درس: آزمایشگاه معماری کامپیوتر(جلسه ششم)

- مروری بر VHDL(ادامه)
- آزمایش پنجم

نیمسال اول ۱۴۰۱

۲ نوع پورت داریم که یا Generic است یا همان پورت عادی که قبلا معرفی کرده بودیم.

برای اینکه از یک ماژولی در جای دیگر استفاده کنیم حتما بایستی یک بار کامپایل شده باشد.

در بخش libraries که وارد شویم می بینیم که library پیش فرض ما work نام دارد

یعنی آن محلی که قرار است ما این کدها را آنجا کامپایل کنیم(work place)

تمام کدهایی که قبلا یکبار simulation آن ها را انجام داده ایم در اینجا می توانیم مشاهده کنیم.

پس مجازیم که از اینها component بسازیم.

در حالت عادی وقتی test bench خود سیستم را استفاده می کنیم به پورت های generic هم مقدار ثابت

می دهد که ما می توانیم همان بخش از کد را کپی نموده و اینجا اضافه کنیم تا حالت کلی داشته باشد.

و با تعریف یک مقدار ثابت به متغیرهای داخل سیگنال مقداردهی کنیم.

و کافیست در بخش port map یک generic map به تعداد مد نظر نوشته شود.

(می توانیم مثال mux را ببینیم.)

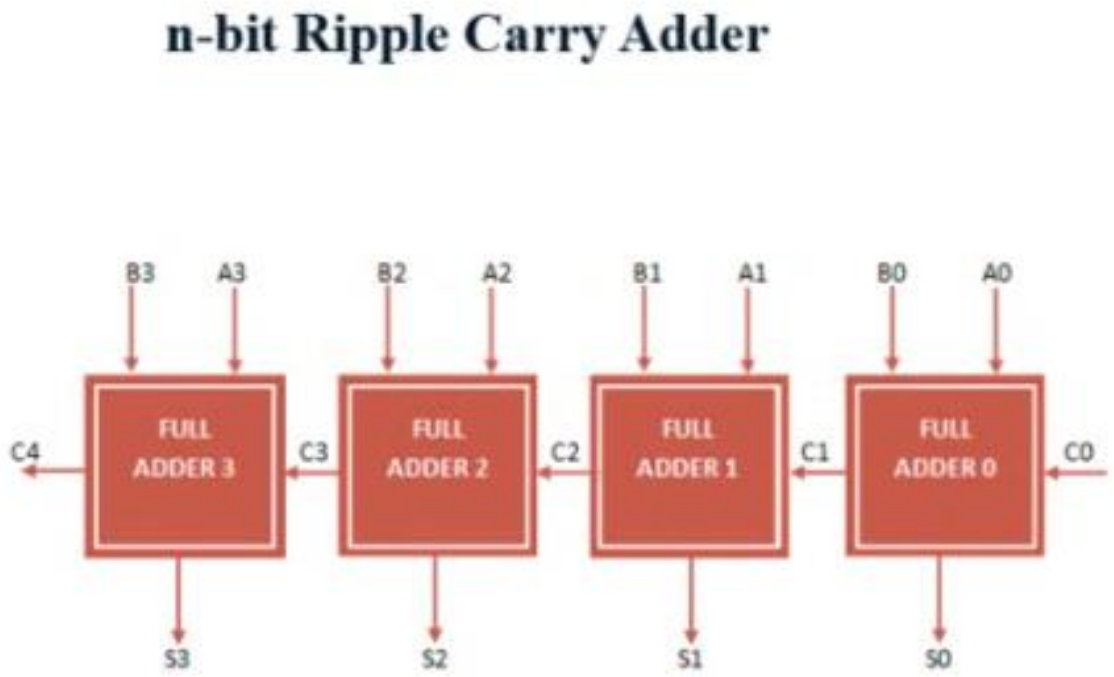
```
circuit_under_test : multiplexer GENERIC MAP (8)
PORT MAP (a_s, b_s, c_s, X"04", e_s, sel_s, OPEN);
```

