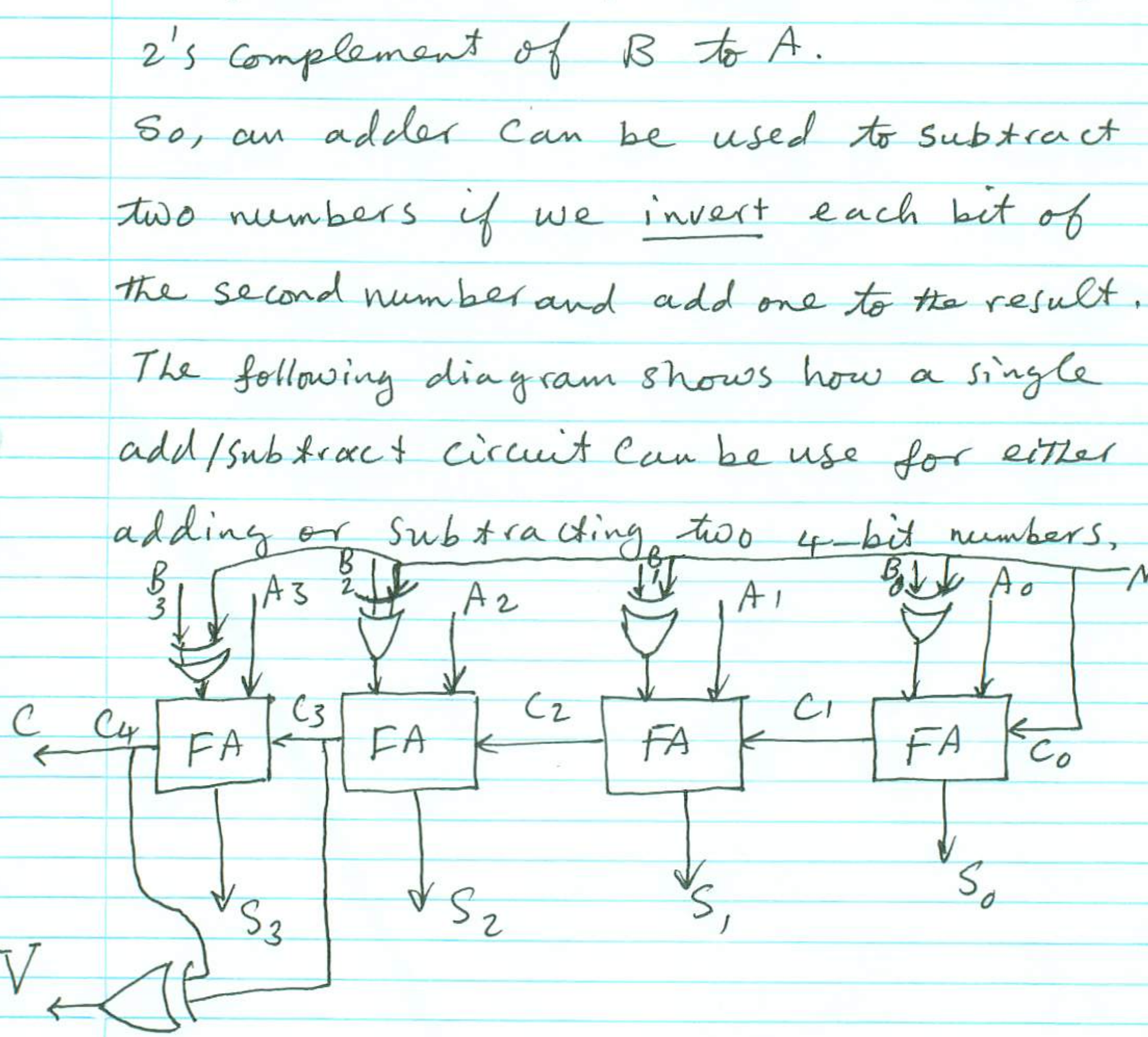
Lecture 10, Feb. 7, 2007

Subtraction

We saw in first lecture that subtracting B from A is equivalent to adding 2's complement of B to A.

So, an adder can be used to subtract two numbers if we invert each bit of the second number and add one to the result.

