Understanding MOS Character Generators

Many of the advances in electronic instrumentation are due to developments in read-out technology. Here's how the character generator forms those alphanumeric symbols on Strip Printers and CRT screens

by DON LANCASTER

THERE ARE LOTS OF GOOD LOW-COST ways to display numbers today, ranging from Light Emitting Diode (LED) displays, liquid-crystal readouts, printing wheels, seven-segment neon and fluorescent displays, and many more. Most of these systems can only handle ten or so numerals, plus possibly a few letters that aren't too attractive. What do you do if you want to display the entire alphabet *plus* numerals *plus* punctuation?

Some of the obvious places this need crops up is in the TV Typewriter (**Radio-Electronics**, September 1973), computer terminal displays, deaf communications aides, time and channel number TV presentations, page and strip printers, oscilloscope scale indications, and anywhere else you want to put down a message and can't afford thousands of dollars worth of mechanical or computer-backed equipment.

Your answer is to use one of a family of integrated circuit *dot matrix* character generators. They use MOS technology, ranging from older p-MOS devices through Silicon Gate and n-channel. Cost ranges from $11 in singles on upward, with questionable surplus units available as low as 50¢ each. What are they and what do they do?

Basically, these devices are really a Read Only Memory, or ROM (**Radio-Electronics**, February 1974). They accept a compact computer code called ASCII (more on this in a bit) and convert it into an open code that represents character shapes. Most often, they have to be combined with fairly elaborate system-timing arranged to get the right part of the right character put in the right place at the right time. All but the oldest devices are directly TTL compatible, while the latest n-channel versions work on the same single +5-volt supply the TTL does, and interfaces without any resistors at all. There's just enough difference between character generators that we're not going to show you specific connections—you have to get this from the individual data sheet on your own. Our interest here is the big picture—seeing what these beasts are and where they can be used, who makes them, and so on.

Why dot matrix?

Several years ago, strong arguments could be made for many different ways of generating alphanumerics, including stroke systems, special CRT symbol generators, projection displays, and the *dot matrix* technique where you put down a bunch of dots to approximate a character shape.

Today, with few exceptions, dot matrix is the only way to go. There are many reasons. Dot-matrix systems take the least in the way of analog circuitry. They interface the simpliest with conventional digital memory and logic. They take the fewest interconnections on a system basis. They use low-cost, widely available, off-the-shelf, standard IC's.

More important, they adopt themselves to the format needed by a recurrent sweep (flyback type) TV style display which is much cheaper, generally, than a true X-Y type video display. The same is true of strip printers and advertising displays where you are going to "roll down" a bunch of characters in one direction.

What's wrong with dot matrix? On the debit side, if you don't use enough dots, the characters may not be attractive enough or will cause eyestrain. And if you are into graphics (artwork, lines, schematics, etc. . . .) you're rather limited in what dot matrix can do for you. But right now, no graphics system is low cost (prices start at $6000), so this really isn't much of a limitation for most alphanumeric applications.

How many dots?

The very first thing we have to decide is how many dots are we going to use in our matrix. Figure 1 shows the letter "R" plotted using matrices of 3×5; 5×5; 5×7; 7×9; and 16×16 dots. Obviously, the more dots we have, the better the character is going to look, and the more the thing is going to cost us in terms of system bandwidth, printhead complexity, interconnections and storage, available characters per line, character writing time, and, of course, dollars.

The 15-dot or "3×5" matrix of Fig. 1-a is normally used only for numbers and an occasional character. Some TV channel and time displays are the only use of this format, and its ugliness limits even that. While no MOS character generator is widely available that generates a 3×5 matrix, you can easily cut your own using a programmable read-only-memory (PROM) or actually use random logic to do the job if you don't mind a bunch of low-cost IC's and diodes. Ignoring spaces, we have to do three things in the horizontal direction. Let's call them "video units", and five things in the vertical direction, which we'll call "column units".

The number of video units often sets the bandwidth we have to use and limits either the maximum print rate in a page or strip printer, or the number of characters per line for a video display of a given bandwidth. The internal ROM storage we would need for 10 numerals and 6 punctuation units (space, colon, etc . . .) would be 16×5×3 or 240 bits. With a little bit of creative logic, you could fold this into a standard 256-bit programmable ROM, although a 512-bit one would be much simpler for system timing.

We also normally need some way to keep the numbers we are using. Its obviously easier to store a 4-bit BCD numeral than a 15-dot numeric character, since we only need 4/15 the bits using the compact code. In the case of a strip printer, the storage only has to last from the time the character is received till the time it is printed. With a video display, we have to get the characters back over and over again, once for each scan.

