آزمایشگاه طراحی سیستم‌های دیجیتال

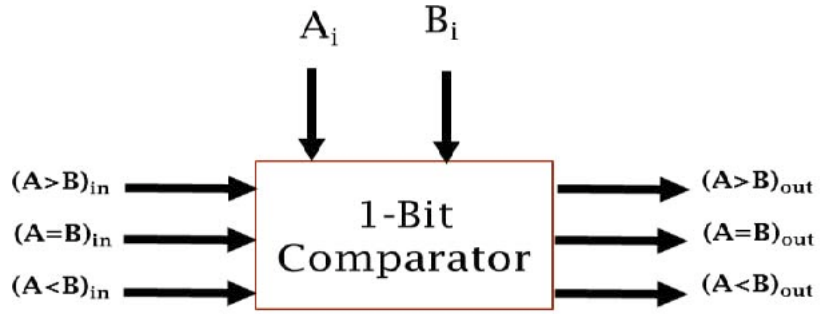
آزمایش شماره ۳: توصیف جریان داده با مقایسه‌کننده

اعضای گروه:

روژین تقی‌زادگان ۴۰۱۱۰۵۷۷۵
رادین شاه‌دایی ۴۰۱۱۰۶۰۹۶
باربد شهرآبادی ۴۰۱۱۰۶۱۲۵

استاد: دکتر انصاری

در بخش اول باید به کمک طراحی سلسله‌مراتبی، یک مقایسه‌کننده چهاربیتی بسازیم. برای این کار ابتدا یک cascadable one-bit comparator طراحی می‌کنیم و سپس با اتصال چهارتا از این مقایسه‌کننده‌ها، یک مقایسه‌کننده چهاربیتی می‌سازیم.

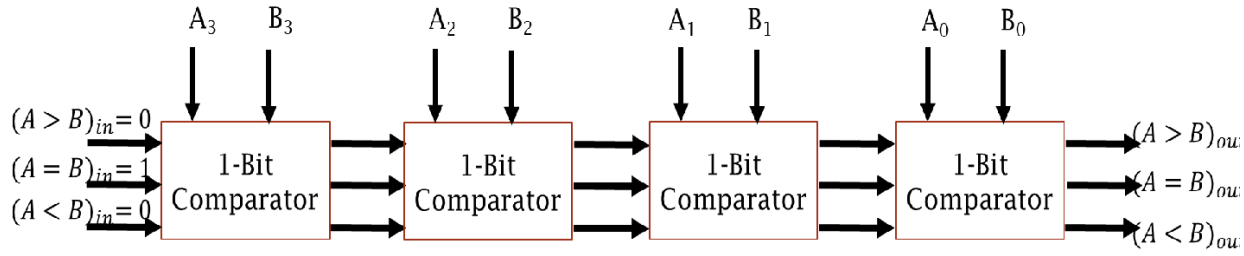


schematic 1-bit comparator

```
module OneBitComp   (input g, e, l, a, b,
                     output g_out, e_out, l_out);
        assign e_out = e & (a == b);
        assign g_out = g | (e & (a > b));
        assign l_out = l | (e & (a < b));
endmodule
```

Cascadable one-bit comparator

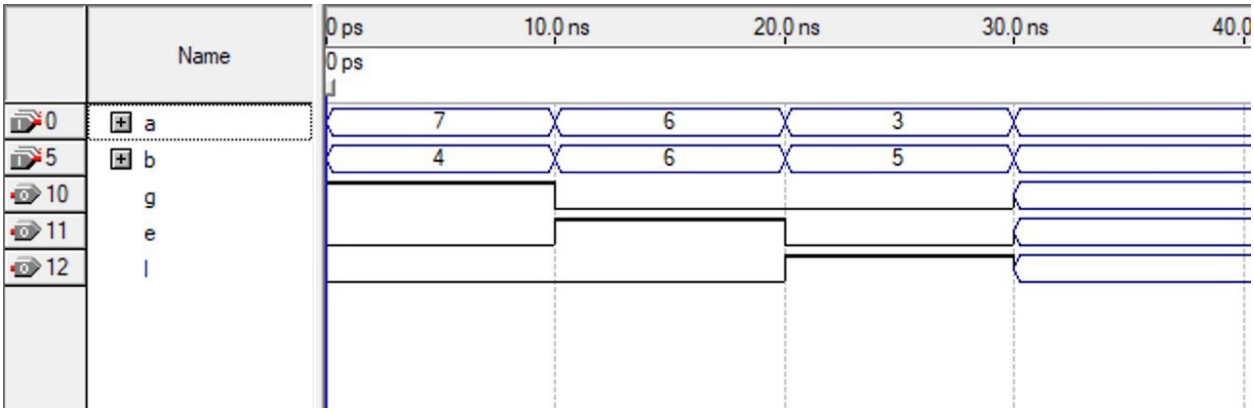
می‌دانیم که در مقایسه اعداد، ارقام پرارزش (سمت چپ) از اهمیت بیشتری برخوردار هستند بنابراین در مقایسه‌کننده یک بیتی ورودی‌هایی که دریافت می‌کنیم، .ضعیت مقایسه اعداد را از قبل (در ارقام پرارزش‌تر) نشان می‌دهند. (ورودی‌های g، e و l). اگر بزرگتر یا کوچکتر بودن عددی در ارقام پرارزش آشکار شده بود، این نتیجه به خروجی منتقل می‌شود و در صورتی که برابری اعداد از قسمت قبل نتیجه شده باشد، حاصل مقایسه برابر مقایسه دو بیت فعلی می‌باشد. در نهایت از کنار هم قرار گرفتن چهار cascadable one-bit comparator می‌توان یک مقایسه‌کننده چهاربیتی ساخت.



