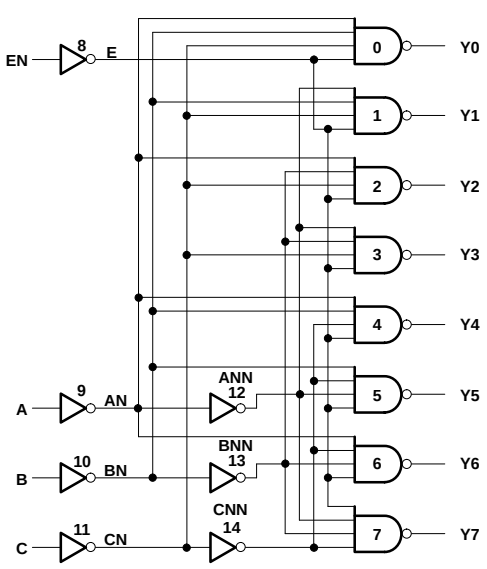
PLHS501 design examples AN049

DESIGN EXAMPLES

Most designers tend to view a PLD as a mechanism for collecting logical glue within a system. That is, those pieces which tie together the larger LSI microprocessors, controllers, RAMs, ROMs, UARTs, etc. However, there is a tendency of viewing a gate array as an entire system on a chip. PML based products will fit well in either casting as will be demonstrated by a series of small but straightforward examples. For starters, we shall examine how the fusing process embeds function, progress to glue-like decoding operations and finally demonstrate some coprocessor like functions as well as homemade “standard products”.

The method of associating gates within the NAND foldback structure is depicted in Figure 1 wherein a simple three to eight decoder is fused into the array. The corresponding inputs are on the left and outputs at the top. This figure shows inputs and their inverse formed in the array resulting in a solution that requires 6 inverting NANDs that would probably be best generated at the input receivers. Hence, this diagram could be trimmed by six gates, down to eight to achieve the function. Figure 2 shows two consecutive D flip-flop fusing images. Note that asynchronous sets and resets may be achieved for free, in this version. In both Figures 1 and 2 the gates are numbered in a one-to-one arrangement. As well, the accompanying equations are in the format used by Philips SNAP design software. For clarity, consider the gate labeled 2A in Figure 1. Schematically, this is shown as a 3 input NAND. However, in the fused depiction, it combines from three intermediate output points with the dot intersect designation. Hence, all gates are drawn as single input NANDs whose inputs span the complete NAND gate foldback structure.

1 OF 8 DECODER/DEMULITPLEXER




```
@LOGIC EQUATION
AN = /A;
BN = /B;
CN = /C;
ANN = /AN;
BNN = /BN;
CNN = /CN;
E = /EN;
Y0 = / (AN * BN * CN * E);
Y1 = / ANN * BN * CN * E);
Y2 = / AN * BNN * CN * E);
Y3 = / ANN * BNN * CN * E);
Y4 = / AN * BN * CNN * E);
Y5 = / ANN * BN * CNN * E);
Y6 = / AN * BNN * CNN * E);
Y7 = / ANN * BNN * CNN * E);
```

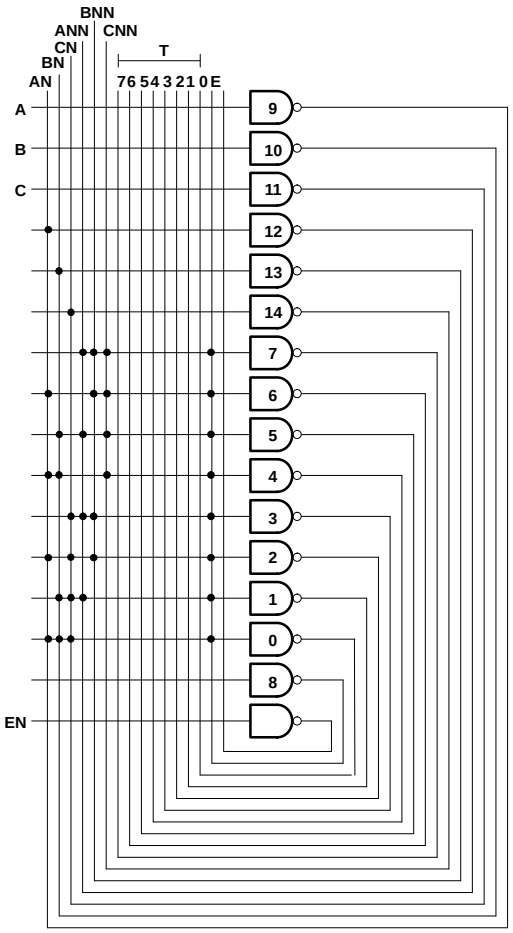
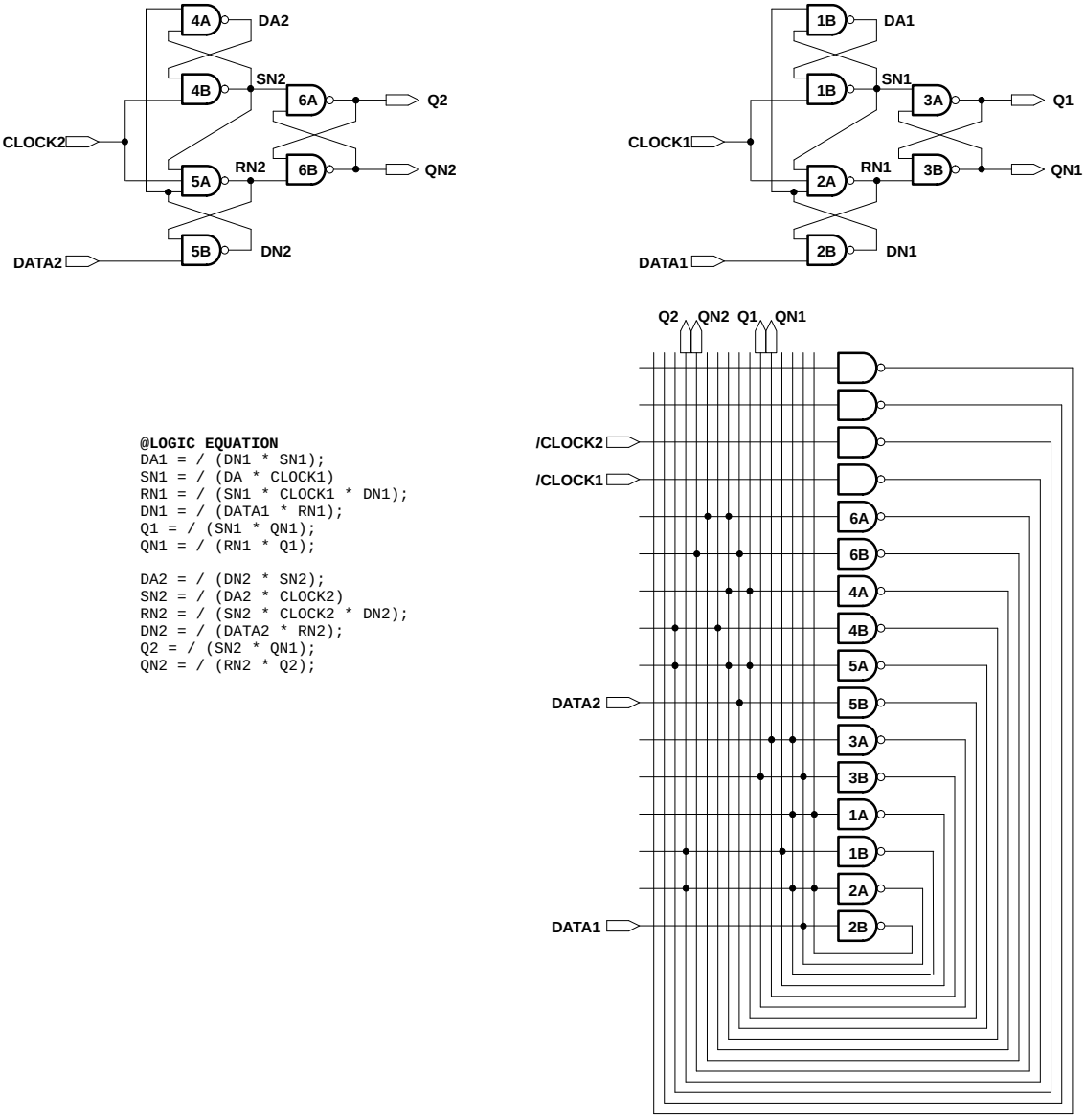


Figure 1. Decoder Implementation in NAND Foldback Structure

PLHS501 design examples AN049




```
@LOGIC EQUATION
DA1 = / (DN1 * SN1);
SN1 = / (DA * CLOCK1)
RN1 = / (SN1 * CLOCK1 * DN1);
DN1 = / (DATA1 * RN1);
Q1 = / (SN1 * QN1);
QN1 = / (RN1 * Q1);

DA2 = / (DN2 * SN2);
SN2 = / (DA2 * CLOCK2)
RN2 = / (SN2 * CLOCK2 * DN2);
DN2 = / (DATA2 * RN2);
Q2 = / (SN2 * QN1);
QN2 = / (RN2 * Q2);
```

Figure 2. Two Flip-Flops Implemented in the NAND Foldback Strucutre

One straightforward example of using a PLHS501 is shown in Figure 3. Here, the device is configured to accept the 23 upper address lines generated by a 68000 microprocessor. By selecting the direct and complemented variables, at least 16 distinct address selections can be made using only the dedicated outputs. The designer can combine additional VME bus strobes, or other control signals to qualify the decode or, define 8 additional outputs for expanded selection. As well, the designer could transform the bidirectionals to inputs and decode over a 32 bit space, selecting combinations off of a 32 bit wide address bus. Because this simple level of design requires only NAND output terms plus 4 NAND gates in the foldback array (for inversion of signals connected to O3.O0), there may be as many as 68 remaining gates to accomplish additional handshaking or logical operations on the input variables.

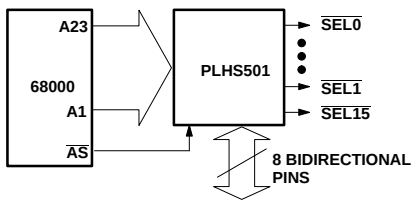


Figure 3. 68000 Microprocessor Address Decode

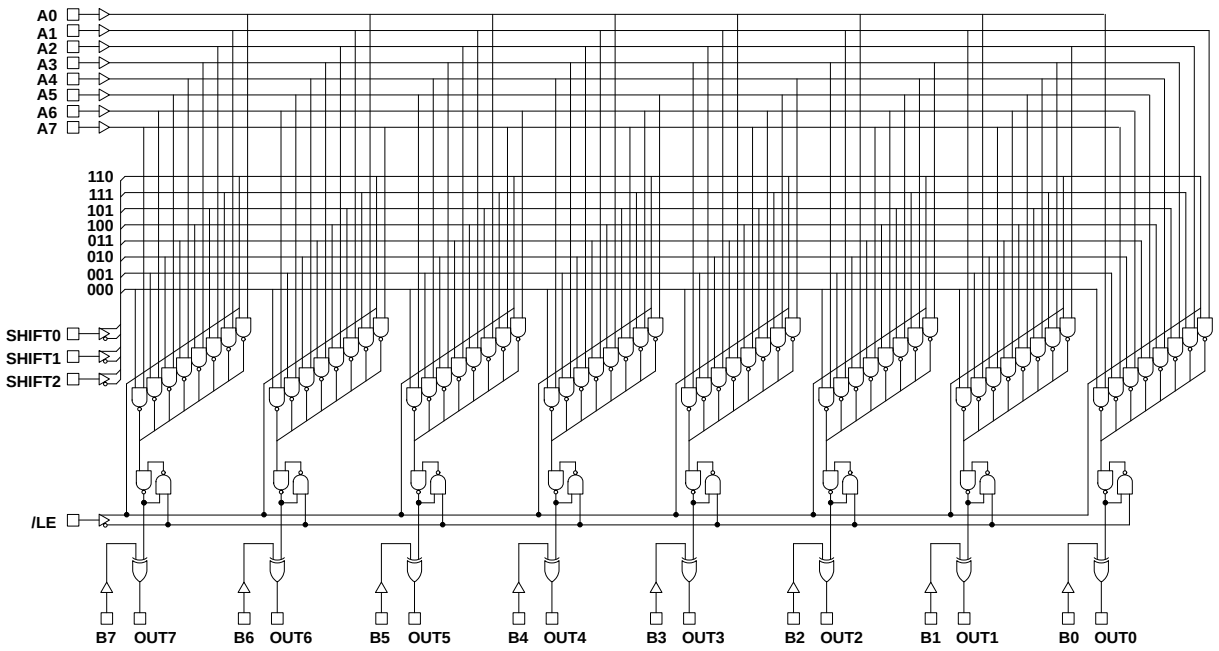


Figure 4. 8-Bit Barrel Shifter Implemented with the PLHS501

An eight bit barrel shifter exploits most of the PLHS501 as depicted in Figure 4. This implementation utilizes all 72 internal foldback NANDs in a relatively brute force configuration as well as 8 output NANDs to generate transparent latched and shifted results. The shift position here is generated by the shift 0, shift 1 and shift 2 inputs which are distinguished and selected from the input cells. Variations on this idea of data manipulation could include direct passing data, mirror imaged data (bit reversal) or byte swapping to name a few.

Part of an eight bit, look-ahead parallel adder is shown in Figure 5. Gates necessary to form the level-0 generate and propagate, as well as the XOR output gates generating the resulting sum are not shown. The reader should be aware that this solution exploits four layers of pyramided gates and only utilizes a total of about 58 gates. Additional comparison or Boolean operations could still be generated with remaining NAND functions to achieve additional arithmetic operations. This application should make the reader aware of a new class of applications achievable with third generation PLDs - user definable I/O coprocessors. The approach of increasing microprocessor performance by designing dedicated task coprocessors is now within the grasp of user definable single chip solutions.

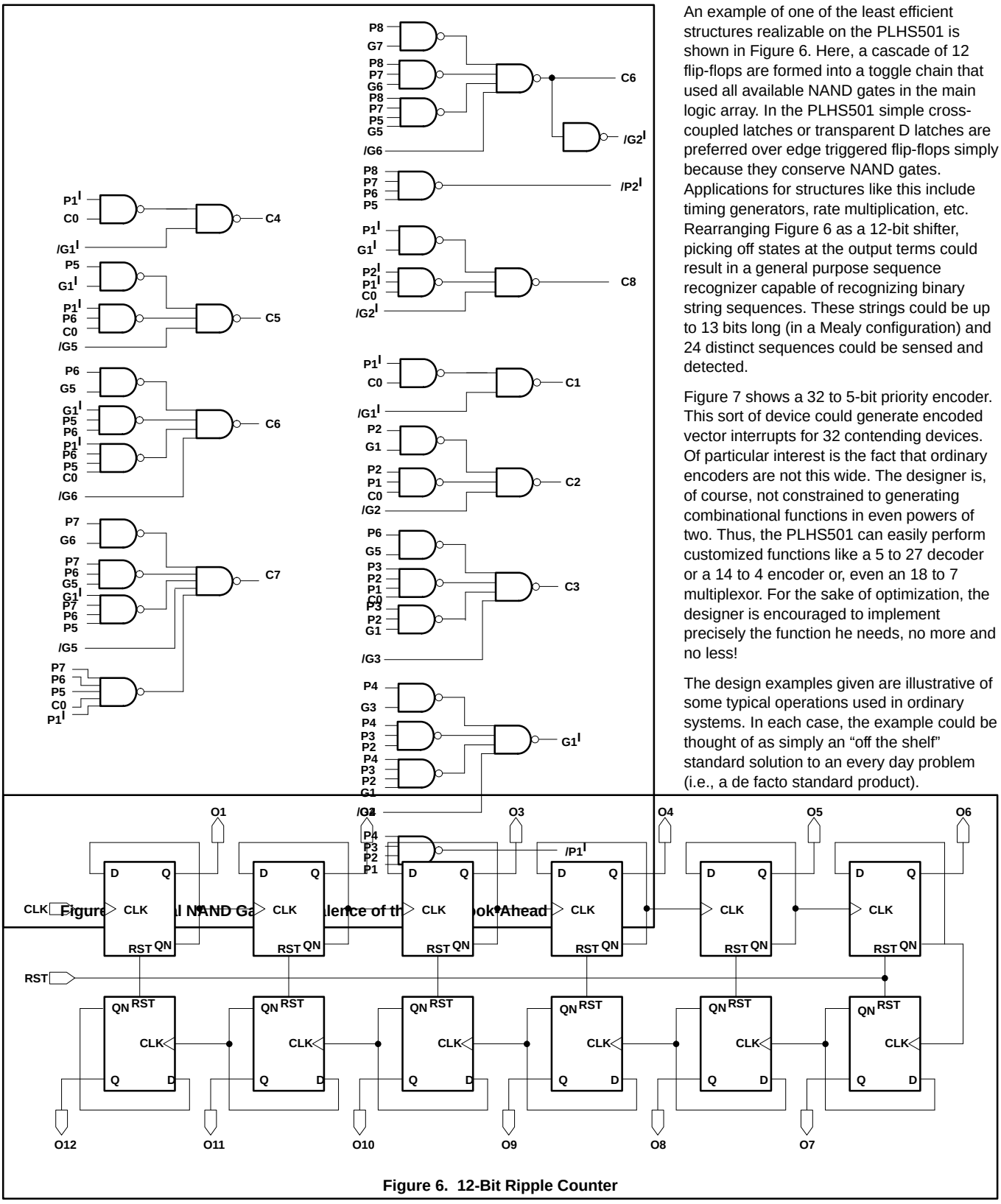


Figure 6. 12-Bit Ripple Counter

An example of one of the least efficient structures realizable on the PLHS501 is shown in Figure 6. Here, a cascade of 12 flip-flops are formed into a toggle chain that used all available NAND gates in the main logic array. In the PLHS501 simple cross-coupled latches or transparent D latches are preferred over edge triggered flip-flops simply because they conserve NAND gates. Applications for structures like this include timing generators, rate multiplication, etc. Rearranging Figure 6 as a 12-bit shifter, picking off states at the output terms could result in a general purpose sequence recognizer capable of recognizing binary string sequences. These strings could be up to 13 bits long (in a Mealy configuration) and 24 distinct sequences could be sensed and detected.

Figure 7 shows a 32 to 5-bit priority encoder. This sort of device could generate encoded vector interrupts for 32 contending devices. Of particular interest is the fact that ordinary encoders are not this wide. The designer is, of course, not constrained to generating combinational functions in even powers of two. Thus, the PLHS501 can easily perform customized functions like a 5 to 27 decoder or a 14 to 4 encoder or, even an 18 to 7 multiplexor. For the sake of optimization, the designer is encouraged to implement precisely the function he needs, no more and no less!

