

© ALSE - Sept 2001

VHDL - Exemple Pratique - Conception d'une 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

Introduction



Afin de démontrer la simplicité de mise en oeuvre des méthodologies et des outils modernes, nous allons développer un petit module UART dont nous vous fournissons les sources à titre pédagogique uniquement.

Cette application appartient à ALSE.

Si vous souhaitez l'utiliser dans vos projets, veuillez nous contacter.

Cahier des Charges



Le besoin se réduit à un type précis de transmissions :

- Emission / Réception avec handshake hardware
- Format “N81”, mais prévoir la parité
- Vitesse 1200..115200, horloge 14.7456 MHz
- Pas de Fifo (en général inutile dans un FPGA !)
- Contrôles limités de timing trame

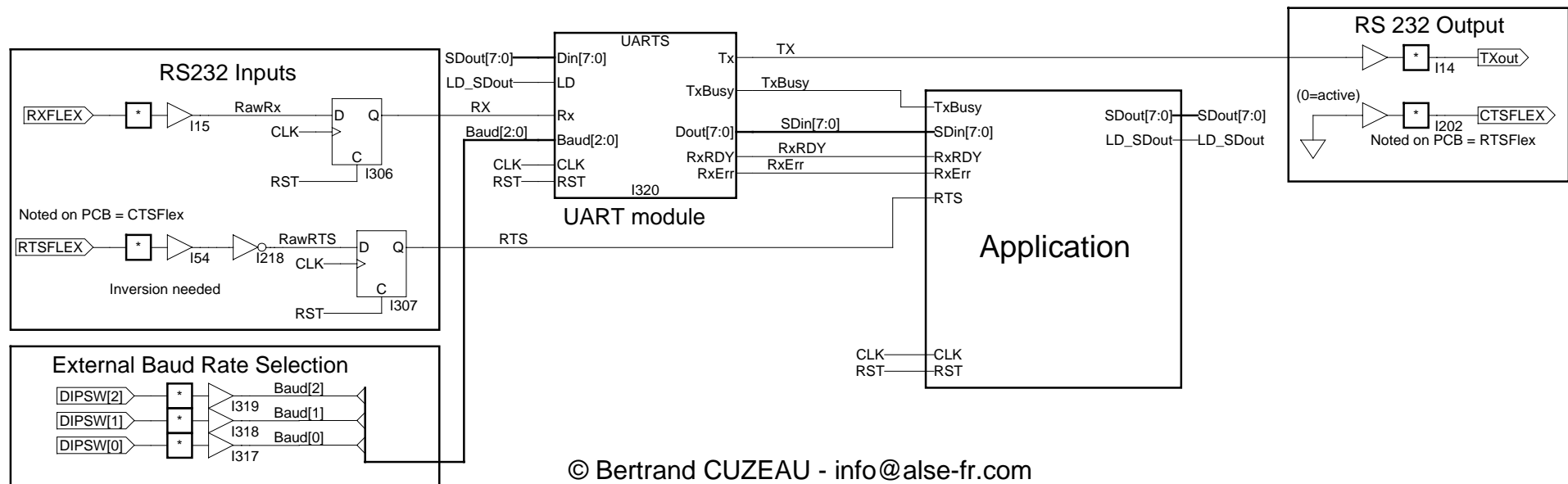
Méthodologie



Nous nous fixons les contraintes suivantes :

- Description VHDL standard et 100% portable :
 - Synthèse
 - Simulation
 - Composant ciblé
- Simulation fonctionnelle complète avec E/S fichiers
- Application sur une maquette pour tests “in vivo”

Architecture de l'Application



Générateur de Bauds



- Inclus dans le module UARTS
- Divise par 8, 16, 28, 48, 96, 192, 384 ou 768 pour obtenir l'impulsion Top16
- Génère deux “tops” par division par 16 de Top16 :
 - Emission : TopTx, fixe
 - Réception : TopRx en mid-bit, resynchronisable à 1/16

```

-----
-- Baud rate conversion
-----
process (RST, CLK)
begin
  if RST='1' then
    Divisor <= 0;
  else if 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;

```

```

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

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

