Designing with Texas Instruments Field-Programmable Logic

Robert K. Breuninger and Loren E. Schiele

Contributors
Bob Gruebel, Renee Tanaka, Jim Ptasinski

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

TI warrants performance of its semiconductor products, including SNJ and SMJ devices, to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems such testing necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

Contents

® PAL is a registered trademark of Monolithic Memories Inc.

List of Illustrations

List of Tables

INTRODUCTION

The purpose of this application report is to provide the first time user of field-programmable logic with a basic understanding of this new and powerful technology. The term "Field-Programmable Logic" refers to any device supplied with an uncommitted logic array, which the user programs to his own specific function. The most common, and widely known field-programmable logic family is the PROM, or Programmable Read-Only Memory. Relatively new entries into this expanding family of devices are the PAL® and FPLA. This report will primarily concentrate on the PAL family of programmable logic.

FIELD-PROGRAMMABLE LOGIC ADVANTAGES

Field-programmable logic offers many advantages to the system designer who presently is using several standard catalog SSI and MSI functions. Listed below are just a few of the benefits which are achievable when using programmable logic.

1. Package Count Reduction: typically, 3 to 6 MSI/SSI functions can be replaced with one PAL or FPLA.
2. PC Board Area Reduced: Fewer devices consume less PC board space. This results in lower PC board cost.
3. Circuit Flexibility: Programmability allows for minor circuit changes without changing PC boards.
4. Improved Reliability: With fewer PC interconnects, overall system reliability increases.
5. Shorter Design Cycle: When compared with standard-cell or gate-array approaches, custom functions can be implemented much more quickly.

The PAL and FPLA, will fill the gap between standard logic and large scale integration. The versatility of these devices provide a very powerful tool for the system designer.

PAL AND FPLA SYMBOLOGY

In order to keep PAL and FPLA logic easy to understand and use, a special convention has been adopted. Figure 1 is the representation for a 3-input AND gate. Note that only one line is shown as the input to the AND gate. This line is commonly refered to as the product line. The inputs are shown as vertical lines, and at the intersection of these lines are the programmable fuses. An X represents an intact fuse. This makes that input, part of the product term. No X represents a blown fuse. This means that input will not be part of the product term (in Figure 1, input B is not part of the product term). A dot at the intersection of any line represents a hard wire connection.

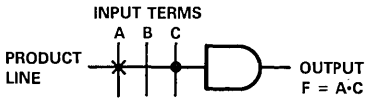


Figure 1. Basic Symbology

In Figure 2, we will extend the symbology to develop a simple 2-input programmable AND array feeding an OR gate. Notice that buffers have been added to the inputs, which provide both true and complement outputs to the product lines. The intersection of the input terms form a 4×3 programmable AND array. From the above symbology, we can see that the output of the OR gate is programmed to the following equation, $A\overline{B}+\overline{A}B$. Note that the bottom AND gate has an X marked inside the gate symbol. This means that all fuses are left intact, which results in that product line not having any effect on the sum term. In other words, the output of the AND gate will be a logic 0. **When all the fuses are blown on a product line, the output of the AND gate will always be a logic 1. This has the effect of locking up the output of the OR gate to a logic level 1.**

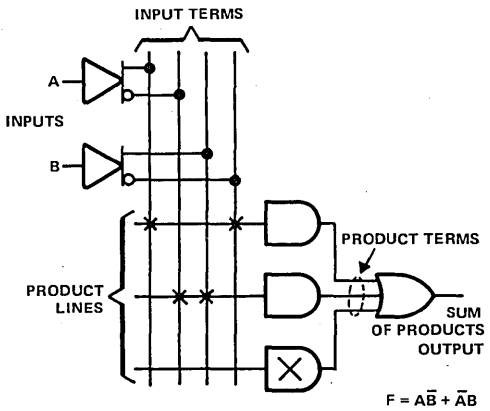


Figure 2. Basic Symbology Example

®PAL is a Registered Trademark of Monolithic Memories Inc.

FAMILY ARCHITECTURES

As stated before, the PROM was the first widely used programmable logic family. Its basic architecture is an input decoder configured from AND gates, combined with a programmable OR matrix on the outputs. As shown in Figure 3, this allows every output to be programmed individually from every possible input combination. In this example, a PROM with 4 inputs has 2^4, or 16 possible input combinations. With the output word width being 4 bits, each of the 16×4 bit words can be programmed individually. Applications such as data storage tables, character generators, and code converters, are just a few design examples which are ideally suited for the PROM. In general, any application which requires every input combination to be programmable, is a good candidate for a PROM. However, PROMs have difficulty accommodating large numbers of input variables. Eventually, the size of the fuse matrix will become prohibitive because for each input variable added, the size of the fuse matrix doubles. Currently, manufacturers are not producing PROMs with over 13 inputs.

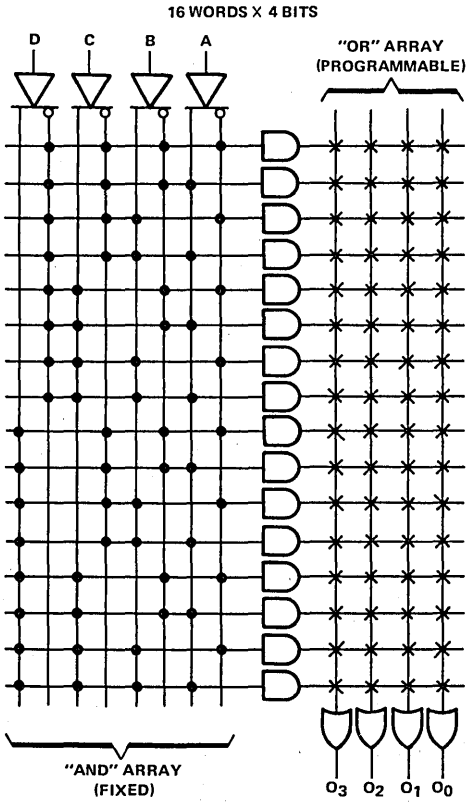


Figure 3. PROM Architecture

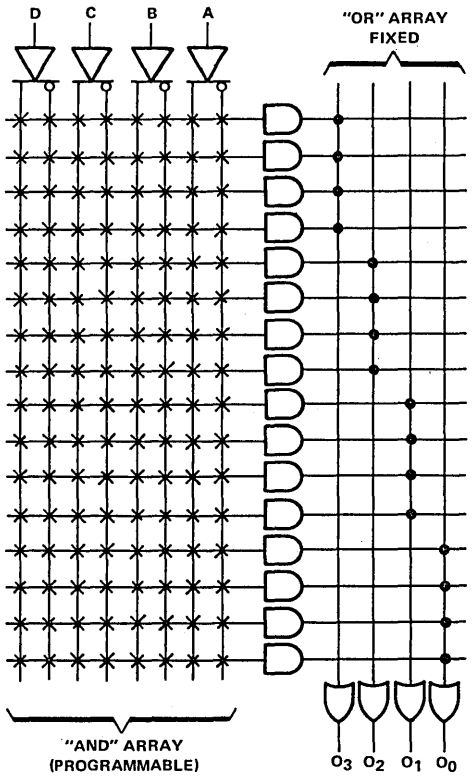


Figure 4. PAL Architecture

To overcome the limitation of a restricted number of inputs, the PAL utilizes a slightly different architecture as shown in Figure 4. The same AND-OR implementation is used as with PROMs, but now the input AND array is programmable instead of the output OR array. This has the effect of restricting the output OR array to a fixed number of input AND terms. The trade-off is that now, every output is not programmable from every input combination, but more inputs can be added without doubling the size of the fuse matrix. For example, If we were to expand the inputs on the PAL shown in Figure 4, to 10, and on the PROM in Figure 3, to 10. We would see that the fuse matrix required for the PAL would be 20×16 (320 fuses) vs 4×1024 (4096 fuses for the PROM). **It is important to realize that not every application requires every output be programmable from every input combination. This is what makes the PAL a viable product family.**

The FPLA goes one step further in offering both a programmable AND array, and a programmable OR array (Figure 5). This feature makes the FPLA the most versatile device of the three, but usually impractical in most low complexity applications.