The design examples given are illustrative of some typical operations used in ordinary systems. In each case, the example could be thought of as simply an “off the shelf” standard solution to an every day problem (i.e., a de facto standard product).

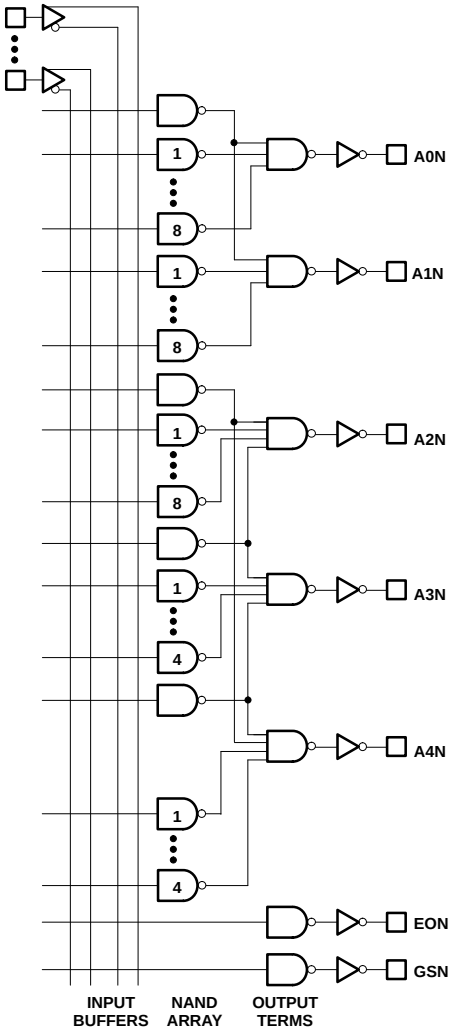


Figure 7. Encoder

DEVELOPMENT SUPPORT

SNAP

Because the architecture encourages deep functional nesting, a new support tool has been developed. Synthesis, Netlist, Analysis and Program (SNAP) software defines a gate array type development environment. SNAP permits several forms of design capture (schematic, Boolean equations, state equations, etc.), a gate array simulator with back annotation, waveform display and a complete fault analyzer and final fusemap compilation and model extraction. SNAP comes with a library of cells, and designs may be captured independently of the ultimate device that will implement the design. This permits the designer to migrate his design among a family of PML devices just as gate array designs can be moved to larger foundations when they do not route on smaller ones. Figure 8 shows the SNAP user interface “Shell” which dictates one sequence of operations to complete a design. Other sequences may be used.

The top portion of the shell depicts the paths available for design entry. Any design may be implemented in any one or a blend of all methods. For instance, a shift register might best be described schematically but a decoder by logic equations. These may be united with a multiplexor described by a text netlist as well. Ultimately, each form of input will be transformed to a function netlist and passed either to the simulation section or to the compiler section. Waveform entry is for simulation stimuli.

The simulator portion of SNAP is a 5-State gate array simulator with full timing information, setup and hold time checking, toggle and fault grade analysis and the ability to display in a wide range of formats, any set of nodes within the design. This permits a designer to zoom in with a synthetic logic state analyzer and view the behavior of any point in the design. Simulations can occur with unit delays, estimations or exact delays. The sequence of operations depicted in Figure 8 is entirely arbitrary, as many other paths exist.

It should be noted that the output of the “merger” block represents the composite design, but as yet is not associated to a PML device. This occurs in the compiler portion wherein association to the device occurs and a fusemap is compiled. This is analogous to placement and routing in a gate array environment. Because of the inter-connectibility of PML, this is not difficult. Once compiled, the exact assignment of pins, gates and flip-flops is known, so timing parameters may be associated and a new simulation model generated with exact detailed timing embedded. The design may be simulated very accurately at this point, and if correct, a part should be programmed.

To facilitate future migration to workstations, SNAP has been written largely in C. The internal design representation is EDIF (Electronic Design Interchange Format) compatible which permits straightforward porting to many commercially viable environments. SNAP currently utilizes OrCAD for schematic entry with eminent availability of FutureNet™ DASH.

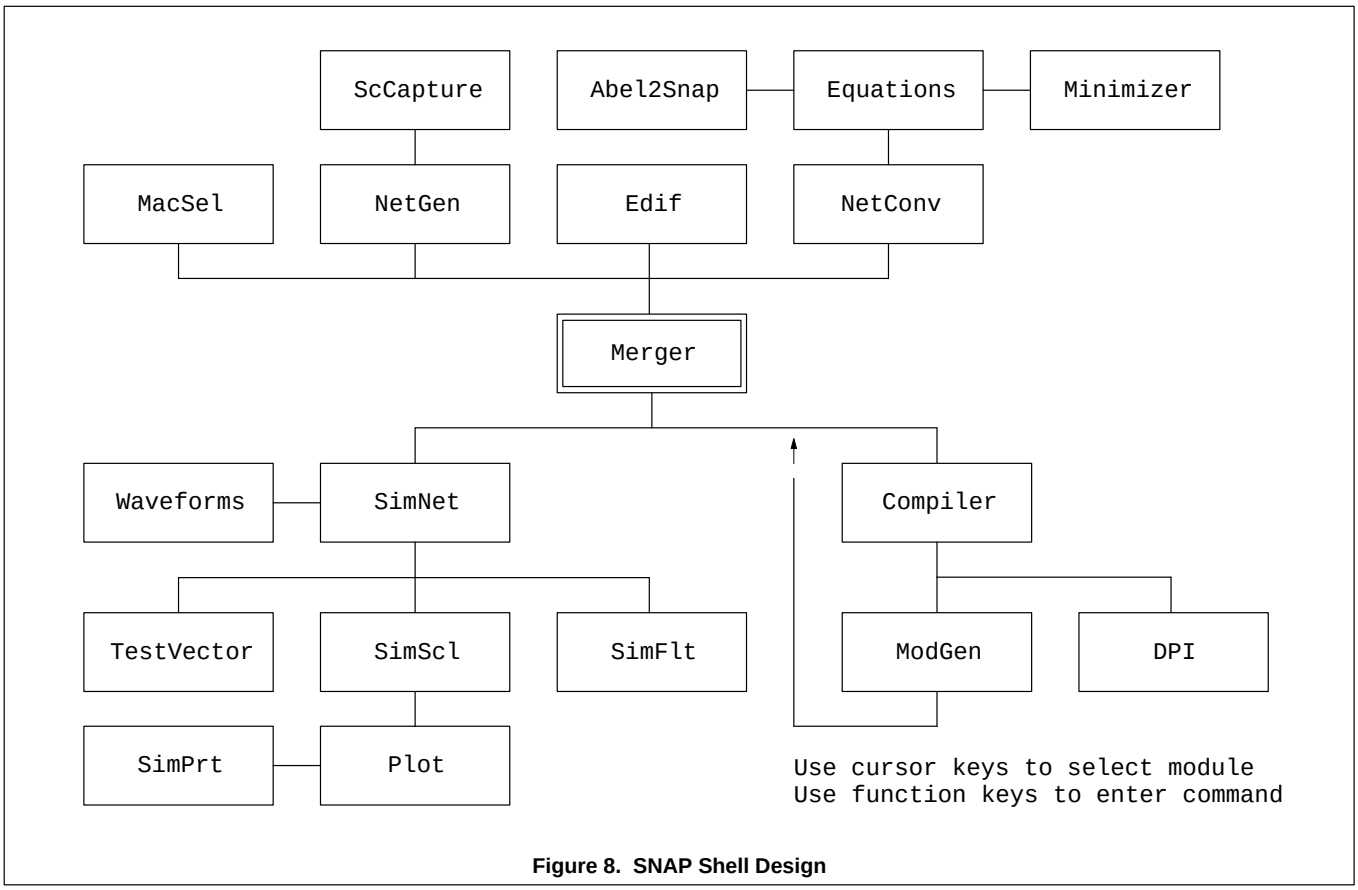


Figure 8. SNAP Shell Design

FutureNet and DASH are trademarks of DATA I/O

PLHS501 EXAMPLES USING SNAP

- 8-bit barrel shifter
- 12-bit comparator with dual 1 of 8 decoders
- 8-bit carry look-ahead adder
- 32 to 5 priority encoder
- 4-bit synchronous counter
- VME EPROM interface
- Microchannel interface
- NuBus interface
- Data bus parity generator
- 16-bit comparator

Following are example applications for the PLHS501 using SNAP. They should not be viewed as showing all possible capabilities of the device. They have been designed to demonstrate some of the PLHS501 features, syntax of SNAP, and to give the reader some ideas for possible circuit implementations.

Note that these examples were written using SNAP Rev. 1.90. Although Philips will try to keep succeeding versions of SNAP compatible, it may be necessary to change some syntax rules. Therefore, please refer to your SNAP manual for any notes on differences, if using a revision later than Rev. 1.90.

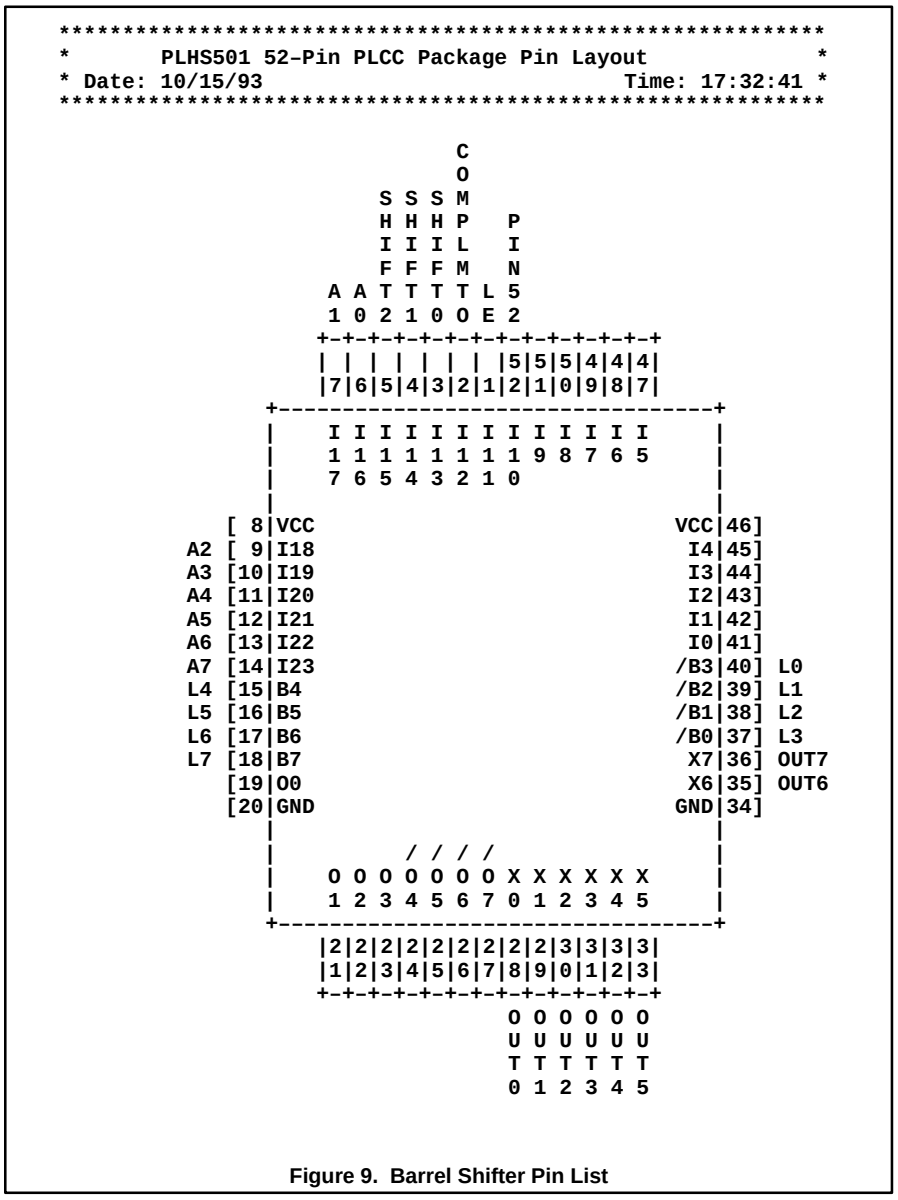


Figure 9. Barrel Shifter Pin List

8-BIT BARREL SHIFTER

This 8-bit shifter will shift to the right, data applied to A7 – A9 with the result appearing on OUT7 – OUT0. Data may be shifted by 1 to 7 places by indicating the desired binary count on pins SHIFT2 – SHIFT0. Data applied to the OUT0 position for a shift of 1. For a shift of 0, A7 will appear on OUT7.

Also included is a transparent latch for the output bits. The input ‘COMPLMTO’ will invert all output bits simultaneously and input /OE will 3-State all outputs.

This design was done by using OrCAD’s SDT with SNAP. The top level drawing is shown in Figure 11. The PLHS501 has various output structures. For the best fit, it was necessary to alter the portion of the schematic connecting to pins 15 – 18 compared to pins 37–40. This is shown in Figures 12 and 13.

