (others => '0');
        end if;
      end if;

    end process;
  end Behavioral;
```

Now the apposite testbench.

```
-- tb_UART_RX_8N1.vhd
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity tb_UART_RX_8N1 is
end tb_UART_RX_8N1;

architecture behavior of tb_UART_RX_8N1 is

  signal board_clk     : std_logic := '0';
  signal reset         : std_logic := '0';
  signal rx            : std_logic := '1';
  signal rd            : std_logic := '0';
  signal rd_en     : std_logic;
  signal data          : std_logic_vector (7 downto 0);

  procedure tx_char (signal txpin : out std_logic;
              txdata : in character; baudrate : in integer) is
    constant bittime : time := (integer(1000000000.0/real(baudrate))) * 1 ns;
    variable c       : std_logic_vector(7 downto 0);
  begin
    c     := std_logic_vector(to_unsigned(character'pos(txdata), 8));
    txpin <= '0';                          -- Startbit
    wait for bittime;
    for i in 0 to 7 loop
      txpin <= c(i);
      wait for bittime;
    end loop;
    txpin <= '1';                          -- Stopbit
    wait for bittime;
  end tx_char;
  --Outputs

component UART_8N1_RX is

  generic (clk_freq : integer;
           baudrate : integer);
  port(
    clk            : in  std_logic;
    reset          : in  std_logic;
    --8bit interface
    rdata          : out std_logic_vector(7 downto 0);
    rd             : in  std_logic;
    rd_en          : out std_logic;
    --physical wire RX
    rx             : in  std_logic
    );
end component;

  constant board_clk_period : time := 10 ns;
  constant board_clk_freq: integer :=100E6;   --100MHz

begin

  -- Instantiate the Unit Under Test (UUT)
UART_RX:UART_8N1_RX

  generic map (
    clk_freq    => board_clk_freq,
    baudrate    =>115200)
  port map(
    clk            => board_clk,
    reset          => reset,
    rdata          => data,
    rd             => rd,
    rd_en          => rd_en,
    --physical wire RX
    rx             => rx
    );

  process
  begin
    wait for 80000 ns;
      tx_char(RX, '$', 115200);
      tx_char(RX, 'g', 115200);
      tx_char(RX, '#', 115200);
    wait for 50 us;
      tx_char(RX, '6', 115200);
      tx_char(RX, '7', 115200);
   wait;                       -- will wait forever
   end process;

  -- Clock process definitions
  board_clk_process : process
  begin
    board_clk <= '0';
    wait for board_clk_period/2;
      board_clk <= '1';
    wait for board_clk_period/2;
  end process;

  -- Stimulus process
  process
```

```
     begin
         reset <= '1';
       wait for 100 ns;
        reset <='0';
      wait;
    end process;

   -- Stimulus process
   stim_proc : process
   begin

   loop
      wait for 200 us;
      wait until rising_edge(board_clk);
      if rd_en='1' then
        rd<= '1';
      end if;
      wait until rising_edge(board_clk);
       rd<= '0';
    end loop;

      wait;
    end process;