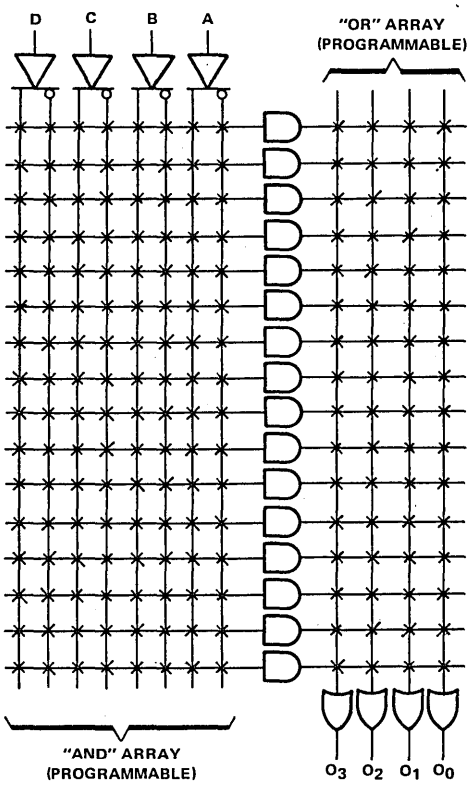


Figure 5. FPLA Architecture

All three field-programmable logic approaches discussed have their own unique advantages and limitations. The best choice depends on the complexity of the function being implemented and the current cost of the devices themselves. It is important to realize, that a circuit solution may exist from more than one of these logic families.

PAL OPTIONS

Figure 6 shows the logic diagram of the popular TIBPAL16L8. Its basic architecture is the same as discussed in the previous section, but with the addition of some special circuit features. First notice that the PAL has 10 simple inputs. In addition, 6 of the outputs operate as I/O ports. This allows feedback into the AND array. One AND gate in each product term controls each 3-state output. The architecture used in this PAL makes it very useful in generating all sorts of combinational logic.

Another important feature about the logic diagram, and all other block diagrams supplied from individual datasheets, are that there are no X's marked at every fuse location. From the previous convention; we stated that everywhere there was a intact fuse, there was an X. However, in order to make the logic diagram useful when generating specific functions, it is supplied with no X's. This allows the user to insert the X's wherever an intact fuse is desired.

The basic concept of the TIBPAL16L8 can be expanded further to include D-type flip-flops on the outputs. An example of this is shown in Figure 7 with the TIBPAL16R8. This added feature allows the device to be configured as a counter, simple storage register, or similar clocked function.

Circuit variations which are available on other members of the TI PAL and FPLA family are explained below.

Polarity Fuse

The polarity of the output can be selected via the fuse shown in Figure 8.

Input Registers

On PALs equipped with this special feature, the option of having D-type input registers is fuse programmable. Figure 9 shows an example of this type of input. If the fuse is left intact, data enters on a low-high transition of the clock. If the fuse is blown, the register becomes permanently transparent and is equivalent to a normal input buffer.

Input Latches

On PALs equipped with this special feature, the option of having input latches is fuse programmable.

