به نام خدا

پروژه پایانی درس VHDL

استاد اسحاقی

اعضای گروه:

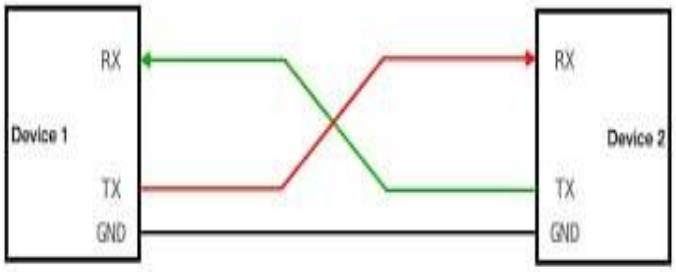
حسین کرمی

پوریا عالیشاه کمندی

مقدمه:

UART که مخفف universal asynchronous receiver-transmitter است، یک نوع سیستم ارتباطی بین است که با استفاده از یک رشته سیم اطلاعات را به صورت متوالی انتقال میدهد.

در اینجا یک نمونه انتقال uart را



مشاهده میکنید، که بین برد FPGA و سیستم کامپیوتری هدف است که با یک کانال ارتباطی بین یکدیگر دیتارا رد و بدل میکنند.

در طراحی ارتباط UART باید به چند مشخصه توجه کرد:

1. سرعت انتقال یا تعداد بیت بر ثانیه(BUAD Rate)
2. بیت مقایسه گر یا بیت (Parity Bit)
3. بیت شروع و بیت استاپ (START And STOP Bit)
4. اندازه دیتا برای انتقال

ما سیستم انتقال دهنده (Transmitter System) را برای برد خود طراحی میکنیم.

سرعت Baud Rate این سیستم برابر با 9600 بیت بر ثانیه است و بیت مقایسه گر نیز ندارد.

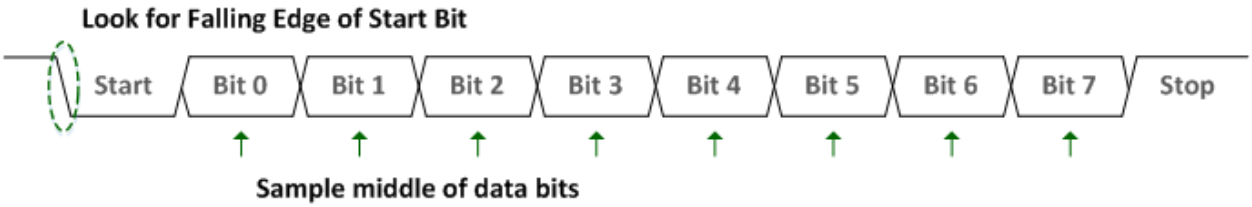
```
entity UART_TX is
  generic (
    gen_clks_per_bit : integer := 7500 - 1 -- for STM32F10ZET6
    );
  port (
    in_clk          : in  std_logic;
    in_data_valid   : in  std_logic; -- Data valid pulse
    in_serial_byte  : in  std_logic_vector(7 downto 0); -- Our Data
    out_active      : out std_logic; -- For simulation
    out_serial      : out std_logic; -- Trasfering data byte by byte
    out_finish      : out std_logic -- End of transfering
    );
end UART_TX;
```

در اینجا ما entity خود را با نام، UART_TX تعریف میکنیم که 3 ورودی:

1. in_Clk : ورودی کلاک
2. in_data_valid : تنظیم کننده فعالیت UART
3. in_serial_byte : دیتا ورودی

همچنین 3 خروجی:

1. out_active : فعال بودن سیستم UART
2. out_serial : بیت دیتا(چرا که دیتا به صورت بیت به بیت وارد میشود.)
3. out_finish : بیت پایان دیتا است



همانطور که مشاهده میکنید، شروع کار UART به این شکل است که، ابتدا بیتی به عنوان بیت ورودی داده میشود که بیت 0 است که یعنی شروع انتقال دیتا اصلی ماست، بعد از انتقال 8 بیت دیتا ما، و بعد بیت استاپ داده میشود که مقدار 1 دارد و به معنی پایان انتقال است.

در این قسمت از کد ما یک Generic تعریف کردیم که مقدار کلاک در هر بیت انتقال را به ما نشان میدهد که مقدار:

$$CLK\ PER\ BIT = \frac{CLK\ OF\ BOARD}{BUAD\ OF\ UART\ SYSTEM}$$

که با توجه به اینکه ما کلاک FPGA را 72 مگا هرتز در نظر میگیریم، این عدد برابر با:

$$CLK\ PER\ BIT = \frac{72000000}{9600} = 7500$$

که این یعنی در هر بیت انتقال دیتا، 7500 کلاک صورت میگیرد.

