به نام خدا

گزارش‌کار آزمایش دوم

علی نوروزبیگی – فرهاد امان

هدف از انجام این آزمایش توصیف چهار مدار مجزای Decoder2x4, Encoder4x2, MUX4x1, Comparator4bit می باشد. تمام توصیف های صورت گرفته به صورت Structural بوده و با استفاده از زبان VHDL صورت گرفته اند.

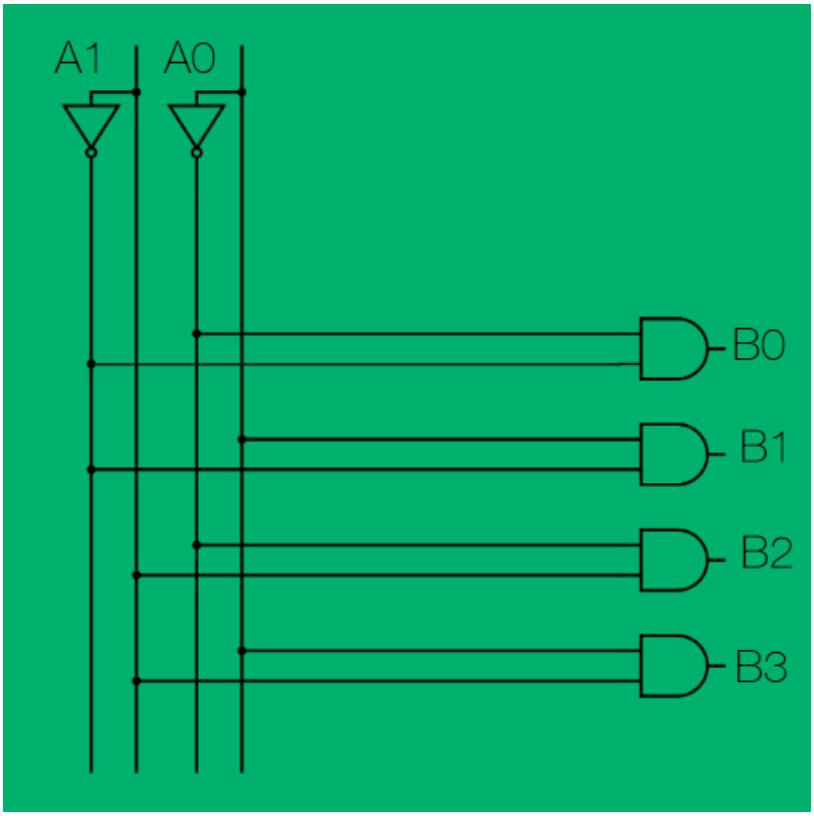
در ابتدای کار نیاز داریم تعدادی از گیت های پایه را توصیف نماییم تا از آن ها به عنوان component در مدار های بزرگترمان استفاده کنیم این گیت های پایه شامل and_gate2, and_gate3, and_gate4, and_gate5 که گیت های and با تعداد ورودی های 2 تا 5 می باشند.

کد توصیف مربوط به and_gate2 به عنوان نمونه در زیر آمده است.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_gate_2 is
	port(a : in std_logic;
		b : in std_logic;
		c : out std_logic
		);
end and_gate_2;
architecture Structural of and_gate_2 is
begin
c <= a and b;
end Structural;
```

در ادامه نیاز به توصیف گیت های xor_gate, nor_gate, not_gate, or_gate و همچنین گیت buffer با تعداد ورودی های متفاوت داریم. توصیف های مربوطه به طور کلی مانند توصیف گیت and می باشند. کد های مربوطه در فایل پیوست پروژه موجود است.

Decoder2x4



این مدار یک مدار کدگشای 2 بیت به 4 بیت می باشد و به این صورت عمل می کند که یک عدد 2 بیتی را در ورودی دریافت کرده و با توجه شماره آن عدد یکی از 2^n خروجی روشن خواهد شد.

همانطور که می بینید مدار دیکودر در شکل بالا مشخص شده است در این مدار نیاز به استفاده از 4 گیت and_gate2 و استفاده از 2 گیت not_gate داریم. این دو گیت را قبلا پیاده سازی کردیم در اینجا تنها نیاز است که در بخش architecture آن ها را به صورت کامپوننت تعریف کنیم.

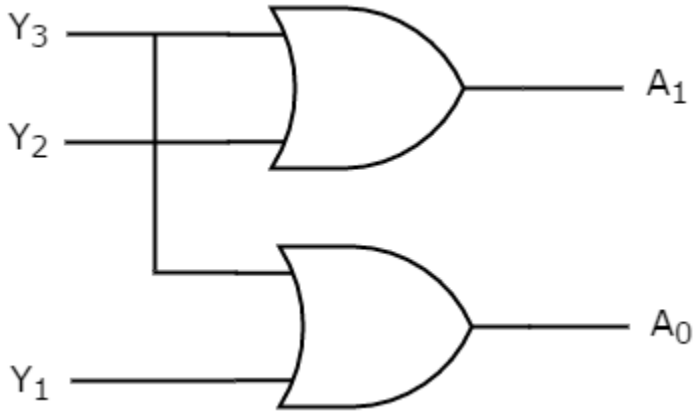
```
component not_gate is
    port( a : in std_logic;
          anot : out std_logic
```

