SAVING ROMs IN HIGH-RESOLUTION DOT-MATRIX DISPLAYS AND PRINTERS

INTRODUCTION

Conventionally, the number of bits in a digital character generator's read only memory is proportional to the number of dots in the character matrix. That is, the ROM array ordinarily doubles and redoubles in size as one scales up the resolution or changes from an upper-case to an upper-case/lower-case font.

Fortunately, such progressions may not be required. Reorganizing the ROMs to suit the specific application often save thousands of bits and allows the designer to use smaller, faster, more economical monolithic ROMs. As a simple example, expanding the array in 32-character subsets rather than the more conventional 64-character subsets will enhance performance and save up to 25% of ROM capacity in typical UC/LC applications.

Savings much greater than 25% are possible when the matrix size reaches a point where several monolithic ROMs are needed to store the font. We have found a two-stage, column-generation approach called "intermediate coding" to be much more efficient than straightforward dot-matrix generation. It exploits the fact that column patterns tend to become highly redundant as the matrix size increases.

One version of this new technique automatically proportions character widths as in letterpress printing. This gives each character a more natural shape and eliminates the irregular spacings usually seen around "I" and other narrow characters. Yet the control logic is simple and the ROM savings approach 40% at typical font sizes.

Such advantages are available immediately, without development of special ROMs. The designs can be implemented with standard MOS or bipolar ROMs currently in production. In fact, intermediate coding broadens the cost/performance options by allowing a combination of MOS and bipolar ROMs to be used.

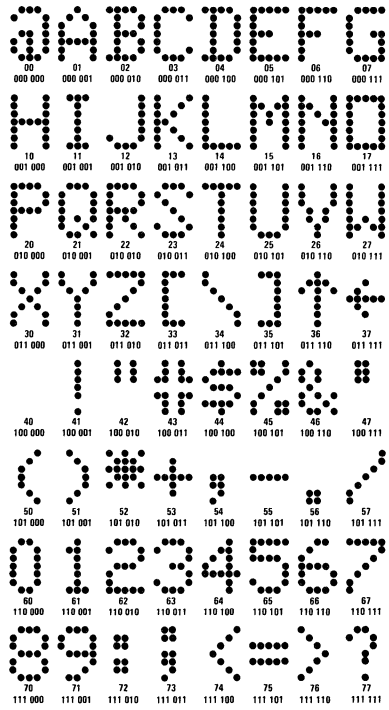
DOT-CHARACTER FONTS

Dot-character styles ranging in complexity from 5 x 7 to 12 x 24 or more dots per character have been developed to meet the human-engineering standard of various industries using digital displays and printers. The more popular sizes are listed in Table I.

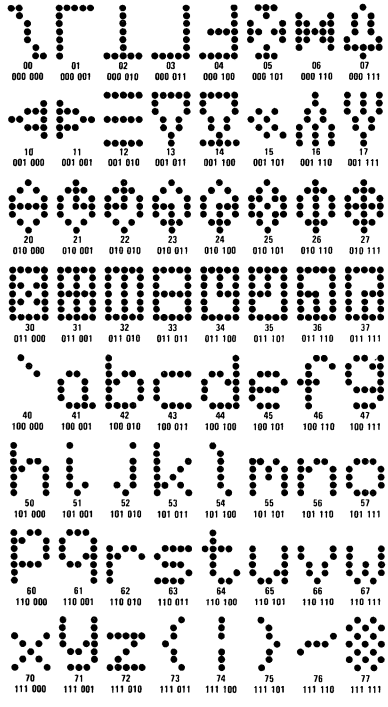
The 5 x 7 fonts, such as Figure 1, lead in applications volume due to their use in low-cost data

