Date: November 29, 1967
From (location & U.S. mail address): Advanced Computing Systems, Menlo Park, California
Dept. & Bldg.: 988/031
Telephone Ext.:

IBM

Subject: Cover Letter for Preliminary Distribution of Logical Design Memorandum

Reference:

To: Mr. S. F. Anderson
Mr. B. O. Beebe
Dr. C. V. Freiman
Mr. M. E. Homan
Mr. B. J. Mooney
Mr. R. J. Robelen
Dr. H. Schorr
Dr. E. H. Sussenguth
Mr. W. P. Wissick

A memorandum describing basic ACS logical design conventions is enclosed.

On joining ACS engineering, I found that there was no single convenient source of this information. Some of the information was not documented in any available references.

Since most of the designers use different notations and conventions, it proved to be a surprisingly time consuming and confusing process to learn the precise details of this very simple basic material. Many of the designers related to me that they had had similar initial experiences.

At that time I made some notes for my own personal use. I have since formed these into a memorandum in the hope that it might prove useful to other newcomers to ACS engineering. It might also be useful to members of other ACS departments.

If you have any comments, criticisms, or discover any errors needing correction, please contact me about them. I will then be able to get the memorandum into shape so that it might be useful during the coming expansion of Dept. 988.

L. Conway

L. Conway

LC:aw

328

November 29, 1967
Advanced Computing Systems
Menlo Park, California
988/031

Subject: ACS Logical Design Conventions: A Guide for the Novice

References:
1. ACS Circuit Manual, February 23, 1967.
2. ACS Packaging Manual, July, 1967.
3. DRKS User's Manual, R. T. Blosk, December 5, 1966.
4. McCluskey and Bartee, A Survey of Switching Circuit Theory, McGraw-Hill, 1962.

To: FILE

L. Conway

L. Conway

LC:aw

CONTENTS

Introduction:

This memorandum describes the various rules and conventions for ACS logical design. The material presented is elementary in nature, but is basic to all ACS logical design.

A description is given of the logical functions of the ACS circuits available to the designer and of the various rules governing the use of these circuits in logical design. A number of different notations are in current use for writing the logical equations for these circuits and for drawing the diagrams of logical circuitry. Some of these different notations are illustrated and explained. Elementary logical design--the transformation from equations to circuits--is briefly described.

If we were designing in AND-OR logic with few restrictions, this memorandum would be unnecessary. However, we are usually designing with NOR-NOR or NOR-OR logic. The physical properties of the circuits force a number of restrictions in addition to simple fan-in and fan-out rules. The fact that designs eventually input a Design Record Keeping System (DRKS) has produced additional conventions and design notation.

These factors have led different designers to use different conventions for writing logical equations and drawing logic circuit diagrams, and to use different logical design techniques. It is true that at the time designs are input into DRKS, they all will be described in the same formal system. However, up to that time most designs will exist in the form of equations and diagrams in the "shorthand" of the originating designer. The newcomer may therefore become confused when attempting to decipher the designs of different engineers until he fully understands the fundamentals from which their different "shorthand" techniques originated.

These fundamentals are presented in this memorandum in the hope that they may assist the newcomer to ACS engineering in his first design efforts and serve as a reference for those outside of engineering who may wish to study some particular logical design in detail.

The newcomer should also study the listed references before undertaking any serious design. This memorandum was formulated from these references, but does not attempt to cover many important topics contained in them. Of particular importance is the information on circuit delays in the ACS Circuit Manual and information on wiring rules in both the ACS Circuit Manual and the ACS Packaging Manual. The DRKS User's Manual specifies the final form in which designs are to be placed.

The ACS Logical Circuits:

This section describes the logical functions of the circuits and connections available to the ACS logical designer. Truth tables and equations are given describing the logical functions. The various conventions, restrictions, and limitations of each circuit are listed.

The truth tables use 0 and 1 as symbols, and these are related to the actual physical voltages in the circuits as follows: 1 symbolizes positive (or ground), and 0 symbolizes negative voltages.

