Logic design

1— Boolean algebra and Karnaugh maps

by B. Holdsworth and L. Zissos *Chelsea College, University of London*

Up to 1969, when the Boolean sequential equations were developed, the design of sequential circuits was achieved through an empirical choice of unrelated informal techniques paying little attention to engineering constraints until, in most cases, the implementation stage. The advent of the sequential equations has made possible the development of clear-cut step-by-step design procedures in which realistic circuit constraints are taken into account at the design level. No engineering or other specialist knowledge is necessary to use these design procedures.

The design philosophy adopted in this series is one that allows the emphasis to be placed on optimal rather than minimal design. This is to enable technicians, users with no specialist knowledge of electronics, and the less experienced designer, to produce sound and economical designs, while at the same time providing the means whereby the specialist designer may improve his technique in dealing with more sophisticated assemblies involving such devices as r.o.ms, r.a.ms, microprocessors, and so on.

The primary design objective is to produce sound and reliable digital systems which are meaningful not only to the designer but also to the user.

Basic concepts

As in conventional algebra, so in Boolean algebra variables are combined into expressions with operators that obey certain laws. The Boolean variables, denoted by letters of the alphabet such as A,B,C etc., are binary variables and may assume one of two values, 0 or 1, or they may be alternatively read as 'false' and 'true' respectively. They are not the 'zero' and 'one' of arithmetic and the operations that can be performed on them are somewhat different and more limited than the normal arithmetical processes.

Although there exists a wide number of Boolean operators, such as NAND, NOR, etc., we need only consider three operators at this stage — all other operators can be expressed in terms of these three. They are:

- Boolean addition,
- Boolean multiplication,
- Boolean inversion.

The addition operator is written as + and may be interpreted as 'OR'. A+B may be read 'A or B' or 'A plus B'. It is true if either A is true or B is true or both are true, otherwise it is false. Thus,

$$0+0=0$$
$$0+1=1$$
$$1+1=1$$
$$1+0=1$$

The multiplication operator is written as . or ×, or omitted when its factors are variables denoted by single letters, and may be interpreted as 'AND'. A.B (or AB) may be read 'A and B' or as 'A times B'. It is true if A and B are both true, and false otherwise. Thus,

$$0\times0=0$$
$$0\times1=0$$
$$1\times1=1$$
$$1\times0=0$$

The inversion operator is written as a bar over the variable and the bar may be interpreted as "NOT". For example, $\bar{A}$ may be read as "NOT A".

$$\text{If } A=1 \quad \text{then } \bar{A}=0$$
$$\text{and if } A=0 \quad \text{then } \bar{A}=1$$

Boolean theorems

Redundancy.

$$A+AB = A$$

Fig. 1. The redundancy theorem implemented in a relay circuit. From the three relays giving $f=A+AB$ is derived the single-relay circuit giving $f=A$, since AB contains A and is therefore redundant.

Proof:
$$\begin{aligned} A+AB &= A.1+AB \\ &= A(B+\bar{B})+AB \\ &= AB+A\bar{B}+AB \\ &= AB+A\bar{B} \\ &= A.1 \\ &= A \end{aligned}$$

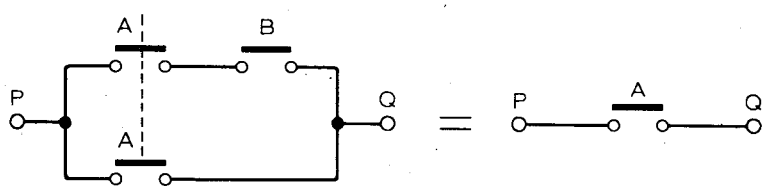
This theorem states that in a sum-of-products Boolean expression, a product that contains all the factors of another product is redundant. As a consequence it allows the elimination of redundant products in a sum-of-products expression. For example, in the Boolean function $f=AB+ABC+ABD$, the products ABC and ABD can be eliminated, because each contains all the factors present in AB.

The application of this theorem to a relay circuit is shown in Fig. 1.

Race-hazards. The main interest of the logic designer in this theorem is in its use in logic circuits for the suppression of race-hazards, which result in the generation of unwanted spikes. For example consider the Boolean function $f=AB+\bar{A}C$. Following changes in A, there is a race-hazard when $B=1$ and $C=1$, since the function then reduces to $f=A+\bar{A}$. The use of an inverter to generate $\bar{A}$ from A implies a delay between the waveforms of A and $\bar{A}$ as shown in Fig. 2. This leads to the generation of a transient signal as indicated in the same diagram.

The unwanted transient can be averted by the introduction of an optional product, that is a Boolean product whose presence in an expression does not affect the value of the Boolean function. A suitable optional product for the function $f=AB+\bar{A}C$ is formed by taking the product of the coefficients A and $\bar{A}$.

Hence, $AB+\bar{A}C = AB+\bar{A}C+BC$



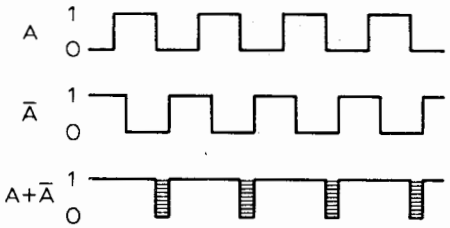


Fig. 2. A race hazard. $\bar{A}$ is obtained by inverting A and is subject to a delay, resulting in the interval during which neither $\bar{A}$ nor A is 'up.' The output $f=A+\bar{A}$ is therefore not true, or 'down' during this time.

Proof: $AB+\bar{A}C+BC$
$=AB+\bar{A}C+(A+\bar{A})BC$
$=AB+\bar{A}C+ABC+\bar{A}BC$
$=AB(1+C)+\bar{A}C(1+B)$
$=AB+\bar{A}C$

The product BC is optional so long as its parent products, AB and $\bar{A}C$ remain in the expression. Should, however, one of its parent products be eliminated (by applying the redundancy theorem), then such a product is no longer optional and cannot be removed from the expression.

If now $B=C=1$ the expression $f=AB+\bar{A}C+BC$ reduces to $f=A+\bar{A}+1$, which always has the value 1 irrespective of the values of A and $\bar{A}$.

The use of optional products will now be demonstrated with the aid of three examples.

(1) **Elimination of parent product.**

$$f=A+\bar{A}BC,$$

Form the optional product BC:

$$f=A+\bar{A}BC+BC,$$

Eliminate parent product $\bar{A}BC$ using theorem of redundancy:

$$f=A+BC.$$

(2) **Elimination of non-parent product.**

$$f=AB+\bar{A}C+BCD$$

Form the optional product BC:

$$f=AB+\bar{A}C+BC+BCD.$$

Eliminate non-parent product BCD using theorem of redundancy:

$$f=AB+\bar{A}C+BC.$$

But BC is an optional product and is redundant, hence

$$f=AB+\bar{A}C.$$

(3) **Elimination of non-parent product and parent product.**

$$f=AB+\bar{A}BC+BCD$$

Form the optional product BC:

$$f=AB+\bar{A}BC+BCD+BC.$$

Eliminate non-parent product BCD and parent product $\bar{A}BC$ using theorem of redundancy:

$$f=AB+BC.$$

De Morgan's theorem. The complement of a Boolean expression can be obtained by replacing each variable by its complement in the corresponding dual expression. For example, the dual of $f=A+BC$ is obtained by replacing the operator + by . and vice versa. Hence the dual expression is

$$f_D=A(B+C)$$

and

$$\bar{f}=\bar{A}(\bar{B}+\bar{C})$$

that this is so can be confirmed with the aid of a truth table as shown in Fig. 3. Examination of columns 8 and 10 of this table show that $\bar{A}(\bar{B}+\bar{C})$ is the complement of $A+BC$.

A	B	C	$\bar{A}$	$\bar{B}$	$\bar{C}$	BC	A+BC	$\bar{B}+\bar{C}$	$\bar{A}(\bar{B}+\bar{C})$
0	0	0	1	1	1	0	0	1	1
0	0	1	1	1	0	0	0	1	1
0	1	0	1	0	1	0	0	1	1
0	1	1	1	0	0	1	1	0	0
1	0	0	0	1	1	0	1	1	0
1	0	1	0	1	0	0	1	1	0
1	1	0	0	0	1	0	1	1	0
1	1	1	0	0	0	1	1	0	0

Fig. 3. The truth table shows that $\bar{A}(\bar{B}+\bar{C})$ is the complement of $A+BC$.

Example. Find the complement of $f=A(BC+\bar{B}\bar{C}+BCD)$.
Apply redundancy

$$f=A(BC+\bar{B}\bar{C})$$

dualise: $f_D=A+(B+C)(\bar{B}+\bar{C})$
invert: $f=\bar{A}+(\bar{B}+\bar{C})(B+C)$

$$f=A+B\bar{C}+\bar{B}C$$

Fan in. This theorem has its application in those logic circuits where there is a fan-in restriction placed on the designer by the availability of gate inputs. This matter will be dealt with more fully in a later article.

It is frequently convenient, when multiplying out two Boolean sums to refer to one section of the sum as its head, H, and to the remaining section as its tail, T. The statement of the theorem then is:

$$(H_1+T_1)(\bar{H}_1+T_2)=H_1T_2+\bar{H}_1T_1$$

Proof: l.h.s. $=(H_1+T_1)(\bar{H}_1+T_2)$
$=H_1\bar{H}_1+H_1T_2+\bar{H}_1T_1+T$
Now $H_1\bar{H}_1=0$ and T_1T_2 is redundant by theorem of race-hazards; therefore
l.h.s. $=H_1T_2+H_1T_1$

This theorem allows us to multiply out two Boolean sums, two sections of which are the complement of each other, without generating algebraically redundant products.

The partition of a Boolean sum into head and tail is arbitrary. For example in the case of the Boolean sum $A+B+C$ any of the following partitions is allowable

head	tail
A	B+C
B	A+C
C	A+B
A+B	C
A+C	B
B+C	A

Example. $f=(A+B+C)(\bar{A}+DE+F)$
Let $H_1=A$ and $T_1=B+C$
$H_2=\bar{A}$ and $T_2=DE+F$,
then $(A+B+C)(\bar{A}+DE+F)$
$=A(DE+F)+\bar{A}(B+C)$
$=ADE+AF+\bar{A}B+\bar{A}C$

If there are terms common to both of the sums to be multiplied the process of multiplication can be further simplified by noting that such terms appear in the product in their original form. For example

$(A+BC)(A+DE)$
$=AA+ADE+ABC+BCDE$
$=A+BCDE.$

Hence, if $P=(I+X)$ and $Q=(I+Y)$ where I is the common term,
then $PQ=(I+XY)$.
Finally, if $P=(H_1+T_1+I)$ and $Q=(\bar{H}_1+T_2+I)$,
then $PQ=H_1T_2+\bar{H}_1T_1+I$

Boolean reduction

A Boolean function is said to be irredundant, or reduced, if it contains no optional products. For example, the factor $\bar{A}$ in the function $f=A+\bar{A}B$ is redundant, since $A+\bar{A}B=A+B$. Redundancies in two-level Boolean expressions can be removed in three steps, using the theorems of redundancy and racehazards. If an expression contains more than two levels, it is converted into its two-level sum-of-products form by multiplying out.

The three steps for eliminating redundancies in Boolean expressions are:

(1) Multiply out.

Consider the Boolean function

$$f=BC+(AB+C)\bar{C}+A$$

Apply (1):

$=BC+AB\bar{C}+C\bar{C}+A$
$=A+BC+AB\bar{C}$

(2) Apply redundancy theorem:

In (1) the expression $f=A+BC+AB\bar{C}$ was derived. Step (2) is commenced by considering the first product, in this case A. Now scan the products to the right of A, looking for a product that contains the factor A. Here $AB\bar{C}$ is such a product and this is eliminated, resulting in $f=A+BC$. Since there are no products to the right of BC the step is not repeated.

(3) Apply theorem of race hazards:

The first variable in the first product is selected and the remainder of the expression is scanned for a product that contains the complement of the selected variable. When such a product is found, an optional product is formed using the second theorem. The optional product is used to eliminate non-parent products and/or to replace parent products as previously described. If a parent product has been replaced, the optional product is inserted at the beginning of the expression and (3) is repeated. If the optional product has not been used, it is discarded. Step (3) is repeated until all first-level optional products have been generated. Repeat (3) if necessary using higher level optional products[1]. For example:

$$f=A+\bar{A}B+BC+\bar{A}\bar{B}D.$$

Form the optional product B:

$$f=A+\bar{A}B+BC+\bar{A}\bar{B}D+B.$$

Eliminate parent product $\bar{A}B$ and non-parent product BC:

$$f=B+A+\bar{A}\bar{B}D.$$

Form optional product $\bar{A}D$:

$f = B + A + \bar{A}\bar{B}D + \bar{A}D.$

Eliminate parent product $\bar{A}\bar{B}D$:

$f = \bar{A}D + B + A.$

Form optional product D:

$f = \bar{A}D + B + A + D.$

Eliminate parent product $\bar{A}D$:

$f = A + B + D,$

which is the required result.

Minimisation

A Boolean sum-of-products expression is said to be minimal if (a) no other sum-of-products expression for the same function has fewer products, and (b) of other sum-of-products expressions for the same function with the same number of products, none has fewer factors.

There are three main methods for minimising Boolean expressions.

These are:

- The Karnaugh map method. In this method the function is displayed on a map and by suitable looping arrangements the minimal form is obtained.
- The Quine-McCluskey method[2]. In this method all irredundant forms of a given Boolean function are generated and the shortest one chosen.
- A step-by-step algebraic method[3] which does not involve expansion of the function.

In this article the Karnaugh map method will be described.

Consider the Boolean function:

$$\begin{aligned} f &= \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC + \bar{A}B \\ &= (A+\bar{A})\,BC + (A+\bar{A})\,\bar{B}C + \bar{A}B \\ &= BC + \bar{B}C + \bar{A}B \\ &= (B+\bar{B})\,C + \bar{A}B \\ &= C + \bar{A}B \end{aligned}$$

The original expression has been transformed algebraically into a simpler Boolean function which requires less hardware for implementation. Certainly in the era before the advent of the integrated circuit, minimization of Boolean functions was a positive advantage. In these days of integrated circuits the advantages of Boolean minimisation at the gate level are less obvious and the designer is now thinking in terms of minimizing the number of chips, both from the point of view of economy of space and cost. However, the formal process of simplification does lead the designer to a facility for handling Boolean equations and in that sense it is still useful.

The simplest and most widely used method of simplification employs a mapping technique. Maps for two, three, four and five variables are shown in Fig. 4, and are called Karnaugh maps.

For the two-variable map there are four cells, each of which represent one of the four possible combinations of the two variables. In the top left hand cell of the map A = 0 and B = 0, that is, the cell represents the minterm $m_0 = \bar{A}\bar{B}$, where a minterm may be defined as a Boolean product which contains all the variables in their true or inverted form. The decimal number in a cell is the decimal equivalent of the binary representation of the minterm associated with that particular cell. For example, the minterm associated with the top right hand cell of the two variable Karnaugh map is $\bar{A}B$ and its binary representation is 01 which has a decimal equivalent of 1.

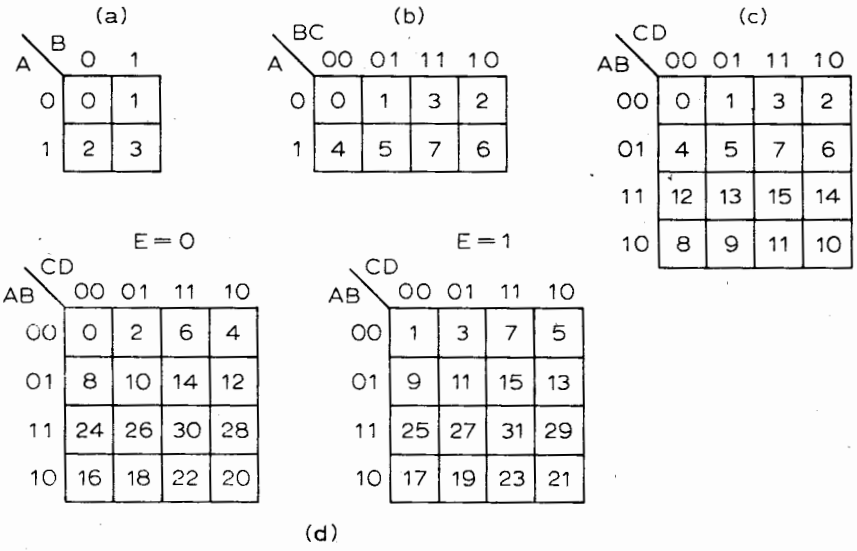


Fig. 4. Karnaugh maps for two(a), three(b), four (c) variables. In the case of five variables, two maps are needed, as shown at (d).

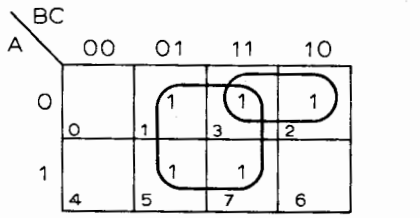


Fig. 5. The Karnaugh map for $f = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC + \bar{A}B$. The ringed '1's show that the expression can be minimized to $f = C + \bar{A}B$.

Any Boolean function of a given number of variables can be plotted on a Karnaugh map. For example, consider again the function:

$f = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC + \bar{A}B$

The first term in the expression $\bar{A}\bar{B}C$ has a binary representation of 001 and the cell corresponding to 001 on the map shown in Fig. 5 is marked with a 1. It follows that the terms $\bar{A}BC$, $A\bar{B}C$, and ABC can be plotted on the map using the same method. The remaining term $\bar{A}B = \bar{A}B(C + \bar{C}) = \bar{A}BC + \bar{A}B\bar{C}$ and the binary representation of these two terms is 011 and 010 respectively, corresponding to cells 2 and 3, but cell 3 has already been covered by the term $\bar{A}BC$ and it is only necessary to enter a 1 in cell 2 to complete the plot of the function.

The above example has shown that a 3-variable term occupies one cell only on a 3-variable map, a two variable term occupies two adjacent cells on the map and a single variable term will occupy four adjacent squares on the map. For example, the term A would be plotted in the cells marked 4, 5, 7 and 6 on the map and these four adjacent squares represent that term.

The process of simplification therefore reduces to the process of identifying plotted adjacencies on the Karnaugh map and then looping these adjacencies as shown in Fig. 5. The four-cell adjacency represents the term C and the two cell adjacency represents the term $\bar{A}B$ and the minimal function is

$f = C + \bar{A}B$

as was previously determined by algebraic methods.

Clearly to get the minimal form of the function the largest possible adjacencies should be chosen.

Example Minimize the Boolean function:

$f = BD + \bar{A}\bar{B}C + A\bar{B}\bar{C} + \bar{A}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D}.$

The function is shown plotted on the Karnaugh map in Fig. 6 and the adjacencies giving the minimal function are shown looped.

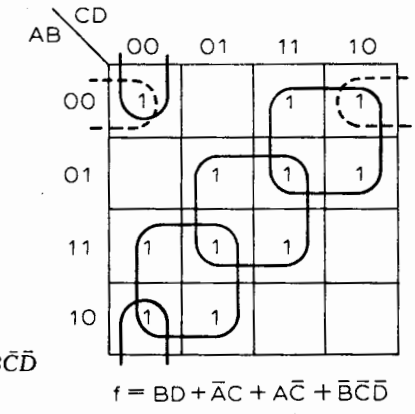


Fig. 6. Minimal function of $f = BD + \bar{A}\bar{B}C + A\bar{B}\bar{C}ABC + \bar{A}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}\bar{D}$ is shown to be $f = BD + \bar{A}C + A\bar{C} + \bar{B}\bar{C}\bar{D}$.

From the map

$$f = BD + \bar{A}C + A\bar{C} + \bar{B}\bar{C}\bar{D}$$

or $$f = BD + \bar{A}C + A\bar{C} + \bar{A}\bar{B}\bar{D}$$

Example Minimize the Boolean function shown plotted in Fig. 7.

For five-variable functions two maps are required as shown in Fig. 7 and the minimisation process can then be carried out in two steps:

Step (1): Minimize the functions plotted in the E=0 and E=1 maps as if dealing with two separate four-variable problems.

This gives $f_1 = \bar{B}\bar{D}\bar{E} + ABD\bar{E} + BCD\bar{E}$

and $f_2 = BDE + A\bar{B}\bar{D}E + A\bar{C}\bar{D}E$

Note that in this case there are two equally valid minimal solutions for the E=1 map, one of which has been chosen.

Step 2: Look for combinations between cells on the E=0 and E=1 maps which will lead to the elimination of factors from any of the terms in f_1 or f_2.

For example, the term $\bar{B}\bar{D}\bar{E}$ in f_1, may be written as $\bar{B}\bar{D}\bar{E} + A\bar{B}\bar{D}\bar{E}$ and the term $A\bar{B}\bar{D}\bar{E}$ can be combined with the term $A\bar{B}\bar{D}E$ in f_2 to generate the term $A\bar{B}\bar{D}$ thus eliminating the factor E between these two terms. The minimal sum is then given by the logical sum of f_1 and f_2 after all possible combinations have been made between the two maps. This leads to the following minimal solution.

$$f = \bar{B}\bar{D}\bar{E} + BDE + ABD + BCD + A\bar{B}\bar{D} + \bar{A}\bar{C}\bar{D}E$$

Obviously, the process of minimization using maps becomes more complicated as the number of variables in a problem increases. However, the method is readily usable up to six variables.

It was shown earlier in this article in the section on the race-hazard theorem that unwanted transient signals can be averted by the introduction of optional products. For example, for the Boolean function $f = \bar{A}B + AC$ a race-hazard occurs, following changes in A, when B=C=1, and it is eliminated by introducing the optional product BC so that the function becomes $f = \bar{A}B + AC + BC$. The original function is shown plotted in Fig. 8(a) and the new function including the optional product is plotted in Fig. 8(b).

Before the introduction of the optional product the Boolean function was irredundant in that it contained no loops, when plotted on Fig. 8(a), that are already covered by other loops. The function was also minimal. However with the introduction of the optional product a loop BC is formed which is already covered by the loops for $\bar{A}B$ and AC. The function is now no longer minimal in that it contains a redundant product BC. This example shows that the introduction of redundancy into a Boolean function is necessary to eliminate race hazards and that the minimal solution is not always the best solution.

Clearly the possibility of a race-hazard occurring can easily be spotted on a Karnaugh map plot of the Boolean function to be minimized.

The minimal form of the function shown plotted in Fig. 9(a) is $f = \bar{A}D + A\bar{B}C + ABC$ but race-hazards will occur at the places indicated by arrowheads on the map. To eliminate these race-hazards two extra loops should be added to the Karnaugh map plot as shown in Fig. 9(b) and the minimum hazard-free function becomes

$$f = \bar{A}D + A\bar{B}C + ABC + B\bar{C}D + \bar{B}CD$$

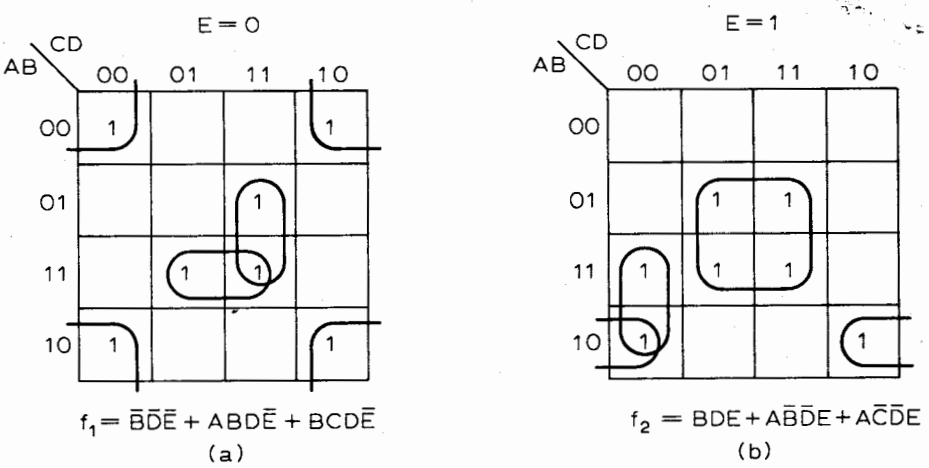


Fig. 7. A further example of minimization.

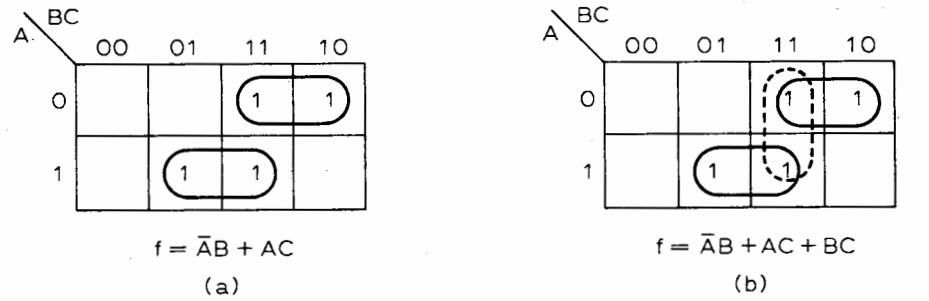


Fig. 8. The use of optional product BC in (b) eliminates the race hazard with changes in A.

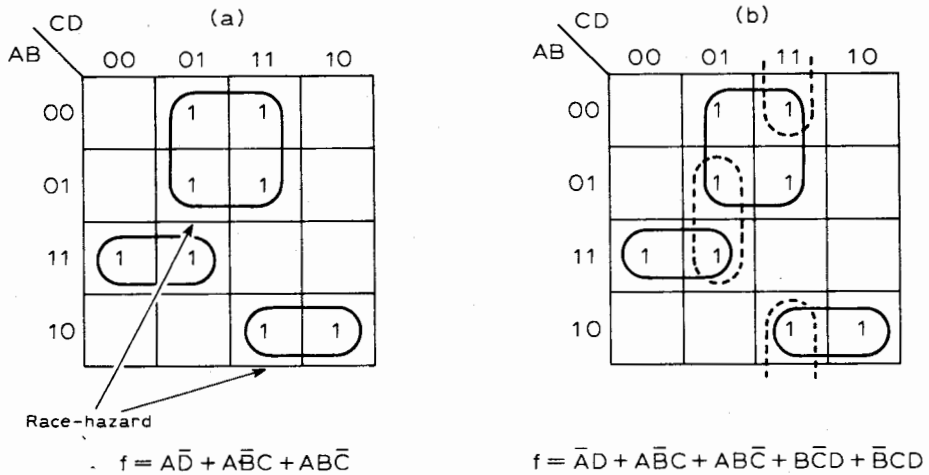


Fig. 9. More elimination of race hazards, shown by arrows in (a) by optional products shown at (b).

