به نام خداوند بخشنده مهربان

طراحی سیستم های دیجیتال

دکتر اجلالی

نیکا قادری
بهار ۱۴۰۳

تاریخ گزارش: ۲۹ اسفند ۱۴۰۲ | **طراحی جمع کننده انتشاری ۸ بیتی** | تمرین اول

---

۱ مقدمه

در این تمرین، هدف بر آن است که با استفاده از جمع کننده های نیم افزا و تمام افزا با تاخیری مشخص، یک جمع کننده ۸ بیتی انتشاری بسازیم و تاخیر کل مدار را تحلیل کنیم. در ابتدا به طور اختصار توضیحاتی در مورد نحوه کار جمع کننده داده می‌شود و سپس در مورد تاخیر آن صحبت می‌شود.

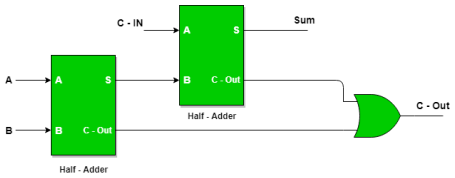
۲ طراحی کلی و ماژول ها

ابتدا طبق معماری bottom-up جمع کننده نیم افزا را می سازیم. با توجه به این که خواسته شده تاخیر بیت جمع، ۳ واحد زمانی و تاخیر بیت نقلی، دو واحد زمانی باشد، از $\#delay$ برای نمایش این تاخیرها استفاده می کنیم. کد درون بلوک always تعریف شده و از دیتا تایپ reg استفاده کرده ایم چرا که هدف، طراحی یک نیم افزا به صورت رفتاری می باشد.

```
1   module HalfAdder(s, c, a, b);
2    input a, b;
3    output s, c;
4    reg s, c;
5
6    always @ (a or b)
7       #3 s = a ^ b;
8
9     always @ (a or b)
10      #2 c = a & b;
11
12  endmodule
```

شکل ۱: جمع کننده نیم افزا

در مرحله بعد، با استفاده از دو نیم افزا، یک جمع کننده تمام افزا می سازیم. منطق کد بر اساس تصویر زیر می باشد:



حال چون قرار است این ماژول به صورت ساختاری توصیف شود، با تعریف wire و اتصال آن ها به هم و instance گرفتن از ماژول نیم افزا، طراحی کامل می شود:

```
1   module FullAdder(s, cout, a, b, cin);
2    input a, b, cin;
3    output cout, s;
4    wire s0, c0, c1;
5
6    HalfAdder ha0(s0, c0, a, b);
7    HalfAdder ha1(s, c1, cin, s0);
8
9    assign cout = c0 | c1;
10  endmodule
```

شکل ۲: جمع کننده تمام افزا

در آخر نیز، در ماژول top ۸ بار از جمع کننده تمام افزا instance می گیریم و بیت نقلی خروجی هر یک را به بیت نقلی ورودی بعدی متصل می‌کنیم (به جز اولی و آخری) تا ساختار آبشاری جمع کننده انتشاری به دست آید.

