## Read Only Memory Technology

Read Only Memories are a useful element of the hardware of microcomputer systems. This month, Don Lancaster provides information on Read Only Memories – a tutorial article taken from Chapter 3 of his forthcoming book, TV Typewriter Cookbook, to be published by Howard W. Sams, Indianapolis, Indiana.

Low cost and compact memory components are the key to simple and reasonable ΤV typewriter and microcomputer systems. Today there are many ICs available that will cram thousands or more bits of storage in a single package at costs of a fraction of a cent per bit. The problem is to pick the memory components that are the cheapest, the easiest to use, and the ones with the fewest unpleasant surprises.

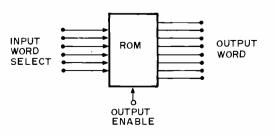
What does memory do for us? Well, it remembers. If it remembers extremely well it can be used as a fixed logic element or to store an often-used software routine.

An important example of this is the character generator memory that converts our ASCII code into a group of dot patterns suitable for video use. The same type of memory might be used to change the keyboard switch closures and shift combinations into selected ASCI1 codes. We also use permanent memory for code conversions such as going from ASCII to SELECTRIC, and permanent memory is usually used to store the program and control commands for an external

microprocessor or minicomputer system.

This particular type of memory is called a read only memory, or ROM. Once taught, it does the same thing forever, even if supply power is repeatedly applied and removed. There are several ways this memory type can be taught. One is to use a factory programmed mask at the time of manufacture. Another is to use some field programming technique such as melting internal silicon or metallic fuses. This is handy whenever you are doing a special, low volume code or program, or whenever you

Fig. 1. Read Only Memories (ROMs) are non-volatile and used typically for programming of "firm" microcomputer software, for fixed lookup tables, or for code conversions.



by Don Lancaster Synergetics aren't sure what you are doing is really what you want to end up with. Such programmable read only memories (PROMs) allow customizing for special purposes. Some of the more expensive programmable read only memories can be bulkerased through exposure to ultraviolet light or X-rays; these erasable read only memories (EROMs) can be re-programmed in the event of an error.

Most read only memories can also be called code converters or table lookup devices, and are usually organized as shown in Fig. 1. Each ROM is a fixed logic block that has several inputs and several outputs. For each and every possible input address combination of ones and zeros, some unique combination of output ones and zeros will result. There doesn't have to be any rational relationship between these two code words. Either you or the manufacturer decides what these combinations are going to be at the time the ROM is programmed. A ROM is completely universal; it's inherently set up to provide all possible combinations of input/output word arrangements. When you program your ROM, you limit these all possible combinations to a single specific word exchange that you want.

One popular smaller ROM arrangement is called a 32 x 8 ROM. This means you can program 32 eight bit words. Since 32 words can be represented with binary combinations on five lines, this particular ROM has five input lines and eight output lines. This type of ROM has 256 possible memory locations. At each and every location, we have the option of permanently or semi-permanently placing a one or a zero. This leaves us with  $2^{256}$  possible programs we can teach our ROM, an incredibly large number. The only thing that changes with a particular program is where you put the ones and zeros. All the rest of the circuit stays the same.

ROMs work by decoding each and every possible input state into a one-of-n code and then recombining certain selected combinations of decodings into output words using OR circuits. Which combinations you use picks what the output word is going to be.

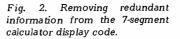
There are several ways to program a ROM. It can be done at the factory where metal jumpers are provided or omitted to the tune of holes or no holes in a mask. Factory programming is cheap but must be done with a high volume product that has one internal code that lots of users can agree to use. Dot matrix character generators, some keyboard encoders, trig lookup tables for calculators and so on are typical factory programmed ROMs. Field programmable ROMs are programmed by the user, or by a distributor or someone else who is set up to do programming. A programmable ROM arrives from the factory either all ones or all zeros, depending on the type. You then do something to change the bits you want to suit your code. In one type of ROM, fusable links are melted. These links are made of a metal such as nichrome, or of a semiconductor such as silicon. These techniques are most commonly used on bipolar or TTL-like ROMs. These ROMs are usually fast and relatively small. Another type of programming injects large voltage pulses that avalanche charge storage areas, electret style. This type of programming is used on MOS read only memories. They are usually slower but have more bits available per package.

Some premium ROMs are reprogrammable. In one type,

you take off an opaque lid and bulk erase the chip with strong ultraviolet light. A second type can have ones or zeros selectively and more or less permanently written into it. Reprogrammable ROMs cost more but can be used over. More important, if you make a programming mistake, you can reuse the same chip, correcting the error later on.

