تمرین ۲ طراحی سیستم‌های دیجیتال

Alireza Habibzadeh `99109393`

1

الف

کنسول warning

```
-- Compiling module test
** Warning: C:/modeltech64_2020.4/examples/test.v(6): (vlog-2600) [RDGN] -
Redundant digits in numeric literal.
```

علت این هشدار برمی‌گردد به خط `6` عبارت `test = 2'b1111` که در آن یک طول عدد را 2 بیت قرار دادیم، اما هنگام مقداردهی عدد باینری 4 بیتی `15 = 1111` را وارد کردیم.
کامپایلر خودش ۲ بیت اضافه را دور ریخته و مقدار `11` را در عدد ذخیره می‌کند.

برای پاسخ ساده‌تر به سوالات در ادامه، در دستور مانیتور مقدار زمان را هم اضافه کردم.

```
$monitor ("%d, %h, %d, %b", $time, test, test, test);
```

پس خروجی اجرای کد می‌شود:

```
run
#                   0, 0010,    16, 0000000000010000
#                  10, 0003,     3, 0000000000000011
#                  20, 000d,    13, 0000000000001101
#                  30, 0zf0,     Z, 0000zzzz11110000
#                  40, 00XZ,     X, 0000000000x1zz00
```

ب

در زمان ۱۰ دقیقا در حال تغییر است. قبل آن مقدار `16 = 5 + 13` ذخیره شده (چون جا هم دارد چیزی دور ریخته نمی‌شود.) و پس از آن دو بیت اضافه‌ی آن دور ریخته می‌شود و برابر با `11` یا عدد ۳ دسیمال است.

پ

اعشار دور ریخته شده و مقدار ۱۳ را دارد.

ت

عدد `h?f0'12` یعنی ۱۲ بیت دارد که چهار بیت راست ۰ سپس چهار بیت عدد 1111) f) و سپس بیت‌های باقی‌مانده به صورت z یا همان high impedance است. اما ۴ بیت از ۱۶ بیت متغییر باقی می‌ماند که همان ۰ می‌مانند. نمایش باینری عدد `0000zzzz11110000` است ولی در نمایش دسیمال `Z` (پر ارزش‌ترین بیت) نشان داده می‌شود که جالب است.

ث

تنها ۶ بیت سمت راست متغییر را assign می‌کنیم. این ۶ بیت هم صراحتا اعلام کرده‌ایم `x1zz00` . اما نمایش دسیمال مانند قبل به پرارزش‌ترین بیت کار دارد که اینجا `x` است.

2

روش‌های مختلفی برای این کار وجود دارد. برخی از نرم‌افزارها به صورت گرافیکی نیز می‌توانند این فایل را تولید کنند.
اما من پس از اجرای simulation در کنسول transcript نرم‌افزار ModelSim این دستورات را وارد کردم:
(توضیح در کامنت‌های داخل کد)

```
vcd file out.vcd  // Set this file as the vcd output
vcd add -r test.v // Add test.v variables to the wave
run               // Run the simulation
vcd help          // Find out what to do next
# ** UI-Msg: (vsim-4002) Invalid keyword 'help'. Expected 'add', 'checkpoi
# Usage:
#   vcd add <arguments>
#   vcd checkpoint <arguments>
#   vcd comment <arguments>
#   vcd dumpports <arguments>
#   vcd dumpportsall <arguments>
#   vcd dumpportsflush <arguments>
#   vcd dumpportslimit <arguments>
#   vcd dumpportsoff <arguments>
#   vcd dumpportson <arguments>
#   vcd file <arguments>
#   vcd files <arguments>
#   vcd flush <arguments>
#   vcd limit <arguments>
#   vcd off <arguments>
#   vcd on <arguments>
vcd on        // Just to make sure
vcd flush     // Write the file up to now without breaking the simulation
```

3

این کد یک شمارنده‌ی ۴ بیتی را به روش behavioral تعریف می‌کند. `reset` که از نوع **ناهمزمان** است، خروجی و مقدار ذخیره شده را صفر می‌کند (در بالا رفتن کلاک). `load` در صورت فعال بودن ورودی `data` را خوانده و آن را در حافظه و خروجی load می‌کند. (و در این کلاک شمارشی نداریم) و در غیر این دو صورت بسته به مقدار `up_down` یک شمارش به بالا یا پایین خواهیم داشت. (اگر فعال باشد بالا و در غیر این صورت پایین)

خطای نمایش داده شده این است: (هر جا که به `count` مقداردهی شده)

```
** Error: C:\modeltech64_2020.4\examples\myproject\counter.v(8): (vlog-211
Illegal reference to net "count".
** Error: C:\modeltech64_2020.4\examples\myproject\counter.v(10): (vlog-21
Illegal reference to net "count".
** Error: C:\modeltech64_2020.4\examples\myproject\counter.v(12): (vlog-21
Illegal reference to net "count".
** Error: C:\modeltech64_2020.4\examples\myproject\counter.v(14): (vlog-21
Illegal reference to net "count".
```

