به نام خدا

"طراحی کامپیوتری سیستم های دیجیتال"

پروژه پایانی

"طراحی واسط سریال (UART Interface)"

استاد درس:

مهندس محسن احمدوند

دستیار استاد:

سامان عبدالله پور

نام و نام خانوادگی دانشجو:

امیرحسین بابائیان

شماره دانشجویی:

۹۸۰۶۴۵۳

۰

فهرست

تعریف پروژه

هدف از انجام این پروژه آشنایی کامل با چگونگی طراحی و پیاده سازی یک سیستم دیجیتال با استفاده از زبان توصیف سخت افزار VHDL می باشد. در این پروژه قصد داریم تا یک واسط سریال یا همان UART Interface را پیاده سازی نماییم.

محیط مورد استفاده

در این پروژه ما برای کد نویسی از محیط ویرایش متن Notepad++ و همچنین محیط توسعه ModelSim استفاده شده است.

پیاده سازی UART با استفاده از VHDL

در بخش های آتی ما قطعه های مختلف پیاده سازی شده پروژه با استفاده از زبان توصیف سخت افزار VHDL را آورده ایم و به فراخور هر بخش توضیحات متناسب با آن را نیز در آن بخش گنجانده ایم.

معرفی کتابخانه های مورد استفاده

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;
```

در این شبیه ساز ما از چهار کتابخانه برای عملیات های مختلف از جمله تبدیل نوع، محاسبات و ... استفاده نموده ایم.

معرفی بدنه اصلی

بدنه اصلی شبیه ساز ما از دو بخش Entity و Architecture تشکیل شده است که در ادامه به توضیحات در مورد هر یک می پردازیم.

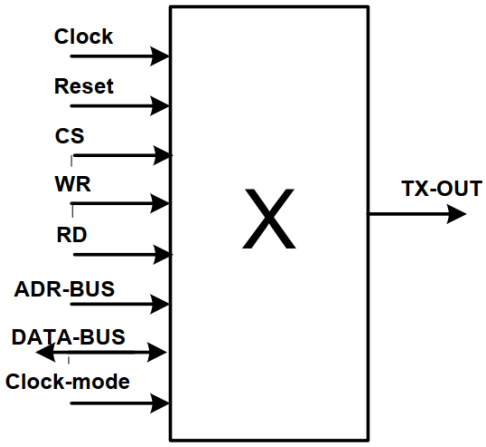
معرفی Entity

```
ENTITY usart IS

        GENERIC (
            Queue_is_Empty_or_full                  : INTEGER RANGE 0 TO 9 := 6;
            --  0 : Empty , 1 : Full
            write_in_Queue_In_the_form_of_serial    : INTEGER RANGE 0 TO 9 := 5

        );

        PORT(
            clk             : IN STD_LOGIC;
            reset           : IN STD_LOGIC;
            cs              : IN STD_LOGIC;
            wr              : IN STD_LOGIC;
            rd              : IN STD_LOGIC;
            address_bus     : IN INTEGER RANGE 0 TO 16;
            data_bus        : IN STD_LOGIC_VECTOR(7 DOWNTO 0 );
            data_bus_test   : BUFFER  STD_LOGIC_VECTOR(7 DOWNTO 0 );
            transmit        : OUT  STD_LOGIC_VECTOR(9 DOWNTO 0 );
            transmit_out    : BUFFER STD_LOGIC

        );

END usart;
```

همانطور که مشاهده می کنید ما یک Entity با نام usart تعریف نموده ایم که شامل دو بخش Generic و Port می باشد.

در بخش Generic ما دو متغییر تعریف نموده ایم که Queue_is_Empty_or_full وظیفه دارد وضعیت پر یا خالی بودن صف را به عهده دارد، در واقع این متغییر آدرس رجیستر کنترل را در خود نگهداری می کند، همانطور که به صورت کامت توضیح نوشته شده است در صورتی که مقدار آن ۰ باشد خالی و اگر ۱ باشد پر است.
write_in_Queue_In_the_form_of_serial نیز در زمان نوشتن به صورت سریال مورد استفاده قرار می گیرد که در آن بخش به صورت جزئی تری توضیحات مربوط به آن اشاره خواهد شد.

در بخش Port نیز می توانید ورودی ها وخروجی های مختلف این شبیه ساز را مشاهده نمایید که از انواع مختلف داده ای همچون IN و OUT و BUFFER مورد استفاده قرار گرفته است. لازم به ذکر است بخش هایی با تغییرات ریزی نسبت به فایل اصلی پروژه تعریف گردیده است.



تصویر فوق پورت های مختلف این شبیه سازی را در فایل اصلی نشان می دهد.

معرفی Architecture