The Current Switch:

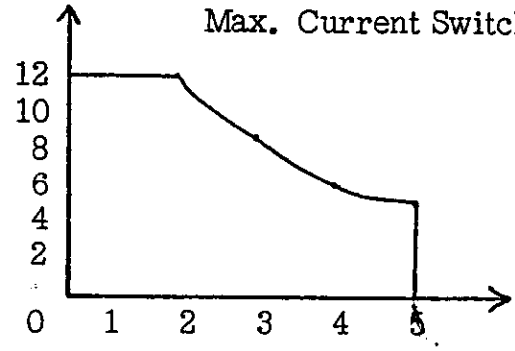
A	B	X	Y
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

$X = \bar{A} \cdot \bar{B}$

$Y = A + B$

Note the significance of the positions in the circuit symbol of the outputs X and Y. The top output X is the NOR of the inputs, and is often called the "out of phase" output. The bottom output Y is the OR of the inputs and is often called the "in phase" output. Note that $Y = \bar{X}$.

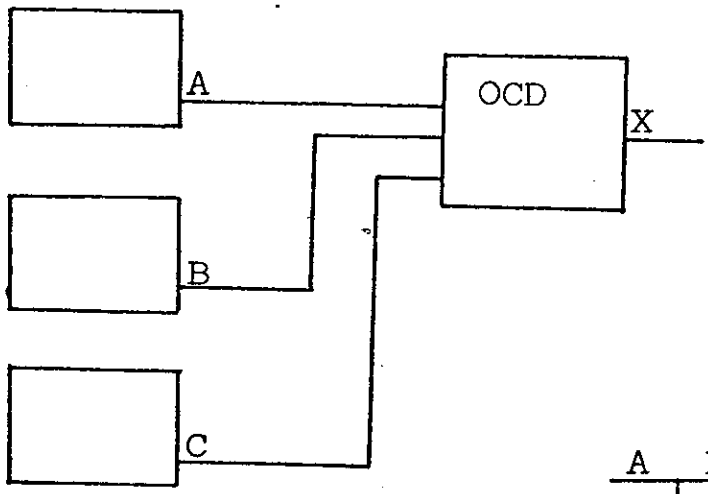
Fan-in: Current switch inputs are outputs of emitter followers or emitter follower dot circuits (see description of e.f. dot later in this section). The maximum number of inputs for a given current switch is a function of the maximum fan-in of those e.f. dot circuits forming the inputs. This function is as follows:



For example, if the e.f. dots feeding a current switch had no more than two inputs each, then the current switch would have a maximum fan-in of 12. However if one of the e.f. dots had a fan-in of five, then the current switch would have a maximum fan-in of five.

Fan-out: The outputs always pass through emitter followers. The fan-out is thus determined by the fan-out of the emitter followers. The maximum fan-out of the emitter follower (emitter follower dot) is 12. See emitter follower dot description later in this section.

The Orthogonal Collector Dot:



$X = A \cdot B \cdot C$

Orthogonality Restriction:

No two inputs may be 0 (negative)

A	B	C	X
0	0	0	N.A.
0	0	1	N.A.
0	1	0	N.A.
0	1	1	0
1	0	0	N.A.
1	0	1	0
1	1	0	0
1	1	1	1

(N.A. = not allowed)

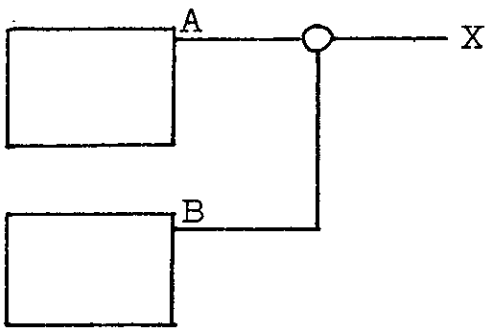
The orthogonal collector dot is the connection of collector outputs of current switches (the in phase outputs) before passing through an emitter follower. This connection performs the AND function--with the important restriction that no two of the inputs may be simultaneously negative. This is called the orthogonality restriction. In the above three input case the restriction requires that: $A \cdot B + A \cdot C + B \cdot C = 1$.

The ultimate physical restriction is somewhat weaker than the stated logical orthogonality restriction. A maximum time of .5 ns of non-orthogonality is allowed, which covers variations in signal delays. See Reference 1, Page 2.

Fan-in: ≤ 5

Fan-out: See fan-out for current switch. Same description applies here.