ممکن است زمانی بخواهیم پورت ما همواره مقدار ثابتی داشته باشد و نمیخواهیم به شکل سیگنال باشد به سادگی می توانیم این کار را انجام دهیم
یا اینکه بخواهیم پورتی را به صورت open یا not connectedبگذاریم و اگر خروجی را بررسی کنیم به شکل U نمایش داده می شود.
در حالت by name هم به همین ترتیب است و برای حالت open میتوانیم اصلا ننویسیم.

```
circuit_under_test : multiplexer GENERIC MAP (8)
PORT MAP (
   a => a_s,
   b => b_s,
   d => X"04",
   c => c_s,
   sel => sel_s,
   e => e_s,

   output => output_s);
```

برای سادگی در ساختارهای تکراری می توانیم از دستوری استفاده کنیم که کار را برای ما ساده تر می کند.



دستور For-Generate

نوشتن label اجباری است

```
<label> : for <index_name> in <lower_limit> to <upper_boundary> generate
(Begin)
        <concurrent statements> ;
End generate (label) ;
```

در حالت کلی بهتره استفاده نشه اما برای ماهیت های تکراری مانند RCA یا SHIFT REGISTER استفاده میشوند تا نوشتن برنامه را ساده تر کنند.

دستور For-Generate

- به منظور کنار هم قرار دادن آرایه یا تعداد معینی از Component های یکسان استفاده می گردد.
- از جمله ساختار های Concurrent می باشد و می تواند شامل ساختارهای Concurrent دیگر باشد.
- می توان به صورت nested یا تو در تو نیز از آن استفاده نمود.
- قابل سنتز است.

