DEVELOPING LARGE BINARY TO BCD CONVERSION STRUCTURES

By M. Benedek

1. INTRODUCTION

Static binary to BCD conversion has been desscribed in many papers during the last decade, but none of the methods presented were practical for the conversion of large number of binary bits.

In this paper it is intended to further develop static conversions by the expansion of the original BIDEC method (Ref. 1). There are quite a few static conversion schemes published which use other methods of conversion. The static method, developed from the BIDEC, lends itself best for the expansion to larger structures as in its original "correct and shift" form there were no basic limitations on the size of the binary words to be converted. With the appearance of large bipolar ROMs and the even larger but somewhat slower MOS ROMs there is a renewed interest in large structures for fast binary to BCD conversions.

The static method will be extended to well over 30 bits by using large ROMs, and then will show a hybrid conversion scheme (static/dynamic) which will use these large conversion blocks to decode and shift by bytes of up to 13 bits with tolerable penality in speed. Further, to save memory space, a "feed-back" correction method will be shown where one set of converters can be used for both BIN/BCD and BCD/BIN conversions.

2. STATIC CONVERSIONS

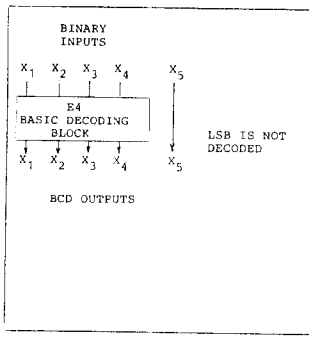
First a brief introduction is given to the algorithm, which is the basis of the method to be developed. The static binary/BCD conversion technique consists of looking at the three most significant bits of the binary number. If the value of the three digits is greater than/or equal to five, add binary three to the number and shift the result one bit to the left. If the three digit value is less than five, shift to the left without adding three. Take the next bit from the right and repeat the operation until reaching the least significant bit which will be carried over without decoding. (See Table 1.)

TABLE 1. BASIC STATIC BINARY/BCD CONVERSION TECHNIQUE

BINARY NUMBER		1 0 1 1 0 1 0 0 = 180
Examine 3 most significant bits		
Add binary 3	1 1	
	1 0 0 0	
Shift left, take next bit 1		0 0 0 1
No add, shift	1 0	0 0 1 0
No add, shift	1 0 0	0 1 0 1
Add 3 to units	1 1	
	1 0 0	1 0 0 0
Shift	1 0 0 1	0 0 0 0
Add 3 to 10's	1 1	
	1 1 0 0	0 0 0 0
Shift	1 1 0 0 0	0 0 0 0
BCD	1 8	0

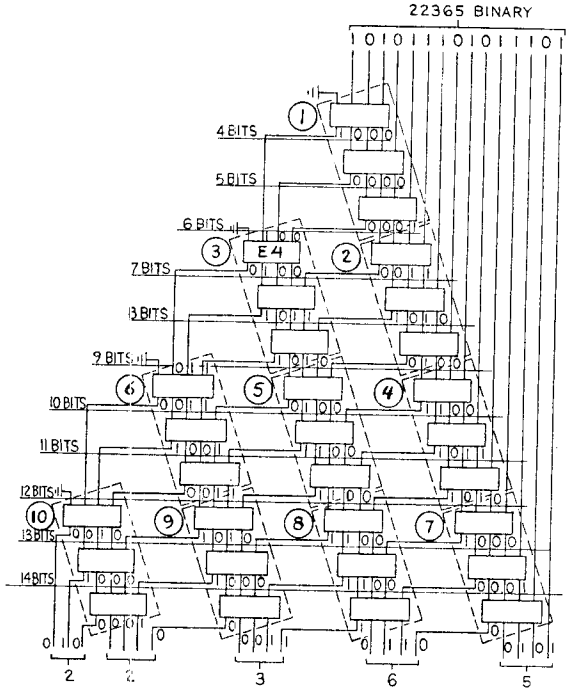
As the algorithm is recursive, it can be synthesized either by gates, as in Reference 2, or by a look up table, as in Reference 3; and it can cascade enough elements to form a decoder for the required binary number. The content of the look up table is shown in Table 2.

TABLE 2. LOOK-UP TABLE CONTENT



INPUT				OUTPUT			
X_1	X_2	X_3	X_4	Y_1	Y_2	Y_3	Y_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

Figure 1 shows the interconnection of these basic (E4) building blocks to form a decoder tree for 4 to 15 bit binary numbers. The tree is cut at the appropriate binary number. As the number of decoder elements increases rapidly with the bits to be converted, this arrangement is very inefficient over 15 binary bits (as shown in Table 4, Column 1).



BINARY TO BCD DECODING TREE WITH E4 DECODERS. THE DOTTED OUTLINES REPRESENT E6 DECODERS.