   end;
```

```
ghdl -i *.vhd
ghdl -m tb_UART_RX_8N1
ghdl -r tb_UART_RX_8N1 --stop-time=800us --wave=RX.ghw
gtkwave RX.ghw
```

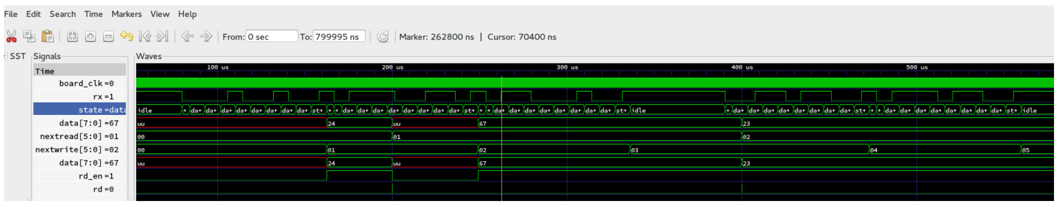


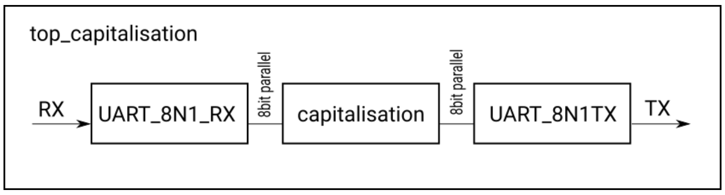
Figure 5: UART RX in

4 example application

In the last sections we have designed an input and an output interface. By combinating the receiving and transmitting example, we can build the full UART. Now let us make a simple capitalisation ASCII engine. This is more a theoretical example for a demonstration. The data stream is very simple. It demonstrates how it works.

```
-- loopback engine
----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity capitalisation is
  port(
    clk         : in   std_logic;
    reset       : in   std_logic;
    --in
    rdata       : in   std_logic_vector(7 downto 0);
    rd_en       : in   std_logic;
    rd          : out  std_logic;
    --out
    wdata       : out  std_logic_vector(7 downto 0);
    wr_en       : in   std_logic;
    wr          : out  std_logic
    );
end;

architecture Behavioral of capitalisation is


begin


  process(clk)
  begin
      wr<='0';
      rd<='0';
      if wr_en='1' and rd_en ='1' then
         wr<='1';
         rd<='1';
        if (unsigned(rdata)>X"60") and
           (unsigned(rdata)<X"7B") then
             wdata<=rdata(7 downto 6 )&'0'& rdata(4 downto 0);
        else
          wdata<=rdata;
        end if;
      end if;
      if reset='1' then
        wr<='0';
      end if;
  end process;


end Behavioral;
```

The application has an 8bit input and a 8bit output interface. A top instance combine the application and the RX und TX interface together. RX and TX are the physical wires and board_clk is the internal clock of all. This design is competent to fit inside an FPGA.



```
-- top_capitalisation.vhd
----------------------------------------------------------------------

-- top_capitalisation
-- |
-- + capitalisation
-- + UART_8N1_RX
-- + UART_8N1_TX

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_capitalisation is
  generic (clk_freq : integer;
           baudrate : integer);
  port(
    board_clk      : in std_logic ;
    reset          : in std_logic ;
    rx             : in std_logic ;
    tx             : out std_logic );
end top_capitalisation;

architecture behavior of top_capitalisation is
  signal rd_en    : std_logic;
  signal rdata    : std_logic_vector (7 downto 0);
  signal rd       : std_logic;

  signal wr_en    : std_logic;
  signal wdata    : std_logic_vector (7 downto 0);
  signal wr       : std_logic;

component capitalisation is
  port(
    clk         : in  std_logic;
    reset       : in  std_logic;
    --in
    rdata       : in  std_logic_vector(7 downto 0);
    rd_en       : in  std_logic;
    rd          : out std_logic;
    --out
    wdata       : out std_logic_vector(7 downto 0);
    wr_en       : in  std_logic;
    wr          : out std_logic
    );
end component;

component UART_8N1_RX is
  generic (clk_freq : integer;
           baudrate : integer);
  port(
    clk            : in  std_logic;
    reset         : in  std_logic;
    --8bit interface
    rdata         : out std_logic_vector(7 downto 0);
    rd            : in  std_logic;
    rd_en         : out std_logic;
    --physical wire RX
    rx            : in  std_logic
    );
end component;

component UART_8N1_TX is
  generic(
    clk_freq   : integer;
    baudrate   : integer;
    addr_depth : integer:=5);
  port(
    clk        : in std_logic;
    reset      : in std_logic;
    --8bit  interface
    wdata      : in  std_logic_vector(7 downto 0);
    wr         : in  std_logic;
    wr_en      : out std_logic;