schematic 4-bit comparator

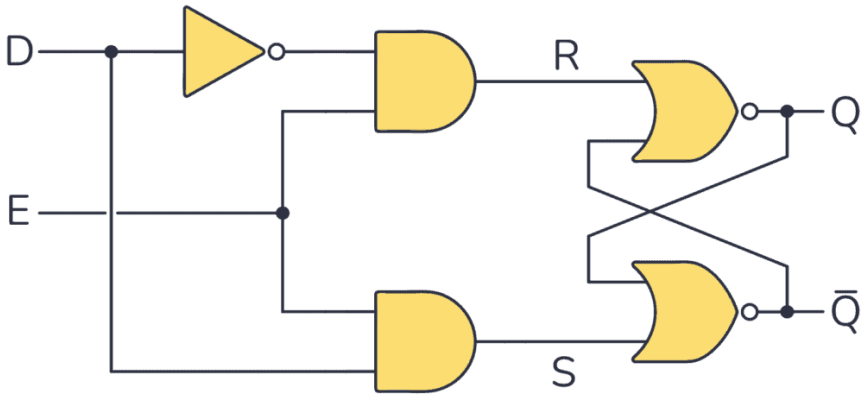
```
module FourBitComp  (input[3:0] a, input[3:0] b,
                     output g, e, l);
        wire c [8:0];
        OneBitComp c3(1'b0, 1'b1, 1'b0, a[3], b[3], c[8], c[7], c[6]);
        OneBitComp c2(c[8], c[7], c[6], a[2], b[2], c[5], c[4], c[3]);
        OneBitComp c1(c[5], c[4], c[3], a[1], b[1], c[2], c[1], c[0]);
        OneBitComp c0(c[2], c[1], c[0], a[0], b[0], g, e, l);
endmodule
```

four-bit comparator



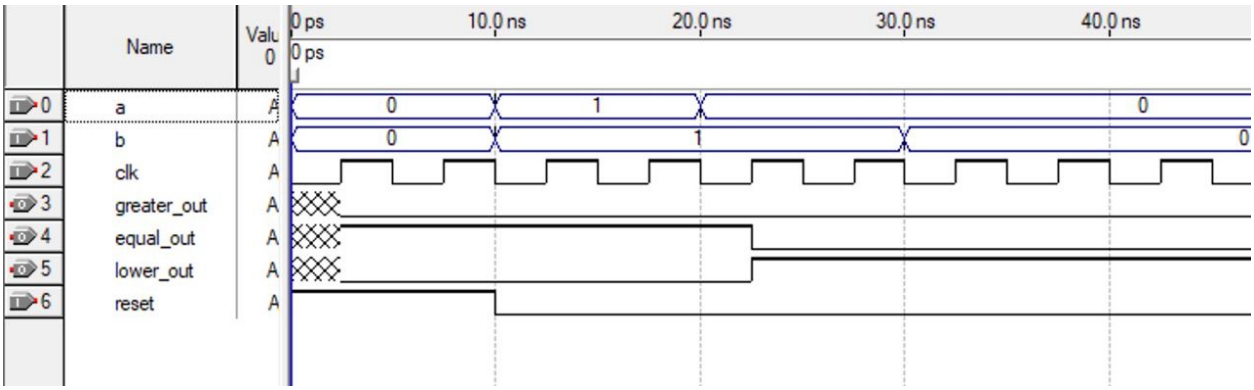
four-bit comparator signals

در بخش دوم، فرض می‌کنیم یک ماژول بیت‌های دو عدد را از بیت پرارزش به کم‌ارزش دریافت می‌کند و خروجی مورد نظر را تولید می‌کند. برای آن که ساختار ترتیبی داشته باشیم، از D-latch استفاده می‌کنیم که ساختار آن به شکل زیر است:



برای هر کدام از خروجی‌های g_out، l_out و e_out به یک latch احتیاج داریم که خروجی‌ها را مستقیما تولید می‌کند.

```
1  module SerialComp (input clk , input reset , input a , input b,
2  output greater_out , output equal_out , output lower_out);
3
4      wire greater_in , equal_in , lower_in;
5      wire not_greater_out , not_equal_out , not_lower_out;
6      wire tmp_not_greater_out, tmp_not_equal_out, tmp_not_lower_out;
7      wire tmp_greater_out, tmp_equal_out, tmp_lower_out;
8
9      assign equal_in = reset | (((a == b) & equal_out) & (~reset));
10     assign greater_in = (~reset) & (((a > b) & equal_out) | greater_out);
11     assign lower_in = (~reset) & (((a < b) & equal_out) | lower_out);
12
13     assign tmp_equal_out = ~(tmp_not_equal_out & ~(~clk & equal_in));
14     assign tmp_not_equal_out = ~(tmp_equal_out & ~(~clk & ~(equal_in)));
15
16     assign tmp_greater_out = ~(tmp_not_greater_out & ~(~clk & greater_in));
17     assign tmp_not_greater_out = ~(tmp_greater_out & ~(~clk & ~(greater_in)));
18
19     assign tmp_lower_out = ~(tmp_not_lower_out & ~(~clk & lower_in));
20     assign tmp_not_lower_out = ~(tmp_lower_out & ~(~clk & ~(lower_in)));
21
22     assign greater_out = clk ? tmp_greater_out : greater_out;
23     assign lower_out = clk ? tmp_lower_out : lower_out;
24     assign equal_out = clk ? tmp_equal_out : equal_out;
25 endmodule
```



one-bit comparator signals