INTERCONNECTION OF BASIC BUILDING BLOCKS
FIGURE 1

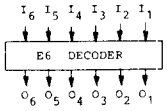
2.1 Developing the E6 Level of Interconnect

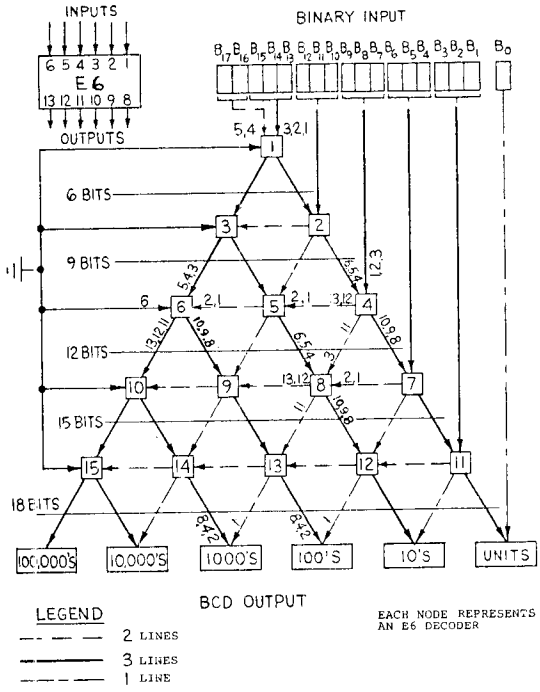
Referring to Figure 1, merge three units into one (dotted line) and create a new decoding unit (E6) with six inputs and six outputs. To generate the truth table for this unit, look back to the basic algorithm. The four least significant output bits have only 10 out of 16 combinations, as in Table 2; and the total number of words stored will be 4 x 10 = 40 (Table 3). This decoder is somewhat similar to the Texas Instruments' 74184. This latter, however, has reduced efficiency as it uses a 32 x 8 Memory Structure (with five inputs).

To build a decoder, for any number of bits, first construct a geometric interconnect map where each node is an E6 Unit (Figure 2). Each arrow represents one or more interconnecting wires as shown on nodes 4, 6 and 8. The advantage of this map is that the decoding tree can be extended to any number of bits by topological similarity and can determine the number of decoder units required for a given number of binary bits (Table 4, Column 6).

TABLE 3. LOOK-UP TABLE FOR E6 DECODER

LINE NO.	INPUT CODE OCTAL $I_6 \sim I_1$	OUTPUT CODE OCTAL $O_6 \sim O_1$
Ø	Ø Ø	Ø Ø
1	Ø 1	Ø 1
2	Ø 2	Ø 2
3	Ø 3	Ø 3
4	Ø 4	Ø 4
5	Ø 5	1 Ø
6	Ø 6	1 1
7	Ø 7	1 2
8	1 Ø	1 3
9	1 1	1 4
1Ø	1 2	2 Ø
11	1 3	2 1
12	1 4	2 2
13	1 5	2 3
14	1 6	2 4
15	1 7	3 Ø
16	2 Ø	3 1
17	2 1	3 2
18	2 2	3 3
19	2 3	3 4
2Ø	2 4	4 Ø
21	2 5	4 1
22	2 6	4 2
23	2 7	4 3
24	3 Ø	4 4
25	3 1	5 Ø
26	3 2	5 1
27	3 3	5 2
28	3 4	5 3
29	3 5	5 4
3Ø	3 6	6 Ø
31	3 7	6 1
32	4 Ø	6 2
33	4 1	6 3
34	4 2	6 4
35	4 3	7 Ø
36	4 4	7 1
37	4 5	7 2
38	4 6	7 3
39	4 7	7 4





Typical interconnections are shown on units 4, 6, and 8. Numbers on connecting lines represent pin numbers.