```
        );
end component not_gate;


component and_gate_2 is
    port(a : in std_logic;
         b : in std_logic;
         c : out std_logic
         );
end component and_gate_2;
```

بعد از تعریف کردن کامپوننت ها می توانیم داخل بخش begin, end از آن ها به تعداد دلخواه نمونه سازی کرده و استفاده کنیم.

```
signal anot: std_logic_vector(1 downto 0);
begin
    not_gate0 : not_gate port map (a => a(0), anot => anot(0));
    not_gate1 : not_gate port map (a => a(1), anot => anot(1));
    and_gate_20 : and_gate_2 port map(a => anot(0), b => anot(1), c => b(0));
    and_gate_21 : and_gate_2 port map(a => a(0), b => anot(1), c => b(1));
    and_gate_22 : and_gate_2 port map(a => anot(0), b => a(1), c => b(2));
    and_gate_23 : and_gate_2 port map(a => a(0), b => a(1), c => b(3));
end Structural;
```

Encoder4x2



در شکل بالا مدار یک انکودر 4 بیت به 2 بیت مشاهده می شود این مدار درست عکس مدار دیکودر عمل کرده و از بین 2^n ورودی آن باید تنها یک ورودی روشن باشد سپس عددی n بیتی به عنوان شماره ورودی که روشن است در خروجی مشاهده خواهد شد.

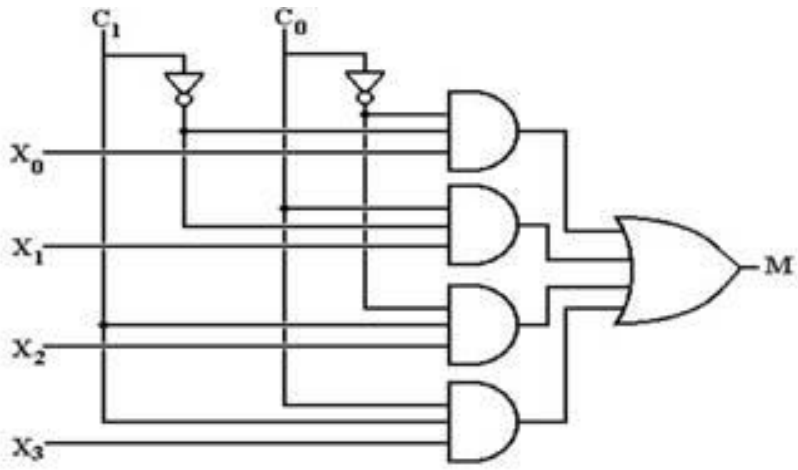
توجه کنید که در یک انکودر عادی باید همیشه مطمئن باشیم که تنها یکی از ورودی ها روشن باشد در غیر این صورت انکودر عملکرد درستی را از خود نشان نخواهد داد. همانطور که می بینید در مدار یک انکودر تنها از دو عدد or_gate استفاده شده همچنین کم ارزش ترین بیت ورودی هیچ تاثیری بر خروجی انکودر ندارد.

مانند قبل باید گیت or خود را به صورت کامپوننت تعریف کنیم و سپس از آن به تعداد لازم نمونه سازی انجام دهیم.

```
architecture Structural of encoder_4bit is
    component or_gate is
        port(a : in std_logic
             b : in std_logic;
             c : out std_logic
             );
     end component or_gate;
begin
    or_gate0 : or_gate port map (a => a(1), b => a(3), c => b(0));
```