In fact, on a raster-scan system, you need *two* storage systems, one to keep *all* the characters from field to field, and one to keep a *line* of characters for the seven or nine passes in the fast sweep direction it takes to put down everything one dot at a time. With shift register memories, you use two seperate ones, while with a random access memory, you use the same memory twice, changing the timing.

In the case of the 3×5 display, we usually need only four bits of storage per character since we are only interested in numbers and

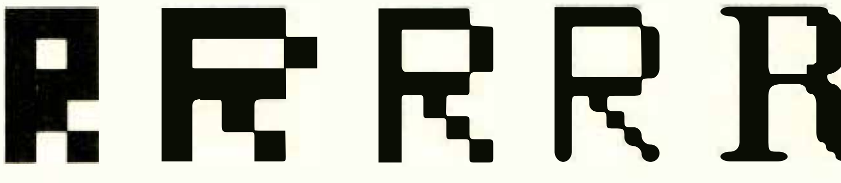
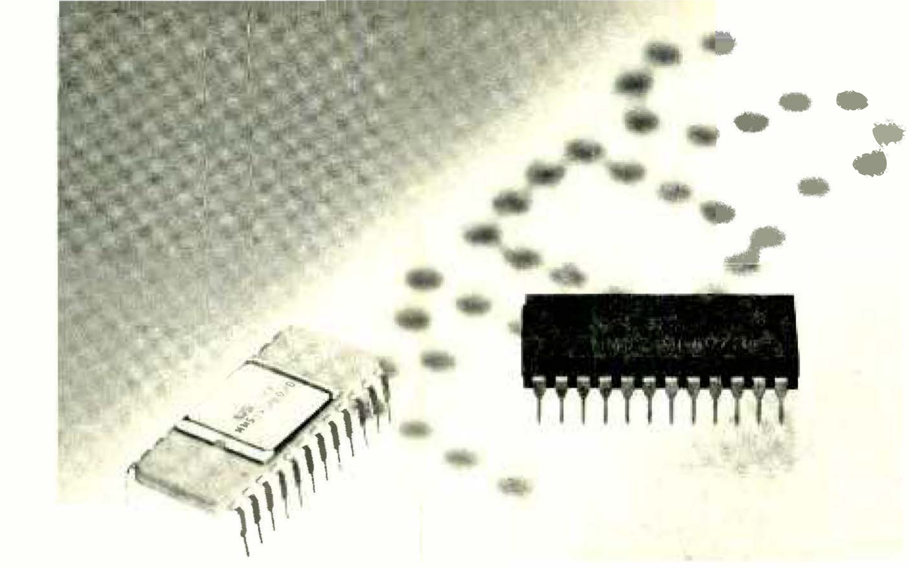
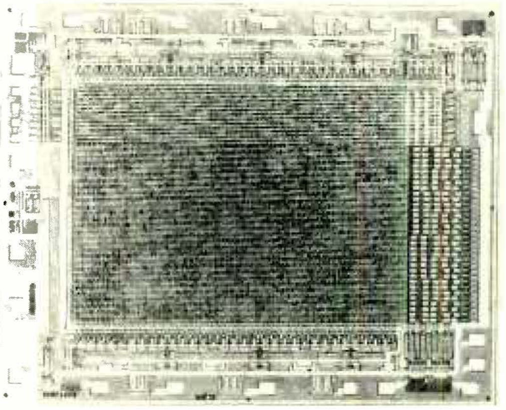


FIG. 1—READABILITY INCREASES as the number of dots or elements in the matrix is increased. Compare the 15-dot 3 × 5 format on the left with the 256-element "R" on the right.





CHARACTER GENERATORS (left) are NS MM5240's. Above is a photomicrograph of the circuit.

a very limited amount of punctuation.

The 5×5 matrix of Fig. 1-b is only slightly better, although you can display the whole upper case alphabet, as well as the numbers. This is used on thermal printers where only 25 printing dots would be needed instead of the 35 of the more standard 5×7 matrix. Five video units are needed horizontally, and five column units are needed vertically. Again, no MOS character generator is widely available for this format, but you can cheat and use a standard 5×7 and have most (but not all!) of the characters turn out OK by dropping the second and sixth horizontal rows.