```
ARCHITECTURE project OF usart IS

BEGIN

END project;
```

آنچه مشاهده می کنید یک تعریف کلی از یک معماری است که نام آن را project گذاشته ایم و خود نیز شامل دو بخش تعریف و بدنه معماری می باشد.

پیاده سازی بخش Deceleration

در بخش تعریف ها ما ابتدا به تعریف Typeهای مورد نیاز پرداخته ایم:

```
-- Type
    TYPE RAM IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    TYPE MYRAM IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(9 DOWNTO 0);
    TYPE MICRO_INTERFACE IS ARRAY (0 TO 15) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
    TYPE CONTROL_REGISTER IS ARRAY (0 TO 7) OF STD_LOGIC;
```

ما چهار Type را طبق کد های بالا تعریف نموده ایم، به جهت سادگی در خوانش از حروف بزرگ الفبای انگلیسی استفاده نموده ایم تا میان انواع عبارات مورد استفاده تمایز قائل شده باشیم.

- RAM که یک نوع آرایه ۱۶ تایی از نوع STD_LOGIC_VECTOR با اندازه ۸ بیت می باشد که برای حافظه نگهدارنده دیتای ورودی استفاده می گردد.
- MYRAM که یک نوع آرایه ۱۶ تایی از نوع STD_LOGIC_VECTOR با اندازه ۱۰ بیت می باشد.
- MICRO_INTERFACE که یک نوع آرایه ۱۶ تایی از نوع STD_LOGIC_VECTOR با اندازه ۸ بیت می باشد.
- CONTROL_REGISTER که یک نوع آرایه ۸ تایی از نوع STD_LOGIC با اندازه یک بیت می باشد.

سپس و در ادامه ما Signal های مورد نیاز را تعریف نموده ایم:

```
-- Signal
    SIGNAL mic_inter        : micro_interface;
    SIGNAL tran             : STD_LOGIC_VECTOR(9 DOWNTO 0);
    SIGNAL connect          : STD_LOGIC_VECTOR(7 DOWNTO 0);
```

هر یک از سیگنال های تعریف شده برای کاربردهایی تعریف گردیده اند که به عنوان مثال می توان اشاره کرد که connect و tran برای ارسال و تست داده ها بر روی خروجی نهایی و صف استفاده می شوند.

در ادامه وارد بدنه مربوط به معماری شده و توضیحات مربوط به بخش های مختلف آن ارائه خواهد شد.

پیاده سازی بخش Body

در بخش بدنه ما دو Process تعریف نموده ایم که کلیات مربوط به بخش های مختلف را شبیه سازی می نماید.

پیش از آن که به طور کامل به توضیحات Processها بپردازیم لازم می دانیم تا با ساختار کلی آن آشنا شویم، به جهت کوتاه و کاربردی شدن گزارش صرفا Process اول بسیار کامل توضیح داده خواهد شد و مورد دوم به اختصار بدان پرداخته شده و بخش های مشترک توضیح داده نخواهد شد.

```
    micro_int : PROCESS(clk, reset) IS

    BEGIN

    END PROCESS;
```

همانطور که در فوق می بینید شکل کلی Process به صورت فوق می باشد، البته micro_int یک لیبل به صورت اختیاری می باشد که می توان از نوشتن آن صرف نظر کرد، همانند معماری، Processها نیز به دو بخش

تعریف و بدنه تفکیک می شوند، همچنین یکی از نکات قابل توجه آن است که این Process با تغییر clk تریگر می شود.

```
    -- Variable
        VARIABLE flag_of_number_data_in_micro    : INTEGER RANGE 0 TO 16 := 0;
```

پیاده سازی پروسس Micro_Interface

آنچه مشاهده می کنید بخش تعریف Process اول می باشد که یک flag به عنوان Variable تعریف شده است که وظیفه پایش داده های وارد شده به micro را دارد.

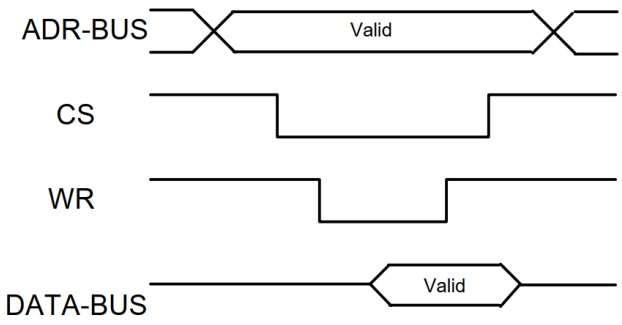
```
IF(clk'EVENT AND clk = '1') THEN
    IF(cs = '0' AND rd = '0' AND reset = '0') THEN
        IF(flag_of_number_data_in_micro /= 16) THEN
            mic_inter(address_bus) <= data_bus;
            flag_of_number_data_in_micro := flag_of_number_data_in_micro + 1;
        ELSIF(flag_of_number_data_in_micro = 16) THEN
            -- NOTHING
        END IF;