```

```

-----
-- Tx Clock Generation
-----
process (RST, CLK)
begin
  if RST='1' then
    TopTx <= '0';
    ClkDiv <= (others=>'0');
  else if 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;

```

```

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

```

L'Emetteur

C'est une Machine d'Etats très simple qui contrôle un registre à décalage en répondant à :

- LD : Chargement du mot à émettre (D_i n)
- TopTx : pulse de décalage

Par simplicité, nous avons choisi une structure de type “Mealy resynchronisée”.

```
-----
-- 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;
```

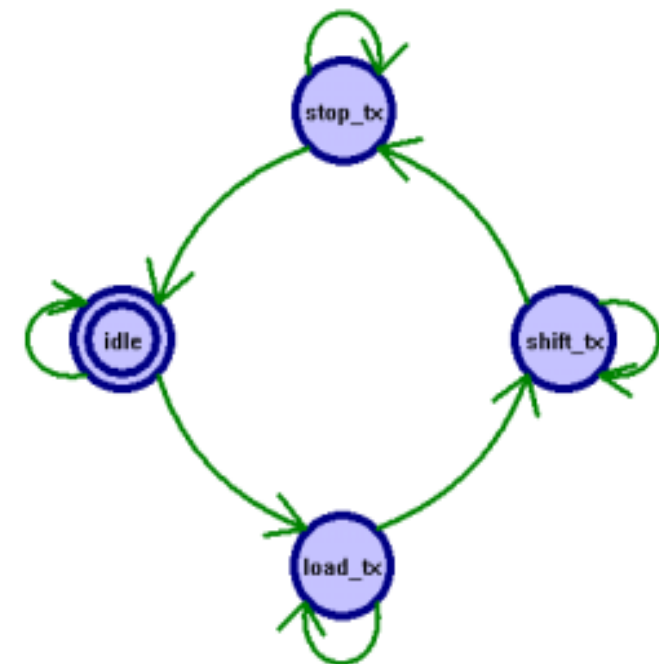
```
      when others =>
```

```
        TxFSM <= Idle;
```

```
    end case;
```

```
  end if;
```

```
end process;
```



Le Récepteur

On fait appel ici aussi à une Machine d'Etats :

- Attente d'un front descendant de RX (Start bit),
- Synchronisation du compteur de demi-bit
- Echantillonnage mid-bit du Start bit
- Boucle sur le nombre de bits de données (+ parité) :
 - * Saut de la transition
 - * Echantillonnage mid-bit
- Test du Stop bit
- Retour en attente de Start

```

-----
-- 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;
    end case;
  end if;
end process;

