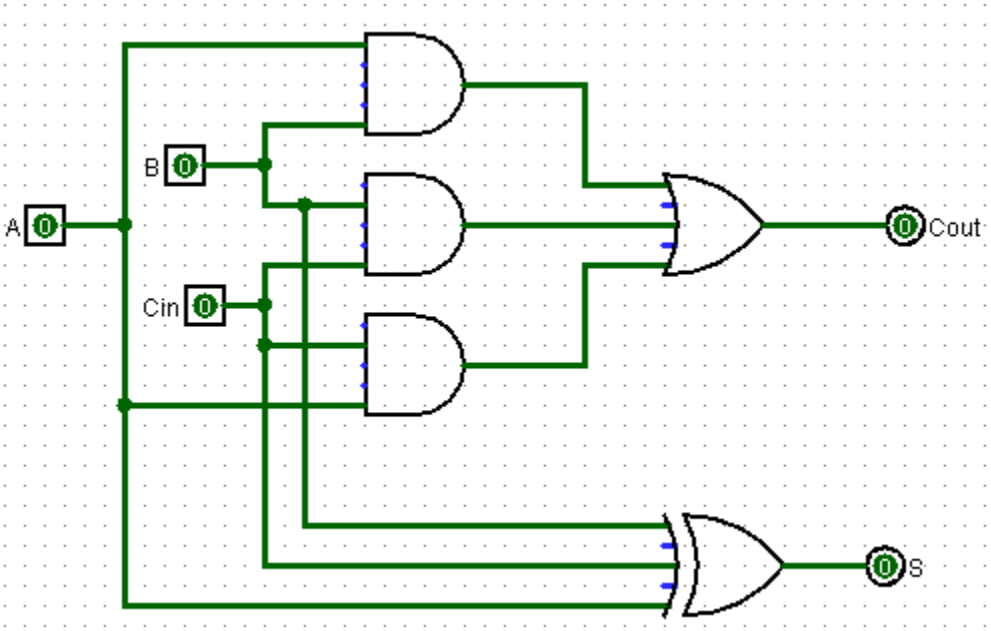
**علی نظری** **به نام خدا** **تمرین سری سوم درس DSD**

قبل از شروع به پاسخ دهی باید سعی کنیم تاخیر مدار را بدون عوض کردن نوع جمع‌کننده کم کنیم. برای این کار طبق راهنمایی صورت سوال، باید ساختار خود full adder را به نوعی عوض کنیم که از xor کمتری استفاده شود. پس طراحی پیشنهادی به شکل زیر می‌شود:



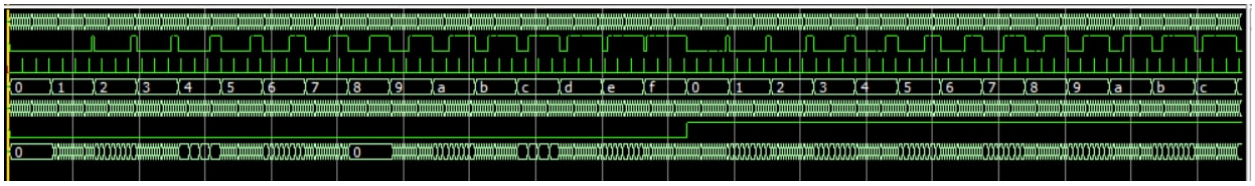

الف)برای طراحی مدار با توجه توضیحات و کدی که استاد سر کلاس زدن، می‌توان به طراحی مدار پرداخت که اینجا فرض شده مدار ۴ بیتی است و یک ماژول برای جمع کننده یک بیتی داریم که به شکل ساختاری تعریف شده و سپس با کنار هم قرار دادن ۴ تا از این واحد ها، جمع کننده ۴ بیتی ساخته ایم و cout هر کدام از واحدها را به cin واحد بعدی داده‌ایم. فایل‌های این بخش در پوشه "بخش الف و ب" موجود است.

ب) برای این بخش هم چون جمع کننده ها ۴ بیتی هستند، در کل ۵۱۲ حالت داریم و cin را هم در نظر بگیریم، ۱۰۲۴ حالت در کل می‌شود که دستی همه حالت‌ها را نمی‌توان زد. پس از حلقه for استفاده می‌کنیم تا همه ورودی‌های ممکن را به مدار بدهیم و از صحت عملکرد مدار مطمئن شویم. فایل این تست بنچ هم در پوشه "بخش الف و ب" موجود است.