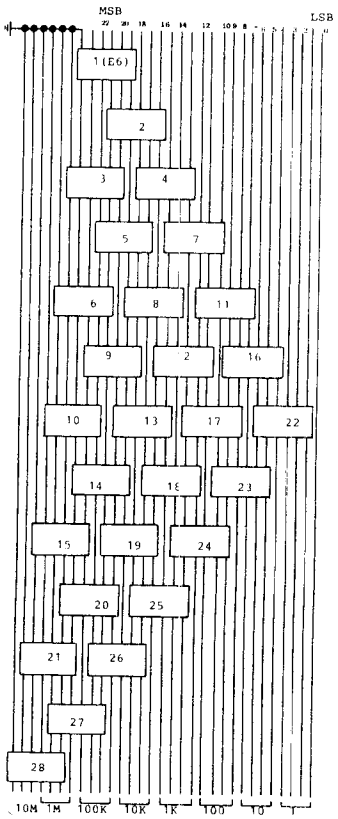
INTERCONNECTION FOR E6 STATIC DECODERS (BINARY/BCD)
FIGURE 2

TABLE 4. TABLE OF COMPARISON STATIC BIN/BCD DECODING

COLUMN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
			E4					E6					E9					E13		
BITS	CONV. UNITS	BITS UNITS	$t_{\mu s}^{ACC}$	NO. OF PINS		CONV. UNITS	BITS UNITS	$t_{\mu s}^{ACC}$	NO. OF PINS		CONV. UNITS	BITS UNITS	$t_{\mu s}^{ACC}$	NO. OF PINS		CONV. UNITS	BITS UNITS	$t_{\mu s}^{ACC}$	NO. OF PINS	
4	1	4	0.05																	
6	3	2	0.15	33		1	6	0.05	15		1	6	0.05	21						
9	9	1	0.30	99		3	3	0.15	45		2	4.5	0.10	42						
12	18	0.67	0.45	198		6	2	0.30	90		4	3	0.20	84		3	4	0.15	87	
15	30	0.5	0.60	330		10	1.5	0.35	150		6	2.5	0.30	126						
18	45	0.4	0.75	495		15	1.2	0.45	225		9	2	0.40	189		6	3	0.25	174	
21	63	0.33	0.90	693		21	1	0.55	315		12	1.75	0.50	252						
24	84	0.28	1.05	924		28	0.86	0.65	420		16	1.5	0.60	336		10	2.4	0.35	290	
27	108	0.25	1.2	1188		36	0.75	0.75	540		20	1.35	0.70	420						
30	135	0.22	1.35	1485		45	0.67	0.85	675		25	1.2	0.80	525		15	2	0.45	435	
33	---					55	0.6	0.95	825		30	1.1	0.90	630						
39	---										36	1	1.00	756		21	1.7	0.55	609	
42	---										42	0.9	1.10	882						
											49	0.85	1.20	1029		27	1.5	0.65	783	
MEMORY REQUIRED	← 10 x 4 →					← 40 x 6 →					← 320 x 9 →					← 3200 x 13 →				

CONVERSION TIME IS CALCULATED BY ASSUMING A 50 ns DELAY PER UNIT.

From this map, construct the actual wiring diagram (Figure 3) by following the interconnect lattices on Figure 2. The interconnection of E6 units will decode efficiently up to about 24 bits (Table 4, Column 6 and 7).



WIRING DIAGRAM FOR 24 BIT BIN/BCD DECODER USING E6 UNITS, DEVELOPED FROM THE GEOMETRIC INTERCONNECT MAP
FIGURE 3

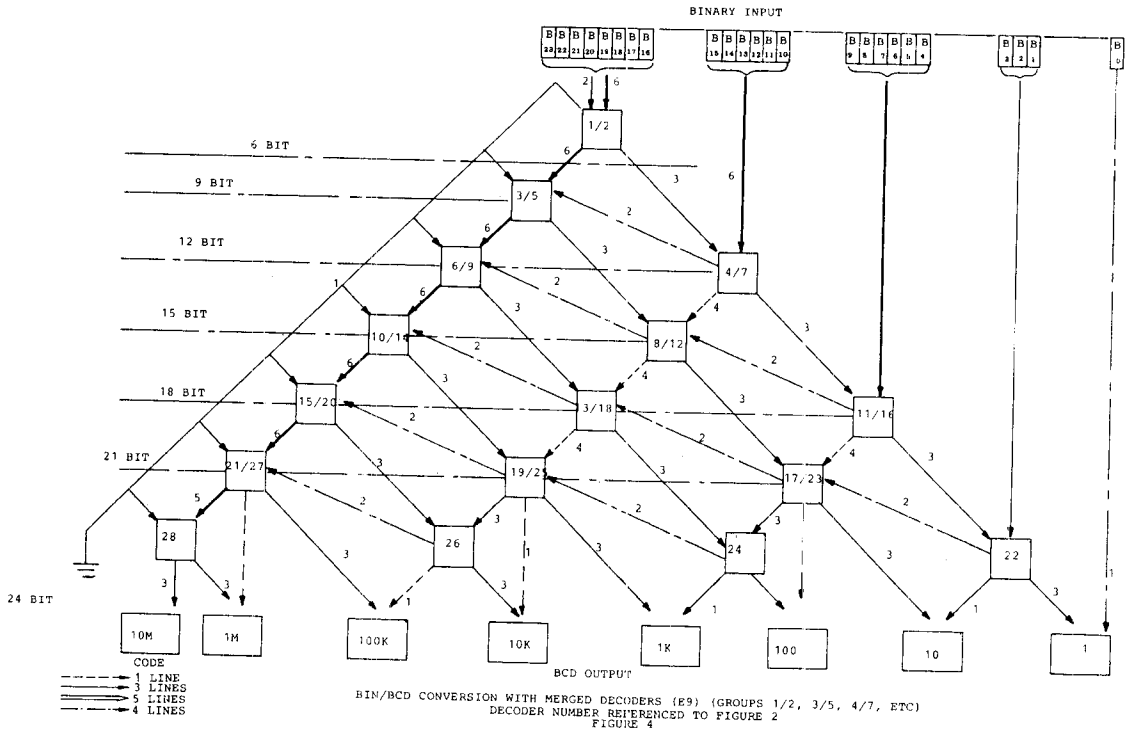
2.2 Developing the E9 Level of Interconnect

With the higher density of the presently available read-only-memories, the complexity of the decoder can be increased. Again use the geometric map of Figure 2 and merge two E6 Units into one, thereby reducing the number of units required for decoding large binary numbers. The new units will have nine inputs and nine outputs (E9).