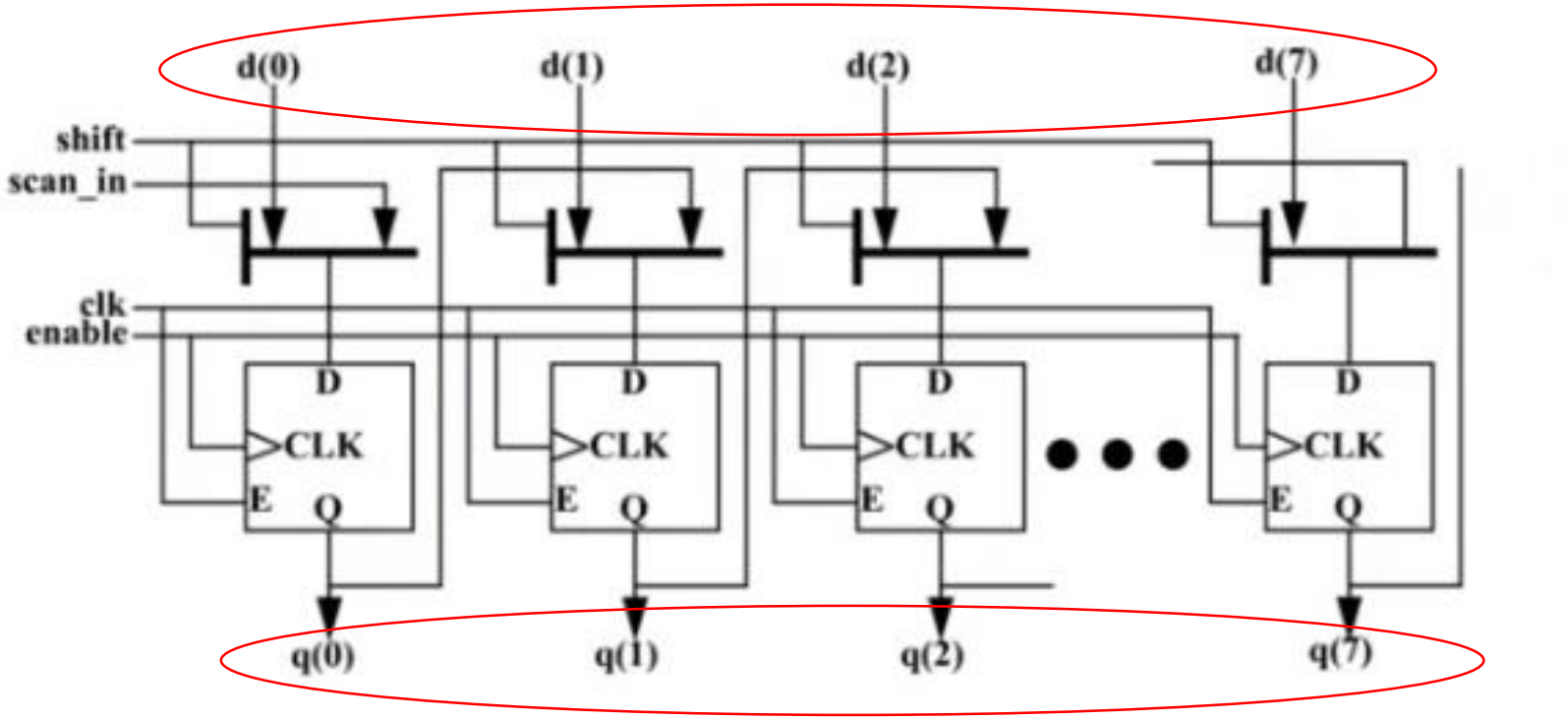
تعمیم دستور قبل

دستور For-If-Generate

```
label : for <index_name> in <lower_limit> to <upper_boundary> generate  (Begin)
        <label> : if  condition(1) generate
                <statements> ;
        end generate <label> ;
        …
        <label> : if  condition(n) generate
                <statements> ;
        end generate <label> ;
End generate <label> ;
```

مثال 8bit Shift Register

DATA های موازی که LOADمی شوند



اگرshift=0 به صورت موازی loadمیشوند واگر نه scan_in وارد شده و به صورت سریال منتقل می شود

Direct Instantiation

- اگر یک کد (طراحی) قبلا Compile شده باشد، فایل آن به کتابخانه Work اضافه می‌شود. در این حالت می‌توان به راحتی آن‌را instantiate نمود.
- در این روش، تمامی‌قابلیت‌هایی که Component instantiation در اختیار ما قرار می‌دهد، وجود ندارد.

Direct Instantiation

```
Architecture arch_name of design_name is

        ... (Component بدون تعريف)

BEGIN

        U1 : ENTITY  WORK.a_design GENERIC MAP (...) PORT MAP (...) ;

        ...

End arch_name;
```

Shift Register

شیفت رجیستر (Shift Register) یا ثبات انتقال‌دهنده یکی از انواع مدارات منطقی ترتیبی است که در ذخیره‌سازی و انتقال داده‌های باینری کاربرد دارد.

این ادوات ترتیبی داده‌های موجود در ورودی خود را بارگذاری (Load) می‌کنند و سپس آن‌ها را در هر پالس ساعت به خروجی منتقل (Shift) می‌کنند .از این‌رو به آن‌ها شیفت رجیستر می‌گویند.

