Lecture 13

Computer Arithmetic

Single Bit Addition

The addition of two bits can be specified by a truth table:

A	B	SUM	$CARRY$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Designing with minterms gives us the Boolean equations:

$$SUM = A' \cdot B + A \cdot B'$$
$$CARRY = A \cdot B$$

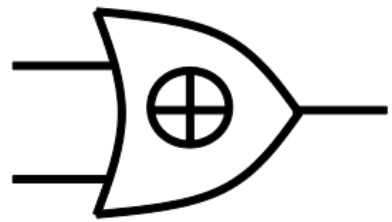
The Exclusive OR (again)

The truth table for SUM is the same as the "exclusive or" gate.

$$SUM = A' \cdot B + A \cdot B' = A \oplus B$$

We have already made use of the exclusive OR gate for circuit design.

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

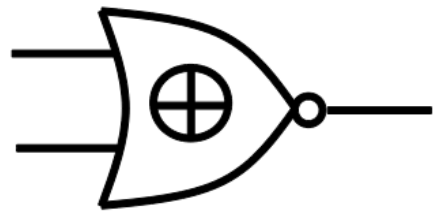


The exclusive NOR (again)

The exclusive NOR gate is also useful for circuit design:

$$(A \oplus B) = A' \cdot B' + A \cdot B$$

A	B	(A⊕B)'
0	0	1
0	1	0
1	0	0
1	1	1

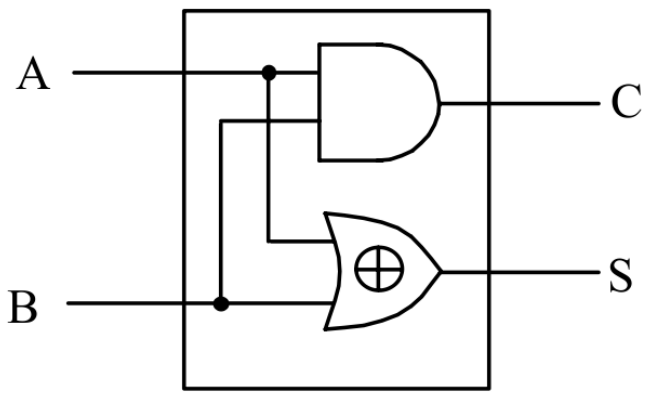


Exclusive OR and NOR equations can be used as simplifying rules in Boolean algebra, but they are difficult to spot on the Karnaugh map.

The Half Adder

To summarise, the one bit adder, which is called a half adder, is represented by the following equations and circuit:

$$CARRY = A \cdot B \qquad SUM = A \oplus B$$



The Full Adder

If we need to add binary numbers longer than 1 bit (which might come in handy from time to time) we need to propagate a carry. This means we must add three binary digits at each stage.

A	B	C_{in}	SUM	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Equation of the full adder