The Emitter Follower Dot:



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

$X = A+B$

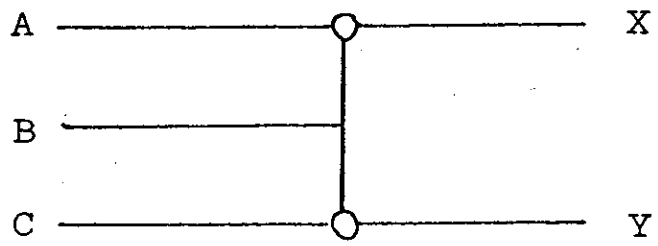
The emitter follower dot circuit is the "dotting" or connection of current switch outputs A and B after their emitter followers. The function performed is OR with no restrictions except fan-in and fan-out. Note that we might have a line connected to an e.f. dot which came from an emitter follower which followed a collector dot.

Fan-in: ≤ 5

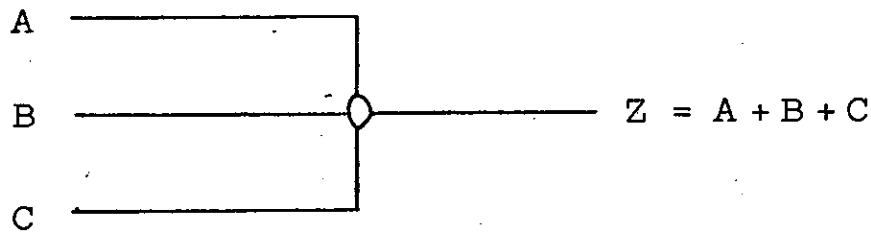
Fan-out: ≤ 12 (try for ≤ 8)

Note: The meaning of "dot" in orthogonal collector dot and emitter follower dot is that the inputs are actually wired or connected together. Thus the O.C. Dot and E.F. Dot are not circuit elements, but are connections of wires which perform particular logical functions on the signals carried by those wires due to their locations in the circuitry (see Reference 1).

Therefore we cannot think of applying the same input to two separate dots. For example, the following diagram is incorrect for it shows B as an input to two separate E.F. Dots, treating these dots as independent circuit elements and expecting that X = A + B and Y = B + C:



Since the E.F. Dot is merely a connection of the inputs, the only possible interpretation of the E.F. Dot of A, B, C is that they are all wired together as follows:



Most beginning logical designers will have had considerable experience in design using AND, OR, and COMPLEMENT "gates" as circuit elements. It is natural for the designer to write logical equations for such designs using AND, OR, and COMPLEMENT logical operators. The primary content of switching theory consists of operations on logical functions expressed using these operators.

However in ACS the actual logic circuit implementation of a design is usually in NOR-NOR or NOR-OR logic.

It turns out that the usual OR-AND or AND-OR formulations of logic equations can be easily transformed and converted directly to the corresponding NOR-NOR or NOR-OR circuitry (see Section 5 for these techniques).

Therefore, for convenience most ACS designers express logical functions using OR, AND, and COMPLEMENT logical operators. The usual minimization techniques of switching theory may then be applied to these formulations before transformation into the final NOR-NOR or NOR-OR form (the circuit diagram itself).

The following different symbols for the logical operators are currently in use by different ACS designers:

AND(A, B): = $A \cdot B = AB = A \wedge B$

OR(A, B): = $A + B = A \vee B$

NOT(A): = $\bar{A} = A' = -A$

These variations in basic operator symbols from one designer to another should cause the newcomer no confusion.

There is one practice, stemming from the ultimate NOR-NOR or NOR-OR implementation of logical functions, which will definitely cause the newcomer confusion if it is not fully understood. It is a common practice in ACS to use two different symbols for complement in the same logic equations. Thus we may see both $\bar{A}$ and -A, or perhaps even $-\bar{A}$ in some equation. The reason some designers use both forms derives from the inversion of variables when using NOR-OR logic. One symbol is usually reserved for true logical complements and the other symbol (usually -) is used to mark variables or expressions which are complemented because they are at an intermediate point in the logic (see Section 5).

It is easy for the newcomer to think that -A must mean something other than $\bar{A}$, perhaps having something to do with negative voltages. This happens easily because some designers also mark uncomplemented variables with + in some cases (using the symbol V for OR).