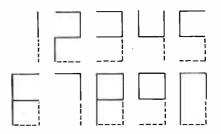
We can also classify ROMs as general purpose devices and dedicated ones. A general purpose ROM can be made into whatever you like and used for just about anything, such as for code conversion, or to store programs for a microcomputer or microprocessor. Dedicated ROMs are usually part of a larger integrated circuit and have very specific uses. Typical examples are in a dot matrix character generator, the word converter in a premium keyboard encoder, and the program storage in many calculator integrated circuits.

Let's see how ROMs work and what devices are available by looking at two important



(a) Segment callouts.

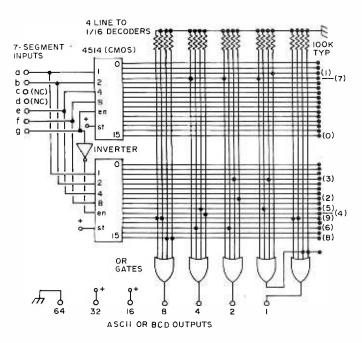
(b) Numerals with segments "c" and "d" missing are still identifiable.



"If you do a PROM design and end up with a ridiculous number of bits, you can almost always go through a rethinking and reduction process that will minimize things a bunch." "Even with bulk erasable PROMs, a mistake on bit #1874 of a 2k PROM can be enough to ruin your whole day."

Fig. 3. ROM-organized logic to convert 7-segment calculator code to BCD or ASCII code.

Fig. 4. Single IC 7 segment code converter uses 32 x 8 read only memory.



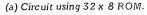
uses for ROMs – a seven bar to ASCII converter that can be used to tie a calculator into a TV typewriter or microcomputer; and an ASCII to SELECTRIC code converter that lets us drive a Selectric typewriter.

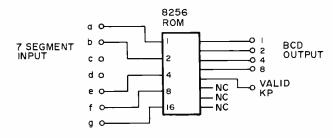
## Seven Segment Converter

Many calculator chips output only a seven segment code that is not directly c o m p a t i b l e w i t h microcomputer software unless it is changed to a Binary Coded Decimal (BCD) or ASCII coding. While several conversion ICs exist, at this writing, they are neither cheap nor readily available. Can we do the job with a read only memory?

At first glance, it would seem that we'd need a ROM with seven inputs or  $2^7 = 128$ words minimum. But, with practically every ROM application, a little bit of rethinking can usually drastically cut down the size and cost of the ROM we'll need. For instance, Fig. 2 shows how we can simply ignore the bottom and bottom right segments ("c" and "d") of the segment code and still have ten distinct and identifiable characters. This cuts us down to 32 words, getting us by with five inputs, and one fourth the size of the ROM we started with.

Fig. 3 shows us how we might build our own "pseudo-ROM" using some CMOS gates and decoders. While you would rarely want to go this route, it's useful to look at since it shows us how the real ROMs work inside. You might occasionally use a circuit like this to verify programs and truth tables





(b) Programming.

	INPUT					NUMERAL			OUTPUT				
g			е	b	а			kp		4	2	1	
	) (	0 0 0	0 0 0 0	0 0 1 1	0 1 0 1		 1 7	0 0 1 1	0 0 0 0	0 0 0 1	0 0 0 1	0 0 1 1	
		0 0 0 0	1 1 1 1	0 0 1 1	0 1 0 1		  	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	
	) <sup>,</sup>	1 1 1 1	0 0 0 0	0 0 1 1	0 1 0 1		- - -	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	
	) ·	1 1 1 1	1 1 1 1	0 0 1 1	0 1 0 1		- - 0	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	
1 1 1 1	(	0 0 0	0 0 0 0	0 0 1	0 1 0 1		  3	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 1	
1 1 1 1	(	D D D D	1 1 1 1	0 0 1 1	0 1 0 1		- - 2	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 1	0 0 0 0	
1 1 1 1		1 1 1 1	0 0 0 0	0 0 1 1	0 1 0 1		- 5 4 9	0 1 1 1	0 0 0 1	0 1 1 0	0 0 0 0	0 1 0 1	
1 1 1 1	•	1 1 1 1	1 1 1 1	0 0 1 1	0 1 0 1		 6  8	0 1 0 1	0 0 0 1	0 1 0 0	0 1 0 0	0 0 0 0	

since it is easy to change.

Our five input lines (ignoring the redundant c and d segments) are decoded to a one-high-out-of-32 code. For each and every possible input combination, one and only one of the horizontal rails goes to a "1"; the rest stay low. The OR gates on the output re-encode this into a 1-2-4-8 Binary Coded Decimal code. We decide what our OR gates do by where we put the dot connections between the horizontal and vertical rails. To get from BCD to ASCII, we can simply tack a hard wired 011 in front of the BCD word.