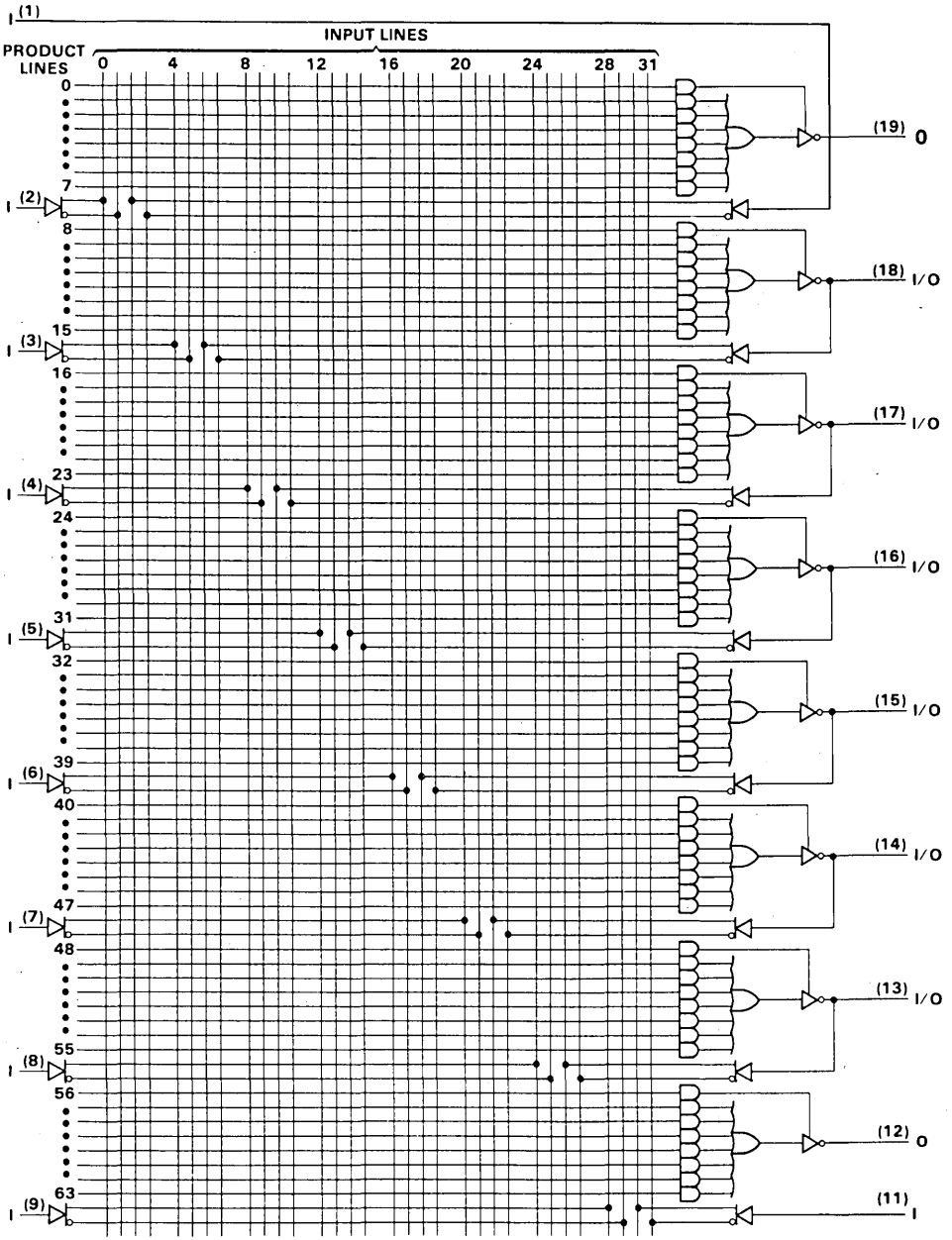


Figure 6. TIBPAL16L8 Logic Diagram

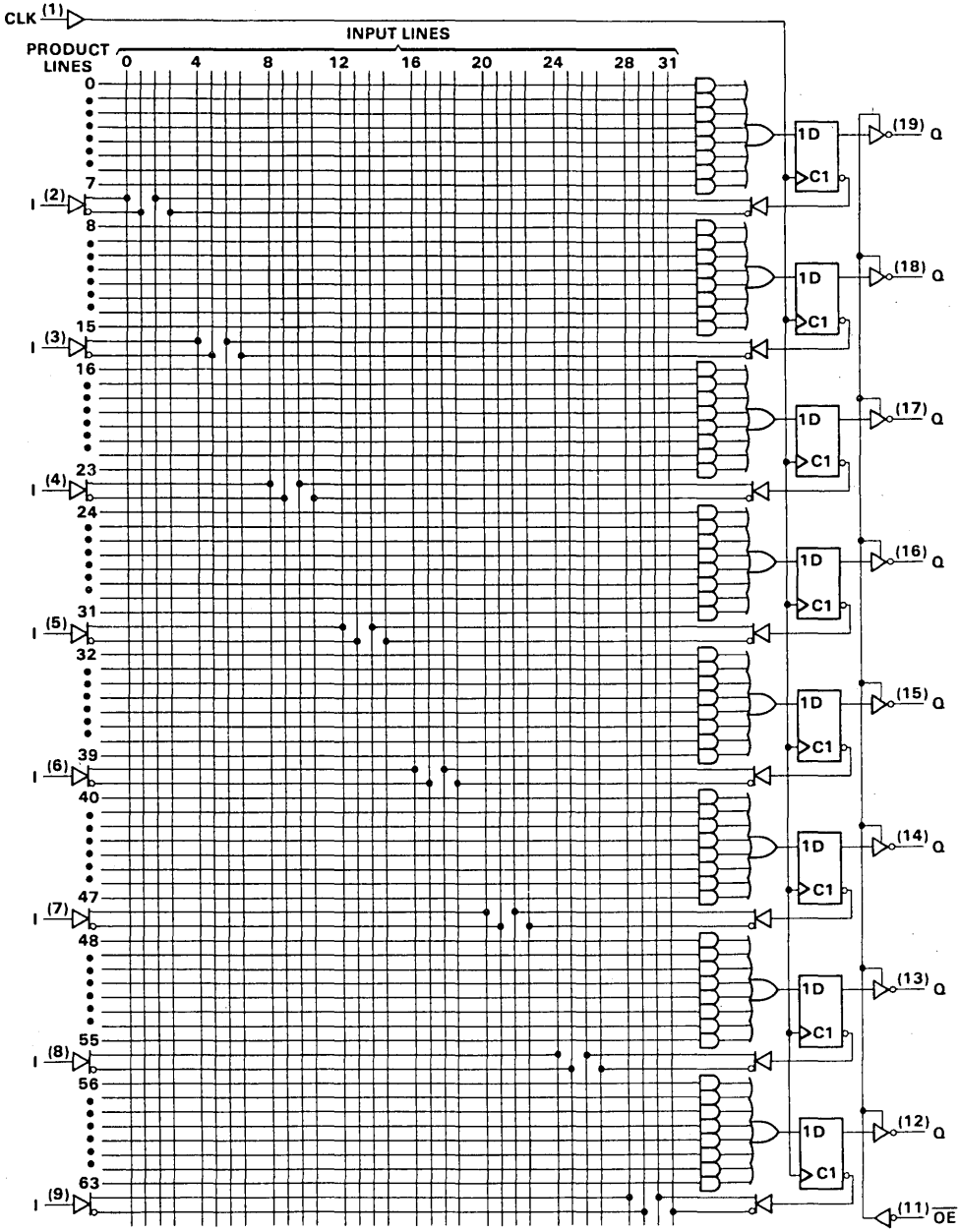


Figure 7. TIBPAL16R8 Logic Diagram

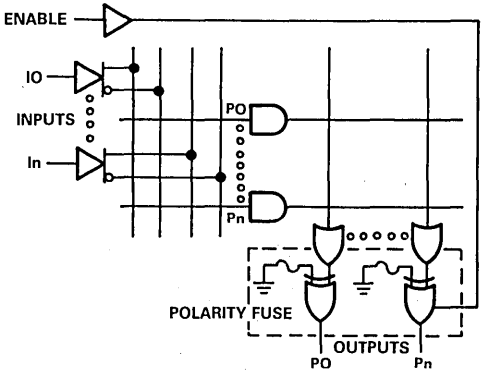


Figure 8. Polarity Selection

Figure 10 shows an example of this type of input. If the fuse is left intact, data enters while the control input is high. When the control input is low, the data that was present when the control input went low will be saved. If the fuse is blown, the latch becomes permanently transparent, and is equivalent to a normal input buffer.

PROGRAMMING

Notice in Figure 7, that the product and input lines are numbered. This allows any specific fuse to be located anywhere in the fuse matrix. When the device is in the programming mode (as defined in the device data sheet), the individual product and input lines can be selected. The fuse at the intersection of these lines, can then be blown (programmed) with the defined programming pulse. Fortunately, the user seldom has to get involved with these actual details of programming, because there exist several commercially available programmers which handle this function. Listed below are some of the manufacturers of this programming equipment.*

Citel	Storey Systems
DATA I/O	Structured Design
Digelec	Sunrise Electronics
Kontron	Valley Data Science
Wavetec	Varix
Stag Micro Systems	

At Texas Instruments, we have coordinated with DATA I/O using their Model 19 for device characterization. Currently, DATA I/O, Sunrise, and Structured Design have been certified by Texas Instruments. Other programmers are now in the certification process. For a current list of certified programmers, please contact your local TI sales representative.

It should now be obvious to the reader, that the actual blowing of the fuses is not a problem. Instead, the real question is what fuses need to be blown to generate a particular function. Fortunately, this problem has also been greatly simplified by recent advances in computer software.

DATA I/O has developed a software package called ABEL™. Also available is CUPL™, from Assisted Technology. Both have been designed to be compatible with several different types of programmers. Both of these software packages greatly extend the capabilities of the original PALASM™ program, and both can be run on most professional computers.

Before proceeding to a design example, it would be instructive to look at the simplified process flow of a PAL (Figure 11). This should help give the reader a better understanding of the basic steps necessary to generate a working device.

DESIGN EXAMPLE

The easiest way to demonstrate the unique capabilities of the PAL is through a design example. It is

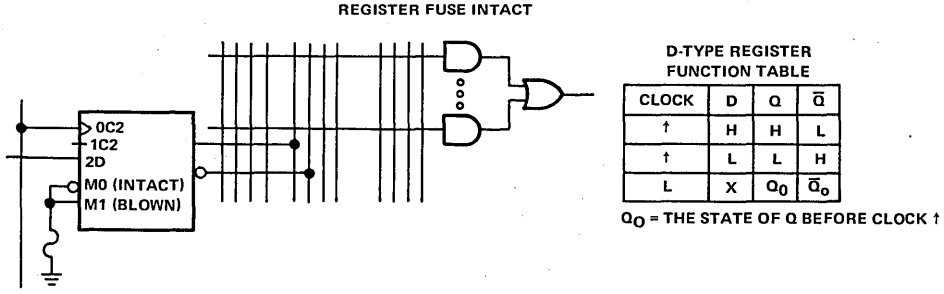
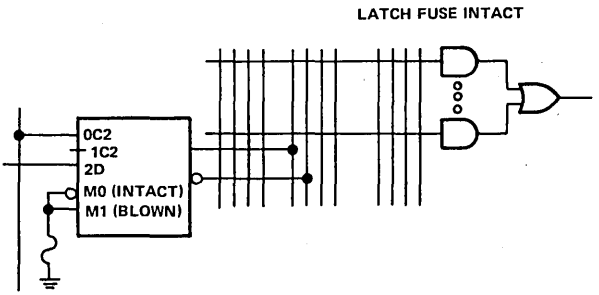


Figure 9. Input Register Selection

ABEL™ is a trademark of DATA I/O.
CUPL™ is a trademark of Assisted Technology, Inc.
PALASM™ is a trademark of Monolithic Memories Inc.