However, remember that -A is equivalent (logically) to $\bar{A}$, and that +A is equivalent (logically) to A. Some designers might argue otherwise, but that is because they have attached some additional heuristic values to these different symbols for complement in order to aid their design efforts. Thus, any difference between -A and $\bar{A}$ is only a heuristic difference, not a logical difference.

For example, the following equations all equate X with the same logical function of A, B, C:

$$\bar{X} = A \cdot B \cdot \bar{C}$$

$$-X = A . B . \bar{C}$$

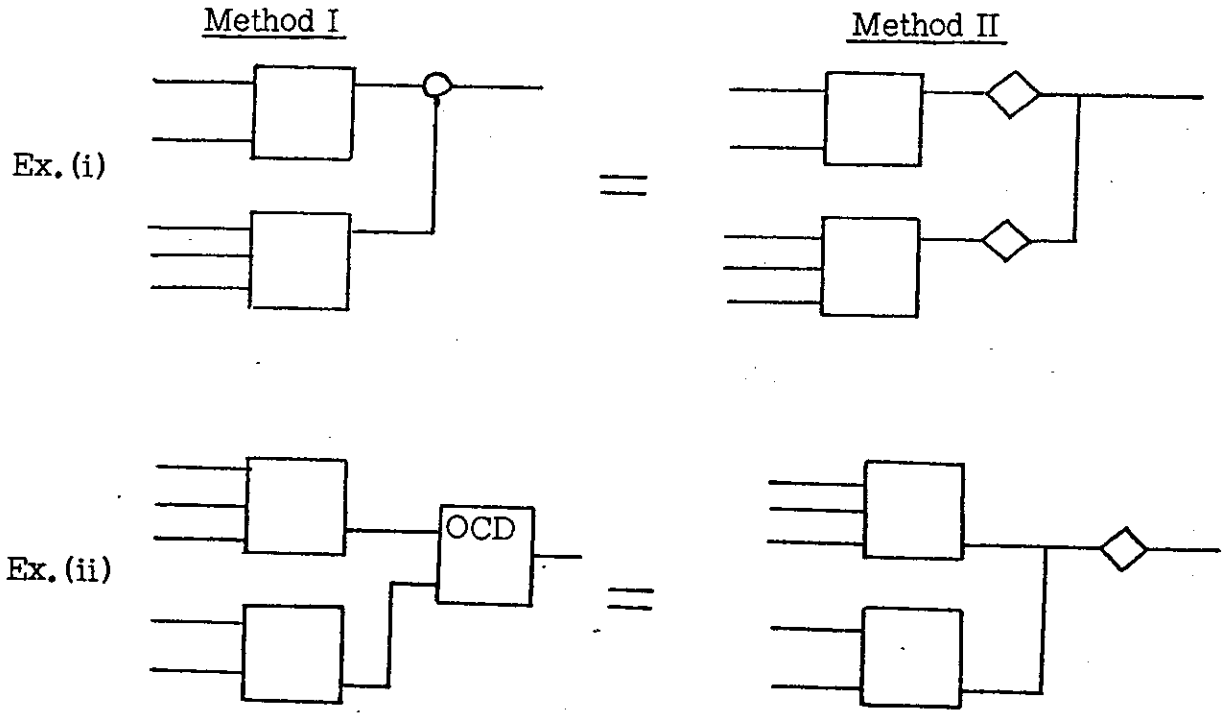
$$+X = -(A \cdot B \cdot \bar{C})$$

After gaining some experience with NOR-NOR and NOR-OR circuit implementations of logical functions, the newcomer may find that it aids him in his design efforts to use $\pm$ symbols in addition to the usual complement symbol.