TABLE I. Typical Dot-Matrix Character Fonts

SIZE AND SCANNING	DOTS PER CHARACTER	THEORETICAL CHARACTER ROM	DESIGN EFFECTIVENESS	PRACTICAL DESIGNS
5 x 7				
Horizontal	35	64 x 7 x 5 = 2 — 1/2k	1.00	Fig. 3
7 x 5				
Vertical	35	64 x 5 x 7 = 2,560	1.00	Fig. 3
7 x 9				
Horizontal	63	64 x 9 x 7 = 4,032	0.67	Fig. 6
9 x 7				
Vertical	63	64 x 7 x 9 = 4,032	1.00	Figs. 5 & 8
7 x 12 & 8 x 12				
Horizontal	96	64 x 12 x 8 = 6,144 and 96 x 12 x 8 = 8,216	1.00 1.00	Figs. 6 & 7
12 x 7 & 12 x 8				
64 Character Vertical	96	64 x 8 x 12 = 6,144	1.00	Fig. 8C
96 Character Vertical	96	96 x 8 x 12 = 9,216	1.00	Fig. 8B
12 x 16				
64 Character Horizontal	192	64 x 16 x 12 = 12,288	1.50 for Fig. 9A	Fig. 9A
96 Character Horizontal	192	96 x 16 x 12 = 18,432	1.50 Fig. 9	Fig. 9B
16 x 12				
64 Character Vertical	192	64 x 12 x 16 = 12,288	1.50 for Fig. 9A	Figs. 6, 7, & 9
96 Character Vertical	192	96 x 12 x 16 = 18,432	1.50 for Fig. 9B	
24 x 12				
64 Character Vertical	288	64 x 12 x 24 = 18,432	2.00 for Fig. 9B	Figs. 6, 7, & 9
64 x 13 x 10 to 64 x 13 x 16				
64 Character Variable Font Width	208	64 x 13 x 16 = 13,312	3.06 for Fig. 10	Fig. 10



(A) Upper-Case Font



(B) Lower-Case Font

FIGURE 1. ASCII Full Set Font of 128 5 x 7 Characters

interface terminals (although some terminal manufacturers are going to larger sizes in response to complaints that 5 x 7 presentations cause eyestrain). In other applications, a standard is often set by older printing techniques. To cite a few examples: business-machine users are accustomed to typewriting; advertisers want characters with "sales appeal" on their billboard displays; scoreboards and traffic-control signs must be read easily at a distance; and electronic printing systems may have to simulate several metal type fonts.

The matrix size is frequently enlarged to improve lower-case character definition in UC/LC applications. A 5 x 7 font typically grows to 7 x 9 for UC and 7 x 12 for LC, as in Figure 2. Likewise, 7 x 9 is expanded to 8 x 12 and 12 x 16 to 12 x 24 for lower-cases.

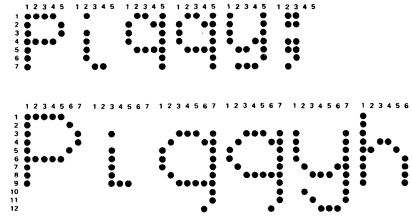


FIGURE 2. UC/LC Characters at 5 x 7 and 7 x 12 Matrix Sizes

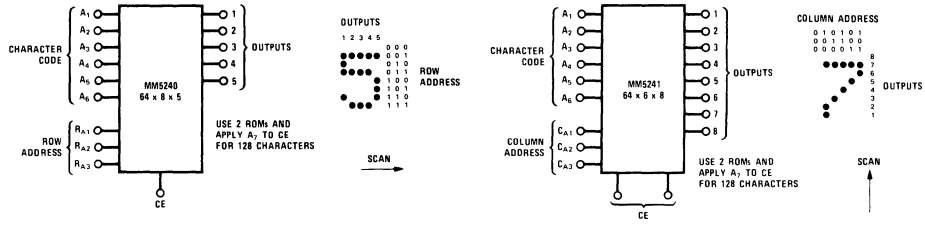
At 5 x 7, it is most economical, as a rule, to program a "full set" of 128 UC/LC characters in standard character-generator ROMs. The full set in Figure 1 is stored in two 64-character MOS ROMs. This provides a mass-production base and equalizes access times. If the 32 special symbols generated with the ASCII control codes are not usable, they are simply blanked by disenabling the lower-case ROM when the seventh ASCII bit is "0." But if the font is scaled up to simulate typewriting, for example, this practice becomes wasteful since 96 characters would suffice.

Another complication is that many specialized font sizes, such as 11 x 9, do not fit neatly into standard ROMs made in building block sizes. In other words, one cannot store the font in a minimum-sized ROM array without paying the extra costs of custom ROM development or specialized low-volumn ROMs.

CHARACTER-GENERATOR ROMs

Consequently, character-generator ROMs have been developed that adapt to a variety of font sizes. They may not exactly fit the theoretical matrix array at odd sizes. but that is easily offset by the economy of parts standardization.