References

1. "Problems and Solutions in Logic Design," D. Zissos, Oxford University Press, 1976.
2. "Minimization of Boolean Functions," E. J. McCluskey, *Bell System Technical Journal*, November 1956.
3. "Boolean Minimization," D. Zissos and F. Duncan, *The Computer Journal* vol. 16, No. 2, May 1972.

Logic design — 2

Combinational logic

by B. Holdsworth* and L. Zissos†

†Department of Computing Science, University of Calgary, Canada.
*Chelsea College, University of London

Two of the most essential features that must be met in the design of logic circuits are the imposed gate *fan-in restrictions* and *hazard-free operation*. Gate fan-in is the number of imput terminals provided in a gate, i.e. the maximum number of input signals to a gate. Race-hazards are unwanted transient signals (signal spikes), which under certain changes of an input signal and with certain relationships of circuit delays appear in a logic circuit.

Combinational circuits can be constructed using AND, OR and INVERTER gates, NOR gates or NAND gates. It is possible to construct curcuits using all of the above elements but such circuit configurations are not, at present, common. Circuits composed entirely of NAND or entirely of NOR gates are generally more economical and convenient to use than circuits using AND, OR and INVERTER gates.

The truth table for a two-input NAND gate is shown in Fig. 1(a) and that of a two-input NOR gate in Fig. 1(b). A NAND gate can be used as an INVERTER if all except one of the inputs are tied to logic 1, a practice which, though not always necessary, is strongly advised. For example, if the input A of the gate shown in Fig. 1(a) is tied to logic 1, then the output of the gate is $\bar{B}$ as indicated by the entries in the bottom two rows of the truth table.

Similarly a NOR gate can be used as an INVERTER if all except one of the inputs are tied to logic 0. The remaining input then appears inverted at the output of the gate. In the case of the gate shown in Fig. 1(b), if input A is connected to logic 0 the output of the gate is $\bar{B}$, as indicated by the entries in the top two rows of the truth table.

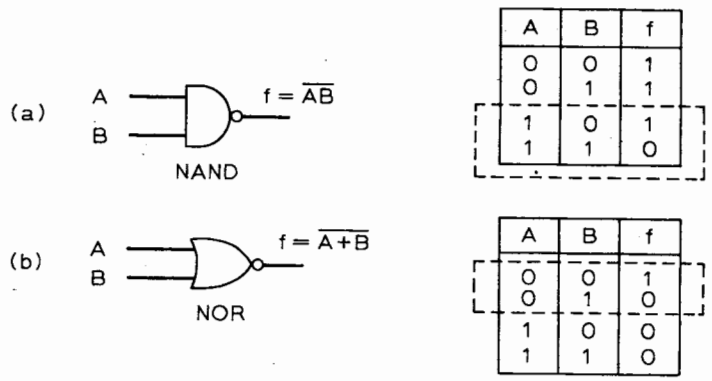
NAND and NOR gates can also be used to generate the OR and AND functions. For example, the output of a NAND gate driven by signals $\bar{A}$ and $\bar{B}$ is $\bar{A}\,\bar{B}$, which may be written as A + B, as shown in Fig. 2(a). The AND function can be generated by connecting two NAND gates in cascade, the first one generating the NAND function of the two input variables A and B, whilst the second gate acts as an INVERTER, as shown in Fig. 2(b). It follows that a NOR gate fed with inverted variables generates the AND function of the true values of the input variables, whilst two NOR gates in cascade generate the OR function of the variables fed to the inputs of the first gate.

Two levels of NAND gates generate a two-level sum-of-products expression, as shown in Fig. 3(a), which indicates the one-to-one relationship that exists between a sum-of-products expression and its NAND implementation. The reader's attention is drawn to the fact that the realisation of a minimal sum-of-products expression does not necessarily result in a minimal circuit. For example, the implementation of the "Exclusive OR" function $f = A\bar{B} + \bar{A}B$, which is a minimal expression, requires five gates, if inverted variables are not available as shown in Fig. 3(b), whereas the NAND circuit satisfying its non-minimal form $f = A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$ requires one gate less, as shown in Fig. 3(c).

In order to implement a function, such as $f = (\bar{A} + BC)E + (\bar{G} + \bar{H})F$ using NAND gates, it is simpler to work backwards from the output gate. The equation is of the form PQ + RS, where

$$P = (\bar{A} + BC) \qquad R = F$$
$$Q = E \qquad S = (\bar{G} + \bar{H})$$

This type of two-level sum-of-products has already been realised in Fig. 3 (a) and is repeated with the relevant input signals in Fig. 4(a). The input line $\bar{G} + \bar{H}$ to gate 3 is the output line of a two-input NAND gate, whose inputs are found by inverting the variables $\bar{G}$ & $\bar{H}$. Similarly, the input line $\bar{A}$ + BC to gate 2 is the output line of a two-input NAND gate, whose inputs are found by inverting the variable $\bar{A}$ and the product BC, as



A	B	f
0	0	1
0	1	1
1	0	1
1	1	0

A	B	f
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 1. Symbols and truth tables for NAND (a) and NOR (b).

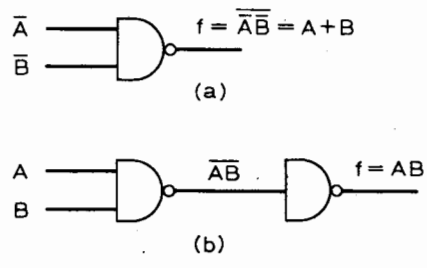


Fig. 2. The OR function using a NAND gate at (a) and the AND using NANDs at (b).

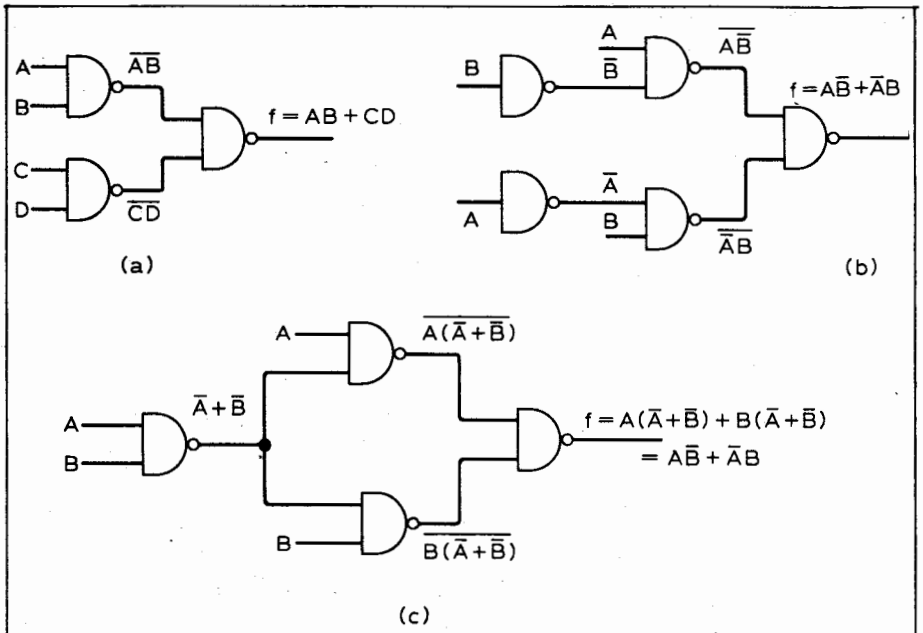


Fig. 3. The use of NAND gates to obtain a sum-of-products function (a). The minimal form of expression need not give a minimal circuit; minimal expression $f = \bar{A}\,\bar{B} + \bar{A}\,B$ in (b) needs one more gate than non-minimal expression $f = A\,(\bar{A} + \bar{B}) + B\,(\bar{A} + \bar{B})$ at (c).

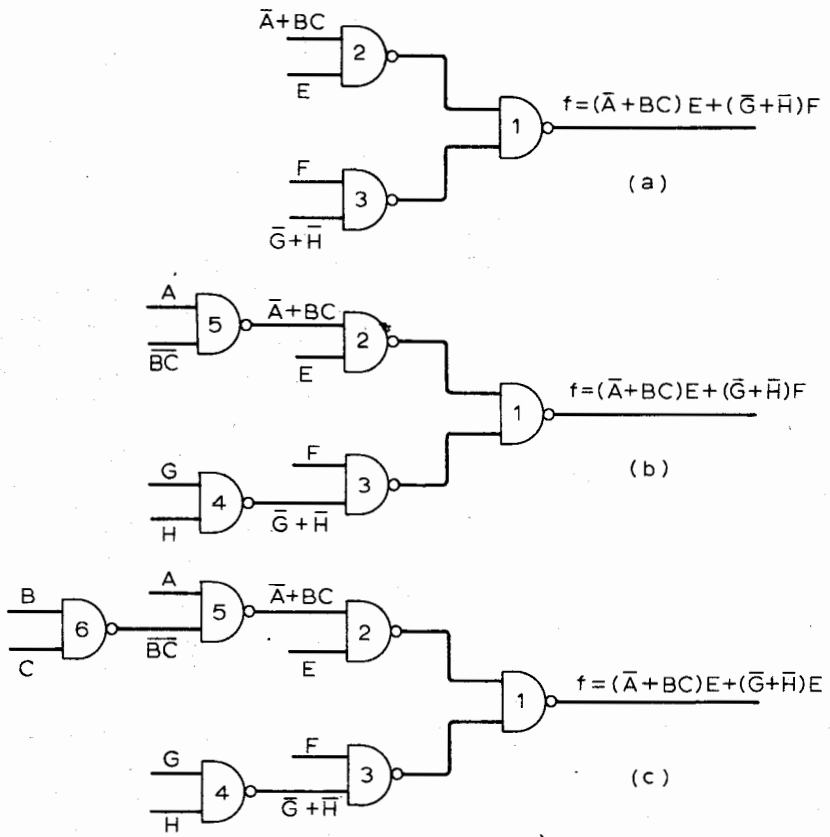


Fig. 4. Building up the expression $f = \bar{A} + B\,C)E + (\bar{G} + \bar{H})F$ from the output end in three levels.

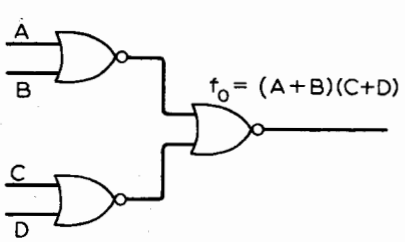


Fig. 5. Dualizing $f = A\,B + C\,D$ with NOR gates.

shown in Fig. 4(b). For the final stage in the implementation it is only necessary to precede gate 5 with a two-input NAND gate whose input variables are B and C as shown in Fig. 4(c).

If the NAND gate in Fig. 3(a) were replaced by NOR gates as shown in Fig. 5 the output function, which the reader can check for himself, will be

$$f_D = (A + B)\,(C + D)$$

which is the dual of the output function of the circuit shown in Fig. 3(a). Hence to implement the NOR circuit of a Boolean function first derive the NAND-circuit of the dual function and replace the NAND gates by NOR gates.

Example. Implement the function

$$f = \bar{A}\bar{B}\bar{C} + ABC.$$

Dualise:

$$f_D = (\bar{A}+\bar{B}+\bar{C})\,(A+B+C)$$

Express in Sum-of-Products form:

$$f_D = \bar{A}B + \bar{A}C + A\bar{B} + \bar{B}C + A\bar{C} + B\bar{C}$$

minimising using the method of Part 1:

$$f_D = \bar{A}B + \bar{B}C + A\bar{C}.$$

The NAND circuit of this function is shown in Fig. 6(a) and the NOR function $f = \bar{A}\bar{B}\bar{C} + ABC$ is given by replacing the NAND gates by NOR gates, as shown in Fig. 6(b).

Hazard-free operation

Race-hazards are unwanted transient signals (signal spikes) which, under certain changes of an input signal and with certain relationships of circuit delays, appear in a logic circuit. The NAND circuit of Fig. 7 shows a combinational logic circuit in which "spikes" are generated during a change of input signal A from 1 to 0 when B = C = 1. The cause of the race-hazard is that immediately following a change in the signal A, $A = \bar{A}$ = either 0 or 1. Hence if a Boolean expression of a signal in a circuit reduces to either $A + \bar{A}$ or $A\bar{A}$, a race-hazard exists at the output of the corresponding gate, otherwise the signal is hazard-free.

In the example shown in Fig. 7, $f = AB + \bar{A}C$ reduces to $A + \bar{A}$ when B = C = 1, revealing the existence of a race-hazard at the output of gate 4. Race-hazards in a circuit can be suppressed by preventing its Boolean expression from reducing to either $A + \bar{A}$ or $A\bar{A}$. This is achieved by the application of the theorem of race-hazards in Part 1. Hence

$$AB + \bar{A}C = AB + \bar{A}C + BC$$

or, alternatively, expressing the same function as a product-of-sums

$$(\bar{A} + B)\,(A + C) = (\bar{A} + B)\,(A + C)\,(B + C).$$

The introduction of the third term prevents the first expression from being reduced to $A + \bar{A}$, since when B = C = 1, $\bar{A}B + AC + BC$ now reduces to $A + \bar{A} + 1 = 1$. Similarly, the second expression, when B = C = 0, reduces to $(\bar{A}+0)\,(A + 0)\,(0 + 0) = \bar{A}.A.0 = 0$.

Fan-in restrictions

The implication of a fan-in restriction (the number of gate inputs) on the realisation of a Boolean function is equivalent to imposing a restriction on the maximum size of the products and sums in the expression of the function to be satisfied. For example the direct realisation of the function $f = AB + A\bar{C} + A\bar{D}$ shown in Fig. 8 requires one three-input NAND gate, three two-input NAND gates and two single-input NAND gates, six gates in all.

If the fan-in restriction is two, implying the use of two-input NAND gates, there are two possible methods of

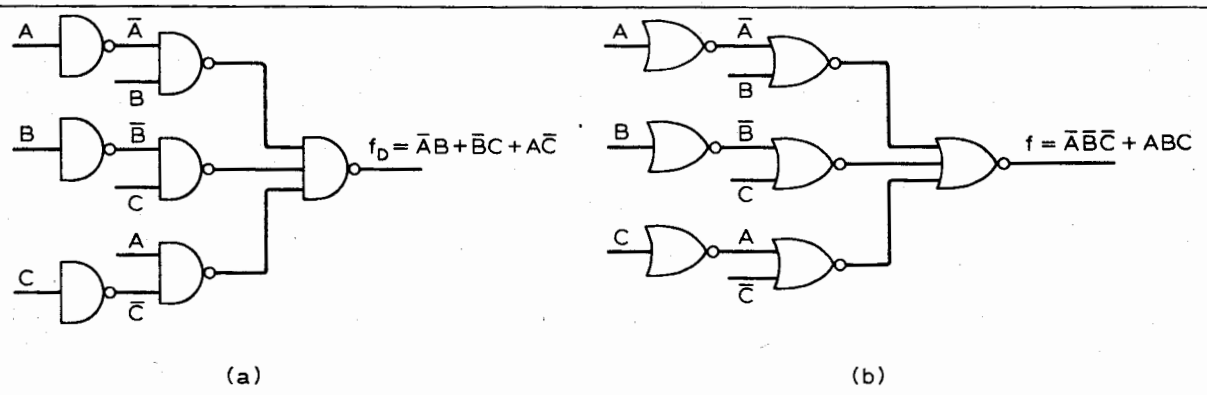


Fig. 6. Generating a function using NOR gates. Function $f = \bar{A}\,\bar{B}\,\bar{C} + A\,B\,C$ is first dualized, minimized and implemented in NAND logic, as at (a). This circuit is then converted to NOR gates to provide the required output.

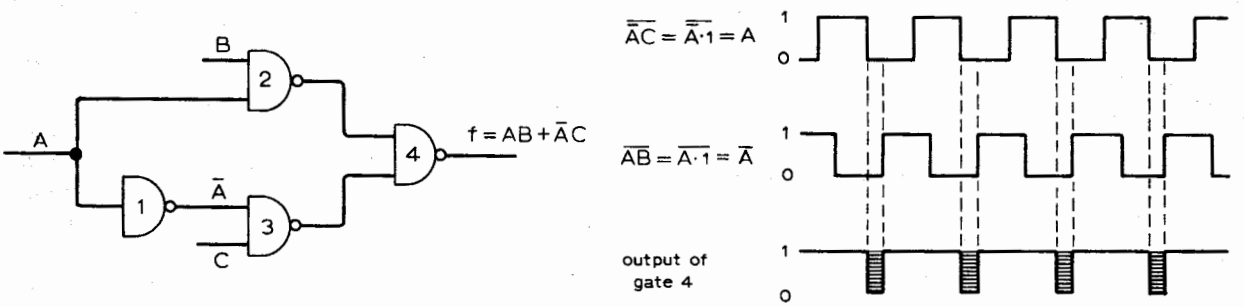


Fig. 7. Mechanism of "spike" generation.

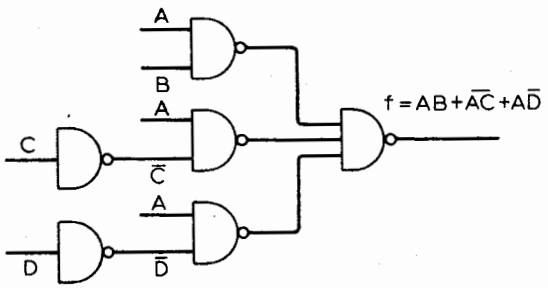


Fig. 8. "Direct" generation of $f = A\,B + A\,\bar{C} + A\,\bar{D}$ when 3-input gates can be unused.

rearranging the given function to satisfy this restriction.

Method 1: bracket two of the three products.

The function is $f = AB + A\bar{C} + A\bar{D}$
bracketing: $f = (AB + A\bar{C}) + A\bar{D}$.
The implementation of this function is shown in Fig. 9(a). It meets the fan-in restriction of two but it requires eight gates, two more than in Fig. 8.

Method 2: remove a common factor:

The function can then be written
$$f = AB + A(\bar{C} + \bar{D}).$$
The realisation of this function is shown in Fig. 9(b). It meets the fan-in restriction of two and requires only four gates, two less than in Fig. 8. Alternatively the function may be written
$$f = A(B + \bar{C}) + A\bar{D}.$$
The implementation of this function is shown in Fig. 9(c). Again it meets the fan-in restriction of two and it requires the same number of gates as realised in Fig. 8. There is one further factorization of interest and that is
$$f = A(B + \bar{D}) + A\bar{C}$$
but this function has the same form as $f = A(B + \bar{C}) + AD$ and can be implemented with six NAND gates, the same number as in Fig. 8. Obviously the optimal implementation is given when the function is written in the form $f = AB + A(\bar{C} + \bar{D})$ even if a fan-in restriction of two had not been imposed.

A systematic method can be used to arrive at an optimal expression for a logic function which to be realised using gates with a specified fan-in. The method described is based on the use of the merging table [1,2].

For the case of NAND circuits the starting point is the irredundant sum-of-products expression of the function to be implemented.
$$f = AB + A\bar{C} + A\bar{D}$$
The function is dualised and the brackets numbered:
$$f_D = (A + B)^1(A + \bar{C})^2(A + \bar{D})^3$$
Next the change in the gate count ΔN, which occurs when pairs of brackets are merged is determined with the aid of the merging table shown in Fig. 10, which has been developed for the case when, there is no increase in the size of the sum ($\Delta Z = 0$) upon merging brackets.

Merging is the process described in the Fan-in theorem in the first article of this series, where two brackets are replaced by a single bracket i.e.
$$(H_1 + T_1)(\bar{H}_1 + T_2) = H_1T_2 + \bar{H}_1T_1$$
It is essential to note that merging does not affect terms which are present in both brackets i.e.
$$(I + X)(I + Y) = I + XY$$
To determine the value of ΔN the components of the two brackets are counted in the following manner.

x= the number of terms in the smaller bracket, excluding common terms.

y= the number of terms in the larger bracket, excluding common terms.

r= the number of terms in the head section of the smaller bracket.

$n = 1$ if a group of terms in one bracket, called the head, is the complement of a group of terms in the other, otherwise $n = 0$.

l= the number of variables true or inverted counted in x and y.

t= the number of true variables in x and y such that for each

(1) its complement does not occur as a variable in any of the other brackets.

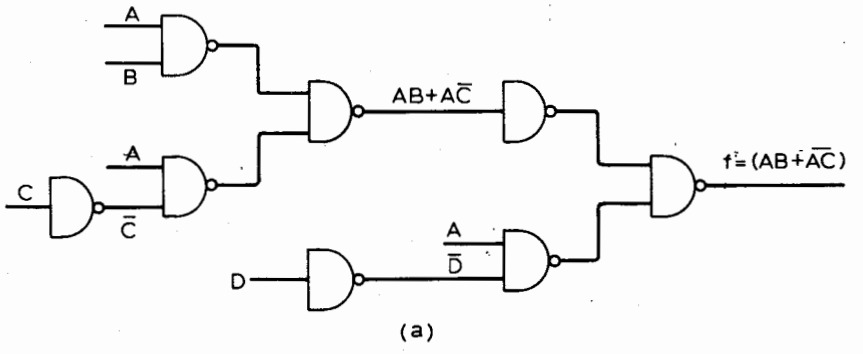
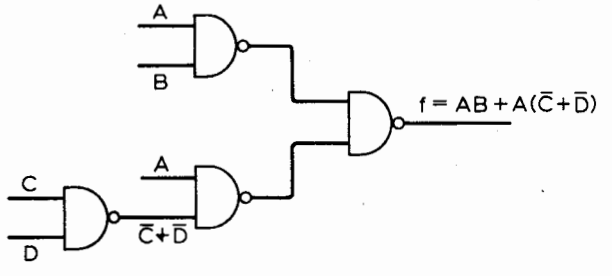
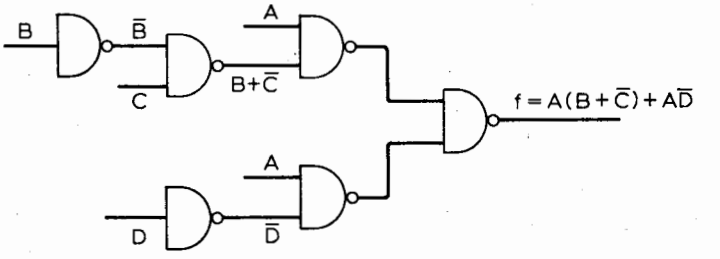


Fig. 9. Bracketing two products in $f = A\,B + A\,\bar{C} + A\,\bar{D}$ enables use of 2-input gates but requires eight instead of six, as in (a). Removing a common factor again meets fan-in restriction to 2 inputs, with varying savings in number of gates, as seen in (b) and (c).





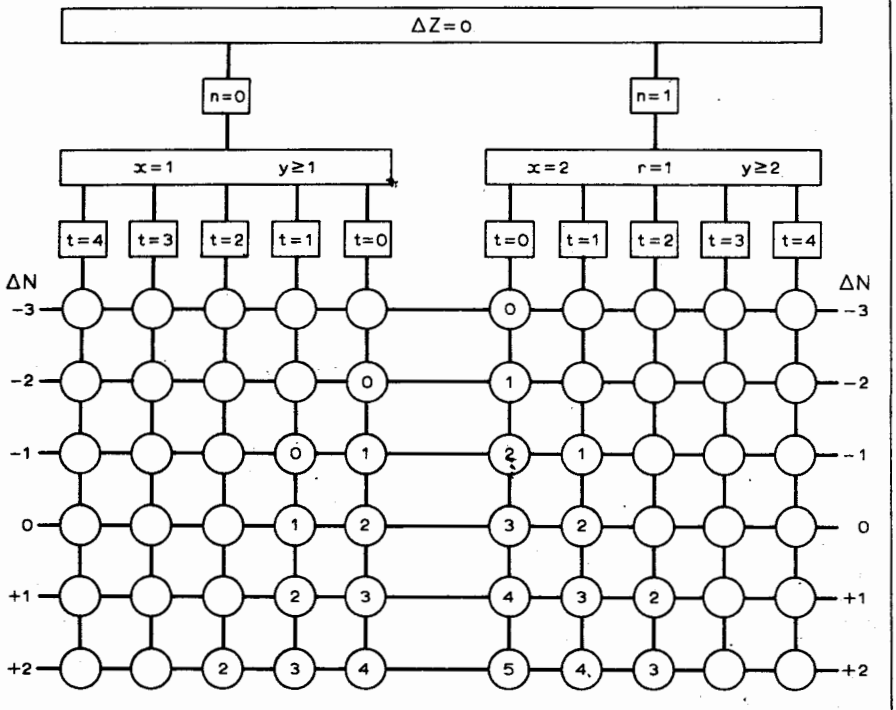


Fig. 10. Merging table for $\Delta Z = 0$.

(2) it does not occur in its true form in a product within the expression.

i= the number of inverted variables such that for each

(1) it is not repeated in the expression as an inverted variable

(2) it does not occur in its true form in a product within the expression.

N is the gate count and ΔN is the change in the value of N caused by merging two brackets.

The quantities detailed above are tabulated below for each bracket pair of the dual function, ΔN being obtained from the table of Fig. 10.

$$f_D = (A+B)^1(A+\bar{C})^2(A+\bar{D})^3$$

b/p	n	x	y	r	t	l	i	l-i	ΔN
1/2	0	1	1	–	1	2	1	1	0
1/3	0	1	1	–	1	2	1	1	0
2/3	0	1	1	–	0	2	2	0	−2

The above tabulation shows that merging brackets 1 and 2 or brackets 1 and 3 does not result in a change in the gate count but that merging brackets 2 and 3 gives a reduction in the gate count by 2, which is the same result obtained working directly with the circuits in Fig. 9.

