درس: آزمایشگاه معماری کامپیوتر(جلسه پنجم)

- مروری بر VHDL(ادامه)
- آزمایش چهارم

نیمسال اول ۱۴۰۱

Process Statement

- تمام object های تعریف شده در یک Process فقط برای همان Process در دسترس هستند و خارج از آن امکان دسترسی به آن object ها وجود ندارد مگر Signal ها.
- دستورهای وقفه را نمی‌توان در Process های دارای لیست حساسیت به کار برد. حتما باید بدون لیست حساسیت باشند.
- هیچ دستور وقفه ای قابلیت سنتز شدن به مدار واقعی را ندارد.

Process Statement – ادامه

- بر خلاف بخش Concurrent، درون Process می‌توان در چند جای مختلف به یک متغیر (Signal یا Variable) مقدار داد. در این حالت آخرین مقدار داده شده لحاظ می‌شود.
- اگر لیست حساسیتی برای Process تعریف نشود، آن Process از آغاز اجرای کد تا ابد اجرا می‌شود مگر اینکه دارای دستور وقفه باشد.
- کار با Process های بدون لیست حساسیت نیازمند دقت بیشتری است. ممکن است در شرایط خاصی، سیستم وارد تکرار بی نهایت (infinite loop) شود که باعث hang کردن سیستم در زمان شبیه سازی می‌شود.
- بهتر است تمامی‌سیگنال های دست راستی در لیست حساسیت وارد شوند، مگر اینکه به صورت آگاهانه بدانیم آن سیگنال ها سنکرون با مثلا Clock هستند.
- به یک سیگنال نمی‌توان در چند Process مختلف مقدار assign نمود؛ حتی اگر بدانیم با هم تداخلی ندارند.

تفاوت Signal و Variable

- درون یک Process نمی‌توان Signal تعریف کرد؛ از طرفی در خارج یک Process نمی‌توان Variable تعریف نمود یا به آنها دسترسی داشت.
- نحوه تعریف و مقداردهی به Variable کاملا مشابه Signal است، همانند آنچه که در درس سوم گفته شده است؛ تنها تفاوت در علامت Assignment است که به جای => از =: استفاده می‌شود.
- امکان دسترسی و استفاده از Signal ها در تمام بخش‌های کد از جمله درون Process وجود دارد. در واقع پل ارتباطی محتویات Process با خارج آن همان Signalها هستند.

تفاوت Signal و Variable - ادامه

- عموما در شبیه‌سازی می‌توان شکل موج Signalهای تعریف شده را در تمام ماژول ها و همچنین در ساختار Hierarchy مشاهده نمود اما Variableها را نمی‌توان به سادگی مشاهده نمود؛ اما برخی از نرم افزار ها مانند ModelSim قابلیت نمایش شکل موج Variableها را دارند.
- Variableها به صورت ترتیبی و بی درنگ مقدار جدید را می‌گیرند اما Signalها پس از پایان هر بار اجرای Process مقدار جدید می‌گیرند.
- در Process تنها دستورات Sequential را می‌توان اجرا کرد و در خارج از آن تنها دستورات Concurrent را.

Variableها خیلی ما به ازای سخت افزاری ندارند ولی قبلا گفته بودیم معادل سخت افزاری signal را می توانیم سیم یا باس در نظر بگیریم که البته برای بخش ها و مدارات ترکیبی است و در بخش ترتیبی می توانیم آن را معادل رجیستر در نظر بگیریم.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity Process_ex4 is
        port( A, B : in std_logic_vector(3 downto 0);
                        sum, sub : out std_logic_vector(3 downto 0));
end Process_ex4;

architecture Behavioral of Process_ex4 is

        signal sub_sig, sum_sig : std_logic_vector(3 downto 0);

begin

        PROCESS (A, B)

                variable sum_var, var : std_logic_vector(3 downto 0);

        BEGIN
                sum_var := A + B;
                sum_sig <= sum_var;
                sum <= sum_var;

