Lab2_110550085

(1)

Half Subtractor

Truth Table:

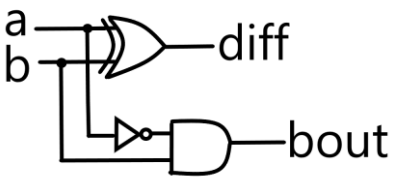
a	b	diff	bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Boolean Expression:

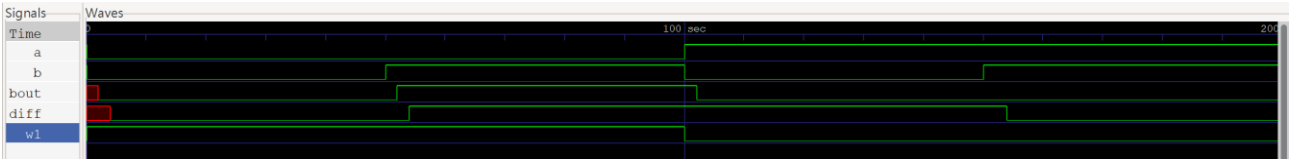
diff=a⊕b

bout=a’b

Logic Diagram:



The simulation:



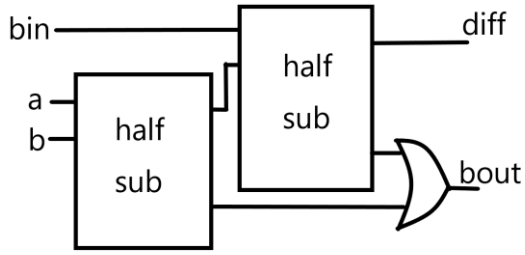
We can know this result is correct by observing the truth table.

The propagation delay for bout is **2** time units, and for diff is **4** time units.

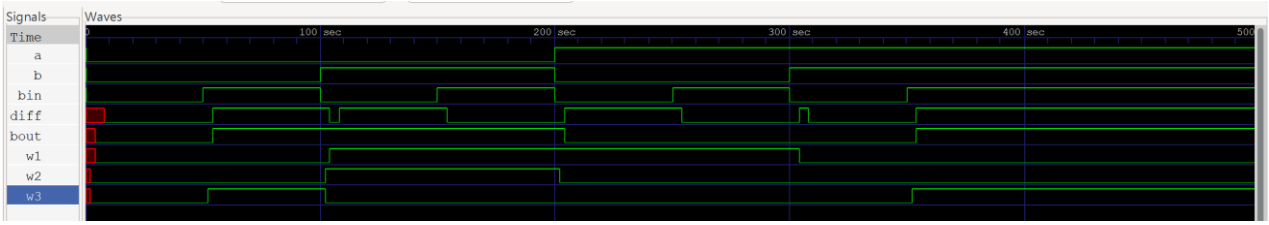
(2)

Full subtractor

Logic Diagram:



The simulation:



We can know this result is correct by observing the truth table.

The truth table:

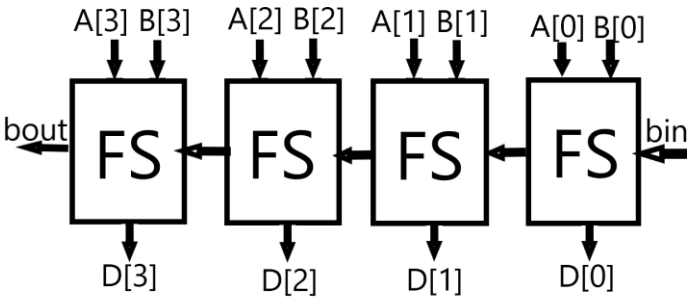
Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Then, the propagation delay for diff is **8** time units, and so is bout.

In the simulation picture, we can see diff and bout have same propagation delay.