The standard computer terminal matrix has been the 5×7 one of Fig. 1-c, and lots of standard IC's and interface circuits are available to handle this format. The 5×7 displays to acceptable resolution all the numbers and all the upper case letters and most punctuation. While it *can* display lower case letters and machine commands, it takes special extra IC's and more storage and is not often done. Five video units are needed in the horizontal direction and seven column units are needed vertically. With upper case letters and numbers only, six bits of storage per character are needed externally. The internal storage needed in the character generator is 5×7×64=2240 bits.

The 7×9 matrix of Fig. 1-d presents very attractive characters including lower case, provides for such things as the tail on a lower case "g" and so on. It inherently takes more speed or bandwidth, as seven video units are needed as well as needing more horizontal lines per character. Seven bits of storage per character are usually needed if lower case and machine commands are to be included, while the number of internal character generator bits needed is 7×9×128=8064. One of the penalties you have to pay for the more attractive characters is that the video bandwidth for longer lines of characters becomes far more than what a standard TV can handle. 8 to 12-MHz video is typical, and whatever is displaying the characters must, of course, be able to handle this.

The 16×16 matrix of Fig. 1-e needs incredible bandwidth and storage compared to the other methods, but it allows reasonably faithful reproduction of various printing type fonts, and is useful in larger systems where very attractive characters and long text readability are needed. Phototypesetting and composition systems are typical uses; some of these use even larger matrices still. The needed storage of 16×16×128=32,768 bits *per font* or typeshape is well beyond what you can do with a single character generator IC at the present time, and several custom units have to be used to share the job.

For most small and simple systems, the 5×7 dot matrix is usually the best choice, with the most widely available circuitry, potential compatibility with unmodified television sets, and lowest cost combined with character shapes acceptable for everything but long text messages.

What code?

We've seen that it's best to store our input character information in the most compact form possible, instead of the highly redundant 35 bits we'd need for actual storage of an entended 5×7 character. The storage is usually done in a code called ASCII, short for *American Standard for Computer Information Interchange* (see Fig. 2). This is the only code most character generators accept. In some isolated systems, other codes may be in use; but these are usually converted to ASCII with another ROM before character generation.

The full ASCII code consists of *eight* bits. The eighth bit is sometimes used for error indication, and is stripped and possibly used long before it gets near the character generator. This leaves us with seven bits that represent 128 possible different charac-

FIG. 2 THE ASCII OR AMERICAN STANDARD FOR COMPUTER INFORMATION INTERCHANGE.

This code is used as an input for practically all MOS character generators.

BIT NUMBERS															
								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
b7 ↓	b6 ↓	b5 ↓	b4 ↓	b3 ↓	b2 ↓	b1 ↓	COLUMN → ROW ↓	0	1	2	3	4	5		7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	8	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	11	VT	ESC	+	;	K	[	k	{
			1	1	0	0	12	FF	FS	,	<	L	\	l	¦
			1	1	0	1	13	CR	GS	—	=	M	]	m	}
			1	1	1	0	14	SO	RS	•	>	N	^	n	~
			1	1	1	1	15	SI	US	/	?	O	—	o	DEL

ASCII-4 (Numbers only) uses only column 3
ASCII-6 (Numbers & Upper Case Alphabet Only) Uses Columns 2, 3, 4 & 5.

ters. These are subdivided into four groups of 32 characters each. One group consists of upper case alphabet. A second consists of numbers and often used punctuation. The third group is made up of lower case alphabet, and the final group consists of *machine commands* that do things like line feed, carriage return, clear, start, stop, etc., but rarely appear in print except when viewing the actual machine programming instructions.

We can use as many bits as we need to provide for a given display. For instance, we could use a code we might call ASCII-4, consisting of only column 3 of the code. This gives us all the numbers with four bits and is identical to the normal BCD code. We also get a colon for the time displays free, along with a ? and some other punctuation. We *don't* get a space for blanking, so this has to worked out some other way. Usually this is easy and no problem.

ASCII-6 is used if we are only interested in the upper case alphabet and numbers and often used punctuation, and takes 6 bits and gives us columns 2, 3, 4 and 5 of the code. It is most often used in the 5×7 character generator systems. Finally, we need all seven bits, less the error feature for all upper and lower case alphabets. This is called ASCII-7. Both ASCII-6 and ASCII-7 provide for a blank or space command.

For more information on ASCII, see the Improved ASCII Encoder story, (**Radio-Electronics**, January 1974).

Some character generators

Our character generator IC has to convert a six or seven bit ASCII character command into 35 or 63 dots on a screen, group of light sources or a piece of paper. Figure 3 shows four possible character generators for the 5×7 format.