--              var := sum_sig;
--              sum <= var;

                sub_sig <= A - B;
                sub <= sub_sig;
        END PROCESS;
```

دستور if-then-else

```
if (condition) then
    sequential_statements;
end if;
```

```
if (condition) then
    sequential_statements;
else
    sequential_statements;
end if;
```

```
if (condition) then
    sequential_statements;
elsif (condition) then
    sequential_statements;
elsif (condition) then
    sequential_statements;
…
else
    sequential_statements;
end if;
```

کد T-FF vhdl

علت تعریف temp این است که میخواهیم به عنوان واسط یا همان سیگنال سمت راستی از آن استفاده کنیم بنابراین نمی توانستیم از نوع پورت تعریف کنیم.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity T_FF is
   port (T, reset, clk, clk_enable : in std_logic;
                 Q : out std_logic);
end T_FF;

architecture Behavioral of T_FF is
        signal temp : std_logic;         -- or change Q port mode to BUFFER
begin

    process (reset, clk) is -- asynchronous reset
    begin
                if reset='1' then
                        temp <= '0';
                --elsif (clk'event and clk='1') then
      elsif (rising_edge(clk)) then    -- this function is better than (clk'event and clk = '1')
                        if clk_enable ='1' then
                                temp <= T xor temp;             -- Q(t+1) = T`.Q(t) + T.Q`(t) = Q(t) XOR T
                        end if;
      end if;
    end process;

         Q <= temp;

end Behavioral;
```

JK-FF vhdl کد

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity JK_FF is
port( J, K, rst, clk, clk_en : in  std_logic;
      output : out std_logic);
end JK_FF;