The following diagram shows how a single add/subtract circuit can be use for either adding or subtracting two 4-bit numbers,



When M = 0, A and B are added

When M = 1, B is Subtracted from A.

Note that when $M=1$ the second input to Full adders ar $B_i \oplus 1 = B_i'$, that is, each bit of B is inverted. Also $M = C_0 = 1$ means that we add a 1 to B to get 2's complement.

The output V is used to show overflow. When $V=1$, there is an overflow. This means that the result of addition or subtraction does not fit in an n bit register.

Assume for example that we are adding two 8-bit signed numbers. This means: 1 sign bit and 7 magnitude bits.

The range of numbers is from -127 to $+127$ (from 11111111 to 01111111).

But, when we add two numbers in this range, we may get a result that is not in the range. For example if we add $+60$ and $+70$ we get $+130$ or if we subtract 70 from -60

we get -130, both being out of range.

$$\begin{array}{rl} & \overset{\text{sign bit}}{\downarrow} \\ +60 & 00111100 \\ +70 & 01000110 \\ \hline +130 & 10000010 \quad \rightarrow -2 \end{array}$$

Notice that the sign bit indicates a minus and the seven remaining bits show 2.
Note that the sign bit becomes 1, when adding two positive numbers, if the carry from the previous bit is 1 since the carry from the last bit (when adding two positive numbers) is always zero (why?) then we have an <u>overflow</u> if $V = C_{n-1} \oplus C_n = 1$.

$$\begin{array}{rl} & \;\;\overset{\text{sign bit}}{\downarrow} \\ -60 & \;\;11000100 \\ -70 & \;\;10111010 \\ \hline -130 & \underset{\text{overflow}}{\underset{\uparrow}{1}}\,01110110 \end{array}$$

Here, we get an overflow and a positive (instead of negative) number if the carry into the last full adder is zero. Since the carry out of the last full adder is one (when adding two negative numbers), we have an overflow when $V = C_{n-1} \oplus C_n = 1$.

BCD Adder

When we add two binary coded decimal (BCD) digits and a carry from a previous decimal digit addition we get a number between 0 and $9+9+1=19$. A 4-bit adder gives the appropriate output while the sum of two digits is less than 10. When the sum is 10 or more we need to have a carry and a 4-bit number representing the balance (the difference between the sum and 10). For example when we add 6 (0110) and 7 (0111) we should get $C=1$ and 0011 or 1,0011 (13). But adding 0110 and 0111 with a 4-bit we get 01101. We need to find a way to Convert the output of a 4-bit adder to what we expect from a BCD adder.

The following table shows the output of the 4-bit adder and the BCD adder.