While we could dream up a possibly simpler "logical minimum" circuit to do the same job, this particular circuit has a unique advantage - if our OR gates are "wide" enough, it will convert ANY five bit input word into ANY four bit output word, with no change in hardware. All that changes is the positions of the dots. This is the beauty of the read only memory - only a single integrated circuit is needed to do an incredible variety of specialized jobs, depending only on how you program it.

Fig. 4 shows how we take a stock 32 x 8, 256 bit Programmable Read Only Memory, or PROM, and do the whole job with one integrated circuit. Since we have extra outputs left over, we can use one for a "valid keypressed" output that can tell the difference between a zero code and no key pressed. The remaining three outputs can be used for detecting a "9" output or for other housekeeping that's handy when you demultiplex the scanning digit outputs of the calculator IC. To program the ROM, the truth table of Fig. 4(b) is entered into the integrated circuit, selectively putting ones and zeros as necded.

MANUFACTURER	PART	BITS	ORG.	TYPE	ERASABLE
AMERICAN MICRO DEVICES	27508, 09	256	32×8	BIPOLAR	NO
	27510.11	1024	256×4	BIPOLAR	NO
	1702	2048	256x8	MOS	
	1702	2046	200X0	MOS	YES
HARRIS SEMICONDUCTOR	1256	256	256x1	BIPOLAR	NO
	8256	256	32×8	BIPOLAR	NO
	0512	512	64×8	BIPOLAR	NO
	1024	1024	256x4	BIPOLAR	NO
	2048	2048	512x4	BIPOLAR	NO
		2010	01241	UN OLAN	NO
AIRCHILD	93421	256	32×8	BIPOLAR	NO
	93416,26	1024	256x4	BIPOLAR	NO
	93436,46	2048	512×4	BIPOLAR	ND
NTEL	3601				
WIEL .		1024	256×4	BIPOLAR	NO
	3602.22	2048	512×4	BIPOLAR	NO
	1702	2048	256×8	MOS	YES
	3604,24	4096	512×8	BIPOLAR	NO
	2704	4096	512x8	MOS	YES
	2708	8192	1024×B	MOS	YES
NTERSIL	5600.10	256	32×8	BIPOLAR	NO
	5603,23	1024	256 x4	BIPOLAR	NO
	5604,24	204B	256×8	BIPOLAR	NO
IONOLITHIC MEMORIES	6330.31	256	32×B	BIPOLAR	NO
nerroen new office	6300,01	1024			
	6305.06	2048	256×4	BIPOLAR	NO
	6340.41		512×4	BIPOLAR	NO
	6340,4 1	4096	512x8	BIPOLAR ,	NO
IATIONAL	8573.74	1024	256×4	BIPOLAB	NO
	5202,03	204B	256×8	MOS	YES
	5204	4096	512x8	MOS	YES
ITRON	7002	1024	512x2	MOS	YES
	7002	1024	1024x1	MOS	YES
	7002	1024	102421	1405	TES
IGNETICS	8223	256	32×8	BIPOLAR	NO
	82126,29	1024	256×4	BIPOLAR	NO
	82130,31	2048	256x8	BIPOLAR	NO
	82115	4096	512x8	BIPOLAR	NO
EXAS INSTRUMENTS	74188	256	32x8	BIPOLAB	NO
	74186 74287	512 1024	64×8 256×4	BIPOLAR	NO NO

Fig. 5. Some commercially available programmable ROMs.

## Working With PROMs

Fig. 5 is a listing of some currently available PROMs. Where two numbers are shown, one is usually an open collector output, the other tri-state. At this writing, PROMs cost from \$5 upwards, with surplus versions (unused) starting at \$3. Bipolar PROMs are based on a TTL technology, usually work off a single +5 volt supply, and are rather fast, typically 50 to 70 nanoseconds access time. MOS PROMs often take two power supplies (+5 and -12usually) and are slower, typically having a one microsecond access time. MOS PROMs are often cheaper per bit and many MOS types are bulk erasable by exposure to strong ultraviolet light. A few ultra-fast ECL PROMs also exist, but are reserved for special uses and are expensive.

Two good choices for home brew computing are the  $32 \times 8$  bipolar PROM such as the Intersil 5600 or the Signetics 8223; and the 256 x 8 erasable MOS PROM, including the Intel 1702 and its second sources.