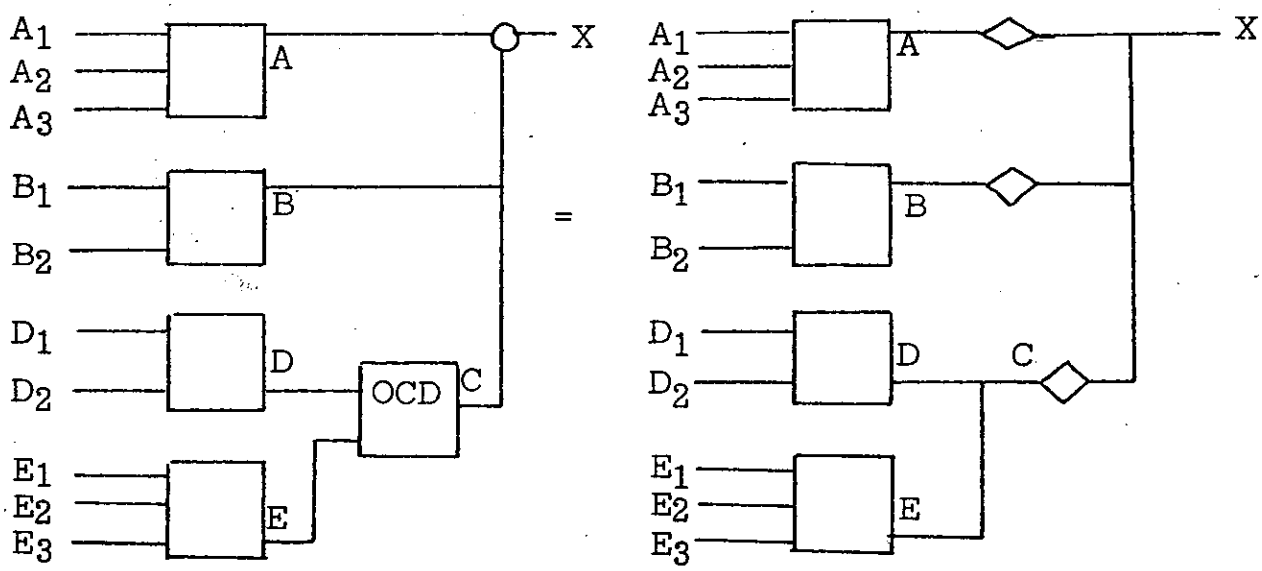
It is not necessary to use these extra symbols and the corresponding heuristic techniques. They may assist those designers who prefer to design in an informal manner. One may, alternatively, design in a formal manner without ever using heuristics. However, all ACS designers should know about the techniques used by other designers and the resulting additional notation so that successful communication is possible between different designers.

A number of different conventions are in current use for drawing logic circuitry composed of ACS circuits. Different designers may use different symbols for the basic circuits. Some designers indicate emitter followers while others do not.

Two methods are shown below which serve to illustrate some of the possible variations in circuit diagram techniques. The two methods differ primarily in the way in which the orthogonal collector dot is symbolized. When the O.C.D. is symbolized by a labelled block, it is not necessary to indicate emitter follower positions. However, if only a simple dot is used to symbolize O.C.D., then it is necessary to show emitter follower positions (symbol: ◊) in order to avoid confusing O.C.D. with emitter follower dot.



Ex. (iii)



$$A = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3$$

$$B = \bar{B}_1 \cdot \bar{B}_2$$

$$D = D_1 + D_2$$

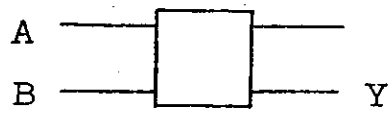
$$E = = E_1 + E_2 + E_3$$

$$C = D \cdot E$$

$$X = A + B + C = A + B + D \cdot E$$

$$X = \bar{A}_1 \cdot \bar{A}_2 \cdot \bar{A}_3 + \bar{B}_1 \cdot \bar{B}_2 + (D_1 + D_2) \cdot (E_1 + E_2 + E_3)$$

In the examples shown above, the basic symbols for the current switch are all the same. Sometimes, however, designers will place a letter inside the current switch symbol to indicate the logical function that it performs. This practice may lead to considerable confusion for the newcomer for two reasons: (i) different function names are often used for the current switch by the same designer, (ii) the output phase of the switch to which the name refers is usually assumed to be obvious and is not explicitly indicated. Let us study these conventions in some detail to avoid confusion.