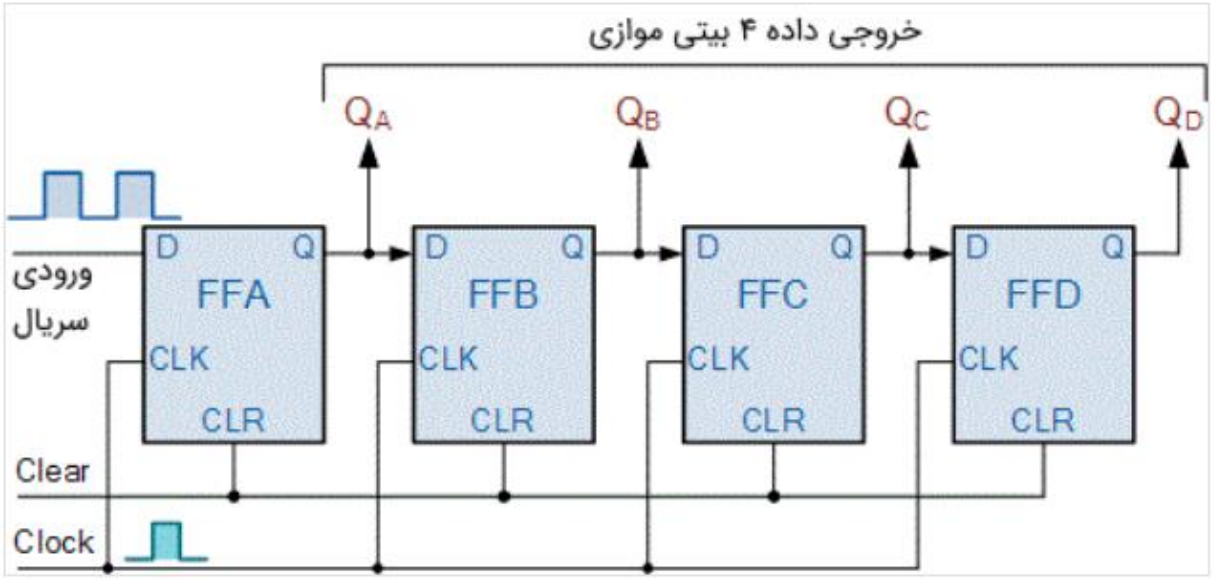
مدهای کاری شیفت رجیسترها

لچ‌های داده تکی که یک شیفت رجیستر را تشکیل می‌دهند، همگی از یک سیگنال کلاک مشترک تغذیه می‌شوند، بنابراین با یکدیگر سنکرون هستند. آی‌سی‌های شیف رجیستر معمولا با یک قابلیت پاک کردن (Clear) یا ریست (Reset) تولید می‌شوند، تا در صورت لزوم، عملیات ست و یا ریست کردن در آن‌ها به وقوع بپیوندد.

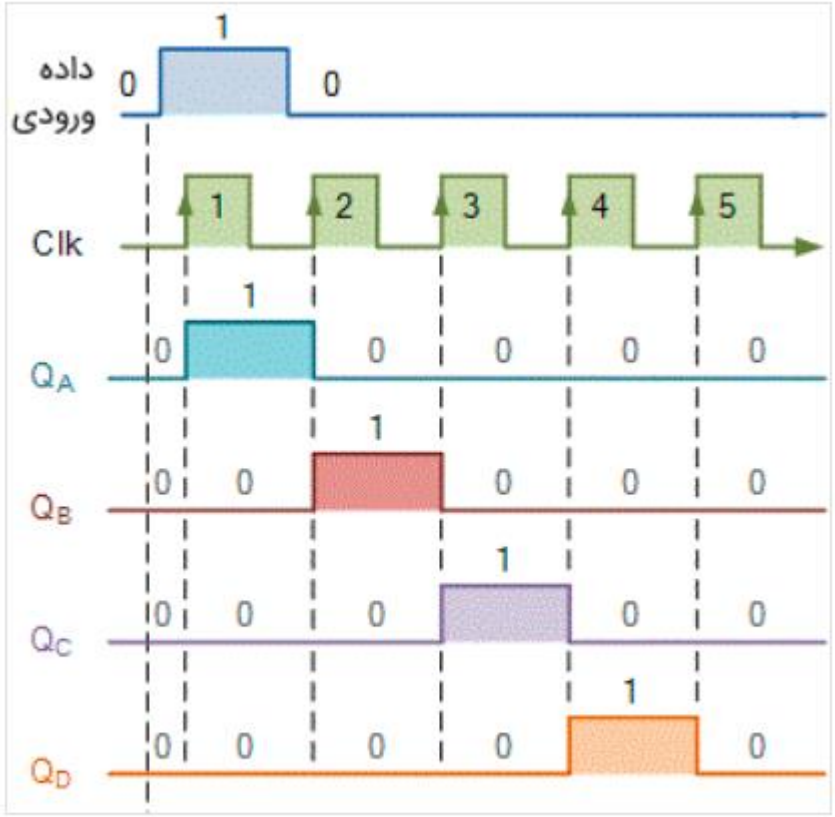
شیفت رجیسترها در یکی از چهار مد کاری مختلف زیر کار می‌کند، که با حرکت داده در طول شیفت رجیستر به وقوع می‌پیوندد.

