© ALSE - Sept 2001

VHDL - Practical Example - Designing an UART

Bertrand CUZEAU
Technical Manager - ALSE
ASIC / FPGA Design Expert
Doulos HDL Instructor (Verilog-VHDL)
info@ALSE-FR.COM
http://www.alse-fr.com
☎ : 33.(0)1 45 82 64 01

We will demonstrate, on a “real-life” example, how a sound HDL methodology can be used in conjunction with modern synthesis and simulation tools.

Note : the source code we provide here if for teaching purpose only.

This code belongs to ALSE.
If you want to use it in your projects please contact us.

© Bertrand CUZEAU - info@alse-fr.com

UART Specification

We want to address the following needs :

- Transmit / Receive with h/w handshake
- “N81” Format , but plan for parity
- Speed : 1200..115200 baud (Clock = 14.7456 MHz)
- No internal Fifo (usually not needed in an FPGA !)
- Limited frame timing checks

© Bertrand CUZEAU - info@alse-fr.com

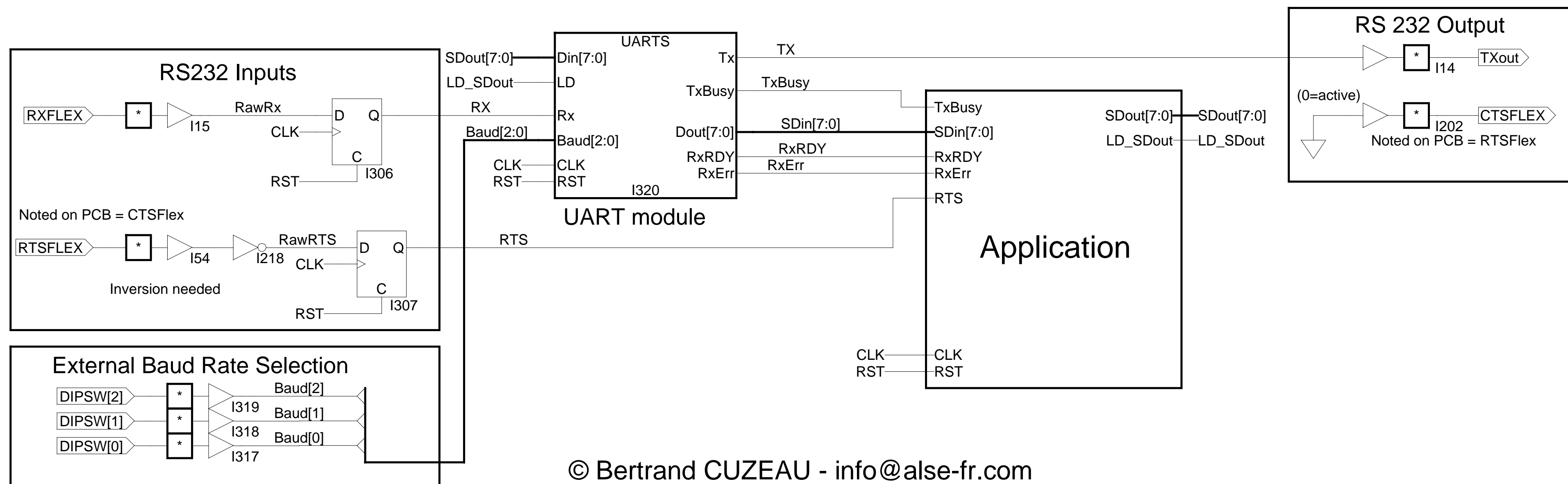
Methodology

We adopt the following constraints :

- Standard & 100% portable VHDL Description :
  - Synthesis
  - Simulation
  - Target FPGA (or CPLD)
- Complete functional Simulation with file I/O.
- Should work "in vivo" on an existing ALSE demo board.

© Bertrand CUZEAU - info@alse-fr.com

Application Architecture



© Bertrand CUZEAU - info@alse-fr.com

Baud Rate Generator

- Embedded in UARTS.
- Divides by 8, 16, 28, 48, 96, 192, 384 or 768 and builds Top16.
- Generates two “ticks” by further dividing Top16 :
  - Transmit : TopTx, fixed rate
  - Receive : TopRx, mid-bit, resynchronized

