** **

**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](#_Toc72608433)

[1. Timing 1](#_Toc72608434)

[2. Transmit 3](#_Toc72608435)

[3. Receive 5](#_Toc72608436)

[4. UART 8](#_Toc72608437)

[5. Uart\_comps 9](#_Toc72608438)

# 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.

## 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;**

## 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;**

## 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;**

## 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);**

## 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.