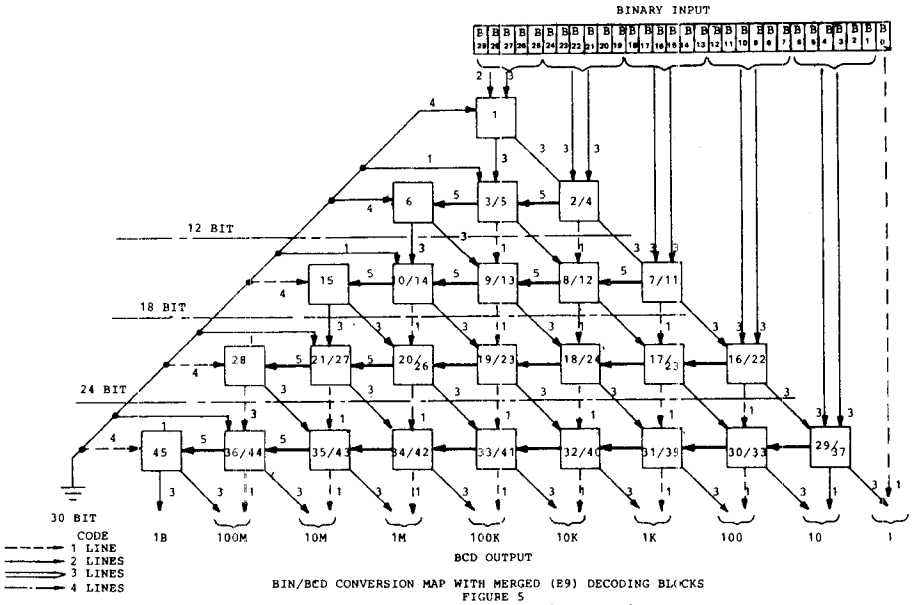
What is the criteria to merge these decoding blocks? First, the newly merged units should be compatible with available ROMs; and second, the signal should flow without feed-back path. (For example: If units 1-3 and units 2-5, are merged there would be two forward signal paths, 1-2 and 3-5 and a feed-back path 2-3. Inevitably a race condition will occur.)

A better choice is the merging of 1-2, 3-5, etc. as shown on Figure 4 or as an alternate, 1, 2-4, 3-5, etc. as shown on Figure 5. By constructing the geometric map and the wiring diagram for both combinations, note that the complexity of both are identical (Figures 6 and 7).

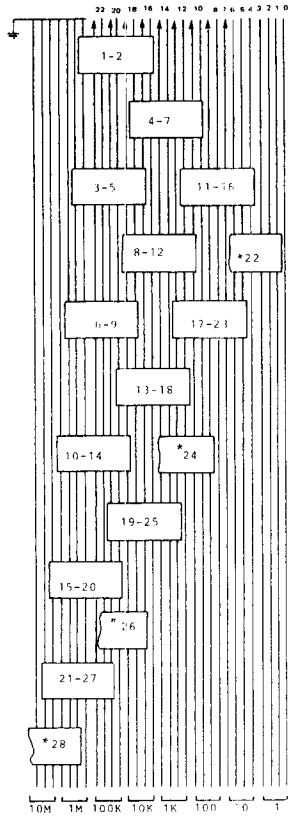
The 1-2, 4-7 approach has a slight advantage since it can be expanded by three binary bits whereas the 1, 2/4 combination can be expanded by six bits only. On the other hand, as will be seen later, the 1, 2/4 etc. is the better combination to be used for the hybrid decoding and has higher decoding speed in the static mode. The look up table for both schemes is identical with nine inputs and nine outputs. Some inefficiencies are being introduced because some of the decoders are not fully used. This can be seen on the interconnect pattern (Unit 1, 6, 15, and 28 on Figure 7). The unused inputs of the truncated units should be grounded or, as an alternate, E6 Units should be used.



BIN/BCD CONVERSION WITH MERGED DECODERS (E9) (GROUPS 1/2, 3/5, 4/7, ETC)
DECODER NUMBER REFERENCED TO FIGURE 2
FIGURE 4



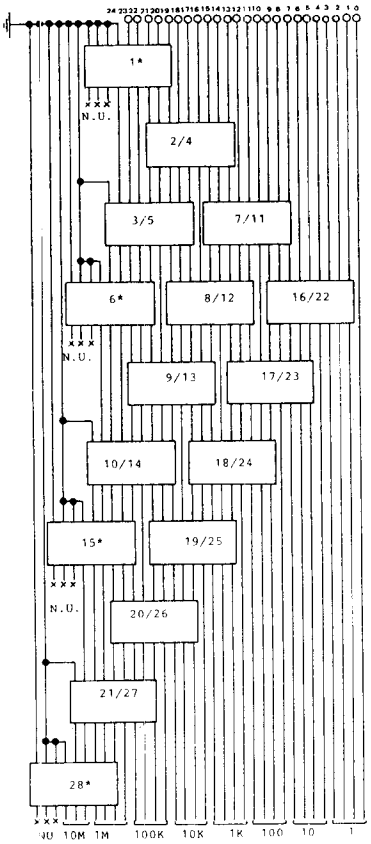
BIN/BCD CONVERSION MAP WITH MERGED (E9) DECODING BLOCKS
FIGURE 5
DECODER NUMBERS REFERENCED TO FIGURE 2



24 BIT IN, 3 DECADES OUT
EXPANSION OF FIGURE 4 MAP

*ONLY 6 LSB, 3 MSB NOT USED

LARGE SCALE BINARY/BCD
CONVERSION USING 16 E9
UNITS. (1/2, 4/7, 3/5
GROUPING)
FIGURE 6



24 BITS IN; 8 DECADES OUT
GROUPING: 1,2/4, 7/11 etc.
EXPANSION OF FIGURE 5 MAP.

*OPTION:
UNIT # 1,6,15,28 CAN BE
REPLACED BY E6 UNITS.