architecture Behavioral of JK_FF is
   signal temp : std_logic;     --see line 24
begin
   process (clk)        --synchronous reset
   begin
               if (clk'event and clk='1') then
      --if rising_edge(Clock) then
         if rst='1' then
            temp <= '0';
         elsif clk_en ='1' then
            if (J='0' and K='0') then
               temp <= temp;
            elsif (J='0' and K='1') then
               temp <= '0';
            elsif (J='1' and K='0') then
               temp <= '1';
            elsif (J='1' and K='1') then
               temp <= not (temp);
            end if;
         end if;
      end if;
   end process;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
entity counter is
port (clk, reset, Loaden, en : in std_logic;
                loadin : in std_logic_vector (3 downto 0) := "0000";
                topvalue : in std_logic_vector (3 downto 0) := "1111";
                dout : out std_logic_vector (3 downto 0);
                ovrf : out std_logic := '0');
end counter;
architecture Behavioral of counter is
begin
        process(clk, reset, loaden, en) --all are asynchronous
                variable c : std_logic_vector (3 downto 0) := (others => '0');
        begin
                if reset='0' then
                        if en = '1' then
                                if loaden = '1' then
                                        dout <= loadin;
                                        c := loadin;
                                elsif rising_edge(clk) then
                                        dout <= c;
                                        if c < topvalue then
                                                c := c + 1;