همانطور که مشاهده می‌شود مدار به درستی کامپایل می‌شود:

```
vlog -work work -vopt -stats=none {D:/Codes/Verilog Modelsim/DSD_HW3/adderFoutBit.v}
Model Technology ModelSim SE-64 vlog 2020.4 Compiler 2020.10 Oct 13 2020
-- Compiling module fullAdder4B

Top level modules:
        fullAdder4B

vlog -work work -vopt -stats=none {D:/Codes/Verilog Modelsim/DSD_HW3/testbench_4B.v}
Model Technology ModelSim SE-64 vlog 2020.4 Compiler 2020.10 Oct 13 2020
-- Compiling module testbench_4B

Top level modules:
        testbench_4B

vlog -work work -vopt -stats=none {D:/Codes/Verilog Modelsim/DSD_HW3/adder_st.v}
Model Technology ModelSim SE-64 vlog 2020.4 Compiler 2020.10 Oct 13 2020
-- Compiling module full_adder_st

Top level modules:
        full_adder_st
```


و به درستی سنتز هم می‌شود و نتایج درست است:



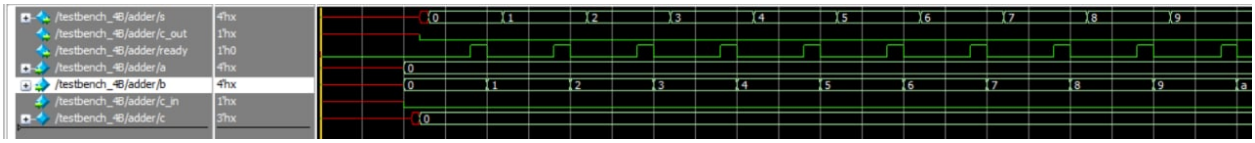
```
#               24 a=0000, b=1011, cin=0, s=1011, cout=0
#               26 a=0000, b=1100, cin=0, s=1100, cout=0
#               28 a=0000, b=1101, cin=0, s=1101, cout=0
#               30 a=0000, b=1110, cin=0, s=1110, cout=0
#               32 a=0000, b=1111, cin=0, s=1111, cout=0
#               34 a=0001, b=0000, cin=0, s=0001, cout=0
#               36 a=0001, b=0001, cin=0, s=0010, cout=0
#               38 a=0001, b=0010, cin=0, s=0011, cout=0
#               40 a=0001, b=0011, cin=0, s=0100, cout=0
#               42 a=0001, b=0100, cin=0, s=0101, cout=0
#               44 a=0001, b=0101, cin=0, s=0110, cout=0
```

پ) در این بخش طبق مداری که برای full adder در فایل سوالات موجود است، برای رسیدن به S نیاز است که از ۲ لایه xor بگذریم. پس تاخیر رسیدن به S در هر لایه ۸ واحد زمانی است. برای Cout باید از یک لایه xor و یک لایه and و یک لایه or بگذریم که باز هم تاخیر آن ۸ واحد زمانی است. پس به عنوان مثال خروجی‌های S و Cout این مدار بعد از ۸ واحد زمانی تولید می‌شوند و چون بیت دوم نیاز دارد که Cin خود را از Cout قبلی داشته باشد، نیاز دارد که صبر کند تا ۸ واحد زمانی

این بیت اول بگذرد. پس S و Cout بیت دوم، بعد از ۸ + ۸ واحد زمانی تولید می‌شود و با همین استدلال، خروجی‌های S و Cout بیت nام بعد از 8n واحد زمانی آماده می‌شوند.
از آنجایی تاخیر 8n واحد زمانی است، اگر جمع کننده را ۴ بیتی در نظر بگیریم، و واحد زمانی را هم یک نانو ثانیه در نظر بگیریم، سیگنال ready با توجه به عوض شدن ورودی‌ها، صفر می‌شود و ۳۲ نانو ثانیه بعد باید یک شود که نشان دهد خروجی آماده است. که این مورد در همان فایل‌های پوشه "بخش الف و ب" رعایت شده و به عنوان کد این بخش پ هم قابل استفاده است. ولی باید تاخیری برای گیت‌های and و or و xor در نظر بگیریم تا درست کار کند و همچنین تاخیر بین ورودی دادن ها را هم باید بیشتر کنیم تا بعد از آماده شدن نتیجه ورودی بعدی داده شود. برای اینکار، کد موجود در پوشه "بخش پ" را زدم. پس هم تاخیر گیت ها اعمال شده و هم تاخیر بین ورودی دادن:

```
6   xor #4 g1(w1, a, b);
7   xor #4 g2(s, w1, cin);
8   and #2 g3(w3, w1, cin);
9   and #2 g4(w2, a, b);
10  or #2 g5(cout, w2, w3);
```