Two such MOS ROMs are outlined in Figure 3 with their addressing for 5 x 7 horizontal scanning and 7 x 5 vertical scanning. The vertical-scan subsystem in Figure 4 shows the amount of support logic typically required in a display.



(A) 5 x 7 Horizontal-Scan, 64-Characters

(B) 7 x 5 Vertical-Scan, 64-Characters

FIGURE 3. MM5240 and MM5241 Standard Character-Generator ROMs

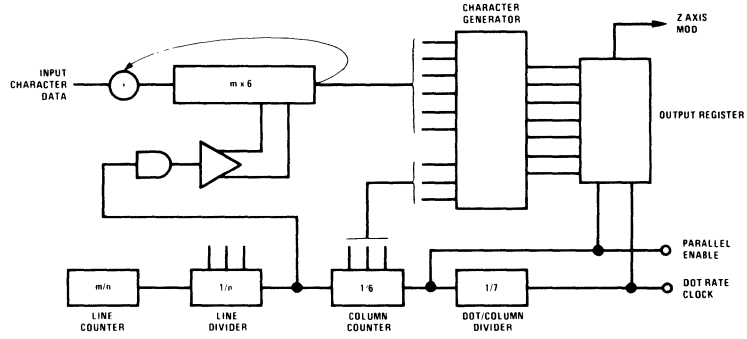


FIGURE 4. Typical 7 x 5 Vertical-Scan Display Generator Subsystem

The MM5240 expands straightforwardly in 64-character increments to larger fonts, such as the 9 x 7 or 10 x 8 arrays in Figure 5A. An expansion such as Figure 5B would be used to provide a full set UC/LC font. These expansions keep the character rate the same as at 5 x 7, whereas doubling the size of each monolithic ROM would not.

32-CHARACTER BLOCKS

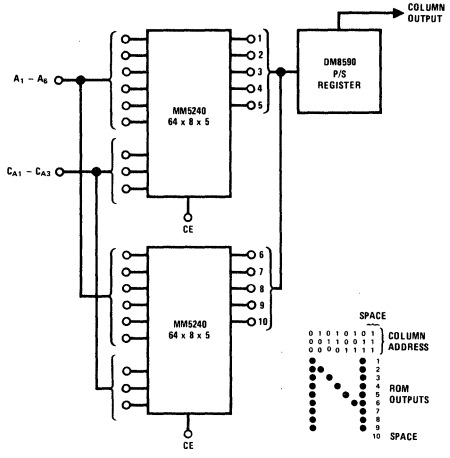
A similar expansion of the MM5241, as in Figure 6A, would provide 7 x 9 to 8 x 12 horizontal-scan fonts. However, the direct 64-character expansion places a ROM-enabling operation in the middle of the character. Such operations are common in large-font generator designs.

A simple solution to this problem is to "steal" a character-address input, use it as a row-address input, and then use a chip-enable input as a character input (Figure 6B). This provides a 32-character or 64-character block enabled during the between-characters spacing interval. A 32-character block would be the only ROM required in a system using only numbers and symbols.

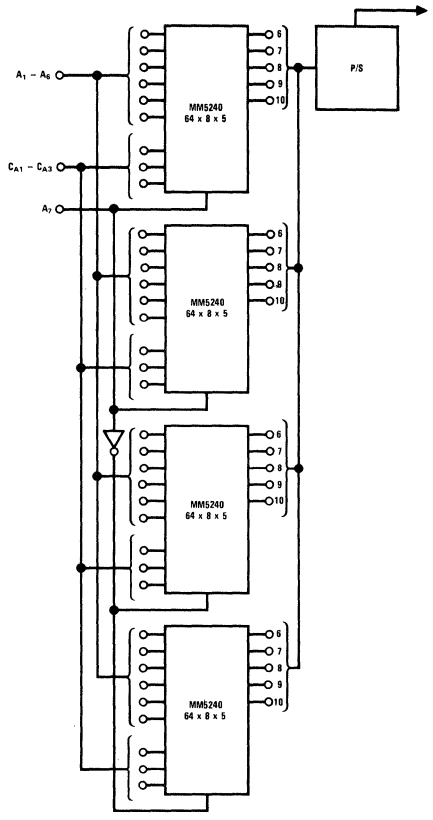
However, the chief attraction of this conversion is in UC/LC applications. Figure 7 shows how to use three ROMs to generate 96 7 x 9 to 8 x 12 horizontal-scan characters—a 25% savings compared with a "full set" expansion. The chip-enable inputs are programmed to sense the sixth and seventh character-address bits. External decoders aren't needed.