    ELSIF(cs = '0' AND wr = '0') THEN
        data_bus_test <= mic_inter(address_bus);
        flag_of_number_data_in_micro := flag_of_number_data_in_micro - 1;
    END IF;

ELSIF(reset = '1') THEN
    flag_of_number_data_in_micro := 0;
END IF;
```

بدنه Process به صورت کلی در بخش فوقانی آورده شده است که قصد داریم به صورت مشروح بدان بپردازیم.

بدنه پروسس شامل یک دستور IF-ELSIF می باشد که شرط اولیه آن برای تشخیص clk بالارونده و در جهت سنکرون سازی کلاک به کار برده می شود، همچنین ELSIF نیز برای آن بکار می رود که در صورت صحیح بودن شرط آن یعنی فعال شدن reset مقادیر معیار محتوای داخل micro صفر گردد.

اگر در بدنه پروسس فوق کلاک بالارونده بیاید دو عمل مختلف را می تواند به فراخور دیگر متغییر های انجام دهد که عبارتند از خواندن و نوشتن که به تریتب به صورت توضیحات ذیل استفاده می شوند.

پیاده سازی بخش Read

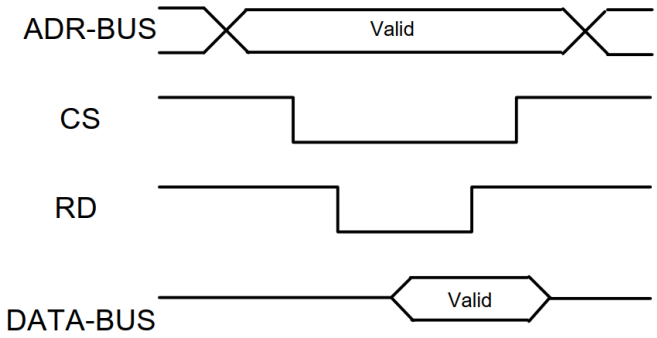


نحوه نوشتن داده ها توسط بخش Micro-Interface

دقیقا آنچه در بخش فوق و در صورت پروژه خواسته شده است در فرمت کد نیز پیاده سازی گردیده است. بدین شرح که اگر مقدار wr و cs برابر با صفر شوند. (فعال شوند.) و همچنین micro ما ظرفیت دریافت مقدار بیشتر را داشته باشد و پر نشده باشد انگاه عملیات نوشتن صورت می پذیرد. (قطعه کد عملیات در ادامه آورده خواهد شد.)

```
        mic_inter(address_bus) <= data_bus;
        flag_of_number_data_in_micro := flag_of_number_data_in_micro + 1;
```

پیاده سازی بخش Write



نحوه خواندن داده ها توسط بخش Micro-Interface

همچنین برای خواندن نیز در صورتی که rd و cs هر دو فعال شوند بر اساس آنچه در فوق می بینیم می توانیم در صورتی که محتوای جهت خواندن وجود داشته باشد. (همان محتوای valid که در دو تصویر فوق مشاده می نمایید.) محتوا خوانده خواهد شد. (قطعه کد عملیات در ادامه آورده خواهد شد.)

```
    data_bus_test <= mic_inter(address_bus);
    flag_of_number_data_in_micro := flag_of_number_data_in_micro - 1;
```

لازم است توجه داشته باشید که این مدار واسط شامل ۱۶ خانه آرایه ای برای نگهداری اطلاعات وارده می باشد و د لیل قرار گیری مقدار ۱۶ در شرایط همین اندازه حافظه اختصاصی می باشد.

همانطور که در بخش های بالاتر نیز اشاره شد در صورت فعال شدن restet کد زیر اجرا خواهد شد،

```
            flag_of_number_data_in_micro := 0;
```

پیاده سازی پروسس Controler

خط فوق مربوط به آخرین خط از کد پروسس اول می باشد و در اادامه به Process دوم می پردازیم.