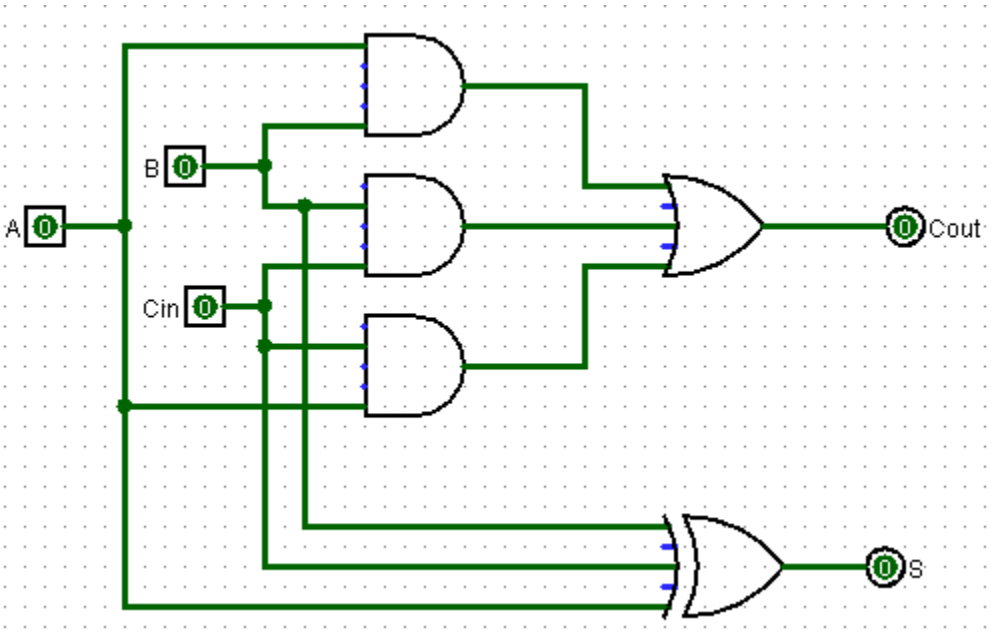
موج خروجی ها هم به شکل زیر است:




و همانطور که مشاهده می‌شود، هر موقع که سیگنال ready لبه پایین رونده‌اش می‌رسد، خروجی آماده است و می‌توان به خروجی استناد کرد و هر ۳۲ نانو ثانیه یک بار هم این اتفاق می‌افتد. خروجی های گفته شده هم در عکس زیر مشخص است که درست محاسبه شده‌اند:

```
13996 a=1011, b=0100, cin=1, s=1100, cout=0
2920 a=0101, b=1010, cin=0, s=1111, cout=0
```

ت)



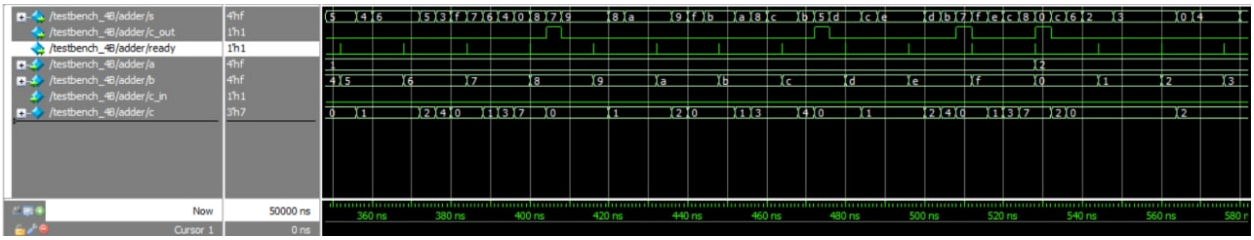

طراحی مداری این به شکل بالا است که باعث کاهش تاخیر مدار می‌شود و همانطور که گفته شده بود، تاخیر را از 8n واحد زمانی به 4n واحد زمانی می‌رساند.
پس بخش جمع کننده تک بیتی به شکل زیر می شود:

```
and #2 g1(w1, a, b);
and #2 g2(w2, b, cin);
and #2 g3(w3, a, cin);
xor #4 g4(s, a, b, cin);
or #2 g5(cout, w1, w2, w3);
```