If each ROM in Figure 7 is replaced with a parallel assembly of three ROMs (24 outputs), the result is a 24 x 12, 96-character vertical-scan generator with the same character rate as at 8 x 12. In other words, the 32-character approach maintains the benefits of parts standardization and performance up to a very high resolution.

Other ROMs can be used in this fashion. In Figure 8, the MM5227 TRI-STATE® and MM5288 256 x 12 ROMs are shown in expansions that complement those of the MM5241. These ROMs provide access times well under a microsecond. For rates in the nanosecond range, general-purpose bipolar ROMs with four or eight outputs, such as the DM8597 256 x 4 and DM8596 512 x 8 can be worked into similar organizations.

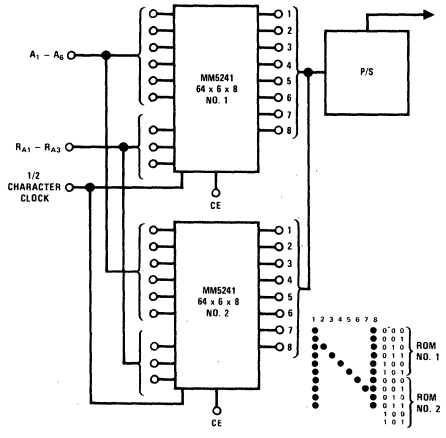


(A) 9 x 7 or 10 x 8 Vertical-Scan, 64-Characters

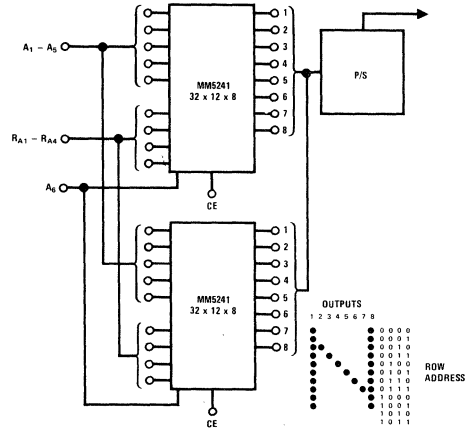


(B) 9 x 7 or 10 x 8 Vertical-Scan, 128-Characters

FIGURE 5. Expansion of MM5240 to Larger Fonts



(A) 7 x 9 to 8 x 12 Horizontal-Scan, 64-Characters



(B) 7 x 9 to 8 x 12 Horizontal-Scan, 64-Characters

FIGURE 6. Conventional and Improved MM5241 Expansions to 8 x 12

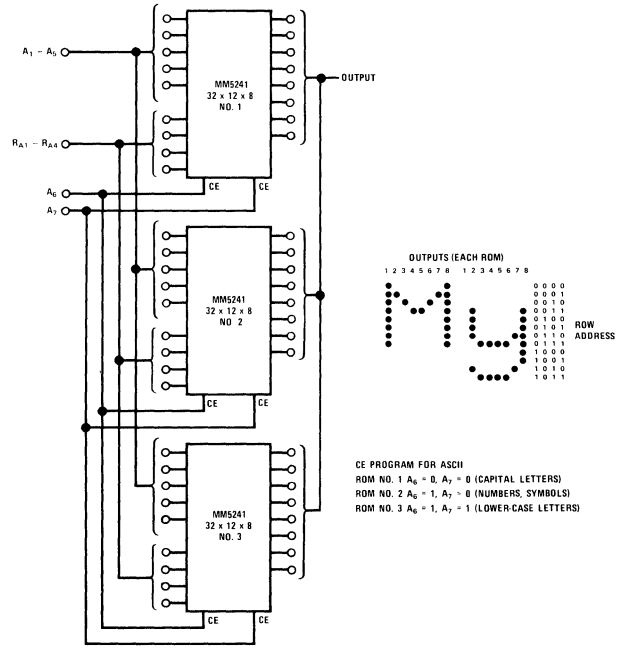
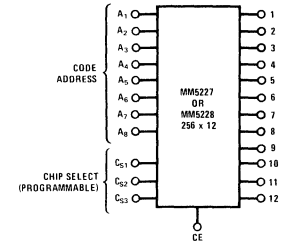
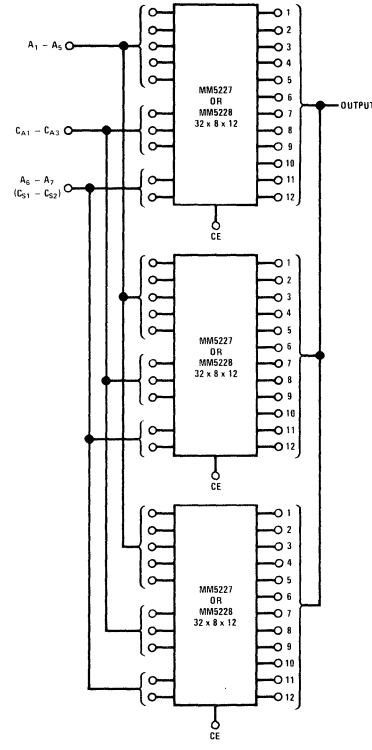