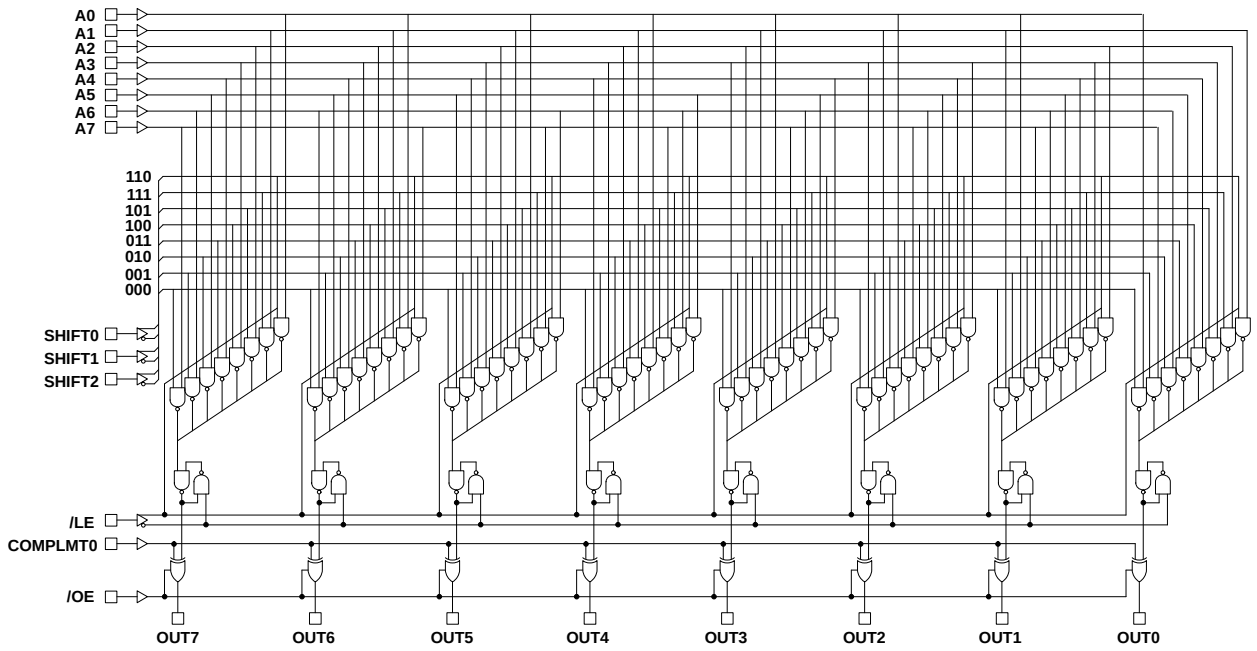


Figure 10. 8-Bit Barrel Shifter Schematic

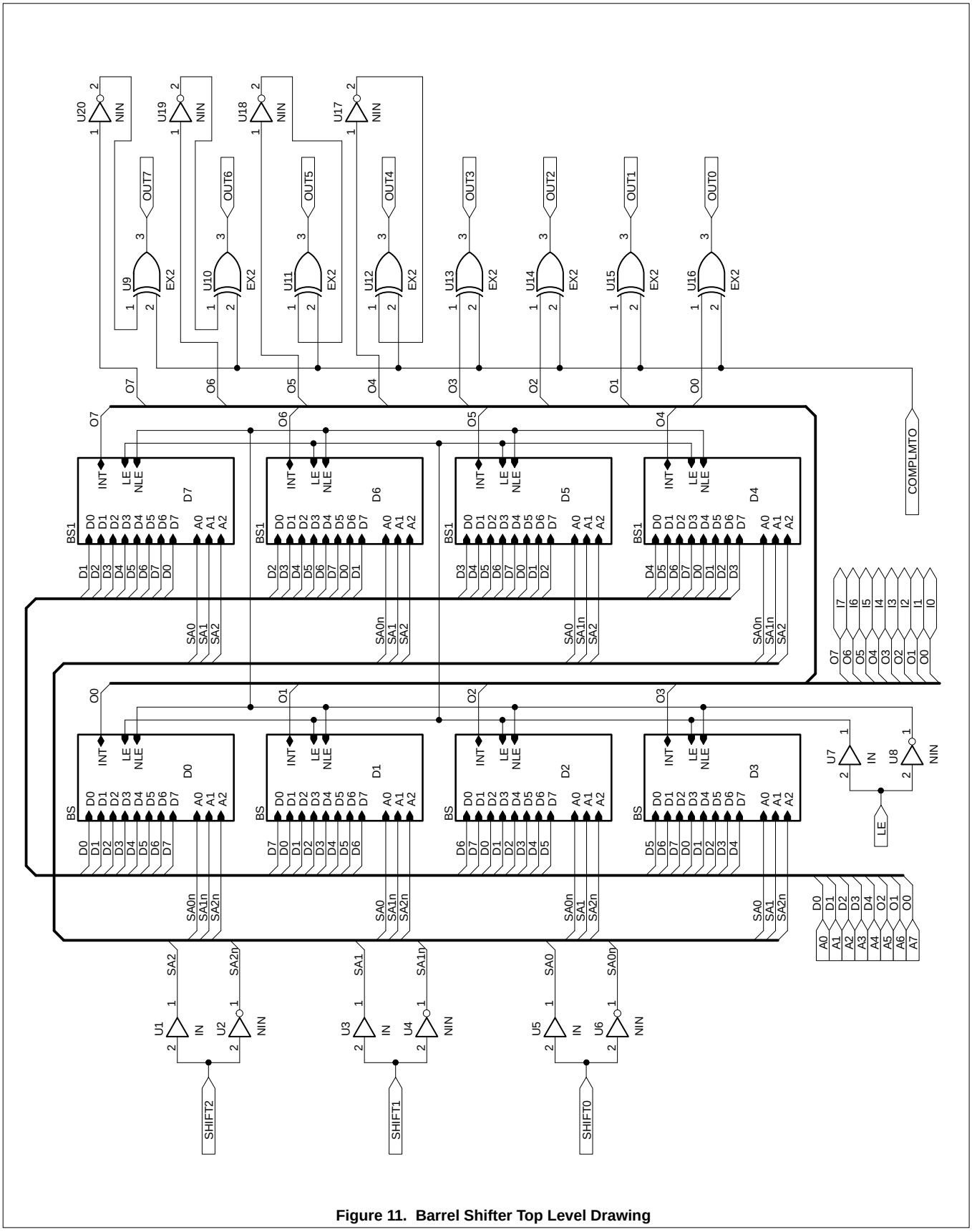


Figure 11. Barrel Shifter Top Level Drawing

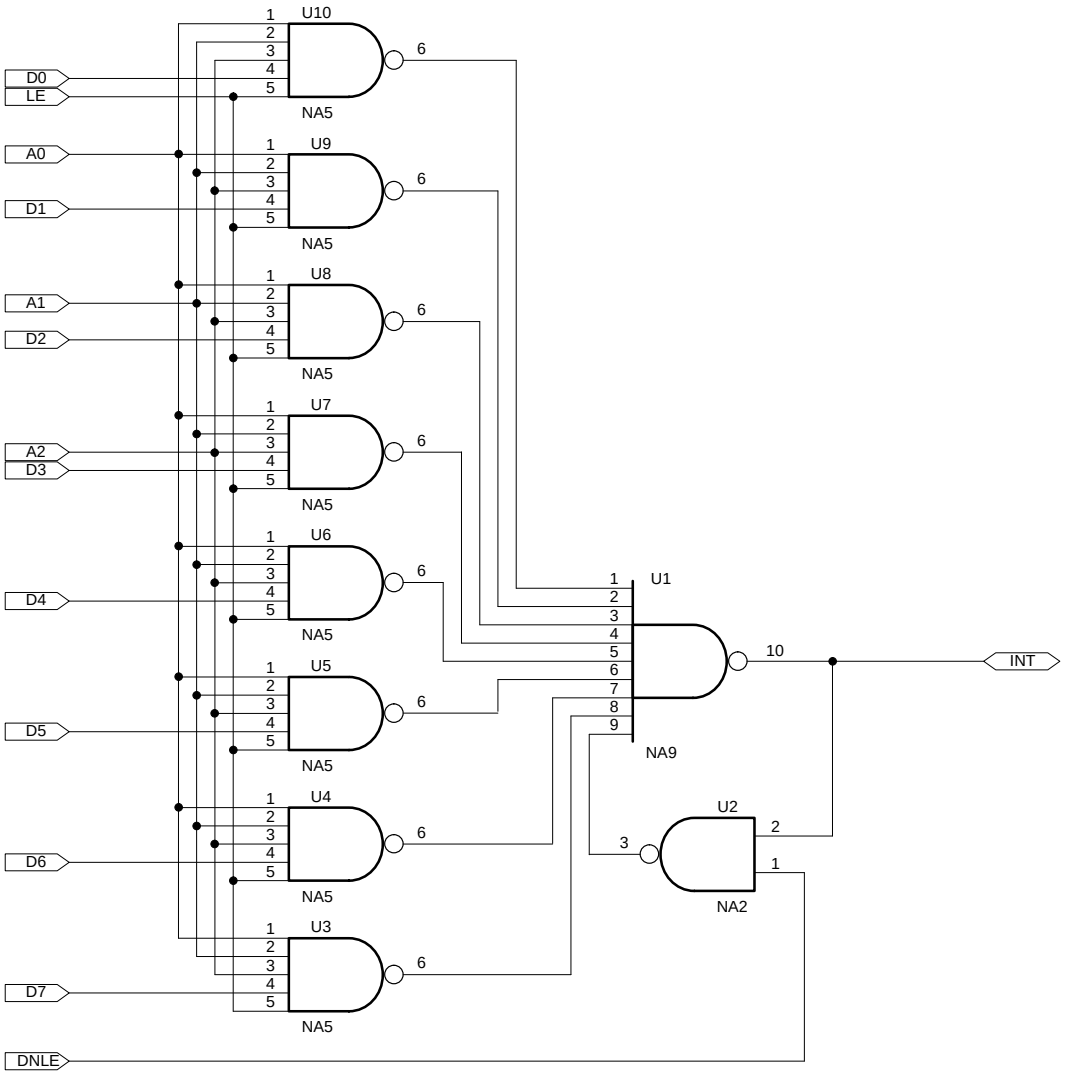


Figure 12. Portion of Shifter to Connect to NAND Output Pins

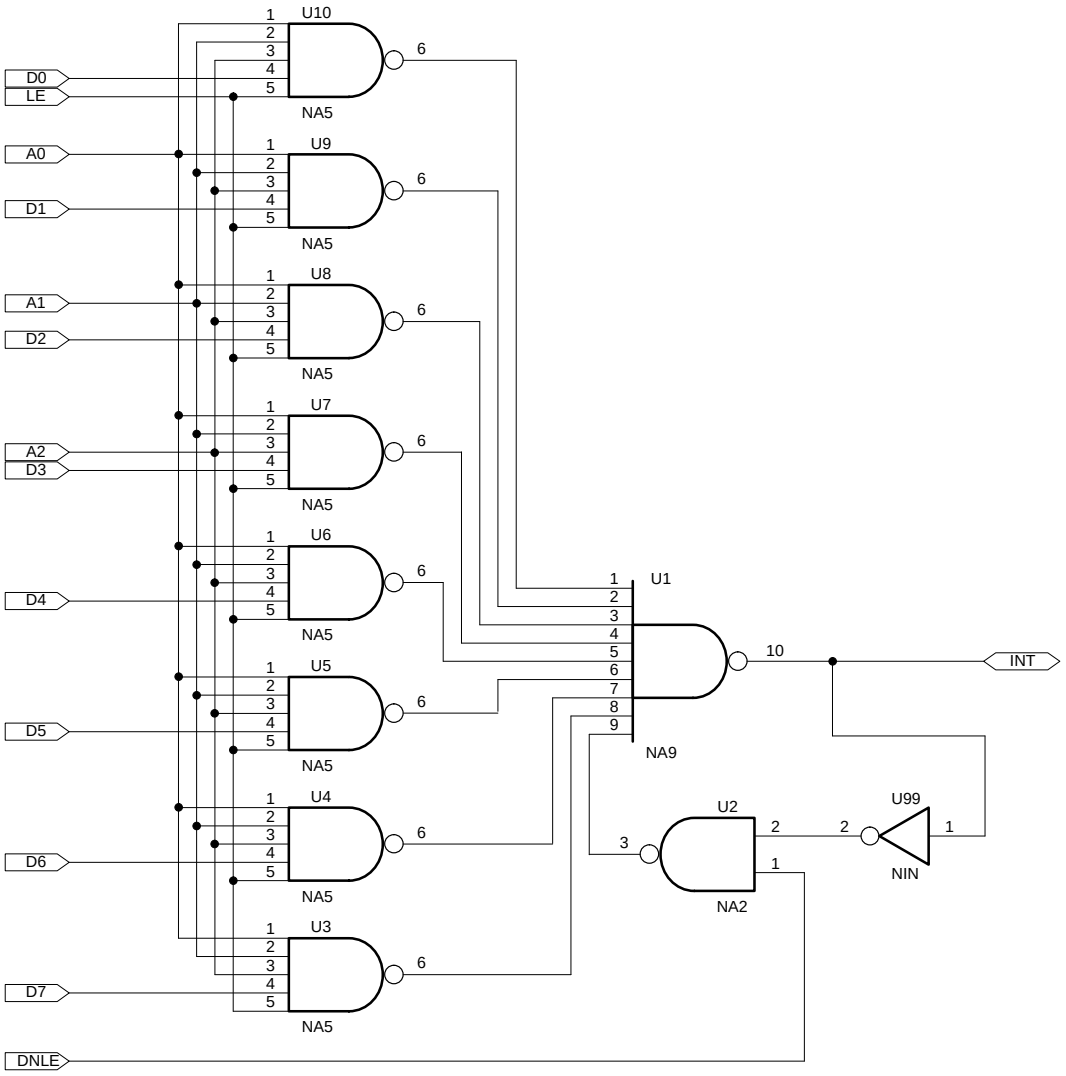


Figure 13. Portion of Shifter to Connect to AND Output Pins

```
*********************************************************
*       PLHS501 52-Pin PLCC Package Pin Layout          *
* Date: 10/15/93                        Time: 17:44:04 *
*********************************************************

                        A A
                    A A A 1 1 A A B B B B B B
                    4 3 2 1 0 1 0 9 8 7 6 5 4
                   +-+-+-+-+-+-+-+-+-+-+-+-+-+
                   | | | | | | |5|5|5|4|4|4|
                   |7|6|5|4|3|2|1|2|1|0|9|8|7|
               +-----------------------------------+
               |    I I I I I I I I I I I I I      |
               |    1 1 1 1 1 1 1 1 9 8 7 6 5      |
               |    7 6 5 4 3 2 1 0                |
               |                                   |
          [ 8|VCC                             VCC|46]
       A5 [ 9|I18                              I4|45] B3
       A6 [10|I19                              I3|44] B2
       A7 [11|I20                              I2|43] B11
       A8 [12|I21                              I1|42] B10
       A9 [13|I22                              I0|41] B1
       B0 [14|I23                             /B3|40] CMPOUT
          [15|B4                              /B2|39] DA2
   ENCOMP [16|B5                              /B1|38] DA1
   DCDREN [17|B6                              /B0|37] DA0
       RW [18|B7                               X7|36] R7
       W0 [19|O0                               X6|35] R6
          [20|GND                             GND|34]
               |                                   |
               |          / / / /                  |
               |    O O O O O O O X X X X X X      |
               |    1 2 3 4 5 6 7 0 1 2 3 4 5      |
               +-----------------------------------+
                   |2|2|2|2|2|2|2|2|2|3|3|3|3|
                   |1|2|3|4|5|6|7|8|9|0|1|2|3|
                   +-+-+-+-+-+-+-+-+-+-+-+-+-+
                    W W W W W W W R R R R R R
                    1 2 3 4 5 6 7 0 1 2 3 4 5
```

Figure 14. 12-Bit Comparator Pin List

12-BIT COMPARATOR WITH DUAL 1-OF-8 DECODERS

Two functions that are very often associated with controlling I/O parts are address comparison and address decoding. In this example, both functions are programmed into a PLHS501 using 52 out of the 72 foldback NAND terms.

The comparator compares 12 bits on inputs A11 – A0 to inputs B11 – B0 when the input 'ENCMP' is High. Output 'CMPOUT' will become Active-Low when all 12 bits of the A input match the B. Selection between the two decoders is done with input 'R/W'. Only one output may be active (Low) at a time. Although currently separate functions, the decoder enable may be derived internally from 'CMPOUT' freeing 2 bidirectional pins which together with available foldback NAND terms, may be used to incorporate a third function.

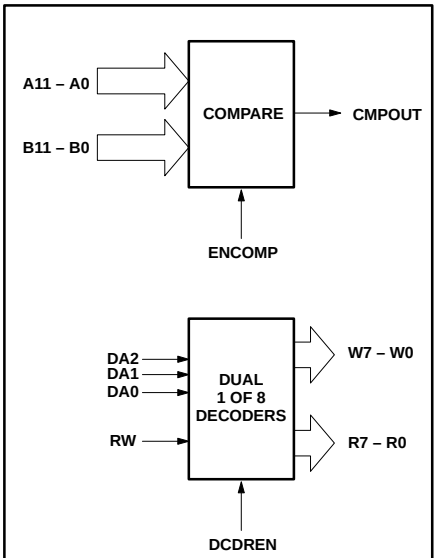


Figure 15. 12-Bit Comparator with Dual 1 – 8 Decoders Block Diagram

```
"FILENAME: CMP12BIT.EQN"
"          12–bit address comparator and dual 1 of 8 decoders"
@PINLIST
B0      I;
B1      I;
B2      I;
B3      I;
B4      I;
B5      I;
B6      I;
B7      I;
B8      I;
B9      I;
B10     I;
B11     I;
A0      I;
A1      I;
A2      I;
A3      I;
A4      I;
A5      I;
A6      I;
A7      I;
A8      I;
A9      I;
A10     I;
A11     I;
DA0     I;
DA1     I;
DA2     I;
RW      I;
DCDREN  I;
ENCOMP  I;
W0      O;
W1      O;
W2      O;
W3      O;
W4      O;
W5      O;
W6      O;
W7      O;
R0      O;
R1      O;
R2      O;
R3      O;
R4      O;
R5      O;
R6      O;
R7      O;
CMPOUT  O;

@LOGIC EQUATION

"COMMON PRODUCT TERM"
ad0=/da2*/da1*/da0*dcdren;
ad1=/da2*/da1* da0*dcdren;
ad2=/da2* da1*/da0*dcdren;
ad3=/da2* da1* da0*dcdren;
ad4= da2*/da1*/da0*dcdren;
ad5= da2*/da1* da0*dcdren;
ad6= da2* da1*/da0*dcdren;
ad7= da2* da1* da0*dcdren;
```

Figure 16. 12-Bit Comparator Boolean Equations (1 of 2)

```
"12–Bit Address Comparator"

axb0 = a0*/b0 + /a0*b0;
axb1 = a1*/b1 + /a1*b1;
axb2 = a2*/b2 + /a2*b2;
axb3 = a3*/b3 + /a3*b3;
axb4 = a4*/b4 + /a4*b4;
axb5 = a5*/b5 + /a5*b5;
axb6 = a6*/b6 + /a6*b6;
axb7 = a7*/b7 + /a7*b7;
axb8 = a8*/b8 + /a8*b8;
axb9 = a9*/b9 + /a9*b9;
axb10 = a10*/b10 + /a10*b10;
axb11 = a11*/b11 + /a11*b11;

cmpout = /(/axb0*/axb1*/axb2*/axb3*/axb4*/axb5*/axb6*/axb7*/axb8*/axb9*
          /axb10*/axb11*encomp);

"Dual 1 of 8 decoders

–  da2–da0 are address inputs
–  dcdren is an enable input
–  rw selects which group of 8 outputs r7–r0 or w7–w0
      will have the decoded active low output"

w7 = /(ad7*/rw);
w6 = /(ad6*/rw);
w5 = /(ad5*/rw);
w4 = /(ad4*/rw);
w3 = /(ad3*/rw);
w2 = /(ad2*/rw);
w1 = /(ad1*/rw);
w0 = /(ad0*/rw);

r7 = /(ad7* rw);
r6 = /(ad6* rw);
r5 = /(ad5* rw);
r4 = /(ad4* rw);
r3 = /(ad3* rw);
r2 = /(ad2* rw);
r1 = /(ad1* rw);
r0 = /(ad0* rw);
```

Figure 16. 12-Bit Comparator Boolean Equations (2 of 2)

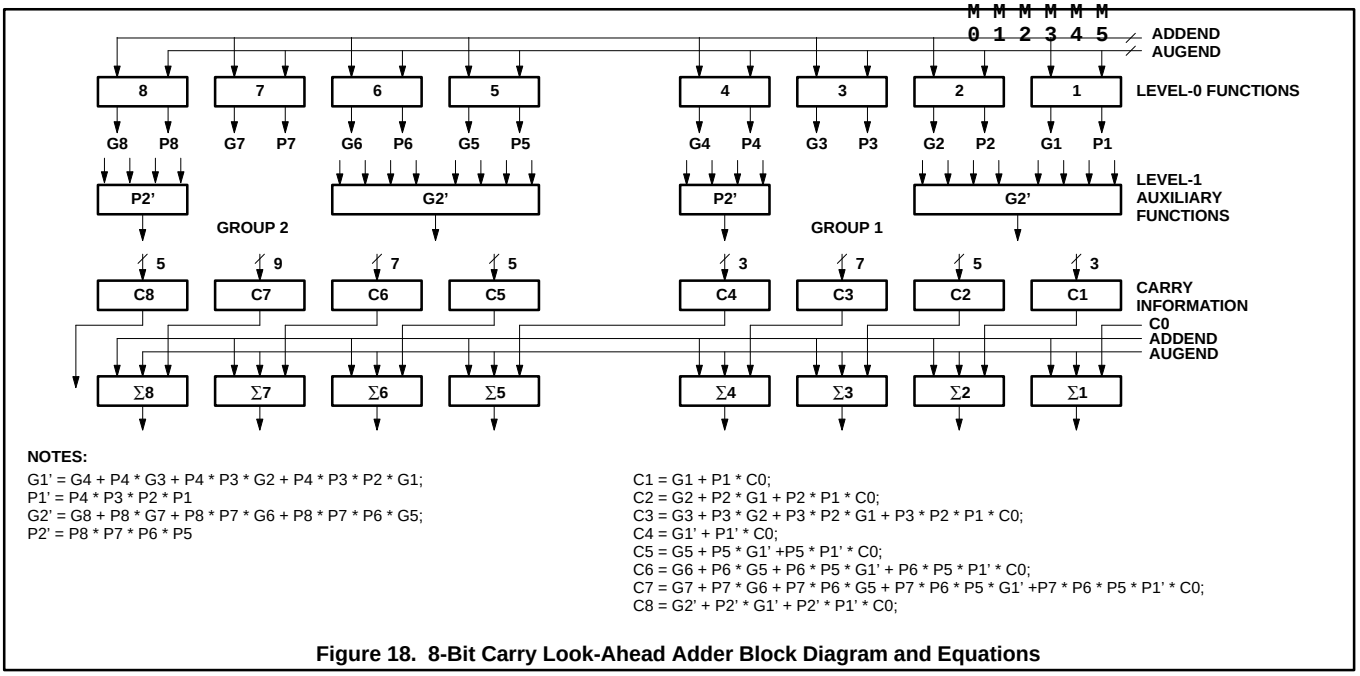
8-BIT CARRY LOOK-AHEAD ADDER

This function may be used as part of an ALU design or simply to off-load a microprocessor. Figure 18 is a block diagram showing the individual components needed for each bit.