و سیگنال ready به شکل زیر می‌شود:

```
15      always @(a or b or c_in)
16      begin
17              ready = 1'b0;
18              #16 ready = 1'b1;
19      end
20
21      initial
22              ready = 1'b0;
```

و نتایج هم درست و به شکل زیر است:




```
#                7780 a=1110, b=0101, cin=1, s=0010, cout=1
#                7784 a=1110, b=0101, cin=1, s=0000, cout=1
#                7788 a=1110, b=0101, cin=1, s=0100, cout=1
#                7792 a=1110, b=0110, cin=1, s=0100, cout=1
#                7796 a=1110, b=0110, cin=1, s=0111, cout=1
#                7800 a=1110, b=0110, cin=1, s=0101, cout=1
#                7808 a=1110, b=0111, cin=1, s=0101, cout=1
#                7812 a=1110, b=0111, cin=1, s=0100, cout=1
#                7816 a=1110, b=0111, cin=1, s=0110, cout=1
#                7824 a=1110, b=1000, cin=1, s=0110, cout=1
#                7828 a=1110, b=1000, cin=1, s=1001, cout=1
#                7832 a=1110, b=1000, cin=1, s=1011, cout=1
#                7836 a=1110, b=1000, cin=1, s=1111, cout=1
#                7840 a=1110, b=1001, cin=1, s=0111, cout=1
#                7844 a=1110, b=1001, cin=1, s=0110, cout=1
#                7848 a=1110, b=1001, cin=1, s=0100, cout=1
#                7852 a=1110, b=1001, cin=1, s=0000, cout=1
#                7856 a=1110, b=1010, cin=1, s=1000, cout=1
```

دقت کنید هر مقدار a و b و cin چند بار در کنسول تکرار شده اند و دلیل این است که $monitor به تغییر هر کدام حساس است و چند با تاخیر هر کدام انجام می‌شوند، به ازای تغییر هر کدام دستور $monitor آن عبارت را چاپ میکند و طبیعتا برخی از آن ها هنوز آماده نشده اند و ناقص هستند ولی آخرین موردی که نمایش داده می‌شود برای یک a و b و cin مشخص، مقدار درست است.
کدهای این بخش در پوشه "بخش ت" موجود است.

ث) برای این بخش، اگر جمع کننده را n بیتی در نظر بگیریم، برای آماده شدن S بیت اول، به ۴ واحد زمانی نیاز است و برای cout هم به ۴ واحد زمانی نیاز است. برای بیت دوم، چون باید cout بیت اول آماده شود، هم S و Cout بیت دوم با ۴ واحد زمانی تاخیر شروع به کار میکنند. و خودشان هم هر کدام ۴ واحد زمانی وقت می‌خواهند. پس در کل ۸ واحد زمانی نیاز دارند. به همین ترتیب برای محاسبه بیت k ام، نیاز است که Cout بیت k-1ام آماده شده باشد که 4(k-1) واحد زمانی نیاز دارد و خود بیت kام هم به ۴ واحد زمانی نیاز دارد پس در نهایت خروجی بیت kام پس از 4k واحد زمانی آماده می‌شود. پس برای بیت nام، 4n واحد زمانی نیاز است. پس اگر جمع کننده ما n بیتی باشد، بیت اول بعد از ۴ واحد زمانی آماده است و بیت دوم بعد از 4*2 واحد زمانی و بیت nام بعد از 4n واحد زمانی، پس بعد از پایان واحد زمانی 4nام، باید سیگنال ready را یک کنیم که یعنی خروجی مدار آماده است. پس بعد از دادن ورودی، با 4n واحد زمانی تاخیر این بیت ready یک می‌شود. پس تاخیر این بیت ready هم 4n واحد زمانی است.