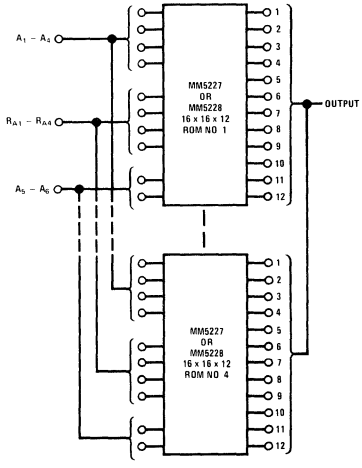
FIGURE 7. Using 32-Character Expansions for 7 x 12 or 8 x 12, 96-Character Generator



(A) Basic ROM



(B) 9 x 7 to 12 x 8 Vertical-Scan, 96-Characters



(C) 12 x 16 Horizontal-Scan, 64-Characters

FIGURE 8. Addressing General-Purpose ROMs as Character Generators

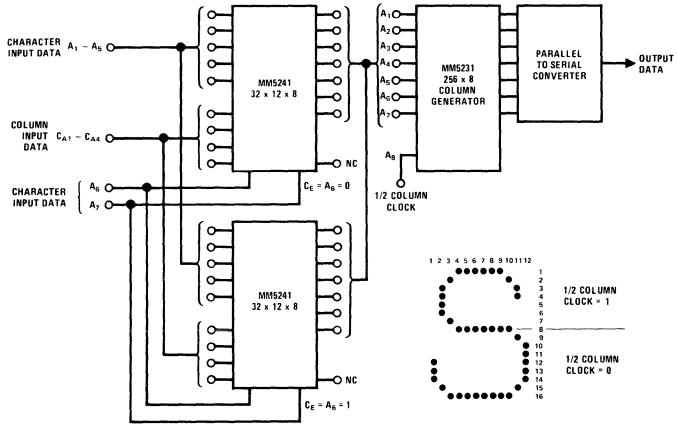
13

INTERMEDIATE CODING

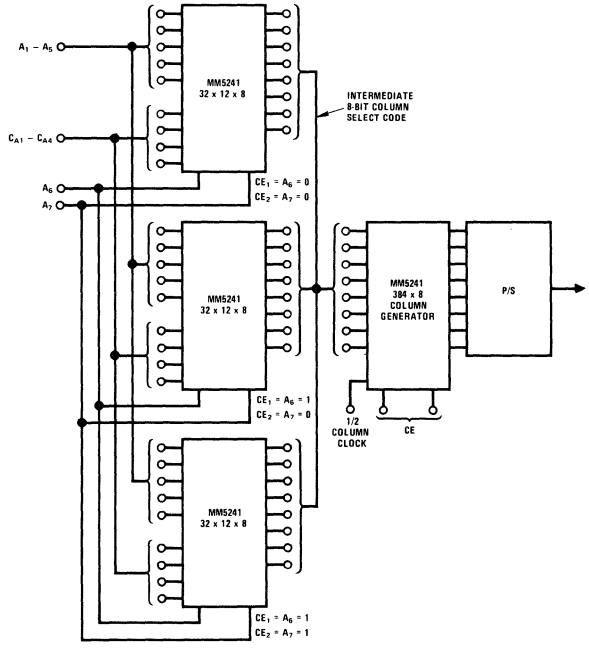
Designs proportioned to the matrix size are not the most efficient at the larger font sizes. It pays to analyze the actual character patterns to determine whether other organizations can be used.