- **ورودی سریال-خروجی موازی (Serial-in to Parallel-out) یا SIPO** : در این مد، شیفت رجیستر با داده سریال (در هر لحظه یک بیت) بارگذاری می‌شود، در حالی‌که داده‌های ذخیره شده به صورت موازی در خروجی ظاهر می‌شوند.
- **ورودی سریال-خروجی سریال (Serial-in to Serial-out) یا SISO** : در این مد، در هر لحظه یک داده به صورت سریال به شیفت رجیستر وارد یا خارج می‌شود و جهت انتقال می‌تواند تحت کنترل پالس ساعت به سمت راست یا چپ باشد.
- **ورودی موازی-خروجی سریال (Parallel-in to Serial-out) یا PISO** : داده‌ها با هم به صورت موازی در ورودی بارگذاری می‌شوند و تحت کنترل پالس ساعت در هر زمان یک بیت به صورت سریال به خروجی شیفت رجیستر منتقل می‌شود.
- **ورودی موازی-خروجی موازی (Parallel-in to Parallel-out) یا PIPO** : در این مد داده‌های موازی با هم در ورودی بارگذاری می‌شوند و با هم تحت یک پالس ساعت به خروجی متناظر منتقل می‌شوند.

ورودی سریال-خروجی موازی

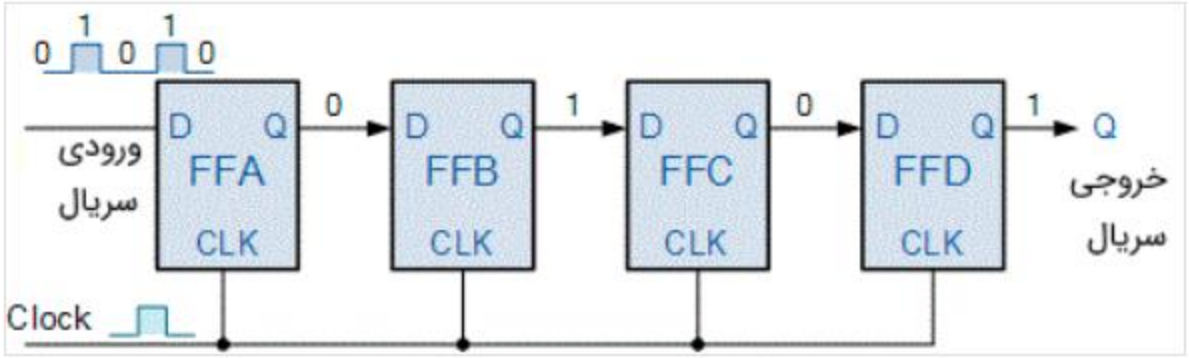


شیفت رجیستر ۴ بیتی ورودی سریال-خروجی موازی

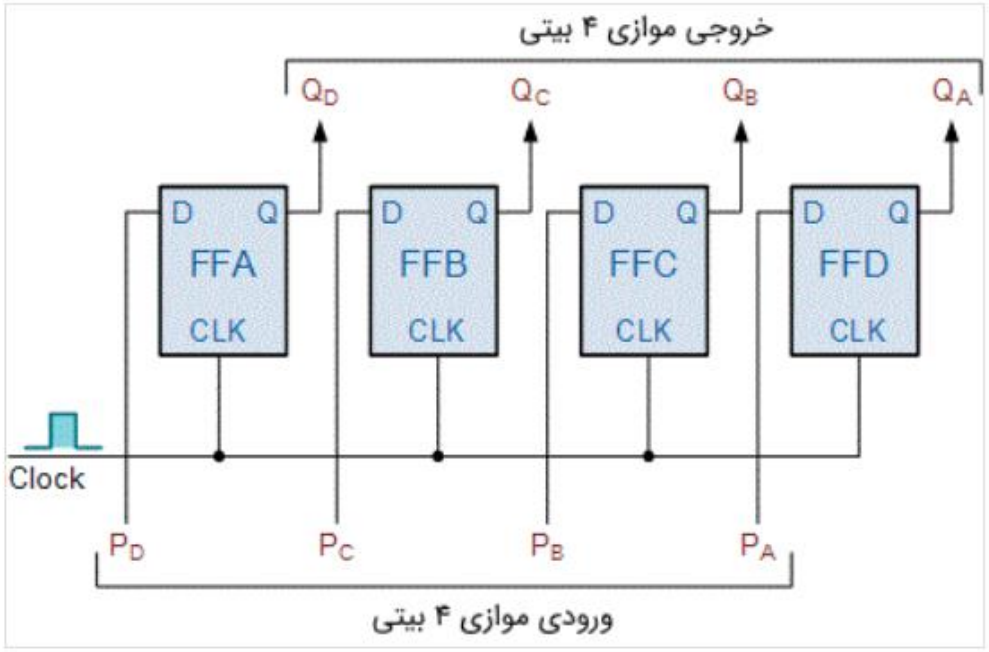


انتقال عدد یک در طول شیفت رجیستر از چپ به راست

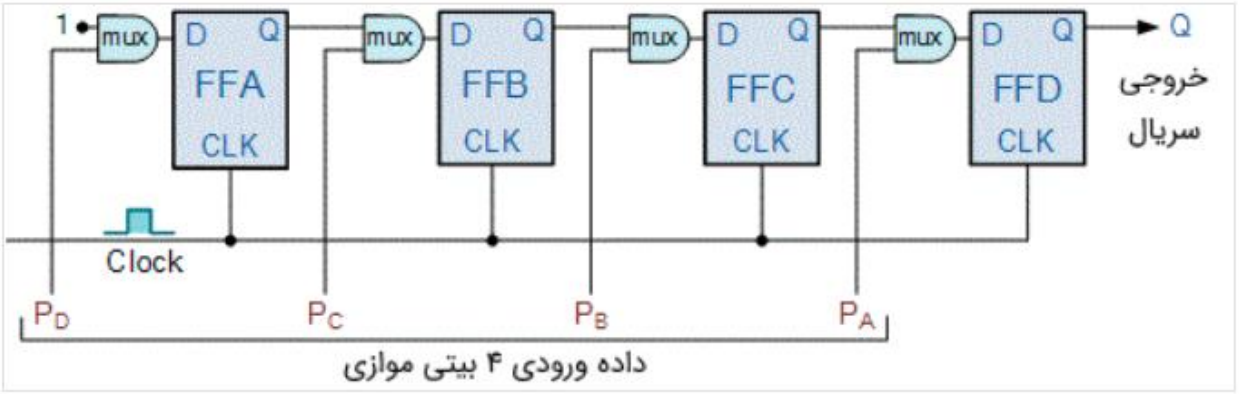
مودهای کاری دیگر



شیفت رجیستر ۴ بیتی ورودی سریال-خروجی سریال