Merging 1/2 gives $f_D = (A+B\bar{C})\,(A+\bar{D})$
redualising: $f = A(B+\bar{C})+A\bar{D}$
see Fig. 9 (c)
Merging 1/3 gives $f_D = (A+BD)\,(A+\bar{C})$
redualising: $f = A(B+\bar{D}) + A\bar{C}$
Merging 2/3 gives $f_D = (A+\bar{C}\bar{D})\,(A+B)$
redualising: $f = A(\bar{C}+\bar{D})+AB$
see Fig. 9(b).

This part will be concluded with two examples, the first one demonstrating the process of minimal design using the merging table and the second one demonstrating the development of a minimal, hazard-free design.

Example 1 Design a minimal two-input NAND circuit to realise the following Boolean function.

$$f = AB+\bar{A}\bar{C}+C\bar{D}$$

This equation is already in its minimal form.

Dualise: $f_D = (A+B)^1(\bar{A}+\bar{C})^2(C+\bar{D})^3$

Attempt merging:

b/p	n	x	y	r	t	l-i	ΔN
1/2	1	2	2	1	2	4−2=2	+1
1/3	cannot be merged						
2/3	1	2	2	1	1	4−3=1	−1

Merging 2 and 3 will result in a

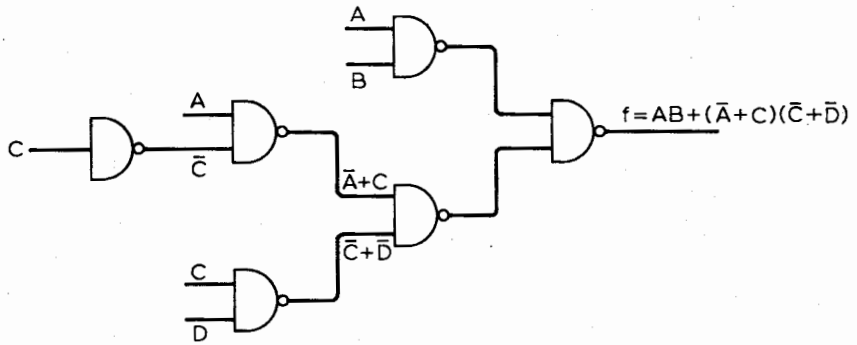


Fig. 11. Minimal circuit for $f = A\,B\,(\bar{A} + C)(\bar{C} + \bar{D})$, using the merging operation.

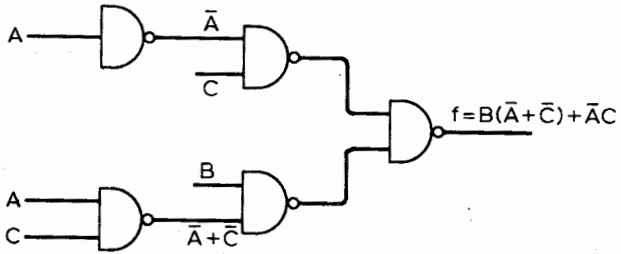


Fig. 12. Two-input NANDS used to realise $f = B\,(\bar{A} + \bar{C}) + \bar{A}C$, which is hazard-free form of $= \bar{A}C + B\bar{C}$.

reduction of the gate count by 1
Merge 2, and 3: $f_D = (A+B)(\bar{A}C+C\bar{D})$.
Redualise: $f = AB + (\bar{A}+C)(\bar{C}+\bar{D})$
Implement as in Fig. 11.

Example 2. Under what circumstances will a spike be generated at the output gate if a direct NAND implementation of the function $f=\bar{A}C+B\bar{C}$ is made? Derive an equivalent hazard-free expression that can be implemented minimally using two-input NAND gates.

If A=0 and B=1 the function $f=\bar{A}C+B\bar{C}$ reduces to $f=A+\bar{A}$ which is the condition for generating a spike when C changes from 1 to 0.

The hazard-free expression is $f = \bar{A}C+B\bar{C}+\bar{A}B$
Dualise: $f_D = (\bar{A}+C)^1(B+\bar{C})^2(\bar{A}+B)^3$

Attempt merging:

b/p	*n*	*x*	*y*	*r*	*t*	*l-i*	ΔN
1/2	1	2	2	1	2	4−1=3	+2
1/3	0	1	1	–	1	2−0=2	+1
2/3	0	1	1	–	0	2−1=1	−1

Merge 2 and 3: $f_D = (B+\bar{A}\bar{C})(\bar{A}+C)$
Redualise: $f = B(\bar{A}+\bar{C}) + \bar{A}C$
Implement, as in Fig. 12.

References

1. Logic Design Algorithms, D.Zissos, Oxford University Press,1972.
2. "Fan-in Restrictions in Logic Circuits," D. Zissos and F. G. Duncan,*Proc. I.E.E.*,Vol. 118, No. 2, Feb. 1971.

Logic design — 3

Event-driven circuits

by B. Holdsworth* and D. Zissos†

A four-step procedure based on the sequential equations, for the design and implementation of event-driven logic circuits is described in this article. Realistic circuit constraints are automatically taken into account by the design process.

The principal factors to be considered in the design of event-driven circuits are:

- Circuit reliability. The circuit must operate correctly and reliably.
- Gate fan-in and fan-out restrictions. These must be observed.
- Speed tolerances. Gate speed tolerances of ± 33⅓ per cent are automatically accommodated by the design process used.

Generally speaking, the solutions obtained do not necessarily use a minimum number of gates, but the design requires minimum effort. The design steps are easy to apply and require no specialist knowledge.

State diagrams

State diagrams can be used to describe both the external and internal operations of event-driven sequential circuits. In a state diagram, states can be represented by squares, rectangles or circles and lines linking the states represent transitions between states. The direction of a transition is indicated by an arrow pointing in the direction of the destination state, and the signal condition that initiates the transition is indicated by its Boolean function inserted either above or below the line. For example the part of a state diagram shown in Fig. 1 indicates that the circuit moves from state S_0 to S_1 when $XY = 1$, i.e. when $X = 1$ and $Y = 0$.

The external and internal-state diagrams of a circuit in which the activation of a switch X in Fig. 2(a) operates, in turn, two lights L_1 and L_2 are shown in Figs. 2(b) and 2c) respectively. Variables X_n and X_{n+1} are used to indicate the n^{th} and $(n+1)^{th}$ activation of the switch. The external-state diagram closely resembles a flow chart, which can be drawn with very little regard to circuit implementation.

There are no hard-and-fast rules for developing internal-state diagrams. Since such diagrams describe the internal operation of a circuit, the designer usually makes arbitrary choices depending on past experience, his understanding of the problem and availability of components, which can lead to different but equivalent results.

The following example is used to illustrate typical variations in the internal-state diagrams of the relatively simple light circuit shown in Fig. 3(a). The function of the circuit is to turn lamp L_1 on when the two switches X and Y are made in that order, and lamp L_2 on when the switches are made in the reverse order. Two different but correct versions of the internal operation of the circuit are shown in Figs. 3(b) and 3 (c).

Most persons attempting this problem would probably derive internal-state diagram 3(b) which uses five internal states. State S_1 is used to record that switch X has been made and state S_3 that switch Y only has been made. In both cases there is no change in the circuit output, although clearly there is a change in the internal-state of the circuit. Very few designers, if any, would arrive at Fig. 3(c) the first time round.

As might be expected the circuit implementation of the state diagram of Fig. 3(c) is the simplest. This state diagram can be obtained by constructing a state table from the state diagram shown in Fig. 3(b), the state table then being reduced by the application of Caldwell's merger procedure. This technique will be described later in this article.

State variables

Each state of an event-driven logic circuit is defined by a unique combination of logic signals called state variables or secondary signals. Clearly one state variable A, defines two states, one by $A = 0$ and the other by $A = 1$. Two state variable define four circuit states each state corresponding to a combination of their values, i.e. 00, 01, 11, and 10. In general, n variables will define 2^n circuit states. As an example of state allocation, the states S_0, S_1, S_2 and S_3 in Fig. 2(c) can be defined by the state variables $AB = 00$, 01, 11, 10. In allocating state variables to states in event-driven circuits it is necessary to ensure that each circuit transition involves a change in the value of a single variable only. The reasons for

* Chelsea College, University of London
† Dept of Computing Science, University of Calgary, Canada

Fig. 1. Elements of a state diagram.

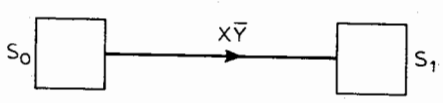
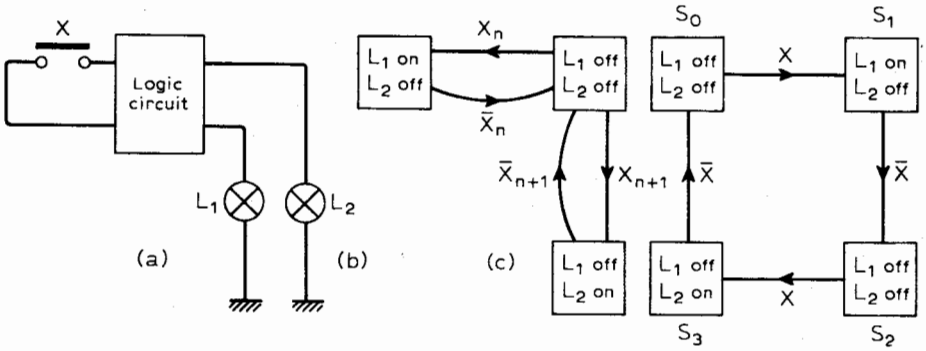


Fig. 2. Internal and external state diagrams of a logic circuit.



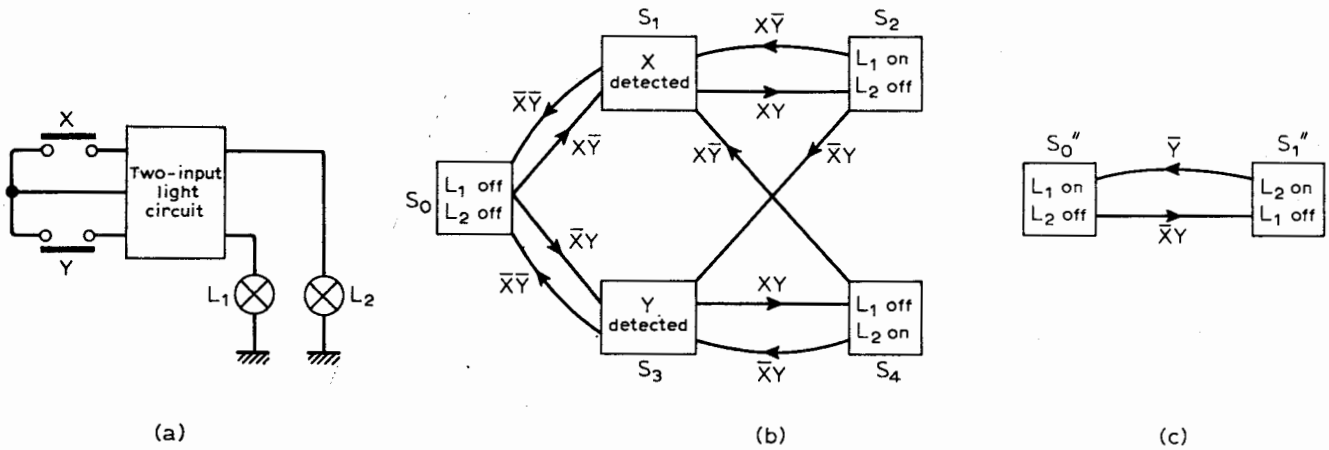


Fig. 3. Internal state diagrams of a two-switch logic circuit.

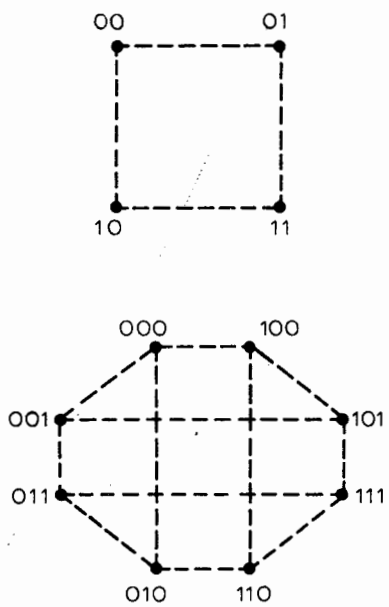


Fig. 4. Race-free diagrams for two and three variables.

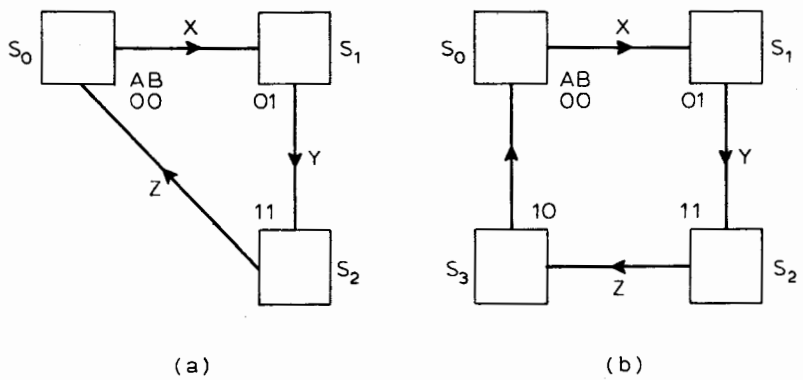


Fig. 5. Use of a dummy state (S_3) avoids simultaneous two-variable change from S_2 to S_0.

doing this is to ensure that races between state variables or secondary signals are automatically avoided.

A race-free assignment of states can be achieved with the aid of a race-free diagram. This is a two-deminsional diagram containing 2^n coded nodes, where all nodes whose codes differ in one variable only are joined by interrupted lines. Hence, races between secondary signals are automatically avoided if each circuit transition lies on a race-free line. Race-free diagrams for two and three variables are shown in Fig. 4.

Dummy states

There are certain patterns of internal-state diagrams that cannot be assigned race-free codes. Such a pattern is shown in Fig. 5(a). If the state codes for S_0, S_1, and S_2 are AB= 00, 01, and 11 respectively the direct transition from state S_2 to S_0 cannot be implemented as this would involve the simultaneous change of two variables. In this case the link between S_2 and S_0 can be turned into a race-free link by interposing a fourth state S_3 between S_2 and S_0, coded A=1 and B=0. This is called a dummy state and replaces line S_2–S_0 with race-free links S_2–S_3 and S_3–S_0, as shown in Fig. 5(b). The S_3–S_0 transition is unconditional and once the circuit assumes state S_3 it moves automatically to S_0. In terms of state variables this ensures that signal B is turned off first and this automatically turns signal A off.

Unused states.

If the number of states to be implemented is N, where $2^{n-1}<N<2^n$, there will be 2^n-N unused or redundant states. For example, in the case of the three-state diagram shown in Fig. 5(a) there will be one unused state. It can never be assumed in practice that a circuit will not move into an unused state either when switching the circuit on or due to the interference of a noise signal. For example, when in state $S_0=00$, a noise signal may turn A on and the circuit enters the unused state $S_3=10$. The circuit may be operating in conjunction with other circuits and moving into state $S_3=10$ may result in the incorrect behaviour of the overall system.

The designer is therefore strongly advised to take such a possibility into account at the design level and take the necessary action. For example, if the misoperation of the above circuit can result in the jamming of a production

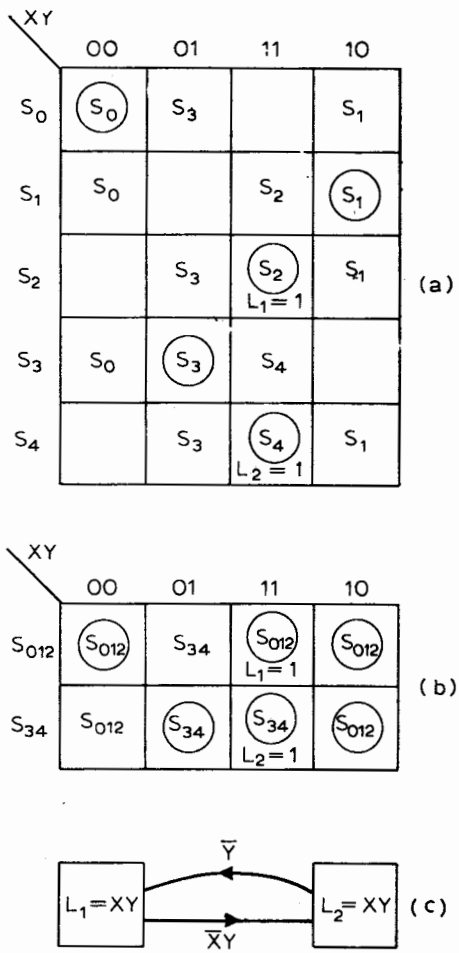
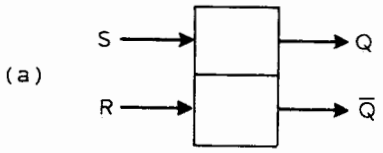


Fig. 6. State-table reduction.



(b)

	S^t	R^t	Q^t	$Q^{t+\delta t}$	
1	0	0	0	0	
2	0	0	1	1	
3	0	1	0	0	
4	0	1	1	0	
5	1	0	0	1	
6	1	0	1	1	
7	1	1	0		Forbidden combinations
8	1	1	1		

Fig. 7. SR flip-flop and its truth table.

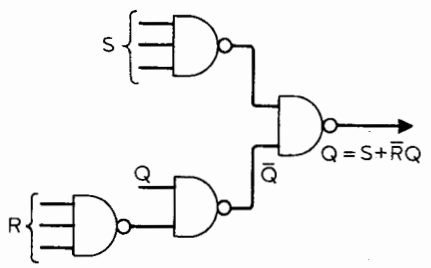


Fig. 8. Implementation of the NAND sequential equation for S and R primary signals.

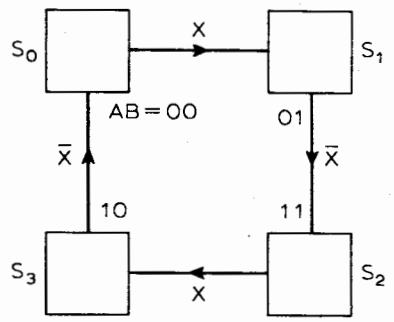
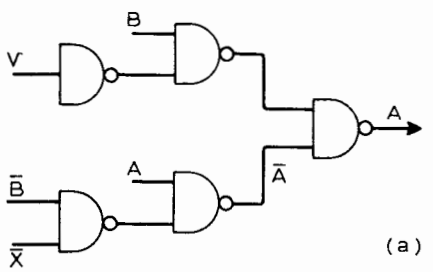
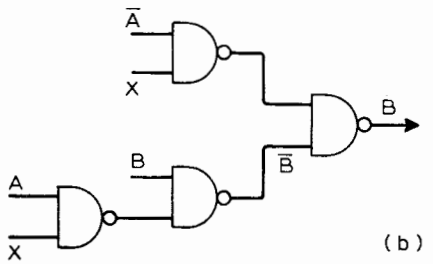
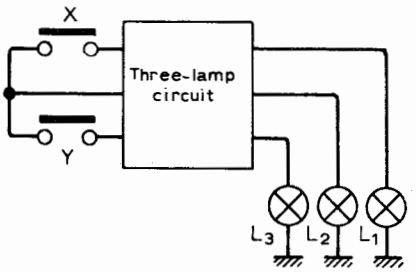


Fig. 9. Determination of turn-on and turn-off sets.







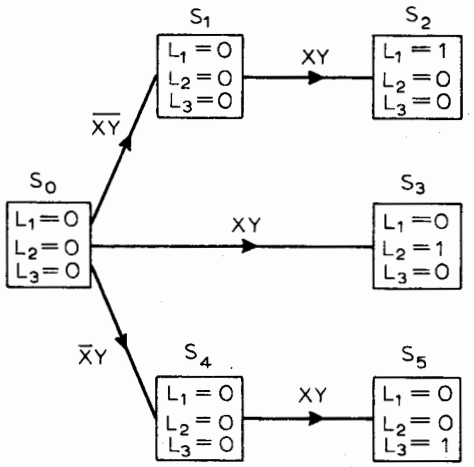


Fig. 11. Three-lamp circuit and its state diagram.

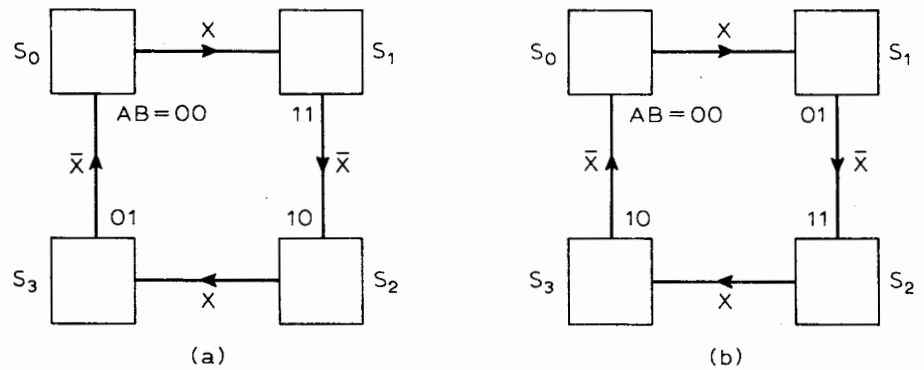


Fig. 12. Elimination of races between secondary signals.

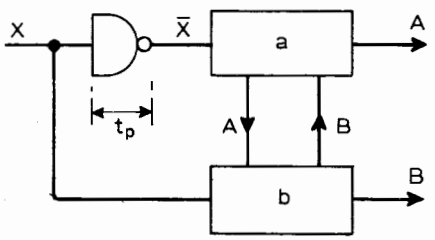


Fig. 13. Races between primary and secondary signals.

line, a possible action would be to use the signal $A\overline{B}$, which defines the unused state, to turn off all machines and raise an alarm.

Frequently, such states are referred to as "don't-care" states. Boolean expressions defining the "don't care" conditions are used as optional products to reduce the circuit equations and hence the complexity of the circuit. This is based on the assumption that a "don't care" condition does not arise in practice, which is only valid for normal operation. Since one cannot exclude the possibility of circuit misoperation, the designer is strongly advised not to leave undefined the response of the circuit under such conditions. In other words the designer "cares" about all circuit conditions.

Summarizing, no state diagram containing other than 2^n states should be implemented. Referring to the light circuit of Fig. 3, only the state diagram in Fig. 3(c) can be implemented. The implementation of the state diagrams in Fig. 3(b) would require the addition of three states. Additionally the reader is strongly advised against the mathematically convenient use of "don't care" states for circuit simplification.

Fig. 10. Implementation of NAND sequential equations for turn-on and turn-off sets obtained from state diagram of, for example, Fig. 9.

State tables

The design restriction of always implementing 2^n states can be met either by introducing dummy states or by reducing the number of internal states. State reduction is carried out by using Caldwell's merging procedure which is based on the state table. Such a table has a row for every state of the circuit and a column for every combination of the input signals. The rows and columns are headed by labels representing the corresponding states and inputs. In each cell the circuit destination is entered, i.e. the next state assumed by the circuit when it is in a state corresponding to the row heading, and it receives input signals defined by the column heading. If the designer does not wish to specify the next state the entry in the appropriate cell is left blank. A second entry is made in each cell which specifies the circuit output,

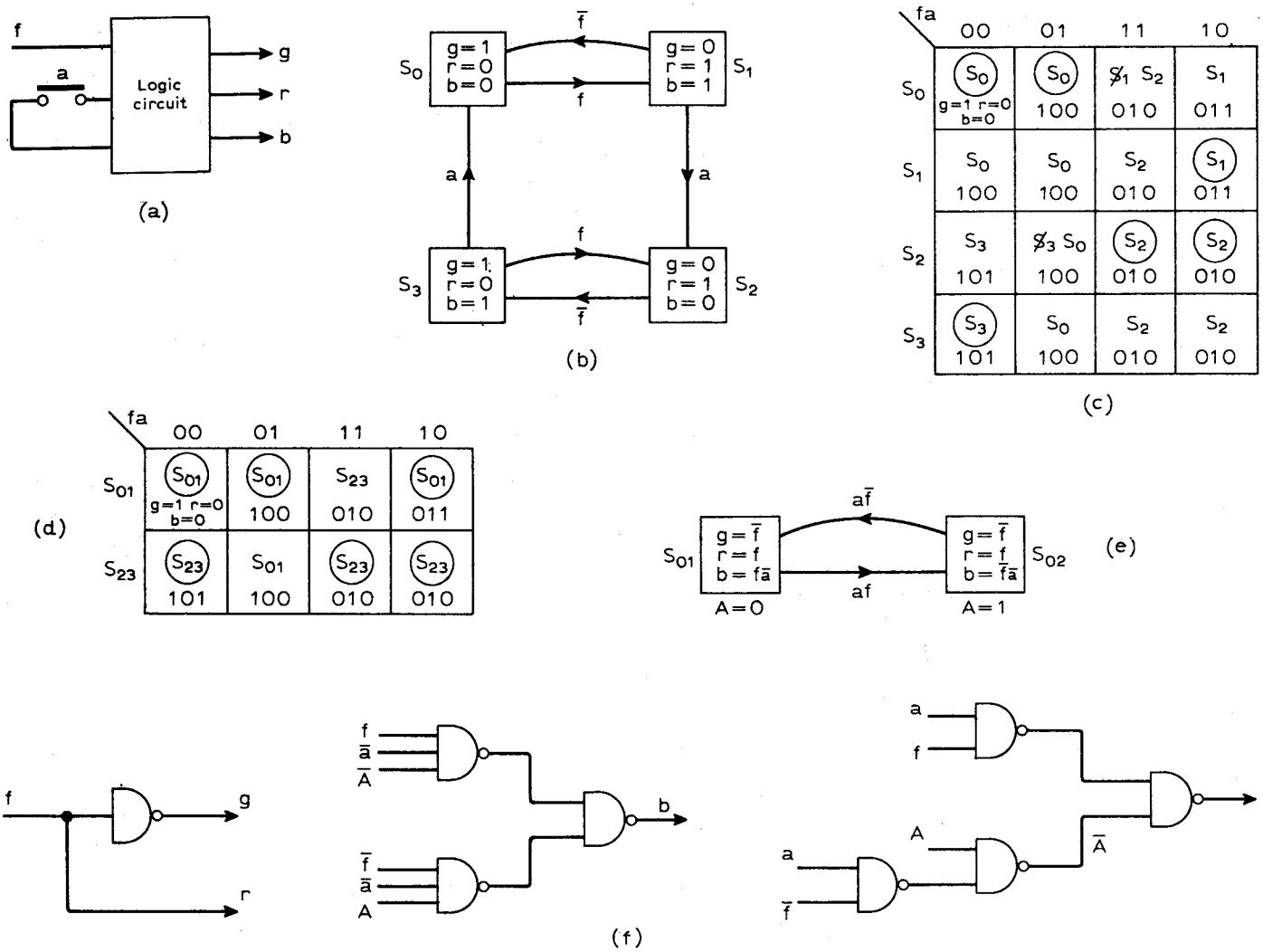


Fig. 14. Function to be realized in Example 1 is at (a) and its state diagram is at (b), while the state table is shown in (c) and in merged form at (d). Initial state diagram based on (d) is shown at (e) and realization of the circuit is (f).

unless it is a blank cell. If the circuit destination is the same as its current state, the circuit is stable and the entry is encircled. The state table corresponding to Fig. 3(b) is shown in Fig. 6(a).