A carry input (C0) is provided along with a carry output (C8). The result of an addition between the inputs A7 – A0 and B7 – B0 occurs on outputs SUM7 – SUM0.

```
*********************************************************
*
*       PLHS501 52-Pin PLCC Package Pin Layout
*
* Date: 10/15/93                          Time: 17:50:08
*
*********************************************************
*
                      A A A A A A A
                      6 5 4 3 2 1 0
                     +-+-+-+-+-+-+-+-+-+-+-+-+-+
                     | | | | | | |5|5|5|4|4|4|
                     |7|6|5|4|3|2|1|2|1|0|9|8|7|
                   +-------------------------------+
                   |    I I I I I I I I I I I I I   |
                   |    1 1 1 1 1 1 1 1 9 8 7 6 5   |
                   |    7 6 5 4 3 2 1 0             |
                   |                                |
                [ 8|VCC                          VCC|46]
           A7   [ 9|I18                           I4|45]
           B0   [10|I19                           I3|44] C0
           B1   [11|I20                           I2|43] B7
           B2   [12|I21                           I1|42] B6
           B3   [13|I22                           I0|41] B5
           B4   [14|I23                          /B3|40]
                [15|B4                           /B2|39]
                [16|B5                           /B1|38]
                [17|B6                           /B0|37]
                [18|B7                            X7|36]
SUM7
           C8   [19|O0                            X6|35]
SUM6
                [20|GND                          GND|34]
                   |                               |
                   |          / / / /              |
                   |    O O O O O O O X X X X X X  |
                   |    1 2 3 4 5 6 7 0 1 2 3 4 5  |
                   +-------------------------------+
                       |2|2|2|2|2|2|2|2|2|3|3|3|3|
                       |1|2|3|4|5|6|7|8|9|0|1|2|3|
                       +-+-+-+-+-+-+-+-+-+-+-+-+-+
                                                 S S S S S S
                                                 U U U U U U
                                                 M M M M M M
                                                 0 1 2 3 4 5
```

Figure 17. 8-Bit Adder Pin List



NOTES:

G1' = G4 + P4 * G3 + P4 * P3 * G2 + P4 * P3 * P2 * G1;
P1' = P4 * P3 * P2 * P1
G2' = G8 + P8 * G7 + P8 * P7 * G6 + P8 * P7 * P6 * G5;
P2' = P8 * P7 * P6 * P5

C1 = G1 + P1 * C0;
C2 = G2 + P2 * G1 + P2 * P1 * C0;
C3 = G3 + P3 * G2 + P3 * P2 * G1 + P3 * P2 * P1 * C0;
C4 = G1' + P1' * C0;
C5 = G5 + P5 * G1' +P5 * P1' * C0;
C6 = G6 + P6 * G5 + P6 * P5 * G1' + P6 * P5 * P1' * C0;
C7 = G7 + P7 * G6 + P7 * P6 * G5 + P7 * P6 * P5 * G1' +P7 * P6 * P5 * P1' * C0;
C8 = G2' + P2' * G1' + P2' * P1' * C0;

Figure 18. 8-Bit Carry Look-Ahead Adder Block Diagram and Equations

```
"FILENAME: ADDR8BIT.EQN
           8 Bit Carry Look-Ahead Adder"

@PINLIST
A0      I;
A1      I;
A2      I;
A3      I;
A4      I;
A5      I;
A6      I;
A7      I;
B0      I;
B1      I;
B2      I;
B3      I;
B4      I;
B5      I;
B6      I;
B7      I;
C0      I;

C8      O;
SUM0    O;
SUM1    O;
SUM2    O;
SUM3    O;
SUM4    O;
SUM5    O;
SUM6    O;
SUM7    O;

@LOGIC EQUATION

"level-0 functions"

gn1 = /(a0*b0);
p1 = /(/a0*/b0);
g1 = /gn1;

gn2 = /(a1*b1);
p2 = /(/a1*/b1);
g2 = /gn2;

gn3 = /(a2*b2);
p3 = /(/a2*/b2);
g3 = /gn3;

gn4 = /(a3*b3);
p4 = /(/a3*/b3);
g4 = /gn4;

gn5 = /(a4*b4);
p5 = /(/a4*/b4);
g5 = /gn5;

gn6 = /(a5*b5);
p6 = /(/a5*/b5);
g6 = /gn6;

gn7 = /(a6*b6);
p7 = /(/a6*/b6);
g7 = /gn7;

gn8 = /(a7*b7);
p8 = /(/a7*/b7);
g8 = /gn8;

"level-1 functions"

g1_1 = g4 + p4*g3 + p4*p3*g2 + p4*p3*p2*g1;
g2_1 = g8 + p8*g7 + p8*p7*g6 + p8*p7*p6*g5;
```

Figure 19. 8-Bit Adder Boolean Equations (1 of 2)

```
"carry information"

c1 = g1   + p1*c0;
c2 = g2   + p2*g1   + p2*p1*c0;
c3 = g3   + p3*g2   + p3*p2*g1   + p3*p2*p1*c0;
c4 = g1_1 + p4*p3*p2*p1*c0;
c5 = g5   + p5*g1_1 + p5*p4*p3*p2*p1*c0;
c6 = g6   + p6*g5   + p6*p5*g1_1 + p6*p5*p4*p3*p2*p1*c0;
c7 = g7   + p7*g6   + p7*p6*g5   + p7*p6*p5*g1_1 + p7*p6*p5*p4*p3*p2*p1*c0;
c8 = g2_1 + p8*p7*p6*p5*g1_1     + p8*p7*p6*p5*p4*p3*p2*p1*c0;

"addition functions"

sum0 = c0 :+: (p1 * gn1);
sum1 = c1 :+: (p2 * gn2);
sum2 = c2 :+: (p3 * gn3);
sum3 = c3 :+: (p4 * gn4);
sum4 = c4 :+: (p5 * gn5);
sum5 = c5 :+: (p6 * gn6);
sum6 = c6 :+: (p7 * gn7);
sum7 = c7 :+: (p8 * gn8);
```

Figure 19. 8-Bit Adder Boolean Equations (2 of 2)

```
**********************************************************
*
*       PLHS501 52-Pin PLCC Package Pin Layout
*
* Date: 10/15/93                          Time: 17:58:06
*
**********************************************************
*
                    I I I I I I   I   I I I I
                    1 1 1 1 1 1 3 I 3 I 2 2 2 2
                    5 4 3 2 1 1 0 0 2 9 8 7 6
                   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                   | | | | | | | |5|5|5|4|4|4|
                   |7|6|5|4|3|2|1|2|1|0|9|8|7|
               +-----------------------------------+
               |    I I I I I I I I I I I I I      |
               |    1 1 1 1 1 1 1 1 9 8 7 6 5      |
               |    7 6 5 4 3 2 1 0                |
               |                                   |
           [ 8|VCC                           VCC|46]
       I16 [ 9|I18                            I4|45] I25
       I17 [10|I19                            I3|44] I24
       I18 [11|I20                            I2|43] I23
       I19 [12|I21                            I1|42] I22
        I1 [13|I22                            I0|41] I21
       I20 [14|I23                           /B3|40] I10
        I3 [15|B4                            /B2|39] I9
        I4 [16|B5                            /B1|38] I8
        I5 [17|B6                            /B0|37] I7
        I6 [18|B7                             X7|36]
        A0 [19|O0                             X6|35]
           [20|GND                           GND|34]
               |                                   |
               |          / / / /                  |
               |    O O O O O O O X X X X X X      |
               |    1 2 3 4 5 6 7 0 1 2 3 4 5      |
               +-----------------------------------+
                   |2|2|2|2|2|2|2|2|3|3|3|3|
                   |1|2|3|4|5|6|7|8|9|0|1|2|3|
                   +-+-+-+-+-+-+-+-+-+-+-+-+-+
                    A A A A       E S
                    1 2 3 4       O S
```

Figure 20. Encoder Pin List

32- to 5-BIT PRIORITY ENCODER

This relatively simple example demonstrates the capability of the PLHS501 to be programmed with functions that are not available in 'standard' device libraries. The equations may look difficult at first glance. However, there is a pattern to the encoding. Referring to Figure 21, Lab4 – Lab1 are terms that are common to several outputs (A4n – A0n). Separating them from the main equations allows a total reduction in the numbers of gates used.

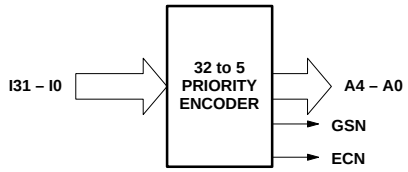


Figure 21. 32 to 5 Priority Encoder Block Diagram

```
”FILENAME: ENCODER.EQN
           32 TO 5 PRIORITY ENCODER”

@PINLIST
I0        I;
I1        I;
I2        I;
I3        I;
I4        I;
I5        I;
I6        I;
I7        I;
I8        I;
I9        I;
I10       I;
I11       I;
I12       I;
I13       I;
I14       I;
I15       I;
I16       I;
I17       I;
I18       I;
I19       I;
I20       I;
I21       I;
I22       I;
I23       I;
I24       I;
I25       I;
I26       I;
I27       I;
I28       I;
I29       I;
I30       I;
I31       I;
A0        O;
A1        O;
A2        O;
A3        O;
A4        O;
GS        O;
EO        O;

@LOGIC EQUATION

”COMMON PRODUCT TERM”
cpt1 = i26*i27*i28*i29*i30*i31;
cpt2 = i20*i21*i22*i23*i24*i25;
cpt3 = i14*i15*i16*i17*i18*i19;
cpt4 = i8*i9*i10*i11*i12*i13;

A0=/( /i31
     +/i29*i30*i31
     +/i27*i28*i29*i30*i31
     +/i25*cpt1
     +/i23*i24*i25*cpt1
     +/i21*i22*i23*i24*i25*cpt1
     +/i19*cpt2*cpt1
     +/i17*i18*i19*cpt2*cpt1
     +/i15*i16*i17*i18*i9*cpt2*cpt1
     +/i13*cpt3*cpt2*cpt1
     +/i11*i12*i13*cpt3*cpt2*cpt1
     +/i9 *i10*i11*i12*i13*cpt3*cpt2*cpt1
     +/i7 *cpt4*cpt3*cpt2
     +/i5 *i6*i7*cpt4*cpt3*cpt2*cpt1
     +/i3 *i4*i5*i6*i7*cpt4*cpt3*cpt2*cpt1
     +/i1 *i2*i3*i4*i5*i6*i7*cpt4*cpt3*cpt2*cpt1);

A1=/( /i31
     +/i30*i31
     +/i27*i28*i29*i30*i31
     +/i26*i27*i28*i29*i30*i31
```

Figure 22. Encoder Boolean Equations (1 of 2)

```
     +/i23*i24*i25*cpt1
     +/i22*i23*i24*i25*cpt1
     +/i19*cpt2*cpt1
     +/i18*i19*cpt2*cpt1
     +/i15*i16*i17*i18*i19*cpt2*cpt1
     +/i14*i15*i16*i17*i18*i19*cpt2*cpt1
     +/i11*i12*i13*cpt3*cpt2*cpt1
     +/i10*i11*i12*i13*cpt3*cpt2*cpt1
     +/i7 *cpt4*cpt3*cpt2*cpt1
     +/i6 *i7*cpt4*cpt3*cpt2*cpt1
     +/i3 *i4*i5*i6*i7*cpt4*cpt3*cpt2*cpt1
     +/i2 *i3*i4*i5*i6*i7*cpt4*cpt3*cpt2*cpt1);

A2=/( /i31
     +/i30*i31
     +/i29*i30*i31
     +/i28*i29*i30*i31
     +/i23*i24*i25*cpt1
     +/i22*i23*i24*i25*cpt1
     +/i21*i22*i23*i24*i25*cpt1
     +/i20*i21*i22*i23*i24*i25*cpt1
     +/i15*i16*i17*i18*i19*cpt2*cpt1
     +/i14*i15*i16*i17*i18*i19*cpt2*cpt1
     +/i13*cpt3*cpt2*cpt1
     +/i12*i13*cpt3*cpt2*cpt1
     +/i7 *cpt4*cpt3*cpt2*cpt1
     +/i6 *i7*cpt4*cpt3*cpt2*cpt1
     +/i5 *i6*i7*cpt4*cpt3*cpt2*cpt1
     +/i4 *i5*i6*i7*cpt4*cpt3*cpt2*cpt1);

A3=/( /i31
     +/i30*i31
     +/i29*i30*i31
     +/i28*i29*i30*i31
     +/i27*i28*i29*i30*i31
     +/i26*i27*i28*i29*i30*i31
     +/i25*cpt1
     +/i24*i25*cpt1
     +/i15*i16*i17*i18*i19*cpt2*cpt1
     +/i14*i15*i16*i17*i18*i19*cpt2*cpt1
     +/i13*cpt3*cpt2*cpt1
     +/i12*i13*cpt3*cpt2*cpt1
     +/i11*i12*i13*cpt3*cpt2*cpt1
     +/i10*i11*i12*i13*cpt3*cpt2*cpt1
     +/i9 *i10*i11*i12*i13*cpt3*cpt2*cpt1
     +/i8 *i9*i10*i11*i12*i13*cpt3*cpt2*cpt1);

A4=/( /i31
     +/i30*i31
     +/i29*i30*i31
     +/i28*i29*i30*i31
     +/i27*i28*i29*i30*i31
     +/i26*i27*i28*i29*i30*i31
     +/i25*cpt1
     +/i24*i25*cpt1
     +/i23*i24*i25*cpt1
     +/i22*i23*i24*i25*cpt1
     +/i21*i22*i23*i24*i25*cpt1
     +/i20*i21*i22*i23*i24*i25*cpt1
     +/i19*cpt2*cpt1
     +/i18*i19*cpt2*cpt1
     +/i17*i18*i19*cpt2*cpt1
     +/i16*i17*i18*i19*cpt2*cpt1);

eo  = /(i0*i1*i2*i3*i4*i5*i6*i7
        *i8*i9*i10*i11*i12*i13*i14*i15
        *i16*i17*i18*i19*i20*i21*i22*i23
        *i24*i25*cpt1);
gs = /eo;
```

Figure 22. Encoder Boolean Equations (2 of 2)

4-BIT SYNCHRONOUS COUNTER

This counter produces a binary count on outputs Count3 – Count0. Note the required reset (RST) input to initialize all of the flip-flops. The inputs for each flip-flop were first determined by drawing the desired output waveforms. Next, Karnaugh maps were used to reduce the number of terms and determine the logic equations for the input to each flip-flop. This technique could be used to construct a counter whose outputs produce some count other than binary.

The simulation only consists of a reset, followed by a number of clocks to count from 0 through 15 and back to 0.

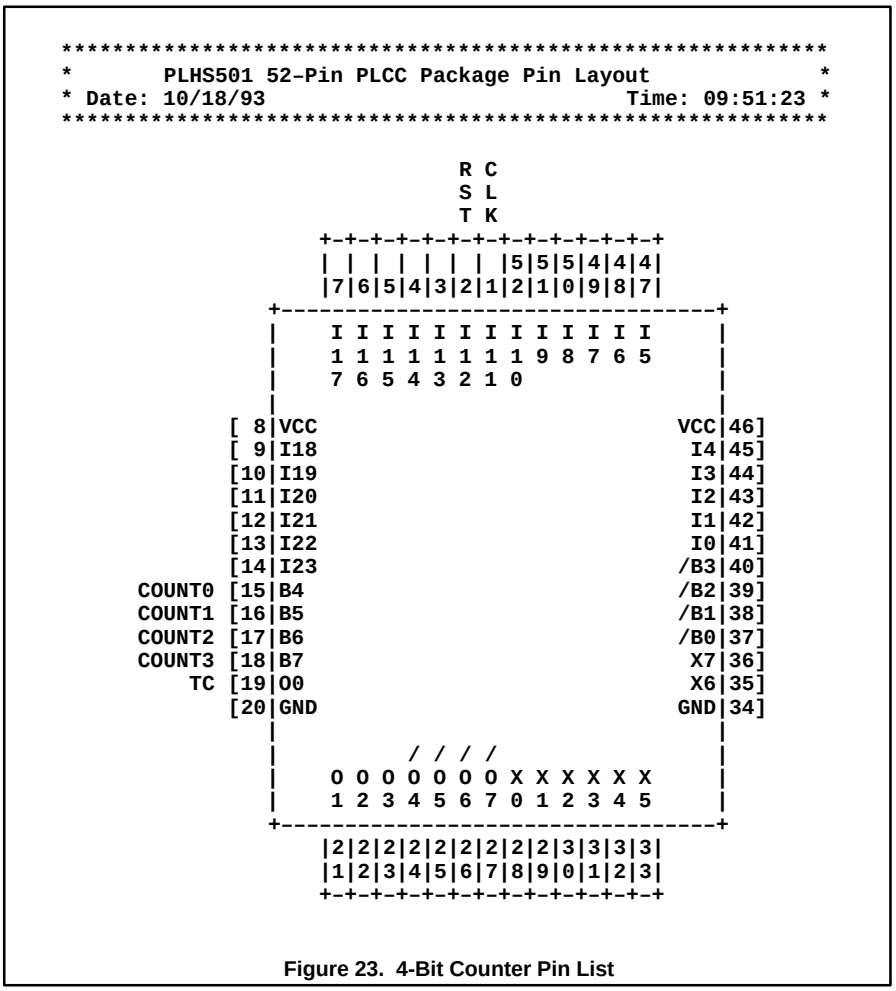


Figure 23. 4-Bit Counter Pin List

```
         ”4 bit synchronous counter”
@PINLIST
CLK     I;
RST     I;
COUNT0  O;
COUNT1  O;
COUNT2  O;
COUNT3  O;
TC      O;

@LOGIC EQUATION

”INPUTS FOR EACH FLIP–FLOP”

DATA1 = ((CQ1*CQN0)+(CQN1*CQ0));
DATA2 = ((CQ0*CQ1*CQN2)+(CQN0*CQ2)+(CQN1*CQ2));
DATA3 = ((CQN2*CQ3)+(CQN0*CQ3)+(CQ0*CQ1*CQ2*CQN3)+(CQN1*CQ3));

”4 D–TYPE FLIP FLOPS CONNECTED AS A SYNCHRONOUS COUNTER”

CSN0 = /(CLK*RST*(/(CSN0*(/(CQN0*RST*CRN0)))));
CRN0 = /(CSN0*CLK*(/(CQN0*RST*CRN0)));
CQ0  = /(CSN0*CQN0);
CQN0 = /(CRN0*CQ0*RST);

CSN1 = /(CLK*RST*(/(CSN1*(/(DATA1*RST*CRN1)))));
CRN1 = /(CSN1*CLK*(/(DATA1*RST*CRN1)));
CQ1  = /(CSN1*CQN1);
CQN1 = /(CRN1*CQ1*RST);

CSN2 = /(CLK*RST*(/(CSN2*(/(DATA2*RST*CRN2)))));
CRN2 = /(CSN2*CLK*(/(DATA2*RST*CRN2)));
CQ2  = /(CSN2*CQN2);
CQN2 = /(CRN2*CQ2*RST);

CSN3 = /(CLK*RST*(/(CSN3*(/(DATA3*RST*CRN3)))));
CRN3 = /(CSN3*CLK*(/(DATA3*RST*CRN3)));
CQ3  = /(CSN3*CQN3);
CQN3 = /(CRN3*CQ3*RST);

”Connection to output pins”

count0=cq0;
count1=cq1;
count2=cq2;
count3=cq3;

”TERMINAL COUNT PIN”

TC=(CQ0*CQ1*CQ2*CQ3);
```

Figure 24. 4-Bit Counter Boolean Equations

VME Bus EPROM Interface

The idea for this VMEbus EPROM board came from *WIRELESS WORLD CIRCUIT IDEAS*, January, 1988. The implementation was done by a Philips' FAE, John McNally.