In Fig. 3-a, we have an IC with some power supply and ground connections, six input lines and 35 output lines. Whenever we put the ASCII command for a "R" on the input, we get a "R" in dot matrix form at the output. This is called a *read only memory* or table lookup code converter and is simply a ROM that has been factory programmed to generate characters. To change characters, change the input code. For a blank, input code 10-0000, and so on.

While this IC would be ideal for scoreboards, you can't buy it because of all the pins. Besides, do we really want all the dots at once, except possibly for scoreboard use? And can we cut down the number of output connections somehow?

For TV raster type use, we only need *one* dot at a time and should be able to get by with only *one* output lead. Now we need two types of inputs. We need six bits worth of code to select the character, and another group of inputs to tell us *which* dot of that character to output. Changing the later group of inputs, called *timing inputs*, generates a sequence of dots that let us produce the whole character one dot at a time. For instance, we could use three lines for a "what column is it?" input command, and three lines for a "what row is it?" input.

This is shown in Fig. 3-b. Now to generate a character, you input the character code and apply timing signals that get the dots out in the proper sequence. In the case of the TV type raster, it rapidly scans in the horizontal direction, and slowly goes vertically, so we put down a burst of dots corresponding to the *top* line of a character, and then go on to the *next* character, and so on, changing the ASCII code as needed. On the next line, when the character we wanted came up again, we'd work on the *second* lines worth of dots from our output and so on.

This is a nice IC, but once again, you can't buy it. This time you run into switching speed and settling time problems, along with an internal IC arrangement that would take far too much silicon to be practical. While this is much closer to what we want than Fig. 3-a is, some compromise is needed to make a practical low-cost IC. What can we do?

Real-world character generators appear in Figs. 3-c and 3-d. One is called COLUMN output character generator. One is called a ROW output character generator. They are NOT interchangeable under any circumstances. For any system, you have to pick either a ROW or a COLUMN device to suit your particular needs.

The ROW output character generator (Fig. 3-c) has five output lines corresponding to a horizontal row of dots. It has six input lines for the ASCII "What character do we want?" input, and three timing input lines for the "What vertical position of dots are we to generate?" command. Thus, at any instant, you get out *five* dots and "undots" corresponding to a *horizontal* ROW of a character.

Figure 4 shows how you can convert the

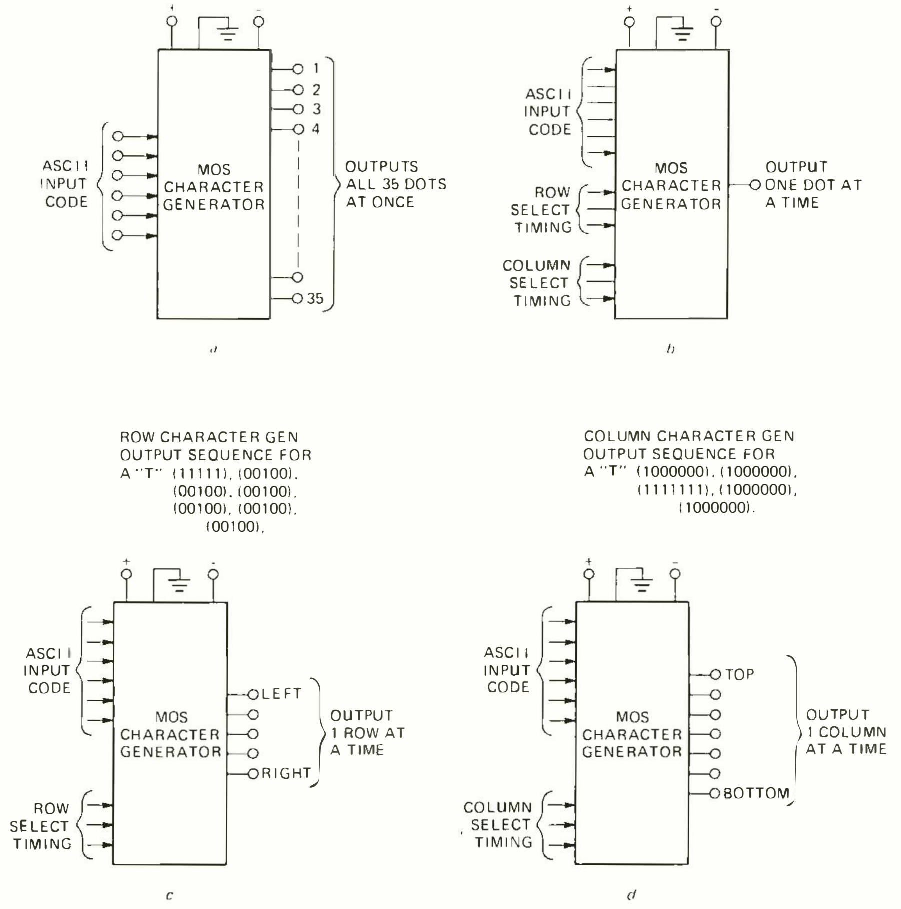
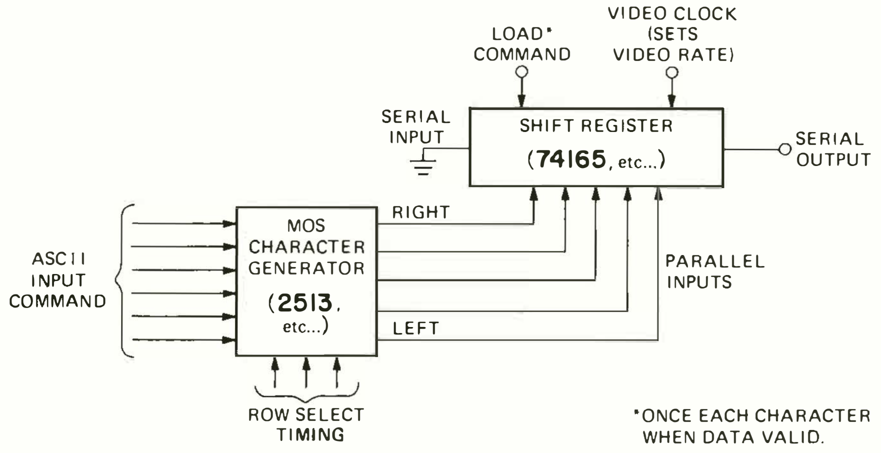