State reduction

The process of combining the rows of a state table is made in accordance with Caldwell's merging rules.

Two rows may be merged if the state numbers and the circuit outputs appearing in corresponding columns of each row are alike, or if the entry in one or both of the rows is blank.

When circled and uncircled entries of the same state number are to be combined, the resulting entry is circled. Thus the two rows

3 ⑤
3 5 6 ⑧

combine into 3 ⑤ 6 ⑧

A change of state from 5 to 8 now involves a change of the input state only.

When a row S_m is combined with row S_n the new row is marked S_{mn}.

Examination of Fig. 6(a) indicates that rows S_0, S_1, and S_2 can be merged to give a new row S_{012} and also that rows S_3 and S_4 can be merged to give a new row S_{34}. The reduced state table is shown in Fig. 6(b) with its corresponding state diagram in Fig. 6(c), and this is identical to the state diagram of Fig. 3(c).

Sequential equations

The sequential equations, allow a state diagram to be translated directly into a circuit, as a consequence, lead to a much simpler solution of event-driven circuit problems. These equations can be obtained directly from a consideration of the logical behaviour of an SR flip-flop.

The SR flip-flop is shown symbolically in Fig. 7(a), the set and reset inputs being labelled S and R respectively, whilst the complementary outputs are labelled Q and $\bar{Q}$. The truth table for the flip-flop is shown in Fig. 7(b).

In the first three columns of this table, all combinations of the present states of S, R, and Q, i.e. their states at time t, are tabulated. In the fourth column the next state of the flip-flop, i.e. its state at time $t + \delta t$, is tabulated.

Examination of this table shows that a change of flip-flop state occurs in rows 4 and 5 only. In row 4 the flip-flop is being reset, i.e. its state is being changed from 1 to 0, by the application of inputs S=0 and R=1. In row 5 the flip-flop is being set, i.e. its state is being changed from 0 to 1, by the application of inputs S=1 and R=0. The reader should also notice that with this type of flip-flop it is inadmissible for S and R both to be logical 1 simultaneously. This restriction can be expressed algebraically as SR=0.

One form of the sequential equations is obtained by taking the logical sum of the combinations in the truth table for which $Q^{t+\delta t}=1$ and adding in the product SR=0. This does not affect the value of $Q^{t+\delta t}$ but leads to a simpler equation for it.

Hence:

$Q^{t+\delta t}=(\bar{S}\bar{R}Q+S\bar{R}\bar{Q}+S\bar{R}Q+SR)^t$.

Minimizing:

$Q^{t+\delta t}=(S+\bar{R}Q)^t$.

The second form of the sequential equations is obtained by excluding the product SR from the equation for $Q^{t+\delta t}$ so that

$Q^{t+\delta t}=(\bar{S}\bar{R}Q+S\bar{R}\bar{Q}+S\bar{R}Q)^t$.

Minimizing:

$Q^{t+\delta t}=[(S+Q)\bar{R}]^t$.

Time is inferred in these equations and they are written

$$Q=S+\bar{R}Q$$

and $$Q=(S+Q)\bar{R}$$

where S is referred to as the turn-on set of Q and R is referred to as the turn-off set of Q.

The most general form of the equations is:

$$Q = \Sigma \text{ turn-on sets of } Q + Q(\Sigma \overline{\text{turn-off sets of } Q})$$

and

$$\overline{Q} = (\Sigma \text{ turn-on sets of } Q + Q)(\Sigma \overline{\text{turn-off sets of } Q})$$

The first of these two equations is used when the design is to be implemented with NAND gates and the second equation when NORs are to be used.

The implementation of the NAND sequential equation, $Q = S + \bar{R}Q$, is shown in Fig. 8. In this circuit S and R, the turn-on and turn-off signals respectively, are the primary signals, whilst Q is the secondary signals which is turned either on or off by the primary signals. When designing an event-driven logic circuit the turn-on and turn-off sets are derived directly from the state diagram; for example, by reference to Fig. 9.

Turn-on set of $A = B\bar{X}$
Turn-off set of $A = \bar{B}\bar{X}$
Turn-on set of $B = \bar{A}X$
Turn-off set of $B = AX$

Substituting these values in the NAND equations

$A = B\bar{X} + A(B + X)$
$B = \bar{A}X + B(\bar{A} + \bar{X})$

and the implementation of these equations is shown in Fig. 10.

Causes of misoperation

Circuit misoperation is said to occur when a circuit assumes an internal state other than the one intended. For example, if on leaving state S_1 in Fig. 11, with $X = 1$ and $Y = 1$ it assumes a state other than S_2, circuit misoperation occurs. Excluding component failure, the causes of circuit misoperation in event-driven circuits are races between primary signals, secondary signals or both. The above three causes will be examined in turn and solutions suggested in the next article.

Logic design — 4

Causes of malfunction in event-driven circuits

by B. Holdsworth* and D. Zissos†

*Chelsea College, University of London †Department of Computing Science, University of Calgary, Canada.

In the last article, the procedure needed for the design of event-driven logic circuits was discussed. This second half of that article goes on to describe the causes of misoperation in such circuits and concludes with some examples of design. It is unfortunate that some of the diagrams concerned with this half of the article appeared in the first half — for this, we apologise.

Races between primary signals. The circuit shown in Fig. 11 is required to operate three lamps L_1, L_2, and L_3, according to the following specifications.

(1) Lamp L_1 is to turn-on when both X and Y are operated, but only if switch X is operated before switch Y.

(2) Lamp L_2 is to turn-on when both input switches are operated simultaneously.

(3) Lamp L_3 is to turn-on when both X and Y are operated, but only if switch Y is operated first.

In practice, a logic circuit responds with different speeds to changes in the input signals. Hence the response time of the circuit to a change in the input signal X must be assumed to be different from the response time to a change in Y. As a consequence the circuit, instead of assuming state S_3 on leaving state S_0, either assumes state S_2, if the circuit responds to the change in X first, or alternatively it enters state S_5, if the circuit responds to a change in Y first. In both cases the circuit operation is not according to specification.

Since there is no remedy to this problem the circuit constraint applied is that only one input signal is allowed to change at a time.

Races between secondary signals. In the internal state diagram shown in Fig. 12(a), the coding of the internal states is such that circuit transitions S_0 to S_1 and S_2 to S_3 involve the change of more than one secondary signal. In practice because of variations in the response times of the two secondary signals to a change in the input signal X from 0 to 1, either A or B will change first.

Assuming that A changes first the circuit, when it leaves S_0, first enters S_2. From state S_2, because X = 1, the circuit assumes state S_3 instead of S_1, and this a stable state for X = 1. This is clearly incorrect operation of the circuit. Obviously a similar analysis of the circuit operation can be performed for the case when B changes faster than A.

The solution to this problem is to ensure that each circuit transition involves the change of one secondary signal only and a race-free assignment of the state variables should be used as described earlier in this article and as shown in Fig. 12(b).

Races between primary and secondary signals. A circuit implementation of Fig. 12(b) is shown in block schematic form in Fig. 13. The letters a and b are assigned to the two sections of the circuit which generate the secondary signals A and B.

Consider the transition from S_0 to S_1 in Fig. 12(b). This transition will take place in the time t_s, which it takes to turn-on the secondary signal B. It will also be assumed that the time taken to invert the primary signal X is t_p. If $t_p > t_s$ the following sequence of events will take place.

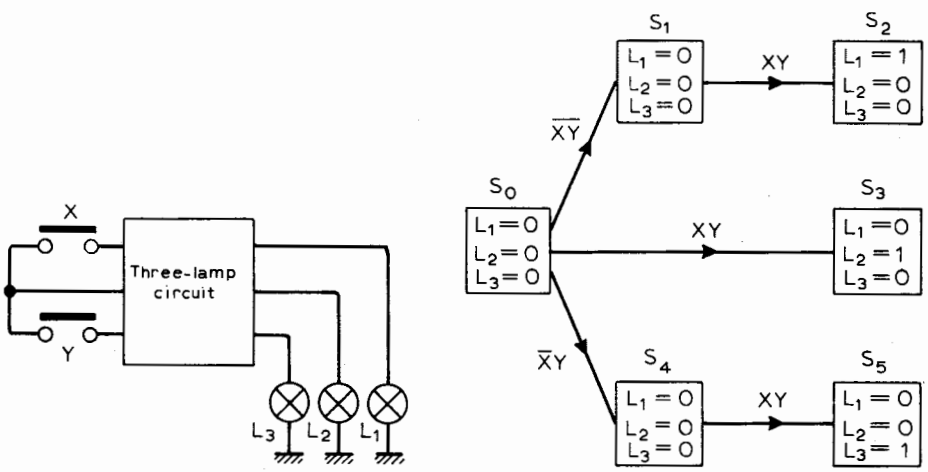
(1) At time t_s, B changes to 1 and the circuit assumes state S_1.

(2) Since $t_p > t_s$, $\overline{X} = 1$, and the condition for turning A on exists.

(3) A turns on causing the circuit to move to state S_2.

(4) On assuming state S_2, the circuit

Fig. 11. Three-lamp circuit and its state diagram.



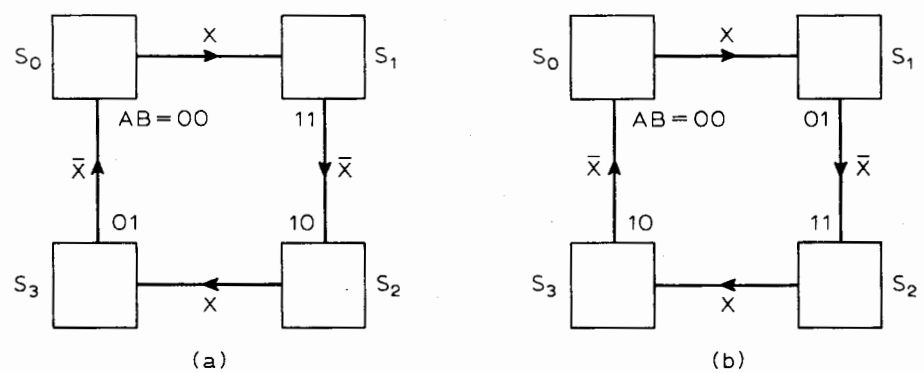


Fig. 12. Elimination of races between secondary signals.

moves to state S_3, since $X=1$.

If $t_p < t_s$ on assuming state S_1 the input signal to section a has already changed, i.e. $\bar{X}=0$, and the circuit remains in state S_1.

Unlike the previous two cases, elimination of races between primary and secondary signals cannot be achieved, since a change in a primary signal initiates a change in a secondary signal. Therefore to avoid circuit misoperation it is necessary to ensure that $t_p \leqslant t_s$. It follows that incorrect circuit behaviour will not occur if the maximum delay associated with a primary signal t_{pmax}, is less than the minimum delay associated with a secondary signal t_{smin}.

Hence

$$\frac{t_{pmax}}{t_{smin}} \leqslant 1$$

The 33⅓% property

The sequential circuits designed with the aid of the sequential equations are hazard-free when implemented with gates whose maximum speed tolerance is ±33⅓%. The justification for this statement is as follows.

The maximum delay by which a primary signal in primitive sequential circuits can be delayed is one gate delay, t_g, when it has to be inverted. Allowing

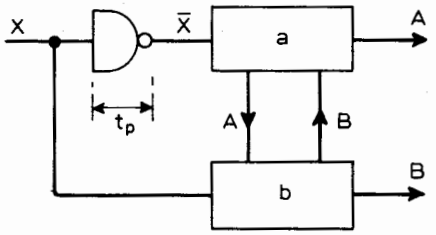


Fig. 13. Races between primary and secondary signals.

x% variation due to production spread, loading etc. $t_{pmax} = t_g(1+x)$.

The minimum delay associated with a secondary signal is $2t_g$, since at least two levels of switching are involved, as an examination of the NAND sequential equation $Q = S + \bar{R}Q$ will show. Allowing x% variation, $t_{smin} = 2t_g(1-x)$.

Substituting these values in the equation developed in the last section gives $t_g(1+x)/2t_g(1-x) \leqslant 1$ for correct circuit behaviour. The reader should observe that this property is valid for

Fig. 14. Function to be realized in Example 1 is at (a) and its state diagram is at (b), while the state table is shown in (c) and in merged form at (d). Initial state diagram based on (d) is shown at (e) and realization of the circuit is (f) Output of r.h. circuit is a.

circuits in which the sequential equations are implemented in their primitive form. Algebraic manipulation of the sequential equations will lead to a modification of the relative delays of the primary and secondary signals and therefore invalidate the 33⅓% property. Hence, processing of the sequential equations is not advised.

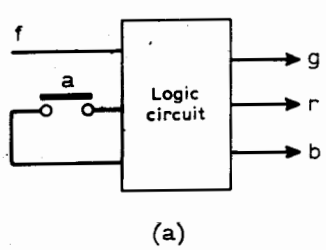
Design steps

Step 1. Draw a block diagram showing the available input signals and the required output signals.

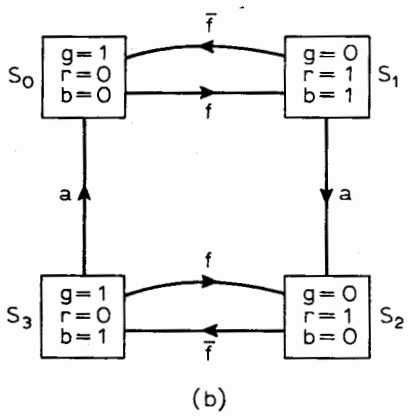
Step 2. Draw a state diagram describing the internal performance of the circuit.

Step 3. This step is optional and can be omitted. Its purpose is to provide the designer with a means of reducing the number of internal states obtained in Step 2, if such a reduction is possible or desirable.

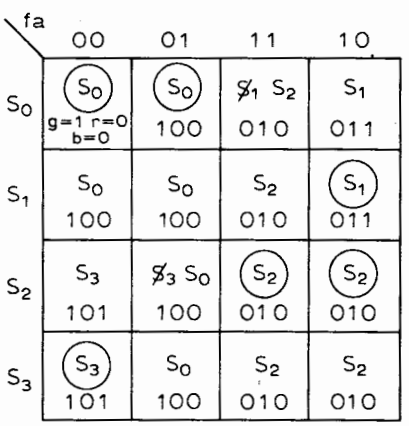
Step 4. With the aid of a race-free diagram if necessary, each internal state is given a unique code. From the coded state diagram the turn-on and turn-off sets for the secondary signals are obtained and these are used to derive the primitive sequential equations. Expressions are also obtained for the output signals. The implementation of these equations is the required circuit.



(a)

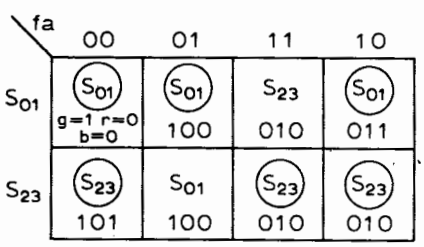


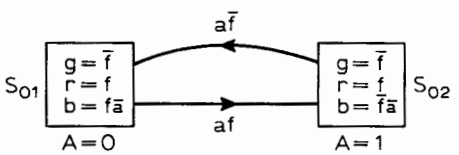
(b)



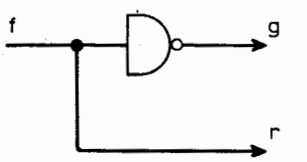
(c)

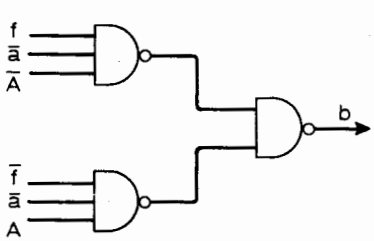
(d)

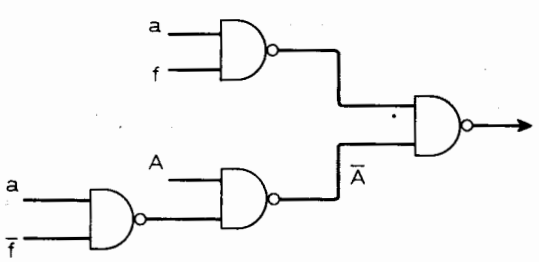




(e)







(f)

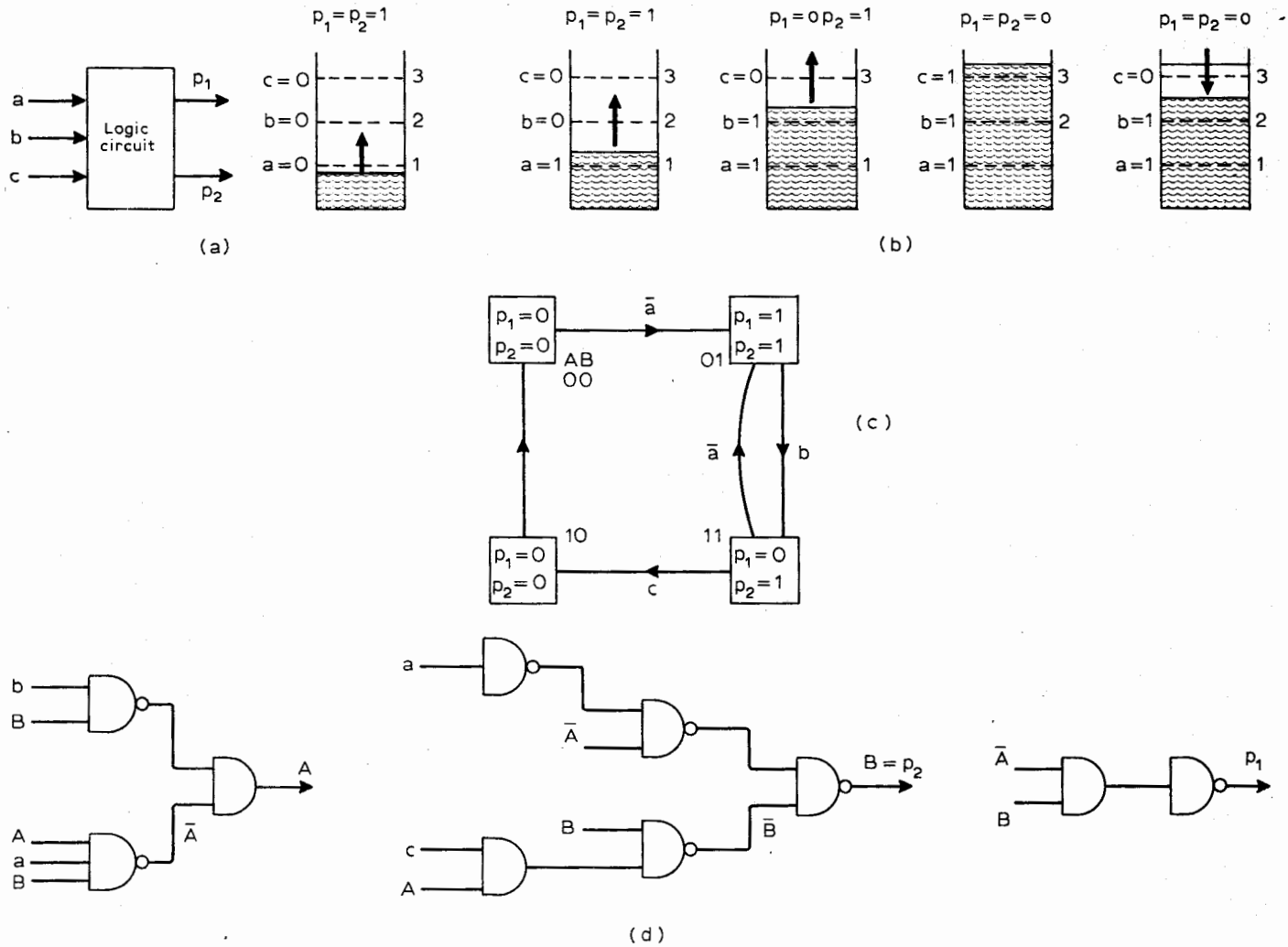


Fig. 15. Function required by Example 2 is seen in (a) and (b). State diagram in (c) provides turn-on and turn-off sets for use in NAND realization. Circuit is shown in (d).

The design procedure will now be applied to the solution of two problems.

Example 1

Design a fault detector with the following terminal characteristics. The appearance of a fault signal f activates an alarm bell, turns a green light off and a red light on. The operator turns off the bell by pressing an acknowledge button a. When the fault is cleared, the red light turns off, the green light turns on and the bell is reactivated to attract the operator's attention. The bell is turned off when the operator presses the acknowledge button. Should the fault clear before the operator has responded, the circuit is to reset. Also if a fault reappears before the operator has responded the green light turns off, the red light turns on and the bell turns off.

Step 1. See Figs. 15(a) and (b)

Step 2. A suitable state diagram is shown in Fig. 15(c).

Step 3. The state table corresponding to Fig. 14(b) is shown in Fig. 14(c).

Applying Caldwell's merging rules to the state table in Fig. 14(c), states S_0 and S_1 can be merged to form state S_{01} and states S_2 and S_3 can be merged to form state S_{23}. The reduced state table is shown in Fig. 14(d).

The internal state diagram based on the reduced state table is shown in Fig. 14(e).

Step 4. By direct reference to Fig. 14(e) the turn-on and turn-off sets are: Turn-on set of $A = af$. Turn-off set of $A = a\bar{f}$. Therefore the NAND circuit equation for A is

$$A = af + Aa\bar{f}$$
$$A = af + A(\bar{a} + f)$$
$$g = \bar{f}$$
$$r = f$$
$$b = f\bar{A}\bar{a} + \bar{f}A\bar{a}$$

The corresponding circuit is shown in Fig. 14(f).

Example 2

Water is pumped into a water tower by two pumps p_1 and p_2, where p_1 is an auxiliary pump used for boosting purposes. Both pumps are to turn on when the water goes below level 1 and are to remain on until the water reaches level 2, when pump p_1 turns off and remains off until the water is below level 1 again. Pump p_2 remains on until level 3 is reached when it also turns off and remains off until the water falls below level 1 again.

Level sensors are used to provide level detection signals as follows:

Signal $a = 1$ when the water is at or above level 1, otherwise $a = 0$.

Signal $b = 1$ when the water is at or above level 2, otherwise $b = 0$.

Signal $c = 1$ when the water is at or above level 3, otherwise $c = 0$.

Develop a sequential logic circuit to control the pumps p_1 and p_2 according to the specification given above.

Step 1. See Figs. 15(a) and (b)

Step 2. A suitable state diagram is shown in Fig. 15(c).

Step 3. It is left as an exercise for the reader to draw the state table and examine the possibility of state reduction.

Step 4. By direct reference to Fig. 15(c) the turn-on and turn-off sets are:

Turn-on set of $A = bB$

Turn-off set of $A = \bar{B} + \bar{a}B$
$= \bar{B} + \bar{a}$

Turn-on set of $B = \bar{a}\bar{A}$

Turn-off set of $B = cA$

Therefore the NAND circuit equations are:

$$A = bB + A(\overline{\bar{B} + \bar{a}})$$
$$= bB + AaB$$
$$B = \bar{a}\bar{A} + (\bar{c} + \bar{A})B$$
$$p_1 = \bar{A}B$$
$$p_2 = \bar{A}B + AB$$
$$= B$$

The corresponding circuit is shown in Fig. 15(d).

Article 5 of the series will be a discussion of clock-driven circuits.

Logic design — 5

Clock-driven circuits

by B. Holdsworth* and D. Zissos†

* Chelsea College, University of London † Dept of Computing Science, University of Calgary, Canada

A four-step algorithm for the design of clock-driven (synchronous) sequential circuits is described. Realistic circuit constraints are automatically taken into account by the design process.

The main features to be considered in the design of clock-driven circuits are reliably correct functioning, observation of gate fan-in and fan-out restrictions and ease of maintenance. It is desirable that maintenance engineers should understand the circuit even though it has undergone simplification – a process which can obscure its function. In general the circuits obtained do not use a minimum number of gates, but the design effort is minimal. The design steps are easy to apply and do not require any specialist knowledge.

Functionally the essential characteristic of synchronous sequential circuits is that their operation is synchronised with clock pulses between which no changes of state can occur.

Clocked flip-flops

Clock driven circuits depend on the use of clocked flip-flops, the principal types of which are described in this section. A clocked flip-flop is a bistable element in which the change of the output signal Q is coincident with either the leading or trailing edge of a pulse signal, commonly referred to as the clock pulse. There are four basic types of flip-flop. Toggle or T flip-flop (TFF); SR flip-flop (SRFF); JK flip-flop (JKFF); D flip-flop (DFF).

Toggle flip-flop. The flip-flop is represented symbolically by the diagram in Fig. 1(a). It has no data input terminals and physically its output "toggles" or changes state with every clock pulse. The logical behaviour of this flip-flop is described by the truth table shown in Fig. 1(b). If the T flip-flop is a modified master/slave JK flip-flop it will turn-on when $Q=0$ and C is changing from 1 to 0, that is on the trailing edge of the C-pulse. Similarly it will turn-off when $Q=1$ and C is changing from 1 to 0. The terminal behaviour of this flip-flop is described by the state diagram shown in Fig. 1(c).