The board contains two banks of EPROMs. Each bank consists of either two 27128s or two 27256s; each of which can be enabled by comparing the address location fo the board. Decoding three other address bits selects which of the banks is accessed. A 4-bit shift register combined with four jumpers provide wait states.

The circuit drawing was entered onto a PC using FutureNet DASH, a schematic capture package (Figures 25, 26, and 27). It was then converted to logic equations using SNAP (Figures 29 and 30) and then assembled into a PLHS501.

This application, which originally needed eight ICs, used forty-four of the available seventy-two NAND Foldback Terms and forth of the available fifty-two pins. As the PLHS501 contains no registers, an edge-triggered D-type flip-flop was designed using NAND gates and this is used as a soft macro in order to implement the shift register function (Figure 27).

As suggested in the original article, the circuit could be expanded to access up to eight ROM banks (Figure 28B). This was achieved by editing the logic equation file and adding extra equations (Figure 32). Modifying the drawing, although fairly easy to do, was not considered necessary as the object was to design with PML and not TTL. The expanded circuit would require another three TTL IC packages, brining the total to eleven if done using TTL devices. The number of foldback terms increased to fifty-five, with the number of pins rising to fifty. Figure 28 shows the pinout of both versions.

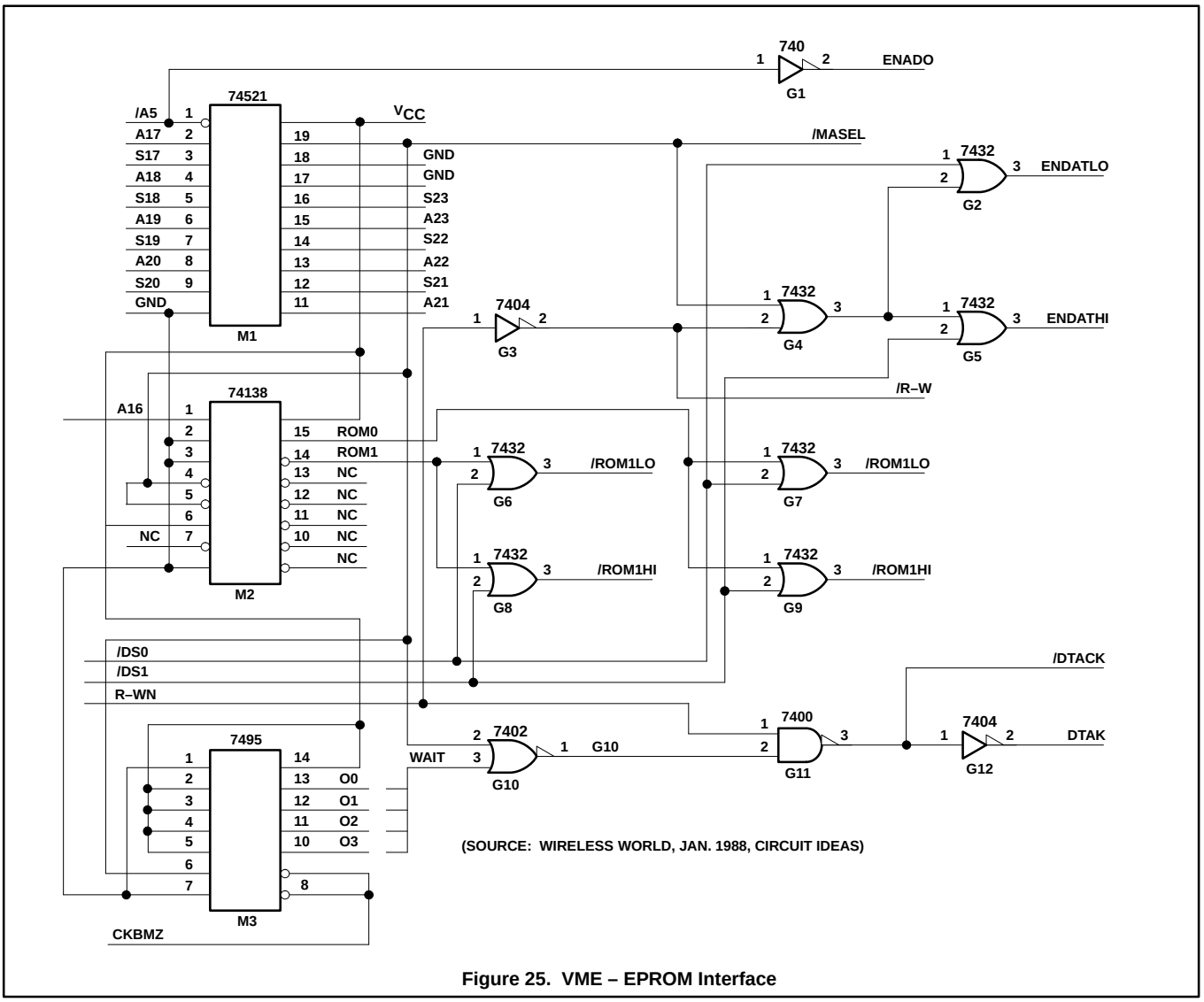


Figure 25. VME – EPROM Interface

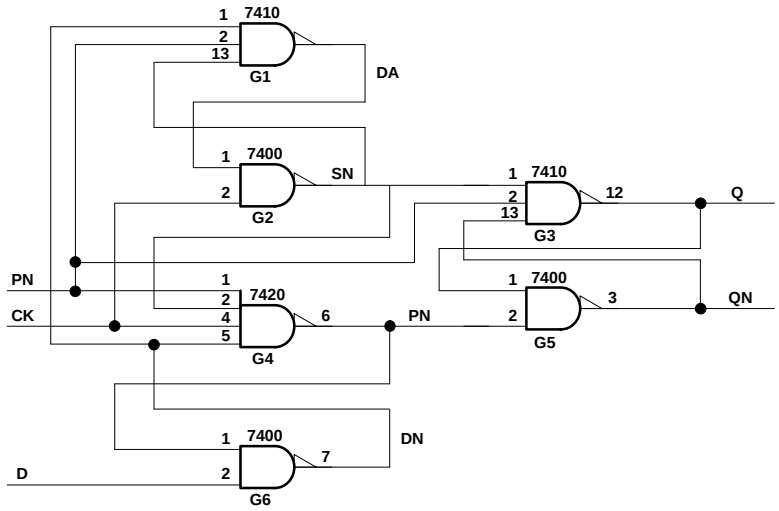


Figure 26. Edge-Triggered D Flip-Flop (DFFS)

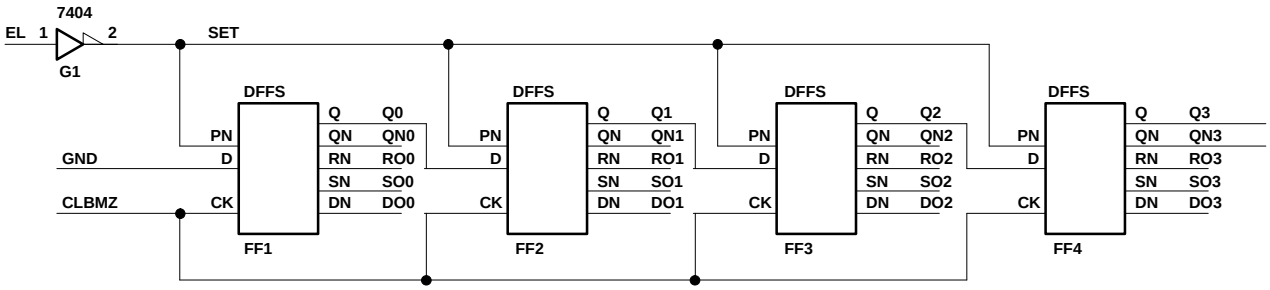


Figure 27. 4-Bit Shifter (7495)

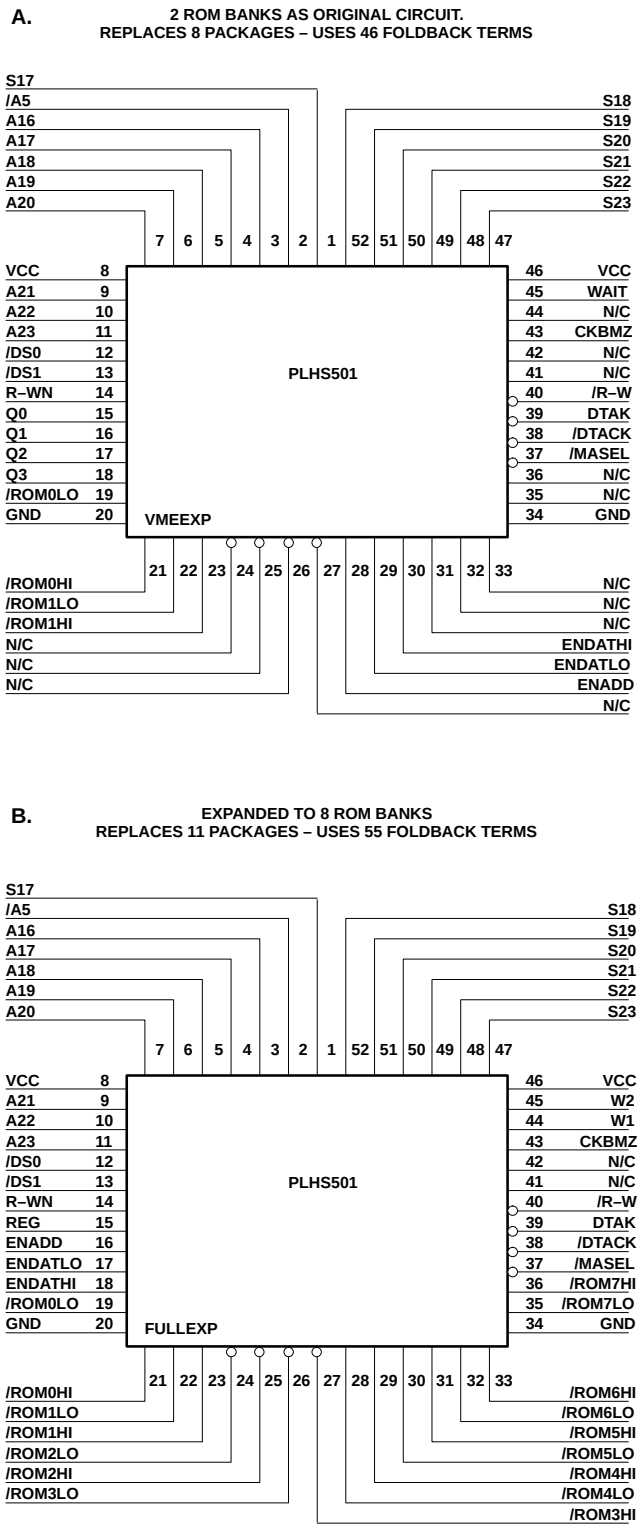


Figure 28. VMEEXP and FULLEXP

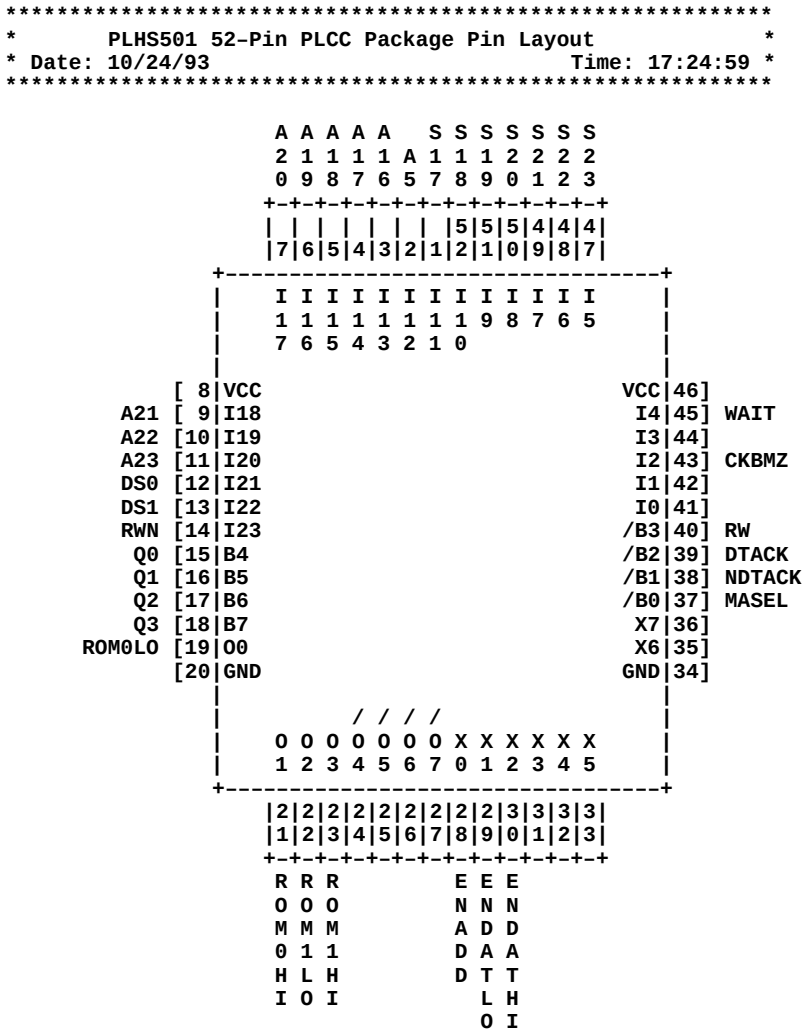


Figure 29. VMEEXP PLHS501 Pinlist

```
@PINLIST

WAIT       I;
CKBMZ      I;
DS[1..0]   I;
RWN        I;
A5         I;
A[16..23]  I;
S[17..23]  I;

Q[0..3]    O;     ENDATLO   O;
ROM0LO     O;     ENDATHI   O;
ROM0HI     O;     RW        O;
ROM1LO     O;     NDTACK    O;
ROM1HI     O;     DTACK     O;
ENADD      O;     MASEL     O;

@LOGIC EQUATIONS

RO3 = (/(MASEL*SO3*CKBMZ*DO3));
SO3 = (/(CKBMZ*(/(SO3*DO3*MASEL))));
DO3 = (/(Q2*RO3));
RO2 = (/(MASEL*SO2*CKBMZ*DO2));
SO2 = (/(CKBMZ*(/(SO2*DO2*MASEL))));
DO2 = (/(Q1*RO2));
RO1 = (/(MASEL*SO1*CKBMZ*DO1));
SO1 = (/(CKBMZ*(/(SO1*DO1*MASEL))));
DO1 = (/(Q0*RO1));
RO0 = (/(MASEL*SO0*CKBMZ*DO0));
SO0 = (/(CKBMZ*(/(SO0*DO0*MASEL))));
DO0 = (/(0*RO0));

ROM0LO = (/DS0+(/(/A16*MASEL)));
ROM0HI = (/DS1+(/(/A16*MASEL)));
ROM1LO = (/DS0+(/( A16*MASEL)));
ROM1HI = (/DS1+(/( A16*MASEL)));

Q0 = (/(((/(RO0*Q0))*SO0*(MASEL)));
Q1 = (/(((/(RO1*Q1))*SO1*(MASEL)));
Q2 = (/(((/(RO2*Q2))*SO2*(MASEL)));
Q3 = (/(((/(RO3*Q3))*SO3*(MASEL)));

MASEL = /(/(/(/((A17*S17+/A17*/S17)
               *(A18*S18+/A18*/S18)
               *(A19*S19+/A19*/S19)
               *(A20*S20+/A20*/S20)
               *(A21*S21+/A21*/S21)
               *(A22*S22+/A22*/S22)
               *(A23*S23+/A23*/S23)
              *(A5)))));

NDTACK  = /(((/(MASEL+WAIT))*RWN);
DTACK   = /NDTACK;
RW      = /(RWN);
ENADD   = A5;
ENDATLO = ((RW+MASEL)+/DS0);
ENDATHI = ((RW+MASEL)+/DS1);
```