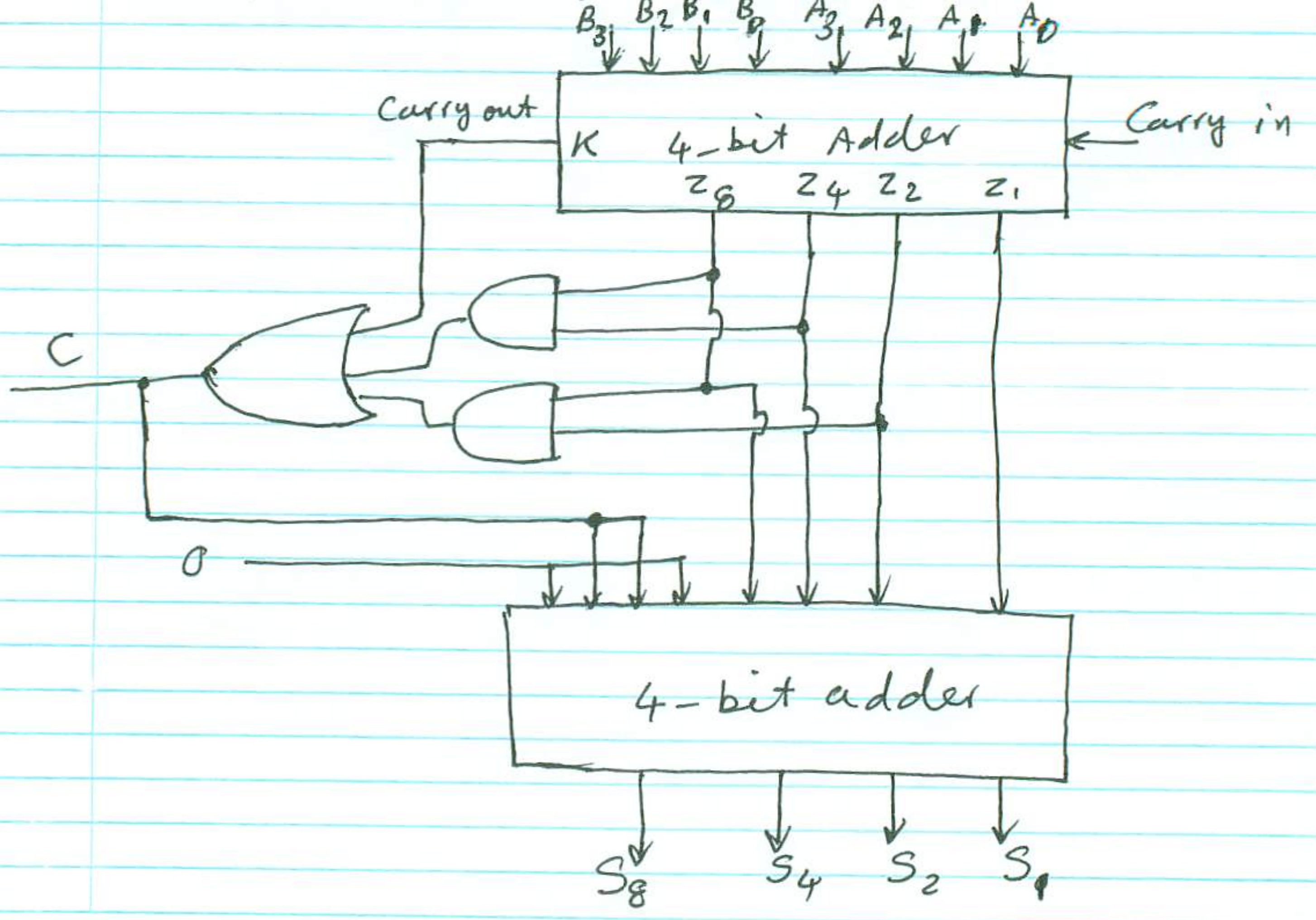
	Binary Sum					BCD Sum				
	K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0		0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

Note that $C = 1$ whenever $K = 1$ or $Z_8 = 1$.
But when $Z_8 = 1$ and both Z_4 and Z_2 are zero $C = 0$. So, we have a carry when $K = 1$ or $Z_8 = 1$ and eith Z_4 or Z_2 one

That is:

$$C = K + Z_8(Z_2 + Z_4) = K + Z_8 Z_4 + Z_8 Z_2$$

when $C = 1$, we need to add 6 (0110) to $Z_8 Z_4 Z_2 Z_1$ to get $S_8 S_4 S_2 S_1$. Following is an implementation of a BCD adder.