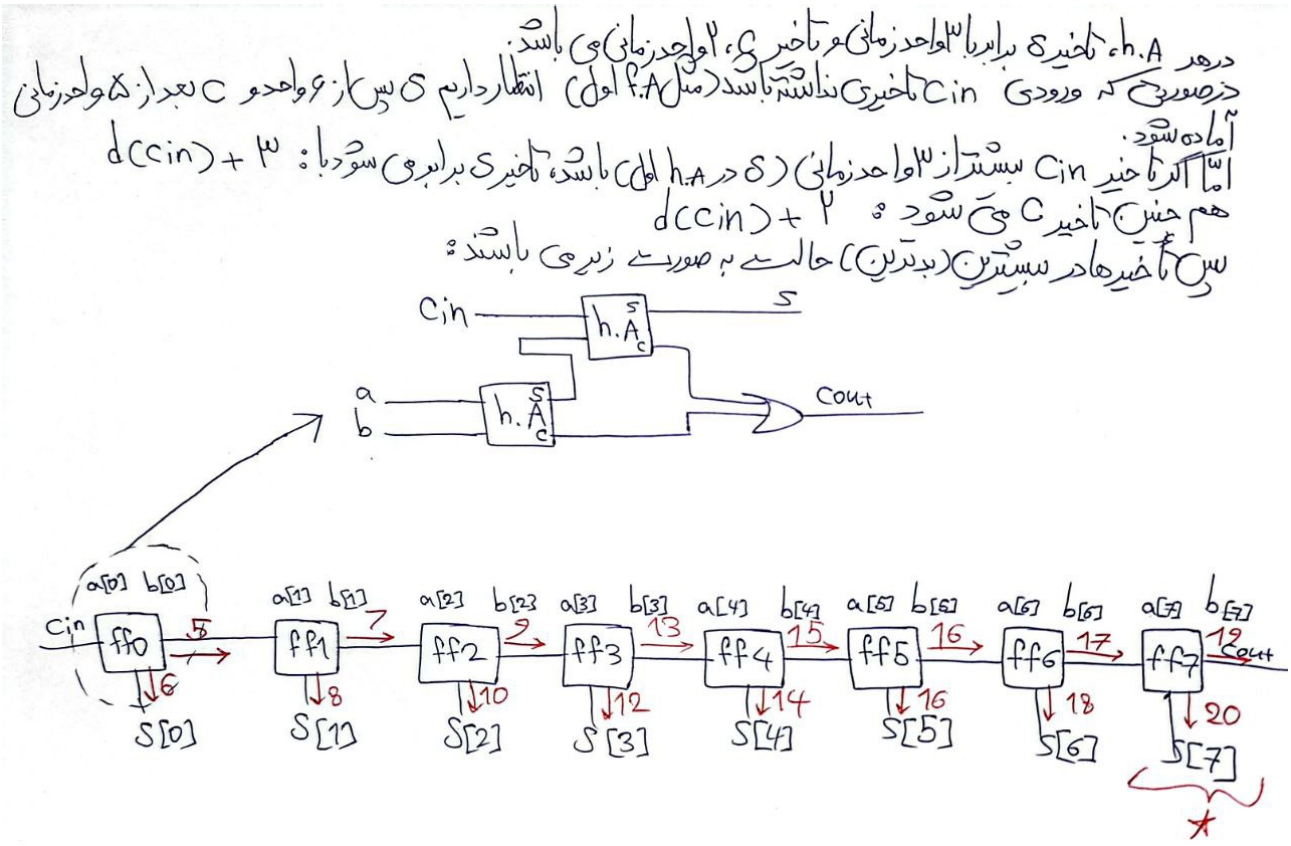
```
module adder(input [7:0] a,b, input cin, output [7:0] s, output cout);
 wire c0, c1, c2, c3, c4, c5, c6;

 FullAdder fa0(s[0], c0, a[0], b[0], cin);  // s = 06, c = 05
 FullAdder fa1(s[1], c1, a[1], b[1], c0);   // s = 08, c = 07
 FullAdder fa2(s[2], c2, a[2], b[2], c1);   // s = 10, c = 09
 FullAdder fa3(s[3], c3, a[3], b[3], c2);   // s = 12, c = 11
 FullAdder fa4(s[4], c4, a[4], b[4], c3);   // s = 14, c = 13
 FullAdder fa5(s[5], c5, a[5], b[5], c4);   // s = 16, c = 15
 FullAdder fa6(s[6], c6, a[6], b[6], c5);   // s = 18, c = 17
 FullAdder fa7(s[7], cout , a[7], b[7], c6);   // s = 20, c = 19

endmodule
```

شکل ۳: جمع کننده انتشاری

۳ محاسبه تأخیر

۱.۳ بخش تئوری

با توجه به تحلیل پایین بیشترین تاخیر برای بیت های مجموع و بیت نقلی به ترتیب ۲۰ و ۱۹ بیت نقلی می باشد. توجه کنید که تاخیر عملی که با استفاده از monitor کردن خروجی ها مشاهده می شود بسته به مقادیر ورودی و مقادیر قبلی گیت ها و سیم ها، ممکن است کمتر از این مقدار باشد.