ایراد کد این است که برای `count` رجیستری تعریف نشده. در wire که نمی‌شود مقدار ذخیره کرد. کد درست در زیر آمده

```
module counter(clk,reset,up_down,load,data,count);
 input clk,reset,load,up_down;
 input [3:0] data;
 output [3:0] count;
 reg [3:0] count;
 always@(posedge clk)
 begin
        if(reset)
                count <= 0;
        else if(load)
                count <= data;
        else if(up_down)
                count <= count + 1;
        else
        count <= count - 1;
 end
endmodule :counter
```

برای تست این ماژول با روش stimulus block کد testbench در زیر آمده است. در این کد سعی شده همه‌ی کارکردهای ماژول مورد ارزیابی قرار گیرد. ابتدا مدار ریست شده و سپس به صورت پایین رونده تعدادی شمارش انجام می‌شود. سپس شمارش به بالارونده تغییر می‌کند. سپس مدار وسط کار ریست شده و سپس یک مقدار در آن load می‌شود. سپس دوباره کمی شماره انجام می‌دهد. موج یا wave خروجی این تست در صفحه‌ی بعد آمده است.

```
module counter_tb();
reg clk,reset,up_down,load;
reg [3:0] data;
wire [3:0] count;
always #5 clk = ~clk;
```

```
 counter UUT (.clk(clk),.reset(reset),.up_down(up_down)
                ,.load(load),.data(data),.count(count));
initial
begin
$monitor("%d", data);
clk = 0;
reset = 0;
up_down = 0;
load = 0;
data = 4'b0;
#20;
reset = 1;
#20;
reset = 0;
#210;
up_down = 1;
#190;
reset = 1;
#10;
reset = 0;
#50;
data = 4'b1010;
load = 1;
#20;
load = 0;
#100;
$stop;

end
endmodule
```