                                                ovrf <= '0';
                                        else
                                                ovrf <= '1';
                                                c := "0000";
                                        end if;
                                end if;
                        end if;
                else    -- if reset = '1'
                        dout <= "0000";
                        c := "0000";
                end if;
        end process;
end Behavioral;
```

سیگنال کنترلی که اگر ۱ باشد مقدار PARALLEL INPUT ای که وارد مدار می شود را به عنوان عدد شروع شمارش در نظر می گیرد

زمانی که ۱ باشد محتویات شمارنده را ۰ می کند

شمارش بالا رونده را فعال می کند

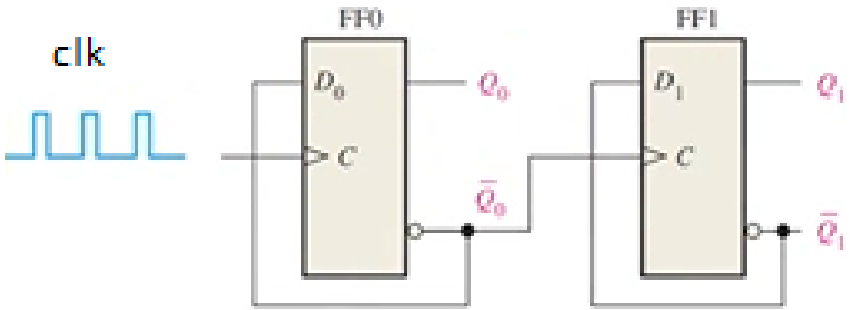
بهترین حالت اینه که این سیگنال ها با کلاک سنکرون باشند

چون مدار به صورت ترتیبی است C به صورت VARIABLE که همان رجیستر در نظر گرفته می شود تعریف شده است

الویت بالاتر نسبت به بقیه

شمارنده آسنکرون

- شمارنده آسنکرون ۲ بیتی

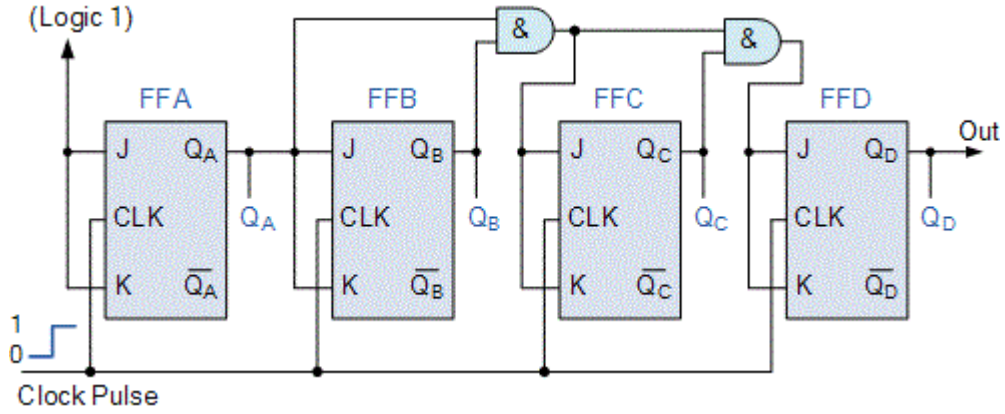


شمارنده ها

در شمارنده های آسنکرون خروجی یک طبقه از شمارنده مستقیما به ورودی کلاک طبقه بعدی شمارنده متصل می‌شود و این روند در تمام طول زنجیره شمارنده ادامه می‌یابد. چنین پیکربندی باعث می‌شود که در این نوع از شمارنده‌ها مشکلی به نام تاخیر انتشاری (Propagation Delay) به وجود آید. تاخیر انتشاری در واقع به این صورت بیان می‌شود که سیگنال زمان‌بندی در طول گذر از هر فلیپ فلاپ دچار مقداری تاخیر می‌شود.

در شمارنده‌های سنکرون، سیگنال کلاک خارجی به ورودی کلاک همه فلیپ فلاپ‌های مدار متصل می‌شود. بنابراین تمام فلیپ فلاپ‌های شمارنده در یک لحظه و به صورت همزمان و موازی با یکدیگر کلاک می‌شوند و یک رابطه زمانی ثابت وجود دارد. به عبارت دیگر، تغییر در سیگنال خروجی به صورت همگام با سیگنال کلاک اتفاق می‌افتد.