Figure 30. VMEEXP PLHS501 .BEE File

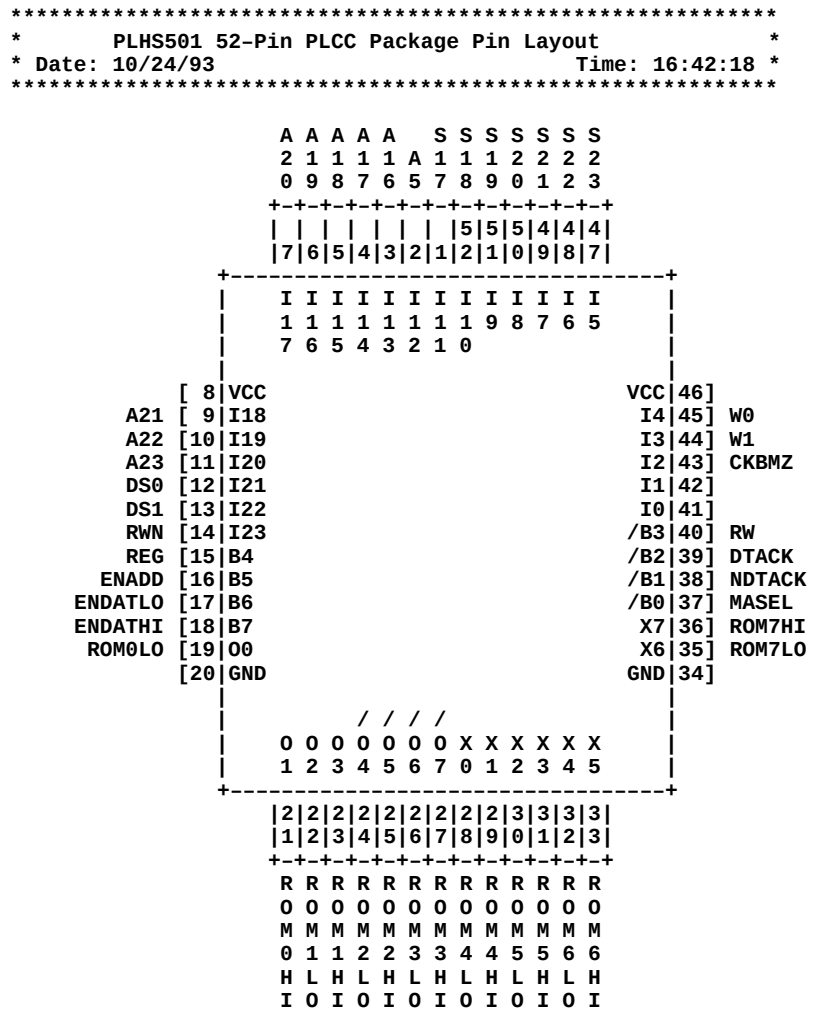


Figure 31. FULLEXP Pinlist

```
@PINLIST
CKBMZ      I;
DS[1..0]   I;
RWN        I;
A5         I;
A[16..23]  I;
S[17..23]  I;
W[1..0]    I;
                   ENDATLO   O;
ROM0LO     O;      ENDATHI   O;
ROM0HI     O;      RW        O;
ROM1LO     O;      NDTACK    O;
ROM1HI     O;      DTACK     O;
ENADD      O;      MASEL     O;
@GROUPS
ADDR = A[18..16];
@LOGIC EQUATIONS
RO3 = (/(MASEL*SO3*CKBMZ*DO3));
SO3 = (/(CKBMZ*(/(SO3*DO3*MASEL))));
DO3 = (/(Q2*RO3));
RO2 = (/(MASEL*SO2*CKBMZ*DO2));
SO2 = (/(CKBMZ*(/(SO2*DO2*MASEL))));
DO2 = (/(Q1*RO2));
RO1 = (/(MASEL*SO1*CKBMZ*DO1));
SO1 = (/(CKBMZ*(/(SO1*DO1*MASEL))));
DO1 = (/(Q0*RO1));
RO0 = (/(MASEL*SO0*CKBMZ*DO0));
SO0 = (/(CKBMZ*(/(SO0*DO0*MASEL))));
DO0 = (/(0*RO0));
ROM0LO = (/DS0+(/((addr == 0h)*MASEL)));
ROM0HI = (/DS1+(/((addr == 0h)*MASEL)));
ROM1LO = (/DS0+(/((addr == 1h)*MASEL)));
ROM1HI = (/DS1+(/((addr == 1h)*MASEL)));
ROM2LO = (/DS0+(/((addr == 2h)*MASEL)));
ROM2HI = (/DS1+(/((addr == 2h)*MASEL)));
ROM3LO = (/DS0+(/((addr == 3h)*MASEL)));
ROM3HI = (/DS1+(/((addr == 3h)*MASEL)));
ROM4LO = (/DS0+(/((addr == 4h)*MASEL)));
ROM4HI = (/DS1+(/((addr == 4h)*MASEL)));
ROM5LO = (/DS0+(/((addr == 5h)*MASEL)));
ROM5HI = (/DS1+(/((addr == 5h)*MASEL)));
ROM6LO = (/DS0+(/((addr == 6h)*MASEL)));
ROM6HI = (/DS1+(/((addr == 6h)*MASEL)));
ROM7LO = (/DS0+(/((addr == 7h)*MASEL)));
ROM7HI = (/DS1+(/((addr == 7h)*MASEL)));
Q0 = (/(((/(RO0*Q0))*SO0*(MASEL)));
Q1 = (/(((/(RO1*Q1))*SO1*(MASEL)));
Q2 = (/(((/(RO2*Q2))*SO2*(MASEL)));
Q3 = (/(((/(RO3*Q3))*SO3*(MASEL)));
MASEL = /(/(/(/((A17*S17+/A17*/S17)
               *(A18*S18+/A18*/S18)
               *(A19*S19+/A19*/S19)
               *(A20*S20+/A20*/S20)
               *(A21*S21+/A21*/S21)
               *(A22*S22+/A22*/S22)
               *(A23*S23+/A23*/S23)
               *(A5)))));
NDTACK  = /((/(MASEL+/((/q0*w0*/w1)+(q1*w0*/w1)+(/q2*/w0*w1)
          +(/q3*w0*w1))))*RWN);
DTACK   = /NDTACK;
RW      = /(RWN);
ENADD   = A5;
ENDATLO = ((RW+MASEL)+/DS0);
ENDATHI = ((RW+MASEL)+/DS1);
```

Figure 32. FULLEXP PLHS501 .BEE File

MICRO CHANNEL INTERFACE

IBM's new Micro Channel Architecture (MCA) bus implements new features not found on the XT/AT bus. One new requirement for adapter designers is that of Programmable Option Select (POS) circuitry. It allows system software to configure each adapter card upon power on, thereby eliminating option select switches or jumpers on the main logic board and on adapter cards.

Each adapter card slot has its own unique –CDSETUP signal routed to it. This allows the CPU to interrogate each card individually upon power up. By activating a card's –CDSETUP line along with appropriate address and control lines two unique 8 bit ID numbers are first read from the adapter. Based upon the ID number, the system then writes into the card's option latches configuration information that has been stored in the system's CMOS RAM. The CPU also activates POS latch address 102h bit 0, which is designated as a card enable bit.

If a new card is added to the system, an auto-configuration utility will be invoked. Each adapter card has associated with it a standardized Adapter Description File with filename of @XXXX.ADF, where XXXX is the hex ID number of the card. The configuration utility prompts the user according to the text provided in the .ADF file and updates the card's latches and the system's CMOS RAM.

IBM reserves 8 addresses for byte-wide POS latches, however, depending on the card's function, not all addresses need to be used. In addition, of those addresses that are used, only the bits used need to be latched. The first two addresses which are reserved for reading the ID bytes, and bit 0 of the third address, which is defined as a card enable bit, are mandatory. Some of the remaining bits of the third address are suggested by IBM to be used as inputs to an I/O or memory address comparator to provide for alternate card addresses. Many adapter cards will not use more than these three POS locations.

The following example describes an implementation of POS circuitry realized in a PLHS501. It uses only 56 of the possible 72 internal foldback NAND gates and only a portion of the device pins, allowing additional circuitry to ba added. Figure 33 shows a block diagram of the circuit, and Figures 35 and 36 are the SNAP files. Pins labeled DO0–DO7 must be connected externally to pins DIN0–DIN7. They also must be connected through a 74F245 transceiver to the Micro Channel. External transceiver direction and enable control is provided for by circuitry within the PLHS501. The external transceiver may also be used by other devices on the adapter card.

In this application, edge-triggered registers are not required and therefore should not be used, as transparent latches use fewer NAND gates to implement. Figure 34 shows the various latch circuits described by the SNAP equations. POS byte 2 was made using four of the /B device pins and four of the B pins. Notice however, from Figure 34(B) that the bits on the /B pins used the complement of the input pin, thereby implementing a non-inverting latch. Also, all 8 bits of this byte were brought to output pins. If some of the bits are not used by external circuitry, then the specific bit latch may not be needed or may be constructed entirely from foldback NAND gates freeing additional pins.

An external F521 may be added to provide for I/O address decoding. As the MCA bus requires all 16 bits of the I/O address to be decoded, 8 bits may be assigned to the F521 and 8 bits to the 501. Bit fields decoded in the 501 may be done so in conjunction with bits from POS byte 2 to provide for alternate I/O addressing. Additionally, some of the available 501 outputs may be used as device enables for other devices on the card.

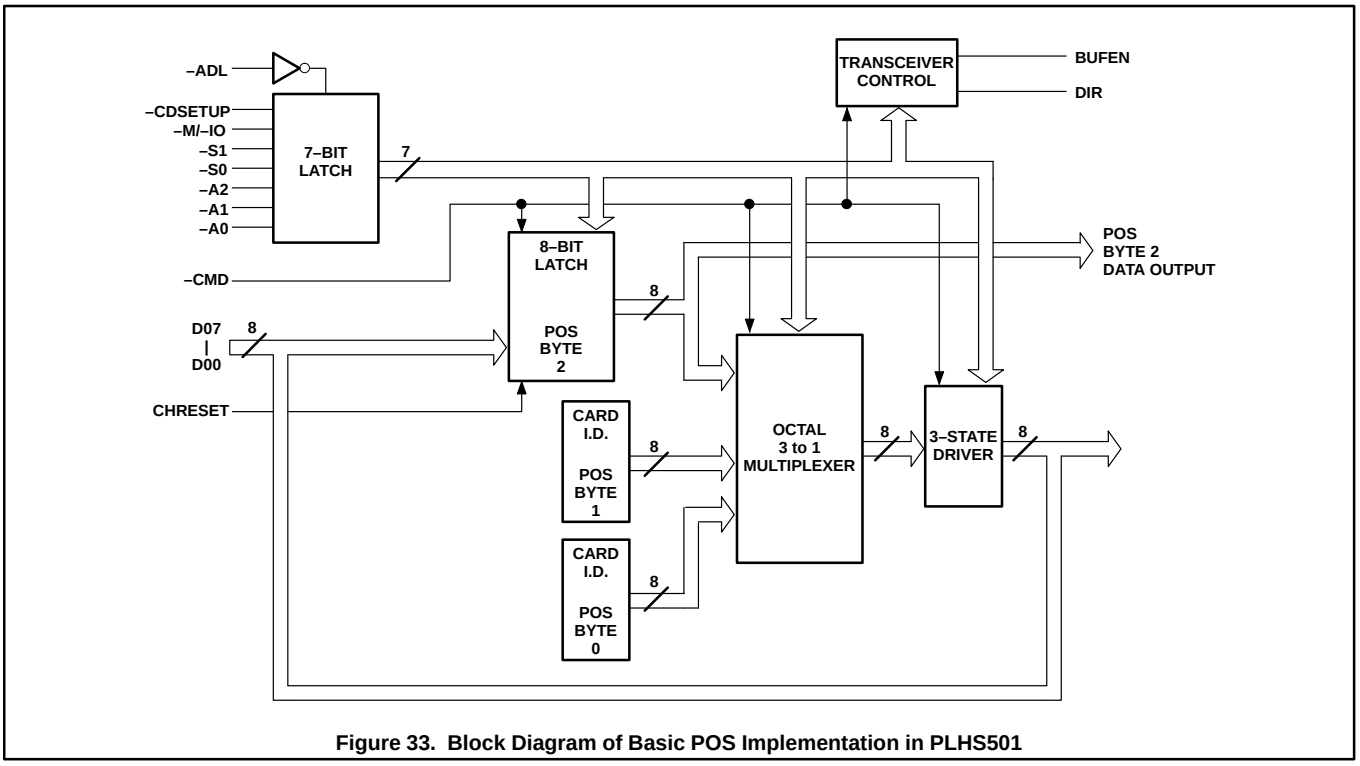
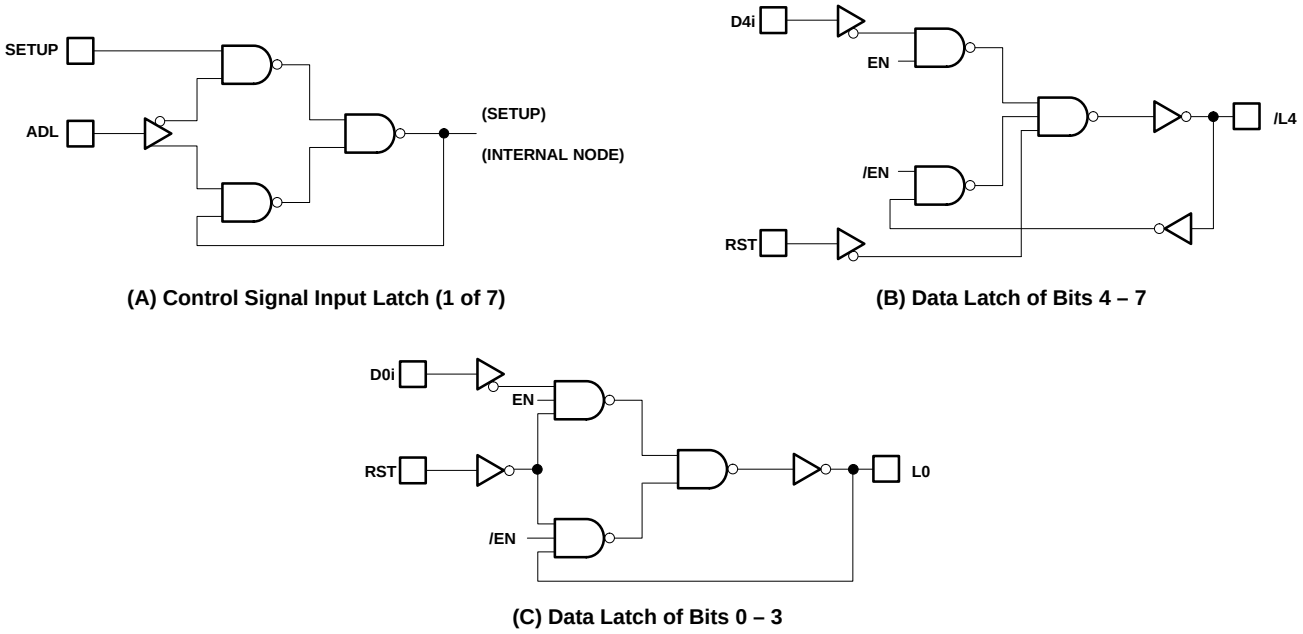


Figure 33. Block Diagram of Basic POS Implementation in PLHS501



(A) Control Signal Input Latch (1 of 7)

(B) Data Latch of Bits 4 – 7

(C) Data Latch of Bits 0 – 3

Figure 34. Latches Used in MCA Interface

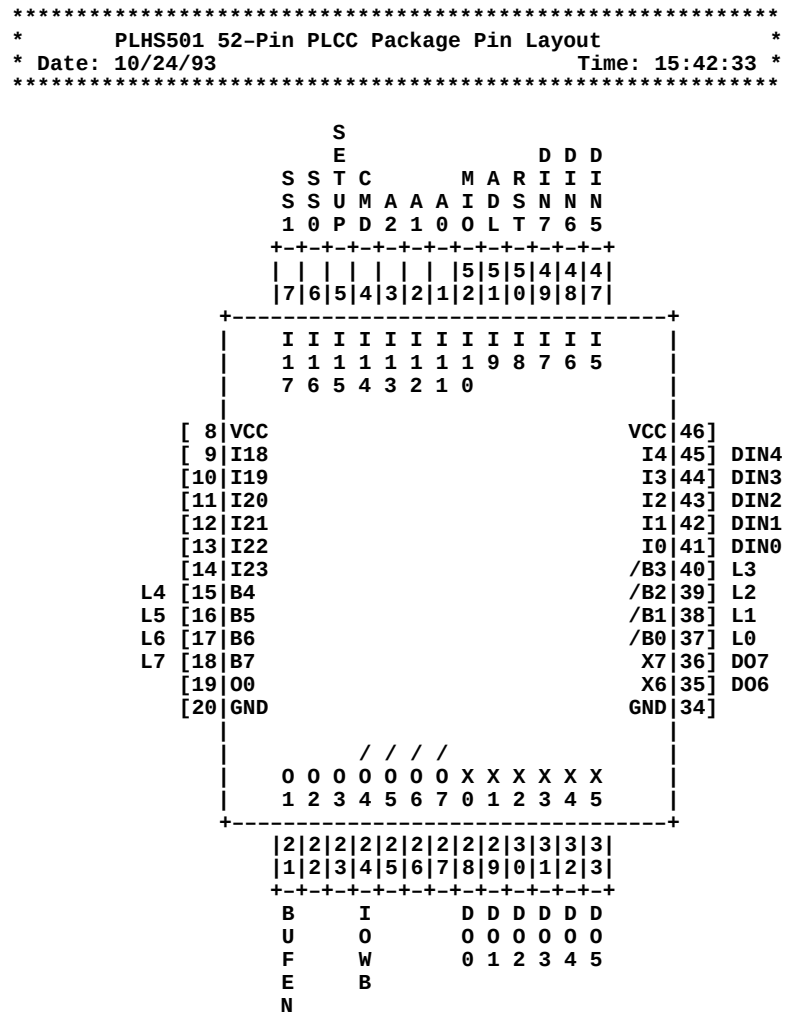


Figure 35. PLHS501 MCPOSREG Pinlist

```
”
 Basic Programmable Option Select circuitry
    for a Micro Channel Adaptor card
”

@PINLIST

a[2..0]   i;        l[7..0]  o;
cmd       i;        do[7..0] o;
setup     i;        bufen    o;
ss[1..0]  i;        iowb     o;
din[7..0] i;
adl       i;
mio       i;
rst       i;

@LOGIC EQUATIONS

 read0 = (nsetupl*/ss1l*ss0l*nmiol*/cmd*na2l*/a1l*/a0l);
 read1 = (nsetupl*/ss1l*ss0l*nmiol*/cmd*na2l*/a1l* a0l);
 read2 = (nsetupl*/ss1l*ss0l*nmiol*/cmd*na2l* a1l*/a0l);

 b7hi = 0;  ” Define high ID byte ”
 b6hi = 1;  ”   (POS byte #1)     ”
 b5hi = 1;  ”      7E hex         ”
 b4hi = 1;
 b3hi = 1;
 b2hi = 1;
 b1hi = 1;
 b0hi = 0;

 b7lo = 1;   ” Define low ID byte ”
 b6lo = 1;   ”    (POS byte #0)   ”
 b5lo = 1;   ”       FF hex       ”
 b4lo = 1;
 b3lo = 1;
 b2lo = 1;
 b1lo = 1;
 b0lo = 1;

   ” 7–Bit Input Latch for Control Signals ”

nsetupl = /setup*/adl + nsetupl*adl;
nmiol   = /mio */adl  + nmiol  *adl;
ss1l    = ss1  */adl  + ss1l   *adl;
ss0l    = ss0  */adl  + ss0l   *adl;
na2l    = /a2  */adl  + na2l   *adl;
a1l     = a1   */adl  + a1l    *adl;
a0l     = a0   */adl  + a0l    *adl;

   ” Option Select Octal Data Latch (POS byte #2) ”
   ” l0 is to be used as a card enable signal”

nen = /(nsetupl*/ss0l*ss1l*nmiol*/cmd*na2l*a1l*/a0l); ”write to latch”

l7 = /(/din7 * /nen)  * /(/l7 * nen) * (/rst);
l6 = /(/din6 * /nen)  * /(/l6 * nen) * (/rst);
l5 = /(/din5 * /nen)  * /(/l5 * nen) * (/rst);
l4 = /(/din4 * /nen)  * /(/l4 * nen) * (/rst);
l3 = /(/( din3 * /nen * /rst) * /(l3 * nen  * /rst));
l2 = /(/( din2 * /nen * /rst) * /(l2 * nen  * /rst));
l1 = /(/( din1 * /nen * /rst) * /(l1 * nen  * /rst));
l0 = /(/( din0 * /nen * /rst) * /(l0 * nen  * /rst));
```

Figure 36. PLHS501 MCPOSREG .EQN File (1 of 2)

```
    ” Octal 3 to 1 Multiplexer ”
    ” This multiplexer selects between reading
      POS[0], POS[1] or POS[2] onto the data bus”

ido7 = (b7hi*read1 + b7lo*read0 + l7*read2);
ido6 = (b6hi*read1 + b6lo*read0 + l6*read2);
ido5 = (b5hi*read1 + b5lo*read0 + l5*read2);
ido4 = (b4hi*read1 + b4lo*read0 + l4*read2);
ido3 = (b3hi*read1 + b3lo*read0 + l3*read2);
ido2 = (b2hi*read1 + b2lo*read0 + l2*read2);
ido1 = (b1hi*read1 + b1lo*read0 + l1*read2);
ido0 = (b0hi*read1 + b0lo*read0 + l0*read2);

  ”3–State output control for do7–do0”

do[7..0] = ido[7..0];
do[7..0].oe = (nsetupl*/ss1l*ss0l*nmiol*/cmd*na2l*outen);
outen =/(a1l*a0l);

  ”External F245 transceiver control”

iowb  = /(na2l * nsetupl * nmiol * ss1l * /ss0l);
niow  = /(na2l * nsetupl * nmiol * ss1l * /ss0l);
bufen = cmd * niow;
```

Figure 36. PLHS501 MCPOSREG .EQN File (2 of 2)

NuBus INTERFACE

In Apple Computer's book* "*Designing Cards and Drivers for Macintosh II and Macintosh SE"*, an application was described for interfacing an 8-bit I/O controller to the NuBus. The controller used was a SCSI controller of the type used on the main Macintosh logic board. Seven devices (three of which were PAL architecture) were used as control circuitry interfacing the SCSI controller and two RAM chips to the bus.