FIG. 3—FOUR POSSIBLE CHARACTER GENERATORS. The top two are not commercially feasible. The row-output device (c) and column-output generator (d) are readily available and easy to use.



LOAD* COMMAND
VIDEO CLOCK (SETS VIDEO RATE)
SERIAL INPUT
SHIFT REGISTER (74165, etc...)
SERIAL OUTPUT
RIGHT
PARALLEL INPUTS
LEFT
ASCII INPUT COMMAND
MOS CHARACTER GENERATOR (2513, etc...)
ROW SELECT TIMING
*ONCE EACH CHARACTER WHEN DATA VALID.

FIG. 4—ADD A SHIFT REGISTER to a row-output character generator for high-speed compatible video. This type of device is used in the R-E TV Typewriter.

five outputs to serial horizontal video with an external shift register. On a *load* command, the five outputs are loaded into the shift register. The video clock then marches things out as serial video. By picking our load command timing just right, we sample the character generator after all the answers have settled. This both increases our potential video speed by a factor of five and eliminates glitches and invalid answers. Note that a data selector cannot normally be used in place of the shift register because of these settling and glitching problems.

A ROW output character generator is ideal for things like TV Typewriters and video terminal displays using television sets or TV like devices that sweep rapidly in the horizontal direction and slowly in the vertical direction.

On the other hand, the COLUMN output character generator of Fig. 3-d has *seven* output lines, corresponding to a vertical column of dots. This time, we have our 6-bit ASCII "What character do you want?" inputs, and three "What horizontal row *position* do you want?" inputs. By changing the timing signals on the column inputs, we get five sequential dot groupings of seven dots each.

The COLUMN output character generator is well suited to a strip printer, where the paper is moving past some sort of print head (pins, thermal, hammer and ribbon, electrostatic, etc. . . .) Seven dots or undots are put down and the paper is shifted 1/5 of a character horizontally and seven more dots or undots are put down, continuing the process five times to generate the final character. COLUMN output character generators are also useful for advertising sequential displays, scoreboards, and anywhere else you want the entire height of the character to appear at once, but want to or at least willing to space things out horizontally. They are also sometimes used on specialized video systems with reversed scans; they can **not** be used on ordinary TV systems.

Remember, for normal TV scans, use a ROW output character generator, for strip printers, other printing devices, film annotation, advertising displays, etc., use a COLUMN output character generator. Do not get the two mixed up.

What's available?

Figure 5 lists some currently popular MOS character generators, while Fig. 6 gives sources of manufacturers and data sheets. The Signetics 2513 (ROW output —TV) and 2516 (COLUMN output—strip printer) are often a good choice in simple low-cost circuits, although their availability has been rather tight recently.

What good are they?

Let's take a quick look, in block diagram form, at some popular character generator applications.