TRANSPARENT LATCH FUNCTION TABLE

ENABLE	D	O	$\overline{Q}$
H	L	L	H
H	H	H	L
L	X	Q_0	$\overline{Q}_0$

Q_0 = THE LEVEL OF Q BEFORE ENABLE ↓

Figure 10. Input Latch Selection

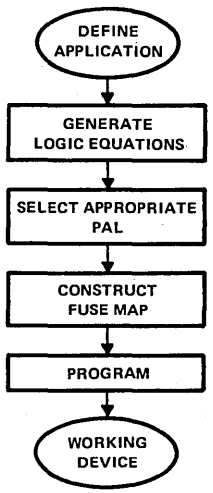


Figure 11. PAL Process Flow Diagram

hoped that through this example the reader will gain the basic understanding needed when applying the PAL in his own application. In some cases, this goal may only be to reduce existing logic, but the overall approach will be the same.

EXAMPLE REQUIREMENTS

It is desired to generate a 4-bit binary counter which is fed by one of four clocks. There are two lines available for selecting the clocks, SEL1 and SEL0. Table 1 shows the required input for the selection of the clocks. In addition, it is desired that the counter be able to switch from binary to decade count. This feature is controlled by an input called BD. When BD is high, the counter should count in binary. When low, the counter should count in decade.

Figure 12 shows how this example could be implemented if standard data book functions were used.

Table 1. Clock Selection

SEL1	SEL0	OUTPUT
0	0	CLKA
0	1	CLKB
1	0	CLKC
1	1	CLKD

As can be seen, three MSI functions are required. The 'LS162 is used to generate the 4-bit counter while the clock selection is handled by the 'LS253. The 'LS688 is an 8-bit comparator which is used for selecting either the binary or decade count. In this example, only five of the eight comparator inputs are used. Four are used for comparing the counter outputs, while the other is used for the BD input. The comparator is hard-wired to go low whenever the BD input is low and the counter output is "9". The $\overline{P=Q}$ output is then fed back to the synchronous clear input on the 'LS162. This will reset the counter to zero whenever this condition occurs.

PAL IMPLEMENTATION

As stated before, the problem in programming a PAL is not in blowing the fuses, but rather what fuses need to be blown to generate a particular function. Fortunately, this problem has been greatly simplified by computer software, but before we examine these techniques, it is beneficial to explore the methods used in generating the logic equations. This will help develop an understanding, and appreciation for these advanced software packages.

From digital logic theory, we know that most any type of logic can be implemented in either AND-OR-INVERT or AND-NOR form. This is the basic concept used in the PAL and FPLA. This allows classical techniques, such as Karnaugh Maps[1] to be used in generating specific logic functions. As with the separate component example above, it is easier to break it into separate functions. The first one that we will look at is the clock selector, but remember that the overall goal will be to reduce this design example into one PAL.

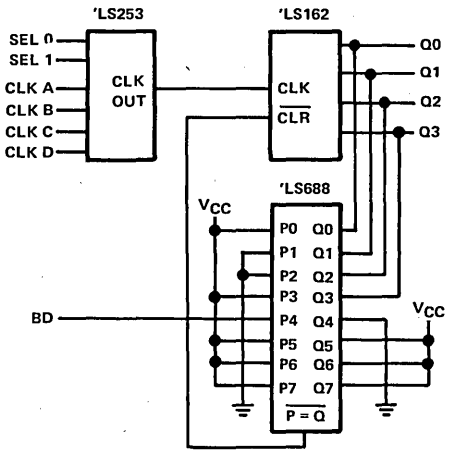


Figure 12. Counter Implementation With Standard Logic

PAL SELECTION

Before proceeding with the design for the clock selector, the first question which needs to be addressed is which PAL to use. As discussed earlier, there are several different types of output architectures. Looking at our example, we can see that four flip-flops with feedback will be required in the 4-bit counter, plus input clock and clear lines. In addition, seven inputs plus two simple outputs will be required in the clock selector and comparator. With this information in hand, we can see that the TIBPAL16R4 (Figure 13) will handle our application.

CLOCK SELECTOR DETAILS

The first step in determining the logic equation for the clock selector is to generate a function table with all the possible input combinations. This is shown in Table 2. From this table, the Karnaugh map can be generated and is shown in Figure 14. The minimized equation for CLKOUT comes directly from this.

Table 2. Function Table

SEL1	SEL0	CLKA	CLKB	CLKC	CLKD	CLKOUT	SEL1	SEL0	CLKA	CLKB	CLKC	CLKD	CLKOUT
0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	1	0	0	1	0	0	0	1	0	1
0	0	0	0	1	1	0	1	0	0	0	1	1	1
0	0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	1	0	0	1	0	1	0
0	0	0	1	1	0	0	1	0	0	1	1	0	1
0	0	0	1	1	1	0	1	0	0	1	1	1	1
0	0	1	0	0	0	1	1	0	1	0	0	0	0
0	0	1	0	0	1	1	1	0	1	0	0	1	0
0	0	1	0	1	0	1	1	0	1	0	1	0	1
0	0	1	0	1	1	1	1	0	1	0	1	1	1
0	0	1	1	0	0	1	1	0	1	1	0	0	0
0	0	1	1	0	1	1	1	0	1	1	0	1	0
0	0	1	1	1	0	1	1	0	1	1	1	0	1
0	0	1	1	1	1	1	1	0	1	1	1	1	1
0	1	0	0	0	0	0	1	1	0	0	0	0	0
0	1	0	0	0	1	0	1	1	0	0	0	1	1
0	1	0	0	1	0	0	1	1	0	0	1	0	0
0	1	0	0	1	1	0	1	1	0	0	1	1	1
0	1	0	1	0	0	1	1	1	0	1	0	0	0
0	1	0	1	0	1	1	1	1	0	1	0	1	1
0	1	0	1	1	0	1	1	1	0	1	1	0	0
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	0	0	0	1	1	1	0	0	0	0
0	1	1	0	0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	0	0	1	1	1	0	1	0	0
0	1	1	0	1	1	0	1	1	1	0	1	1	1
0	1	1	1	0	0	1	1	1	1	1	0	0	0
0	1	1	1	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	0	1	1	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1