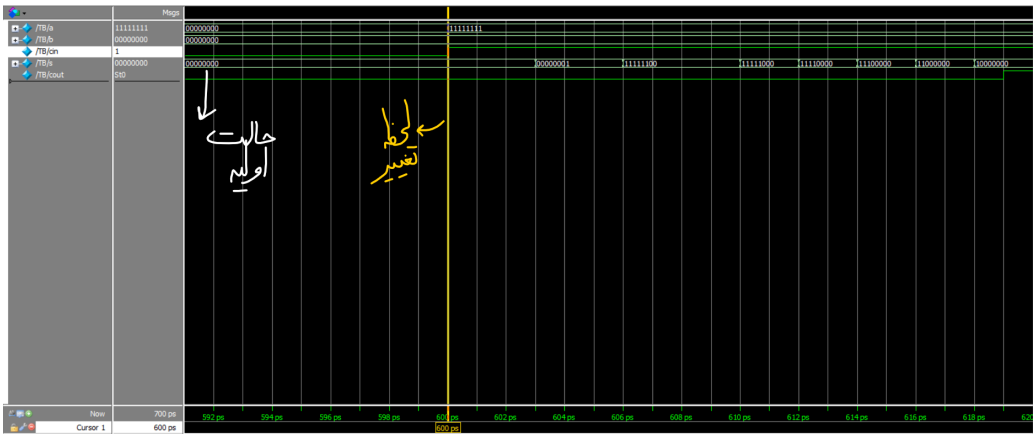


شکل ۴: تأخیرها در یک جمع کننده انتشاری

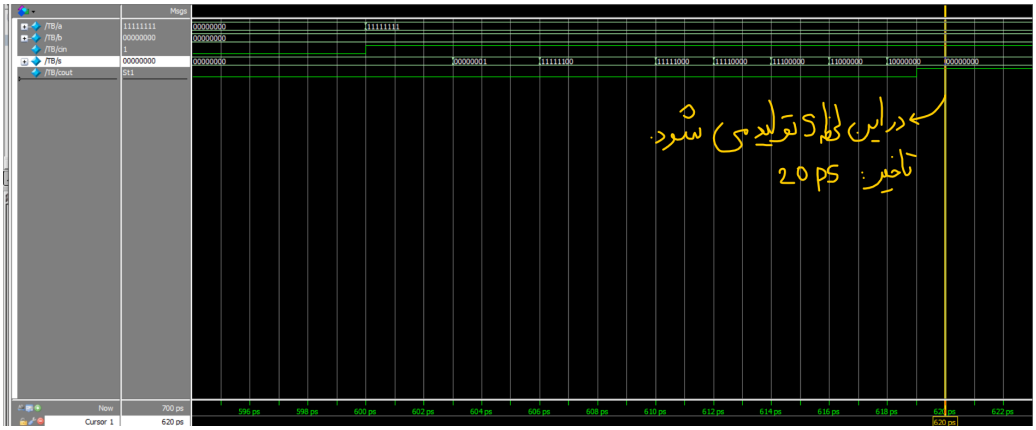
۲.۳ بخش عملی

ابتدا باید ورودی مناسب به مدار داده شود. با توجه به این که می خواهیم به بیشترین تأخیر ممکن برسیم، باید ورودی را طوری تعیین کنیم که بیت های نقلی یک شوند تا بتوانند بر روی خروجی تاثیر مستقیم داشته باشند. پس ابتدا مدار را به حالت $a = ۰, b = ۰, cin = ۰$ می بریم که a و b و cin سیگنال های ورودی هستند و سپس آن را به حالت $a = ۲۵۵, b = ۰, cin = ۱$ می بریم.
علت این کار به این دلیل می باشد که برای عملکرد درست، هر بیت وابسته به بیت نقلی قبل از خود می باشد و تازمانی که این بیت به صورت زنجیروار تولید نشود، به نتیجه درست نمی رسیم.
در ادامه با استفاده از دو روش مختلف این تاخیر را می سنجیم:

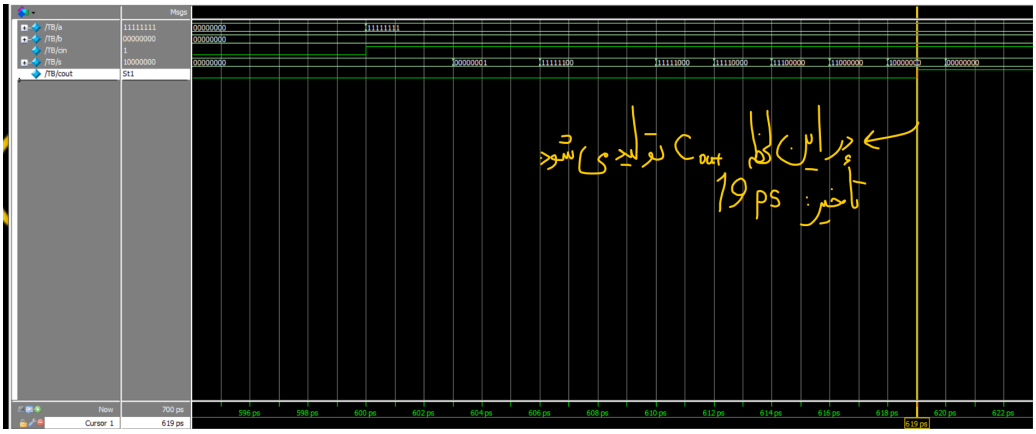
۱.۲.۳ waveform



شکل ۵: حالت آغازین



شکل ۶: حالت پایانی s



شکل ۷: حالت پایانی cout

۲.۲.۳ monitor

در این روش با استفاده از خروجی های خود ترمینال و $time$، خود به صورت دستی تغییرات ورودی و خروجی را رصد می کنیم.

مشاهده می شود که تاخیر – و بازه‌های تغییر آن – منطبق با بخش تئوری می‌باشد.

```
#                  500) a: 00000000      b: 00000000      sum: 00010010   carry: 0
#                  505) a: 00000000      b: 00000000      sum: 00000000   carry: 0
#                  600) a: 11111111      b: 00000000      sum: 00000000   carry: 0
#                  603) a: 11111111      b: 00000000      sum: 00000001   carry: 0
#                  606) a: 11111111      b: 00000000      sum: 11111100   carry: 0
#                  610) a: 11111111      b: 00000000      sum: 11111000   carry: 0
#                  612) a: 11111111      b: 00000000      sum: 11110000   carry: 0
#                  614) a: 11111111      b: 00000000      sum: 11100000   carry: 0
#                  616) a: 11111111      b: 00000000      sum: 11000000   carry: 0
#                  618) a: 11111111      b: 00000000      sum: 10000000   carry: 0
#                  619) a: 11111111      b: 00000000      sum: 10000000   carry: 1
#                  620) a: 11111111      b: 00000000      sum: 00000000   carry: 1
# Break in Module TB at D:/verilog/HW1/TB.v line 20
```