A	B	C_{in}	SUM	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

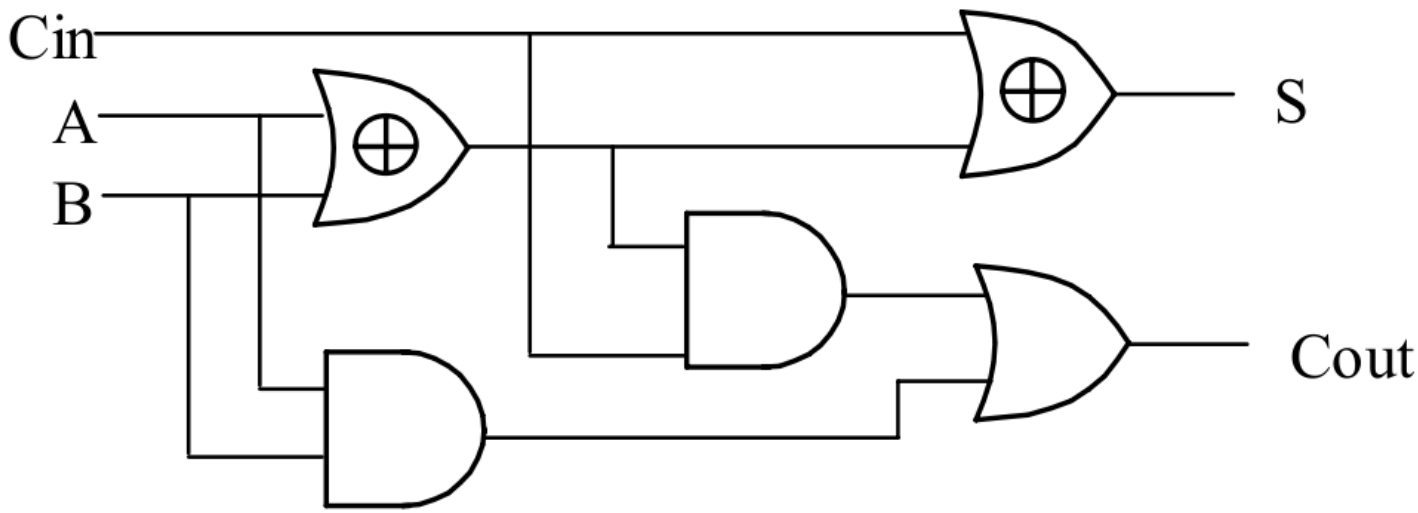
$$
\begin{aligned}
SUM &= A' \cdot B' \cdot C + A' \cdot B \cdot C' + A \cdot B' \cdot C' + A \cdot B \cdot C \\
&= A' \cdot (B' \cdot C + B \cdot C') + A \cdot (B' \cdot C' + B \cdot C) \\
&= A' \cdot (B \oplus C) + A \cdot (B \oplus C)' \\
&= A \oplus B \oplus C
\end{aligned}
$$

$$
\begin{aligned}
C_{out} &= A' \cdot B \cdot C + A \cdot B' \cdot C + A \cdot B \cdot C' + A \cdot B \cdot C \\
&= C \cdot (A' \cdot B + A \cdot B') + A \cdot B \\
&= C \cdot (A \oplus B) + A \cdot B
\end{aligned}
$$