This example of using the PLHS501 shows a method of interfacing the same SCSI controller and RAM chips to the NuBus using only three parts. The adapter card schematic is shown in Figure 38, the SNAP pin listing is in Figure 42, and the SNAP .EQN listing is in Figure 43. Although the SNAP listing may seem confusing at first glance, the circuitry fused into the PLHS501 can be broken down into small blocks of latches, flip-flops, and schematically in Figures 40 and 41. Circuit timing is shown in Figure 39.

Referring to Figure 40 and Figure 41, the circuitry starts a transaction by first detecting a valid address in either the slot or super slot range. The detection is accomplished by two wide-input NAND gates, and controlled by the /CLK signal. Following each NAND gate is an S-R latch to hold the signal until near the end of the cycle. The two S-R latch signals are combined into one signal named ST0 such that if either NAND gate output was low, then some delay time after the rising edge of /CLK, ST0 will go low. The next rising edge of /CLK will cause signal ST1 to go low. This sets signal DE2 low, which is an input to an external flip-flop to cause ST2 to go low at the next rising /CLK edge terminating the cycle. An external flip-flop was necessary to achieve a high-speed /CLK to /IOR and /ACK transition. Also, an external FI25 buffer was added to meed the soon to be approved IEEE P1196 specification requirement of 60mA I_{OL} for signal /NMRQ and 24mA I_{OL} for signals /TM0/TM1 and /ACK. Figure 41(B) shows an easily implemented latch which controls interrupts generated by the SCSI controller passing onto the bus. Upon /RESET the latch is put into a known state. Under software control, by writing to a decoded address, the latch may be set or reset, thereby gating or blocking the interrupt signals.

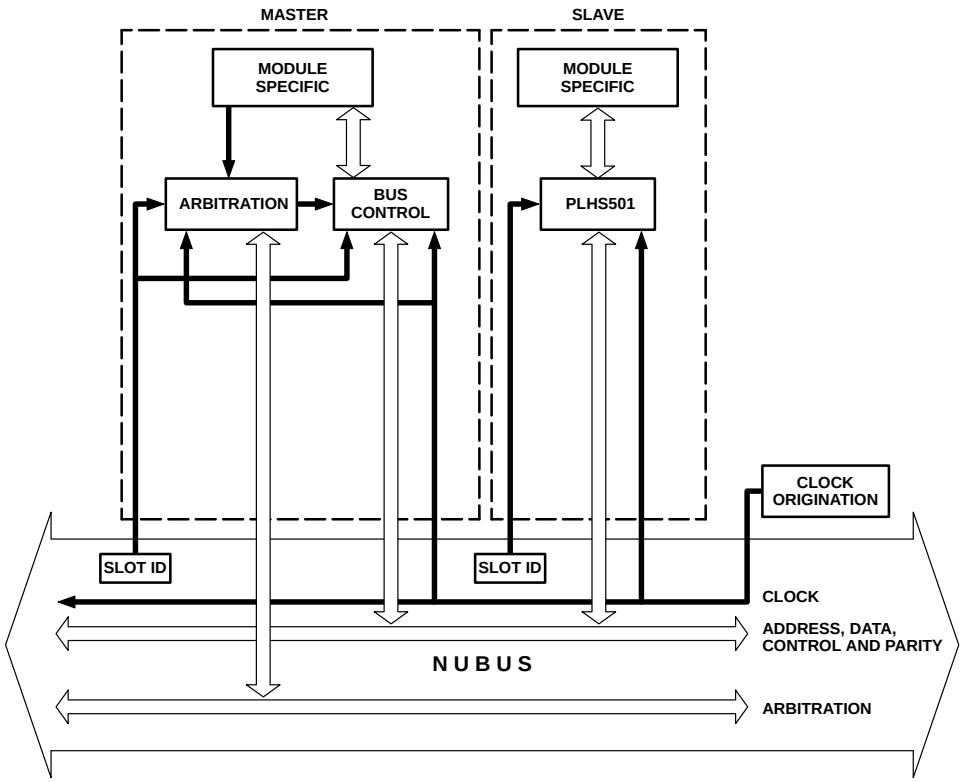


Figure 37. Simplified NuBus™ Diagram

* *Designing Cards and Drivers for Macintosh II and Macintosh SE*, Addison–Wesley Publishing Company, Inc. 1987.

PLHS501 design examples AN049

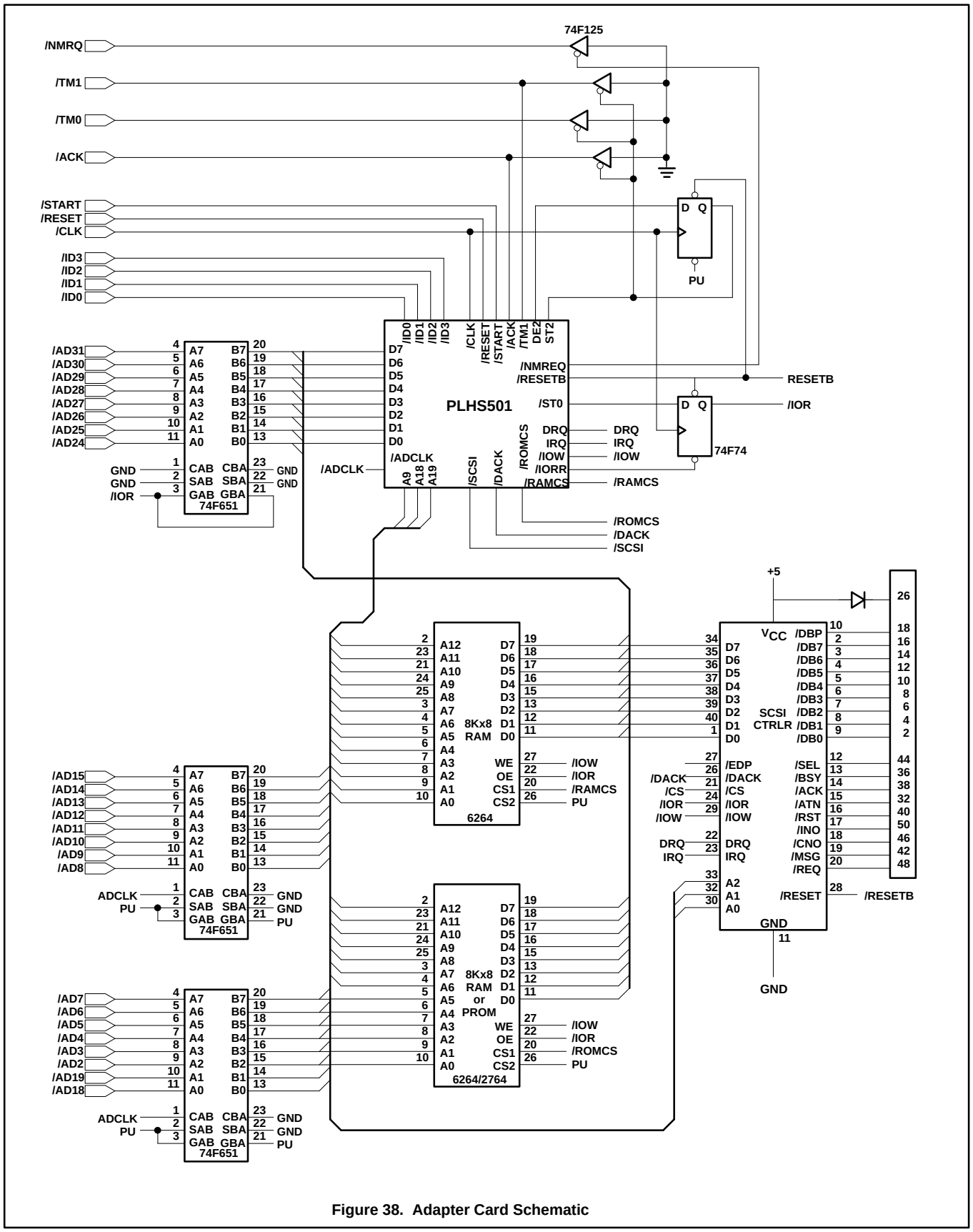


Figure 38. Adapter Card Schematic

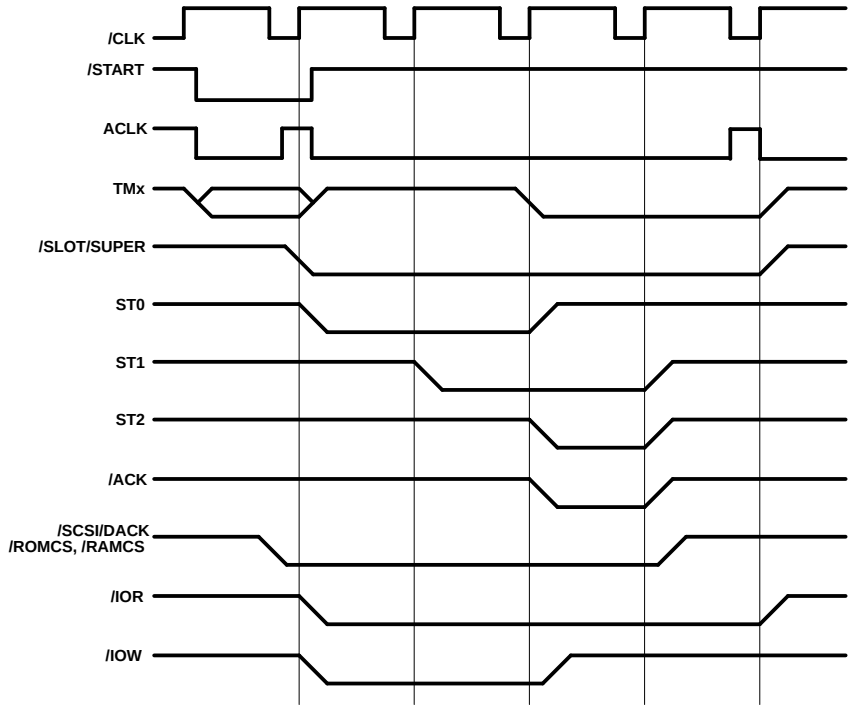


Figure 39. Timing Diagram

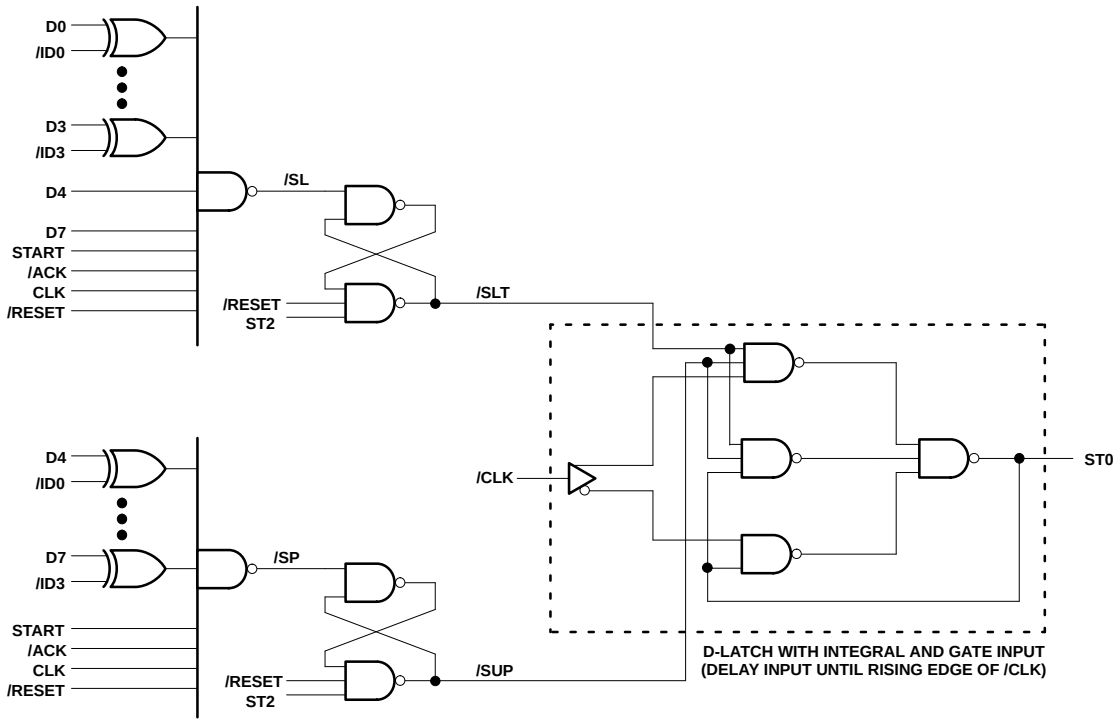
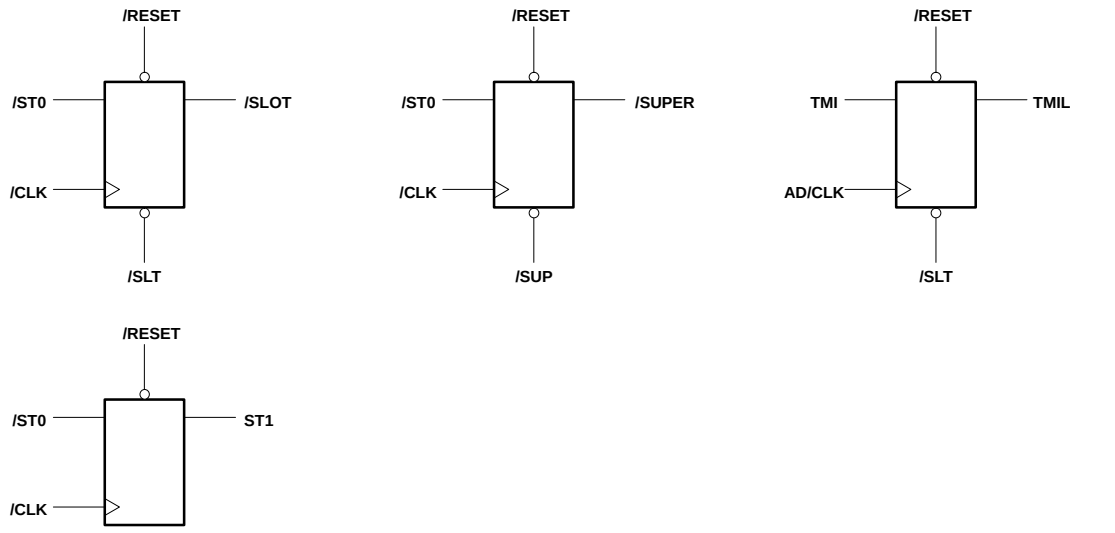
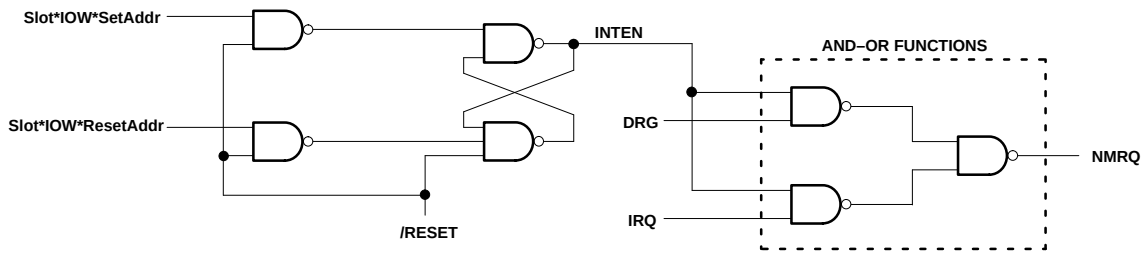


Figure 40. Decoding and Latch Circuitry



(A) Four Internal Flip-Flops Constructed from NAND Gates.



