FIRST BASE

by Mike Wharton

A Beginner's Guide to Logic Design Part 3

Solution to Problem

If you recall, there was a little problem left for you to sort out in the last section. This was to deduce the Truth Table of an array made up of two-input NAND gates, and the result which you should have arrived at is given in Fig. 1. Comparison of this table with published ones will show it to be that of the Exclusive-OR gate, (EX-OR). The common symbol for this gate, also known as the Difference gate, is shown in Fig. 2a. It is called the Difference gate since a look at its Truth Table will reveal that the output is high only when the inputs are different; the complement of this gate is the Exclusive-NOR gate, (EX-NOR), whose symbol is shown in Fig. 2b. This gate is also known as an Equivalence gate, since its output is high when the inputs are the same, and the Truth Table for this gate is shown in Fig. 3.

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Figure 1. Derived truth table for 2 input Exclusive OR gate.

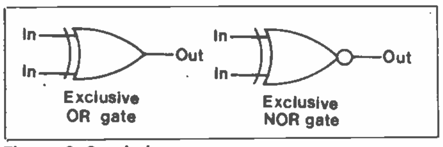


Figure 2. Symbols

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

Figure 3. Truth table for Exclusive NOR gate

It would be possible to produce an EX-NOR gate by adding an inverter to the output of the previous EX-OR gate made up from NAND gates, thus using a total of five 2-input NAND gates. This would be quite wasteful of gates, and not surprisingly it is possible to obtain both of these devices in a single package. Thus Fig. 4a. shows the pinout of the 7486, a quad 2-input EX-OR gate package, and Fig. 4b. gives the pinout of the 74266, the EX-NOR gate package.

This now completes the list of main logic gates, although there are a few others which

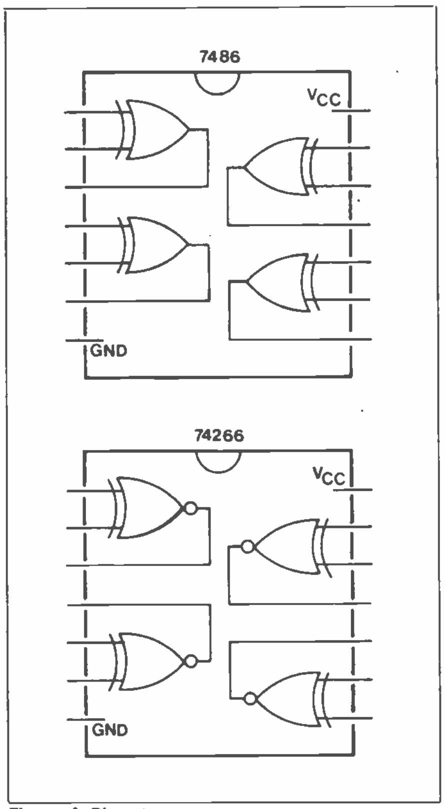


Figure 4. Pinouts

can be obtained, but these really combinations of the above types in order to obtain 'programmable' gates in the one package. An example of this is the 7451 AND-OR-INVERT gate, shown in Fig. 5; here it may be seen that the package contains two AND gates connected to the input of the NOR gate. It is left as an exercise for the reader to derive the Truth Table for this arrangement of gates.

Multi-input gates

So far we have really only concerned ourselves with gates having one or two inputs. Many of the devices available have more than this, as a glance at the relevant pages of the Maplin Catalogue will reveal. For example, the 7430 is an 8-input NAND gate, shown for reference in Fig. 6 along with its Truth Table. Fortunately, this does not make the understanding of these gates that much more difficult. If you look back at the previous Truth Tables, as well as the one for the 8-input NAND gate, you will see that they all have a unique output state. An exception to this rule are the Truth Tables for the EX-OR and EX-NOR gates, which are special cases. The other gates have just one value of logic output for a particular set of inputs; for example, in a 2-input AND gate the output is always low except when both inputs are high. In a 2-input NAND gate, the output is always high, except when both inputs are high, and this follows on for the 8-input NAND gate, where the output is always high except when all the inputs are high.