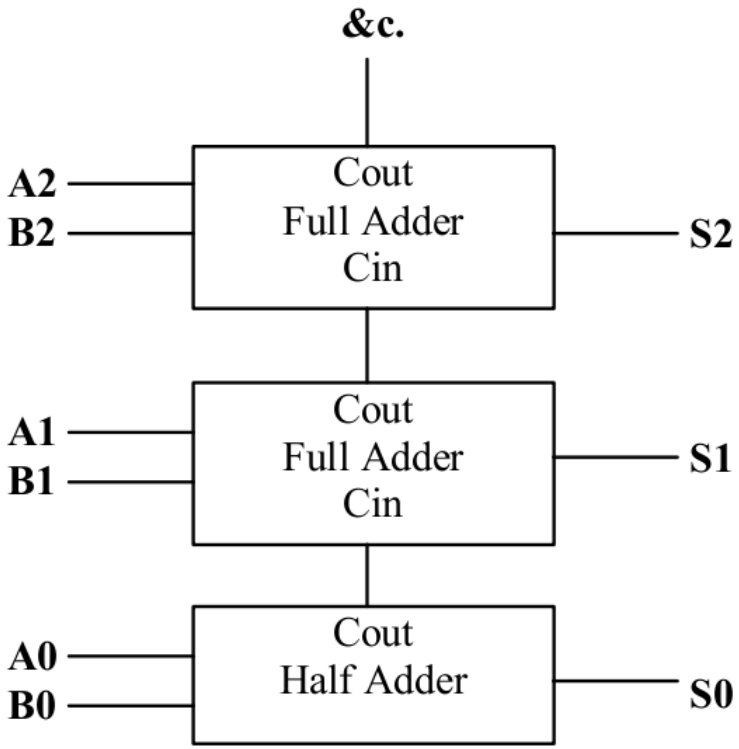
Circuit of the Full Adder

$$SUM = A \oplus B \oplus C$$

$$C_{out} = C \cdot (A \oplus B) + A \cdot B$$

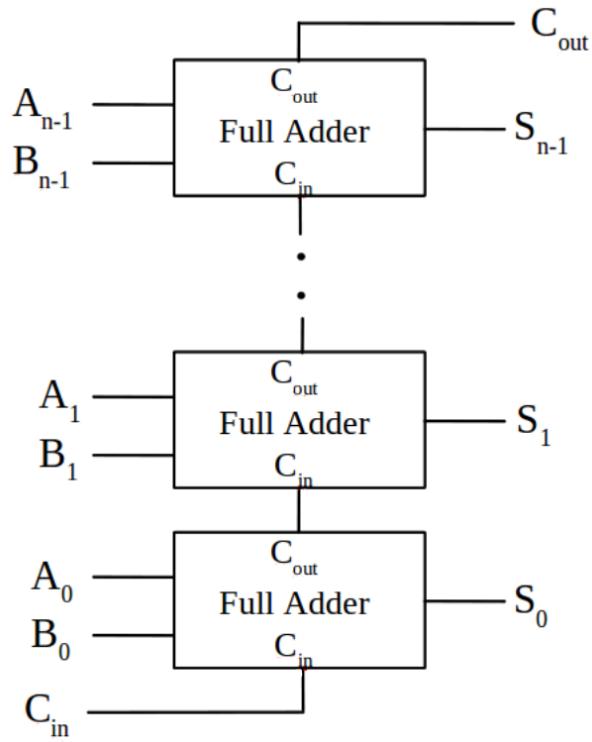


Building an *n*-bit Adder



Building an *n*-bit Adder with Carry in

- It is usual to provide a carry in when designing an *n* bit adder.
- It allows us to use the functional design approach, for example to build a 32 bit adder by combining four 8 bit adders.
- The carry in can also be used for other purposes.



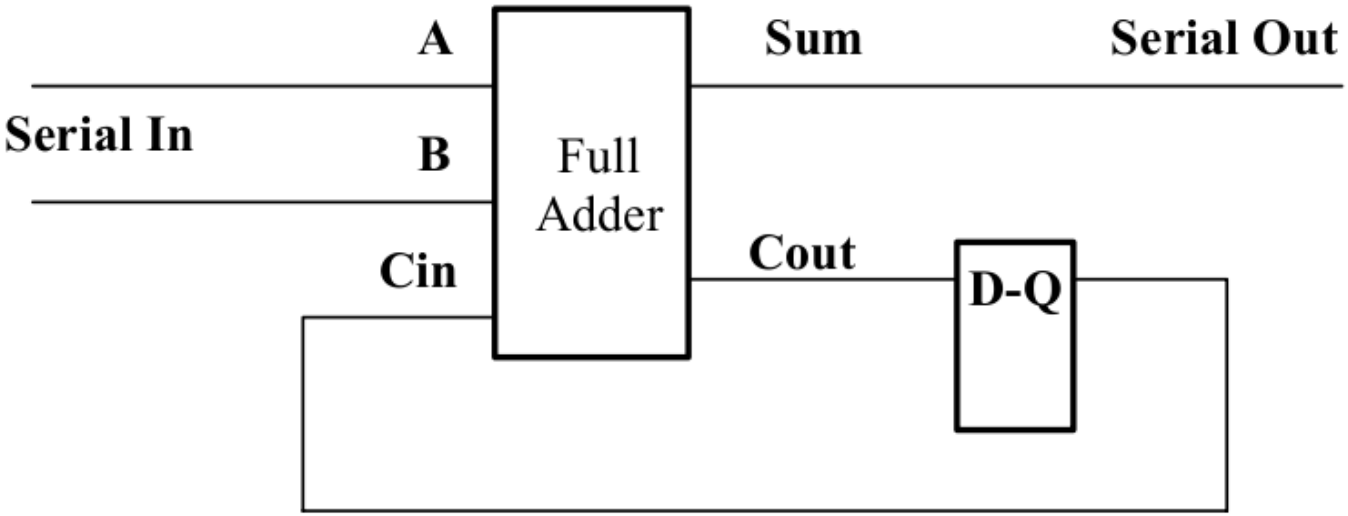
Ripple Through Carry

- The previous circuit is called the *ripple through carry adder*.
- There are faster circuits which are designed to propagate the carry faster. In the books you will find many others, e.g. *the look ahead carry adder*.
- However, we will not discuss these further in this course.

The Serial Adder

Sometimes in engineering applications it is useful to be able to add binary numbers which are in serial form.

The circuit below assumes that the bits arrive with the least significant first.



Subtraction

Subtraction can be defined by a truth table using the borrow and payback method. The truth table for $A - B$ is as follows.