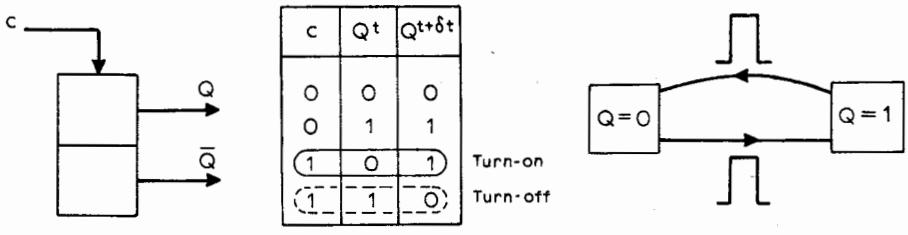


Fig. 1. Symbol (a), truth table (b) and state diagram for a toggle or T-type flip-flop.

SR flip-flop. The sequential equation, $Q=S+\bar{R}Q$, for the SR flip-flop, shown symbolically in Fig. 2(a), was developed in Part 3 of this series. An implementation of an unclocked SR flip-flop, using NAND gates, is shown in Fig. 2(c), and this is frequently drawn in the form shown in Fig. 2(d). A condensed form of the truth table for this flip-flop, called the steering table, is shown in Fig. 2(b) where the entry Φ in the S and R columns means that the input can be either 0 or 1.

By means of the simple modification shown in Fig. 2(e) the SR flip-flop can be clocked. An examination of this diagram shows that if $C=0$ the outputs of g_1 and g_2 will always be logical 1 irrespective of the present values of S and R, or of any changes in these two inputs. The flip-flop can only change its output during a clock pulse transition and, assuming zero gate delay, the output Q will change state on the leading edge of a clock pulse, when C is changing from 0 to 1.

Examination of the steering table or the circuit shows that a clocked SR flip-flop is turned on when $S=1$, $R=0$, and C changes from 0 to 1. Conversely it is turned off when $S=0$, $R=1$, and C is changing from 0 to 1. Hence the terminal behaviour of the flip-flop can be described with the aid of the state diagram shown in Fig. 2(g).

Besides the S, R and C inputs, a clocked SR flip-flop may have one or two additional controls which allow it to assume one of its two states irrespective of whether $C=0$ or $C=1$. These controls are frequently called Clear and Preset. Most commercially-available flip-flops are provided with a clear control, whereas the preset control is not nearly as common. The operation of these controls is described by the table shown in Fig. 2(h) and it should be observed that in the circuit of Fig. 2(f) these signals are active when low.

With both controls at logical 1 the flip-flop is enabled and operates in the normal way. If $R=0$ and $P=1$ the output $\bar{Q}$ of g_4, in Fig. 2(f) becomes $\bar{Q}=1$. Hence $Q=0$, and the flip-flop is unconditionally reset. If $R=1$ and $P=0$ the output Q of g_3 becomes $Q=1$, and the flip-flop is now preset. The inclusion of these controls leads to a modified state diagram as shown in Fig. 2(i).

The reader should note that if a preset facility is required when the P terminal is not provided it is possible to interchange the Q and $\bar{Q}$ terminals and the input terminals. The clear terminal can then be used as a preset control.

JK flip-flop. The symbolic representation of the JK flip-flop is shown in Fig. 3(a) and the truth table describing its logical operation in Fig. 3(b). The operation of this flip-flop differs in one respect from that of the SR flip-flop in that it is allowable for J and K to be simultaneously equal to 1. If $J=K=1$ the flip-flop "toggles", that is, in row 7 the flip-flop changes state from 0 to 1, whilst in row 8 the converse action takes place. In rows 4 and 5 normal reset and set operations take place as described for the SR flip-flop in the last article.

An examination of the truth table shows that the flip-flop is turned on in

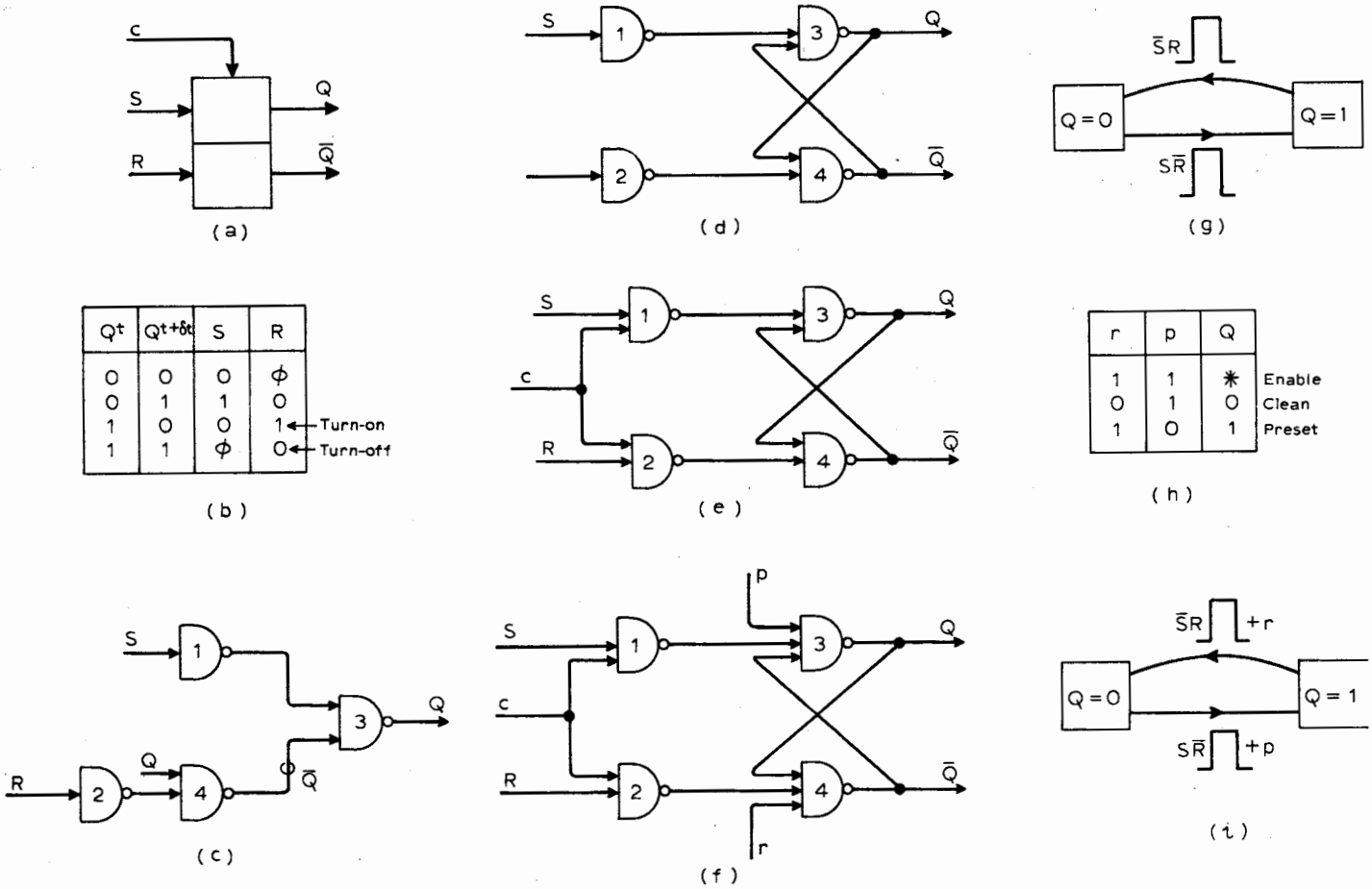


Fig. 2. (a) Symbol for the SR flip-flop, whose steering table is at (b), where Φ indicates either 0 or 1. The SR can be realized, in unclocked form, by NAND gates, as in (c) shown rearranged in a more familiar form at (d). A clocked type of SR is seen at (e) and, with preset and clear, at (f). State diagram for the clocked SR is at (g) and the truth table for P and C can be seen at (h). At (i) is the state diagram for a clocked SR with P and C controls.

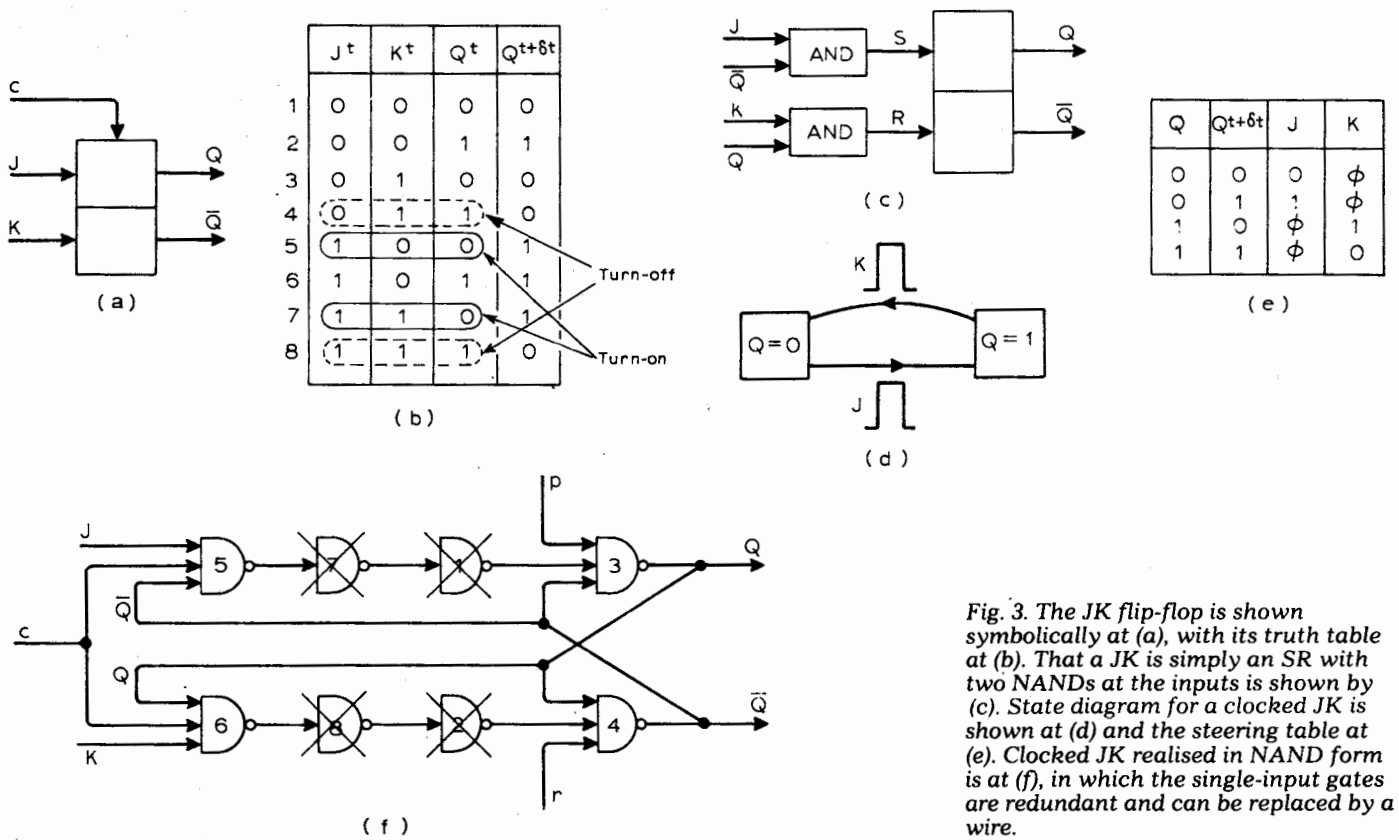


Fig. 3. The JK flip-flop is shown symbolically at (a), with its truth table at (b). That a JK is simply an SR with two NANDs at the inputs is shown by (c). State diagram for a clocked JK is shown at (d) and the steering table at (e). Clocked JK realised in NAND form is at (f), in which the single-input gates are redundant and can be replaced by a wire.

rows 5 and 7, whilst it is turned off in rows 4 and 8.

The turn-on set of Q: $S = J\bar{K}\bar{Q} + JK\bar{Q} = J\bar{Q}$

The turn-off set of Q: $R = \bar{J}KQ + JKQ = KQ$

These two equations indicate that a JK flip-flop is in practice an SR flip-flop preceded by two AND gates which implement the functions $J\bar{Q}$ and KQ respectively, as shown in Fig. 3(c).

The state diagram describing the terminal behaviour of the flip-flop is shown in Fig. 3(d). If the flip-flop is in the state Q = 0 with J = 1 and C changes from 0 to 1, it makes a transition to the state Q = 1. Similarly if in the state Q = 1 with K = 1 and C changes from 0 to 1, it makes a transition to Q = 0.

A steering table for the JK flip-flop is shown in Fig. 3(e). Comparing the steering tables of the SR and JK flip-flops shown in Figs. 2(b) and 3(e) respectively, it will be observed that the JK flip-flop has more Φ or optional input conditions and consequently this type of flip-flop leads to simpler logic when used in the design of clock-driven circuits.

A JK flip-flop can be implemented by connecting the output of the two AND gates in Fig. 3(c) to the S and R inputs of the SR flip-flop of Fig. 2(f). Simultaneously the Q and $\bar{Q}$ outputs of this flip-flop and its clock connections are fed to the inputs of the two AND gates, in conjunction with the J and K lines, as shown in Fig. 3(f). Notice that the AND gates are formed from two pairs of NAND gates in cascade, namely g_5 and g_7, and g_6 and g_8. Clearly gates g_7 and g_1 and gates g_8 and g_2 provide a double inversion. These four gates are therefore redundant and can be omitted from the implementation.

The race-around condition. Unfortunately, satisfactory flip-flop operation is not possible with the circuit shown in Fig. 3(f), for the following reason. If the outputs of the flip-flop, Q and $\bar{Q}$, in Fig. 3(f), change before the termination of the clock pulse the input conditions at gates g_5 and g_6 will also change. For example if J = K = 1 and Q = 0, when the clock pulse is first applied Q changes to a 1. This change takes place at $t = \Delta t$ after the start of the clock pulse, as shown in Fig. 4, where Δt is equal to the propagation delay through two NAND gates. At $t = \Delta t$, J = K = 1, Q = 1 and C = 1, consequently there will now be a further change in the output to Q = 0 at $t = 2\Delta t$. The conclusion is that the output of Q oscillates between 0 and 1 for the duration of the clock pulse. Further, at the end of the clock pulse the value of Q is indeterminate.

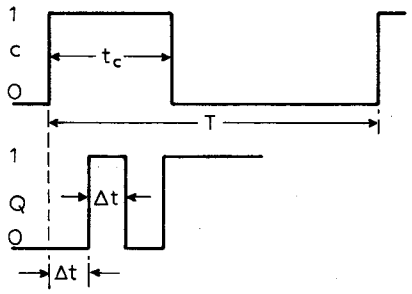


Fig. 4. Illustration of a "race-around", where the output oscillates during the duration of the trigger pulse, t_c.

This phenomenon is called the "race-around" condition. It can be avoided if $t_c < \Delta t < T$. Unfortunately, with modern integrated circuits $t_c >> \Delta t$ and the inequality is not satisfied. This has led to the development of the master/slave or double-rank flip-flop.

Master/slave flip-flop. This consists of two flip-flops in cascade. The leading one, called the master, is connected as a JK flip-flop, whilst the second one, the slave, is connected as an SR flip-flop. Clock pulses are used to enable the master whilst inverted clock pulses are used to enable the slave.

A NAND implementation of a master/slave flip-flop is shown in Fig. 5. Examination of this diagram shows that the master flip-flop changes its state on the leading edge of a clock pulse. For example if J = 1, $Q_m = 0$ and C is changing from 0 to 1, then the output state of the flip-flop changes to $Q_m = 1$. Since Q_m is also the set input of the slave flip-flop, S = 1.

The slave flip-flop is enabled when $\bar{C}$ is changing from 0 to 1, that is on the trailing edge of the clock pulse. If $Q_s = 0$, S = 1 and $\bar{C}$ is changing from 0 to 1 the output state of the slave changes to $Q_s = 1$. The change which occurred at the output of the master on the leading edge of the clock pulse is transferred to the output of the slave on the trailing edge of the same clock pulse.

The reader will observe that the slave output cannot change state until after the termination of the clock pulse and consequently the race-around condition can never occur with this type of flip-flop.

D flip-flop. The symbolic representation of a D flip-flop is shown in Fig. 6(a) and its logical operation is described by the truth table in Fig. 6(b).

From the truth table:

$$Q^{t+\delta t} = (D\bar{Q} + DQ)^t,$$

or: $$Q^{t+\delta t} = D^t.$$

The interpretation of this equation is that the output Q assumes the logical value of the input at the time of the clock pulse.

In Fig. 6(c) the terminal behaviour of the flip-flop is described with the aid of a state diagram. Assuming that the flip-flop is of the master/slave type, and if Q = 0, D = 1 and C changes from 1 to 0, it makes a transition to Q = 1. Similarly if the state is Q = 1, D = 0 and C changes from 1 to 0, it makes a transition to Q = 0.

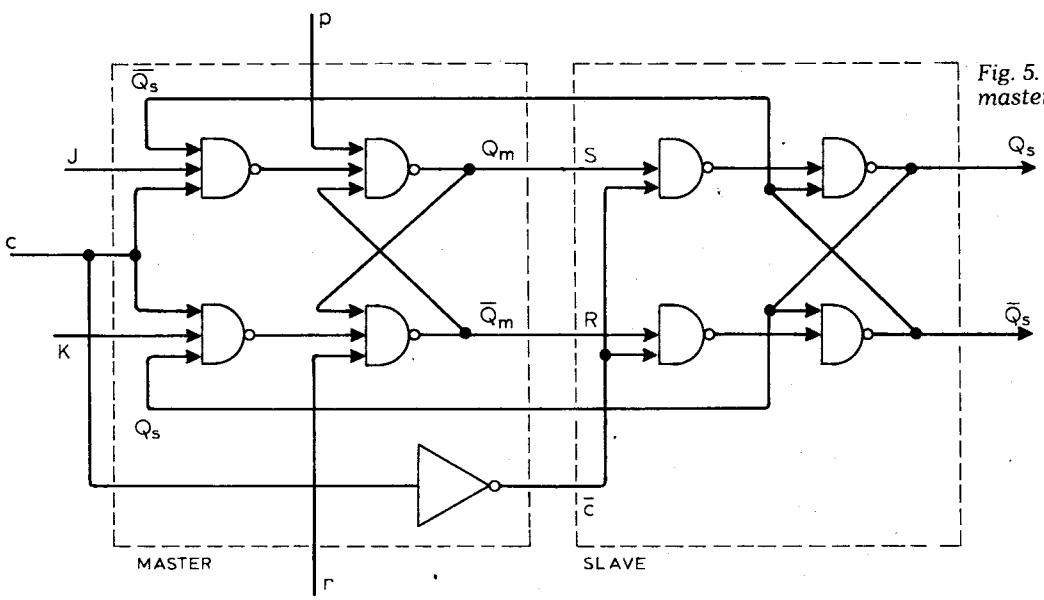


Fig. 5. NAND embodiment of a master/slave flip-flop.

JK versatility. A JK flip-flop can be easily converted to a T type by connecting the J and K lines to logical 1, as shown in Fig. 7(a). The flip-flop then toggles on the receipt of every clock pulse.

To convert a JK flip-flop to a D type the J line, besides being connected to the J input, is also connected to the K input through an inverter, as seen in Fig. 7(b). Referring to the truth table for the JK flip-flop shown in Fig. 3(b), the only entries valid for the configuration of Fig. 7(b) are those in rows 3, 4, 5 and 6. If the column headed J is identified as D and the column headed K is omitted, then the entries in these rows are identical to the entries in the truth table for the D flip-flop shown in Fig. 6(b).

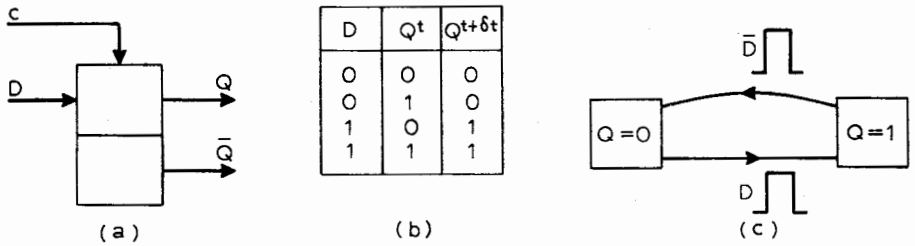


Fig. 6. D type flip-flop symbol (a), truth table (b) and state diagram (c).

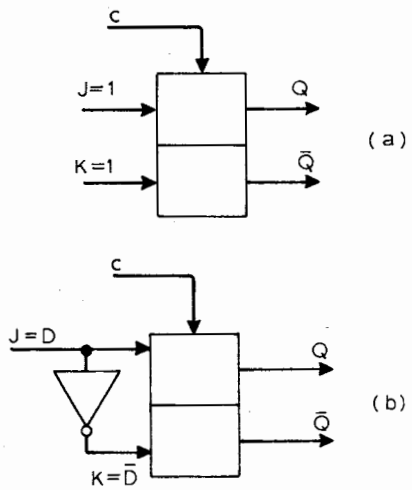


Fig. 7. Illustration of the JK used as a T type flip-flop (a) and as a D type (b)

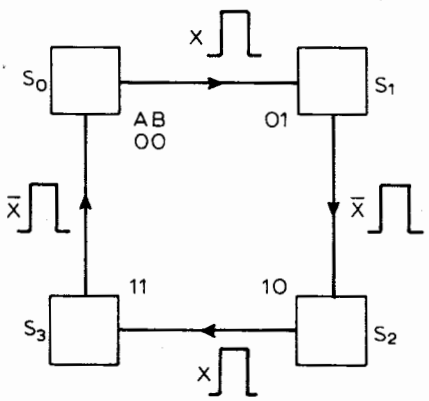


Fig. 8. State diagram for a clock-driven circuit.

Design steps

The sequence of four design steps for clock-driven circuits is as follows:

(1) **I/O characteristics.** In this step a block diagram is drawn to show the available input signals and the required output signals.

(2) **Internal characteristics.** In the second step the designer specifies the internal performance of the circuit with the aid of a state diagram. The inexperienced designer should be primarily concerned that the specification of the internal circuit operation is complete and free of ambiguities.

(3) **State reduction.** This step is optional and can be omitted. Its main purpose is to provide the designer with the means for reducing the number of internal states used in step 2, if such a reduction is possible. To avoid redundant states this step would be used to reduce the number of states to some power of 2. For example, whereas it would be used to reduce five states to four, it would not be used to reduce four states to three.

(4) **Primitive circuits.** In contrast to the situation with event-driven circuits, the design of clocked circuits does not require that only one secondary signal may change during a transition between two states. This is based on the assumption that all changes of secondary signals take place on the trailing (or leading) edge of the clock pulse that initiates them, and of course before the next clock pulse.

Having allocated the secondary signals, the turn-on and turn-off conditions are written down for each of these signals. For example, in the state diagram of Fig. 8,

Turn-on set of A: $S_A = S_1\bar{X} + (S_2X)$
Turn-off set of A: $R_A = S_3\bar{X} + (S_0X)$
Turn-on set of B: $S_B = S_0X + S_2X$
Turn-off set of B: $R_B = S_1\bar{X} + S_3\bar{X}$

Examination of these equations shows that the turn-on conditions of secondary signal B, S_B, is the disjunction (ORing) of the total states which are necessary for the next clock pulse to cause B to change value from 0 to 1. Similarly the turn-off condition of secondary signal B, R_B, is the disjunction of the total states which are necessary to cause B to change value from 1 to 0.

The expressions for the turn-on and turn-off conditions of the flip-flops can be reduced using as optional products those terms which define "don't care" circuit conditions or alternatively products which define total states involved in transitions in which the signal concerned does not change its value. For example when moving from S_2 to S_3 in Fig. 8, signal A retains its value of 1 and its turn-on conditions can be allowed to arise during this transition. Hence the turn-on equation for A consists of the disjunction of a genuine turn-on condition $S_1\bar{X}$ and an optional product (S_2X). Similarly the turn-off condition for A consists of the disjunction of a genuine turn-off condition $S_3\bar{X}$ and an optional product (S_0X).

The turn-on and turn-off conditions derived by the foregoing process define directly the set and reset signals respectively for a pair of SR flip-flops. However the most readily available and versatile flip-flop is the JK type. As this is used extensively it is worthwhile recalling the relationships derived earlier in this article between S and J, and R and K respectively. They are:

$$S_Q = J\bar{Q} \text{ and } R_Q = KQ$$

Clearly the expressions for J and K can be obtained from the expressions for S and R by dropping $\bar{Q}$ and Q respectively. This is a very useful result and the reader is advised to make a note of it.

The design procedure described above will be illustrated in the next article with the aid of a series of examples.

Literature Received

Catalogue of power supply components (transistor, rectifiers, regulators) and comple Abbey Barn Road, Electronics Co. High Wycombe, Bucks WW401

Application notes from Hewlett-Parkard on the use of spectrum analysers in noise figure (AN150-9), field strength (AN150-10) and distortion (AN150-11) measurements. Hewlett-Packard Ltd, King Street Lane, Winnersh, Workingham, Berks. WW402

Microwave Newsletter from Walmore, on video detectors, balanced amplifiers, fluorglas laminates, Gunn oscillators and a log amplifier. Walmore Electronics Ltd, 11-15 Betterton Street, London WC2H 9BS WW403

Short-form catalogue of digital-to-analogue and a.-to-d. converters, sample-and-hold amplifiers and data acquisition units, all in dual-in-line packages, from Micro Networks. Tranchant Electronics (UK) Ltd, Tranchant House, 100a High Street, Hampton, Middlesex WW404

Guide to the specification and use of surface-coating resins of many types, prepared by Cray Valley Products Ltd, St Mary Cray, Kent BR5 3PP WW405

Instrument-case catalogue from Lektrokit details the complete ranges of Motek and Lektrokit modular cases, including the newer Transistek types. Available from Lektrokit Ltd, 3 Trafford Road, Reading, RG1 8JR WW406

Data sheet on the Weir 250mA, plug-in power supply for op-amps, with an output variable from ±12V to ±15V. Weir Instrumentation Ltd, Durban Road, Bognor Regis, Sussex WW407

Logic design — 6

Examples of clock-driven circuits

by B. Holdsworth* and D. Zissos†

*Chelsea College, University of London
†Dept. of Computing Science, University of Calgary, Canada

Some examples of the design of clock-driven circuits using the techniques set out in the last article can now be considered.