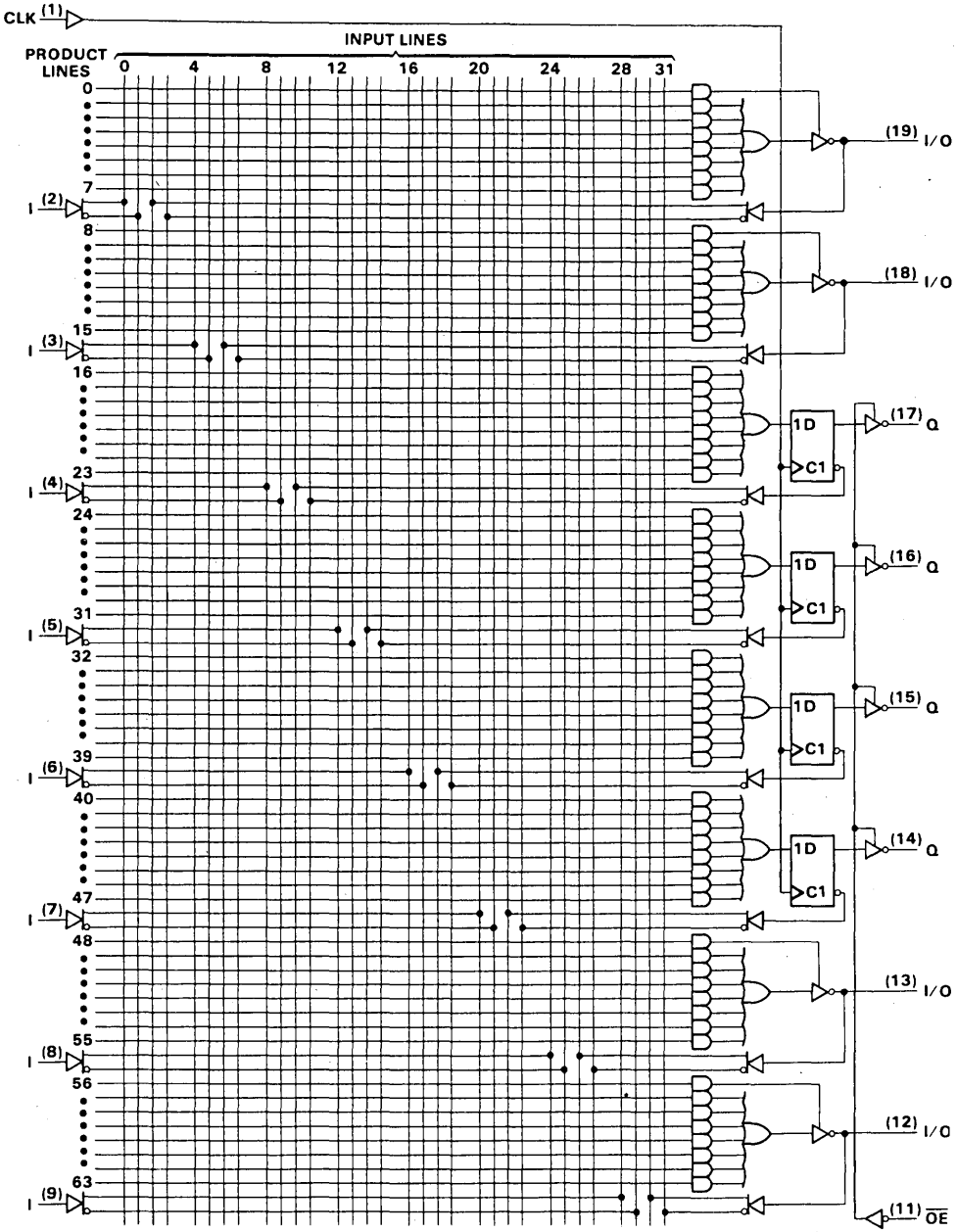
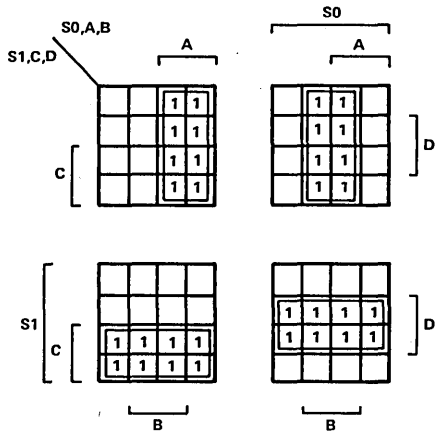


Figure 13. TIBPAL16R4 Logic Diagram

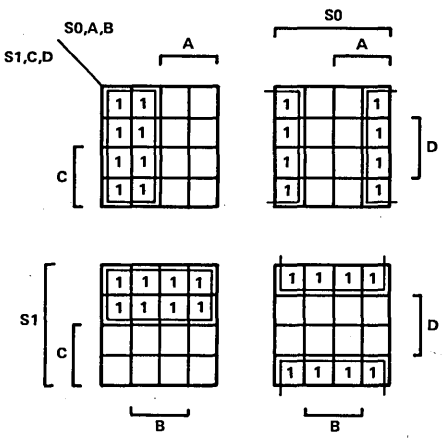
It is important to notice that the equation derived from the Karnaugh map is stated in AND-OR notation. The PAL that we have selected is implemented in AND-NOR logic. This means we either have to do DeMorgan's theorem on the equation, or solve the inverse of the Karnaugh map. Figure 15 shows the inverse of the Karnaugh map and the resulting equation. This equation can be easily implemented in the TIBPAL16R4.



$CLKOUT = \overline{S1}\,\overline{S0}A + \overline{S1}S0B + S1\overline{S0}C + S1S0D$

$CLKOUT = \overline{S1}\,\overline{S0}A + \overline{S1}S0B + S1\overline{S0}C + S1S0D$

Figure 14. Karnaugh Map for CLKOUT



$\overline{CLKOUT} = \overline{S1}\,\overline{S0}A + \overline{S1}S0B + S1\overline{S0}C + S1S0D$

$\overline{CLKOUT} = \overline{S1}\,\overline{S0}A + \overline{S1}S0B + S1\overline{S0}C + S1S0D$

Figure 15. Karnaugh Map for $\overline{CLKOUT}$

4-BIT BINARY COUNTER DETAILS

The same basic procedure used in determining the equations for the clock selector, is used in determining the equations for the 4-bit counter. The only difference is that now we are dealing with a present state, next state situation. This means a D-type flip-flop will be required in actual circuit implementation. As before, the truth table is generated first, and is shown in Table 3.

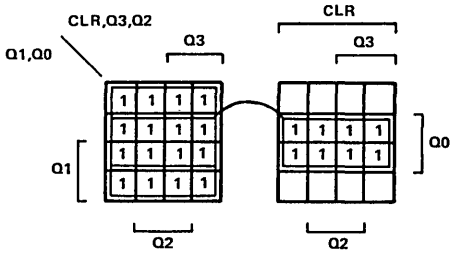
Table 3. Truth Table

	PRESENT STATE				NEXT STATE			
CLR	Q3	Q2	Q1	Q0	Q3	Q2	Q1	Q0
0	X	X	X	X	0	0	0	0
1	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	1	1
1	0	0	1	1	0	1	0	0
1	0	1	0	0	0	1	0	1
1	0	1	0	1	0	1	1	0
1	0	1	1	0	0	1	1	1
1	0	1	1	1	1	0	0	0
1	1	0	0	0	1	0	0	1
1	1	0	0	1	1	0	1	0
1	1	0	1	0	1	0	1	1
1	1	0	1	1	1	1	0	0
1	1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	1	0
1	1	1	1	0	1	1	1	1
1	1	1	1	1	0	0	0	0

From the truth table, the equations for each output can be derived from the Karnaugh map. This is shown in Figure 16. Note that the inverse of the truth table is being solved so that the equation will come out in AND-NOR logic form.

BINARY/DECADE COUNT DETAILS

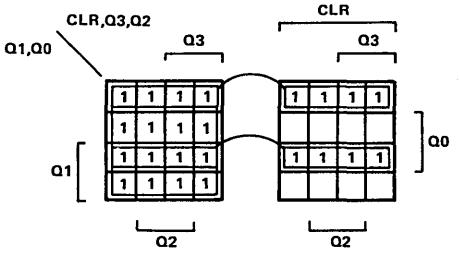
Recalling from the example requirements that the counter should count in decade whenever the BD input is low, we can again generate a truth table for this function (Table 4). Since the counter is already designed to count in binary, we can use this feature to simplify our design. What we desire is a circuit whose output goes low, whenever the BD input is equal to a logic level "0", and the counter output is equal to "9". This output can then be fed back to the CLR input of the counter so that it will reset whenever the BD input is low. Whenever the BD input is high, the output of the circuit should be a high since the counter will automatically count in binary. Notice that $\overline{Q}$ shown in the truth table is the function we desire.



