Lecture 13: Arithmetic

Addition

The addition of two binary numbers is carried out in a bitwise fashion, just as normal addition. We add any two bits according to the following truth table:

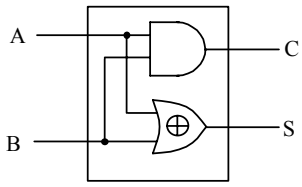
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SUM = A'•B + A•B'
CARRY = A•B

Notice that in our design method for combinational circuits we do not use the "exclusive or" gate. The truth table for this gate is the same as that for SUM above. At the transistor level it is possible to make "exclusive or" and “exclusive nor” gates as simply as nand and nor gates, and therefore we should always look for "exclusive or" and “exclusive nor” simplification.
The minterm formulation of XOR and XNOR simplifications is :

XOR : A'•B + A•B' = A⊕B

XNOR : (A•B + A'•B') = (A⊕B)'

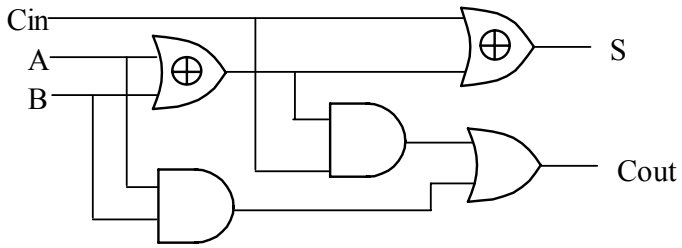


Where ⊕ stands for the "exclusive or" function. In the above equations there is one instance of this:

SUM = A⊕B

The above equations make up a "half adder". When adding numbers of more than one bit in length we need to add the carry from the previous stage as well as the two digits. The full adder has a carry in (Cin the table below) and is represented by the following truth table:

A	B	Cin	SUM	Cout
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



Which yields the equations:

SUM = A'•B' •C + A'•B•C' + A•B'•C' + A•B•C
= A'•(B'•C+B•C') + A•(B'•C'+B•C)
= A'•(B⊕C) + A•(B⊕C)'
= A⊕B⊕C

CARRY = A'•B•C + A•B'•C + A•B•C' + A•B•C
= C•(A'•B+A•B') + A•B
= C•(A⊕B) + A•B

Any precision arithmetic can be implemented by means of a half adder for the bottom two bits followed by a set of full adders for all the other bits connected as shown in Diagram 13.1. Notice that the more bits that are required, the longer it

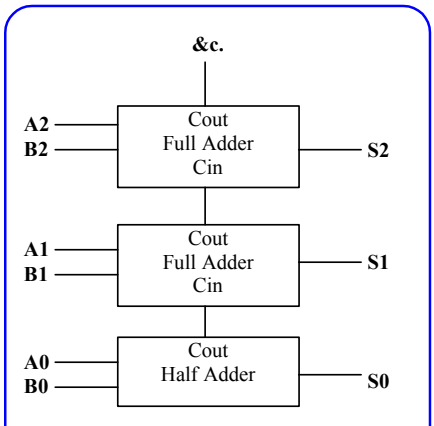


Diagram 13.1: The n-Bit Adder

will take for the adder to complete the sum.

It is possible also to add two streams of serial data using a full adder and a flip flop to delay the carry from one stage to the next as shown in Diagram 13.2. Notice that the serial data streams must be ordered with the least significant bit first.

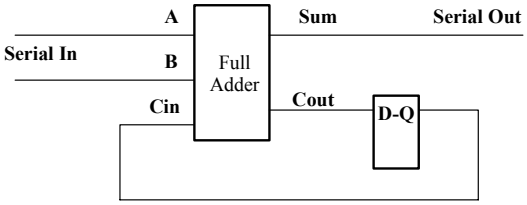


Diagram 13.2 The Serial Adder

Subtraction

Subtraction circuitry can be designed in an analogous manner, but this time we need to considering borrowing from the next most significant bit. The truth table for a full subtractor which takes B from A with a pay back P from the previous stage is:

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

$DIFFERENCE = A \oplus B \oplus P$

$BORROW = B \bullet P + A' \bullet (B + P)$

The full subtractor for one bit can be implemented as shown in diagram 13.3, and circuit for n bits is connected in an equivalent manner to Diagram 13.1. In practice, subtraction is often achieved by twos complement addition. The twos complement of a binary digit is found by complementing the individual bits of a number and adding one to the bottom digit, losing the carry in the case where the number was a zero. The method is illustrated by Diagram 13.4

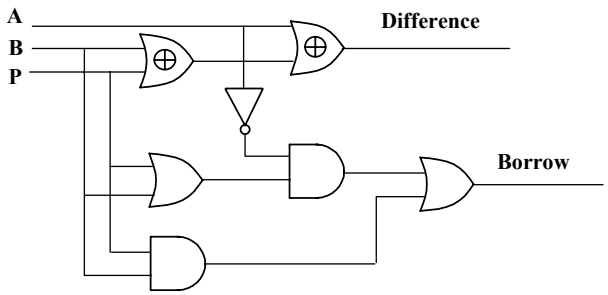


Diagram 13.3 One bit full subtractor

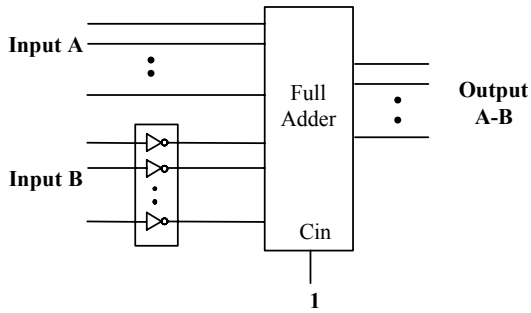


Diagram 13.4 Twos Complement Subtractor

Multiplication

Multiplication of two numbers is carried out by a process of multiplying all combinations of the individual digits, and adding them up in the appropriate positions. For example, we can multiply 13 by 42 by taking the four individual products 4×3, 4×1, 2×3 and 2×1, and adding them raised to the appropriate power of 10 to form: $4\times1\times10^{2} + 4\times3\times10^{1} + 2\times1\times10^{1} \times 2\times3$. This is the way that long multiplication is normally carried out. We can apply the same principal to binary arithmetic. In the simplest case let us consider multiplying two two digit numbers:

$$A_1A_0 \times B_1B_0 = A_1 \times B_1 \times 2^2 + A_1 \times B_0 \times 2^1 + A_0 \times B_1 \times 2^1 + A_0 \times B_0$$

Now, since A_1, A_0, B_1 and B_0 are all binary digits we can replace the multiplies by ANDs

$$A_1A_0 \times B_1B_0 = A_1 \bullet B_1 \times 2^2 + A_1 \bullet B_0 \times 2^1 + A_0 \bullet B_1 \times 2^1 + A_0 \bullet B_0$$

And, since multiply by two is equivalent to shift right, we can replace these by shifts which we hard wire to obtain the multiplier of Diagram 13.5.

The next step is to apply the same reasoning recursively. In other words we can design a four bit multiplier by breaking each four digit number into two groups of two, and writing:
PQ×RS =

$$P\times R\times 2^4 + P\times S\times 2^2 + Q\times R\times 2^2 + Q\times S$$

We observe that since P,Q,R and S are two digit numbers, the products P×R, P×S, Q×R and Q×S can be computed using the two bit multiplier that we just designed. Thus the circuit for our four bit multiplier is given in Diagram 13.6. Clearly we can extend this idea to design a multiplier for a bit length of any power of two.