In Fig. 7, we have a single character alphanumeric display. This is handy for keyboard verification, deaf communications, teaching touch typing, remote message signalling, etc. It's intended for use where you want to transmit only a single character at a time. Input signals you need are the ASCII character on six lines, a blanking signal when you want the character to light, and power supplies. You use a 35-dot LED matrix display such as the

FIG. 5 SOME TYPICAL MOS CHARACTER GENERATORS

Number	Mfg.*	Matrix	Type	Power	Package	Notes
S8866	AMI	5×7	Row	+5, −12	28	
S8564	AMI	5×7	Column	+5, −12	28	
EA3501	EA	5×7	Row	+10, −12	24	
EA3513	EA	5×7	Row	+10, −12	24	lower case only
EA3701	EA	5×7	Column	+10, −12	24	
3257	FAIR	5×7	Column	+5, −12	24	with counter
3258	FAIR	5×7	Row	+5, −12	16	with counter
3260	FAIR	7×9	Column	+5, −12	24	with lower case
1-2240	GI	5×7	Row	+12, −12	24	needs clocks
2561	HARS	5×7	Row	+5, −10	24	needs clocks
MF7107	MSI	7×9	Column	+5, −9	22	
MK2002	MOSK	5×7	Column	+14, −14	24	
MK2302	MOSK	5×7	Row	+5, −12	24	dual output; counter
MK2408	MOSK	5×7	Row	+5, −1w	28	dual output
MCM6571	MOT	7×9	Row	+5	24	with lower case
4240AA	NAT	5×7	Row	+12, −12	24	
2513	SIG	5×7	Row	+5, −5, −12	24	
2516	SIG	5×7	Column	+5, −5, −12	24	
2526	SIG	7×9	Row	+5, −12	24	type CM3490
2526	SIG	7×9	Column	+5, −12	24	type CM3400
TMS2400	TI	5×7	Row	+12, −12	28	
TMS4100	TI	5×7	Column	+12, −12	28	

*See Fig. 6

FIG. 6 MANUFACTURERS OF MOS CHARACTER GENERATORS

AMERICAN MICROSYSTEMS (AMI)
3800 Homestead Road
Santa Clara, California, 95051

ELECTRONIC ARRAYS (EA)
501 Ellis Street
Mountain View, Calif, 94040

FAIRCHILD SEMICONDUCTOR (FAIR)
313 Fairchild Drive
Mountain View, Calif, 94040

GENERAL INSTRUMENTS (GI)
600 West John Street
Hicksville, NY, 11802

HARRIS SEMICONDUCTOR (HARS)
Box 883
Melbourne, Florida, 32901

MICROSYSTEMS INTERNATIONAL (MSI)
Box 3529 Station C
Ottawa, Canada, K1Y4J1

MOSTEK (MOSK)
1215 West Crosby Road
Carrollton, TX, 75006

MOTOROLA SIMICONDUCTOR (MOT)
Box 20912
Phoenix, Arizona, 85036

NATIONAL SEMICONDUCTOR (NAT)
2900 Semiconductor Drive
Santa Clara, California, 95051

SIGNETICS (SIG)
811 E. Arques Avenue
Sunnyvale, Calif., 94086

TEXAS INSTRUMENTS (TI)
Box 5012
Dallas, Texas, 75222

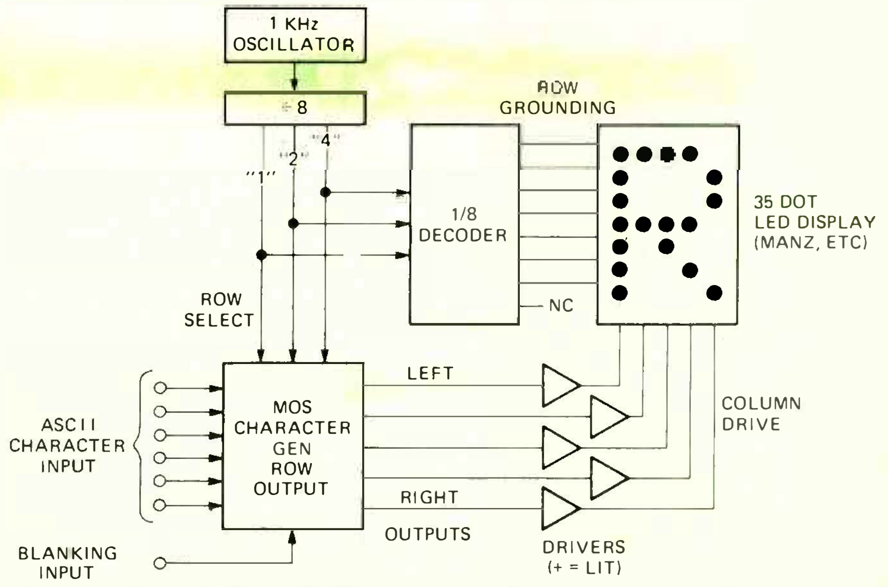


FIG. 7—SINGLE-CHARACTER ALPHANUMERIC DISPLAY. You need one for each unit in the message.