$\overline{Q0} = \overline{CLR}\overline{Q3Q2Q1Q0} + \overline{CLR}Q3Q2Q1Q0$

$\overline{Q0} = \overline{CLR} + Q0$

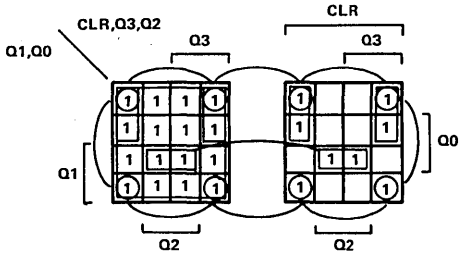
(a) KARNAUGH MAP FOR $\overline{Q0}$



$\overline{Q1} = \overline{CLR}\overline{Q3Q2Q1Q0} + \overline{CLR}Q3Q2\overline{Q1}\overline{Q0} + \overline{CLR}Q3Q2Q1Q0$

$\overline{Q1} = \overline{CLR} + \overline{Q1}\overline{Q0} + Q1Q0$

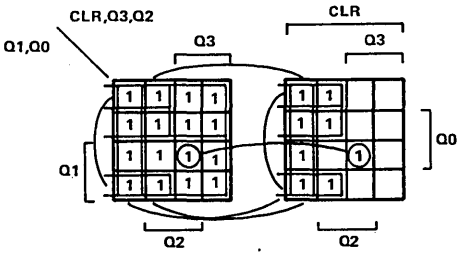
(b) KARNAUGH MAP FOR $\overline{Q1}$



$\overline{Q2} = \overline{CLR}\overline{Q3Q2Q1Q0} + \overline{CLR}Q3\overline{Q2}\overline{Q1}\overline{Q0} + \overline{CLR}Q3Q2Q1Q0 + \overline{CLR}Q3\overline{Q2}Q1\overline{Q0}$

$\overline{Q2} = \overline{CLR} + \overline{Q2}\overline{Q1} + Q2Q1Q0 + \overline{Q2}\overline{Q0}$

(c) KARNAUGH MAP FOR $\overline{Q2}$



$\overline{Q3} = \overline{CLR}\overline{Q3Q2Q1Q0} + \overline{CLR}\overline{Q3}Q2\overline{Q1}\overline{Q0} + \overline{CLR}\overline{Q3}Q2Q1\overline{Q0} + \overline{CLR}Q3Q2Q1Q0$

$\overline{Q3} = \overline{CLR} + \overline{Q3}Q2 + \overline{Q3}\overline{Q1} + \overline{Q3}\overline{Q0} + Q3Q2Q1Q0$

(d) KARNAUGH MAP FOR $\overline{Q3}$

Figure 16. Karnaugh Maps

In this particular example, a Karnaugh map is not required because the equation cannot be further simplified. The resulting equation is given below.

$$\overline{BD\ OUT} = \overline{BD}Q3\overline{Q2}\overline{Q1}Q0$$

Table 4. Truth Table

BD	Q3	Q2	Q1	Q0	Q	$\overline{Q}$	BD	Q3	Q2	Q1	Q0	Q	$\overline{Q}$
0	0	0	0	0	0	1	1	0	0	0	0	0	1
0	0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	0	0	1	1	0	0	1	0	0	1
0	0	0	1	1	0	1	1	0	0	1	1	0	1
0	0	1	0	0	0	1	1	0	1	0	0	0	1
0	0	1	0	1	0	1	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0	1	1	0	0	1
0	0	1	1	1	0	1	1	0	1	1	1	0	1
0	1	0	0	0	0	1	1	1	0	0	0	0	1
0	1	0	0	1	1	0	1	1	0	0	1	0	1
0	1	0	1	0	0	1	1	1	0	1	0	0	1
0	1	0	1	1	0	1	1	1	0	1	1	0	1
0	1	1	0	0	0	1	1	1	1	0	0	0	1
0	1	1	0	1	0	1	1	1	1	0	1	0	1
0	1	1	1	0	0	1	1	1	1	1	0	0	1
0	1	1	1	1	0	1	1	1	1	1	1	0	1

FUSE MAP DETAILS

Now that the logic equations have been defined, the next step will be to specify which fuses need to be blown. Before we do this however, we first need to label the input and output pins on the TIBPAL16R4. By using Figure 12 as a guide, we can make the following pin assignments in Figure 17.

PIN

1	CLK	20	VCC
2	SEL0	19	CLKOUT
3	SEL1	18	NC
4	CLKA	17	Q0
5	CLKB	16	Q1
6	CLKC	15	Q2
7	CLKD	14	Q3
8	CLR	13	NC
9	BD	12	BD OUT
10	GND	11	OE

With this information defined, we now need to insert the logic equations into the logic diagram as shown in Figure 17.

