UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Sistema UART

Manual de programador

TAREA 6 DE LA ASIGNATURA

Dispositivos lógicos programables

REALIZADO POR

Sergio Bou Grau

Contenido

Proyecto: documentación UART 1

1. Timing 1
2. Transmit 3
3. Receive 5
4. UART 8
5. Uart_comps 9

Proyecto: documentación UART

El siguiente proyecto tiene como finalidad el estudio y descripción de un proyecto informático escrito en lenguaje VHDL para comprender su funcionamiento y la utilidad de cada bloque que lo componen. El proyecto consta de cinco ficheros que se enumeran y describen a continuación.

1. Timing

Este fichero se encarga de gestionar la elección de la velocidad de transmisión y la generación de los relojes de la velocidad, de la transmisión y de la recepción.

En primer lugar, en este fichero tenemos incluidas las siguientes librerías:

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

El entity está compuesto por las siguientes entradas y salidas:

```
Entity timing is
        generic (F : natural := 50000;
                  min_baud : natural :=  1200);
        port (
                CLK :    in std_logic;
                 RST :   in std_logic;
                baud :   in std_logic_vector(2 downto 0);
                ClrDiv : in std_logic;
                Top16 : buffer std_logic;
                TopTx : out std_logic;
                TopRx : out std_logic
            );
end timing;
```

En la Architecture tenemos las siguientes señales:

```
        signal baud_value : natural;
        constant max_div : natural := ((F*1000)/(16*min_baud));

        subtype div16_type is natural range 0 to max_div-1;
        signal Div16     : div16_type;

        signal ClkDiv    : integer;
        signal RxDiv     : integer;
```

A continuación tenemos un process que mediante la entrada BAUD compuesta por tres bits nos permite seleccionar entre ocho distintas frecuencias para comunicarnos mediante el protocolo UART, este process es el siguiente:

```
    process (CLK)
      begin
        if rising_edge(CLK) then
         case Baud is
```

```
            when "000" => baud_value <= ((F*1000)/(16*115200));
            when "001" => baud_value <= ((F*1000)/(16*57600));
            when "010" => baud_value <= ((F*1000)/(16*38400));
            when "011" => baud_value <= ((F*1000)/(16*19200));
            when "100" => baud_value <= ((F*1000)/(16*9600));
            when "101" => baud_value <= ((F*1000)/(16*4800));
            when "110" => baud_value <= ((F*1000)/(16*2400));
            when "111" => baud_value <= ((F*1000)/(16*1200));
            when others => baud_value <= ((F*1000)/(16*1200)); -- n.u.
        end case;
      end if;
  end process;
```

A continuación tenemos el process que realiza la cuenta para generar la señal del reloj a partir de la opción elegida en el process anterior:

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

Otro process que tenemos en este fichero es el que realiza la cuenta para el reloj que establece la escritura, que es el siguiente:

```
  process (RST, CLK)
    begin
      if RST='0' then
        TopTx <= '0';
        ClkDiv <= 0;
      elsif rising_edge(CLK) then
        TopTx <= '0';
        if Top16='1' then
          ClkDiv <= ClkDiv + 1;
          if ClkDiv = 15 then
            TopTx <= '1';
            ClkDiv <= 0;
          end if;
        end if;
      end if;
  end process;
```

A continuación y por último tenemos el process que realiza la cuenta del reloj de lectura:

```
process (RST, CLK)
 begin
   if RST='0' 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;
```

2. Transmit

En el fichero “transmit” podemos observar que se gestiona una maquina de estados para la transmisión de los datos.

En primer lugar, tenemos las siguientes librerías:

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

Y en el entity las siguientes entradas y salidas:

```
entity transmit is
  generic (NDBits : natural := 8);
        port (CLK                 : in std_logic;
                        RST       : in std_logic;
                        TopTx     : in std_logic;
                        StartTx   : in std_logic;
                        Din       : in std_logic_vector (NDBits-1 downto 0);
                        Tx        : out std_logic;
                        TxBusy    : out std_logic );
end;
```

En la architecture inicialmente tenemos las siguientes señales:

```
  type State_Type is (Idle, Load_Tx, Shift_Tx, Stop_Tx);
  signal TxFsm : State_Type;
```

```
signal Tx_Reg : std_logic_vector (NDBits downto 0);
signal RegDin : std_logic_vector (NDBits-1 downto 0);

signal TxBitCnt : natural range 0 to 15;
```