That this is so can be tested by connecting up a 7430 on a bread-board with a

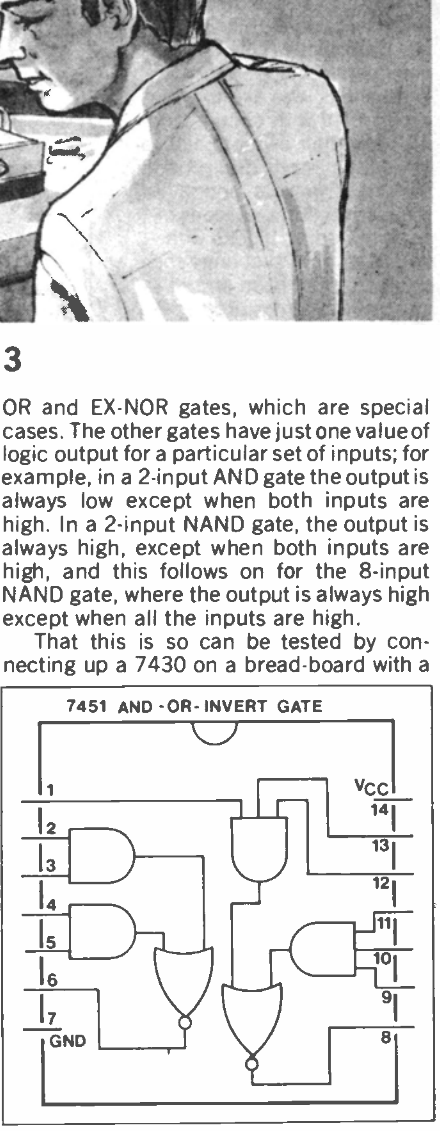
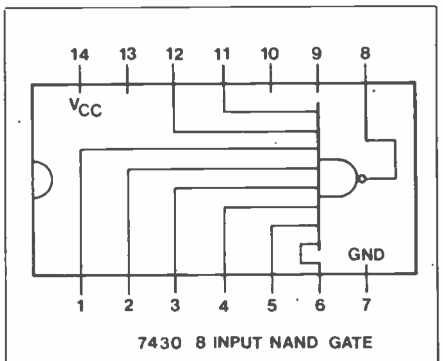


Figure 5. Pinout



A	B	C	D	E	F	G	H	Q
0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1
				etc				
0	0	0	0	0	0	0	1	1
1	1	1	1	1	1	1	1	0

Figure 6. Pinout and truth table

LED wired to the output, as shown in the last issue. If each of the inputs is connected to logic 1 then the output will be found to be at logic 0, with the LED extinguished. If one of the inputs is now taken to logic 0, then the LED will light up, and will remain alight while any number of inputs are held at logic 0.

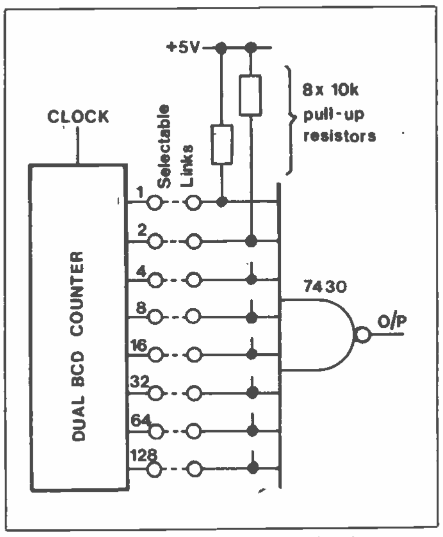


Figure 7. Part of counter/decoder circuit

The use of such a device may be demonstrated by referring to the part of a circuit shown in Fig. 7. The problem here was to produce a signal from the output of the 8-input NAND gate after the counter had counted a selectable number of clock pulses. To achieve this action, each of the inputs is connected to logic 1 by a 'pull-up' resistor, thereby ensuring that the output will be logic 0. The numbers shown by the outputs from the BCD counter are the number of clock pulses which need to be counted before that particular output goes high, assuming a start from zero. Without going into any further detail of how the outputs from the counter would appear, by connecting the appropriate links it is possible to set the circuit to count any value of pulses from 1 to 255. For example, if it were required to count up to 23 clock pulses before a logic 0 appeared at the output of the NAND gate, then the links for 1,2,4 and 16 would be made, since 1+2+4+16=23.

The individual pull-up resistors are needed on the inputs in order to ensure that any unconnected inputs are held at logic 1; the value of these resistors is not all that critical, but it must be remembered that the output of the counter will be required to sink the current through them when it goes low. The BCD counter is a rather different type of animal from the ones we have encountered so far, belonging to the breed of sequential logic devices. This is a whole range of beasties which will be dealt with in a lot more detail in a subsequent article.

Arithmetic Logic Units

Any reader who has perused books or articles on the subject of micro-processors or micro-computers, and these days it's hard to avoid them, may well have come across the term Arithmetic Logic Unit, or ALU. This is the part of the micro-processor which is concerned with 'doing sums' and other logical operations. Needless to say, in a real life processor, this section contains a multitude of functional devices, but it is possible to emulate one of its basic building blocks, the Adder. Side-stepping the old jokes about venomous snakes, the digital adder comes in two types, the half-adder and the full-adder. However, before we delve into the workings of these circuits, it may well be a good idea to brush up on some binary arithmetic.