    --physical wire
    tx         : out std_logic);
end component;


begin

UART_RX:UART_8N1_RX
  generic map (
    clk_freq   => clk_freq,
    baudrate   =>115200)
  port map(
    clk           => board_clk,
    reset         => reset,
    rdata         => rdata,
    rd            => rd,
    rd_en         => rd_en,
    --physical wire RX
    rx            => rx
    );

trans: capitalisation
  port map(
    clk        => board_clk,
    reset      => reset,
    --in
    rdata      => rdata,
    rd_en      => rd_en,
    rd         => rd,
    --out
    wdata      => wdata,
    wr_en      => wr_en,
    wr         => wr
    );

UART_TX: UART_8N1_TX
  generic map(
    clk_freq   => clk_freq,
    baudrate   => 115200)
  port map(
    clk         => board_clk,
    reset       => reset,
    --8bit  interface
    wdata       => wdata,
    wr          => wr,
    wr_en       => wr_en,

    --physical wire TX
    tx          => tx
    );
end;
```

Also for this the testbench.

```
-- tb_capitalisation.vhd
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_capitalisation is
end tb_capitalisation;

architecture behavior of tb_capitalisation is

  constant board_clk_period : time := 10 ns;
  constant board_clk_freq: integer :=100E6;   --100MHz

  signal board_clk     : std_logic := '0';
  signal reset         : std_logic := '0';
  signal rx            : std_logic := '1';
  signal tx            : std_logic;

  procedure tx_char (signal txpin : out std_logic;
            txdata : in character; baudrate : in integer) is
    constant bittime : time := (integer(1000000000.0/real(baudrate))) * 1 ns;
    variable c       : std_logic_vector(7 downto 0);
  begin
    c      := std_logic_vector(to_unsigned(character'pos(txdata), 8));
    txpin <= '0';                          -- Startbit
    wait for bittime;
    for i in 0 to 7 loop
      txpin <= c(i);
      wait for bittime;
    end loop;
    txpin <= '1';                          -- Stopbit
    wait for bittime;
  end tx_char;

  -- Component Declaration for the Unit Under Test (UUT)
component top_capitalisation is
  generic (clk_freq : integer;
          baudrate : integer);
  port(
    board_clk      : in std_logic ;
    reset          : in std_logic ;
    rx             : in std_logic ;
    tx             : out std_logic );
end component;
```

```
begin
  -- Instantiate the Unit Under Test (UUT)
UART_example: top_capitalisation
  generic map (
    clk_freq    => board_clk_freq,
    baudrate    =>115200)
  port map(
    board_clk     => board_clk,
    reset         => reset,
    rx            => rx,
    tx            => tx );

  process
  begin
    wait for 80000 ns;
      tx_char(RX, '$', 115200);
      tx_char(RX, 'g', 115200);
      tx_char(RX, '#', 115200);
    wait for 50 us;
      tx_char(RX, 'b', 115200);
      tx_char(RX, 'c', 115200);
   wait;                       -- will wait forever
   end process;

  -- Clock process definitions
  board_clk_process : process
  begin
    board_clk <= '0';
    wait for board_clk_period/2;
      board_clk <= '1';
    wait for board_clk_period/2;
  end process;

  -- Stimulus process
  process
     begin
         reset <= '1';
       wait for 100 ns;
        reset <='0';
      wait;
    end process;