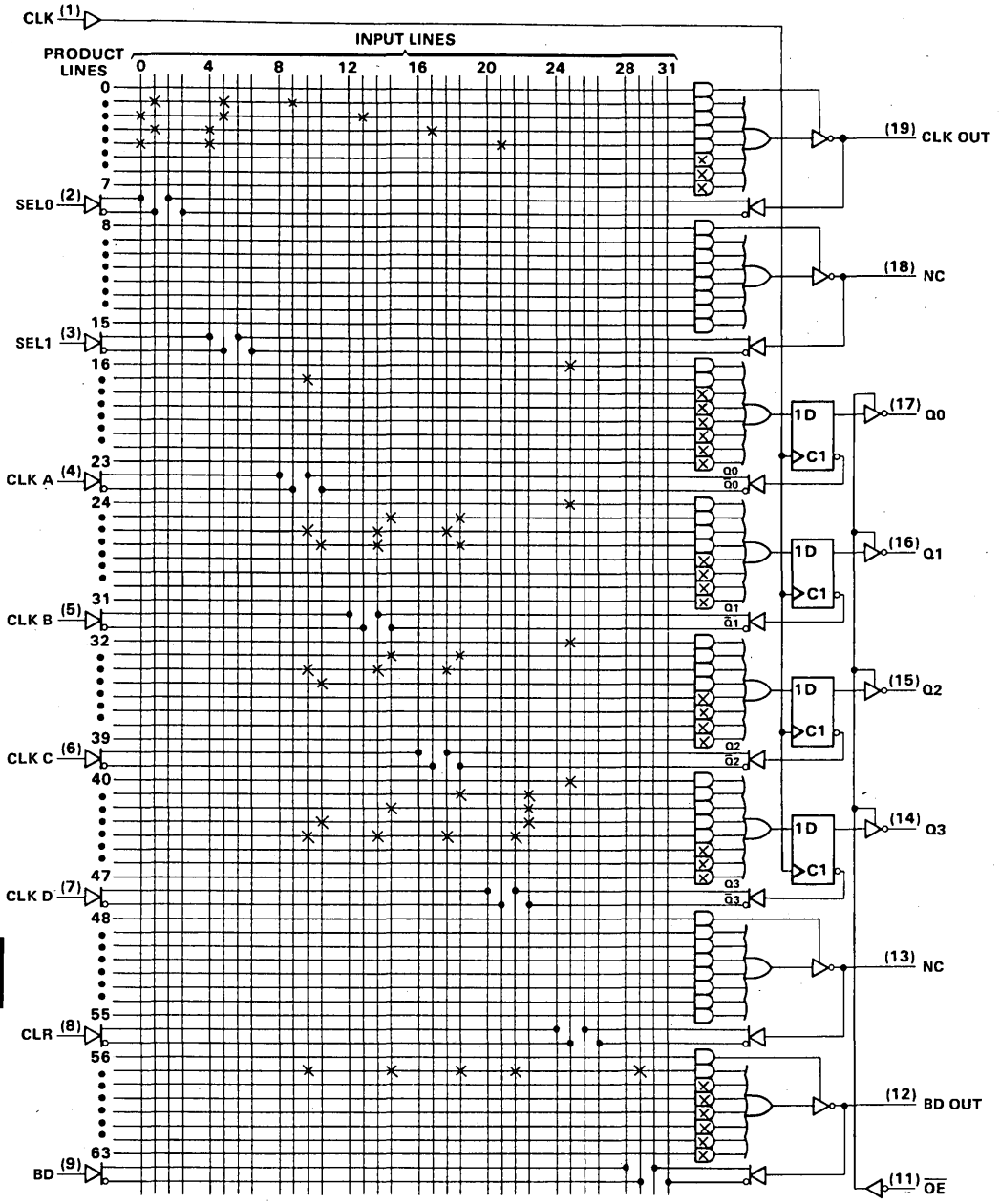


Figure 17. Programmed TIBPAL16R4

It is now probably obvious to the reader, that inserting the logic equations into the logic diagram is a tedious operation. Fortunately, a computer program called PALASM will perform this task automatically. All that is required is telling the program which device has been selected, and defining the input and output pins with their appropriate logic equations (Figure 18). The program will then generate a fuse map (Figure 19) for the device selected. Notice that the fuse map looks very similar to the block diagram (Figure 17) which we have just completed by hand. In addition, this information can now be down loaded into the selected device programmer.

```
DEVICE TYPE 16R4


PIN LIST NAMES =
PIN NUMBER =  1    PIN NAME = CLK
PIN NUMBER =  2    PIN NAME = SEL0
PIN NUMBER =  3    PIN NAME = SEL1
PIN NUMBER =  4    PIN NAME = CLKA
PIN NUMBER =  5    PIN NAME = CLKB
PIN NUMBER =  6    PIN NAME = CLKC
PIN NUMBER =  7    PIN NAME = CLKD
PIN NUMBER =  8    PIN NAME = CLR
PIN NUMBER =  9    PIN NAME = BD
PIN NUMBER = 10    PIN NAME = GND
PIN NUMBER = 11    PIN NAME = /OE
PIN NUMBER = 12    PIN NAME = BDOUT
PIN NUMBER = 13    PIN NAME = NC
PIN NUMBER = 14    PIN NAME = Q3
PIN NUMBER = 15    PIN NAME = Q2
PIN NUMBER = 16    PIN NAME = Q1
PIN NUMBER = 17    PIN NAME = Q0
PIN NUMBER = 18    PIN NAME = NC
PIN NUMBER = 19    PIN NAME = CLKOUT
PIN NUMBER = 20    PIN NAME = VCC

EXPRESSIONS  AND DESCRIPTION =
EXPRESSION[ 1] =
/CLKOUT=/SEL1*/SEL0*/CLKA +/SEL1*SEL0*/CLKB +SEL1*/SEL0*/CLKC +SEL1*SEL0*/CLKD


EXPRESSION[ 2] =
/Q0=/CLR +Q0


EXPRESSION[ 3] =
/Q1=/CLR +/Q1*/Q0 +Q1*Q0


EXPRESSION[ 4] =
/Q2=/CLR +/Q2*/Q1 +Q2*Q1*Q0 +/Q2*/Q0


EXPRESSION[ 5] =
/Q3=/CLR +/Q3*/Q2 +/Q3*/Q1 +/Q3*/Q0 +Q3*Q2*Q1*Q0


EXPRESSION[ 6] =
/BDOUT=/BD*Q3*/Q2*/Q1*Q0
```

Figure 18. Pin ID and Logic Equations

```
0000 0000 0011 1111 1111 2222 2222 2233
0123 4567 8901 2345 6789 0123 4567 8901
/CLKOUT  =
---- ---- ---- ---- ---- ---- ---- ----  0 -
-X-- -X-- -X-- ---- ---- ---- ---- ----  1 - /SEL1*/SEL0*/CLKA+
X--- -X-- ---- -X-- ---- ---- ---- ----  2 - /SEL1*SEL0*/CLKB+
-X-- X--- ---- ---- -X-- ---- ---- ----  3 - SEL1*/SEL0*/CLKC+
X--- X--- ---- ---- ---- -X-- ---- ----  4 - SEL1*SEL0*/CLKD
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX  5 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX  6 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX  7 -
        =
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX  8 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX  9 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 10 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 11 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 12 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 13 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 14 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 15 -
/Q0      =
---- ---- ---- ---- ---- ---- -X-- ---- 16 - /CLR+
---- ---- --X- ---- ---- ---- ---- ---- 17 - Q0
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 18 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 19 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 20 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 21 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 22 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 23 -
/Q1      =
---- ---- ---- ---- ---- ---- -X-- ---- 24 - /CLR+
---- ---- ---X ---X ---- ---- ---- ---- 25 - /Q1*/Q0+
---- ---- --X- --X- ---- ---- ---- ---- 26 - Q1*Q0
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 27 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 28 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 29 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 30 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 31 -
/Q2      =
---- ---- ---- ---- ---- ---- -X-- ---- 32 - /CLR+
---- ---- ---- ---X ---X ---- ---- ---- 33 - /Q2*/Q1+
---- ---- --X- --X- --X- ---- ---- ---- 34 - Q2*Q1*Q0+
---- ---- ---X ---- ---X ---- ---- ---- 35 - /Q2*/Q0
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 36 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 37 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 38 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 39 -
/Q3      =
---- ---- ---- ---- ---- ---- -X-- ---- 40 - /CLR+
---- ---- ---- ---- ---X ---X ---- ---- 41 - /Q3*/Q2+
---- ---- ---- ---X ---- ---X ---- ---- 42 - /Q3*/Q1+
---- ---- ---X ---- ---- ---X ---- ---- 43 - /Q3*/Q0+
---- ---- --X- --X- --X- --X- ---- ---- 44 - Q3*Q2*Q1*Q0
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 45 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 46 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 47 -
        =
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 48 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 49 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 50 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 51 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 52 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 53 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 54 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 55 -
/BDOUT   =
---- ---- ---- ---- ---- ---- ---- ---- 56 -
---- ---- --X- ---X ---X --X- ---- -X-- 57 - /BD*Q3*/Q2*/Q1*Q0
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 58 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 59 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 60 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 61 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 62 -
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX 63 -
```

Figure 19. Fuse Map

ADVANCED SOFTWARE

PALASM, while extremely useful in generating the fuse map, does little to help formulate the logic equations. This is what the new software packages such as ABEL and CUPL address. They not only generate the fuse map, but they also help in developing the logic equations. In most cases, they can generate the logic equations from simply providing the program with either a truth table or state diagram. In addition, they can test the logic equations against a set of test vectors. This helps ensure the designer gets the desired function.

These are only a few of the features available on these new advanced software packages. We recommend that the reader contact the specific manufacturers themselves to obtain the latest information available. For your convenience, at the end of this application note we have included the addresses and phone numbers for many of these programming and software companies.

As an example, we will approach our previous design utilizing DATA I/O's ABEL package. The purpose here is not to teach the reader how to use ABEL, but rather to give them a basic overview of this powerful software package. Figure 20 shows the source file required by ABEL. Note that the 4-bit counter has been described with a state diagram table. When the ABEL program is complied, the logic equations will be generated from this. The equations for CLK OUT and BD OUT have been given in their final form to demonstrate how ABEL would handle these. Also notice that test vectors are included for checking the logic equations. This is especially important when only the logic equations has been given.

Figure 21 shows some of the output documentation generated by the program. Notice that the equations generated for the counter, match the the ones generated by the Karnaugh maps. A pinout for the device has also been generated and displayed. The fuse map for the device has not been shown, but looks very similar to the one in Figure 19. As with the PALASM program, this information can be down loaded into the device programmer.

PERFORMANCE

Up to this point, nothing has been said about the performance of these devices. The Standard High Speed PAL (indicated by an "A" after the device number) offered by TI has a maximum propagation of 25 ns from input to output, and 35 MHz f_{max}. Also available is a new, higher speed family of devices called TIBPALs. These devices are functionally equivalent with the current family and offer a maximum propagation delay of 15 ns from input to output. They are also rated at 50 MHz f_{max}. The higher speeds on these devices make them compatible with most high-speed logic families. This allows them to be designed into more critical speed path applications.