```
    -- Variable
        VARIABLE con_reg                        : CONTROL_REGISTER := "00000000";
        VARIABLE FIFO                           : RAM;
        VARIABLE trans_out                      : STD_LOGIC_VECTOR(9 DOWNTO 0);
        VARIABLE flag_of_number_data_in_fifo    : INTEGER RANGE 0 TO 16 := 0;
        VARIABLE temp                           : STD_LOGIC;
        VARIABLE flag                           : STD_LOGIC := '0';
        VARIABLE memory                         : MYRAM;
        VARIABLE count                          : INTEGER RANGE 0 TO 10 := 0;
        VARIABLE counter_out                    : INTEGER RANGE 0 TO 9;
```

این بار و بدون توضیح اضافه تری مستقیم به توضیح بخش Deceleration یا همان تعاریف می پردازیم، بر خلاف مورد قبلی در این پروسس تعداد زیادی Variable تعریف شده است که هر کدام در با انواع مختلفی از جمله Type های تعریفی در ابتدای کار مورد استفاده قرار گرفته اند، در ادامه و هنگام استفاده از هریک در صورت لزوم توضیحاتی مربوط به هریک داده خواهد شد.

با توجه به اینکه باقی پیاده ساری مربوط به فرایند ها در این پروسس با نام controller اورده شد است از این رو بدنه این بخش طولانی تر از موارد گفته شده در قبل است و ما برای سادگی محتوای ارائه شده در ابن گزارش در بخش های مختلف به توضیح بدنه می پردازیم.

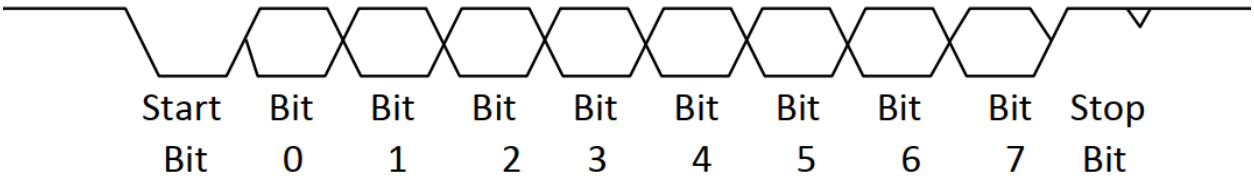
```
IF(clk'EVENT AND clk = '1') THEN

    IF(cs = '1') THEN
        IF(con_reg(Queue_is_Empty_or_full) = '0') THEN
            FIFO(flag_of_number_data_in_fifo) := connect;
            flag_of_number_data_in_fifo := flag_of_number_data_in_fifo + 1;
            transmit(7 DOWNTO 0) <= mic_inter(flag_of_number_data_in_fifo);
        END IF;
```

```
        IF(flag_of_number_data_in_fifo = 16) THEN
            con_reg(Queue_is_Empty_or_full) := '1';
        ELSIF(flag_of_number_data_in_fifo /= 16) THEN
            con_reg(Queue_is_Empty_or_full) := '0';
        END IF;
    END IF; -- End Of "IF(cs = '1')"