۴ تست بنچ

برای بررسی عملکرد درست ماژول‌ها، فایلی تحت عنوان تست بنچ می نویسیم که با وارد کردن ورودی‌های مختلف، تغییرات خروجی‌ها و زمان آن‌ها را در صفحه Transcript ثبت می کند. محتویات این فایل به این صورت می باشد:

```
module TB;
 reg [7:0] a;
 reg [7:0] b;
 reg cin;
 wire [7:0] s;
 wire cout;

 adder ebains(a, b, cin, s, cout);

 initial begin
    cin = 8'd0;
    a = 8'd0;
    b = 8'd0;
    #100 a = 8'd5; b = 8'd6;
    #100 b = 8'd3;
    #100 a = 8'd9;
    #100 b = 8'd9;
    #100 a = 8'd0; b = 8'd0;
    #100 a = 8'd255; b = 8'd0; cin = 8'd1;
    #100 $stop;
 end

 initial
    $monitor($time, ") a: %b\t b: %b\t sum: %b\t carry: %b", a, b, s, cout);

 initial begin
    $display("Here is the result of our counter");
    $display("---------------------------------");
 end

endmodule
```

پس از شبیه سازی کد، نتیجه به این صورت قابل مشاهده است:

```
# Here is the result of our counter
# ---------------------------------
#                    0) a: 00000000      b: 00000000      sum: xxxxxxxx   carry: x
#                    5) a: 00000000      b: 00000000      sum: xxxxxx0x   carry: 0
#                    6) a: 00000000      b: 00000000      sum: 00000000   carry: 0
#                  100) a: 00000101      b: 00000110      sum: 00000000   carry: 0
#                  105) a: 00000101      b: 00000110      sum: 00001000   carry: 0
#                  106) a: 00000101      b: 00000110      sum: 00001011   carry: 0
#                  200) a: 00000101      b: 00000011      sum: 00001011   carry: 0
#                  205) a: 00000101      b: 00000011      sum: 00000001   carry: 0
#                  206) a: 00000101      b: 00000011      sum: 00000000   carry: 0
#                  208) a: 00000101      b: 00000011      sum: 00001000   carry: 0
#                  300) a: 00001001      b: 00000011      sum: 00001000   carry: 0
#                  306) a: 00001001      b: 00000011      sum: 00001100   carry: 0
#                  400) a: 00001001      b: 00001001      sum: 00001100   carry: 0
#                  405) a: 00001001      b: 00001001      sum: 00011100   carry: 0
#                  406) a: 00001001      b: 00001001      sum: 00010110   carry: 0
#                  408) a: 00001001      b: 00001001      sum: 00010010   carry: 0
#                  500) a: 00000000      b: 00000000      sum: 00010010   carry: 0
#                  505) a: 00000000      b: 00000000      sum: 00000000   carry: 0
#                  600) a: 11111111      b: 00000000      sum: 00000000   carry: 0
#                  603) a: 11111111      b: 00000000      sum: 00000001   carry: 0
#                  606) a: 11111111      b: 00000000      sum: 11111100   carry: 0
#                  610) a: 11111111      b: 00000000      sum: 11111000   carry: 0
#                  612) a: 11111111      b: 00000000      sum: 11110000   carry: 0
#                  614) a: 11111111      b: 00000000      sum: 11100000   carry: 0
#                  616) a: 11111111      b: 00000000      sum: 11000000   carry: 0
#                  618) a: 11111111      b: 00000000      sum: 10000000   carry: 0
#                  619) a: 11111111      b: 00000000      sum: 10000000   carry: 1
#                  620) a: 11111111      b: 00000000      sum: 00000000   carry: 1
```

۵ نتیجه گیری

در این تمرین، با استفاده از ماژول‌های کوچک‌تر، یک جمع کننده انتشاری ۸ بیتی ساختیم و مقدار تأخیر آن را ابتدا به صورت تئوری تخمین زدیم و سپس با استفاده از روش‌های عملی در محیط نرم افزار modelsim سنجیدیم. در انتها با استفاده از یک فایل تست بنچ، عملکردی کلی از جمع کننده را نمایش دادیم.