© Bertrand CUZEAU - info@alse-fr.com

```vhdl
-- -------------------------
--  Baud rate selection
-- -------------------------
process (RST, CLK)
begin
  if RST='1' then
    Divisor <= 0;
  elsif rising_edge(CLK) then
    case Baud is
      when "000" => Divisor <=   7;  -- 115.200
      when "001" => Divisor <=  15;  --  57.600
      when "010" => Divisor <=  23;  --  38.400
      when "011" => Divisor <=  47;  --  19.200
      when "100" => Divisor <=  95;  --   9.600
      when "101" => Divisor <= 191;  --   4.800
      when "110" => Divisor <= 383;  --   2.400
      when "111" => Divisor <= 767;  --   1.200
      when others => Divisor <=  7;  -- n.u.
    end case;
  end if;
end process;
```

```vhdl
-- -------------------------
--  Clk16 Clock Generation
-- -------------------------
process (RST, CLK)
begin
if RST='1' then
    Top16 <= '0';
    Div16 <= 0;

  elsif rising_edge(CLK) then
    Top16 <= '0';
    if Div16 = Divisor then
      Div16 <= 0;
      Top16 <= '1';
    else
      Div16 <=  Div16 + 1;
    end if;
  end if;
end process;
```

```vhdl
-- -------------------------
--  Tx Clock Generation
-- -------------------------
process (RST, CLK)
begin
  if RST='1' then
    TopTx <= '0';
    ClkDiv <= (others=>'0');
  elsif rising_edge(CLK) then
    TopTx <= '0';
    if Top16='1' then
      ClkDiv <=  ClkDiv + 1;
      if ClkDiv = 15 then
        TopTx <= '1';
      end if;
    end if;
  end if;
end process;
```

```vhdl
-- -----------------------------
--  Rx Sampling Clock Generation
-- -----------------------------
process (RST, CLK)
begin
  if RST='1' then
    TopRx <= '0';
    RxDiv <= 0;
  elsif rising_edge(CLK) then
    TopRx <= '0';
    if ClrDiv='1' then
      RxDiv <= 0;
    elsif Top16='1' then
      if RxDiv = 7 then
        RxDiv <= 0;
        TopRx <= '1';
      else
        RxDiv <=  RxDiv + 1;
      end if;
    end if;
  end if;
end process;
```

© Bertrand CUZEAU - info@alse-fr.com

Transmitter

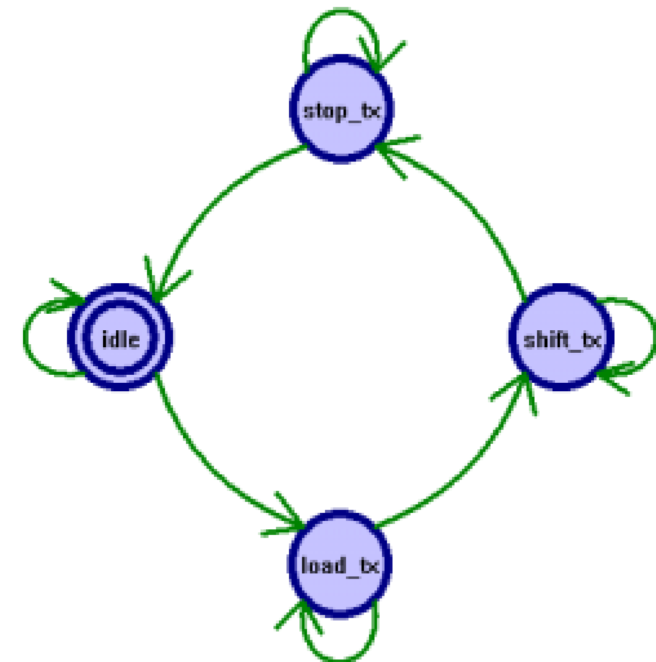
We use a very simple State Machine to control the transmit shift register. The FSM inputs are :

- LD : Loads the character to transmit (Din)
- TopTx : Bit shifting command

For simplicity we code the FSM as a "re-synchronized Mealy".

© Bertrand CUZEAU - info@alse-fr.com

```
-- --------------------------
--  Transmit State Machine
-- --------------------------

TX <= Tx_Reg(0);

Tx_FSM: process (RST, CLK)
begin
  if RST='1' then
    Tx_Reg    <= (others => '1');
    TxBitCnt <= 0;
    TxFSM     <= idle;
    TxBusy    <= '0';
    RegDin    <= (others=>'0');

  elsif rising_edge(CLK) then

    TxBusy <= '1';   -- except when explicitly '0'
    case TxFSM is

      when Idle =>
        if LD='1' then
          -- latch the input data immediately.
          RegDin <= Din;
          TxBusy <= '1';
          TxFSM <= Load_Tx;
        else
          TxBusy <= '0';
        end if;

      when Load_Tx =>
        if TopTx='1' then
          TxFSM <= Shift_Tx;
          if parity then
            -- start + data + parity
            TxBitCnt <= (NDBits + 2);
            Tx_Reg <= make_parity(RegDin,even) & Din & '0';
          else
            TxBitCnt <= (NDBits + 1);    -- start + data
            Tx_reg <=  '1' & RegDin & '0';
          end if;
        end if;

      when Shift_Tx =>
        if TopTx='1' then
          TxBitCnt <= TxBitCnt - 1;
          Tx_reg <= '1' & Tx_reg (Tx_reg'high downto 1);
          if TxBitCnt=1 then
            TxFSM <= Stop_Tx;
          end if;
        end if;

      when Stop_Tx =>
        if TopTx='1' then
          TxFSM <= Idle;
        end if;

      when others =>
        TxFSM <= Idle;

    end case;
  end if;
end process;
```