I am sure everyone reading this is fully conversant with denary arithmetic, that is working in powers of ten. In binary arithmetic the same rules apply, but in this case we are using the number base of two, with the digits 0 and 1. When two denary (or decimal) digits are added together there are two possible situations:

a) a third digit, larger than the other two results, but smaller than the base of the number system, eg,

```
 5     1
+3    +4
 8     5
```

The new digit, 8 or 5 in these examples, is called the SUM.

b) the third digit is equal to or larger than the base of the number system,

```
       5          8
      +6         +7
 1     1      1    5
CARRY SUM   CARRY SUM
```

In this case the position of the digits comes into play and the answer consists of two parts, the SUM and the CARRY. The generation of Sum and Carry occurs whatever number base is in use. In binary addition the generation of Carry bits occurs much more often, as there are only two digits.

```
 0      0             1
+0     +1            +1
 0      1         1   0
SUM    SUM    CARRY  SUM
```

These examples cover nearly all the possible combinations of binary addition, the only other one being where the 0 and 1 are reversed in the middle example!

Where binary numbers containing more than one digit are to be added, then the process can be broken down into a series of repeated two-digit additions, until the process is complete. For example:-

```
        10        111
       +01       +010
CARRY   11   SUM 1001
```

In the second example, the addition of the first (right-hand) digits of 0 and 1 gives a Sum of 1, and no Carry; adding the next two digits, 1 and 1, produces a Sum of 0 and a Carry of 1. The next stage is to add together 0, 1 and the Carry; as before 0 and 1 give a Partial Sum of 1, and adding the 1 carried over gives a Sum of 0 and a Carry into the next column. The simple rules of binary addition may be summarised in a Truth Table, shown in Figure 8.

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

Figure 8. Binary addition truth table

Looking at this Table it is possible to see that a Sum OR a Carry is the result of a binary addition, never a Sum AND a Carry. To perform this operation with logic gates, it i only necessary to find ones which have the same Truth Table as that for binary addition. The circuit would require two inputs, A and B and two outputs to correspond to the Sum and Carry. This can, in fact, be achieved in several different ways; if you look back at the Truth Table for the EX-OR gate and the AND gate it is apparent that the Sum part is the same as the EX-OR truth table and the Carry part is the same as the AND gate. Actually, this is not quite a full solution, since no account has been taken of the fact that a Carry bit may have been produced by an earlier stage, and hence this is known as the half-adder.

Half-Adder Circuit

A digital half-adder circuit may be made up, on a bread-board, following the diagram given in Figure 9. Here it can be seen that the two gates which are required are the EX-OR and the AND gates. Possibly the most convenient method of making up this circuit is to use single gates from a 7486 and a 7408, and connect them up as shown. In this case the two bits to be added are applied to inputs A and B to give the Sum and Carry appear at the corresponding outputs. It is also possible, remember, to make up such gates as these from the common NAND gate. We have already seen how the EX-OR gate may be made up from four 2-input NAND gates, and so to complete the picture figure 10 shows how the AND gate may be fashioned. It is left as a further exercise for the reader to make up the half-adder circuit from NAND gates and confirm that it is logically identical to the first design.

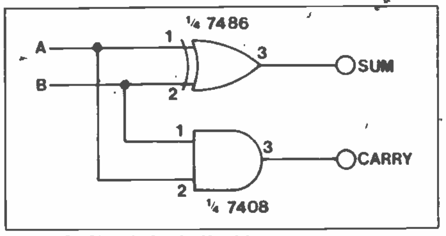


Figure 9. Circuit for half-adder

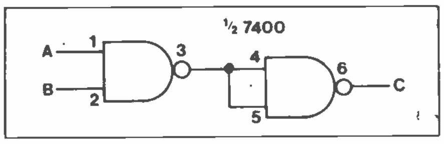


Figure 10. AND gate using NAND gates

Full-Adder Design

The half-adder is incomplete in that no provision is made for a 'carry-in' from a previous stage. In the case of the full-adder, not only is account taken of this, but also a provision is made for the possible generation of a 'carry-out' to subsequent stages. Again, the requirements of the full-adder may best be summarised in the form of a Truth Table; this will need to have three inputs, A,B and Carry In, with two outputs, Sum and Carry Out, as shown in Figure 11.

A	B	CARRY IN	SUM	CARRY 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

Figure 11. Truth table for binary full adder

The full-adder is, in essence, two half-adders connected together to take account of the extra bit carried in. The circuit for the full-adder is given in Figure 12. Again, although this is shown made up from discrete gates, it can also be done with NAND gates in the same manner as the half-adder.

If more than two bits are to be summed then the block can be repeated, with the carry out from one stage being connected to the carry in of the next stage. Finally, Figure 13 shows a couple of full-adders being used to add binary 11 and 11, giving 110; ie decimal 3+3=6.