```

```

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

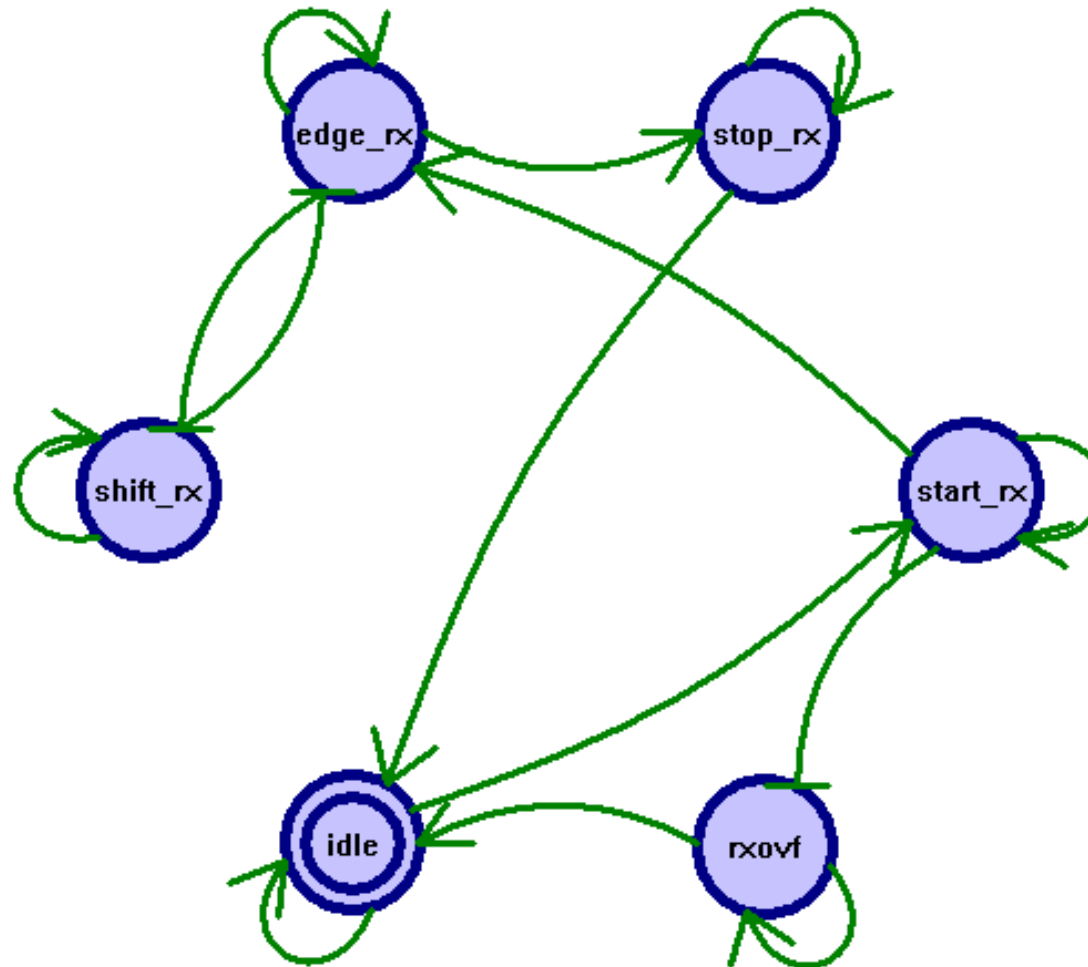
when Shift_Rx => -- Sample data !
  if TopRx = '1' then
    RxBitCnt <= RxBitCnt + 1;
    -- shift right :
    Rx_Reg <= Rx & Rx_Reg (Rx_Reg' high downto 1);
    RxFSM <= Edge_Rx;
  end if;

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 process;

```

Machine d'Etats de Réception



Application de Test



Pour tester notre UART, nous utiliserons une pseudo-application triviale qui se contente de ré-émettre les caractères reçus, après les avoir incrémentés !

(Exemples : “A”→“B”, “f”→“g”, “HAL”→”IBM”...)

Il sera ainsi facile de vérifier le fonctionnement du système, tant en simulation que sur la maquette.

```

-- 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;

```

Banc de Test



Le Banc de test envoie simplement un caractère ASCII ('A') et trace le/les caractères (r)envoyés par le système.

Il donc est basé sur les deux routines d'E/R asynchrones (comportementales) décrites dans une précédente conférence.

Un banc de test plus sophistiqué (livré dans l'IP d'ALSE) effectue une entrée/sortie de caractères depuis des fichiers, en émulant ainsi la console et des délais inter-caractères variables.

```

-----
-- 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 RSDData : 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
RSDData <= 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 <= RSDData(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;

```

Mise en Oeuvre



Pour terminer cette présentation, nous allons maintenant mettre en oeuvre cette application (environ 10 minutes) :

1. Construction du Projet
2. Vérification syntaxique
3. Simulation fonctionnelle unitaire
4. Synthèse logique unitaire
5. Simulation fonctionnelle globale
6. Synthèse Globale
7. Placement-routage
8. Mise en Oeuvre sur la maquette (test par HyperTerminal !)

Conclusion



La conception initiale de l'UART que nous venons de voir représente environ une journée de travail (tests et mise en oeuvre compris). Nous voyons donc que les outils modernes et les méthodologies HDL permettent au concepteur averti de développer très rapidement et très sûrement des applications sur mesure, sans nécessairement avoir à faire appel à des compétences extérieures.

Cet UART, simple et efficace, est toutefois disponible (avec des améliorations) auprès d'ALSE pour un coût très modique.