N.U. = NOT USED

LARGE SCALE BINARY TO
BCD CONVERSION, USING
E9 UNITS
FIGURE 7

To develop the program for the E9 Units, observe again the basic algorithm or refer to the map. Unit No. 2 (E6) contains 40 words, by merging it with Unit No. 4, three extra lines are added which can have all possible combinations (eight). Therefore, E9 will store 8 x 40 = 320 words. A simple program will generate the look up table for E9. By decoding with E9 blocks, the conversion efficiency has been improved to over 30 bits.

2.3 Developing the E13 Level of Interconnect

Well, why not go further. An attempt to merge three E6 Units will fail because of feed-back paths. It is possible to merge four E6 units; such as (1, 0, 0, 0), (2, 3, 4, 5), etc. and come up with a 13 in,-13 out composite decoder (E13) (Figures 8, 9 and 10). The problem, however, with this kind of arrangement is that as of today, there are no commercially available 3.2K x 13 ROMs but PLAs (Programmable Logic Array) can be built into this complexity. The E13, can be used very efficiently in the hybrid decoding scheme.

The most important factor of the static BIN/BCD decoding is the conversion delay or the maximum propagation delay through all the decoders. This can be readily calculated from the interconnection diagram by simply counting the number of decoding levels. Notice that the E9 conversion for 24 bits with 1/2, 4/7 grouping (Figure 6) has $12xt_{acc}$ propagation delay where t_{acc} is the access time of one unit. The same E9 conversion with 1, 2-4 etc. grouping (Figure 7) has only $10xt_{acc}$ delay.

Table 4 gives a composite picture of the performance and cost effectiveness of the different structures for up to 42 binary bits.

Columns 1, 6, 11, and 16 give the number of building blocks required for a given number of binary bits using the four configurations discussed. The next column gives a relative efficiency figure or the number of bits converted per building blocks. The ratio of one, considered arbitrarily as the limit of efficiency, is reached at a higher number of bits for large building blocks, so the design efficiency is definitely increased. To establish an approximate measure of the cost effectiveness, all the pins which are connected will be added up regardless of the package or configuration used. Therefore, each E4 block has eight active pins; two pins for the power and one for chip enable making a total of 11 pins connected.

For the E6: 12 active pins + 2 power pins + 1 chip enable = total 15 pins connected

For the E9: Total 21 pins connected

For the E13: Total 29 pins connected.

The total number of interconnected pins in Columns 4, 9, 14 and 19 have been shown. The pin count gives an approximation of the overall cost of the system (including the cost of hardware and assembly), and the cost effectiveness can be determined by comparing the "# of Pin" columns. It emerges that the most effective way of decoding up to 24 bits is by cascading E9's. Above 24 bits, the effectiveness of the E9's are falling behind the E13's, however, not to sharply.

Another conclusion is that the E13 becomes cost effective over 36 bits.

Columns 3, 8, 13, and 18 give the worst case conversion time for a unit delay of 50 ns. For easy conversion to MOS devices, the figure shown should be multiplied by 10.

3. PROGRAMMING THE BUILDING BLOCKS

To derive an algorithm for the programming of the building blocks, again, make use of the conversion maps.

3.1 Algorithm for the E6

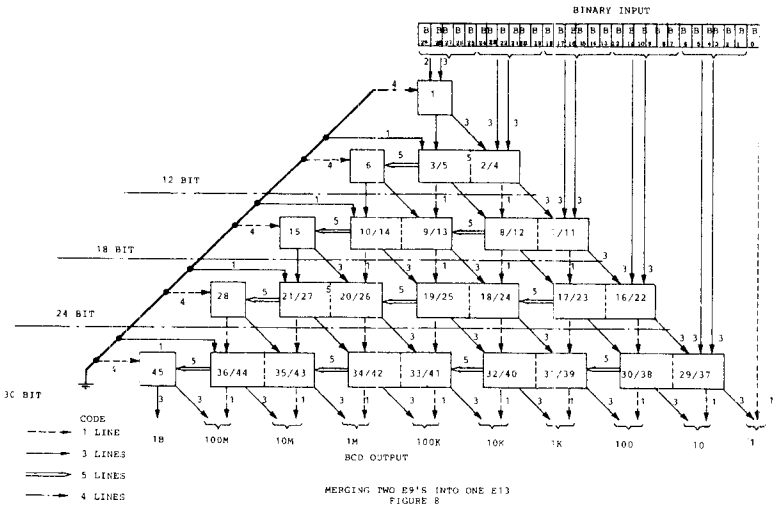
To determine the input/output requirements of the E6 building block, choose a random element (#4) of the E6 map (Figure 2). Unit No. 4 has three inputs from B_7, B_8, and B_9. These inputs can be in any one of eight combinations. The most significant three inputs, however, are already decoded and can assume only one out of five combinations. As for the output lines, the three least significant bits are decoded and will have five combinations, whereas the most significant three have no restrictions. Summarized below:

Input Code		Output Code	
MSB		MSB	
XXX	XXX	XXX	XXX
Mod	Mod	Mod	Mod
5	8	8	5

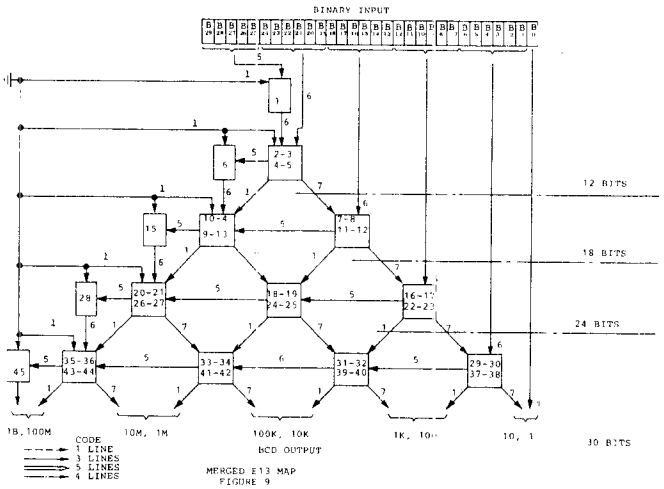
The total number of memory locations required will be: 5 x 8 = 40, and the input/output codes can be calculated as follows:

Line Numbers: 0 to 39

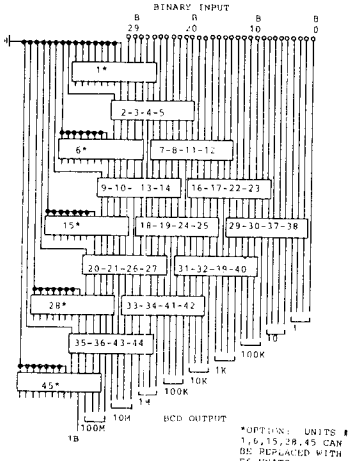
Input Code: Convert the line number into a two digit number: modulo 5 for the higher and modulo 8 for the lower digit by dividing successively by 8 and 5. Then convert the remainder into binary.



MERGING TWO E9'S INTO ONE E13
FIGURE 8



MERGED E13 MAP
FIGURE 9



WIRING DIAGRAM DEVELOPED FROM MAP OF FIGURE 9
FIGURE 10

Output Code: Same, but the lower digit is modulo 5 and the higher, modulo 8.

Example:

Line: 19

Input Code:

19 ÷ 8 = 2, remainder 3 --
2 ÷ 5 = 0, remainder 2 -- input code: $2_5\ 3_8$ = 010 011

Output Code:

19 ÷ 5 = 3, remainder 4 --
3 ÷ 8 = 0, remainder 3 -- output code: $3_8\ 4_5$ 011 100

3.2 Algorithm for the E9

With similar reasoning, establish the following input/output code relationships (Figure 5):

Input Code			Output Code		
MSB			MSB		
XXX	XXX	XXX	XXX	XXX	XXX
Mod	Mod	Mod	Mod	Mod	Mod
5	8	8	8	8	5

Total memory requirements: 8 x 8 x 5 = 320 (number of lines).

Input Code: Convert line number into a three digit number: MOD 5, MOD 8, MOD 8 (LSB).

Output Code: Convert line number into a three digit number: MOD 8, MOD 8, MOD 5 (LSB).

Example:

Line: 177

Input Code: $2_5\ 6_8\ 1_8$ = 010 110 001

Output Code: $4_8\ 3_8\ 2_5$ = 100 011 010

3.3 Algorithm for the E13

The programming requirement for the E13 can be established by the E13 conversion map (Figure 9) and shown as follows:

Input Code				Output Code			
MSB							
XXXX	XXX	XXX	XXX	MSB	XXX	XXXX	XXX
Mod	Mod	Mod	Mod	Mod	Mod	Mod	Mod
10	5	8	8	8	8	10	5

Total memory requirements: 10 x 5 x 8 x 8 = 3200

Input Code: Convert line number to four digit composite modulo number: MOD 10, MOD 5, MOD 8, MOD 8.

Output Code: Convert line number to four digit composite modulo number: MOD 8, MOD 8, MOD 10, MOD 5.