Address Decoding

Still on the micro-processor scene, another important use of TTL combinational logic designs is in the area of address decoding. The essential problem here is to produce a signal in response to a unique pattern of bits on the micro-processor address bus. This pattern of bits is, of course, the address of the device which is being sought in order to send or receive data along the data bus of the system. Typically,

Continued on page 64

FIRST BASE *Continued from Page 47*

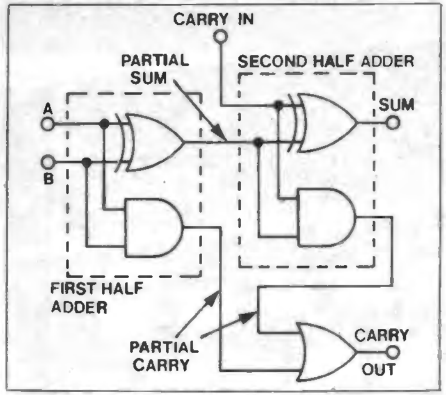


Figure 12. Full adder design

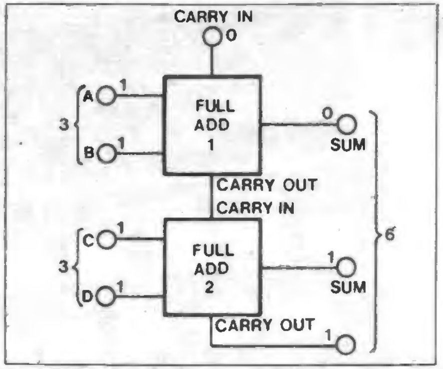


Figure 13. Full-Adders cascaded.

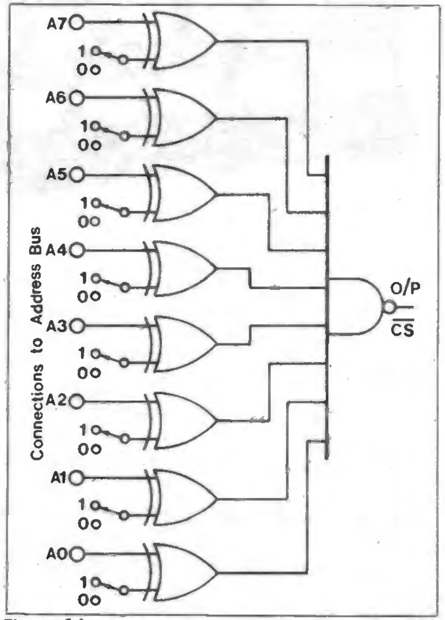


Figure 14.

there may be 16 address lines, each of which is set to either 1 or 0 according to the specific address which the micro-processor wishes to access. The address is set in response to the requirements of the controlling program or soft-ware, and the logic must ensure that only one device is enabled if data bus contention is not to arise. With 16 address lines there are 65,536 possible unique addresses, corresponding to the locations in the memory map of the system. There are a number of logic devices which have been specially devised for address decoding, but we will consider a smaller problem using devices already described.

In some systems, the lower eight address lines are used by the micro-processor for a special purpose, that of addressing input or output devices which allow information to be fed between the processor and the 'outside world'. With only eight lines the number of possible addresses is reduced to 256, which helps to bring the problem down to more manageable proportions. What is needed, then, is a logical 'black box' into which may be sent the eight address lines along with signals to set a certain address, and from which emerges one line carrying the logic signal to select the particular device being addressed.

One solution to this problem is given in Figure 14 and again this may be bread-boarded to see how it works. One input of the EX-NOR gates is connected to the address bus, and the other used to select the address of the device. The output of each EX-NOR gate is then NANDed, so that the final output goes low when the appropriate address appears on the address bus. This low signal could be connected to the 'chip select', (CS) pin of the chosen device or combined with other control bus signals for further decoding. Suppose the address of the input/output device corresponds to the following bit pattern:-

Most Significant Bit Least Significant Bit
(MSB) 1 1 0 1 0 0 1 1 (LSB)

If this pattern is set on the inputs to the EX-NOR gates then all the outputs from them will go high when the two bit patterns coincide. This in turn will set the inputs to the 8-input NAND gate all high, which is the only condition for the output to go low.

The required address may be fixed in a practical application by 'hard-wiring' the selecting inputs to the desired pattern; alternatively, the inputs may be connected via DIP switches, so that the address may be changed by altering the position of the switches.

A more convenient way of describing a bit pattern, such as the one in the above example, is to use the hexadecimal system. We shall be looking at this in more detail next time for any readers who are not familiar with the system. It will also be useful when dealing with the other main group of TTL devices, viz. those concerned with Sequential logic, which we shall also start to have a look at in the next article in the series.