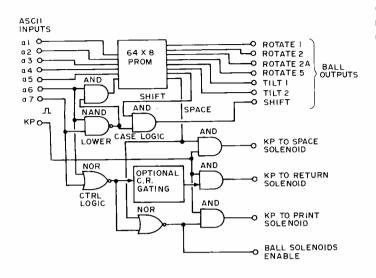
While you can program your own PROM with nothing but a power supply and a meter, the "zero defects" nature of this work and its "up the wall" aspects turn the job into quite a hassle. Even with bulk erasable PROMs, a mistake on bit #1874 of a 2k PROM can be enough to ruin your whole day. Instead of this, you can buy programming services at very low cost from many electronic distributors, as well as from surplus and computer hobby supply houses. Programming machines that simplify the job a bunch are available for several hundred dollars. [Once you have a microcomputer system up and running, it is quite possible to construct an ROM programming peripheral for the purpose of permanently burning in your software. Local computer clubs might consider building the ROM burner peripheral and related software as an attraction of membership . . . cth

A quarter's worth of gating can cut the size of a ROM in half.

When you design your own PROM circuit, be absolutely sure your truth table is correct before you order any programming. The program service will only guarantee that what you sent in is what you get back, and nothing more. They have no way of second guessing what you really wanted.

If you do a PROM design and end up with a ridiculous number of bits, you can almost always go through a rethinking and reduction process that will minimize things a bunch. Leaving off the two redundant segments of a seven bar code is one obvious example. Other possibilities are to put simple logic outside the PROM, for

## Fig. 6. ASCII to SELECTRIC interface using PROM.



an up down counter driving the address inputs. Another possibility is to let the PROM's output set the next input address to the PROM, or at least influence it. Some outside latch or storage is needed to prevent an unchecked wild race, but this is easily added.

This particular technique is called microprogramming, and is, of course, the key to calculator and microcomputer operation. Even without a CPU, a PROM plus additional logic can be used as a programmable controller. Loops and branches are easily added by external gating and using extra PROM inputs. Several additional details on PROM and ROM design appeared in the February, 1974, Radio Electronics.

A ROM or PROM can be used to change ASCII coded

signals into SELECTRIC outputs suitable for the hard copy techniques output if you have a converted Selectric typewriter. While a few ICs are commercially available to do this job (such as the Fairchild 3512 and the National 4230), at this writing, it's much cheaper and simpler to program your own PROM. You can also add custom features of your own, such as converting the ASCII

"E" and so on.

Fig. 6 shows us a circuit that only needs 512 bits worth of ROM and a few gates to do the one-way conversion for us. The PROM basically works with the ASCII 6 bit code of upper case alphabets, numbers and punctuation. It converts these ASCII commands into the seven Selectric shift, rotate

often a gate or two can significantly reduce the PROM size, Bypassing control commands around a PROM is one way to do this. Sometimes symmetry and mirror techniques can be used, particularly when working with trig waveforms, music waveshapes, and other data tables that have some sort of symmetry. In PROM microcomputer programs, sneaky programming tricks can often drastically cut the number of steps needed; extensive use of subroutines is one route to this end.

In code converter and table lookup applications you usually address your PROM in a random fashion and you have no way of knowing what is going to be needed next. There are other ways to address ROMs that open up other types of applications. For instance, if you sequentially clock the PROM, changing the address one bit at a time at a constant rate, you can generate an output sinewave or a musical timbre waveform. The clocking rate will select the output frequency, and you can get a symmetrical output by using Fig. 7. Listing of ASCII inputs (octal) and Selectric outputs for the PROMs in Fig. 6.

ASCII		SELECTRIC	ASCI		SELECTRIC
INPUT	CHARACTER	OUTPUT	INPUT	CHARACTE	R OUTPUT
100	0	166	040	SPACE	200
101	А	134	041	1	177
102	В	140	042		125
103	с	154	043	#	176
104	D	155	044	\$	171
105	E	145	045	%	135
106	F	116	046	&	175
107	G	117	047	•	025
•					
110	н	141	050	(	160
111	1	124	051	)	161
112	J	107	052	•	174
113	к	144	053	+	106
114	L	151	054	,	014
115	м	137	055	-	000
116	N	146	056		026
117	0	131	057	/	011
120	Р	105	060	0	061
121	Q	104	061	1	077
122	R	135	062	2	066
123	S	121	063	3	076
124	Т	147	064	4	071
125	U	156	065	5	065
126	V	136	066	6	064
127	w	120	067	7	075
130	x	157	070	8	074
131	Y	101	071	9	060
132	z	167	072	:	115
133	Į	111	073	; _	015
134	N N	164	074	<	027
135 136	1	111	075	=	006
130		145	076	>	127
137	-	100	077	?	111
	ASCII		ELECTRIC		READING
	654 321	SP SH	T2 T1 R5 R2	2A R2 R1	BITS OCTAL
		U Q1	ل	110	BINARY
	<b>* *</b> -	۲	¥	•	
	2 5	U 1	5	6	OCTAL

and tilt ball commands. The program appears in Fig. 7. We've shown it in octal coding to make it more compact.