- نکته: البته این اطلاعات را با توجه به تجربیات در درس آزمایشگاه میکرو در نظر گرفتیم چرا که سرعت کلاک میکرو STM32F10ZET6 برابر با 72 مگاهرتز بوده و برای اینکه در سیستم دریافت کننده، نمونه برداری راحت تر باشد، باید مقدار کلاک بر بیت کمتر از 14000 باشد. چراکه به دلیل بالا بودن رزولوشن سیستم UART، نمونه برداری دشوارتر میشود. و اینکه مقدار Generic را برابر با 7499 قرار دادیم، چراکه شمارش ما در کد از صفر است.

```
architecture Behavioral of UART_TX is

    type type_SM is (s_idle, s_start_bit, s_data_bits, s_stop_bit, s_cleanup);

    signal s_current    : type_SM := s_idle;
    signal clk_counter  : integer range 0 to gen_clks_per_bit := 0;
    signal bit_index    : integer range 0 to 7 := 0;  -- 8 Bits Total
    signal data         : std_logic_vector(7 downto 0) := (others => '0');
    signal finish       : std_logic := '0';

 begin
```

حال نوبت به architecture میرسد.

ابتدا تایپی از حالت های ماشین خود ایجاد میکنیم که داری 5 حالت که به ترتیب نشان دهنده:

1. حالت اولیه یا حالت بیکار : s_idle
2. شروع انتقال دیتا : s_start_bit
3. درحال ارسال دیتا : s_data_bits
4. حالت توقف انتقال : s_stop_bit
5. حالت ریست : s_cleanup

حال سیگنالی از تایپ خود ایجاد کرده و حالت سیستم را برابر با حالت بیکار قرار میدهیم، و تعداد شمارنده کلاک که از 0 تا 7499 میباشد را ایجاد کرده و سپس تعداد بیت مورد انتقال، سیگنالی برای بیت دیتا ورودی مورد انتقال که 8 بیت میباشد و همچنین بیت انجام را تعریف میکنیم که finish نام دارد.

```
    process (in_clk)
    begin
        if rising_edge(in_clk) then

            case s_current is

            when s_idle =>
                out_active  <= '0';
                out_serial  <= '1';         -- Drive Line High for Idle
                finish      <= '0';
                clk_counter <= 0;
                bit_index   <= 0;

                if in_data_valid = '1' then
                    data <= in_serial_byte;
                    s_current <= s_start_bit;
                else
```

```
                    s_current <= s_idle;
                end if;
```

حال در این قسمت، process را تعریف میکنیم تا بتوانیم به صورت ترتیبی کدهای خود را بنویسیم.

و این process حساس به کلاک (in_clk) بوده و زمانی که کلاک به صورت صعودی بود وارد if شده و مقدار حالت کنونی سیستم را مورد بررسی قرار میدهیم.

حال اگر حالت اولیه سیستم برابر با s_Idel بود تنظیمات بالا را برای سیگنال و خروجی انجام میدهیم.

به ترتیب یعنی:

1. خروجی است که به منظور درک بهتر از شبیه ساز طراحی شده است، که زمانی که UART فعال بود، 1 و در غیر اینصورت 0 باشد.
2. مقدار اولیه بیت انتقال دهنده، برابر 1 قرار میدهیم، که زمانی که برابر 0 شد، نشان دهنده بیت شروع ماست.(باتوجه با تصویر صفحه 3)
3. بیت پایان روند انتقال هست که در ابتدا برابر صفر قرار میگیرد.
4. شمارنده تعداد کلاک در هر بیت است.
5. شماره خانه بیت مورد انتقال است که در حالت بیکار برابر با 0 قرار میگیرد.

حال در شرط اولی زمانی که ورودی in_data_valid برابر با 1 بود که به معنا فعال کردن UART است، مقدار دیتا ورودی را در متغییر data قرار داده و سپس حالت سیستم را به s_start_bit تغییر میدهیم که یعنی حالت بیت شروع.

```
if i_TX_DV = '1' then
            r_TX_Data <= i_TX_Byte;
            r_SM_Main <= s_TX_Start_Bit;
          else
            r_SM_Main <= s_Idle;
        end if;
          o_TX_Active <= '1';
          o_TX_Serial <= '0';
```

حال اگر مقدار i_TX_DV برابر 1 بود، مقدار بیت مورد انتقال را به سیگنال r_TX_Data داده و حالت کنونی سیستم را به شروع انتقال تغییر میدهیم. و در غیر این صورت حالت را به حالت اولیه بازمیگردانیم.

