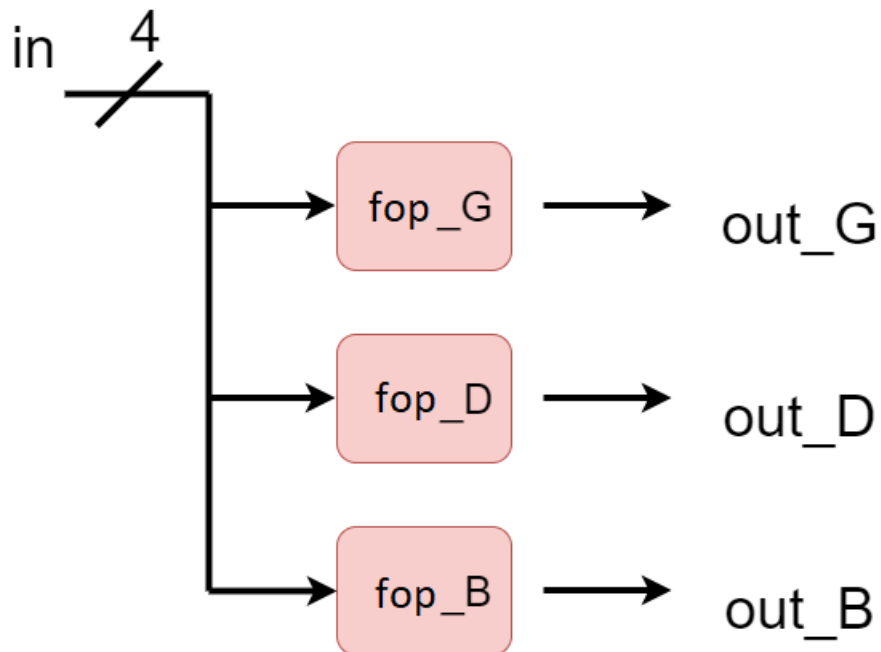


Lab1: Fibonacci or Prime number detector

Output 1 if input is a Fibonacci number or a prime number. Otherwise, output 0.

Input range= [0,15] (Use 4 bit to represent the input)

A) Concept



B) Truth table

Input	in[3]	in[2]	in[1]	in[0]	out
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

C) K-map (a=in[3], b=in[2], c=in[1], d=in[0])

ab \ cd	00	01	11	10
00	1	0	0	1
01	1	1	1	0
11	1	1	0	1
10	1	0	0	0

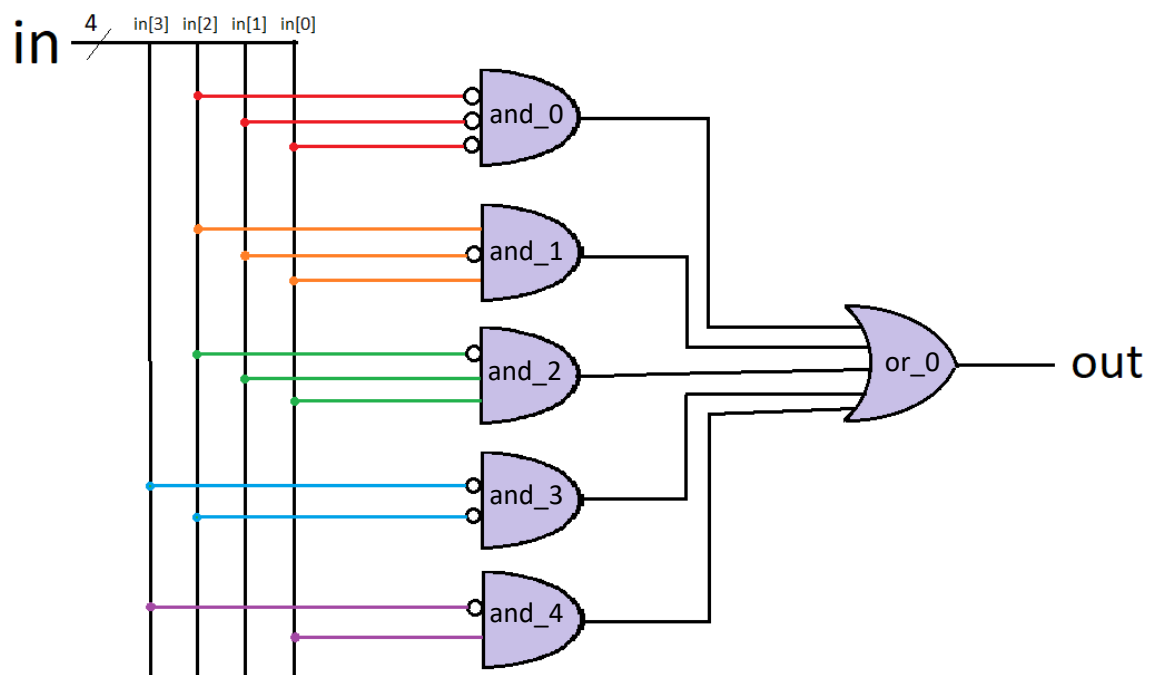
PI: $(\sim b)(\sim c)(\sim d)$, $b(\sim c)d$, $(\sim b)cd$, $(\sim a)(\sim b)$, $(\sim a)d$