end;
```

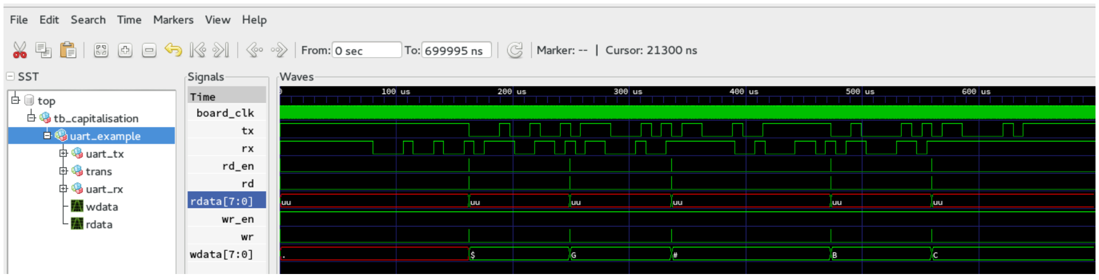


Figure 6: capitalisation

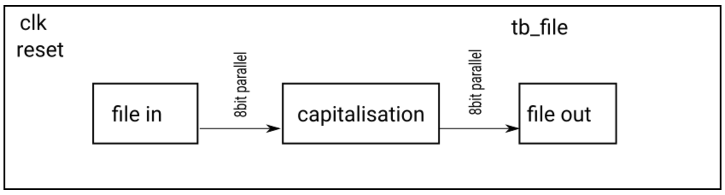
5 Makefile for workflow

Make is a build tool from the GNU Project. Mostly it is used to control compiling a huge number of files ans solve dependencies. For simplify your life and enhance your productivity you can use is also the make tool to prepair the target in the ghdl simulation. The makefile is like a script with sequential commands. Sometimes rules are insert. If you have larger designs, you should also divide the VHDL design into different files. I use **make** for this. Store the makefile into the same folder. For run in GHDL type make, this starts all commandos under all. And for view inside the timingdiagram type **make view**. You find many examples but I give you my makefile for a simple start in GHDL application.

```
all:
        rm -rf work
        mkdir work

        ghdl -a  --work=work --workdir=work top_capitalisation.vhd
        ghdl -a  --work=work --workdir=work capitalisation.vhd
        ghdl -a  --work=work --workdir=work ../rx/UART_RX_8N1.vhd
        ghdl -a  --work=work --workdir=work ../tx/UART_TX_8N1.vhd
        ghdl -a  --work=work --workdir=work capitalisation.vhd

        ghdl -a  --work=work --workdir=work tb_capitalisation.vhd
        ghdl -e  --workdir=work -Pwork tb_capitalisation
        ghdl -r tb_capitalisation --wave=tbench.ghw --stop-time=700us


view:
        gtkwave tbench.ghw a.gtkw
```

6 file in/out

In former section we had simulate the whole desgin. In larger designs this can be difficult. It needs time to run and it can be also possible some parts are not read. You have to divide the design in moduls and you have to test the moduls by itselfs. Tested moduls can be integated inside the design. The simple 8bit interface is an good standard. Now I show you. How a module can be tested with datas from an file and also put the output into a file. Sometimes is this simpiler to check as a view in a timing diagram.



The file **test.txt** is read and the characters goes into the capitalisation and are written in the file **text1.txt**.

```
--tb_file.vhd
-----------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_file IS
END tb_file;
--vhdl no seek or rewind function in VHDL

ARCHITECTURE behavior OF tb_file IS

signal clk          : std_logic := '0';
signal reset        : std_logic;

signal rdata            : std_logic_vector(7 downto 0);
signal rd_en        : std_logic;
signal rd           : std_logic;

signal wdata        : std_logic_vector(7 downto 0);
signal wr_en        : std_logic:='1';
signal wr           : std_logic;

    -- Component Declaration for the Unit Under Test (UUT)
component capitalisation is
  port(
    clk        : in  std_logic;
    reset      : in  std_logic;
    --in
    rdata      : in  std_logic_vector(7 downto 0);
    rd_en      : in  std_logic;
    rd         : out std_logic;
    --out
    wdata      : out std_logic_vector(7 downto 0);
    wr_en      : in  std_logic;
    wr         : out std_logic
    );
end component;

   -- Clock period definitions
   constant clk_period : time := 10 ns;

subtype by_te is character;
type f_byte is file of by_te;