For example, the full-dot columns in such vertical-scan characters as b, B, d, D, H, T, etc., are usually identical. An upper-case font typically contains only 60 unique column patterns at 7 x 5, 110 at 9 x 7, 120 at 11 x 9, and 122 at 16 x 12. Theoretically, they could all have been unique, since there is a possible pattern variation ranging from 128 to 65,536 (2^7 to 2^{16}). UC/LC and horizontal-scan fonts are more variable than upper-case vertical-scan fonts, but they are still far from worst-case.

This analysis led to the organization in Figure 9A. Instead of doubling the 64 x 12 x 8 organization to produce fonts up to 16 x 12, it adds only a 2k



(A) 12 x 16 Vertical Scan (UC)



(B) 16 x 12, 96-Character Generator (UC/LC)

FIGURE 9. Intermediate Coding Designs (MOS ROM Organizations)

ROM. Up to 128 unique column patterns are stored in 8-bit, half-column segments in the MM5231 256 x 8 ROM. These are accessed with 7-bit intermediate codes selected with the input array and a half-column clock at a submultiple of the dot rate. The intermediate codes necessary to form each character are simply listed in character-generator fashion in the input code converters. At 16 x 12, the savings for an upper-case font are 12k − 8k or 4 kilobits—33%.

Since the 8-bit outputs of the MM5241 ROMs actually allow 128 unique columns to be selected, the savings could grow rapidly through several expansion levels even without further rearranging. If more than 128 unique column codes are required the second ROM in the storing can be changed to possibly a 512 x 8 ROM (MM5232) therefore giving 256 unique columns which can be generated for the larger fonts and character group sets (96 characters or 128 characters).

Assume a 16 x 12, 96-character requirement. MM5241 ROMs added as in Figure 9B would provide 192 unique column patterns and the savings would be at least 18k − 12k = 6k.

It might be necessary at the 24 x 12 UC/LC size to use two MM5227 256 x 12 ROMs in parallel, but this would still save 27k − 15k = 12k (or perhaps 36k − 18k in a 128-character application, using four input and two output ROMs). The efficiency grows with font size because the column patterns become more redundant.

At first glance, the organization appears to double the access time because there are two stages to be accessed in sequence. But since there is no feedback, the stages can operate in a ripple mode. Thus, an 8-bit bipolar register can be inserted between the stages to temporarily store each intermediate code. This restores the overall access time to that of the slowest ROM in the series (e.g., less than a microsecond for MOS ROMs) and the character rate is essentially the same as that achieved with conventional ROM techniques.

Alternatively, the output ROM, input ROM, or both may be bipolar to increase the rate. The DM8596 512 x 8 ROM fits most large-font geometries quite well and costs less than sub-assemblies of 1k bipolar ROMs. Again, an intermediate register will maximize the rate.

REPEAT-PATTERN CODING

Some character styles have bold "double dot" or similar patterns that result in a high probability of the same column pattern repeating sequentially in the same character. This characteristic is common in "ticker tape" systems, large-panel and billboard displays, news bulletins broadcast to appear as a running line across a television picture, and so forth. Typical fonts exhibit less than 256 actual changes of column patterns through a 64-character sequence, not the worst-case of 320 at 7 x 5, 640 at 10 x 8, and so forth. Therefore, an organization that holds the column output static until it has to change would be highly efficient.

Figure 10 is a practical design for upper-case fonts with 10 x 10 to 13 x 10 matrices. It saves nearly 40% of ROM capacity. Moreover, the matrix width varies with the character shape as can be seen in the example word LIMB. The characters look more natural and are evenly spaced. Column height is changed by programming the outputs to be used.

Proportional spacing makes this organization an excellent choice for ink-dot spray printers and other "line of type" printing applications, as well as vertical-scan displays.

This technique does not lend itself directly to raster-scan displays since characters are scanned sequentially on one raster line at a time rather than completing a character before starting a new character. To use this technique on raster-scan an intermediate storage memory would be necessary for as a line memories.

PROGRAMMING AND OPERATION

Assume a nominal matrix size of 13 x 10. This takes a 256 x 16 ROM array. Each 16-bit output word contains a 13-dot column pattern, a 2-bit repeat code and, in the last word of a character, an EOC bit (end of character "1" bit). Address location 0000 0000 is reserved for an all-zero spacing column.