© Bertrand CUZEAU - info@alse-fr.com

Receiver

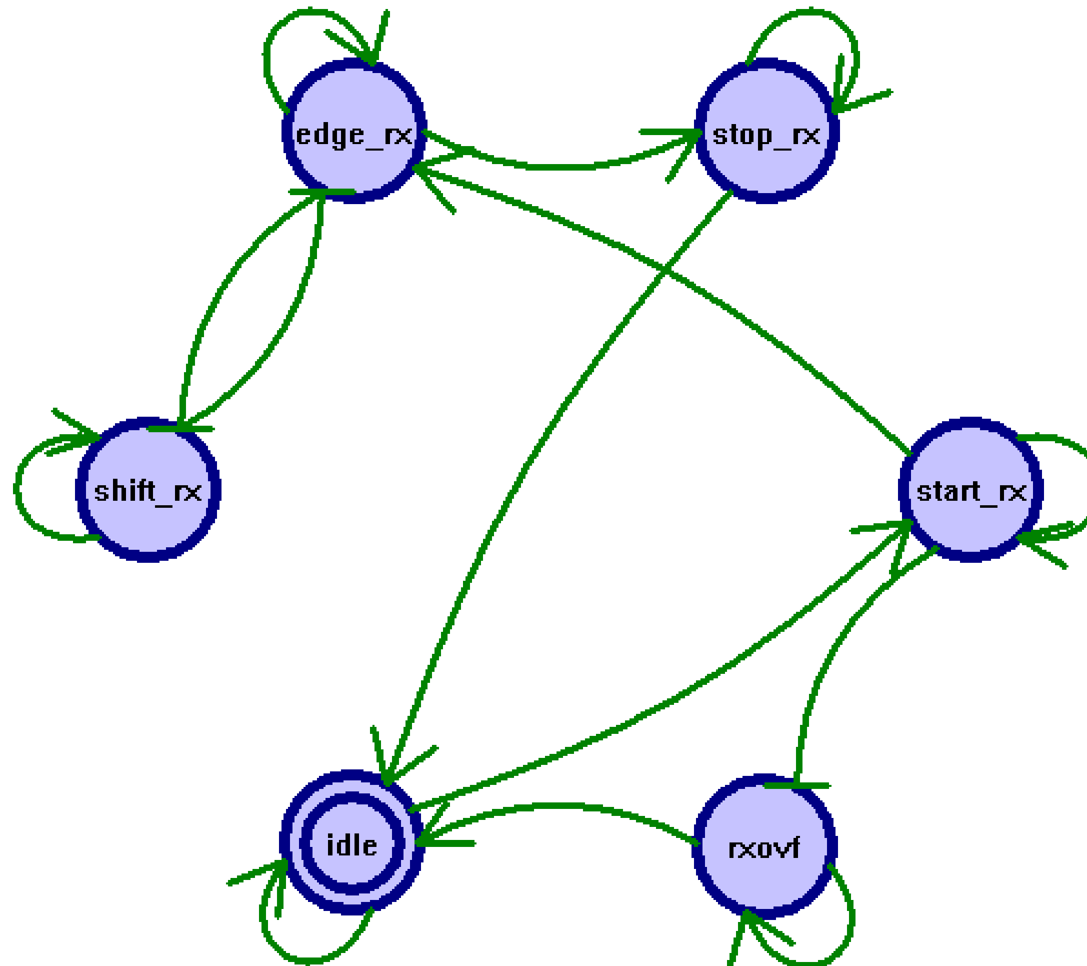
We also use a State Machine :

- Wait RX (Start bit) falling edge,
- Synchronize the Half-bit counter
- Sample RX at mid-bit and verify the Start bit
- Loop on the data bits (+ parity) :
  * Skip transition
  * Sample at mid-bit
- Sample and Test Stop bit
- Return to Idle state (waiting for a new Start condition)