For the current switch shown, the output Y equals the OR of the inputs A, B:

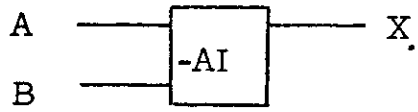
$$Y = A + B$$

Suppose we complement both sides of the equation to yield:

$$\overline{Y} = \overline{A} \cdot \overline{B}$$

We thus find that the complement of Y equals the AND of the complements of A, B. Now, even though this equation expresses Y as the same function of A, B, many designers call this the "MINUS AND" function. Thus one may see different current switches in the same circuit diagram labelled in both of the following ways:

These circuit symbols both stand for current switches and both perform exactly the same logical function on their inputs. Some designers choose to view them differently depending on whether or not complemented variables appear as inputs. This is another heuristic aid to the designer. Clearly it is not necessary to view the circuit element in these two different ways. It is just that some designers find that this technique assists them in their design efforts. Note that the output phase in the above examples to which the function name applies is found to be the "in"phase. This is not explicitly indicated, but is "obvious" because of the known function of the switch. This sort of duplicate naming can be carried further if desired. For example:



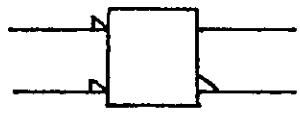
Here we have named the function as "MINUS AND INVERT." The meaning is that the output X is the complement of the MINUS AND function.

This duplicate naming of functions may sometimes be applied to the other circuit connections. The emitter follower dot performs an OR function and so may also be thought of as performing the "MINUS AND." The orthogonal collector dot performs the AND function and so may be thought of as performing a "MINUS OR" function.

It is important to note that "MINUS AND" and "MINUS OR" are not equivalent to the logical functions NAND and NOR. It is unfortunate that the use of MINUS (-) here conflicts with our previous definition of (-) as equivalent to complement. One might therefore be led to believe that MINUS AND (-A) is equivalent to $\overline{AND}$ (and thus equivalent to NAND), which it is not.

"MINUS AND" and "MINUS OR" may best be viewed by the beginner as merely other names for OR and AND, used by some designers for their heuristic value when circuit input variables are in complemented form.

There is another circuit diagram symbol which the newcomer will occasionally see and which is bound to confuse him. This is the "wedge" symbol appended to certain circuit block inputs/outputs. Wedges might be found on a current switch symbol as follows:



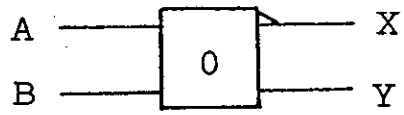
These wedges have no functional meaning to the logical designer. They do not change the identity or function of the circuit element. The wedges are normally produced by the DRKS system and automatically affixed to the circuit blocks appearing on the DRKS sheets. The wedges appear to be used primarily by CE's who service the hardware. Wedges appear mainly on the MACRO circuit blocks defined and used in DRKS. To quote Reference 3, Section 2.2.8.5:

"Wedges will be printed in the edge of box print position for all input or output lines that are in the "down" signal condition when the logic block function is being performed. The designer need not draw these wedges on his diagram. They will be automatically inserted by DRKS, according to the block definition in the macro file, when the sheet is printed."

In other words, given a circuit block performing some logical function as stated by a logical equation, DRKS affixes wedges to those input and output lines which must be down (0; negative) when both sides of the equation are TRUE (1).

Examples: note that although both examples use the same circuit, the wedge placement is different. This is because wedge placement depends on the statement of the function of the circuit. If we complement both sides of the equation defining the circuit, then the wedge placement changes.

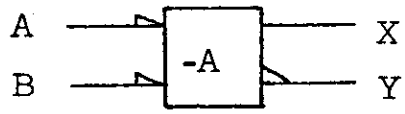
(i) Current Switch as an "OR"



$Y = A + B$

When both sides of the equation are TRUE (1), then Y must equal 1, neither A nor B must equal 0, and since $X = \overline{Y}$, then X must equal 0.

(ii) Current Switch as a "MINUS AND":



$\overline{Y} = \overline{A} \cdot \overline{B}$

When both sides of the equation are TRUE (1), then Y must equal 0, A must equal 0, B must equal 0, and since $X = \overline{Y}$, then X must equal 1.

Now, even though the wedges have no functional meaning, some designers may attach them to the circuit blocks in their circuit diagrams. This is especially true when MACRO circuit blocks are used. A reason for this is that the wedges can be used as a memory aid in locating particular inputs and outputs on the MACRO blocks which have many input/output lines. But remember that there is no additional information contained in the wedges. DRKS can produce them automatically when given the function of the block.