```
    or_gate1 : or_gate port map (a => a(2), b => a(3), c => b(1));
end Structural;
```

MUX4x1



در شکل بالا مدار یک مالتیپلکسر 4 بیت به 1 بیت را مشاهده می کنید. مالتیپلکسر ها در واقع مدار های انتخاب گر هستند به این صورت که 2^n ورودی اصلی و n ورودی سلکت دارند. ورودی های سلکت مشخص می کنند که کدام یک از ورودی های اصلی بر حسب شماره باید به خروجی متصل شوند.

همانطور که می بینید در این مدار از 2 عدد not_gate، 4 عدد and_gate3 و 1 عدد or_gate4 استفاده شده است. مانند قبل ابتدا باید گیت های خود را به صورت کامپوننت تعریف کرده و سپس از آن ها نمونه سازی کنیم.

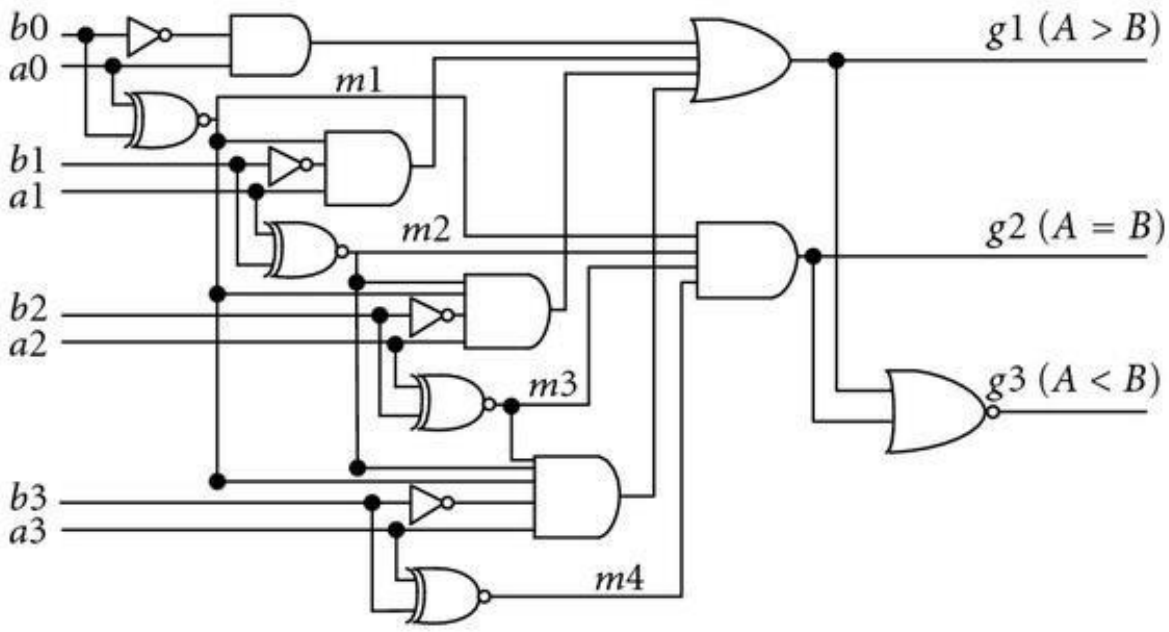
```
architecture Structural of mux_4bit is
    component not_gate is
        port( a : in std_logic;
              anot : out std_logic
             );
    end component not_gate;
```

```
component or_gate_4 is
    port( a0 : in std_logic;
    a1 : in std_logic;
    a2 : in std_logic;
    a3 : in std_logic;
    b : out std_logic
          );
end component or_gate_4;
component and_gate_3 is
    port(a0 : in std_logic;
          a1 : in std_logic;
          a2 : in std_logic;
          b : out std_logic
         );
end component and_gate_3;
signal s0, s1, s2, s3 : std_logic;
signal selnot : std_logic_vector(1 downto 0);
begin
    not_gate0 : not_gate port map (a => sel(0), anot => selnot(0));
    not_gate1 : not_gate port map (a => sel(1), anot => selnot(1));
    and_gate_30 : and_gate_3 port map (a0 => selnot(0), a1 => selnot(1), a2 =>
data(0), b => s0);
    and_gate_31 : and_gate_3 port map (a0 => sel(0), a1 => selnot(1), a2 =>
data(1), b => s1);
```