Example:

Line: 3150

Input Code: $9_{10}\ 4_5\ 1_8\ 6_8$ = $(1001\ 100\ 001\ 110)_2$

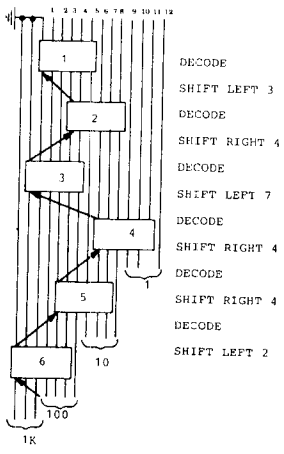
Output Code: $7_8\ 7_8\ 0_{10}\ 0_5$ = $(111\ 111\ 0000\ 000)_2$

Using these algorithms, a simple computer program can be generated to print out the complete look-up table.

4. HYBRID CONVERSION

As it was stated before, the main advantage of the static decoding is the speed of conversion, which has been achieved with a high degree of hardware complexity.

However, by compromising somewhat on speed, the large decoder blocks can be put to another use. If one remembers that the interconnection scheme is a static implementation of the original "decode and shift" method, then a relatively fast converting scheme can be devised by using one of the large blocks as a single decoder and generating shift signals by a small program or subroutine. Figure 11 shows an example. First go back to the wiring diagram for E6 and establish the shifting requirements by pointing an arrow from the most significant bit of a decoder to the most significant bit of the next higher level. The number of interconnecting lines crossed by an arrow is the number of shifts required for the register (in the direction pointing by the arrows). Note that the total number of left shifts has to be equal to the total number of right shifts, so a correction may be necessary at the end of the conversion. From the shifting schedule, a program can be generated either by discrete logic or as a subroutine to a higher level program (Figure 12). This particular example requires 24 clock pulses. Operating at 10 MHz, the conversion would require 3μs as compared to 0.5μs with the total static approach. The conversion can be performed on an accumulator without additional registers



TOTAL SHIFT LEFT : 3 + 7 + 2 =12
TOTAL SHIFT RIGHT: 4 + 4 + 4 =12

DEVELOPMENT OF THE HYBRID DECODING METHOD FROM THE WIRING DIAGRAM OF THE STATIC 12 BIT (E6) DECODING
FIGURE 11

therefore, this configuration requires minimum hardware. The system is optimal where the utmost in speed is not required. Further refinement can be made by using a double (or triple) decoder instead of the single one shown in Figure 12. This would reduce the timing requirements to 16 clock pulses or to 1.6μs (at 10 MHz) but would require two decoders. The decoder which is not in use during a particular program step would not be loaded into the register. More improvement could be made by using static shift registers such as AM2510 or by cascading 54153's for single or multiple shifting in one or both directions and from then on the possible variations are limitless depending on the ever conflicting requirements for high speed and minimum hardware.

Table 5 gives the conversion times for hybrid decoding using E6, E9, or E13 decoders in single, double, and for the E9 in triple configuration. In mini-or micro-computer applications the conversion can be made part of the main program by storing the truth table in the main memory and using part of the binary number as the memory address. Note, that speed would not be greatly affected even with machines with slow instruction cycles, because the memory fetch (decode and load) instruction is infrequent compared to the shift sequence; and many computers can perform multiple shifting during one instruction cycle.

Because of the extent of the subject, the discussion is restricted to the Binary to BCD conversion. Structures for the BCD/BIN conversions have been developed simultaneously. The E6 modules have a companion decoder, the E7 (7 BCD in; 7 BIN out) and E9 has an opposite, the E11.

TABLE 5. CONVERSION TIME FOR HYBRID DECODING (μS) (50 ns UNIT DELAY, 10 MHz SHIFT CLOCK)

COLUMN		1	2	3	4	5	6	7	8	9	10
DECODER		E4			E6		E9			E13	
METHOD		SINGLE	DOUBLE	TRIPLE	SINGLE	DOUBLE	SINGLE	DOUBLE	TRIPLE	SINGLE	DOUBLE
NO OF PINS		11	22	33	15	30	27	42	63	29	58
BITS	12	7.2	2.4	1.6	2.4	1.6	1.6			1.6	
	15	8.6	6.0	3.0	0.4	2.4					
	18				5.6	0.4	5.2	3.2		4.4	
	21				11	6.2					
	24				15	8.2	11.0	5.6	4.2	8.8	
	27										
	30				2.0	16*	21*	10.5	6.4	14.4	7.6
	36				80*	32*	40*	20*	11*	25*	
	42				12.0*	64*	75*	38	16.5	42*	

*BY EXTRAPOLATION

5. BCD/BINARY DECODING USING BINARY/BCD DECODER IN THE "FEED-BACK" MODE

BCD/Binary decoding can be accomplished by assigning an arbitrary value to a binary register, converting the content of this register to BCD, comparing it to the original BCD number, and correcting the binary register until the two BCD numbers are equal. This method is essentially a successive approximation scheme, and works as follows (Figure 13).

R1 register holds the binary number. Rx is a static binary to BCD decoder and Ro register holds the BCD number to be converted to binary. Start by setting R1 to all ones. After decoding the content of R1, compare it to Ro. If R1 > Ro, the most significant bit of R1 is set to zero. Decode and compare again. If R1 is still high, set the next most signficiant bit to zero and set the previous bit to one and continue until the last bit, or until R1 = Ro. Figure 12 gives the schematic and the algorithm of this approximation method. The setting and resetting of the successive bits are accomplished either by a ring counter or a shift register with a one rippling through, as shown in Figure 13.