شیفت رجیستر ۴ بیتی ورودی موازی-خروجی موازی



شیفت رجیستر ۴ بیتی ورودی موازی-خروجی سریال

شیفت رجیستر عمومی (Universal):

این قطعات به صورت چهار بیتی و چندکاره (Multi-Function) هستند، یعنی می‌توانند برای انتقال به چپ، انتقال به راست و نیز در هر چهار مد سریال به سریال، سریال به موازی، موازی به سریال و موازی به موازی مورد استفاده قرار گیرند؛ به همین دلیل به آن‌ها آی‌سی‌ها عمومی می‌گویند.

شیفت رجیستر عمومی (Universal):

هم ورودی و هم خروجی می توانند parallel یا serial باشند.

چیز جدیدی که اینجا می بینیم:
استفاده از چند architecture است.
(در این مثال ۲ تا)
و روشی که برای نوشتن چند architecture و تست کردن آن ها استفاده می شود مشابه است.
بخش entity برای هر دو architecture مشابه است یعنی پورت ها و بخش interface مشابهی دارند.
متفاوت در توصیف و behavior هستند.

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


entity universal_reg is
    generic(n : positive := 4);
    port(parallelin : in std_logic_vector(n-1 downto 0);
      load_en, shift_r, shift_l, serial_in, clk, en : in std_logic;
      p_out : out std_logic_vector(n-1 downto 0);
      s_out : out std_logic);
end universal_reg;