```
    and_gate_32 : and_gate_3 port map (a0 => selnot(0), a1 => sel(1), a2 =>
data(2), b => s2);

    and_gate_33 : and_gate_3 port map (a0 => sel(0), a1 => sel(1), a2 => data(3), b
=> s3);

    or_gate_40 : or_gate_4 port map (a0 => s0, a1 => s1, a2 => s2, a3 => s3, b =>
output);

end Structural;
```

سیگنال ها در واقع سیم های داخلی یک مدار هستند و هنگامی که یک مقدار نه مربوط به خروجی و نه مربوط به ورودی باشد از سیگنال برای نگهداری آن استفاده می کنیم. در این مدار از سیگنال های s0 تا s3 برای نگهداری نتیجه and های سه ورودی استفاده می کنیم.

4bit Comparator



در شکل بالا مدار مربوط به یک مقایسه کننده 4 بیتی را مشاهده می کنید که دو عدد 4 بیتی در ورودی گرفته و و برحسب مقایسه انجام شده روی آن ها یکی از سه خروجی بزرگتر، مساوی یا کوچکتر را روشن می کند. دقت کنید که تنها پیاده سازی بزرگتر و مساوی کافی است و اگر هیچکدام از آن ها روشن نشد با استفاده از یک گیت nor خروجی مربوط به کوچکتر فعال می شود.

همانطور که می بینید در مدار بالا تعداد نسبتا زیادی گیت استفاده شده از جمله xor_gate, or_gate, and_gate, not_gate, nor_gate با تعداد ورودی های متفاوت پس طبق معمول ابتدا باید در بخش architecture گیت هایی که به آن ها نیاز داریم را به صورت کامپوننت تعریف کنیم و سپس بعد از begin از آن کامپوننت ها نمونه سازی کرده و مدار خود را توصیف کنیم. دقت کنید که تمام توصیف هایی که در این آزمایش انجام شدند به صورت Structural بودند.

```
architecture Structural of comparator_4bit is
    component xor_gate is
        port( a : in std_logic;
              b : in std_logic;
              c : out std_logic
             );
    end component xor_gate;
    component and_gate_4 is
        port( a0 : in std_logic;
              a1 : in std_logic;
              a2 : in std_logic;
              a3 : in std_logic;
              b : out std_logic
             );
    end component and_gate_4;
    component not_gate is
        port( a : in std_logic;
              anot : out std_logic);
    end component not_gate;
```

```
component and_gate_2 is
    port(a : in std_logic;
         b : in std_logic;
         c : out std_logic
         );
end component and_gate_2;
component and_gate_3 is
    port(a0 : in std_logic;
         a1 : in std_logic;
         a2 : in std_logic;
         b : out std_logic
         );
end component and_gate_3;
component and_gate_5 is
    port( a0, a1, a2, a3, a4 : in std_logic;
          b : out std_logic
         );
end component and_gate_5;
component or_gate_4 is
    port( a0 : in std_logic;
          a1 : in std_logic;
          a2 : in std_logic;
          a3 : in std_logic;
          b : out std_logic
         );
end component or_gate_4;
```

```
    component nor_gate is
        port(a, b : in std_logic;
             c : out std_logic);
    end component nor_gate;
    component buff is
        port(a : in std_logic;
             b : out std_logic);
    end component buff;
signal c0, c1, c2, c3 : std_logic;
signal g0, g1, g2, g3 : std_logic;
signal greater, equal : std_logic;
signal bnot : std_logic_vector(3 downto 0);
begin
    xor_gate0 : xor_gate port map (a => a(0), b => b(0), c => c0);
    xor_gate1 : xor_gate port map (a => a(1), b => b(1), c => c1);
    xor_gate2 : xor_gate port map (a => a(2), b => b(2), c => c2);
    xor_gate3 : xor_gate port map (a => a(3), b => b(3), c => c3);
    and_gate_40 : and_gate_4 port map (a0 => c0, a1 => c1, a2 => c2, a3 => c3, b
=> equal);
    buff0 : buff port map (a => equal, b => eq);
    not_gate0 : not_gate port map (a => b(0), anot => bnot(0));
    not_gate1 : not_gate port map (a => b(1), anot => bnot(1));
    not_gate2 : not_gate port map (a => b(2), anot => bnot(2));
    not_gate3 : not_gate port map (a => b(3), anot => bnot(3));
    and_gate_20 : and_gate_2 port map (a => a(3), b => bnot(3), c => g3);
```

```
    and_gate_30 : and_gate_3 port map (a0 => a(2), a1 => bnot(2), a2 => c3, b =>
g2);

    and_gate_41 : and_gate_4 port map (a0 => a(1), a1 => bnot(1), a2 => c3, a3 =>
c2, b => g1);

    and_gate_50 : and_gate_5 port map (a0 => a(0), a1 => bnot(0), a2 => c3, a3 =>
c2, a4 => c1, b => g0);

    or_gate_40 : or_gate_4 port map (a0 => g0, a1 => g1, a2 => g2, a3 => g3, b =>
greater);

    buff1 : buff port map (a => greater, b => gr);

    nor_gate0 : nor_gate port map (a => greater, b => equal, c => lt);

end Structural;
```

فایل پروژه همراه گزارش آپلود شده است.