A continuación en un único process tenemos la máquina de estados que gestiona la transmisión de datos, en este process en primer lugar mediante una condición observamos que si la entrada "RST" es igual a cero el vector "Tx_Reg" se completa con unos, el vector "RegDin" se completa con ceros, a la señal "TxBitCnt" y a la salida "TxBusy" se les asigna el valor de cero, y se establece en la señal "TxFSM" el estado "idle". Sí no está la entrada "RST" a nivel bajo y la señal del reloj está a nivel bajo se cumple la condición para leer la señal "TxFSM" que puede tener cuatro estados: *Idle, Load_Tx, Shift_Tx y Stop_Tx.*

Estado Idle:

En el estado "idle" si la entrada "StartTx" está a nivel alto se le asigna al vector "RegDin" el valor del vector de entrada "Din", a la salida TxBusy se le asigna nivel alto y a la señal "TxFSM" se le asigna el estado "LoadTx". Sí la señal "StartTx" no estuviese a nivel alto no se cumpliría la condición y se le asignaría nivel bajo a la salida "TxBusy".

Estado LoadTx:

En este estado si la entrada "TopTx" está a nivel alto se le asigna a la señal "TxBitCnt" el valor de "NDBits" incrementado en uno, al vector señal "Tx_reg" se le asignan los valores de "RegDin" con un cero concatenado y por último se le asigna a la señal "TxFSM" el estado de "Shift_Tx".

Estado ShiftTx:

En el estado "ShiftTx" si cumple la condició de que la entrada "TopTx" está a nivel alto se decrementa en uno la señal "TxBitCnt", a continuación a la señal vector "Tx_reg" se le asigna el valor de uno concatenado con el mismo vector desde la posición máxima hasta la posición uno. En la siguiente línea dentro de esta condición se comprueba si el valor de la señal "TxBitCnt es igual a uno, y si se cumple se le asignará el estado de "Stop_Tx" a la señal "TxFSM".

Estado Stop_TX:

En este estado si la entrada "TopTx" tiene un nivel alto se le asignará a la señal "TxFSM" el estado "idle".

```
Tx_FSM: process (RST, CLK)
begin
        if RST='0'then
                Tx_Reg <= (others => '1');
                RegDin <= (others=>'0');
                TxBitCnt <= 0;
                TxBusy <= '0';
                TxFSM <= idle;
        elsif rising_edge(CLK) then
                TxBusy <= '1';
                case TxFSM is
                        when Idle =>
                                if StartTx='1' then
                                        RegDin <= Din;
                                        TxBusy <= '1';
                                        TxFSM <= Load_Tx;
                                else
                                        TxBusy <= '0';
                                end if;
                        when Load_Tx =>
```

```
                        if TopTx='1' then
                                TxBitCnt <= (NDBits + 1);
                                Tx_reg <= RegDin & '0';
                                TxFSM <= Shift_Tx;
                        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;
```

3. Receive

El fichero “receibe” tiene la utilidad de gestionar los datos recibidos de igual forma que el fichero “transmit”, mediante una máquina de estados finitos.

Las librerías que se incluyen ene este fichero son las siguientes:

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

En el entity tenemos las siguientes entradas y salidas:

```
entity receive is
  generic (NDBits : natural := 8);
  port (
        CLK     : in std_logic;
        RST     : in std_logic;
        Rx      : in std_logic;
        Dout    : out std_logic_vector (NDBits-1 downto 0);
        RxErr   : out std_logic;
        RxRdy   : out std_logic;
        Top16   : in std_logic;
        ClrDiv  : out std_logic;
        TopRx   : in std_logic);
end;
```

En la architecture tenemos las siguientes señales:

```
signal Rx_Reg : std_logic_vector (NDBits-1 downto 0);
type State_Type is (Idle, Start_Rx, Edge_Rx, Shift_Rx, Stop_Rx, Rx_Ovf);
signal RxFsm : State_Type;
signal RxBitCnt : integer;
signal Sig_RxRdy : std_logic;
```

A continuación tenemos la comprobación de la señal "RST", si se cumple la condición de que está a nivel bajo a la señal vector "Rx_Reg" se le asigna el valor de cero, a la salida vector "Dout" se le asigna el valor de coro, a la señal "RxBitCnt" se le da el valor de cero, a la señal "Rx_FSM" se le asigna el estado "Idle", a la salida "Sig_RxRdy" se le asigna nivel bajo, a la salida "ClrDiv" se le asigna nivel bajo y a la salida "RxErr" se le asigna nivel bajo. Si la entrada "RST" no esta a nivel bajo y la señal de reloj está en flanco de bajada, se le asigana nivel bajo a la salida "ClrDiv" y se evalua mediante un condicional si la señal "Sig_RxRdy" está en uno, si se cumple esto último a la salida "RxErr" se le da nivel bajo y a la señal "Sig_RxRdy" se le asigna nivel bajo también. A continuación, en el case, se comprueba la señal "RxFSM" que puede tener los siguientes estados: Idle, Start_Rx, Edge_Rx, Shift_Rx, Stop_Rx, Rx_Ovf.