A	B	P	*DIFFERENCE*	*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

Problem

Evaluate 11001 – 10110 using the truth table below

A	*B*	*P*	*DIFFERENCE*	*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

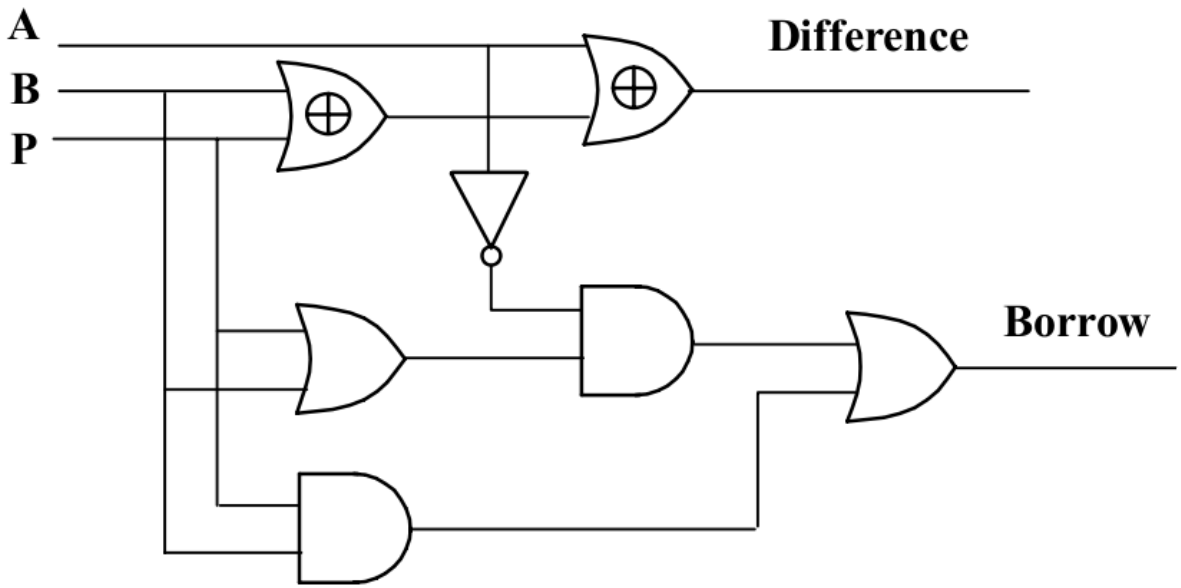
Subtractor Circuit

- The minterm method can be used to determine a circuit for the subtractor.
- We will not bother with the algebra, but just note that the simplified equations are:
  - $DIFFERENCE = A \oplus B \oplus P$ (again!)
  - $BORROW = B \cdot P + A' \cdot (B + P) = B \cdot P + A' \cdot (B \oplus P)$
- Note that we can change $B + P$ to $B \oplus P$ since the case $B = P = 1$ is covered by the term $B \cdot P$. This saves one gate.