BEGIN

process(clk)
begin
```

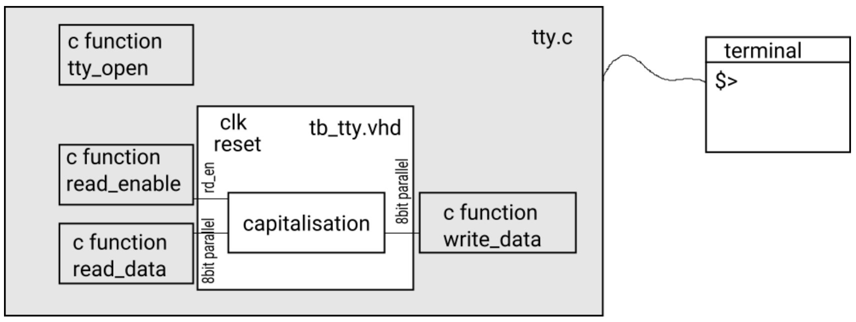
```
  if rising_edge (clk) then
    if reset='0' then
      wr_en<='1';
    else
      wr_en<='0';
    end if;
  end if;
end process;

--read a file
process (reset,clk)
constant file_name: string:="test.txt";
file in_file: f_byte open read_mode is file_name;

variable a:character;

begin
  if reset='1' then
    rd_en<='0';
  else

    if rising_edge(clk) then
      if rd_en='0' or rd='1' then
        if not endfile (in_file) then
          read(in_file,a);
          rdata<=std_logic_vector(to_unsigned(character'pos(a),8));
          --very tricky the conversation
          rd_en<='1';
        else
          rd_en<='0';
        end if;
      end if;
    end if;
       --wait until rising_edge(CLK) and rd='1';
  end if;
end process;



--write a file
process (clk)
constant file_name: string:="test1.txt";
file out_file: f_byte open write_mode is file_name;

----variable in_line,out_line: line;
variable b:character;

  begin
    if rising_edge(CLK) then
      if reset='0' then
        if wr='1' then
          b:=character'val(to_integer(unsigned(wdata)));
          write(out_file,b);
        end if;
      end if;
    end if;
  end process;

   stim_proc : process
     begin
       reset <= '1';
      wait for 50 ns;
         reset <='0';

      wait;
    end process;

  clk_process :process
   begin
                 clk <= '0';
                 wait for clk_period/2;
                 clk <= '1';
                 wait for clk_period/2;
   end process;

engine: capitalisation
  port map(

    clk        => clk,
    reset      => reset,

    rdata      => rdata,
    rd_en      => rd_en,
    rd         => rd,

    wdata      => wdata,
    wr_en      => wr_en,
    wr         => wr
     );

END;
```

```
[test.txt]
Hallo world! Can you see my text?

[test1.txt]
HALLO WORLD! CAN YOU SEE MY TEXT?
```

VHDL has only restricted file operations. It is not possible to seek the fileposition.

7 VHPI Interface to an other language

Here I show one of the highest feature of GHDL. It is possible to link code from another language into the simulation. This can be used for a model or an interface to a real hardware. The possible speed is lower as in an FPGA. For testing is this a change to use mor realistic posibilities. You can data interchange of other programs, for instance testing software. In our case the simulation can communicate with a comport emulator. Linux has pseudoterminals witch is a serial interface in driver lowlever exchange of programs. First start the c source file. There are only some c functions.



```
/*
  VPI code allowing you to connect terminal emulator or other program to pty "connected"
  to the UART-like port in IP core simulated in GHDL.