architecture Behaviorall of universal_reg is
begin
    process (clk)  -- all inputs are synchronous with clk
        variable temp_p_out : std_logic_vector(n-1 downto 0);
    begin
        if rising_edge(clk) then
            if (load_en = '1') then
                temp_p_out := parallelin;
                s_out <= temp_p_out(n-1);    -- for instance, 'Z' or temp_p_out(0)
            elsif (en = '1') then
                if (shift_l = '1') then
                    s_out <= temp_p_out(n-1);
                    temp_p_out := temp_p_out(n-2 downto 0) & serial_in;
                elsif (shift_r = '1') then
                    s_out <= temp_p_out(0);
                    temp_p_out := serial_in & temp_p_out(n-1 downto 1);
                else
                    temp_p_out := parallelin;
                    s_out <= temp_p_out(n-1);  -- for instance, 'Z' or temp_p_out(0)
                end if;
            end if;
            p_out <= temp_p_out;
        end if;
    end process;
end Behaviorall;
```

در مرحله loadمعنی نداره و در مرحله شیفت معنی پیدا می کند
جنبه مقداردهی اولیه دارد.

شیفت به سمت چپ

شیفت به سمت راست

بدون شیفت

چون پورت s_out در هیچ جا به عنوان سیگنال سمت راست استفاده نشده
مثلا برخلاف متغیر temp_p_out
پس مستقیما می توانیم مقداردهی کنیم.
اما برای p_out یک متغیر میانی تعریف کردیم و در نهایت مقدارش را داخل p_out ریختیم.

```
44  architecture Behavioral2 of universal_reg is
45      shared variable temp_p_out : std_logic_vector(n-1 downto 0);
46  -- signal temp_p_out : std_logic_vector(n-1 downto 0);
47  begin
48     process (clk)  -- all inputs are synchronous with clk
49     begin
50        if rising_edge(clk) then
51           if (load_en = '1') then
52              temp_p_out := parallelin;
53              s_out <= temp_p_out(n-1);     -- for instance, 'Z' or temp_p_out(0)
54           elsif (en = '1') then
55              if (shift_l = '1') then
56                 s_out <=  temp_p_out(n-1);
57                 temp_p_out :=  temp_p_out(n-2 downto 0) & serial_in;
58              elsif (shift_r = '1') then
59                 s_out <=  temp_p_out(0);
60                 temp_p_out :=  serial_in & temp_p_out(n-1 downto 1);
61              else
62                 temp_p_out :=  parallelin;
63                 s_out <= temp_p_out(n-1);  -- for instance, 'Z' or temp_p_out(0)
64              end if;
65           end if;
66        end if;
67  --    p_out <= temp_p_out;
68     end process;
69     process (clk)
70     begin
71        if rising_edge(clk) then
72           p_out <= temp_p_out;
73        end if;
74     end process;
75  -- p_out <= temp_p_out;
76  end Behavioral2;
```

در این architecture نیز همه چیز مشابه قبل است اما چیز جدیدی که داریم shared variable است فقط به این ترتیب نوشتیم تا تعدد architecture و نحوه استفاده از shared variable را ببینیم.

برای ارتباط بین پروسس ها و ارتباط با بیرون از آن از سیگنال استفاده می کردیم.
و همچنین گفتیم هر متغیر برای هر پروسس به صورت اختصاصی استفاده می شد.
اما shared variable این امکان رو میده که این متغیر را بین پروسس ها استفاده کنیم.(در استفاده از این ها باید دقت کنیم زیرا multiple assignment بین پروسس ها نباید اتفاق بیفتد.)
معمولا استفاده نمی شود.
مزیتش طبق چیزی که قبلا هم گفتیم در جا مقدار میگیرد و نیازی نیست یک سیکل بگذرد مشابه مقدار سیگنال.
نمی توانند در لیست حساسیت قرار بگیرند.
در مدلسازی ها و برنامه نویسی شی گرا استفاده می شوند.
قابلیت سنتز دارند.
در test bench فقط timing رو باید دقت کرد.

کد درست و برنامه درست همون architecture اول است و به ازای آن می توانیم شماتیک را ببینیم.

اگر در سطح تکنولوژی نگاه کنیم خواهیم دید که از FF های خود FPGA استفاده شده است.

در شماتیک اول چیزی بابت متغیر نمیبینیم اما در شماتیک دوم چون اشتراکی بوده است تعدادی FF اضافه مشاهده می گردد.

گزارش کار

۱. شیفت رجیسترهای ۴ بیتی در مدهای

- PIPO
- PISO
- SISO
- SIPO

طراحی نموده و صحت عملکرد هر یک را بررسی نمایید و تصویری از شبیه سازی صحیح هر یک را الصاق نمایید.

۲. کاربرد ثبات ها در کامپیوتر را بنویسید.

۳. طراحی ای از یک رجیستر ارائه نمایید.(دلخواه)