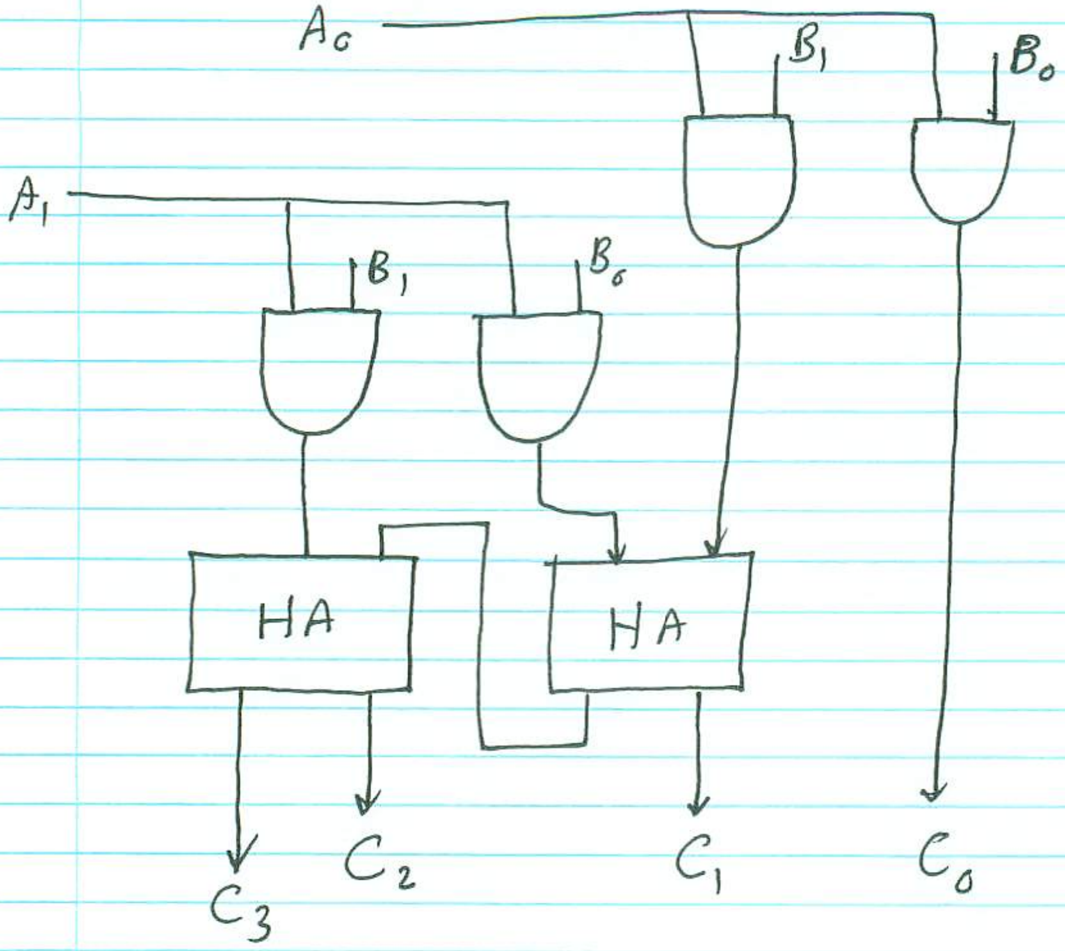
Binary Multiplication

Binary multiplication is done the same way we do decimal multiplication, i.e., we multiply the multiplicand by each bit of the multiplier to form a partial product. Then we shift each partial multiplication result and add them to gether.

As an example take the multiplication of $B_1 B_0$ and $A_1 A_0$. That is we want to design a two bit multiplier.

$$\begin{array}{ccccl}
 & & B_1 & B_0 & \\
 & & A_1 & A_0 & \\
\hline
 & & A_0B_1 & A_0B_0 & \text{multiplying } B_1B_0 \text{ by } A_0 \\
 & A_1B_1 & A_1B_0 & & \text{multiplying } B_1B_0 \text{ by } A_1 \text{ and Shifting the result.} \\
\hline
 & A_1B_1 & A_0B_1+A_1B_0 & A_0B_0 & \text{Adding two partial products.} \\
C_3 & C_2 & C_1 & C_0 &
\end{array}$$