Example 1. Paper Tape Reader
Design a circuit that will stop the paper tape reader, shown in Fig. 9(a), by turning signal m off when the character sequence 4-5-6 is detected, and at the same time generates a buzzer signal b.

A synchronizing pulse is generated by the reader each time a new character is output.

(1) **I/O characteristics.** See Fig. 9(a)

(2) **Internal characteristics.** A suitable state diagram is shown in Fig. 9(b)

(3) **State reduction.** The state table corresponding to Fig. 9(b) is shown in Fig. 9(c). Examination of this table shows that merging of rows is not possible.

(4) **Primitive circuits.** Suitable binary codes are allocated on the state diagram. By direct reference to this diagram the input equations to the JK flip-flops are obtained.

$$S_A = S_1 5 + (S_2 6)$$

where the term in brackets is an optional product.

$$S_A = \bar{A}B5 + (AB6)$$

The optional product cannot be used for reduction purposes.

Hence, $S_A = \bar{A}B5$ and $J_A = B5$

$$R_A = S_2 4 + S_2\bar{4}\bar{6} + (S_0)$$
$$= S_2\bar{6} + (S_0)$$
$$= AB\bar{6} + (\bar{A}\bar{B})$$

The optional product cannot be used for reduction purposes.

Hence, $R_A = AB\bar{6}$ and $K_A = B\bar{6}$

$$S_B = S_0 4 + (S_1 4) + (S_1 5) + (S_2 4)$$
$$= \bar{A}\bar{B}4 + (\bar{A}B4) + (\bar{A}B5) + (AB4)$$

The optional product ($\bar{A}B4$) need not be used for simplification purposes since $\bar{B}$ will be eliminated when converting from S_B to J_B.

Hence, $S_B = \bar{A}\bar{B}4$ and $J_B = \bar{A}4$

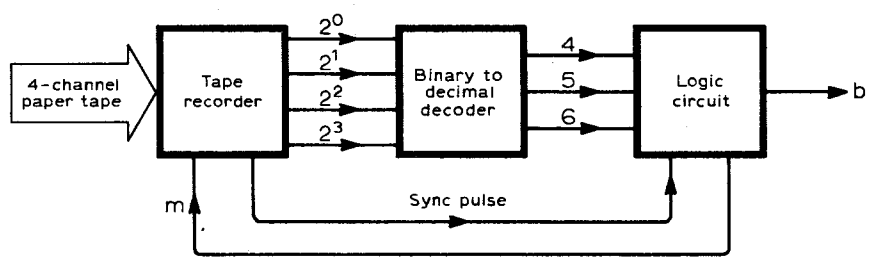
$$R_B = S_1\bar{4}\bar{5} + S_2\bar{4}\bar{6} + S_2 6 + S_0\bar{4})$$
$$= S_1\bar{4}\bar{5} + S_2\bar{4} + (S_0\bar{4})$$
$$= \bar{A}B\bar{4}\bar{5} + AB\bar{4} + (\bar{A}B\bar{4})$$
$$= B\bar{4}\bar{5} + AB\bar{4} + (\bar{A}B\bar{4})$$

The optional product cannot be used for simplification purposes, hence

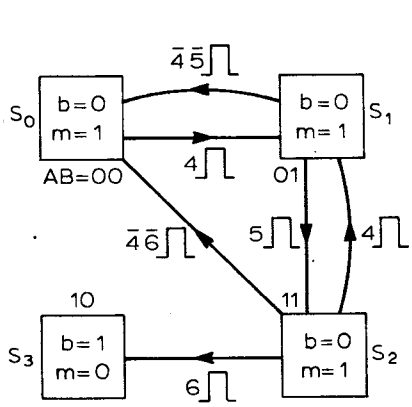
$$R_B = B\bar{4}\bar{5} + AB\bar{4} \text{ and } K_B = \bar{4}\bar{5} + A\bar{4}$$

The circuit is shown in Fig. 9(d).

Example 2. One-shot circuit
High-frequency clock pulses are fed to terminal X in Fig. 10(a). Design a circuit



(a)

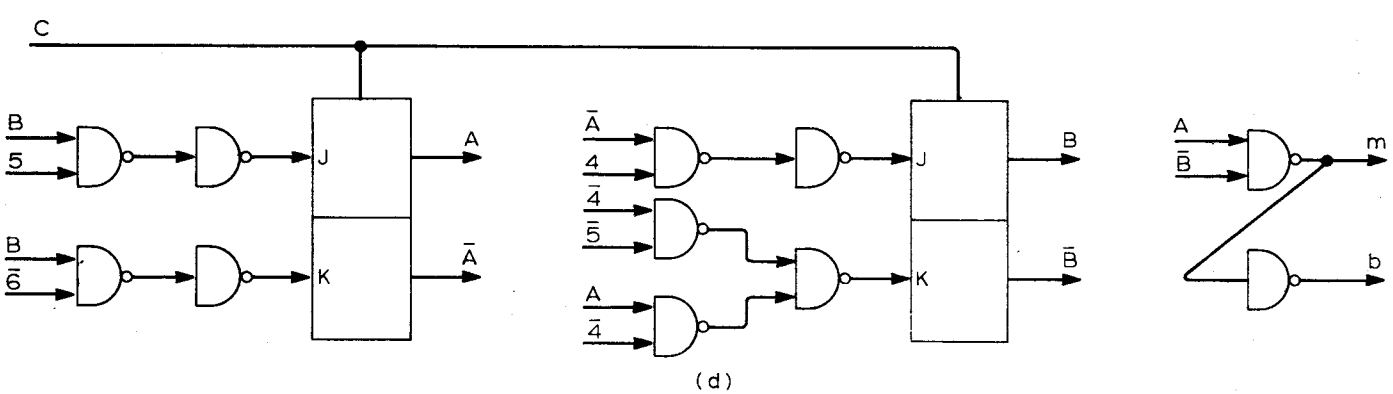


(b)

Input	4	5	6	$\bar{4}\bar{5}\bar{6}$
S_0	S_1 m=1 b=0	S_0 m=1 b=0	S_0 m=1 b=0	S_0 m=1 b=0
S_1	S_1 m=1 b=0	S_2 m=1 b=0	S_0 m=1 b=0	S_0 m=1 b=0
S_2	S_1 m=1 b=0	S_0 m=1 b=0	S_3 m=0 b=1	S_0 m=1 b=0
S_3				

(c)

Fig. 9. Circuit of Example 1 is shown at (a). Its state diagram is at (b) and its state table at (c). The resulting circuit is shown at (d).



(d)

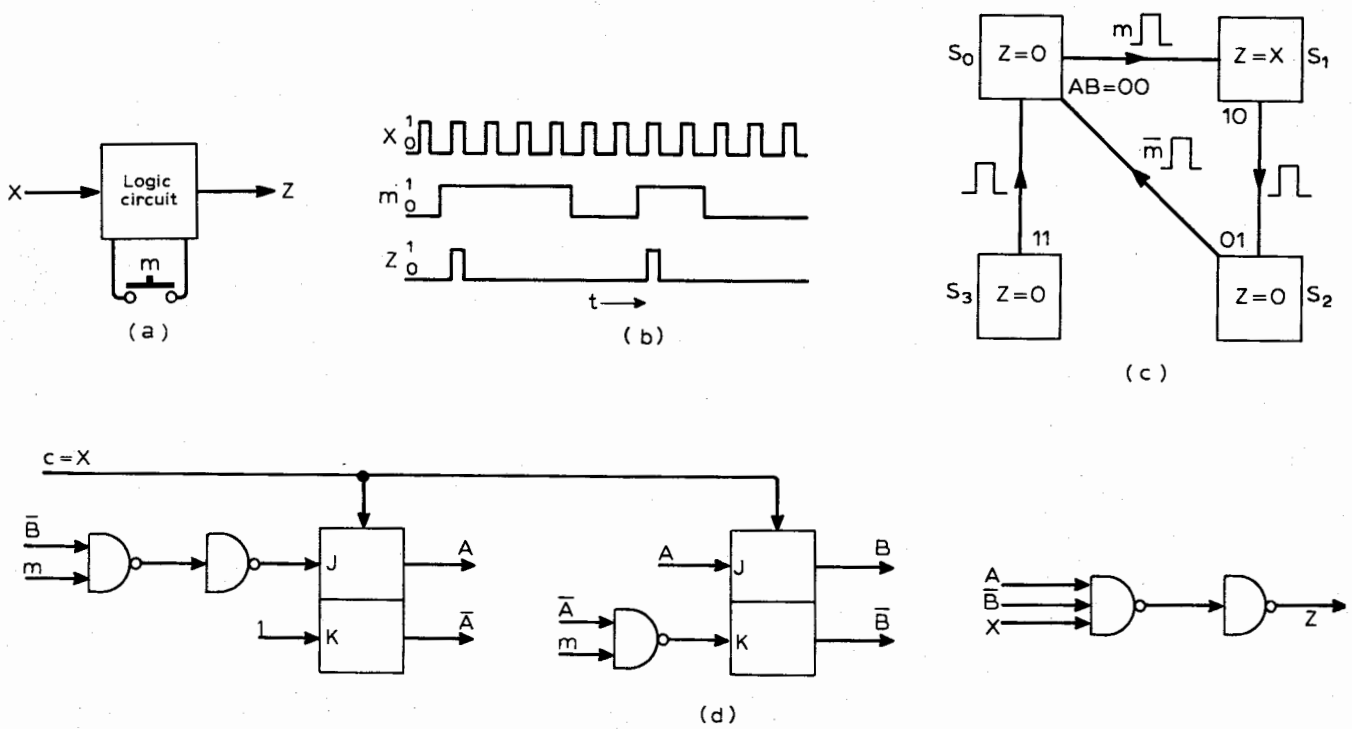


Fig. 10. Problem of Example 2, (a) and the required timing at (b). The state diagram is seen at (c) and the circuit realization is shown at (d).

so that each activation of a manual switch m allows one complete clock pulse output on line Z. The duration of signal m can be assumed to be greater than the pulse width.

(1) **I/O characteristics.** These are shown in the time diagrams of Fig. 10(b).

(2) **Internal characteristics.** A suitable state diagram is shown in Fig. 10(c).

(3) **State reduction.** It is left as an exercise for the reader to construct the state table and examine the possibility of state reduction.

(4) **Primitive circuit.** By direct reference to the state diagram the following turn-on and turn-off equations are obtained.

$S_A = S_0m = \bar{A}\bar{B}m$. Therefore $J_A = \bar{B}m$.

$$R_A = S_1 + S_3 + (S_2) + (S_0\bar{m})$$
$$= A\bar{B} + AB + (\bar{A}B) + (\bar{A}\bar{B}m)$$
$$= A. \text{ Therefore, } K_A = 1.$$

$S_B = S_1 + (S_2m) = A\bar{B} + (\bar{A}Bm) = A\bar{B}$.
Therefore $J_B = A$.

$$R_B = S_2\bar{m} + S_3 + (S_0)$$
$$= \bar{A}Bm + AB + (\bar{A}\bar{B})$$
$$= B\bar{m} + AB. \text{ Therefore } K_B = \bar{m} + A.$$

$$Z = S_1X = A\bar{B}X$$

The circuit implementation of these equation is shown in Fig. 10(d).

Example 3. Pulse distributor

Signal X in Fig. 11(a) is a pulse train. The input pulses are to appear at the output terminals as shown in Fig. 11(b).

continued on p.74

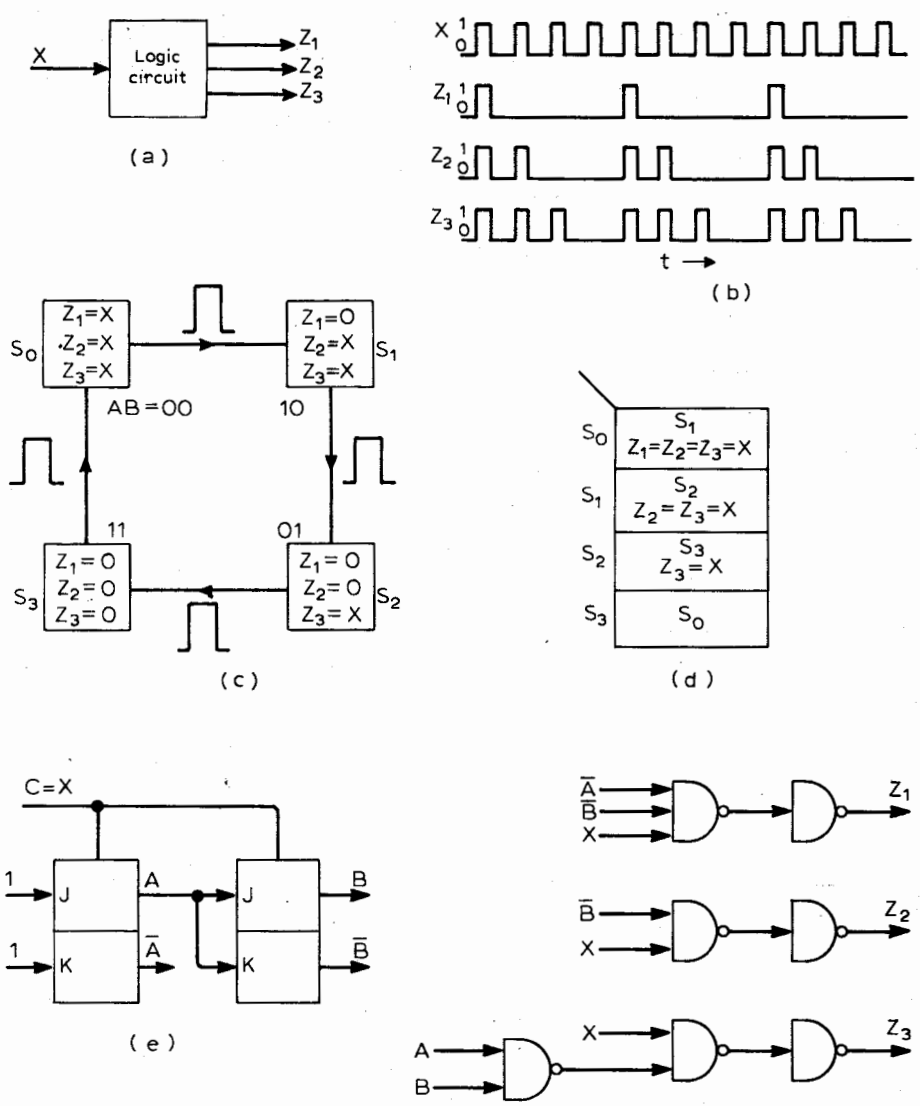


Fig. 11 (a) is the problem for Example 3, with the specified output at (b). State diagram (c) and state table (d) result in the circuit shown at (e).

Logic design — 7

Designing synchronous and asynchronous counters

by B. Holdsworth* and D. Zissos†

*Chelsea College, University of London †Dept. of Computing Science, University of Calgary, Canada.

Counters are cyclic sequential circuits which return to their initial state after a specified number of changes in the input state. The output of a counter in its specified code gives the number of changes of the input signal or the number of input pulses received since the circuit was last in its initial state. Counters are being used extensively in industrial plants for such functions as controlling the position of a machine tool or for packing a specified number of items in a box. They are also used in laboratory environments for such functions as counting frequency, recording time, speed and acceleration.

Codes

The most commonly used codes in electronic counters are:

- True binary (8-4-2-1) code,
- Gray codes,
- B.c.d. codes and
- Ordered codes, for example the excess-3 (XS-3).,

The true binary code, often referred to simply as the "binary code" is the simplest because each digit is represented in a conventional binary system. Gray codes are those in which adjacent numbers differ in one bit only, eliminating races which arise when two or more bits attempt to change simultaneously. The true binary code is shown in Table 1, for four binary digits.

If all the sixteen combinations in the sequence in Table 1 are used, the counter is called a maximum-length counter; if, on the other hand, only the first ten combinations are used the counter is called a scale-of-ten counter.

A Gray code in which only one digit changes at a time is called a single-step code, the best known one being the reflected binary code. This code is tabulated in Tables 2(a) and 2(b) for both three and four binary digits. Examination of Table 2(a) shows that reflection of the three least significant digits takes place about the centre line of the code. All those combinations above the centre line have a most significant digit of 0 whilst those below have a most significant digit of 1. Similar comments can be made about the three-digit code except that, in this case, reflection of the two least significant digits takes place.

The sequence of the 4-bit reflected binary code is shown plotted on a Karnaugh map in Fig. 1(a). The plot shows that, as the code proceeds from one combination to the next, only one cell boundary is crossed. It is clear that any single-step Gray code can be developed immediately from a Karnaugh map by tracing a single step path through the map as shown in Fig. 1(b). The code sequence for this example is shown in Fig. 1(c).

d	D	C	B	A
dec.	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Unused code combination in decade counters

Table 1. True binary code, with unused combinations for decade counters.

In b.c.d. (binary-coded-decimal) codes, each of the ten decimal digits 0 to 9, is represented by a binary code, frequently the 8-4-2-1 code. For example the b.c.d. (8-4-2-1) representation of 456 is 0100, 0101, 0110. B.c.d. codes provide a useful link between the counting systems used by digital machines and those used by human beings.

The codes tabulated in Tables 3(a) and 3(b) are examples of weighted b.c.d. codes.

In a weighted code a weight W_j is assigned to the j^{th} binary digit. For example, for the 8-4-2-1 code combination 1001, $W_4 = 8$, $W_3 = 4$, $W_2 = 2$ and $W_1 = 1$

Hence,

$$Z_{dec} = \sum_{j=1}^{j=4} W_j S_j$$

d	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	0	1	1	1
6	0	1	0	1
7	0	1	0	0
8	1	1	0	0
9	1	1	0	1
10	1	1	1	1
11	1	1	1	0
12	1	0	1	0
13	1	0	1	1
14	1	0	0	1
15	1	0	0	0

Reflection

d	C	B	A
0	0	0	0
1	0	0	1
2	0	1	1
3	0	1	0
4	1	1	0
5	1	1	1
6	1	0	1
7	1	0	0

Reflection

Table 2. Four-bit reflected binary (a) and three-bit (B) reflected binary code.

d	D	C	B	A
	2	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

d	D	C	B	A
	5	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Table 3. Weighted codes. 2-4-2-1 code is at (a) while (b) shows 5-4-2-1 code.

d	D	C	B	A
0	0	0	1	1
1	0	1	0	0
2	0	1	0	1
3	0	1	1	0
4	0	1	1	1
5	1	0	0	0
6	1	0	0	1
7	1	0	1	0
8	1	0	1	1
9	1	1	0	0

Table 4. Excess-3 code (XS-3).

d	D	C	B	A	p
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1

d	D	C	B	A	p
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0

Table 5. Parity. 8-4-2-1 code at (a) has extra bit to give odd parity and that at (b) has even parity.

where S_j is the value of the j^{th} binary digit, and

$$Z_{dec} = 1\times8+0\times4+0\times2+1\times1=9.$$

The various code combinations in the 2-4-2-1 and the 5-4-2-1 codes can be evaluated in a similar manner.

In an ordered code, the various combinations are assigned to the different decimal digits by means of a mathematical equation. An example of this is the XS-3 code. For this code

$$Z_{dec}=\sum_{j=1}^{j=4} W_j S_j - 3, \quad \text{where}$$

$$W_4=8, W_3=4, W_2=2, W_1=1.$$

Hence, the code combination 0100 = (0 × 8 + 1 × 4 + 0 × 2 + 0 × 1) − 3 = 1. The XS3 code is shown tabulated in Table 4.

Codes can be made error-detecting by the addition of extra bits, called parity bits. In Table 5(a) the 8-4-2-1 code has an additional bit in the column headed *p* which establishes odd parity in each code combination, i.e., each code combination contains an odd number of 1's. Similarly in Table 5(b) a parity bit has been added to the same code which, in this instance, establishes even parity for each code combination. Detection equipment is now required at the receiving end which, in the case of odd parity, is used to determine whether each code combination has an odd number of 1's.

Codes can also be made error-correcting by the addition of extra bits whose function is to detect an error and its position. The most important codes of this kind are the Hamming codes, in which the bit positions are numbered in sequence from left to right. Those positions numbered as a power of 2 are reserved for parity check bits, whilst the remaining positions are used for the information bits.

For a seven bit code combination:

1 2 3 4 5 6 7

p_1 p_2 x_3 p_4 x_5 x_6 x_7

p_1, p_2 and p_4 are the parity bits and x_3, x_5, x_6 and x_7 are the information bits. The parity bits are obtained from the information bits as follows:

p_1 is selected to establish even parity over bits 1,3, 5 and 7

p_2 is selected to establish even parity over bits 2, 3, 6 and 7

p_4 is selected to establish even parity over bits 4, 5, 6 and 7

The Hamming code combinations for the natural n.b.c.d. code are shown below in Table 6.

The correction process for this code is carried out on the assumption that only one bit is in error and that it is only necessary to locate that bit. This is achieved by checking for odd parity over the same three code combinations for which even parity was established at the transmitting end. The check is carried out with the aid of the exclusive–OR function.

For the exclusive–OR function $A \oplus B = \bar{A}B + A\bar{B}$ and hence

d	p_1	p_2	x_3	p_4	x_5	x_6	x_7
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1
2	0	1	0	1	0	1	0
3	1	0	0	0	0	1	1
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1
6	1	1	0	0	1	1	0
7	0	0	0	1	1	1	1
8	1	1	1	0	0	0	0
9	0	0	1	1	0	0	1

Table 6. Hamming combinations for n.b.c.d. code.

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

The above tabulation shows that the value of the exclusive-OR function is 1 when either A or B are 1, and is 0 when both A and B are either 0 or 1. In other words the value of the exclusive-OR function is 1 when odd parity exists.

The check functions are:

$$c_1 = p_1 \oplus x_3 \oplus x_5 \oplus x_7$$
$$c_2 = p_2 \oplus x_3 \oplus x_6 \oplus x_7$$
$$c_4 = p_4 \oplus x_5 \oplus x_6 \oplus x_7$$

If $c_1 = 1$ there must be an error in p_1, x_3, x_5 or x_7. The bit in error, E, may be obtained from the table below

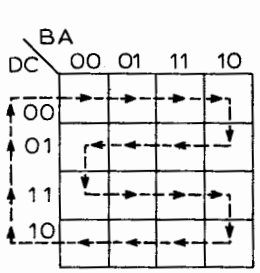
c_4	0	0	0	0	1	1	1	1
c_2	0	0	1	1	0	0	1	1
c_1	0	1	0	1	0	1	0	1
E	0	1	2	3	4	5	6	7

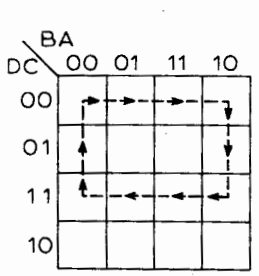
For example, suppose the code combination received is 1101101. Then $c_1 = 1$, $c_2 = 0$ and $c_4 = 1$. Hence the 5th bit is in error and the code combination should read 1101001.

Synchronous counters

The design steps for synchronous counters are (1) draw a state diagram, (2) code the states with the selected counting code, and (3) derive the input equations for the counter flip-flops.

Binary counters (maximum length). For the sake of consistency, variable A is assigned to the 2^0 bit, B to the 2^1 bit, C





d	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	0	1	0
4	0	1	1	0
5	1	1	1	0
6	1	1	1	1
7	1	1	0	1
8	1	1	0	0
9	0	1	0	0

Fig. 1. Karnaugh plots of reflected binary (a) and Gray code (b). Tabulation of Gray code is at (c).

to the 2^2 bit and so on. In deriving the general form of maximum-length binary counters, use will be made of the fact that the addition of higher order counting stages does not affect the lower order counting stages. This, of course, is also the case in conventional decimal counts – for example, the "units" and "tens" of a car odometer change at the end of every one and ten miles travelled, irrespective of the number of stages in the odometer.

Scale-of-2 'up' counter. Figure 2(a) shows the state diagram and codes.
The flip-flop equations are:
$S_A = S_0 = \bar{A}$, therefore, $J_A = 1$
$R_B = S_1 = A$, therefore, $K_A = 1$
The corresponding circuit is shown in Fig. 2(b)

Scale-of-4 'up' counter. $J_A = K_A = 1$, as for a scale-of-2 counter. The state diagram and codes are in Fig. 3(a). The flip-flop equations are:
$S_B = S_1 + (S_2) = AB$, therefore, $J_B = A$
$R_B = S_3 + (S_0) = AB$, therefore, $K_B = A$
The corresponding circuit is shown in Fig. 3(b).

Scale-of-8 'up' counter. $J_A = K_A = 1$ and $J_B = K_B = A$, as for the scale-of-4 counter. The state diagram and codes are in Fig. 4(a) and the flip-flop equations are;
$S_C = S_3 + (S_4) + (S_5 + (S_6) = AB\bar{C}$, therefore, $J_C = AB$
$R_C = S_7 + (S_0) + (S_1) + (S_2) = ABC$, therefore, $K_C = AB$
The corresponding circuit is shown in Fig. 4(b).

Scale-of-2^n 'up' counter. By observation, the flip-flop equations are;
$J_A = K_A = 1$
$J_B = K_B = A$
$J_C = K_C = AB = BJ_B$
$J_D = K_D = ABC = CJ_C$
$J_E = K_E = ABCD = DJ_D$ and so on.
If speed is essential, large input gates must be used to implement directly the functions in the third column.

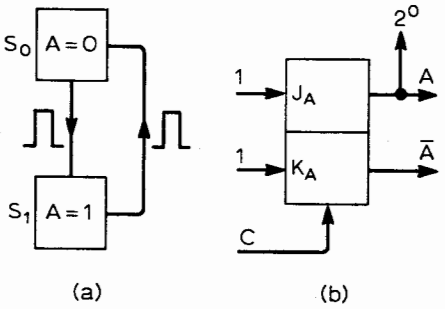
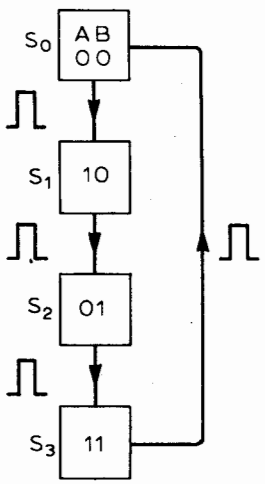


Fig. 2. State diagram for one-stage (scale-of-two) counter (a) and its circuit realization (b).