(3)

Logic Diagram:



The simulation:

We can know this result is correct by simply calculating it.
(The correct result is the longer results, e.g.: A=0, B=C, bin=1, then D= 3, bout=1.)
(The reason why there are smaller parts is because the changes of lesser significant bits requires more time to change, so results may become things that are not belong to the given information.)
The propagation delay for it should be 8+4+4+4=**20** time units, this can be known by counting delay for the critical path.

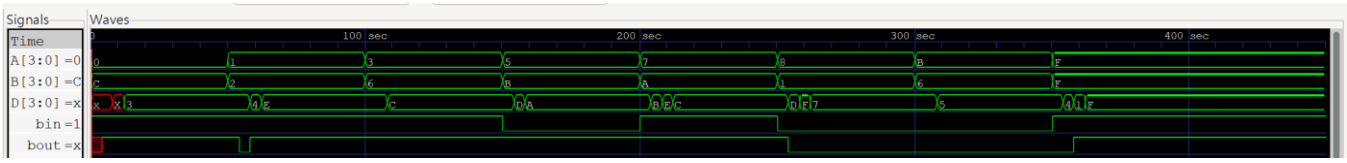
(4)
For a borrow-lookahead subtractor, we define that:
Pi=(Ai⊕Bi)'

Gi=Ai'Bi
Di=(Pi⊕bi)'

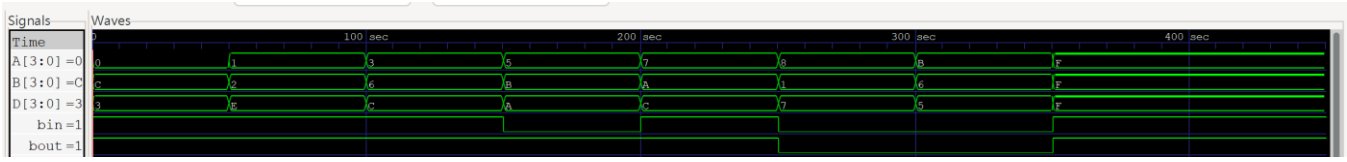
(bi is the borrow, which is different from Bi).
Then,
P0=(A0⊕B0)'

G0=A0'B0
b1=G0+P0bin
b2=G1+P1b1=G1+P1G0+P1P0bin
b3=G2+P2b2=G2+P2G1+P2P1G0+P2P1P0bin
bout=G3+P3b3=G3+P3G2+P3P2G1+P3P2P1G0+P3P2P1P0bin

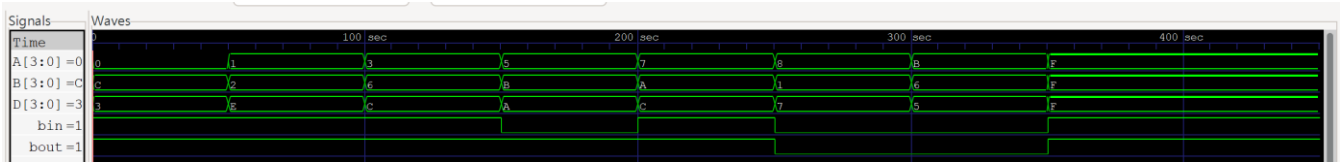
The simulations are:
Gatelevel modeling:



Dataflow modeling:



Behavioral modeling:



We could know the three results are all correct by simply calculating it.

The propagation delay time for gatelevel modeling is **12** time units, which equals to

4(xnor for Pi)+2(“or” to create terms of the SoP form)

+2(“and” to sum all the terms)+4(xnor for Di, i=1, 2, or 3).

This can be observed clearly through the simulation wave.

(5)

Excess-3-to-binary-converter

The Karnaugh map for B[3], B[2], B[1], B[0], and v are:

CD \ AB	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

B[3]

CD \ AB	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

B[2]

CD \ AB	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

B[1]

CD \ AB	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

B[0]

CD \ AB	00	01	11	10
00	0	0	1	0
01	1	1	1	1
11	1	0	0	0
10	1	1	1	1

v

Where C=E[3], D=E[2], A=E[1], B=E[0].