© Bertrand CUZEAU - info@alse-fr.com

```
-- -----------------------
--  RECEIVE State Machine
-- -----------------------
Rx_FSM: process (RST, CLK)
begin
  if RST='1' then
    Rx_Reg   <= (others => '0');
    Dout     <= (others => '0');
    RxBitCnt <= 0;
    RxFSM    <= Idle;
    RxRdyi   <= '0';
    ClrDiv   <= '0';
    RxErr    <= '0';

  elsif rising_edge(CLK) then

    ClrDiv   <= '0'; -- default value
    -- reset error when a word has been received Ok:
    if RxRdyi='1' then
      RxErr  <= '0';
      RxRdyi <= '0';
    end if;

    case RxFSM is

      when Idle =>  -- wait on start bit
        RxBitCnt <= 0;
        if Top16='1' then
          if Rx='0' then
            RxFSM  <= Start_Rx;
            ClrDiv <='1'; -- Synchronize the divisor
          end if; -- else false start, stay in Idle
        end if;

      when Start_Rx =>  -- wait on first data bit
          if TopRx = '1' then
            if Rx='1' then -- framing error
              RxFSM <= RxOVF;
              report "Start bit error." severity note;
            else
              RxFSM <= Edge_Rx;
            end if;
          end if;

      when Edge_Rx =>  -- should be near Rx edge
        if TopRx = '1' then
          RxFSM <= Shift_Rx;
          if RxBitCnt = NDbits then
            RxFSM <= Stop_Rx;
          else
            RxFSM <= Shift_Rx;
          end if;
        end if;

      when Shift_Rx =>  -- Sample data !
        if TopRx = '1' then
          RxBitCnt <= RxBitCnt + 1;
          -- shift right :
          Rx_Reg <= Rx & Rx_Reg (Rx_Reg'high downto 1);
          RxFSM <= Edge_Rx;
        end if;

      when Stop_Rx =>  -- during Stop bit
        if TopRx = '1' then
          Dout <= Rx_reg;
          RxRdyi <='1';
          RxFSM <= Idle;
          assert (debug < 1)
            report "Character received in decimal is : "
              & integer'image(to_integer(unsigned(Rx_Reg)))
              severity note;
        end if;

      when RxOVF =>  -- Overflow / Error
        RxErr <= '1';
        if Rx='1' then
          RxFSM <= Idle;
        end if;

    end case;
  end if;
end process;
```

Receiver State Machine



© Bertrand CUZEAU - info@alse-fr.com

To test our UART, we use a trivial “application” which increments the characters received and resends them !

(Example : “A”→“B”, “f”→“g”, “HAL”→”IBM”…)

This way, it is easy to verify the reveive and transmit operations, both by simulation and on the demo board.

© Bertrand CUZEAU - info@alse-fr.com

```vhdl
-- APPLIC.vhd
-- ------------------------------------------------------
--    Demo for UART module
-- ------------------------------------------------------
-- Bertrand Cuzeau / info@alse-fr.com
-- Receives a char, and re-emits the same char + 1

LIBRARY ieee;
  USE ieee.std_logic_1164.ALL;
  USE ieee.numeric_std.ALL;

-- ------------------------------------------------------
    Entity APPLIC is
-- ------------------------------------------------------
      Port (     CLK : In  std_logic;
                 RST : In  std_logic;
                 RTS : In  std_logic;
               RxErr : In  std_logic;
               RxRDY : In  std_logic;
                SDin : In  std_logic_vector (7 downto 0);
              TxBusy : In  std_logic;
            LD_SDout : Out std_logic;
               SDout : Out std_logic_vector (7 downto 0)
            );
end APPLIC;


-- ------------------------------------------------------
    Architecture RTL of APPLIC is
-- ------------------------------------------------------
type State_Type is  (Idle, Get, Send);
signal  State : State_Type;

signal  RData : std_logic_vector (7 downto 0);
signal  SData : std_logic_vector (7 downto 0);

begin

SDout <= SData;

process (CLK,RST)
begin
  if RST='1' then
    State <= Idle;
    LD_SDout <= '0';
    SData <= (others=>'0');
    RData <= (others=>'0');

  elsif rising_edge(CLK) then

    LD_SDout <= '0';

    case State is

      when Idle =>
        if RxRDY='1' then
          RData <= SDin;
          State <= Get;
        end if;

      when Get =>
        if (TxBusy='0') and (RTS='1') then
          case to_integer(unsigned(RData)) is
            when 10|13|32 => -- do not translate Cr Lf Sp !
              SData <= RData;
            when others =>
              SData <= std_logic_vector(unsigned(RData)+1);
          end case;
          State <= Send;
        end if;

      when Send =>
        LD_SDout <= '1';
        State <= Idle;

      when others => null;
    end case;
  end if;
end process;

end RTL;
```

© Bertrand CUZEAU - info@alse-fr.com

Test Bench

The VHDL Test Bench simply sends the ASCII character ‘A’ and displays the character(s) sent back by the system.