Monsanto MAN-2 as an output. An internal oscillator scans everything much faster than the eye can follow. This is followed by a divide-by-eight counter that sequentially tells the character generator which vertical row of dots to output. At the same time, the counter tells a one-of-eight decoder which bank of five LED's to ground. As the commands to the character generator advance down the character, the respective rows of LED's are also grounded so that at any instant the character generator is putting out the right code to the proper group of five LED's. If you ran things too slowly, the character would start at the top and work its way to the bottom, repeating over and over again. By speeding things up faster than the eye can follow, you see everything as normal brightness and continuously lit. Since any one dot only gets hit one eighth of the time, you run things at eight times normal brightness to make everything come out even.

Figure 8 shows a TV time display that can also be used to put the channel number or any other short (8 unit) message on a TV or an oscilloscope. Complete construction details will appear in the TV Revideo Unit (**Radio-Electronics**, later this year). Basically, we have the character generator of Fig. 4 (ROW output driving a shift register serial video converter), along with some added system timing. A four-pole (for numbers) or a six-pole (for everything) data selector of eight positions picks the character you want to display, while system timing selects the proper line number.

In operation, the vertical sync pulse of the TV starts off a delay for vertical position which in turn picks the next whole horizontal line and starts off a horizontal position delay. A video clock of 64 counts is counted out and then shut down, allowing five dots and three spaces per each of the eight character positions. The process repeats for the next seven lines and then shuts down for the rest of the active scan. Adjusting the video clock frequency handles horizontal size, while vertical size can be adjusted in increments, by using one, two or three interlaced horizontal line pairs per character generator address changes.

Figure 9 shows a video display similar to the TV Typewriter (**Radio-Electronics**, September 1974). Here a MOS memory (either RAM's or shift registers) stores the total number of characters to be put on the screen, often 512 or 1024. Six bits per character are needed for this *buffer* storage. Note that we receive each character only *once*, but that we have to get it back seven times on seven sequential lines of each sweep of the TV's field. System timing advances the characters and the "What line is it?" commands in the proper sequence put down one part of a row of characters one scan and then put down the part immediately below on the next scan, and so on.

Note that you can't arbitrarily increase the number of characters per line without something giving somewhere, particularly if you are using an unmodified television set. The two things that get to you are the settling and access time of the character generator and the bandwidth of the TV in use. Let's look at some numbers.

Overscan on many stock TV sets is extreme, so it's not unreasonable to allow *one-half* of our active scan for retrace and positioning. Less than this, and you will almost certainly get into corner problems at the beginning and end of the message. Suppose we use normal TV scanning standards with an interlaced horizontal frequency of 15,750 Hz. This gives us 63.4 μs of scan time. Cut this in half for an active scan time of 31.7 μs. To ease the math, call this 32 μs. Suppose we have 32 characters per line. It takes us 1 μs per character, so the 500 μs or so access time of the 2513 generator leaves us with lots of daylight. No problem here. Now, what about the video bandwidth?

We have five active dots, but we can't put the characters beside each other. Suppose we save two dots for spacing. This means there are seven dots in 1 μs, or a video clocking rate of 7 MHz. With a few crude approximations, a data rate of 7 MHz equals a video rate of one half this or 3.5 MHz.

Now, the video frequency response of a television set is 4.5 MHz for black and white and 6 MHz for color, so this is no problem, right! WRONG. Those are the *i.f bandwidths*! A black and white set must reject the 4.5-MHz sound subcarrier in its video, so the best we can hope for is about a 3.5-MHz bandwidth. Color *video* bandwidth is even LESS, for it has to reject a 3.58-MHz color subcarrier, leaving us only with a 2.5 MHz or so bandwidth. Thus, at 32 characters per line, we can brightly display a black and white image but can only do a color image at somewhat reduced contrast.