نتیجه این همگام‌سازی این است که تمام بیت‌های تکی خروجی دقیقا در یک زمان یکسان در پاسخ به سیگنال کلاک عمومی تغییر حالت می‌دهند و هیچ حالت موجی وجود نخواهد داشت، دقیقا به همین دلیل است که در مدار تاخیر انتشار نیز از بین می‌رود.

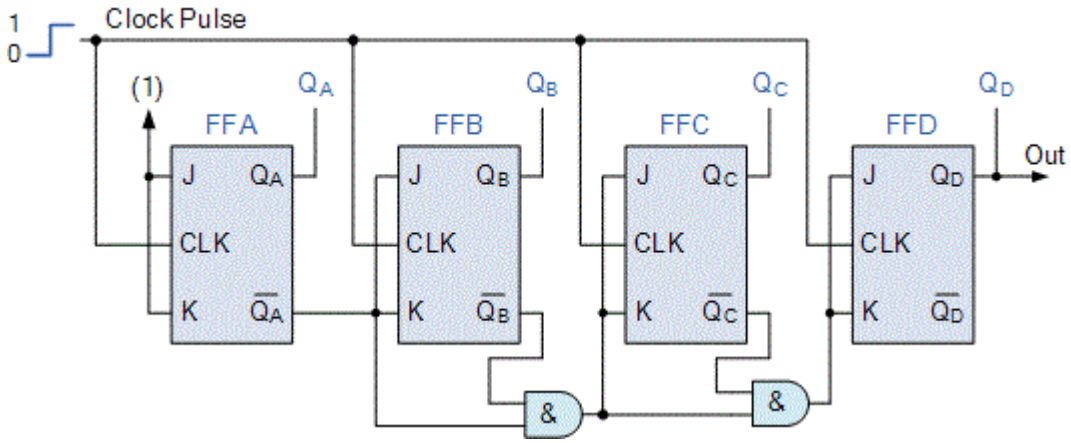


به دلیل این‌که یک شمارنده سنکرون ۴ بیتی در هر پالس ساعت شمارش را انجام می‌دهد، در نتیجه شمارنده، شمارش را به سمت بالا و از ۰ (۰۰۰۰) تا ۱۵ (۱۱۱۱) انجام می‌دهد.

به همین علت است که این نوع از شمارنده‌ها را شمارنده سنکرون ۴ بیتی بالا شمار (4Bit Synchronous Up Counter) نیز می‌گویند.

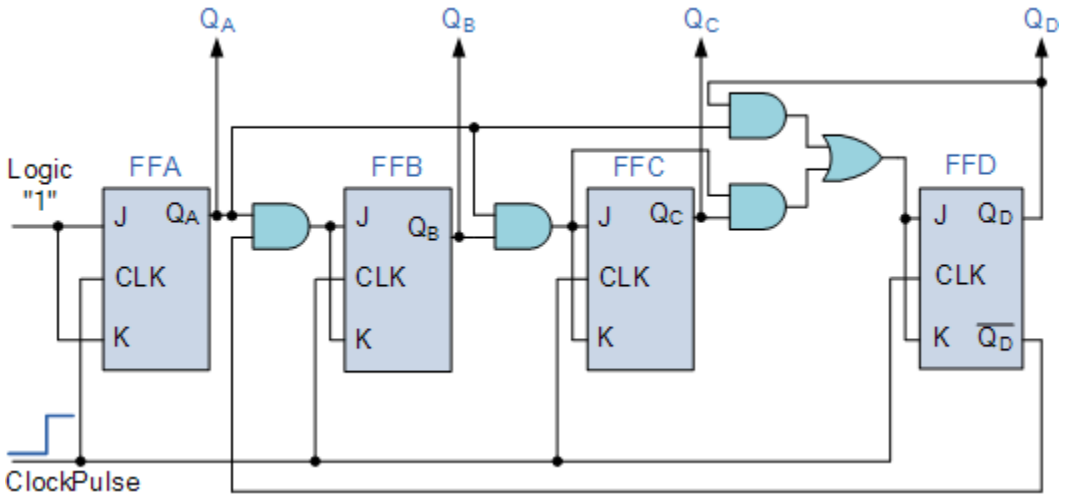
یک شمارنده سنکرون ۴ بیتی پایین شمار را نیز می‌توان به راحتی پیاده‌سازی کرد. توجه کنید که این بار گیت AND به خروجی $\overline{Q}$ فلیپ فلاپ‌ها متصل می‌شود.

تصویر زیر نمایی از چنین مداری را نشان می‌دهد. در این حالت شکل موج دیاگرام زمان‌بندی دقیقا معکوس دیاگرام زمان‌بندی شمارنده آسنکرون ۴ بیتی بالا شمار است.



شمارنده سنکرون ده دهی چهار بیتی

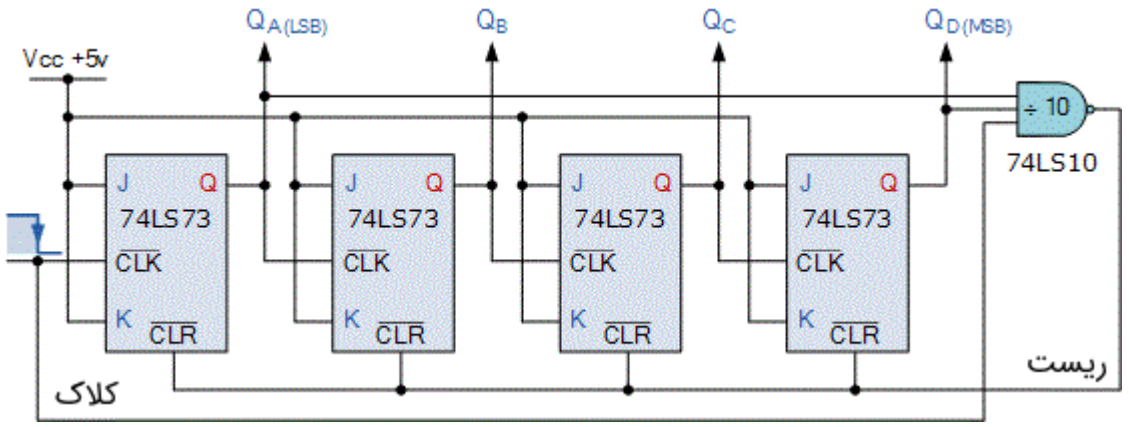
یک شمارنده سنکرون ده دهی چهار بیتی نیز می‌تواند با استفاده از شمارنده باینری سنکرون برای تولید دنباله شمارش اعداد از ۰ تا ۹ ساخته شود. به کمک چند وسیله منطقی اضافی (برای پیاده‌سازی دنباله حالت‌های مطلوب)، می‌توان یک شمارنده باینری استاندارد را به یک شمارنده ده دهی تبدیل کرد. در این شمارنده، بعد از این که شمارنده به عدد ۱۰۰۱ برسد، شمارنده به مقدار ۰۰۰۰ ریست می‌شود. در تصویر زیر مدار یک شمارنده سنکرون ده دهی چهار بیتی دیده می شود.

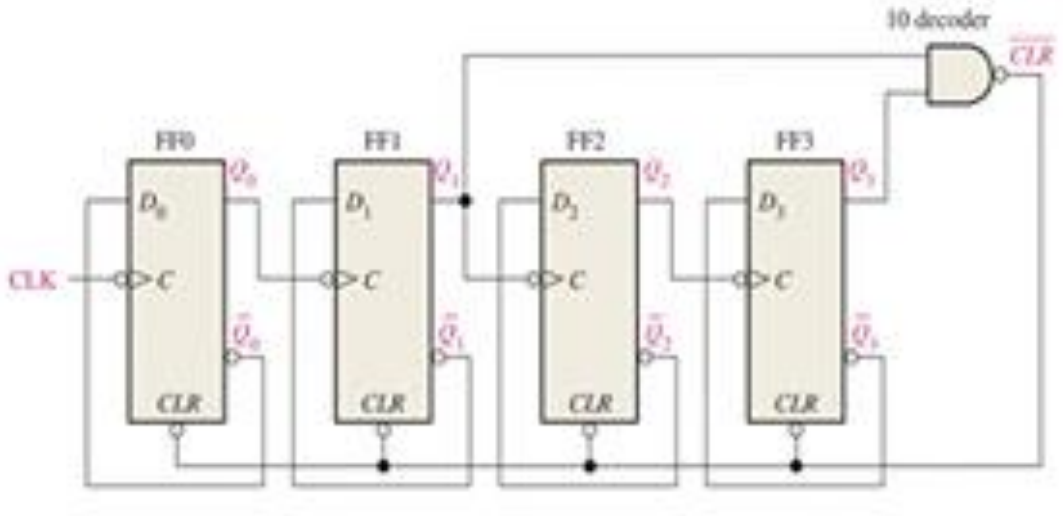


شمارنده آسنکرون ده دهی

اگر یک شمارنده آسنکرون با ۱۶MOD را انتخاب کنیم و آن را با گیت‌های منطقی اضافی اصلاح کنیم، می‌توانیم به خروجی شمارنده دهدهی (مدار مقسم بر ده) دست یابیم و از آن در شمارنده‌های دسیمال استاندارد و مدارات حسابی استفاده کنیم.