  This code is written by Wojciech M. Zabolotny (wz...@ise.pw.edu.pl) on 2nd June 2011
  and is published as PUBLIC DOMAIN

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/ioctl.h>
#include <sys/stat.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>

char *ptsname(int fd);

int ptyf = -1;



int tty_open(int a)
{
  ptyf = open("/dev/ptmx",O_RDWR);
  if(ptyf<0) {
    perror("I␣can't␣open␣pseudoterminal\n");
    return -1;
  }
  if(unlockpt(ptyf)<0) {
    perror("I␣can't␣unlock␣pseudoterminal\n");
    return -1;
  }
  if(grantpt(ptyf)<0) {
    perror("I␣can't␣grant␣pseudoterminal\n");
    return -1;
  }
  printf("Pseudoterminal:␣%s\n",ptsname(ptyf));
 // sleep(10);
  return 0;
}


int read_enable(void)
{
  //In the masks below you may omit POLLHUP in this case
  //disconnection of the terminal emulator from pty will not
  //stop simulation, and you'll be able to reconnect
  //the same or different program to pty and running simulation
  struct pollfd pfd[1]={{ptyf,POLLIN | POLLERR | POLLHUP,0}};
  int res;
  res=poll(pfd,1,0);
  if(res==0) return 0;
  if(res<0) return 0; //error
  //If you removed POLLHUP from the mask above, you should remove it below too
  if(pfd[0].revents & (POLLERR|POLLHUP)) return 0; //disconnected or error?
  if(pfd[0].revents & POLLIN)       return 1;
  return 0;
}


int read_data(void)
{
  unsigned char c;
  read(ptyf,&c,1);
  return c;
}

int write_data(int byte)
{
  unsigned char c = byte;
  write(ptyf,&c,1);

  // Debug: printf("Writing %x to pty\n", c);
  return 0;
}
```

This package is the glue between external code. All c functions get a wrap of a vhdl definition. That makes possible to call a c function in vhdl. This is the advantage of this VHPI interface.

```
--tty_pkg.vhd
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package tty_pkg is

  function to_integer( s: std_logic) return integer;
  function to_std_logic( s : integer ) return std_logic;

  function tty_open (portn : integer) return integer;
  attribute foreign of tty_open :
    function is "VHPIDIRECT tty_open";

  function read_data ( dummy: integer) return integer;
  attribute foreign of read_data :
    function is "VHPIDIRECT read_data";

  function read_enable ( dummy: integer) return integer;
  attribute foreign of read_enable :
    function is "VHPIDIRECT read_enable";

  procedure write_data ( data: in integer);
    attribute foreign of write_data :
    procedure is "VHPIDIRECT write_data";

end;


package body tty_pkg is

  function to_integer( s : std_logic ) return integer is
  begin
    if s = '1' then
      return 1;
    else
      return 0;
    end if;
  end function;

  function to_std_logic( s : integer ) return std_logic is
  begin
    if s > 0 then
      return '1';
    else
      return '0';
    end if;
  end function;


  function tty_open (portn : integer) return integer is
  begin
    assert false report "VHPI" severity failure;
  end tty_open;

  function read_data (dummy: integer) return  integer is
  begin
    assert false report "VHPI" severity failure;
  end read_data;

   function read_enable (dummy: integer) return  integer is
  begin
    assert false report "VHPI" severity failure;
  end read_enable;

  procedure write_data ( data: in integer) is
  begin
    assert false report "VHPI" severity failure;
  end write_data;
end tty_pkg;
```

Also for that case the testbench.

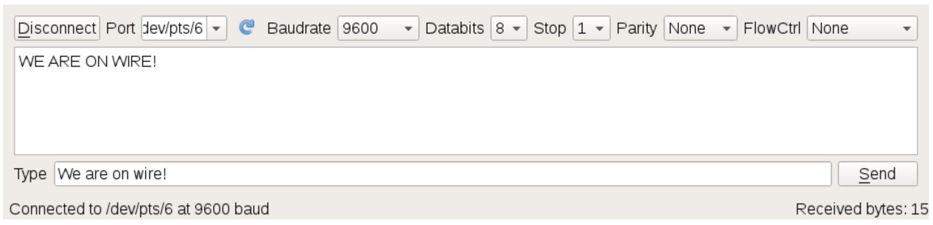
```
--tb_tty.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use work.tty_pkg.all;
```

```
ENTITY tb_tty IS
END tb_tty;