One bit full subtractor circuit

$$DIFFERENCE = A \oplus B \oplus P$$

$$BORROW = B \cdot P + A' \cdot (B \oplus P)$$



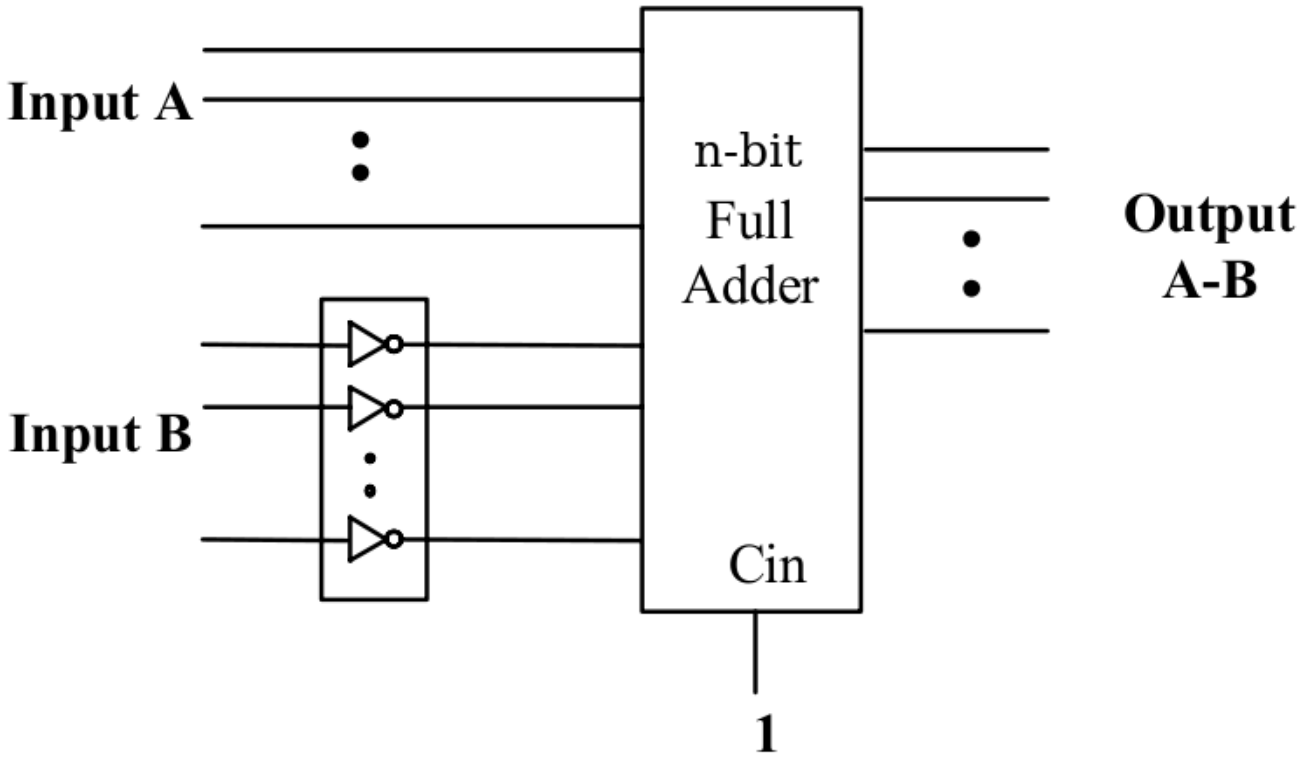
The n-bit subrtactor can be built following the same method used for the n-bit adder.

Two's Complement Subtractor

- It is also common practice to use a two's complement subtractor in hardware.
- $A - B = A + TwosComplement(B)$
- The two's complement is formed by flipping each bit of a number and adding one.

This method is attractive since we re-use the adder design we have already created.

Two's Complement Subtractor Circuit



Note the use of the C_{in} input for incrementing.

Multiplication

Hardware multipliers work by using the familiar method of long multiplication. For example:

$$\begin{array}{cccc} & & 2 & 1 \\ & & 3 & 4 \\ \hline & 6 & 0 & 0 \\ & & 3 & 0 \\ & & 8 & 0 \\ & & & 4 \\ \hline & 7 & 1 & 4 \\ \hline \end{array}$$

or in general for a 2 digit decimal number:

$$a_1 a_0 \times b_1 b_0 = a_1 \times b_1 \times 10^2 + a_0 \times b_1 \times 10 + a_1 \times b_0 \times 10 + a_0 \times b_0$$

Binary Multiplication

Binary multiplication is much the same, but uses powers of 2 rather than 10:

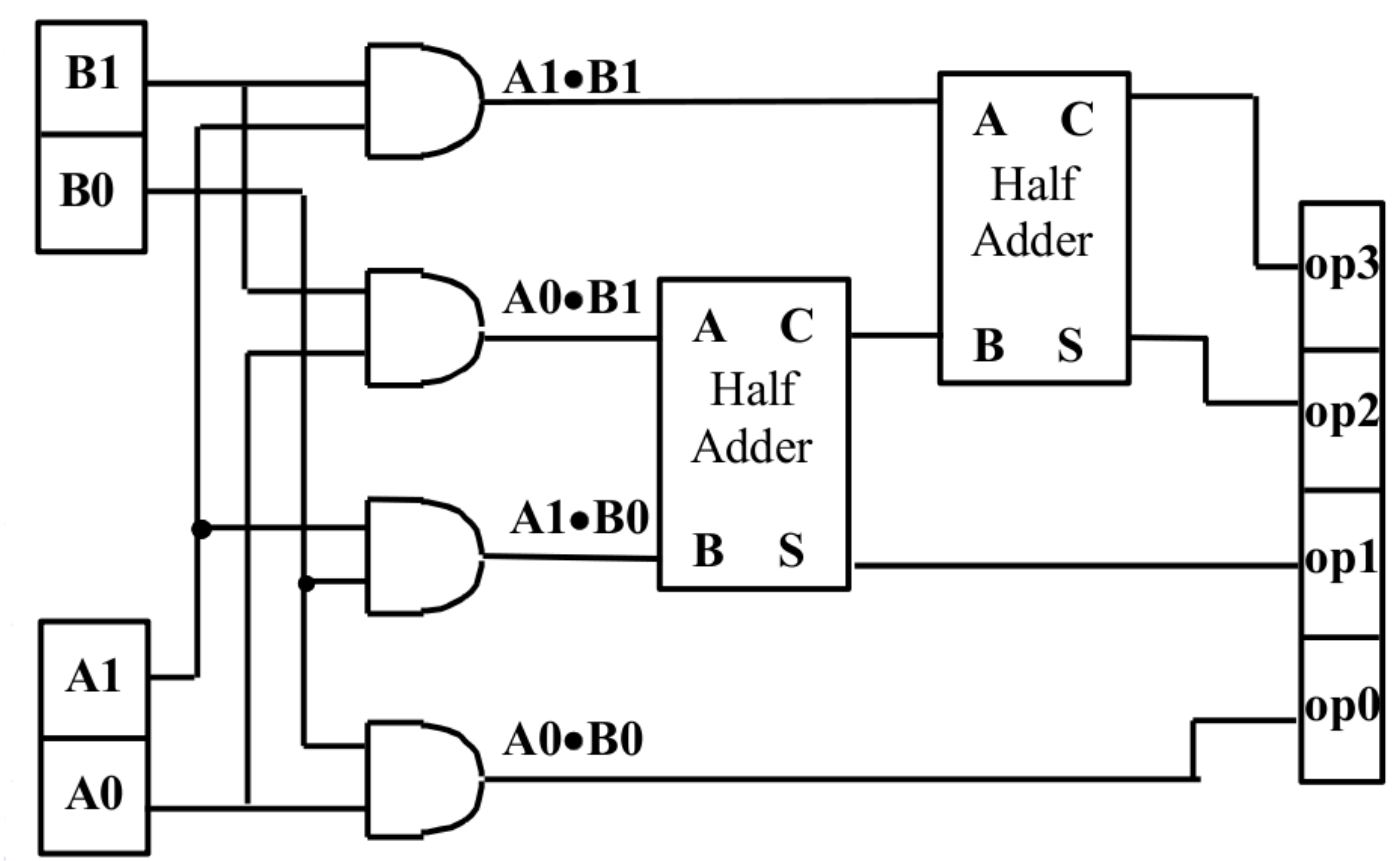
$$a_1 a_0 \times b_1 b_0 = a_1 \times b_1 \times 2^2 + a_0 \times b_1 \times 2 + a_1 \times b_0 \times 2 + a_0 \times b_0$$

For binary digits we can replace $\times$ by *AND*, and implement 'multiply by 2' with shift left:

$$a_1 a_0 \times b_1 b_0 = (a_1 \cdot b_1) \Leftarrow 2 + (a_0 \cdot b_1) \Leftarrow 1 + (a_1 \cdot b_0) \Leftarrow 1 + a_0 \cdot b_0$$

A 2 bit multiplier circuit

The shifts are created directly by wiring the adders as shown below:



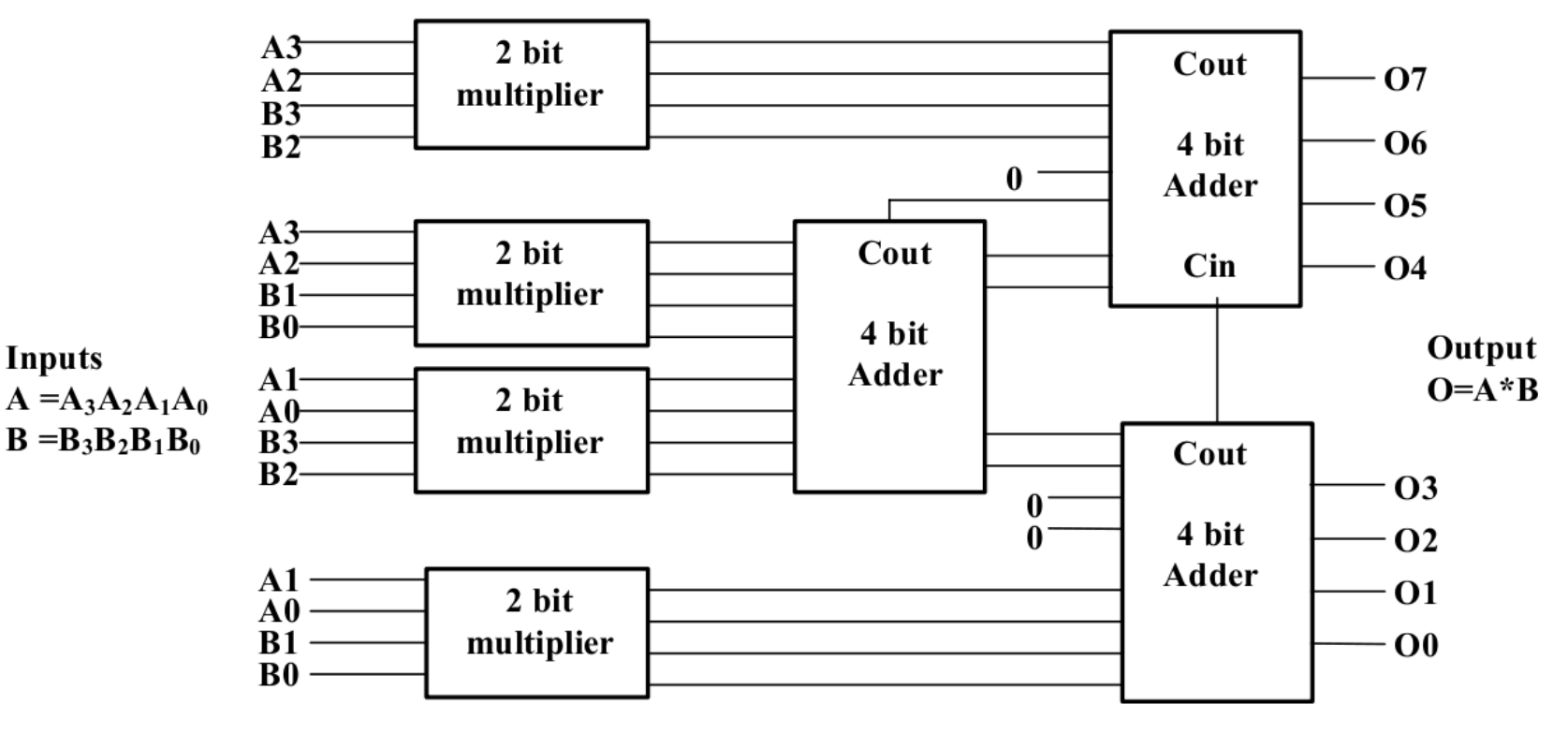
A four bit multiplier

- A four bit multiplier works in the same way using base 4 arithmetic.
- If we write two four bit numbers as pq and rs, where p, q, r and s are 2 bit numbers then:

  $$pq \times rs = p \times r \times 4^2 + p \times s \times 4 + q \times r \times 4 + q \times s$$

- The two bit products: $p \times r, p \times s, q \times r$ and $q \times s$ can be computed using the two bit multiplier just designed.
- The 'multiplies by 4' can be implemented by shifts created by wiring the adders appropriately

The four bit multiplier circuit



Eight, sixteen and thirty two bit multipliers

- We can now use the same method to design an eight bit multiplier from four four bit multipliers and three eight bit adders. Care is needed to wire the adders to achieve the required shifts.
- Similarly a sixteen bit multiplier can be built out of four eight multipliers, and so on up to the precision that we need.
- The scaling up isn't as elegant as that for the adder, multiplexer or decoder, but it still gives us a design method which manages the complexity of a big circuit.
- The circuit works for unsigned integers only. For signed multiplication extra hardware is required.

Division

- On older systems division was done procedurally using shifts and subtracts. This process is algorithmic and can be thought of as programming.
- Combinatorial hardware dividers have been designed, but they are complex and we will not discuss them here.

The ALU

- In practice, arithmetic circuits are bundled together in a multi-function package called the:

  Arithmetic and Logic Unit (ALU)

- We will look in more detail ALU next lecture, but for now we illustrate the principal by designing an add/subtract circuit.

The add-subtract circuit