    IF(con_reg(Queue_is_Empty_or_full) = '0') THEN
        FOR i IN 0 TO 15 LOOP
            IF(con_reg(write_in_Queue_In_the_form_of_serial) = '0') THEN
                trans_out(0) := '0';
                trans_out(9) := '1';
                trans_out(8 DOWNTO 1) := FIFO(0)(7 DOWNTO 0 );
                memory(i)(9 DOWNTO 0):= trans_out(9 DOWNTO 0);
                flag := '1';
                transmit <= trans_out;
            END IF;
        END LOOP;
    END IF; -- End Of "IF(con_reg(Queue_is_Empty_or_full) = '0')"
    flag_of_number_data_in_fifo := 0;
END IF; -- End Of "IF(clk'EVENT AND clk = '1')"
```

بخش اول مربوط به بدنه شامل یک دستور شرط IF با شرط های تودرتو می باشد که وظیفه اش تشخیص کلاک بالارونده است سپس در صورتی که مقدار cs برابر با یک باشد و مقدار رجیستر کنترلی در بیت کنترل وضعیت صف برابر صفر باشد آنگا عملیات نوشتن در FIFO و خواندن از آن انجام می شود و سیگنال connect اطلاعات رم را به FIFO منتقل می کند.با اضافه شدن هر مقداری در FIFO مقدار flag مربوط به آن نیز یک واحد اضافه می گردد تا بتوان عملیات های کنترلی مختلفی را بر روی شبیه ساز اعمال نمود. فرایند این flag بدین صورت است که در صورتی که عدد آن به ۱۶ برسد مقدار رجیستر کنترلی آن برابر با ۱ خواهد شد و در غیراینصورت برابر ۰ می باشد.

پیاده سازی بخش ساخت Frame خروجی

سپس و در ادامه مجدد وضعیت صف بررسی گردیده و پس از آن نیز همانند تصویر ذیل بیت های آغازین و پایانی به دیتای ما ملحق می گردد.



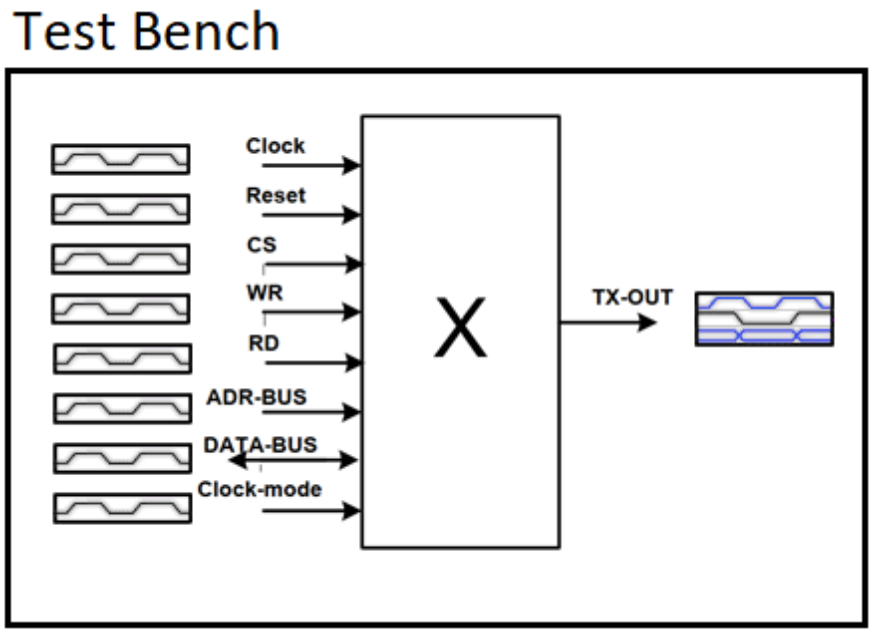
فعالیت بدین صورت است که بیت آغازی برابر با صفر و بیت پایانی نیز یک خواهد شد.

```
IF(flag = '1') THEN
        FOR p IN 0 TO 1 LOOP
            FOR j IN 0 TO 9 LOOP
                temp := memory(p)(j);
                tran <= trans_out;
                count := count + 1;
            END LOOP;
            count := 0;
        END LOOP;
        flag := '0';
        transmit_out<=tran(counter_out);
        counter_out := counter_out + 1;
        IF(counter_out = 9) THEN
            counter_out := 9;
            transmit_out<='X';
        END IF;

END IF; -- End Of "IF(flag := '1')"
```

حال با انجام این عملیات در اصل داده های ما اماده ارسال از طریف UART شده اند، ما در این بخش از دو حلقه برای پیمایش حافظه استفاده نموده ایم که برای اشاره بهتر از tran به عنوان یک سیگنال داخلی برای انتقال داده ها استفاده می شود و شمارنده ای نیز جهت ارسال خروجی به صورت بیت بیت با عنوان counter_out تعریف شده است که با رسیدن مقدار آن به ۹ طی احراز شرایط مشخص به مقدار ۰ تغییر می یابدو همچنین از مقدار X برای موارد نا متعبر در transmit_out که خروجی ما می باشد، استفاده گردیده است.

پیاده سازی TestBench

همانطور که می دانیم برای تست کردن شبیه ساز می توانیم از تست بنچ استفاده کنیم، در شکل زیر می توانید توصیف کلی این پروژه را مشاهده نمایید که تست بنچ چگونه عمل می کند.



توضیحات کلی پیاده سازی

بخش هایی از کد تست بنچ از کد اصلی آورده شده است از این رو ما نیازی به توضیح مجدد آن نداریم از این رو در ابتدای کار و در این بخش توضیحاتی را ارائه می دهیم. تست بنچ ما از کتابخانه هایی که قبلا معرفی نموده ایم استفاده می کند.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
```