ARCHITECTURE behavior OF tb_tty IS

signal clk          : std_logic := '0';
signal reset        : std_logic;

signal data_in          : std_logic_vector(7 downto 0);
signal wdata            : std_logic_vector(7 downto 0);
signal wr_en        : std_logic:='1';
signal wr           : std_logic;
signal rd_en        : std_logic;
signal rd           : std_logic;
signal a            : integer;
signal c            : integer;


component capitalisation is
  port(
    clk        : in  std_logic;
    reset      : in  std_logic;
    --in
    rdata      : in  std_logic_vector(7 downto 0);
    rd_en      : in  std_logic;
    rd         : out std_logic;
    --out
    wdata      : out std_logic_vector(7 downto 0);
    wr_en      : in  std_logic;
    wr         : out std_logic
    );
end component;

   -- Clock period definitions
   constant clk_period : time := 10 ns;

subtype by_te is character;
type f_byte is file of by_te;

BEGIN

--file open
  process
    begin c<=tty_open(0);
    wait;
  end process;

--read
process (clk)

variable b: integer;
begin
  if rising_edge(CLK) then
    a<= read_enable(0);
    if a=1 then
      data_in<=std_logic_vector(to_unsigned(read_data(0),8));
      rd_en<='1';
    else
      rd_en<='0';
    end if;
  end if;
end process;

--write
process (clk)
variable b: integer;

  begin
    if rising_edge(CLK) then
      if reset='0' then
        if wr='1' then
          b:=to_integer(unsigned(wdata));
          write_data(b);
        end if;
      end if;
    end if;
  end process;

   stim_proc : process
     begin
       reset <= '1';

      wait for 50 ns;
         reset <='0';
      wait;
    end process;

  clk_process :process
   begin
                 clk <= '0';
                 wait for clk_period/2;
                 clk <= '1';
                 wait for clk_period/2;
   end process;

engine: capitalisation
  port map(
    clk         => clk,
    reset       => reset,
    --in
    rdata       => data_in,
    rd_en       => rd_en,
    rd          => rd,
    --out
    wdata       => wdata,
    wr_en       => wr_en,
    wr          => wr
    );
END;
```

And the makefile that you can compile and link all together.

```
all:

        rm -rf work
        mkdir work
        ghdl -a  --work=work --workdir=work tty_pkg.vhd
        gcc -c -fPIC tty.c  -o tty.o

        ghdl -a  --work=work --workdir=work ../capitalisation/capitalisation.vhd
        ghdl -a  --work=work --workdir=work tb_tty.vhd

        ghdl -e -Wl,tty.o  --ieee=synopsys -fexplicit --workdir=work -Pwork tb_tty
#        ghdl -r tb_tty  --wave=tbench.ghw
        ghdl -r tb_tty  --wave=tbench.ghw  --stop-time=500000us


view:
        gtkwave tbench.ghw a.gtkw
```

In action the simulation prints out the name of the pseudotherminal. I can open a serial terminal program and send datas to the simulation and can also receive the output.


Disconnect Port /dev/pts/6 Baudrate 9600 Databits 8 Stop 1 Parity None FlowCtrl None

```
WE ARE ON WIRE!
```

Type We are on wire! Send

Connected to /dev/pts/6 at 9600 baud   Received bytes: 15

8 closing words

The chapter of testbench is very short in the most books. In this article you see the code for testing is adequate to code of application. A simulation is irrecoverable for good results. The development is only effective with simulation. Debug at a later development phase is very difficult. The testbench is the pusher for all effects. The code should also be reusable code. The generic is an interface to set some parameter for a simpler practical integration in the whole application. I hope you could follow all demonstrations and give you idea for better work.

```
the procedure tx_char was posted of Lothar Miller on
http:\\www.mikrocontroller.net

the VHPI interface for pseudotherminal was posted
of Wojciech M. Zabolotny on comp.arch.FPGA 2011
```