But, the computer people use 72 or 80 characters per line. How do they do it? Very simply, by modifying the TV set for *direct* video entry and extreme bandwidth. An 80-character system takes a bandwidth of 80/32nds of 3.5 MHz or 8.75 MHz. This is why there are very few color displays of long line width. Note that at 72 or 80 characters per second, you are also at the limit of the character generator access time and have to be extremely careful with system timing.

What all these numbers are trying to say is that it's real hard to put more than 32 or 40 characters per line on an unmodified TV, particularly a color one. More than this, and you have to do some modifying. **R-E**

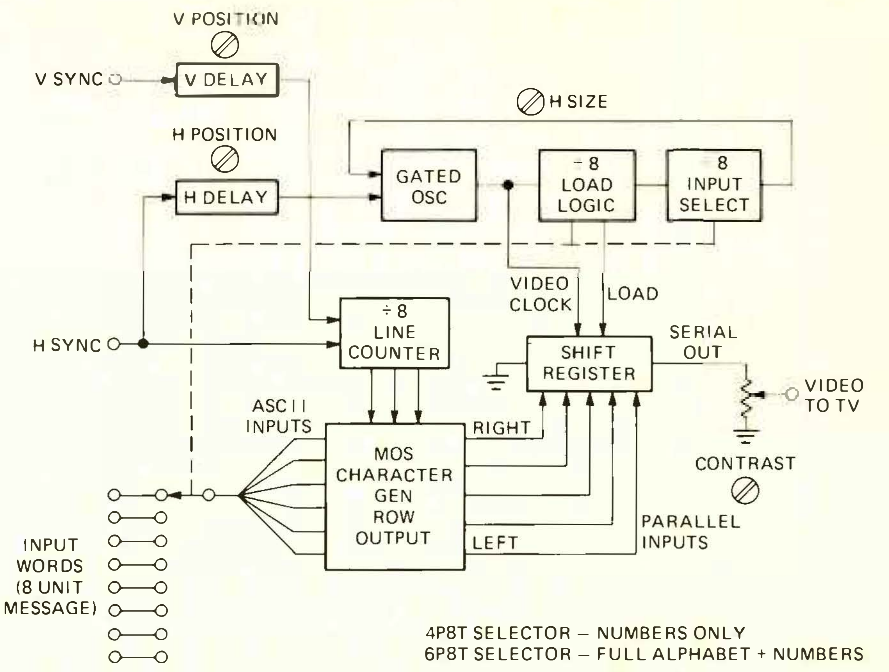
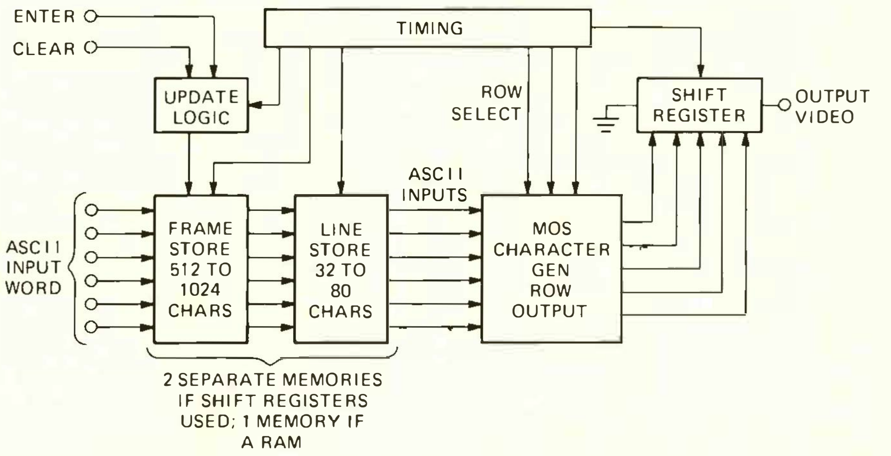


FIG. 8—TV TIME OR CHANNEL DISPLAY uses a character generator, shift register and system timing. A construction article on a device of this type will appear later this year.



ENTER
CLEAR
TIMING
UPDATE LOGIC
ROW SELECT
SHIFT REGISTER
OUTPUT VIDEO
ASCII INPUTS
ASCII INPUT WORD
FRAME STORE 512 TO 1024 CHARS
LINE STORE 32 TO 80 CHARS
MOS CHARACTER GEN ROW OUTPUT
2 SEPARATE MEMORIES IF SHIFT REGISTERS USED; 1 MEMORY IF A RAM

FIG. 9—VIDEO DISPLAY as used in a TV typewriter or a computer terminal. Details on a do-it-yourself video display with keyboard input appeared in September 1973.