Most of the characters are directly converted from ASCII to their SELECTRIC equivalents. ASCII < and >become the Selectric 1/2 and 1/4 respectively, the ASCII

> becomes the Selectric ¢, and opening and closing brackets are disallowed and produce question marks. ASCII ↓ becomes a capital "E" to indicate exponentiation, particularly when using the BASIC language on a microcomputer output display.

The eighth output of our PROM is used to detect an ASCII space and break it out of the code, for a Selectric space is a machine command, and an ASCII space is a printing character. If a space is detected, the keypressed output is diverted to the space solenoid and the ball moving solenoids are disabled.

If a lower case alphabet is provided, the input logic on bits a6 and a7 detects lower case and converts it to its equivalent upper case ASCII six bit input code and at the same time forces the shift output line to the lower case low state. This is a good example of how a quarter's worth of gating can cut the size of a ROM in half.

Another Selectric machine command we need is a return command to move the ball back to the beginning of a line. This command can be detected with the lower left NOR gate (Fig. 6) and used to divert the keypressed output to the carriage return solenoid, at the same time disabling the ball-moving solenoids. If there should be any other control commands which are to be ignored by the typewriter, carriage return detection logic (OPTIONAL CR GATING in Fig. 6) can be added to distinguish carriage returns from the added control commands.

You can build this ASCII to Selectric interface using a single 64 x 8 PROM or a pair of 32 x 8 PROMs with parallel outputs selected by the chip enable inputs. Using two 1Cs is sometimes less expensive than one larger PROM because the smaller PROMs are more widely available as surplus. Fig. 5 lists several typical PROM parts.

All Selectric functions including bell, tab, backspace and index are accommodated in the circuit of Fig. 8. It takes a PROM of four times the size, provides more functions, and is a simpler circuit than the one shown in Fig. 6. The eighth PROM output is used to decide whether the ASCII code is a printed character or a machine command (which might be ignored). This eighth output line feeds back to the PROM's eighth input line via the resistor-capacitor time delay circuit. When bits 1 to 7 of the PROM input represent a printable character, the eighth bit output line switches to a high level which enables printing.

The ball output codes are the same as in the simpler circuit, but their solenoids must be disabled when the machine command codes are received. When a character is to be printed, the keypressed pulse (KP in Fig. 8) is routed by the upper AND gate (enabled by the logical one output of the PROM's eighth bit) to the print solenoids; the ball solenoid enable is also taken from the eighth PROM bit. The result is movement of the ball combined with a print stroke.

If, however, a machine command is presented to the PROM at the ASCII inputs. the eighth bit of the PROM output goes low and disables both the ball solenoids and the gate which would route KP to the print solenoid. This bit is inverted by the inverter to present a logical one level to the lower AND gate in the figure. This enables the keypressed pulse (KP) to go to the machine solenoids. Any legitimate command code will result in selection of one of the machine outputs Fig. 8. The i n selected machine output line will enable the corresponding machine control solenoid's driver - resulting in one of the machine control actions such as a tab. As an example, suppose a horizontal tab function (HT in ASCII) is presented to the PROM: The PROM decodes a machine command, inhibits ball solenoids and the print solenoid, presents a decoded logical one level to the tab solenoid driver, enabling it, and routes the keypressed signal (KP) to the other tab solenoid driver input resulting in a tab action. Since the PROM is to be set up for only six (or seven) legitimate machine commands, any unwanted ASCII machine control codes will be ignored.

In this article, we've seen some background information on ROM technology and several of the many uses to which ROMs can be put in microcomputer and logic systems. These are by no means the only uses of ROMs. The uses of ROM technology are for the most part limited only by your own imagination. ■

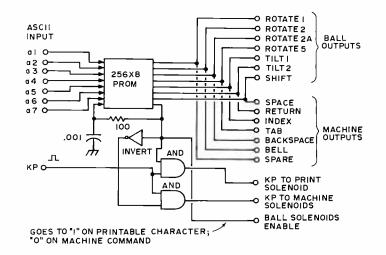


Fig. 8. Full function ASCII to SELECTRIC code converter uses larger PROM.