Elementary Logic Design

Logic design in ACS, and in any case where implementation will be made in real circuitry, is essentially an iterative procedure consisting of making a design, then testing that design against technological restrictions, then redesigning and retesting until a valid design is found.

First the logical functions to be implemented in the design are formulated in a set of logical equations. Then the set of equations is operated upon to minimize the logic according to some selected criteria such as number of circuits and/or number of circuit levels. Note that the minimization may be performed on the equations (which use AND, OR, NOT operators) even though the final implementation may be in NOR-NOR, or NOR-OR logic (see Reference 4, page 101).

Next, the minimized equations are examined to determine if all circuit restrictions are satisfied. These restrictions, such as fan-in and fan-out, can be checked while the design is still in the form of logical equations.

If the restrictions are not satisfied, we must iterate by going back and perhaps reformulating the equations and minimizing again, until equations are found which satisfy the restrictions.

At this point we can convert the equations directly into a logical circuit implementation. Descriptions of procedures, both formal and heuristic, for performing these conversions follow later in this section.

Now, if the design specification is beyond the preliminary stage and unlikely to be changed, then the circuitry must be checked against all the many and complex wiring and packaging rules. If the design cannot be wired or packaged as is, then additions or changes may have to be made, or perhaps another entire design iteration may be required.

Implementing Logic Equations in ACS Circuitry:

With a little experience a designer can directly sketch out the logic circuitry to implement some logical function. This is particularly easy to do if AND-OR or OR-AND logic circuits are used. For these cases the designer can place the equation for a function in "sum of products" or "product of sums" form and transform directly to a circuit diagram.

In the ACS technology, however, we have available only a restricted form of AND circuit (the orthogonal collector dot; inputs must be orthogonal). Thus OR-AND logic is seldom used. Instead, we normally use NOR-NOR or NOR-OR logic.

The beginner should therefore learn the transformations for quickly and automatically drawing the circuit diagrams for NOR-NOR and NOR-OR logic implementing a logical function. This material is covered in detail in Reference 4, pages 94-102. A summary is presented here for reference:

Let us draw the logic circuitry to implement the function

$$f = (a + b)(b + \bar{d})(a + c) = ab + bc + a\bar{d}$$

Ex. (i): NOR-NOR logic circuit implementation:
(2 circuit levels: current switch to current switch)

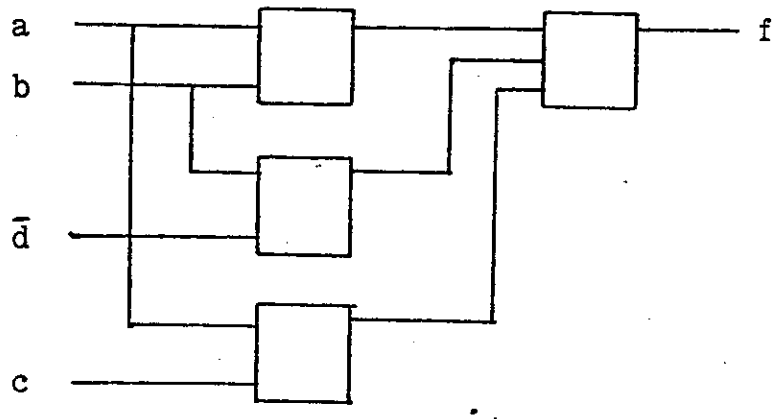
Step 1: Express function in product of sums form:

$$f = (a + b)(b + \bar{d})(a + c)$$

Step 2: Let NOR $(a, b) = \overline{(a + b)}$. Transform the equation to NOR-NOR form by simply replacing all OR, AND operators with NOR operators, leaving the variables in the original order and form:

$$f = \text{NOR}(\text{NOR}(a, b), \text{NOR}(b, \bar{d}), \text{NOR}(a, c))$$

Step 3: Draw the logic circuit diagram directly from the equation in Step 2.