Thus, we could know the Boolean functions are:

B[3]=E[3]E[2]+E[3]E[1]E[0],

B[2]=E[2]'E[0]'+E[3]E[1]'E[0]+E[2]E[1]E[0],

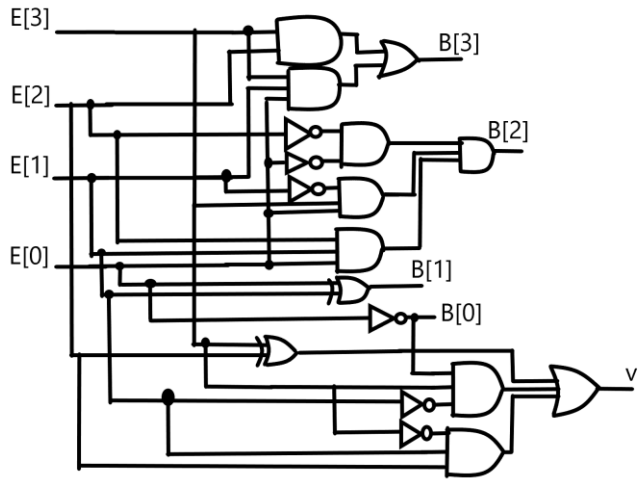
B[1]=E[1]'E[0]+E[1]E[0]'=**E[1]⊕E[0]**,

B[0]=E[0]',

v=E[3]'E[2]+E[3]E[2]'+E[3]E[1]'E[0]'+E[3]'E[1]E[0]

=**E[3]⊕E[2]+E[3]E[1]'E[0]'+E[3]'E[1]E[0]**.

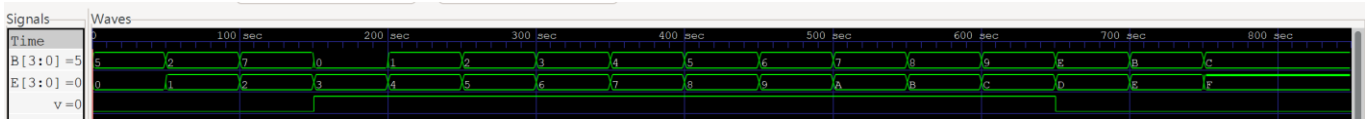
Then we could draw the logic diagram for this converter:



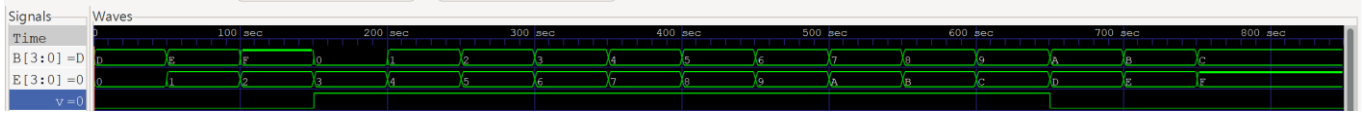
The simulations are:

Gatelevel modeling:

Dataflow modeling:



Behavioral modeling:



The three results are all correct, since when v=1, all B's equal to E-3.

(6)

我覺得這次的 Lab 很有趣，讓我回想起了小時候在玩 Minecraft 時接電路的心情，從完成一個小電路(Half Subtractor)到接出相對大的機器(Ripple Borrow Subtractor)，令人相當有成就感。

這次的 Lab 中我也遭遇了不少挑戰，其一是會忘記上次 Lab 學過的東西，比如說一個 testbench 測 4 組測資時要把輸出改成不同代號，這樣他們的結果才不會互相影響，這點就困擾了我一段時間，後來討論時發現不是只有我有遇到這個問題。

另外，always 裡面的語法也困擾了我一段時間，最後寫出來的東西雖然是對的，但卻感覺難以實際應用，電機的同學也說我把它當 Python 在寫，但我實在想不出，也討論不出更好的解法，只好這樣了。

另外，延遲的判定一直是我不拿手的地方，常常一個圖要看好幾次才能算對，這些都是我未來仍然值得精進的地方。