6. CONCLUSION

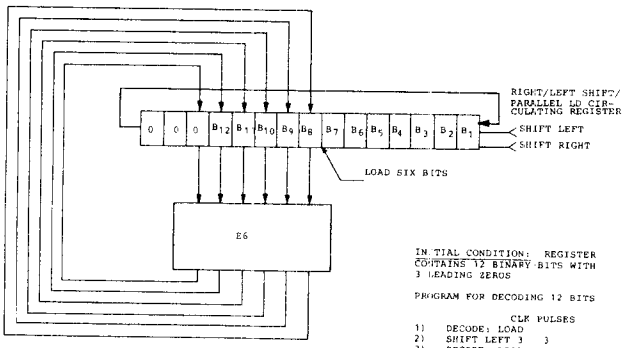
A BIN/BCD conversion method has been developed which lends itself to unlimited expansion by using the geometrical similarity of interconnecting maps. Properly designed, the maps then contain all the necessary information to determine the size, the content, and the actual wiring of the decoding ROMs. Wiring diagrams were used to develop the hybrid conversion scheme. Conversion systems for practically any speed or size can be designed by using either the static or the hybrid method. This scheme is not limited to binary/BCD conversion, but can be extended to other types of conversion as well, such as, synchro/BCD, etc.

7. REFERENCES

1) Couleur, J.F., A Binary to Decimal or Decimal to Binary Converter, IRE Transaction on Electronic Computers, EC-7, No. 4, 1958, p. 313.

2) Benedek, Z.M. and Moskowitz, B., Convert Binary to BCD Without Flip-Flops, Electronic Design, October 10, 1968, p. 58.

3) Linford, John, Code Conversion with Semiconductor Read Only Memories, Motorola Application Note, No. 506.



IN TIAL CONDITION: REGISTER CONTAINS 12 BINARY BITS WITH 3 LEADING ZEROS

PROGRAM FOR DECODING 12 BITS

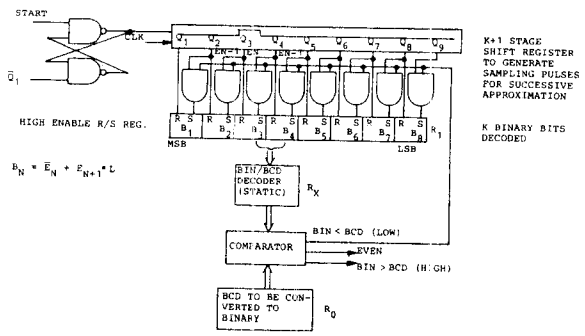
		CLK PULSES
1)	DECODE; LOAD	
2)	SHIFT LEFT 3	3
3)	DECODE; LOAD	
4)	SHIFT RIGHT 4	4
5)	DECODE; LOAD	
6)	SHIFT LEFT 7	7
7)	DECODE; LOAD	
8)	SHIFT RIGHT 4	4
9)	DECODE; LOAD	
10)	SHIFT RIGHT 4	4
11)	DECODE; LOAD	
12)	SHIFT LEFT 2	2
	TOTALS	34

EX. INPUT: 2967_{12}

	PROGRAM STEPS
000101110010111	START
E6 (1)	DECODE
001000110010111	LOAD
000110010111001	SHIFT LEFT 3
E6 (2)	DECODE
001010000111001	LOAD
110000111001101	SHIFT RIGHT 4
E6 (3)	DECODE
100110111000111	LOAD
110000111001101	SHIFT LEFT 7
E6 (4)	DECODE
110000111001101	LOAD
110111000011100	SHIFT RIGHT 4
E6 (5)	DECODE
110101100011100	LOAD
110011101011001	SHIFT RIGHT 4
E6 (6)	DECODE
110101001011001	LOAD
010100101100111	SHIFT LEFT 2, END

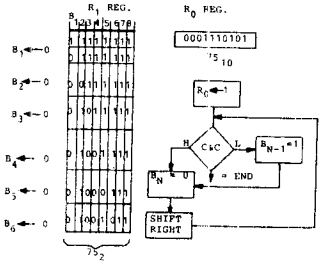
$2\ 9\ 6\ 7$

HYBRID (STATIC/DYNAMIC) DECODING FOR MINIMUM HARDWARE WITH E6
FIGURE 12



CONVERSION ALGORITHM

SET R_1 TO ALL ONE
4 CONVERT TO BCD; COMPARE: R_1 = HIGH;
3 SHIFT RIGHT R_S
2 CONVERT; COMPARE: R_1 = HIGH;
3 SHIFT RIGHT R_S
REPEAT 2 ; R_1 = LOW; B_2 = 1;
REPEAT 3
REPEAT 2 ; R_1 = HIGH;
REPEAT 3
REPEAT 2 ; R_1 = HIGH;
REPEAT 3
REPEAT 2 ; R_1 = LOW; B_5 = 1
REPEAT 2 ; R_1 = EVEN; END



BCD/BIN CONVERSION BY SUCCESSIVE APPROXIMATION USING BIN/BCD CONVERTERS

FIGURE 13