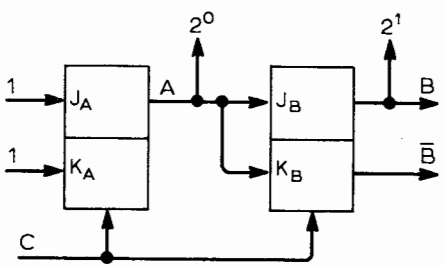


Fig. 3. Two-stage (scale-of-four) counter state diagram and codes (a) and circuit embodiment (b).

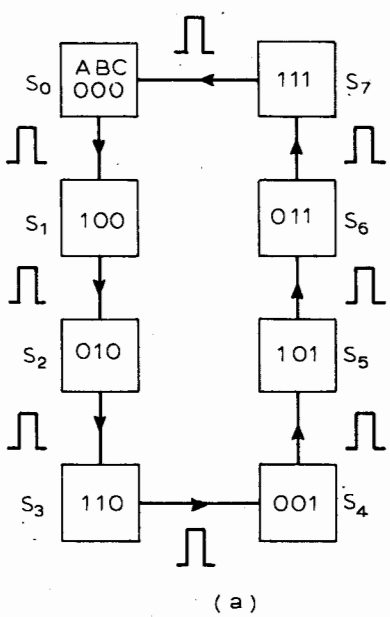
Synchronous 'down' binary counters (maximum length) can be designed in precisely the same manner and the following flip-flop equations are obtained.
$J_A = K_A = 1$
$J_B = K_B = \bar{A}$
$J_C = K_C = \bar{A}\bar{B} = \bar{B}J_B$
$J_D = K_D = \bar{A}\bar{B}\bar{C} = \bar{C}J_C$ and so on

Note that in the case of binary counters it is possible to use an 'up' counter to count down by utilizing the complementary flip-flop outputs as shown in Table 7.

d	C	B	A	d	$\bar{C}$	$\bar{B}$	$\bar{A}$
0	0	0	0	7	1	1	1
1	0	0	1	6	1	1	0
2	0	1	0	5	1	0	1
3	0	1	1	4	1	0	0
4	1	0	0	3	0	1	1
5	1	0	1	2	0	1	0
6	1	1	0	1	0	0	1
7	1	1	1	0	0	0	0

Table 7. Using the complementary outputs of a chain of flip-flops to count down.

The next part of this article will continue the treatment of counters, going on to discuss Gray code types, up-down counters and their control and ripple-through counters.



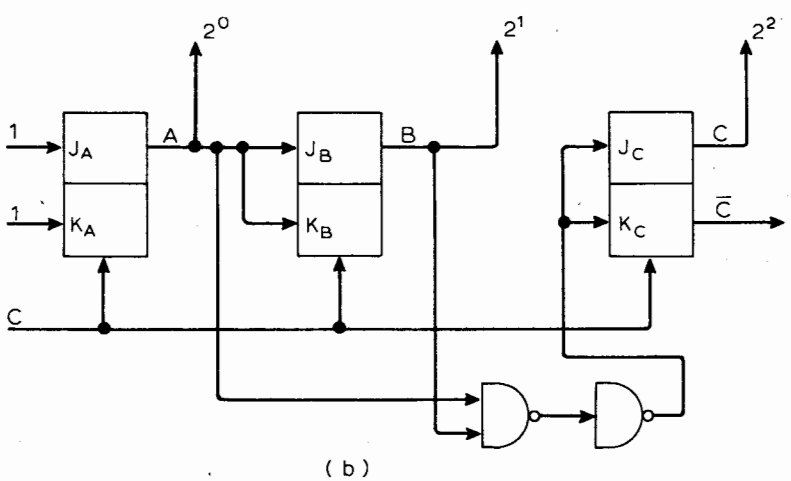


Fig. 4. State diagram (a) and circuit (b) of three-stage (scale-of-eight) counter.

Logic design — 11

Design with m.s.i. — multiplexers and demultiplexers

by B. Holdsworth* and **D. Zissos†**

*Chelsea College, University of London †Dept. of Computing Science, University of Calgary, Canada

The introduction of m.s.i. circuits is tending to result in the replacement of the old methods of logic design. Traditionally, the design engineer has developed a logic function as the solution to a particular problem. This function has then been minimized using the methods described earlier in this series and has been implemented using s.s.i. circuits. However when implementing logic functions with m.s.i. circuits such as the multiplexer, the Boolean function is used in its canonical form (i.e. each term in the Boolean function contains all the variables in the true or complemented form), and is implemented directly without minimization.

THE COST of a digital system is approximately proportional to the number of i.cs in the system, hence, to reduce the cost, the number of packages used should be minimized. The logic designer should therefore be looking for the replacement of a large number of s.s.i. circuits by one or more m.s.i. packages. It is frequently better to use a standard m.s.i. package even if this introduces redundant or unused gates rather than to design with s.s.i. circuits.

Data selector or multiplexer

The multiplexer selects one out of n lines where n is usually 4, 8 or 16. A block diagram of a data selector having 4 input lines, D_0, D_1, D_2 and D_3 and 2 output lines f and $\bar{f}$ is shown in Fig. 1(a). The device also has 2 control lines X and Y and may have an "enable" line E. The selector may be regarded as a single-pole switch which selects 1 out of 4 lines as shown in Fig. 1(b). The implementation of the multiplexer using gates is shown in Fig. 1(c).

In essence the circuit is an AND-OR-INVERT gate having complementary outputs. The Boolean function which represents the output of this circuit is: $f = \bar{X}\bar{Y}D_0 + \bar{X}YD_1 + X\bar{Y}D_2 + XYD_3$. Data lines can be selected by applying the appropriate binary coded signal to the control lines X and Y: when the control signal $\bar{X}\bar{Y} = 1$ the output of the circuit is D_0, and so on. Some multiplexers are provided with an input enable line as shown in Fig. 1(c). When the input to this line is logical 0 the four AND gates are enabled.

The number of data lines to be selected can be increased either by choosing a multiplexer with a larger number of data lines or alternatively by combining multiplexers. A combination of two data-selectors, which allows the selection of 1 out of 8 lines, is shown in Fig. 2, the enable signal in this case being used as an additional control signal. The data lines are sequentially selected with the aid of a binary counter, the control signals X and Y being clocked through the sequence 00, 01, 10 and 11, thus accessing the data

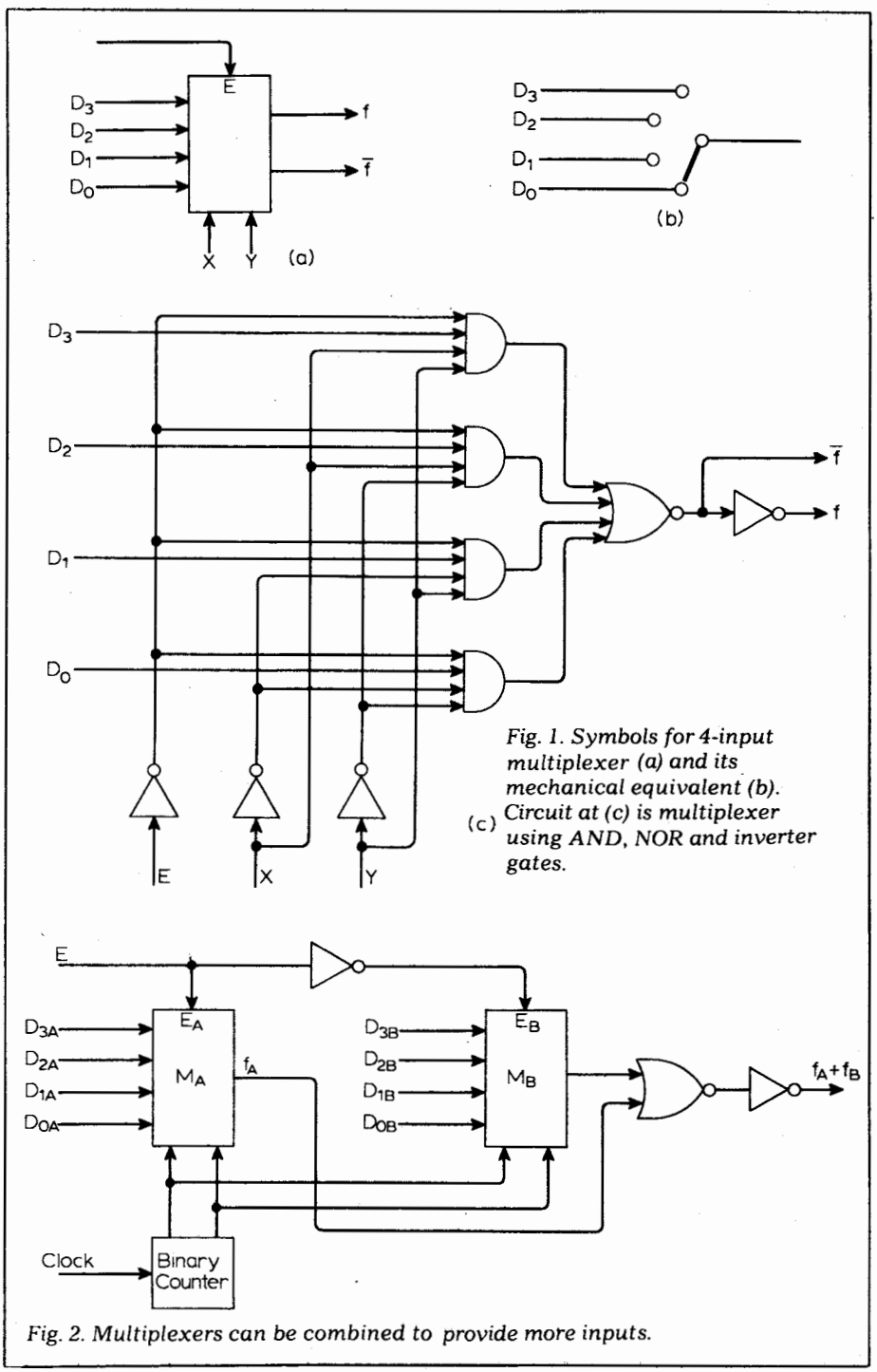


Fig. 1. Symbols for 4-input multiplexer (a) and its mechanical equivalent (b). Circuit at (c) is multiplexer using AND, NOR and inverter gates.

Fig. 2. Multiplexers can be combined to provide more inputs.

lines in the order D_0, D_1, —D_7. A truth table for the circuit is shown in Table 1. This principle can be extended to allow the selection of a larger number of data lines. For example, the selection of 1 out of 64 lines can be achieved using nine 8-input multiplexers, as shown in Fig. 3, arranged in two levels of multiplexing.

An alternative way of looking at the multiplexer is to regard it as a device which converts parallel information into serial information. For example, in the arrangement shown in Fig. 2(a), the two multiplexers M_A and M_B can be presented with an 8-bit word on the 8 input lines in parallel form, and this can be taken off in serial form by using the sequential accessing technique.

Multiplexer as logic function generator

The equation for a multiplexer having four input lines is:

$f = A\bar{B}D_0 + \bar{A}BD_1 + A\bar{B}D_2 + ABD_3$,

where the Boolean variables A and B are used as the signals for the control lines X and Y. Hence A and B can be factored out of any function of n variables, and the residue functions of n 2 variables can then be applied to the data lines. For example if n = 3, four signals of one variable can be applied to each of the data lines. Assuming that the third variable is C the possible signals that can be applied to these lines are C, $\bar{C}$, 0 and 1. In all there are $4^4 = 256$ possible combinations of four input signals which can be applied to the 4-input lines; a multiplexer with 4 input lines can generate any of the 256 possible Boolean functions of 3 variables.

For the 4-input multiplexer there are three possible choices for the control variables – AB, AC and BC. These various combinations of the control variables can be associated with individual data lines as indicated in Fig. 4. For example, with control variables A and B, the input line D_0 is associated with those cells marked A = 0 and B = 0, that is the two top left-hand cells on the K-map of Fig. 4(a). In effect, the K-map for 3-variables has now been split into four 2-cell, 1-variable K-maps, each of these 2-cell maps being associated with a data line.

Table 1

E	A	B	E_A	E_B	
0	0	0	0	1	
0	0	1	0	1	M_B
0	1	0	0	1	inhibited
0	1	1	0	1	
1	0	0	1	0	
1	0	1	1	0	M_A
1	1	0	1	0	inhibited
1	1	1	1	0	

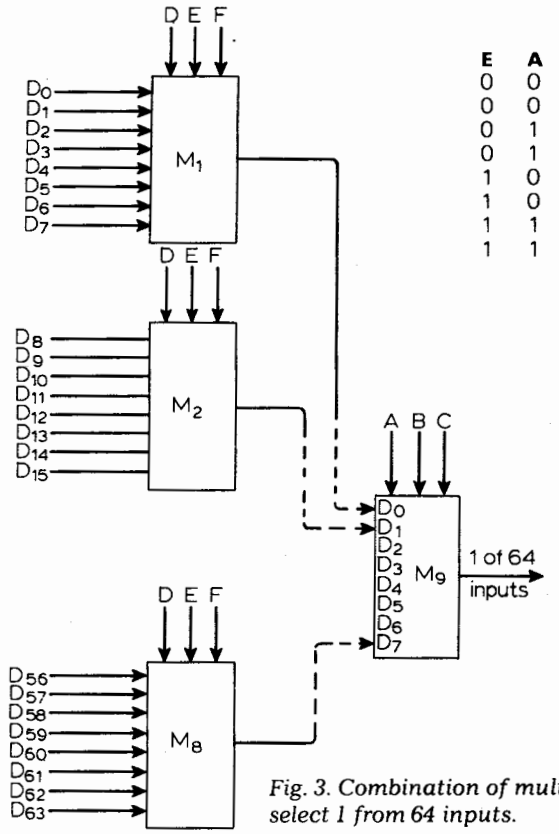


Fig. 3. Combination of multiplexers to select 1 from 64 inputs.

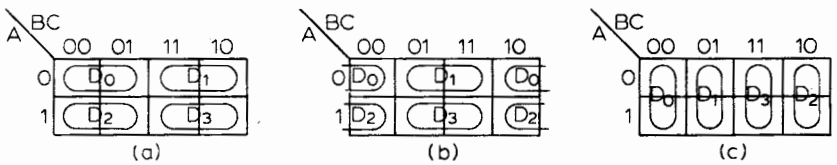


Fig. 4. Association of data lines with control signals for 4-input multiplexer. Control variables are A and B in (a), A and C in (b) and B and C in (c).

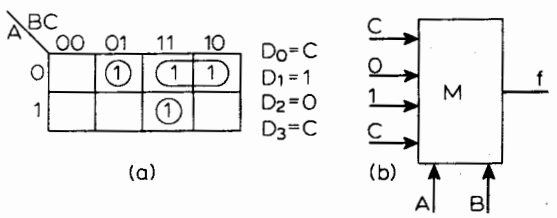
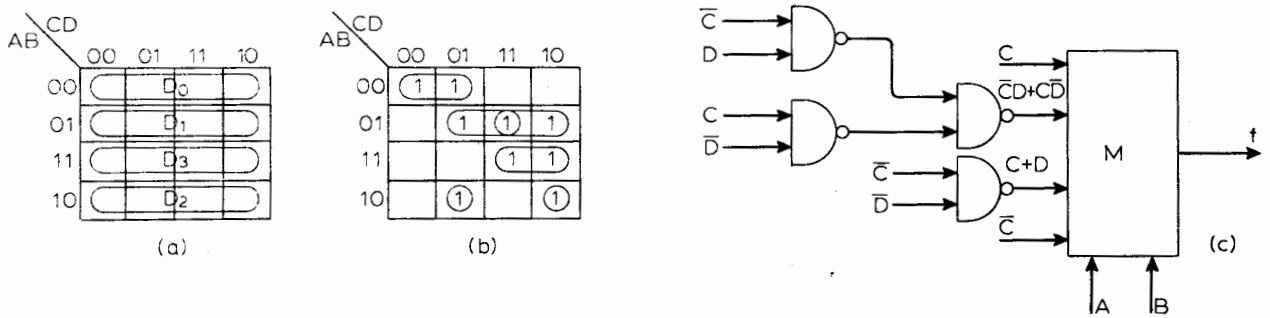


Fig. 5. Generation of $f = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + ABC$ using a 4-input multiplexer.

Fig. 6. Generation of f = 0,1,5,6,7,9,10,14,15. Association of data lines with variable A and B is seen at (a) and the Karnaugh map is at (b). The diagram at (c) is the implementation.



Example 1. Implement the 3-variable function

$$f=\bar{A}\bar{B}C+\bar{A}BC+A\bar{B}\bar{C}+ABC$$

using a 4-input multiplexer.

Plot the function on a K-map as shown in Fig. 5(a) and make an arbitrary choice of control variables, say A and B. Next simplify the four 1-variable functions associated with each data line. For example, the two cells associated with D_1 are both marked with a 1, hence the input to data line D_2 is $C+\bar{C}=1$. The remaining inputs are determined in the same manner and the implementation of the function is shown in Fig. 5(b).

Example 2 Implement the 4-variable function

$$f=\Sigma 0,1,5,6,7,9,10,14,15$$

using a 4-input multiplexer.

The function has been represented as the sum of a number of canonical terms, each term being represented as a decimal number. For example the term $\bar{A}BC\bar{D}$, represented in binary, is $0110=6$ in decimal.

Since a 4-input multiplexer is to be used, the application of two variables to its control lines will leave residue functions of two variables to be applied to the data lines. There are six possible ways of choosing the control variables – AB, AC, AD, BC, BD, and CD. These various combinations of control variables can be associated with the data lines as indicated previously in Fig. 4. It will be assumed in this example that A and B are chosen as the control variables and the K-map associating these control variables with the data lines is shown in Fig. 6(a). The 4-variable K-map has now been divided into four 4-cell, 2-variable maps and simplification can only take place within the confines of the 2-variable maps.

The K-map plot of the function is shown in Fig. 6(b) and the data line inputs obtained from the four rows of this map are:

$D_0=\bar{C}$ address $\bar{A}\bar{B}$
$D_1=C+D$ address $\bar{A}B$
$D_2=\bar{C}D+C\bar{D}$ address $A\bar{B}$
$D_3=C$ address AB

The implementation of the function is shown in Fig. 6(c).

It should be pointed out that it is useful to examine the various possible choices of control variables to ascertain whether there is a simpler solution. In this case it is left to the reader to show that a simpler solution is obtained if C and D are chosen as control variables.

As the number of variables associated with the Boolean function to be implemented increases, it becomes necessary to use more than one level of multiplexing and this technique is illustrated in the next example.

Table 2. Determination of the inputs to the 1st level multiplexer.

	$\bar{D}\bar{E}$	$\bar{D}E$	$D\bar{E}$	DE
$f=\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$	$\bar{A}\bar{B}\bar{C}$			
$+\bar{A}\bar{B}\bar{C}DE$		$\bar{A}\bar{B}\bar{C}$		
$+\bar{A}\bar{B}C\bar{D}E$				$\bar{A}\bar{B}\bar{C}$
$+\bar{A}\bar{B}C\bar{D}\bar{E}$	$\bar{A}B\bar{C}$			
$+\bar{A}\bar{B}C\bar{D}E$		$\bar{A}B\bar{C}$		
$+\bar{A}\bar{B}CD\bar{E}$				$\bar{A}B\bar{C}$
$+\bar{A}\bar{B}CDE$	$\bar{A}BC$			
$+\bar{A}BC\bar{D}E$		$\bar{A}BC$		
$+\bar{A}BCD\bar{E}$			$\bar{A}BC$	
$+A\bar{B}C\bar{D}\bar{E}$	$A\bar{B}\bar{C}$			
$+A\bar{B}C\bar{D}E$		$A\bar{B}\bar{C}$		
$+A\bar{B}CD\bar{E}$			$A\bar{B}C$	
$+A\bar{B}CDE$				$A\bar{B}C$
$+AB\bar{C}D\bar{E}$			$AB\bar{C}$	
$+ABCDE$				ABC

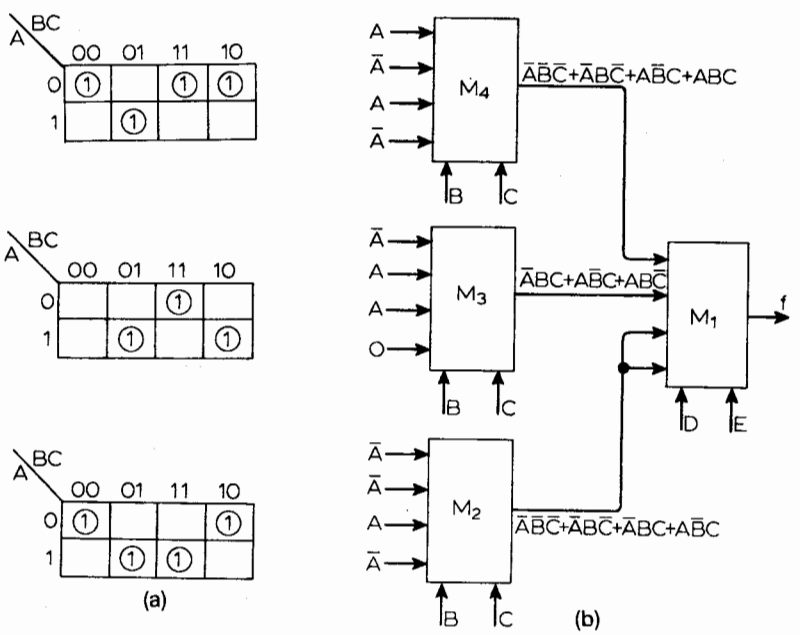
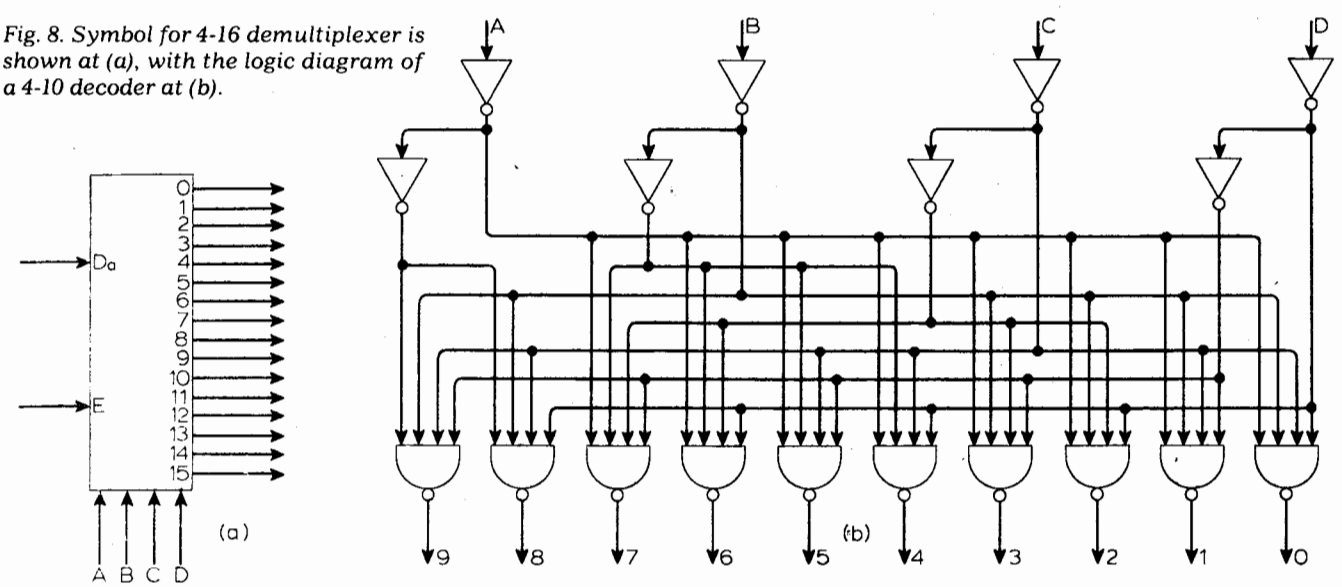


Fig. 7. Five-variable multiplexer circuit to produce the function of example 3.

Fig. 8. Symbol for 4-16 demultiplexer is shown at (a), with the logic diagram of a 4-10 decoder at (b).



Example 3. Implement the 5-variable function

$f = \Sigma 0,1,3,8,9,11,12,13,14,20,21,22,23,26,31$

For the first level of multiplexing the control variables D and E have been arbitrarily chosen. The function is now listed at the left-hand side of Table 2, which contains four columns headed $\bar{D}\bar{E}$, $\bar{D}E$, $D\bar{E}$ and DE respectively. In the column headed $\bar{D}\bar{E}$ are listed all those terms of three variables A, B, and C which are associated with $\bar{D}\bar{E}$. For example, in the case of the term $\bar{A}B\bar{C}\bar{D}\bar{E}$ the entry in the $\bar{D}\bar{E}$ column will be $\bar{A}B\bar{C}$. This procedure is repeated for each term in the 5-variable function and an entry is made in the appropriate column in each case.

The input functions for the first level multiplexer are now seen to be:

$$D_{01} = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C$$
$$D_{11} = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C$$
$$D_{21} = \bar{A}BC + A\bar{B}C + AB\bar{C}$$
$$D_{31} = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}C + ABC$$

These three variable functions can be generated with 4-input multiplexers, as described in example 1, at the second level of multiplexing. However it should be noticed that $D_{01} = D_{11}$ and this function need only be generated once, hence only three second level multiplexers are required.

For the second level of multiplexing B and C have been chosen as the control variables. The K-maps for determining the inputs to the data lines for the second level multiplexers are shown in Fig. 7(a) and from these maps the various input signals are found to be:

$$D_{02} = \bar{A} \quad D_{03} = 0 \quad D_{04} = \bar{A}$$
$$D_{12} = A \quad D_{13} = A \quad D_{14} = A$$
$$D_{22} = \bar{A} \quad D_{23} = A \quad D_{24} = \bar{A}$$
$$D_{32} = \bar{A} \quad D_{33} = \bar{A} \quad D_{34} = A$$

The implementation of the function is shown in Fig. 7(b).