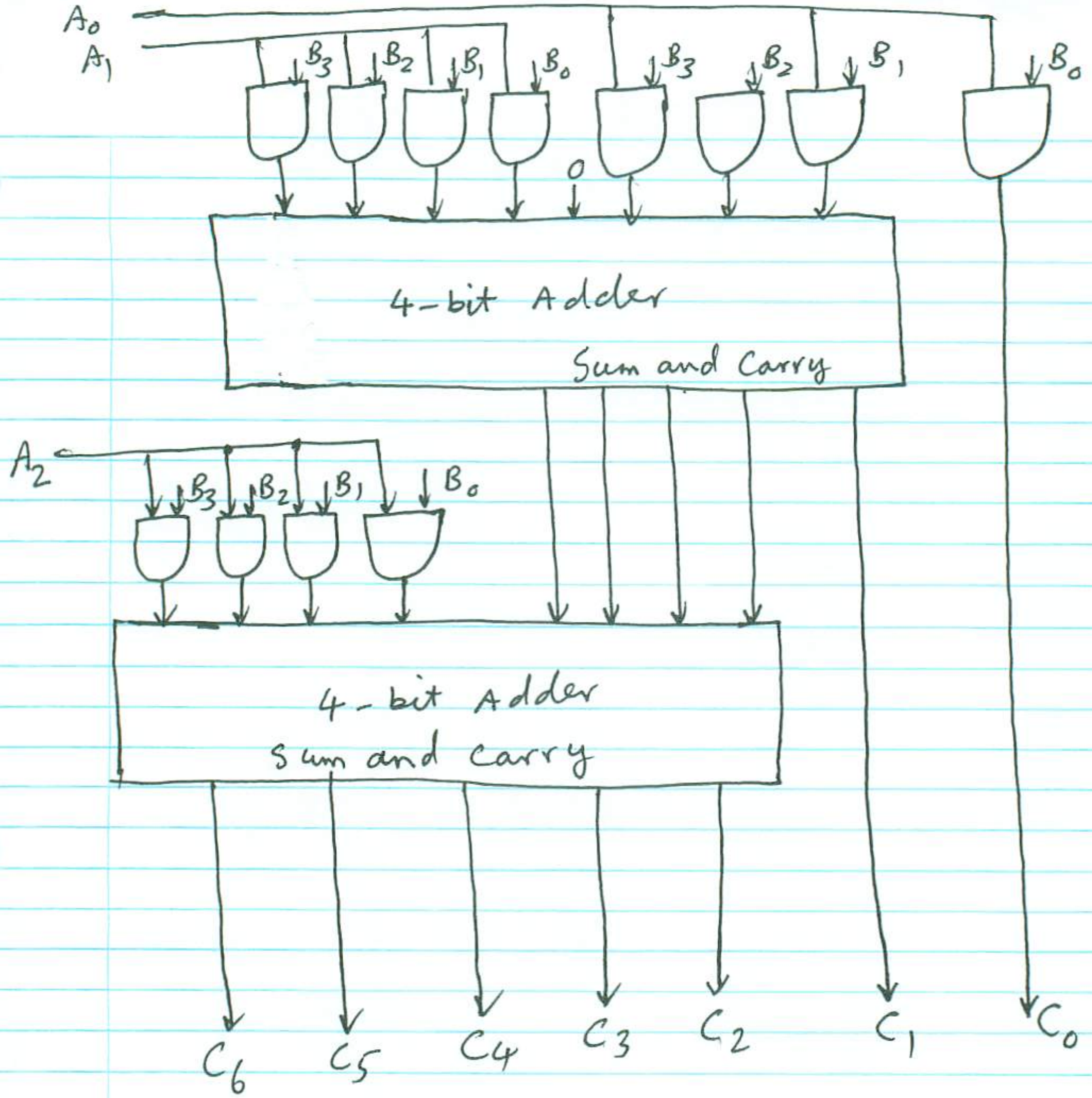
The 2-bit multiplier can be implemented as,



4-bit by 3-bit multiplier

In this example we multiply $B_3\ B_2\ B_1\ B_0$ by $A_2\ A_1\ A_0$

$$
\begin{array}{ccccccc}
 & & & B_3 & B_2 & B_1 & B_0 \\
 & & & & A_2 & A_1 & A_0 \\
\hline
 & & & A_0B_3 & A_0B_2 & A_0B_1 & A_0B_0 \\
 & & A_1B_3 & A_1B_2 & A_1B_1 & A_1B_0 & \\
 & A_2B_3 & A_2B_2 & A_2B_1 & A_2B_0 & & \\
\hline
C_6 & C_5 & C_4 & C_3 & C_2 & C_1 & C_0
\end{array}
$$


A_0
A_1
B_3
B_2
B_1
B_0
B_3
B_2
B_1
B_0
0
4-bit Adder
Sum and Carry
A_2
B_3
B_2
B_1
B_0
4-bit Adder
Sum and carry
C_6
C_5
C_4
C_3
C_2
C_1
C_0