این شمارنده‌ها معمولا تحت عنوان شمارنده‌های دهدهی شناخته می‌شوند. در یک شمارنده دهدهی نیاز است تا زمانی که شمارش به ضرایب ده برسد، خروجی به مقدار صفر ریست شود. به عبارت دیگر خروجی باید برابر با DCBA=1010باشد. برای رسیدن به این هدف، باید وقوع این حالت را به ورودی ریست مدار فیدبک دهیم. یک شمارنده با رشته شمارش از ۰۰۰۰ باینری تا ۱۰۰۱ شمارنده BCDیا شمارنده دسیمال کد باینری (Binary Coded Decimal) نامیده می‌شود؛ زیرا دنباله ده حالتی خروجی آن همان کد BCD است. اما شمارنده‌های دهدهی باینری بسیار متداول‌تر هستند.





•شمارنده‌های سنکرون می‌توانند از فلیپ فلاپ‌های نوع T و یا D ساخته شوند.

•طراحی شمارنده‌های سنکرون از شمارنده‌های آسنکرون بسیار ساده‌تر است.

•این شمارنده‌ها به این دلیل سنکرون نامیده می‌شوند که ورودی کلاک فلیپ فلاپ‌ها همگی با هم در یک لحظه و با یک پالس کلاک تغذیه می‌شوند.

•به دلیل این پالس کلاک مشترک تمام حالت‌های خروجی همزمان با یکدیگر تغییر وضعیت می‌دهند.

•به دلیل این که همه فلیپ فلاپ‌ها به صورت جداگانه از پالس کلاک مشترکی تغذیه می‌شوند، بر خلاف شمارنده‌های آسنکرون، هیچ تاخیر انتشاری در شمارنده های سنکرون وجود ندارد.

•شمارنده‌های سنکرون گاهی با نام شمارنده‌های موازی هم شناخته می‌شوند؛ زیرا در این شمارنده‌ها پالس کلاک به صورت موازی به تمام فلیپ فلاپ‌ها وارد می‌شود.

•دنباله شمارش با استفاده از یک گیت منطقی کنترل می‌شود.

•در مقایسه با یک شمارنده آسنکرون، شمارنده‌های سنکرون دارای سرعت عملکرد بسیار بالاتری هستند.

گزارش کار:

کد توصیف هر یک از موارد زیر

- یک شمارنده بالاشمار دهدهی آسنکرون با استفاده از هر فلیپ فلاپ دلخواه
- یک شمارنده پایین شمار دودویی ۴ بیتی با استفاده از هر فلیپ فلاپ دلخواه

بررسی صحت عملکرد هریک از موارد فوق با استفاده از testbech و یا Simulation و تصویری از شکل موج ها