Decoders or Demultiplexers

A decoder or demultiplexer performs the opposite function to that of a multiplexer. A block diagram of the device is shown in Fig. 8(a). A single data input line can be connected to one of many output lines by the appropriate choice of signal on the control lines. With 4 control lines A, B, C, and D there are sixteen possible addresses and hence the maximum number of output lines that can be selected is sixteen.

A commonly used decoder has 4 input lines and 10 output lines. The logic diagram for this device is shown in Fig. 8(b). If A = 0, B = 0, C = 0 and D = 0, the output line marked 0 will be at logical 0 whilst all the other outputs will be at logical 1.

The device illustrated in Fig. 8(b) can be used as a decoder, but in a 4-to-16 line demultiplexer there are additionally enable and data lines as shown in Fig. 8(a). These are connected to the sixteen output gates via the circuit shown in Fig. 9 which is in effect a NOR gate. This input

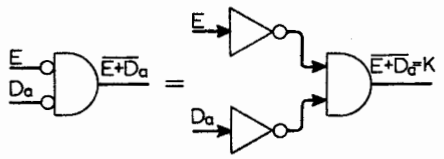


Fig. 9. Input data and enable arrangements.

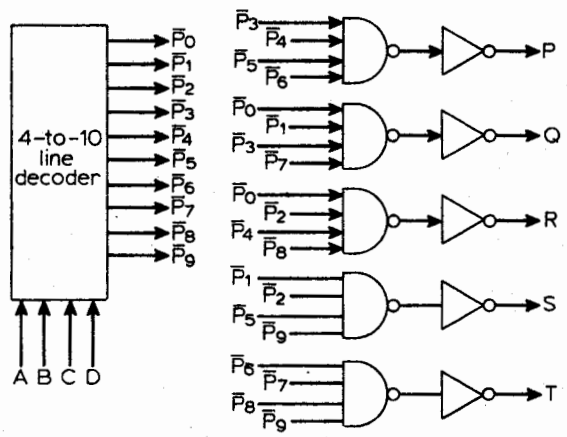


Fig 10. Natural binary-coded decimal to Lorenz converter.

arrangement allows of two modes of operation. In the first mode, if E = O & $D_a = 0$, K = 1, thus enabling all output gates. For any other values of E & D_a, K = 0, thus disabling all output gates.

In this mode the 4-to-16 line demultiplexer will act as a decoder allowing, for example, a b.c.d. input on lines A. B. C and D to be converted to a decimal output. Alternatively the circuit can be operated as a generator of the sixteen canonical terms of four Boolean variables. If $P_3 = \bar{A}\bar{B}CD$ is the input to the control lines then the output on line 3 = $\bar{P}_3$.

In the second mode E = 0, $D_a = 0$, hence K = 1. Control signal $P_2 = \bar{A}B\bar{C}\bar{D}$. The output on line $2 = 0 = D_a$.

E = 0, $D_a = 1$, hence K = 0. Control signal $P_2 = \bar{A}B\bar{C}\bar{D}$. The output on line $2 = 1 = D_a$.

In this mode the data on the data line is transferred to the output gate selected by the address applied to the control lines, in this case $\bar{A}B\bar{C}\bar{D}$.

Example 4 Using a 4-to-10 line decoder develop a circuit for converting n.b.c.d. to the Lorenz code.

The two codes are tabulated alongside each other in Table 3.

	NBCD				Lorenz				
	A	**B**	**C**	**D**	**P**	**Q**	**R**	**S**	**T**
P_0	0	0	0	0	1	0	0	1	1
P_1	0	0	0	1	1	0	1	0	1
P_2	0	0	1	0	1	1	0	0	1
P_3	0	0	1	1	0	0	1	1	1
P_4	0	1	0	0	0	1	0	1	1
P_5	0	1	0	1	0	1	1	0	1
P_6	0	1	1	0	0	1	1	1	0
P_7	0	1	1	1	1	0	1	1	0
P_8	1	0	0	0	1	1	0	1	0
P_9	1	0	0	1	1	1	1	0	0

From the tabulation:

$$P = P_0 + P_1 + P_2 + P_7 + P_8 + P_9$$
$$Q = P_2 + P_4 + P_5 + P_6 + P_8 + P_9$$
$$R = P_1 + P_3 + P_5 + P_6 + P_7 + P_9$$
$$S = P_0 + P_3 + P_4 + P_6 + P_7 + P_8$$
$$T + P_0 + P_1 + P_2 + P_3 + P_4 + P_5$$

Now $\bar{P} = P_3 + P_4 + P_5 + P_6$

Hence $\overline{\bar{P} = P_3 + P_4 + P_5 + P_6}$

and $\bar{P} = \overline{\bar{P}_3\,\bar{P}_4\,\bar{P}_5\,\bar{P}_6}$

Similarly $\bar{Q} = \overline{\bar{P}_0\bar{P}_1\bar{P}_3\bar{P}_7}$

$\bar{R} = \overline{\bar{P}_0\bar{P}_2\bar{P}_4\bar{P}_8}$

$\bar{S} = \overline{\bar{P}_1\bar{P}_2\bar{P}_5\bar{P}_9}$

$\bar{T} = \overline{\bar{P}_6\bar{P}_7\bar{P}_8\bar{P}_9}$

The implementation of the code converter is shown in Fig. 10.

The technique used in this example is useful where there are many functions of the same number of variables to be implemented. In comparison the multiplexer requires less additional gating, but one multiplexer at least is required to implement each function.

The second part of this article will deal with the applications of read-only memories.

Logic design — 12

M.s.i. — applications of read-only memories

by B. Holdsworth* and **D. Zissos†** *Chelsea College, University of London
†Dept. of Computing Science, University of Calgary, Canada

This is the second part of the article on applications of medium-scale integrated logic circuits. The first part was a discussion of the use of multiplexers and decoders, and the article now continues with a look at the use of read-only memories as function generators.

Read only memories

THE CIRCUIT shown in Fig. 11 is that of a 64-bit r.o.m. organised as 8 words of 8 bits each. It consists of a 3-bit address decoder, a 64-bit memory, and 8 output buffers. An enable input, when at logical 0, enables all the gates in the address decoder. The vertical lines in the memory section are called word lines and the horizontal ones bit lines.

Words are programmed into the r.o.m. at each address and the output on the bit line depends on whether it is connected to the addressed word line or not, the connexion being made by the presence of an m.o.s. or bipolar transistor, depending upon which technology is used. If connected, the bit line is raised to a logical 1 and if not it remains at logical 0. In this way the word programmed at the selected address is transferred to the output.

A schematic way of representing a programmed 64-bit r.o.m. is shown in Fig. 12, where at those intersections of bit lines and word lines marked by a dot there is an OR input for the output function. For example, the output at Z_8 is $P_0+P_5+P_7$. Hence the r.o.m. shown in Fig. 12 is being used to generate eight 3-variable functions, each of which is expressed in canonical form.

If a customer wishes to realise the functions shown in Fig. 12 he must supply the manufacturer with either a connexion matrix, such as the one shown in that diagram, or a truth table, as shown in Table 1. Alternatively, the customer may have his own programming facilities and in those circumstances he would purchase a programmable read only memory (p.r.o.m.).

Addressing. The connexion matrix shown in Fig. 13(a) is for a two 8-bit word r.o.m. addressed in one dimension only. The total capacity of the r.o.m. is 16 bits and the Boolean functions generated by it are:

$$Z_1=P_2+P_5+P_7$$

and $$Z_2=P_0+P_4+P_5+P_7$$

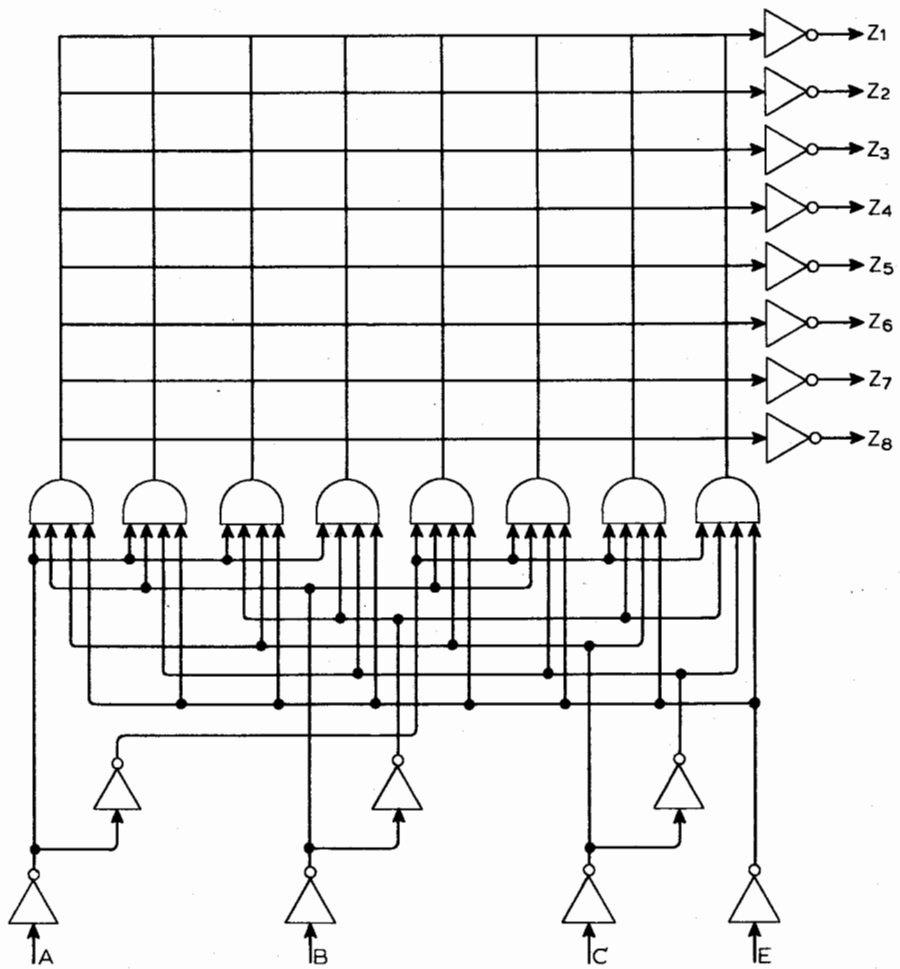


Fig. 11. 64-bit r.o.m. with address decoder and enable input. No intersections are programmed.

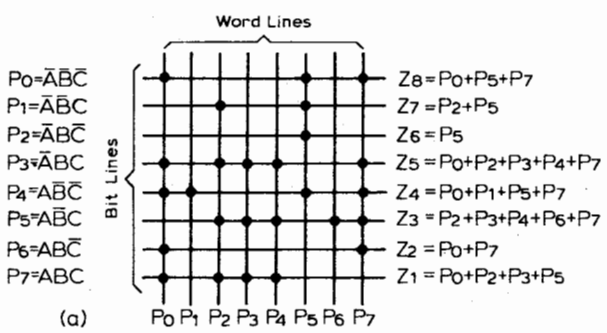


Fig. 12. Connexion matrix for a 64-bit r.o.m. at and the equivalent truth table in Table 1.

Table 1

Address			Output functions							
A	B	C	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	Z_8
0	0	0	1	1	0	1	1	0	0	1
0	0	1	0	0	0	1	0	0	0	0
0	1	0	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	1	0	0	0
1	0	0	0	0	1	0	1	0	0	0
1	0	1	1	0	0	1	0	1	1	1
1	1	0	0	0	1	0	0	0	0	0
1	1	1	0	1	1	1	1	0	0	1

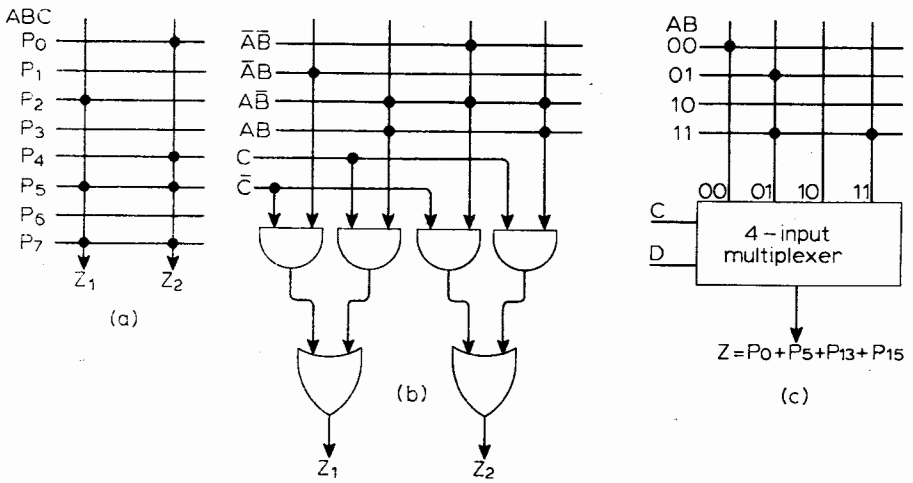


Fig. 13. A 2×8-bit word r.o.m. addressed in one dimension is shown at (a), while at (b) is the two-dimensional method. Generation of one 4-variable function using two-dimensional scheme is seen at (c).

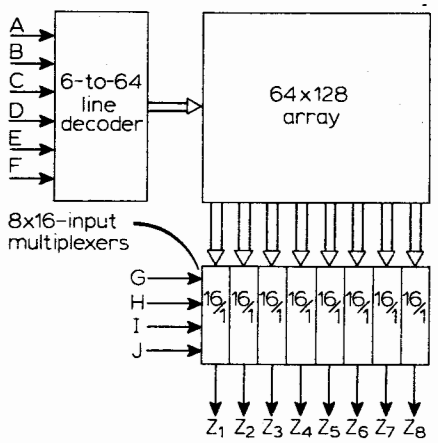


Fig. 14. Reduction of number of lines to and from r.o.m. by two-dimensional addressing.

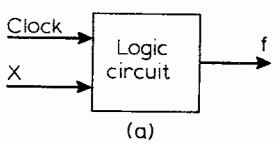
Table 2.

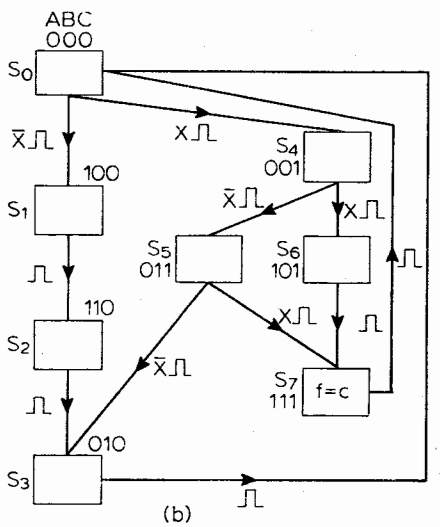
Present State				Output Word				Present State				Output Word			
A	B	C	X	A	B	C	Z	A	B	C	X	A	B	C	Z
0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0
0	0	0	1	0	0	1	0	0	0	1	1	1	0	1	0
1	0	0	0	1	1	0	0	0	1	1	0	0	1	0	0
1	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0
1	1	0	0	0	1	0	0	1	0	1	0	1	1	1	0
1	1	0	1	0	1	0	0	1	0	1	1	1	1	1	0
0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	1
0	1	0	1	0	0	0	0	1	1	1	1	0	0	0	1

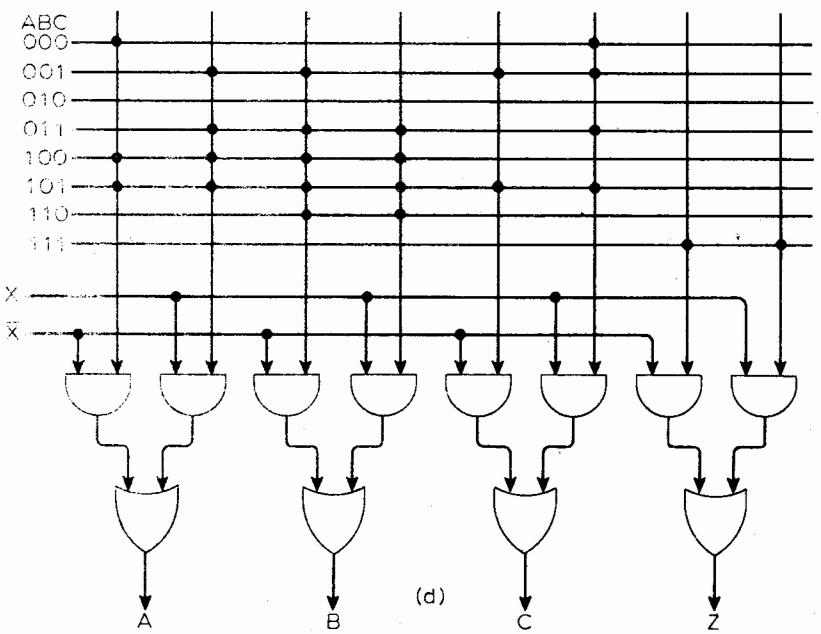
Table 3

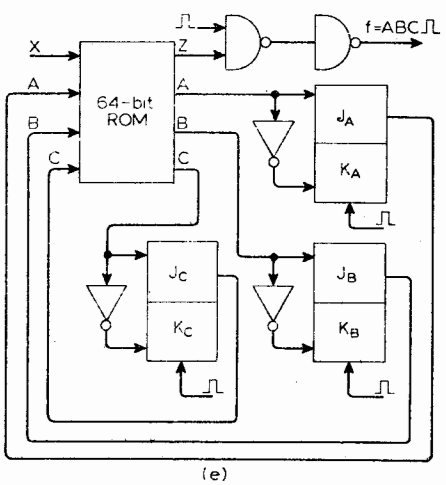
Present State					Output Word								
A	B	C	X	P	A	B	C	g_1	a_1	r_1	g_2	a_2	r_2
0	0	0	0	d	0	1	0	0	1	0	0	0	1
0	0	0	1	d	0	0	0	1	0	0	0	0	1
0	1	0	0	d	0	1	0	0	1	0	0	0	1
0	1	0	1	d	1	1	0	0	0	1	0	0	1
1	1	0	0	0	1	1	0	0	0	1	0	0	1
1	1	0	0	1	1	1	1	0	0	1	1	0	0
1	1	0	1	d	1	1	0	0	0	1	0	0	1
1	1	1	0	d	1	1	1	0	0	1	1	0	0
1	1	1	1	0	0	1	0	0	0	1	0	1	0
1	1	1	1	1	1	1	1	0	0	1	1	0	0
0	1	1	0	d	0	0	1	0	0	1	0	0	1
0	1	1	1	d	0	1	1	0	0	1	0	1	0
0	0	1	0	d	0	0	1	0	0	1	0	0	1
0	0	1	1	d	0	0	0	1	0	0	0	0	1
1	0	0	d	d	1	0	1	0	0	1	0	0	1
1	0	1	0	0	1	0	1	0	0	1	0	0	1
1	0	1	0	1	1	1	1	0	0	1	1	0	0
1	0	1	1	d	1	0	1	0	0	1	0	0	1

Fig. 15. Block diagram from problem of Example 5 at (a) and state diagram at (b). State table is in Table 2 and connexion matrix for 64-bit r.o.m. is at (c). R.o.m. implementation is shown in (d).









An alternative two-dimensional method of addressing a 16-bit r.o.m. is illustrated in Fig. 13(b), in which examination of the connexion matrix shows that the same Boolean functions are generated as in the previous case. Using this technique there is a reduction in the number of address lines required, but additional gating is needed.

The same r.o.m. in conjunction with one 4-input multiplexer can be used to generate one 4-variable function as shown in Fig. 13(c), where six address lines only are needed as compared to the sixteen address lines required with one-dimensional addressing.

Clearly, large capacity r.o.ms can be similarly addressed in two dimensions. For example, a 1024-word by 8-bit r.o.m., using single dimensional addressing, would require 1024 address lines for the 10 input variables. On the other hand, a two dimensional addressing scheme such as the one shown in Fig. 14 can be used, in which the numbering of lines entering and leaving the 64 × 128 array has been reduced from 1032 to 192. For this particular r.o.m. a 1 or a 0 can be specified in any of 8192 locations and the number of possible stored combinations is, therefore, 2^{8192}.

Sequential circuits using r.o.m.

R.o.ms are suitable devices for the implementation of clock driven and event driven logic circuits and their use in this application will be illustrated with the aid of two examples.

Example 5. Serial b.c.d. messages arrive on line X, most significant digit first. Each data bit is synchronised with a clock pulse. Design a circuit using a r.o.m. that generates a fault signal on terminal f each time an invalid code is received.

Step 1. I/O characteristics. These are described in the statement of the problem and are summarised in the block diagram Fig. 15(a).

Step 2. Internal characteristics. A suitable state diagram is shown in Fig. 15(b)

Step 3. State table. This is shown in Table 2 and is displayed in a suitable form for r.o.m. implementation.

Step 4. Connexion matrix. This is shown in Fig. 15(c) for a two dimensionally addressed 64-bit r.o.m.

Step 5. Circuit implementation. This is shown in Fig. 15(d). Besides the 64-bit r.o.m. additional logic is required to produce the output signal $f = ABC\sqcap\!\!\sqcup$. Additionally three D-type flip-flops are required in each feedback line to synchronise the operation of the circuit to the clock.

The next example illustrates the implementation of an event-driven logic circuit with a r.o.m.

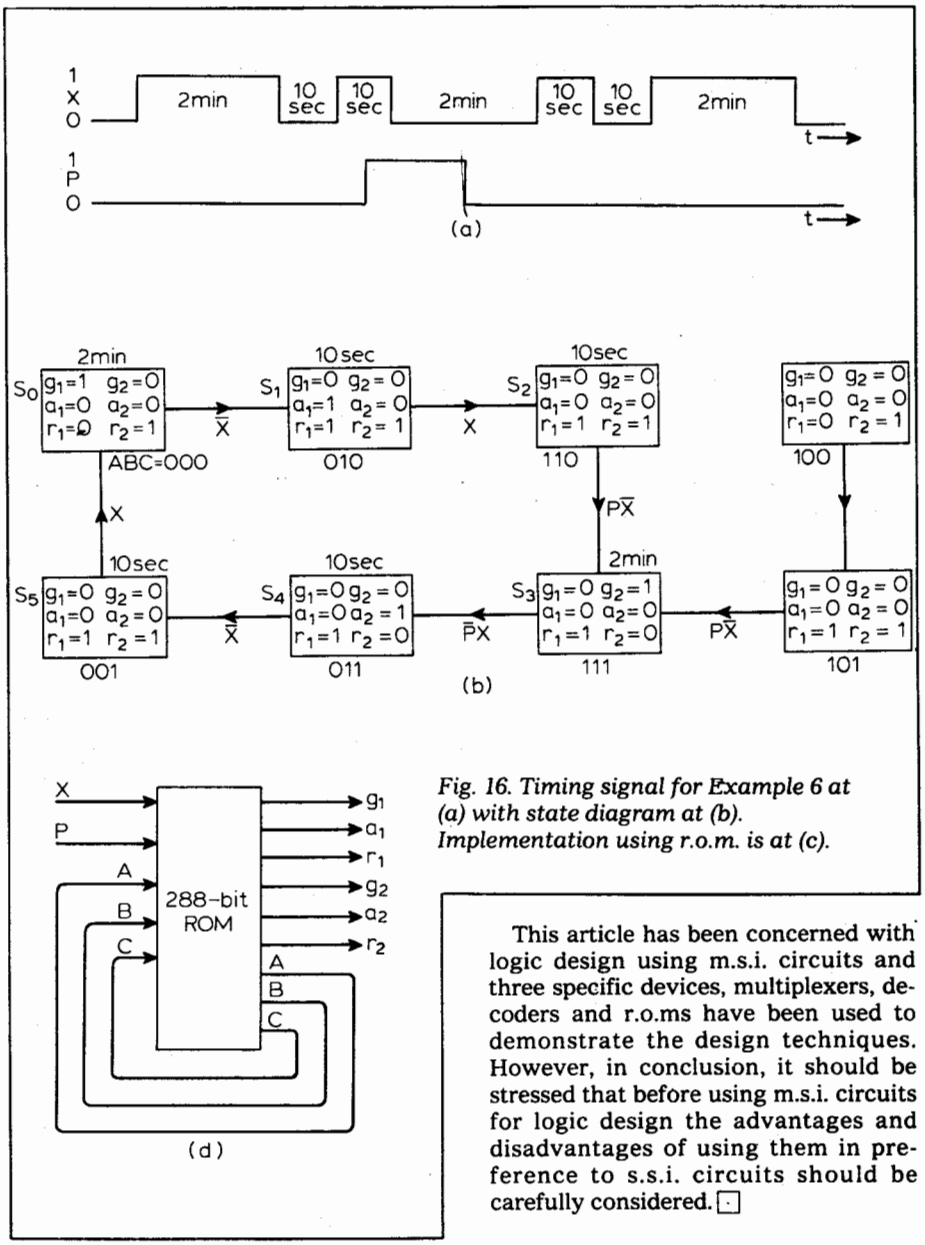


Fig. 16. Timing signal for Example 6 at (a) with state diagram at (b). Implementation using r.o.m. is at (c).

Example 6. A road intersection is controlled by a set of traffic lights. For each road the light sequences are tabulated below:

Road 1—green amber red red red red
Road 2—red red red green amber red
The lights are driven by a timing signal X and a synchronisation signal P as shown in Fig. 16(a).

Step 1. I/O characteristics. These are described in the statement of the problem.

Step 2. Internal characteristics. A suitable state diagram is shown in Fig. 16(b).

Step 3. State table. This is shown in Table 3 and is displayed in a suitable form for r.o.m. implementation.

Step 4. Circuit implementation. This is shown in Fig. 16(c). In practice, r.o.ms are manufactured in standard sizes and a suitable r.o.m. or combination of r.o.ms would have to be chosen from those available.

This article has been concerned with logic design using m.s.i. circuits and three specific devices, multiplexers, decoders and r.o.ms have been used to demonstrate the design techniques. However, in conclusion, it should be stressed that before using m.s.i. circuits for logic design the advantages and disadvantages of using them in preference to s.s.i. circuits should be carefully considered. □