The first address of each character is listed in the small input ROM at locations where they will be accessed by the standard code. The intermediate code will then be the starting address and the next column-select codes for each character will be generated sequentially by the logic.

Suppose characters @ through K occupy locations 0000 0001 through 0010 1111 in the main ROM. Then, L's three words occupy locations 0011 0000 through 0011 0010, M starts at 0011 0011, and so forth. L takes three words at the 13 x 10 size since the 2-dot bar pattern can be repeated only four times with a 2-bit repeat code. If the columns were programmed 12 or less dots high, a 3-bit repeat code could be used. L would be generated with two words and single-pattern characters like "dash" with one word. This solution uses only 2 x 16 bits of storage for the character L and compared with its present technique of 10 x 12 = 120 bits.

Now, let's generate LIMB. First, the standard code for L (e.g. 001 100) is converted to 0011 0000, which sets the address counter. The address counter access that word in the main ROM, and the repeat code in the output sets the master counter to time out in two column scanning intervals.

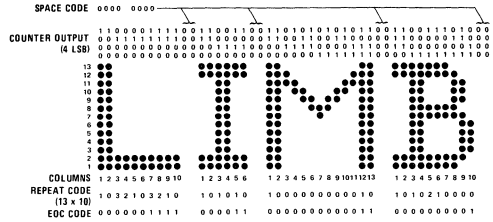
L's first two columns are thus formed. At the master counter's terminal state, the address counter advances to 0011 0001, the master counter is reset

to time out in four column intervals, the address counter advances again, and word 0011 0010 is accessed during four intervals.

This last word includes an EOC bit. When EOC and the time-out state of the master counter coincide, gate 1 clears the address counter. Now address 0000 0000 generates the space pattern in two spacing columns. When the master counter reaches its second state, gate 2 enables the address counter's parallel preset inputs. Finally, the input ROM sets the counter to the starting address for I and the process continues through I, M and B.

Proportional spacing is inherent. So is high-speed since the input ROM is a small bipolar array. The main ROM can be either MOS or bipolar general-purpose ROMs. This organization should also expand efficiently since the repeat probability tends to rise with matrix width.

In printing applications involving more complex characters, the operational advantages might be of more interest than ROM savings. For example, two 64 x 6 x 8 or 512 x 8 ROMs might be used as an UC/LC generator with the ninth address input a direct shift control.



(Repeat Pattern Character Quality and Coding Example)

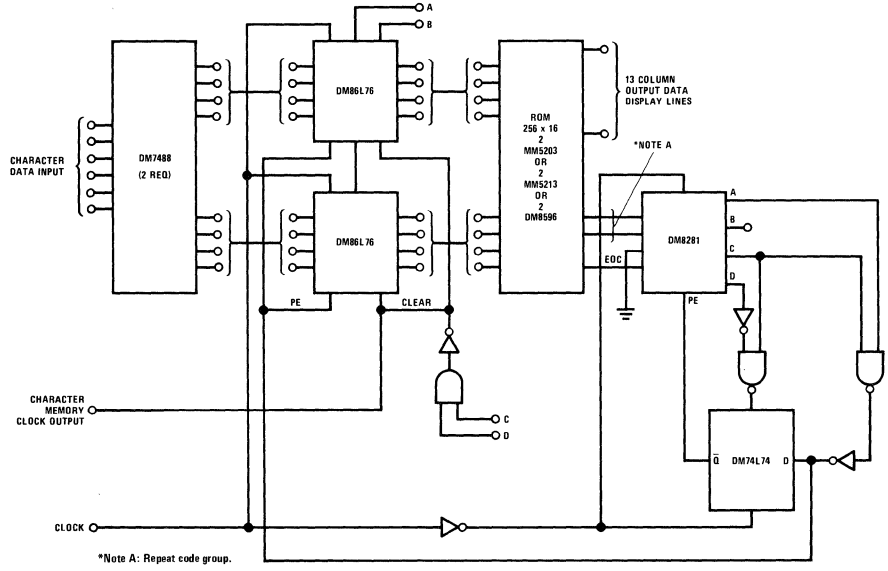


FIGURE 10. Repeat-Pattern Vertical-Scan Generator