Estado Idle:

En este estado primeramente se le asigna el valor de cero a la señal "RxBitCnt" y a continuación se compueba el valor de la entrada "Top16" mediante un condicional, si tiene un nivel alto mediante otro condicional se comprueba el nivel de la entrada "Rx" y si esta última está a nivel bajo se le asigna el estado "Start_RX" a la señal "RxFSM" y un nivel alto a la salida "ClrDiv".

Estado Start_Rx:

En el estado "Start_Rx" mediante un condicional se comprueba el nivel de la entrada "TopRx", y si está a nivel alto se comprueba mediante otro condicional el estado de la entrada "Rx", si esta última también está a nivel alto se le asigna el estado "Rx_OVF" a la señal "Rx_FSM" y se comunica que ha habido un error de lectura en el bit de "start". Sí "Rx" no está a nivel alto se le asigna el estado "Edge_Rx" a la señal "RxFSM".

Estado Edge_Rx:

En este estado mediante un condicional se comprueba inicialmente el valor de la entrada "TopRx", si está a nivel alto se le asigna el estado "Shift_Rx" a la señal "RxFSM" y mediante otro condicional se comprueba el valor de la señal "RxBitCnt", que si es igual a la variable "NDBits" se le asigna el estado "Stop_Rx" a la señal "RxFSM" y si fuesen diferentes se le asignría el estado "Shift_Rx".

Estado Shift_Rx:

Inicialmente en este estado se comprueba la entrada "TopRx", si esta a nivel alto se le asigana a la señal "RxBitCnt" su valor incrementado en uno, a continuación a la señal vector "Rx_Reg" se le asignan el valor de "Rx" concatenado con sus propios valores desde la posición máxima hasta la posición uno, por último en este estado se le asigna el estado "Edge_Rx" a la señal "RxFSM".

Estado Stop_Rx:

En este estado se comprueba nuevamente el valor de la entrada "TopRx" y si esta a nivel alto se le asigna al vector de salida "Dout" los valores contenidos en la señal vector "Rx_reg", a continuación se le asígna el valor de uno a la señal "Sig_RxRdy" y el estado "Idle a la señal "RxFSM", por último en este estado se muestra por consola el valor recibido en decimal.

Estado Rx_OVF:

En este estado primeramente se le asigna un nivel alto a la salida "RxErr" y a continuación mediante un condicional se comprueba el estado de la entrada "Rx", si está a nivel alto se le asigna el estado "Idle" a la señal "RxFSM".

```
Rx_FSM: process (RST, CLK)
begin
        if RST='0' then
                Rx_Reg <= (others => '0');
```

```
        Dout <= (others => '0');
        RxBitCnt <= 0;
        RxFSM <= Idle;
        Sig_RxRdy <= '0';
        ClrDiv <= '0';
        RxErr <= '0';
elsif rising_edge(CLK) then
        ClrDiv <= '0';
                if Sig_RxRdy='1' then
                        RxErr <= '0';
                        Sig_RxRdy <= '0';
                end if;
        case RxFSM is
                when Idle =>
                        RxBitCnt <= 0;
                                if Top16='1' then
                                        if Rx='0' then
                                                RxFSM <= Start_Rx;
                                                ClrDiv <='1';
                                        end if;
                                end if;

                when Start_Rx =>
                        if TopRx = '1' then
                                if Rx='1' then
                                        RxFSM <= Rx_OVF;
                                        report "Start bit error." severity note;
                                else
                                        RxFSM <= Edge_Rx;
                                end if;
                        end if;
                when Edge_Rx =>
                        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 =>
                        if TopRx = '1' then
                                RxBitCnt <= RxBitCnt + 1;
                                Rx_Reg <= Rx & Rx_Reg (Rx_Reg'high downto 1);
                                RxFSM <= Edge_Rx;
                        end if;
                when Stop_Rx =>
                        if TopRx = '1' then
                                Dout <= Rx_reg;
                                Sig_RxRdy <='1';
                                RxFSM <= Idle;
```

```
                                report "Character received in decimal is : "
                                & integer'image(to_integer(unsigned(Rx_Reg)))
                                severity note;
                        end if;
                when Rx_OVF =>
                        RxErr <= '1';
                        if Rx='1' then
                                RxFSM <= Idle;
                        end if;
        end case;
    end if;
  end process;
```

4. UART

El fichero “uart” es el fichero principal donde se crean las instanciaciones de los componentes.