یک ENTITY بدون ورودی خروجی دارد.

```
ENTITY test_usart IS

END test_usart;
```

پیاده سازی ARCHITECTURE

معماری تست بنچ خود شامل دو بخش می باشد که عبارتند از بخش تعریف و بدنه:

```
ARCHITECTURE test OF test_usart IS
      -- Deceleration
BEGIN
      -- Body
END test;
```

پیاده سازی بخش Deceleration

در بخش تعاریف ما ابتدا بایستی Component از usart بسازیم و تا در ادامه از آن در این تست استفاده نماییم.

```
 COMPONENT usart IS


       GENERIC (
           Queue_is_Empty_or_full                  : INTEGER RANGE 0 TO 9 := 6;
           write_in_Queue_In_the_form_of_serial    : INTEGER RANGE 0 TO 9 := 5

       );

       PORT(
           clk             : IN STD_LOGIC;
           reset           : IN STD_LOGIC;
           cs              : IN STD_LOGIC;
           wr              : IN STD_LOGIC;
           rd              : IN STD_LOGIC;
           address_bus     : IN INTEGER RANGE 0 TO 16;
           data_bus        : IN STD_LOGIC_VECTOR(7 DOWNTO 0 );
           data_bus_test   : BUFFER  STD_LOGIC_VECTOR(7 DOWNTO 0 );
           transmit        : OUT  STD_LOGIC_VECTOR(9 DOWNTO 0 );
           transmit_out    : BUFFER STD_LOGIC

       );

END COMPONENT;
```

آنچه مشاهده می کنید در بخش مربوط به توضیحات کد اصلی به صورت مشروح توضیح داده شده است.

```
    FOR ALL : usart  USE ENTITY work.usart (project);
```

سپس با استفاده از خط کد بالا ارتباط بین پروژه و تست را برقرار می سازیم.

در انتهای این بخش نیز ما به سراغ تعریف سیگنال هایی می رویم که در ادامه قصد داریم با مقدار دهی هریک از آن ها پروژه را تست نماییم، در ابتدای بخش مربوط به تست بنچ یک توصیف تصویری آورده شده است که در

اصل می توان گفت این سیگنال ها همان wave های ورودی و خروجی را شامل می شود که از طریق آن می توانیم تا حد مشخصی کارایی برنامه را ارزیابی نماییم که در بخش های بعدی مواردی را خواهید دید.

```
-- Signal
    -- * Inputs :
    SIGNAL clk              : STD_LOGIC := '0';
    SIGNAL reset            : STD_LOGIC := '0';
    SIGNAL cs               : STD_LOGIC := '1';
    SIGNAL wr               : STD_LOGIC := '1';
    SIGNAL rd               : STD_LOGIC := '1';
    SIGNAL address_bus      : INTEGER RANGE 0 TO 16 := 0;
    -- * Inputs/Output :
    SIGNAL data_bus         : STD_LOGIC_VECTOR(7 DOWNTO 0 ) := (OTHERS => '0');
    -- * Outputs :
    SIGNAL data_bus_test    : STD_LOGIC_VECTOR(7 DOWNTO 0 );
    SIGNAL transmit         : STD_LOGIC_VECTOR(9 DOWNTO 0 );
    SIGNAL transmit_out     : STD_LOGIC;
```

تنها نکته ای که در این بخش می توانیم به آن اشاره کنیم مقدار دهی اولیه از طریق Signal Assignment می باشد.

```
    -- * Clock Period :
    CONSTANT clk_period : TIME := 10 NS;
```

با استفاده خط کد بالا ما دوره زمانی کلاک را ۱۰ نانو ثانیه قرار می دهیم، همچین به جهتی که تغییری در این متغیر تعریفی ایجاد نشود آن را از نوع Constant تعریف نموده ایم و می دانیم که چنین متغیر هایی الزما باید مقداردهی اولیه شوند.

پیاده سازی بخش Body

حال به بخش استفاده از component تعریف شده در فوق رسیدیم در این بخش سیگنال های تعریف شده در بخش قبلی را به عنوان ورودی های component به آن می دهیم.

```
    UUT: usart PORT MAP
            (
               clk              => clk,
               reset            => reset,
               cs               => cs,
               wr               => wr,
               rd               => rd,
               address_bus      => address_bus,
```

```
            data_bus        => data_bus,
            data_bus_test   => data_bus_test,
            transmit        => transmit,
            transmit_out    => transmit_out

        );
```

پیاده سازی پروسس clk_process

در ادامه برای انجام فعالیت های مربوط به این پروژه نیاز ما به سازنده کلاک احساس می شود که با استفاده از این پروسس انجام می شود که مقدار زمانی هر یک از صفر یا یک برابر است با نصف زمان دوره تناوب کلاک تعریف شده.

```
        clk <= '0';
        WAIT FOR clk_period/2;
        clk <= '1';
        WAIT FOR clk_period/2;
    END PROCESS;
```