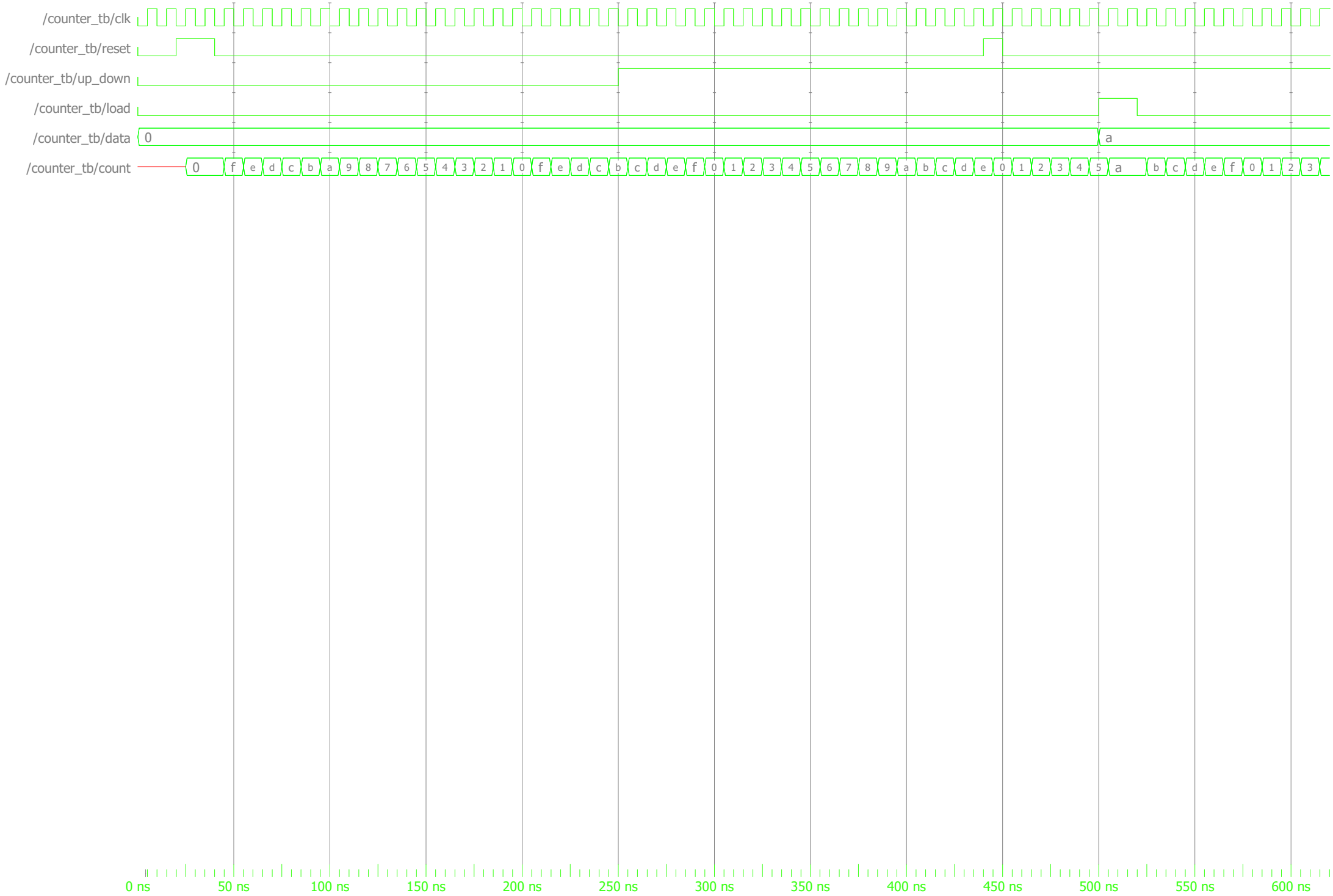
4

A

```
always @(posedge clock)
if(reset)
out <= 0;
else
out <= in;
```

در این مدار reset **ناهمزمان** است. چرا که بلاک `always` تنها با هر لبه‌ی بالارونده‌ی کلاک اجرا می‌شود.

در هر لبه‌ی بالارونده‌ی کلاک مقدار `in` در `out` ظاهر می‌شود و مستقل از تغییر `in` بین دو کلاک، این مقدار در `out` تا کلاک بعدی حفظ می‌شود. پس می‌توان گفت این قطعه یک D-flipflop است که ورودی `in` به `D` وصل شده. ورودی‌های `clock` و `reset` هم به پایه‌های متناظرشان وصل هستند. فلیپ‌فلاپ استفاده شده هم از نوع Active High است و مقدار اولیه‌ی خروجی پس از ریست `0` می‌باشد.


/counter_tb/clk
/counter_tb/reset
/counter_tb/up_down
/counter_tb/load
/counter_tb/data
/counter_tb/count
0
a
0 f e d c b a 9 8 7 6 5 4 3 2 1 0 f e d c b c d e f 0 1 2 3 4 5 6 7 8 9 a b c d e 0 1 2 3 4 5 a b c d e f 0 1 2 3
0 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
450 ns
500 ns
550 ns
600 ns

B

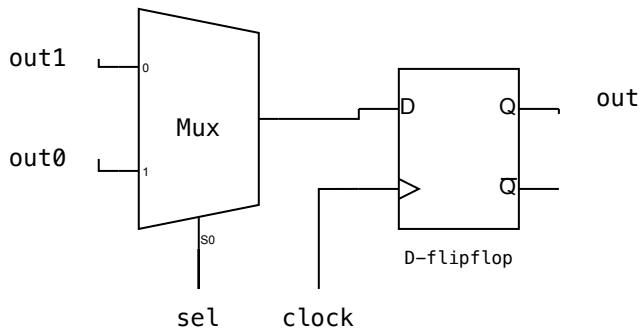
```
always @(posedge clock)
if(reset)
out <= 0;
else if(!clear)
out <= in;
```

در این مدار نیز به همان دلیل قبلی reset **ناهمزمان** است.

مدار اینجا نیز همان D-flipflop قبلی است با این تفاوت که فلیپ‌فلاپ ما دارای پایه‌ی `clear` نیز هست. این پایه از نوع Active Low است یعنی وقتی `clear = 1` باشد هیچ اتفاقی نمی‌افتد و وقتی `clear = 0` باشد مقدار قبلی فلیپ‌فلاپ clear شده و مقدار جدید از `in` خوانده می‌شود. (ریست فلیپ‌فلاپی که اینجا قرار دارد از نوع ناهمزمان است.)

C

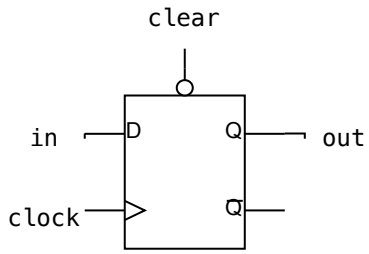
```
always @(posedge clock)
if(sel)
out <= in0;
else
out <= in1;
```



یک مالتی‌پلکسر داریم که `sel` بین `in1` و `in0` (با همین ترتیب یعنی `out(sel = 0) = in1`) انتخاب می‌کند. سپس خروجی تا کلاک بعدی latch می‌شود.

D

```
always @(posedge clock)
if(!clear)
out <= in;
```



مشابه B است فقط `reset` نداریم. تنها اگر `clear = 0` باشد خروجی تازه شده و از روی `in` خوانده می‌شود.

E

```
always @(posedge clock or
posedge
reset)
if(reset)
out <= 0;
else
out <= in;
```

در این مدار reset **همزمان** است. چرا که با لبه‌ی بالارونده‌ی reset نیز بلاک `always` اجرا شده و کاری که برای reset تعریف کردیم را انجام می‌دهد.

مانند بخش A یک D-flipflop است. اما اینجا ریست به صورت همزمان طراحی شده. پایه‌ی `in` به `D` و `out` به `Q` فلیپ‌فلاپ وصل شده و `reset` و `clock` به پایه‌های متناظرشان.