EPI: $(\sim b)(\sim c)(\sim d)$, $b(\sim c)d$, $(\sim b)cd$, $(\sim a)(\sim b)$, $(\sim a)d$

out = $(\sim b)(\sim c)(\sim d) + b(\sim c)d + (\sim b)cd + (\sim a)(\sim b) + (\sim a)d$

*不知道這樣的寫法是否會有 hazard 問題? 例如 7->8 沒有共同的圈圈。

D) Circuit design



E) Simulation Result

The screenshot shows the nHucad software interface with the following content:

```

Building instance overlay tables: ..... Done
Generating native compiled code:
worklib.fop.tb.v <0x5a8a79c>
streams: 1, words: 297
worklib.fop.tb.v <0x60ebc6bc>
streams: 1, words: 388
Building instance specific data structures.
Loading native compiled code: ..... Done
Design hierarchy summary:
Instances Unique
Modules: 4 4
Primitives: 14 3
Registers: 3 3
Scalar wires: 2 -
Expanded wires: 4 1
Vectored wires: 2 -
Always blocks: 1 1
Initial blocks: 1 1
Cont. assignments: 1 1
Pseudo assignments: 1 1
Writing initial simulation snapshot: worklib.fop.tb.v
Loading snapshot worklib.fop.tb.v ..... Done
Vord13: Loading libscore_lut141.so
Vord13: Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
time= 5,in=0000,out_G=1,out_D=1,out_B=1
time= 10,in=0001,out_G=1,out_D=1,out_B=1
time= 15,in=0010,out_G=1,out_D=1,out_B=1
time= 20,in=0011,out_G=1,out_D=1,out_B=1
time= 25,in=0100,out_G=0,out_D=0,out_B=0
time= 30,in=0101,out_G=1,out_D=1,out_B=1
time= 35,in=0110,out_G=0,out_D=0,out_B=0
time= 40,in=0111,out_G=1,out_D=1,out_B=1
time= 45,in=1000,out_G=1,out_D=1,out_B=1
time= 50,in=1001,out_G=0,out_D=0,out_B=0
time= 55,in=1010,out_G=0,out_D=0,out_B=0
time= 60,in=1011,out_G=1,out_D=1,out_B=1
time= 65,in=1100,out_G=0,out_D=0,out_B=0
time= 70,in=1101,out_G=1,out_D=1,out_B=1
time= 75,in=1110,out_G=0,out_D=0,out_B=0
time= 80,in=1111,out_G=0,out_D=0,out_B=0
Congratulations!!
Simulation complete via $finish(1) at time 80 NS + 0
./fop_tb.v:21 $finish;
ncsim> exit
(dld0119@ic30 ~/lab1)$
  