پیاده سازی بخش مقدار دهی

ما در این بخش از دو راه می توانیم مقادیر مورد نظرمان برای تست را به ورودی ها بدهیم، راه حل اول روش عادی می باشد که در فرمت ذیل قابل استفاده است:

```
cs              <= '0', '1' AFTER 500 NS;
wr              <= '0', '1' AFTER 300 NS;
rd              <= '1', '0' AFTER 300 NS;
address_bus     <= 9, 10 AFTER 100 NS, 11 AFTER 200 NS, 11 AFTER 400 NS;
data_bus        <= "11110001", "00110001" AFTER 100 NS, "10000001" AFTER 200 NS;
```

راه حل دوم نیز بدین صورت است که می توانیم برای این عملیات نیز یک پروسس بنویسیم تا مقادیر را به سیگنال ها نسبت دهد که کد آن به شرح ذیل می باشد.

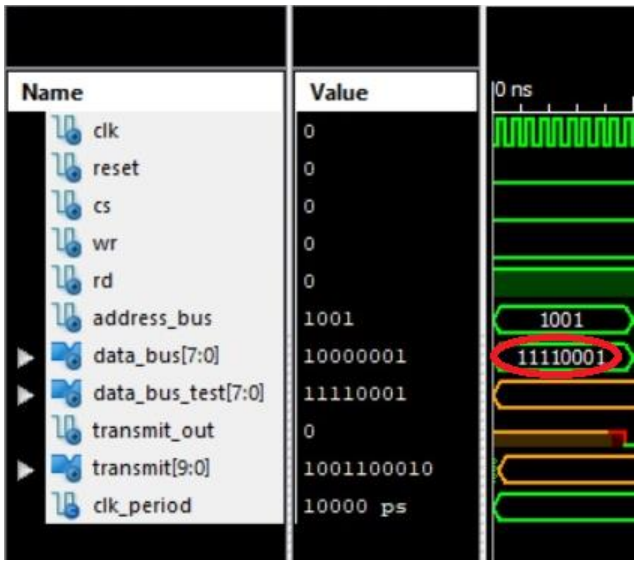
```
    sim : PROCESS
    BEGIN
        cs <= '0';
        wr <= '0';
        address_bus <= 9;
        data_bus <= "11110001";
        WAIT FOR 100 NS;
        address_bus <= 10;
        data_bus <= "00110001";
        WAIT FOR 100 NS;
        address_bus <= 11;
```

```
        data_bus <= "10000001";
        WAIT FOR 100 NS;
        rd<='0';
        wr<='0';
        WAIT FOR 100 NS;
        address_bus <= 9;
        WAIT FOR 100 NS;
        cs <= '1';
        WAIT FOR 100 NS;
        cs <= '0';
        WAIT FOR 100 NS;
        WAIT;
    END PROCESS;
```

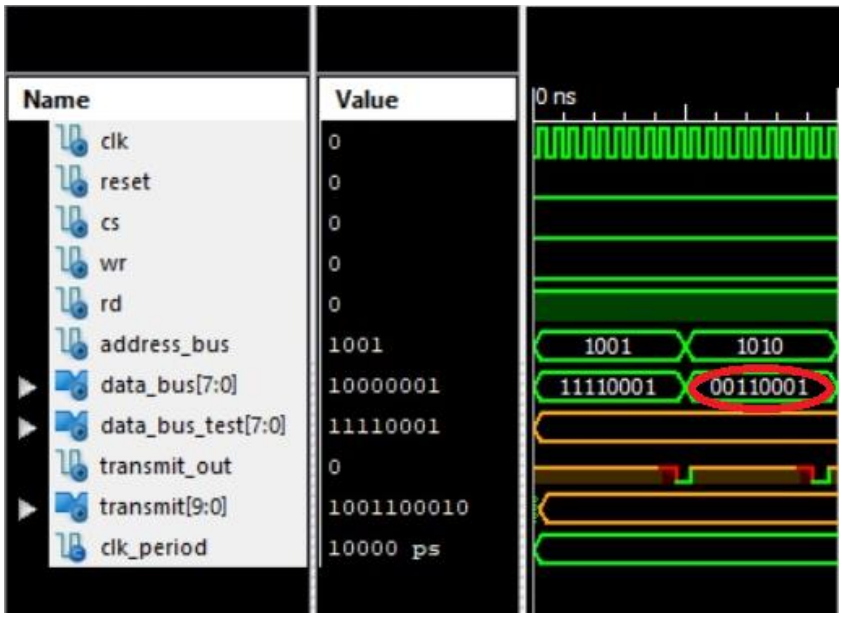
در بخش بعدی اجرای این تست را به صورت تصویری و در قالب یک سناریو دنیال خواهیم کرد.