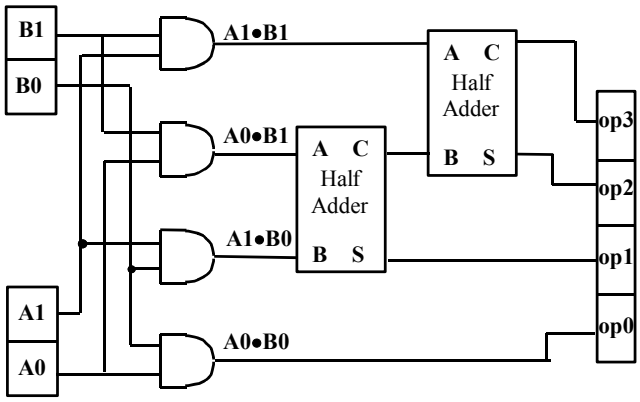


Diagram 13.5:The two bit combinatorial multiplier

The above design only works for unsigned binary digits. In order to extend it to signed numbers we need to add further circuitry to detect if either of the input numbers is negative. This can be done quite simply by looking at the top bit, which in twos compliment arithmetic is 1 for negative numbers. This top bit can be used to control a multiplexer which either selects the number or its twos complement. The twos complement can be implemented by an inverting each bit then incrementing with an adder. The output sign can be determined by the exclusive or of the input signs, and a twos complement circuit with a multiplexer is also required to set it correctly.

Division

It is not usual to build a combinational circuit to carry out division. Instead this is done procedurally, with a sequential circuit, or in the machine code using shifts and subtracts.

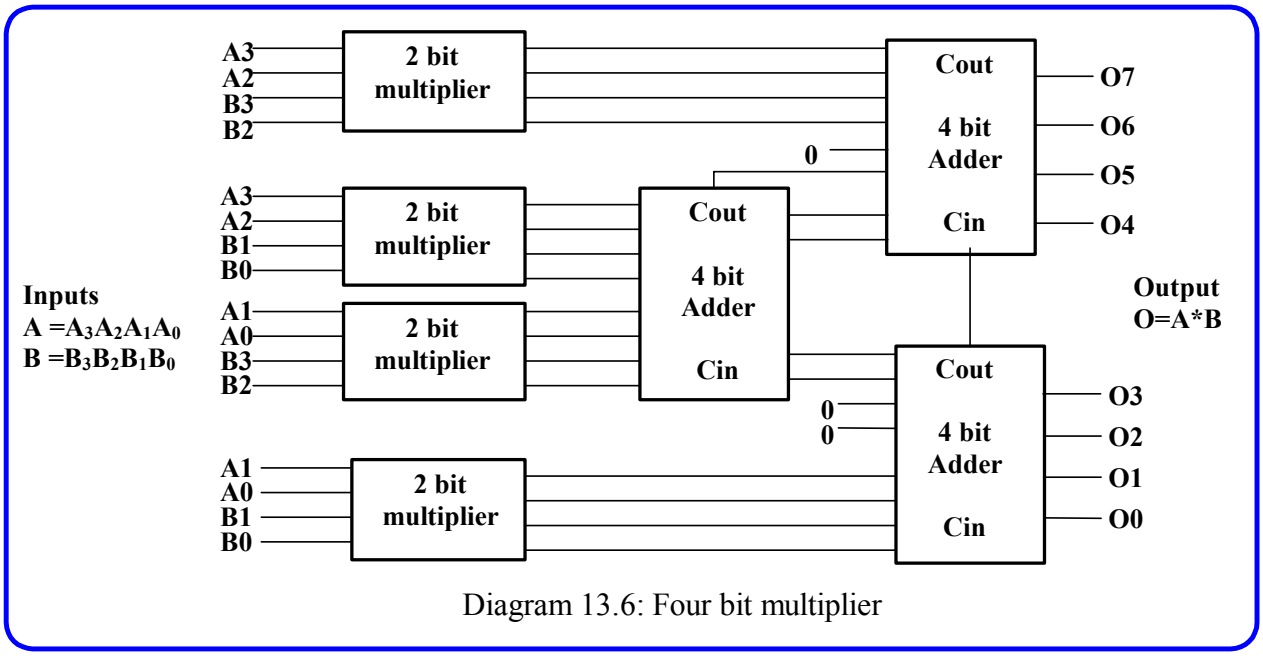


Diagram 13.6: Four bit multiplier