```

```

ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
time= 5,in=0000,out_G=1,out_D=1,out_B=1
time= 10,in=0001,out_G=1,out_D=1,out_B=1
time= 15,in=0010,out_G=1,out_D=1,out_B=1
time= 20,in=0011,out_G=1,out_D=1,out_B=1
time= 25,in=0100,out_G=0,out_D=0,out_B=0
time= 30,in=0101,out_G=1,out_D=1,out_B=1
time= 35,in=0110,out_G=0,out_D=0,out_B=0
time= 40,in=0111,out_G=1,out_D=1,out_B=1
time= 45,in=1000,out_G=1,out_D=1,out_B=1
time= 50,in=1001,out_G=0,out_D=0,out_B=0
time= 55,in=1010,out_G=0,out_D=0,out_B=0
time= 60,in=1011,out_G=1,out_D=1,out_B=1
time= 65,in=1100,out_G=0,out_D=0,out_B=0
time= 70,in=1101,out_G=1,out_D=1,out_B=1
time= 75,in=1110,out_G=0,out_D=0,out_B=0
time= 80,in=1111,out_G=0,out_D=0,out_B=0
Congratulations!!
Simulation complete via $finish(1) at time 80 NS + 0
./fop_tb.v:21 $finish;
ncsim> exit
  
```

F) Discussion

課程網頁上的介紹十分詳細，很多我們內心的疑問例如 `wire` 和 `reg`、`i++`和 `i+1` 等在上面都有解答，令我們茅塞頓開。可是我還是有一些疑問：

1. 為什麼 0 被包含在 `fop` 裡面？據我所知 0 不是 `prime number` 也不是 `fibonacci`……

2. 如何考慮 `hazard` 問題？

是只要沒有 `size` 為 1 的圈圈就可以了嗎？還是看兩個連續而且都為 1 的數字有沒有被同一個圈圈起來？像這題的 K-map 中的 7 (0111) 和 8 (1000) 並沒有被圈在同一個圈圈裡面，因此想問助教這樣會出現 `hazard` 嗎？老師有解釋過 `hazard` 的原因，但是我還是不知道如何看出要在哪些地方加圈圈來避免 `hazard`。

3. 在不同的 `description` 要用不同的數字表示法？

為什麼在 Behaviour Description 的 `always` block 中可以直接以數字 0, 1, 2, 3, 5, 7, 8, 11, 13 來表示，而不需分開 4 個 input 來表示（像 Dataflow Description）？

而為什麼 `out` 要寫成 `1'b1`、`1'b0`，`in` 可以寫成 0, 1, 2, 3, 5, 7, 8, 11, 13 不需要寫 `bit` 和表示法？是因為 `verilog` 會自動理解 `in` 嗎？

4. 關於程序碼執行順序的問題

課程網頁上有說硬體語言的程式不是一行行執行的。

但是 `always` block 裡面寫的程式有開始跟結束，是“動作”，所以是從上到下一行行執行的。而其它的描述就像是在描述一個圖片/結構而不是描述從頭到尾的一串事件/動作，所以沒有一行行先後順序的差別。請問這樣的理解對嗎？

5. Testbench

課程網頁上的 `testbench` 寫法和範例程式的 `testbench` 寫法似乎有很大的不同，感覺範例程式的 `testbench` 寫法比較好懂，前者幾乎看不太懂……

```
fop_G hvg
(
    .in(in),
    .out(out_G)
);
```

想問 `fop_G` 後面的 `hvg` 是什麼意思啊？是幫呼叫取名字嗎？

6. Vim 很難用

一般都是在外面寫完再 `copy` 進去 `vim`，但是要修改 `code` 很難，用 `gg`（`cursor` 到第一行）然後 `dG`（清除後面全部）把 `code` 清掉後再重新貼上新的 `code`，有時會出現舊的 `code` 參雜在裡面的情況。