Clearly we may proceed directly from Step 1 to Step 3. The NOR-NOR logic uses the same connections of circuits to implement a function as does OR-AND logic. We merely replace all OR and AND circuits with NOR circuits

EX. (ii): NOR-OR logic circuit implementation:
(1 circuit level: current switch to E. F. Dot)

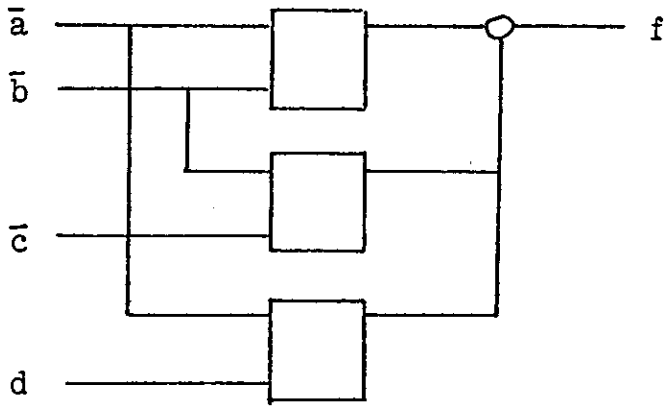
Step 1: Express function in sum of products form:

$$f = ab + bc + a\bar{d}$$

Step 2: Transform the equation to NOR-OR form by complementing each variable and replacing the AND operators with NOR operators:

$$f = \text{NOR}(\bar{a}, \bar{b}) + \text{NOR}(\bar{b}, \bar{c}) + \text{NOR}(\bar{a}, d)$$

Step 3: Draw the logic circuit diagram directly from the equation in Step 2:



Here also we see that it is easy to proceed directly to Step 3 from Step 1. The NOR-OR logic uses the same connections of circuits to implement a function as does AND-OR logic. We merely replace the AND circuits with NOR circuits and use the complementary inputs.

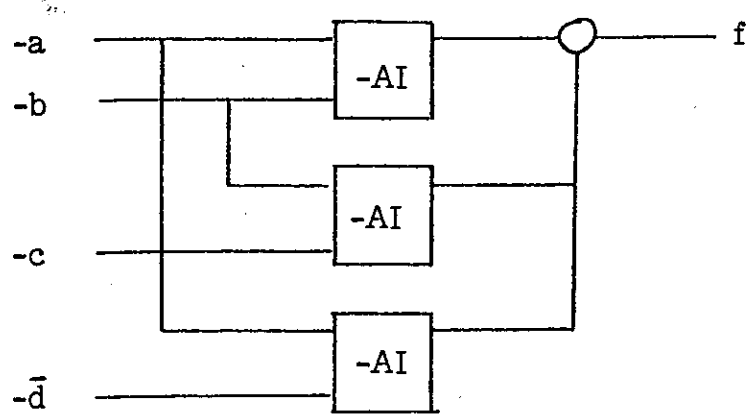
Heuristic Design Techniques:

The extensive use of the NOR-OR logic has caused the evolution of many heuristic design practices, including the use of two different symbols for complementation and the duplicate naming of the logical function performed by the current switch.

To clarify all the points developed in this memorandum concerning heuristic design techniques, let us implement the same function f of the preceding examples in NOR-OR logic using one of the heuristic techniques rather than the formal, automatic procedure just described.

Suppose we have available as inputs both phases of a, b, c, d, i.e., $\pm a$, $\pm b$, $\pm c$, $\pm d$ and wish to form $f = ab + bc + a\bar{d}$.

Using minus (-) inputs we can use "MINUS AND INVERT" circuits to obtain the terms ab, bc, and $a\bar{d}$. Then we can use the emitter follower dot to OR these terms.



Clearly this is the same circuit as that developed in the preceding formal NOR-OR example. However, here the designer is thinking directly in terms of pseudo AND-OR logic by renaming the functions of his circuit elements and making a sequence of appropriate complementations.

The beginner is warned not to attempt to imitate such techniques at first. The heuristic techniques, used by the novice as though they were formal methods, will prove far more unwieldy and confusing than the previously illustrated formal techniques. The novice using these heuristics will put a great deal of effort into the essentially trivial process of forming circuit diagrams from logic equations.

When the time comes that the designer has a good "feeling for" NOR-NOR, NOR-OR logic design, he may then find that some of the existing heuristic techniques are useful. Experienced ACS designers can sometimes find "tricky" implementations using these techniques which have less delay or lower circuit count than those derived by formal approaches. This occurs especially when both the O.C. Dot and E.F. Dot are used in the implementation.