It is based on two behavioral UART routines (described in another of our conferences).

A much more sophisticated Test Bench (with file I/O and console emulation with inter-character spacing) is provided by ALSE in the commercial version.

© Bertrand CUZEAU - info@alse-fr.com

```vhdl
-- ---------------------------------------------
--  Simple VHDL test bench for UART Top_Level
-- ---------------------------------------------
-- (c) ALSE - Bertrand Cuzeau
-- info@alse-fr.com

  USE std.textio.all;
LIBRARY ieee;
  USE ieee.std_logic_1164.ALL;
  USE ieee.numeric_std.ALL;
  USE ieee.std_logic_textio.ALL;

entity testbench is
end testbench;
-- ---------------------------------------------
Architecture TEST of testbench is
  component ALSE_UART
  Port (      RST : In  std_logic;
              CLK : In  std_logic;
           RXFLEX : In  std_logic;
          RTSFLEX : In  std_logic;
            DIPSW : In  std_logic_vector (2 downto 0);
          CTSFLEX : Out std_logic;
            TXout : Out std_logic );
  end component;

  constant period : time := 68 ns;
  constant BITperiod : time := 8680 ns; -- 115.200
  signal   RSData : std_logic_vector (7 downto 0);
  signal      CLK : std_logic := '0';
  signal      RST : std_logic;
  signal   RXFLEX : std_logic;
  signal  RTSFLEX : std_logic;
  signal    DIPSW : std_logic_vector (2 downto 0);
  signal  CTSFLEX : std_logic;
  signal    TXout : std_logic;
begin
-- UUT Instanciation  :
UUT : ALSE_UART
 Port Map (CLK=>CLK, CTSFLEX=>CTSFLEX, DIPSW=>DIPSW,
           RST=>RST, RTSFLEX=>RTSFLEX, RXFLEX=>RXFLEX,
           TXout=>TXout );
-- Clock, Reset & DIP-Switches
RST <= '1', '0' after period;
CLK <= not CLK after (period / 2);
DIPSW <= "000";   - - 115.200 bauds
RSData <= x"41";   -- 'A'
RTSFlex <= '0';
-- Emission d'un caractère
process begin
 RXFLEX <= '1'; -- Idle
 wait for 100 * period;
 RXFLEX <= '0'; -- Start bit
 wait for BITperiod;
 for i in 0 to 7 loop
    RXFLEX <= RSData(i); wait for BITperiod;
 end loop;
 RXFLEX <= '1'; -- Stop bit
 wait for BITperiod;
wait;
end process;
-- Reception
process
  Variable L   : line;
  Variable MOT : std_logic_vector (7 downto 0);
begin
loop
   wait until TXout='0';        -- get falling edge
   wait for (0.5 * BITperiod); -- Middle of Start bit
   assert TXout='0'
     report "Start Bit Error ???" severity warning;
   wait for BITperiod;          -- First Data Bit
   for i in 0 to 7 loop         -- Get word
     MOT(i) := TXout; wait for BITperiod;
   end loop;
   wait for BITperiod;          -- Stop bit
   assert TXout='1'
     report "Stop bit Error ???" severity warning;
   WRITE (L,string'("Character received (hex) = "));
   HWRITE (L,MOT);        -- trace :
   WRITELINE (output,L);        -- char -> transcript
 end loop;
end process;
end TEST;
```

After the theory, we are now going to follow the entire design flow, down to the Demo Board (~10 minutes) :

1. Build the Project
2. Syntactic Verification
3. Unitary Functional Simulation
4. Unitary Logic Synthesis
5. System-Level Simulation
6. Global Synthesis
7. Place and Route
8. Download & tests on the demo board (using HyperTerminal !)

© Bertrand CUZEAU - info@alse-fr.com

Conclusion

It takes *less than a working day* to design and test a simple UART like this one. Powerful HDL languages, as well as capable Simulation and Synthesis Tools are now widely available.

With the right methodology and some design practice, projects that used to be considered as "complex" become almost trivial.

Note : an enhanced version of this UART, still simple and efficient, is available at ALSE, at a very affordable cost.