En primer lugar, en este fichero tenemos las siguientes librerías:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;
use work.uart_comps.all;
```

A continuación en el entity tenemos la asignación de los valores para las variables genéricas y las entradas y salidas:

```
entity uart is
generic (F : natural := 50000;
        min_baud : natural := 1200;
        num_data_bits : natural := 8 );

        port    (Rx             : in std_logic;
                Tx              : out std_logic;
                Din             : in std_logic_vector(num_data_bits-1 downto 0);
                StartTx         : in std_logic;
                TxBusy          : out std_logic;
                Dout            : out std_logic_vector(num_data_bits-1 downto 0);
                RxRdy           : out std_logic;
                RxErr           : out std_logic;
                baud            : in std_logic_vector(2 downto 0);
                clk             : in std_logic;
                rst             : in std_logic);
end;
```

En la architecture tenemos las siguientes señales:

```
        signal top16            : std_logic;
        signal toprx            : std_logic;
        signal toptx            : std_logic;
        signal Sig_ClrDiv       : std_logic;
```

Y las instanciaciones de los componentes siguientes:

Receibe:

```
reception_unit: receive
generic map (NDBits    => num_data_bits)
port map (     CLK     => clk,
               RST     => rst,
               Rx      => Rx,
               Dout    => Dout,
               RxErr   => RxErr,
               RxRdy   => RxRdy,
               ClrDiv  => Sig_ClrDiv,
               Top16   => top16,
               TopRx   => toprx);
```

Transmit:

```
transmission_unit: transmit
generic map (NDBits    => num_data_bits)
port map (   CLK       => clk,
             RST       => rst,
             Tx        => Tx,
             Din       => Din,
             TxBusy    => TxBusy,
             TopTx     => toptx,
             StartTx   => StartTx);
```

Timing:

```
timings_unit: timing
generic map    (F => F,min_baud => min_baud)
port map (     CLK     => clk,
               RST     => rst,
               baud    => baud,
               ClrDiv  => Sig_ClrDiv,
               Top16   => top16,
               TopTx   => toptx,
               TopRx   => toprx);
```

5. Uart_comps

Este fichero es la biblioteca donde se declaran las entradas y salidas de los distintos componentes que incorpora el sistema.
Inicialmente incorpora las siguientes librerías:

```
library ieee;
use ieee.std_logic_1164.all;
```

Los componentes que contiene está biblioteca son:

Timing:

```
component timing is
    generic (F : natural := 50000;
             min_baud : natural :=  1200);
    port (
```

```
                CLK :    in std_logic;
                RST :    in std_logic;
                baud :  in std_logic_vector(2 downto 0);
                ClrDiv : in std_logic;
                Top16 :          buffer std_logic;
                TopTx :          out std_logic;
               TopRx :           out std_logic);
end component;
```

Este componente es el usado para seleccionar las velocidades de transmisión y para configurar las señales de los distintos relojes.

Transmit:

```
component transmit is
        generic (NDBits : natural := 8);
        port (  CLK             : in std_logic;
                RST             : in std_logic;
                TopTx           : in std_logic;
                StartTx         : in std_logic;
                Din             : in std_logic_vector (NDBits-1 downto 0);
                Tx              : out std_logic;
                TxBusy          : out std_logic );
end component;
```

El componente “transmit” es usado para manejar la transmisión de datos.

Receibe:

```
component receive is
  generic (NDBits : natural := 8);
  port (CLK             : in std_logic;
        RST             : in std_logic;
        Rx              : in std_logic;
        Dout            : out std_logic_vector (NDBits-1 downto 0);
        RxErr           : out std_logic;
        RxRdy           : out std_logic;
        Top16           : in std_logic;
        ClrDiv          : out std_logic;
        TopRx           : in std_logic);
end component;
```

Este ultimo componente es usado para gestionar la recepción de datos.