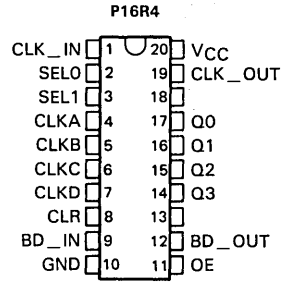
```
module BD_COUNT flag '-r2'
title '4-bit binary/decade counter

      IC1  device 'P16R4';

"  pin assignments and constant declarations
      CLK_IN,SEL0,SEL1,CLKA    pin  1,2,3,4;
      CLKB,CLKC,CLKD           pin  5,6,7;
      CLR,BD_IN,OE             pin  8,9,11;
      BD_OUT,CLK_OUT           pin  12,19;
      Q3,Q2,Q1,Q0              pin  14,15,16,17;
      CK, L, H, X, Z  =        .C. , 0 , 1 , .X. , .Z.;
      OUTPUT          =        [Q3,Q2,Q1,Q0];

"  counter states
      S0=^b0000;   S4=^b0100;   S8=^b1000;   S12=^b1100;
      S1=^b0001;   S5=^b0101;   S9=^b1001;   S13=^b1101;
      S2=^b0010;   S6=^b0110;   S10=^b1010;  S14=^b1110;
      S3=^b0011;   S7=^b0111;   S11=^b1011;  S15=^b1111;

equations
"  clock selector
      CLK_OUT = CLKA & !SEL0 & !SEL1 # CLKB & !SEL1 & SEL0
                # CLKC & SEL1 & !SEL0 # CLKD & SEL1 & SEL0;

"  count nine indicator for decade counting
      BD_OUT  = !(!BD_IN & Q3 & !Q2 & !Q1 & Q0);

state_diagram [Q3,Q2,Q1,Q0]
      State  S0:  IF CLR == 0 THEN S0 ELSE  S1;
      State  S1:  IF CLR == 0 THEN S0 ELSE  S2;
      State  S2:  IF CLR == 0 THEN S0 ELSE  S3;
      State  S3:  IF CLR == 0 THEN S0 ELSE  S4;
      State  S4:  IF CLR == 0 THEN S0 ELSE  S5;
      State  S5:  IF CLR == 0 THEN S0 ELSE  S6;
      State  S6:  IF CLR == 0 THEN S0 ELSE  S7;
      State  S7:  IF CLR == 0 THEN S0 ELSE  S8;
      State  S8:  IF CLR == 0 THEN S0 ELSE  S9;
      State  S9:  IF CLR == 0 THEN S0 ELSE S10;
      State S10:  IF CLR == 0 THEN S0 ELSE S11;
      State S11:  IF CLR == 0 THEN S0 ELSE S12;
      State S12:  IF CLR == 0 THEN S0 ELSE S13;
      State S13:  IF CLR == 0 THEN S0 ELSE S14;
      State S14:  IF CLR == 0 THEN S0 ELSE S15;
      State S15:  IF CLR == 0 THEN S0 ELSE  S0;

test_vectors   'clock selector'
     ([CLKA, CLKB, CLKC, CLKD, SEL1, SEL0] -> CLK_OUT)
      [ L  ,  X  ,  X  ,  X  ,   L,    L ] ->    L;
      [ H  ,  X  ,  X  ,  X  ,   L,    L ] ->    H;
      [ X  ,  L  ,  X  ,  X  ,   L,    H ] ->    L;
      [ X  ,  H  ,  X  ,  X  ,   L,    H ] ->    H;
      [ X  ,  X  ,  L  ,  X  ,   H,    L ] ->    L;
      [ X  ,  X  ,  H  ,  X  ,   H,    L ] ->    H;
      [ X  ,  X  ,  X  ,  L  ,   H,    H ] ->    L;
      [ X  ,  X  ,  X  ,  H  ,   H,    H ] ->    H;

test_vectors   'counter'
     ([CLK_IN, OE, CLR, BD_IN] -> [OUTPUT, BD_OUT])
      [  CK,   L ,  L,     X  ] -> [  S0,   H ];
      [  CK,   L ,  H,     X  ] -> [  S1,   H ];
      [  CK,   L ,  H,     X  ] -> [  S2,   H ];
      [  CK,   L ,  H,     X  ] -> [  S3,   H ];
      [  CK,   L ,  H,     X  ] -> [  S4,   H ];
      [  CK,   L ,  H,     X  ] -> [  S5,   H ];
      [  CK,   L ,  H,     X  ] -> [  S6,   H ];
      [  CK,   L ,  H,     X  ] -> [  S7,   H ];
      [  CK,   L ,  H,     X  ] -> [  S8,   H ];
      [  CK,   L ,  H,     L  ] -> [  S9,   L ];
      [  CK,   L ,  H,     X  ] -> [ S10,   H ];
      [  CK,   L ,  H,     X  ] -> [ S11,   H ];
      [  CK,   L ,  H,     X  ] -> [ S12,   H ];
      [  CK,   L ,  H,     X  ] -> [ S13,   H ];
      [  CK,   L ,  H,     X  ] -> [ S14,   H ];
      [  CK,   L ,  H,     H  ] -> [ S15,   H ];
      [  CK,   L ,  H,     X  ] -> [  S0,   H ];
      [   X,   H ,  X,     X  ] -> [   Z ,  H ];
end BD_COUNT
```

Figure 20. Source File for ABEL

```
ABEL(tm) Version  1.00  - Document Generator
4-bit binary/decade counter

Equations for Module BD_COUNT

Device IC1


     Reduced Equations:

       CLK_OUT = !((SEL1 & SEL0 & !CLKD
                  # (SEL1 & !SEL0 & !CLKC
                  # (!SEL1 & SEL0 & !CLKB
                  # !SEL1 & !SEL0 & !CLKA)));


       BD_OUT = !(Q3 & !Q2 & !Q1 & Q0 & !BD_IN);

       Q3 := !((Q3 & Q2 & Q1 & Q0
             # (!Q3 & !Q2
             # (!Q3 & !Q1
             # (!Q3 & !Q0
             # !CLR)))));


       Q2 := !((Q2 & Q1 & Q0 # (!Q2 & !Q1 # (!Q2 & !Q0 # !CLR))));

       Q1 := !((Q1 & Q0 # (!Q1 & !Q0 # !CLR)));

       Q0 := !((Q0 # !CLR));
```



```
ABEL(tm) Version  1.00  - Document Generator
4-bit binary/decade counter

Chip diagram for Module BD_COUNT

Device IC1
```




```
end of module BD_COUNT
```

Figure 21. ABEL Output Documentation

ADDRESS FOR PROGRAMMING AND SOFTWARE MANUFACTURERS*

HARDWARE MANUFACTURERS

Citel
3060 Raymond St.
Santa Clara, CA 95050
(408) 727-6562

DATA I/O
10525 Willows Rd.
Redmond, WA 98052
(206) 881-6444

DIGITAL MEDIA
3178 Gibralter Ave.
Costa Mesa, CA 92626
(714) 751-1373

Kontron Electronics
630 Price Avenue
Redwood City, CA 94063
(415) 361-1012

Stag Micro Systems
528-5 Weddell Drive
Sunnyvale, CA 94086
(408) 745-1991

Storey Systems
3201 N. Hwy 67, Suite H
Mesquite, Tx 75150
(214) 270-4135

Structured Design
1700 Wyatt Dr., Suite 7
Santa Clara, CA 95054
(408) 988-0725

Sunrise Electronics
524 S. Vermont Avenue
Glendora, CA 91740
(213) 914-1926

Valley Data Sciences
2426 Charleston Rd.
Mountain View, CA 94043
(415) 968-2900

Varix
1210 Campbell Rd.
Richardson, TX 75081
(214) 437-0777

Wavetec/Digelec
586 Weddel Dr., Suite 1
Sunnyvale, CA 94089
(408) 745-0722

SOFTWARE MANUFACTURERS

Assisted Technologies (CUPL)
2381 Zanker Road, Suite 150
Santa Clara, CA 95050
(408) 942-8787

DATA I/O (ABEL)
10525 Willows Rd.
Redmond, WA 98052
(206) 881-6444

*Texas Instruments does not endorse or warrant the suppliers referenced.

Reference

1. H. Troy Nagle, Jr., B.D. Carroll, and David Irwin, *An Introduction to Computer Logic*. New Jersey: Prentice-Hall, Inc., 1975.