سپس بیت خروجی o_TX_Active را برابر 1 قرار میدهیم تا نشان دهد که UART فعال بوده و سپس با توجه به تصویر بالا، مقدار 0 را به o_TX_Serial نسبت میدهیم که یعنی شروع انتقال. و در غیر اینصورت حالت سیستم را به حالت اولیه یا همان حالت بیکار قرار میگیرد.

```
            when s_start_bit =>
                out_active <= '1'; -- set high for entire transmittion proccess
```

```
                out_serial <= '0';

                -- Wait gen_clks_per_bit clock cycles for start bit to finish
                if clk_counter <= gen_clks_per_bit then
                    clk_counter <= clk_counter + 1;
                    s_current   <= s_start_bit;
                else
                    clk_counter <= 0;
                    s_current   <= s_data_bits;
                end if;
```

در این قسمت که قسمت بیت شروع است، مقدار خروجی out_active را برابر با 1 قرار میدهیم که یعنی سیستم UART ما فعال بوده و اماده به کار است، و مقدار 0 را به خروجی out_serial میدهیم.(با توجه به شکل صفحه سوم)

در شرط اولی تعداد کلاک هایی که زده میشود را چک میکنیم، یعنی ما باید به اندازه 7500 کلاک صبر کنیم تا از انتقال دیتا و دریافت ان توسط گیرنده اطمینان حاصل کنیم. و زمانی که مقدار شمارنده کلاک ما به مقدار 7500 رسید، مقدار شمارنده برابر با صفر قرار میگیرد و حالت انتقال دیتا به حالت سیستم نسبت میگیرد.

```
            when s_data_bits =>
                out_serial <= data(bit_index);

                if clk_counter <= gen_clks_per_bit then
                    clk_counter <= clk_counter + 1;
                    s_current   <= s_data_bits;
                else
                    clk_counter <= 0;

                    -- Check if we have sent out all bits
                    if bit_index <= 7 then
                        if bit_index /= 7 then
                            bit_index <= bit_index + 1;
                        end if;
                        s_current   <= s_data_bits;
                    else
                        bit_index <= 0;
                        s_current   <= s_stop_bit;
                    end if;
                end if;
```

در این قسمت هنگامی که به حالت انتقال دیتا رسیدیم، ابتدا دیتا را با توجه به مقدار bit_index فرستاده، و سپس برای هر بیت به اندازه 7500 کلاک صبر میکنیم و دوباره مقدار شمارنده کلاک

را برابر با صفر قرار داده و در قدم بعدی شماره خانه را یک مقدار اضافه میکنیم و تا این شرط برقرار است حالت سیستم هم برابر حالت در حال انتقال میباشد. زمانی که مقدار برابر با 7 شد، وارد بدنه else شده و شماره خانه را 0 کرده و حالت را به حالت توقف تغییر میدهد.

```
            when s_stop_bit =>
                out_serial <= '1';

                -- Wait gen_clks_per_bit-1 clock cycles for Stop bit to finish
                if clk_counter <= gen_clks_per_bit then
                    clk_counter <= clk_counter + 1;
                    s_current   <= s_stop_bit;
                else
                    finish   <= '1';
                    clk_counter <= 0;
                    s_current   <= s_cleanup;
                end if;
```

در این قسمت که قسمت توقف هست، مقدار out_serial ما با توجه با شکل صفحه سوم، برابر 1 باید قرار گیرد و در شرط نیز مانند انتقال سایر بیت ها باید به اندازه 7500 کلاک منتظر بمانیم. و بعد از انجام این عملیات یعنی در صورتی که شمارش بیت ما به پایان رسید، finish را برابر با 1 قرار میدهیم و مقدار شمارنده را نیز صفر میکنیم و حالت سیستم را نیز به کلین اپ تغییر میدهیم.

```
            when s_cleanup =>
                out_active <= '0';
                finish   <= '1';
                s_current   <= s_idle;

            when others =>
                s_current <= s_idle;

            end case;
        end if;
    end process;

    out_finish <= finish;
```

در حالت ریست، مقدار خروجی out_active برابر با صفر قرار میگیرد که UART غیر فعال شده و سیگنال finish هم برابر با 1 قرار میگیرد که به معنی پایان انتقال است. و حالت سیستم را نیز به به حالت بیکار یا همان حالت اولیه تغییر میدهیم.

و در اخر هم اگر هیچ یک از حالات برقرار نبودند، با بررسی شرط when others اطمینان حاصل میکنیم که سیستم به حالت اولیه بازگردد.

و در پایان نیز مقدار out_finish برابر با finish قرار داده که وضعیت پایان یا عدم اتمام انتقال را به ما نشان میدهد.