اجرای یک سناریو

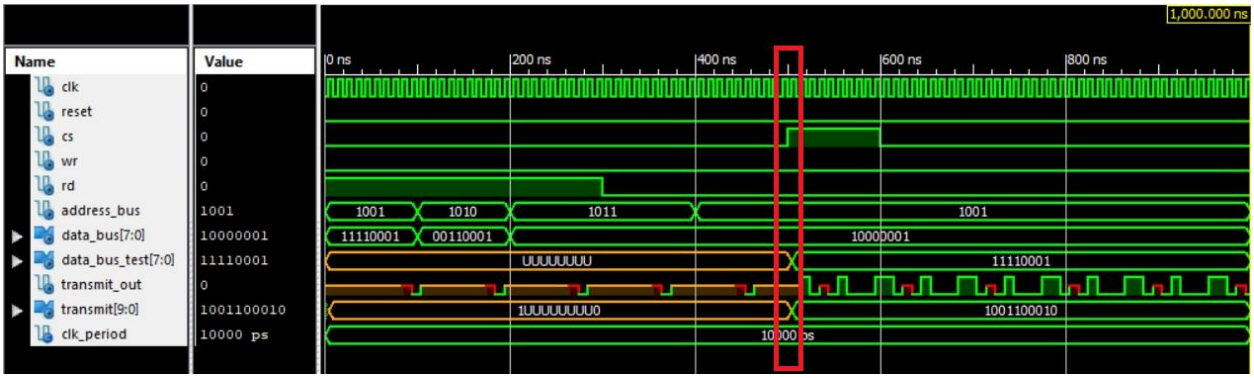
در بخش قبلی ما تست بنچ پروژه را تکمیل نمودیم و پس از آن نیاز است تا نحوه اجرای آن را مشاهده نماییم.



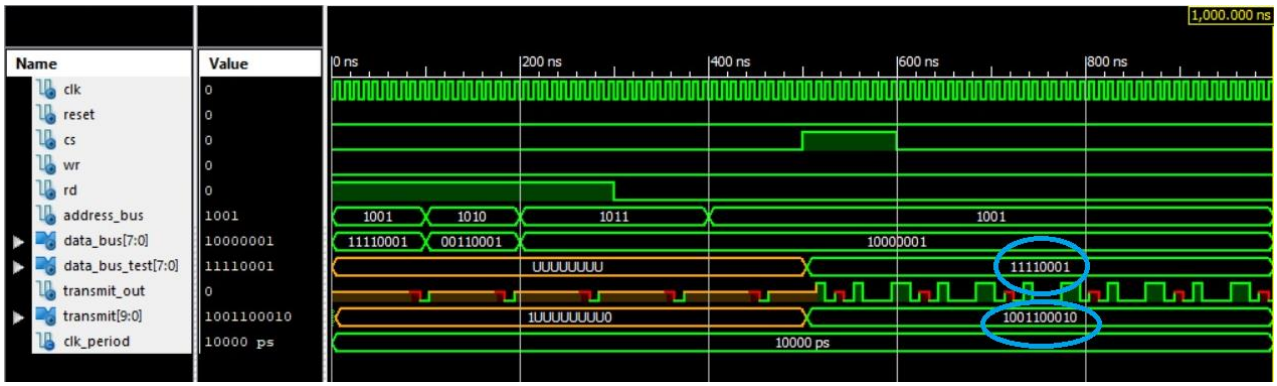
آنچه مشاهده می کنید ۱۰۰ نانوثانیه ابتدایی می باشد و مقداد داخل بیضی آبی رنگ بر روی دیتا باس قرار دارد و تنها مقدار سیگنال rd برابر یک می باشد.



در ۱۰۰ نانو ثانیه دوم نیز یک مقدار جدید بر روی دیتا باس قرار می گیرد که در آدرس ۱۰ جای می گیرد.



در لحظه ۵۰۰ نانو ثانیه مقدار cs برابر یک می شود و همچنین در زمان ۴۰۰ نانوثانیه نیز مقدار rd ۰ می گردد.



آنچه در این تصویر می بینید انتقال مقادیر می باشد که انجام شده است.