(B) Interrupt Enable Control Latch
Internal Flip-Flops and Latches

Figure 41. Internal Flip-Flops and Latches

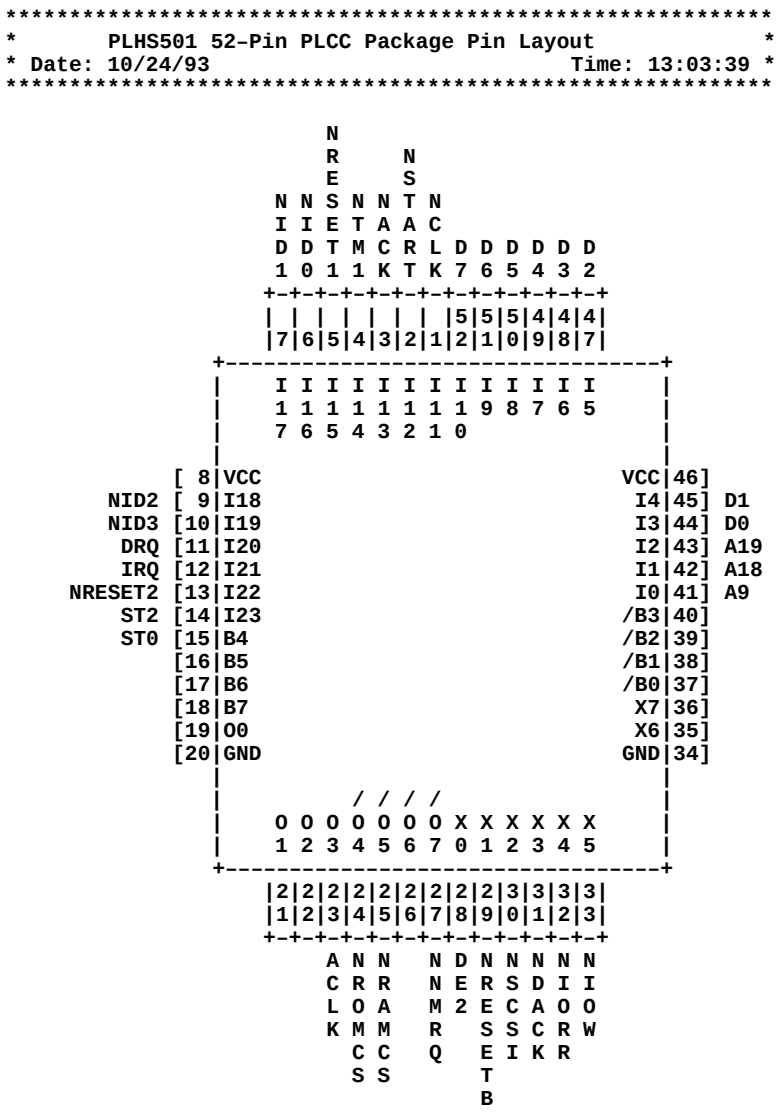


Figure 42. SNAP Pin List

```
   "SCSI-NuBus Interface"

@PINLIST

nid[0..3] i;          nromcs   o;
d[7..0]   i;          nramcs   o;
a9        i;          nnmrq    o;
a[19..18] i;          nresetb  o;
nreset1   i;          nscsi    o;
nreset2   i;
ntm1      i;          ndack    o;
nack      i;          niorr    o;
nstart    i;          niow     o;
nclk      i;          st0      b;
drq       i;          aclk     o;
irq       i;          de2      o;
st2       i;

@LOGIC EQUATION

                        "Address Decode"
cmp0a = (d0*/nid0+/d0*nid0);
cmp1a = (d1*/nid1+/d1*nid1);
cmp2a = (d2*/nid2+/d2*nid2);
cmp3a = (d3*/nid3+/d3*nid3);
cmp0b = (d4*/nid0+/d4*nid0);
cmp1b = (d5*/nid1+/d5*nid1);
cmp2b = (d6*/nid2+/d6*nid2);
cmp3b = (d7*/nid3+/d7*nid3);

nsl = /(d7*d6*d5*d4*cmp0a*cmp1a*cmp2a*cmp3a*/nstart*nack*/nclk);
nsp = /(cmp0b*cmp1b*cmp2b*cmp3b*/nstart*nack*/nclk);

                    "latch slot signal"
nslt = /(nreset1*st2*/(nsl*nslt));
                    "latch super signal"
nsup = /(nreset1*st2*/(nsp*nsup));

                     "Let nslt or nsup through only
                      until after the rising edge
                      of nclk"
ist0 = /(/(nslt*nsup*nclk) * /(st0*/nclk) * /(nslt*nsup*st0) * nreset1);
st0 = ist0;
st0.oe = 1;

                     "Slot signal D-type Flip Flop"
nslot.d = st0;
nslot.clk = nclk;
nslot.set = nreset2;
nslot.rst = nslt;

                     "Super signal D-type Flip Flop"
nsuper.d = st0;
nsuper.clk = nclk;
nsuper.set = nreset2;
nsuper.rst = nsup;

                      "State 1 D-type Flip Flop"
st1.d = st0;
st1.clk = nclk;
st1.set = nreset2;

                      "output to external flop"
de2  = /(/st1 * st2);

                      "address latch clock"
adclk = /nclk*st0*st1;
aclk  = /nclk*st0*st1;

                      "latch tm1 signal for r/w info"
tm1l.d = ntm1;
tm1l.set = nreset2;
tm1l.clk = adclk;
tm1l.rst = nslt;
```

Figure 43. SNAP .EQN Listing (1 of 2)

```
”
tm1l  –> 1 read, 0 write
”
                        ”straight decode stuff”
niorr = /(/st0*tm1l                    * nreset1);
niow  = /(/tm1l*/st0                   * nreset1);
nscsi = /(/nslot*/a19*/a18*/a9         * nreset1);
ndack = /(/nslot*/a19*/a18* a9         * nreset2);
nromcs= /(/nslot* a19* a18             * nreset2);
nramcs= /(/nsuper                      * nreset2);
nresetb= nreset2;
                         ”interrupt control latch”
setad = /(/tm1l*/st0*/nslot* a19*/a18* a9);
rstad = /(/tm1l*/st0*/nslot* a19*/a18*/a9);
inten = /(setad*(/(inten*rstad*nreset2)));
nnmrq = /(inten*drq+inten*irq);
```

Figure 43. SNAP .EQN Listing (1 of 2)

Data Bus Parity

The PLHS501 can span 32 bits of input data. It has four output Ex-OR gates, and the ability to generate literally any function of the inputs. It would seem that there must be some "best" way to generate and detect parity. Recall that the PLHS501 can generate both deep logic functions (lots of levels) and wide logic functions (lots of inputs). The best solution would require the fewest gates and the fewest number of logic levels. Let's review the basics, first. Table 1(A) shows the parity function for two variables and Table 1(B) shows it for three variables. The Ex-OR function generates even parity.

It is noticeable that there are precisely 50% logical 1 entries in the truth tables. This yields the famous checkerboard Karnaugh Maps. With a checkerboard K-map, no simplification of Ex-OR functions is possible by Boolean simplification. The two variable Ex-OR has two ones (implying 3 gates to generate), the 3 variable has four ones (implying 5 gates to generate). In general, $2^{n-1}+1$ product terms could generate Ex-OR functions in two levels of NAND gates (assuming complementary input variables exist). You must have an unlimited number of gate inputs for this to hold.

The PLHS501 could do this for 7 input variables in two levels ($2^6+1=65$), but cannot support 8 ($2^7+1=129$). Hence, it is appropriate to seek a cascaded solution, hopefully taking advantage of the available output Ex-OR functions. Let's solve a 16 input Ex-OR function, by subpartitioning. First, consider Figure 44(A) where two literals are Exclusive-ORed to generate an intermediate Ex-OR function. This requires available complementary inputs and generates even parity in two levels. Figure 44(B) also does this (by factoring), requiring 3 gate levels, but does not require complementary inputs.

Assuming inputs must get into the PLHS501 through the pin receivers, it is best to generate as wide of an initial Ex-OR as possible, so a structure like Figure 44(A) expanded is appropriate. Figure 44 shows a 2-level 4 input Ex-OR function which may be viewed as a building block. This structure may be repeated four times, across four sets of four input bits generating partial intermediate parity values which may then be treated through two boxes similar to Figure 44(B). These outputs are finally combined through an output Ex-OR at a PLHS501 output pin. Figure 46 shows the complete solution which requires 44 NANDs plus one Ex-OR.

Figures 47 and 48 show the pin layout and SNAP equations for a parity generator. This example uses a cascade with a different partitioning than just previously discussed.

Table 1. Even Parity Functions

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 1(A).

A	B	C	A ⊕ B
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 1(B).

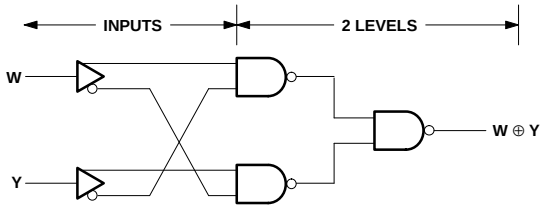


Figure 44(A).

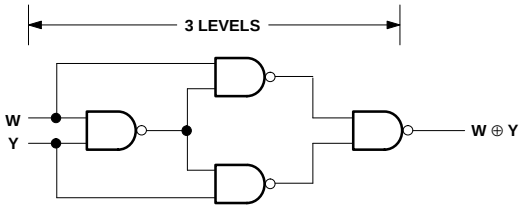


Figure 44(B).

Figure 44. Complementary Input Levels

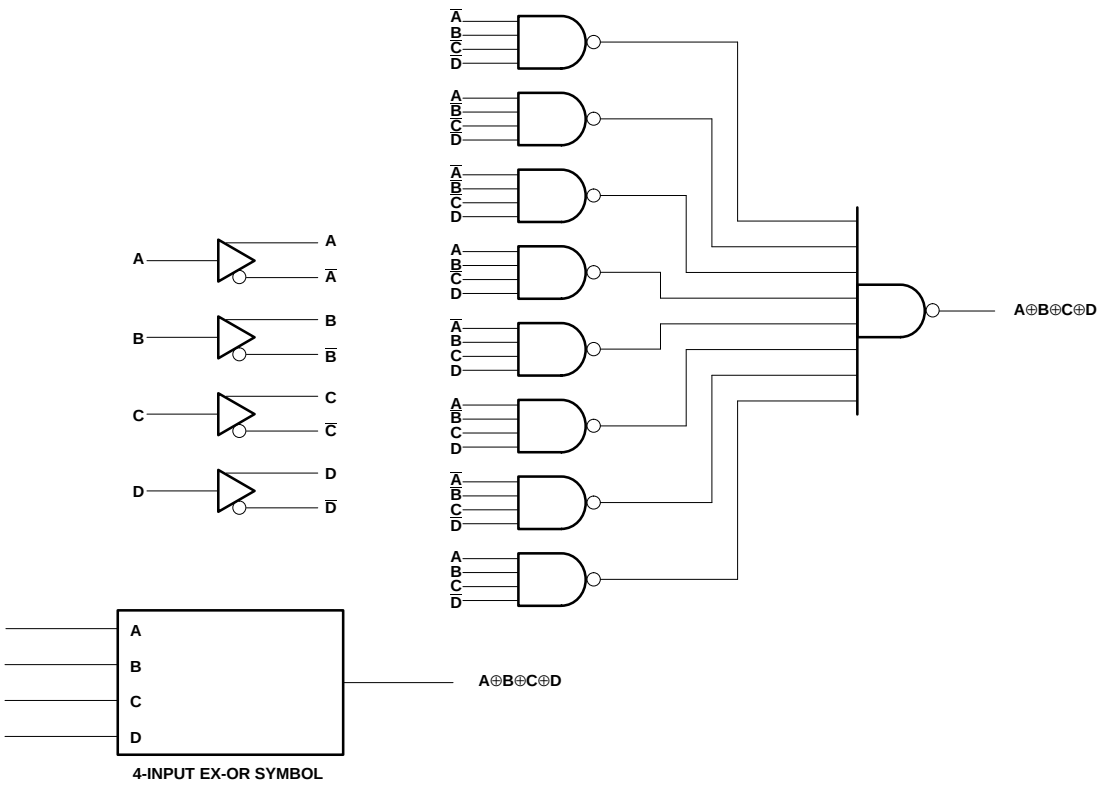


Figure 45. Four Variable Ex-ORs

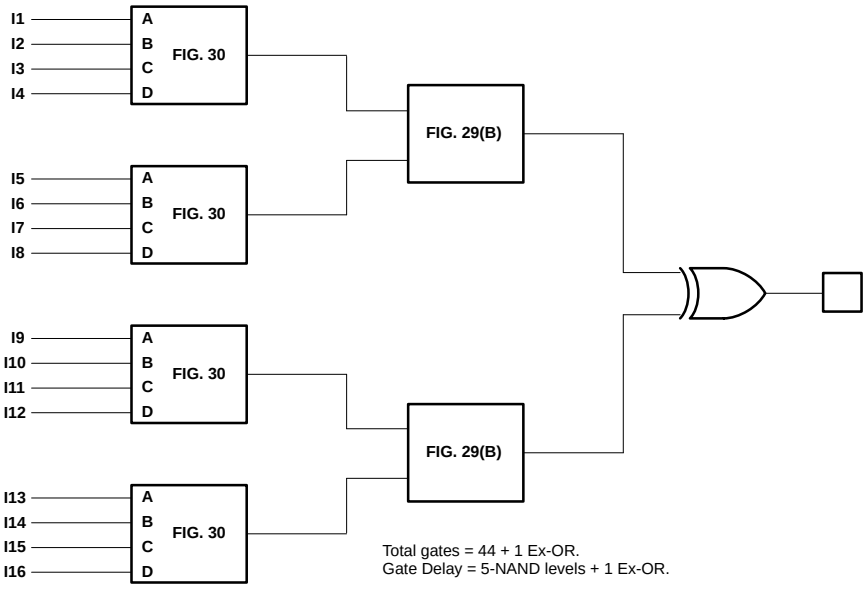


Figure 46. 16 Input Even Parity Generation

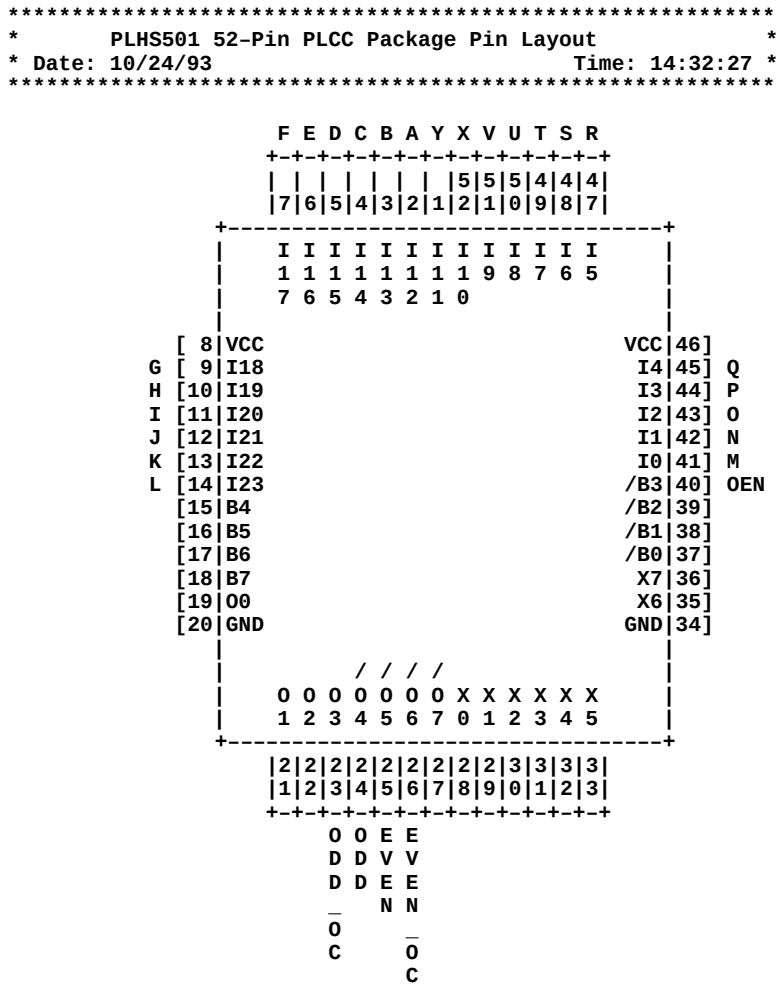


Figure 47. PARITET PLHS501 Pinlist

```
”FILENAME:PARITET.EQN
          24 BIT PARITY CIRCUIT”
@PINLIST
A       I;
B       I;
C       I;
D       I;
E       I;
F       I;
G       I;
H       I;
I       I;
J       I;
K       I;
L       I;
M       I;
N       I;
O       I;
P       I;
Q       I;
R       I;
S       I;
T       I;
U       I;
V       I;
X       I;
Y       I;
OEN     I;
ODD     O;
EVEN    O;
ODD_OC  O;
EVEN_OC O;

@LOGIC EQUATION

     ”FIRST LEVEL: ‘EVEN’ FROM GROUPS OF THREE INPUTS”
   J0=/A*/B*/C + /A*B*C + A*/B*C + A*B*/C;
   J1=/D*/E*/F + /D*E*F + D*/E*F + D*E*/F;
   J2=/G*/H*/I + /G*H*I + G*/H*I + G*H*/I;
   J3=/J*/K*/L + /J*K*L + J*/K*L + J*K*/L;
   J4=/M*/N*/O + /M*N*O + M*/N*O + M*N*/O;
   J5=/P*/Q*/R + /P*Q*R + P*/Q*R + P*Q*/R;
   J6=/S*/T*/U + /S*T*U + S*/T*U + S*T*/U;
   J7=/V*/X*/Y + /V*X*Y + V*/X*Y + V*X*/Y;

     ”SECOND LEVEL: ‘EVEN’ FROM FOUR GROUPS AT A TIME”
   J8=/J0*/J1*/J2*/J3 + /J0*/J1*J2*J3 + J0*J1*/J2*/J3 + /J0*J1*J2*/J3
      + J0*/J1*/J2*J3 + /J0*J1*/J2*J3 + J0*/J1*J2*/J3 + J0*J1*J2*J3;
   J9=/J4*/J5*/J6*/J7 + /J4*/J5*J6*J7 + J4*J5*/J6*/J7 + /J4*J5*J6*/J7
      + J4*/J5*/J6*J7 + /J4*J5*/J6*J7 + J4*/J5*J6*/J7 + J4*J5*J6*J7;

   T0=/(J8*J9);
   T1=/(/J8*/J9);
   T2=/(J8*/J9);
   T3=/(/J8*J9);

   ODDI=/(T2*T3);
   EVENI=/(T0*T1);
   ODD=ODDI;
   EVEN=EVENI;
   ODD.OE = /OEN;
   EVEN.OE = /OEN;
   ODD_OCI = 0;
   EVEN_OCI = 0;
   ODD_OC=ODD_OCI;
   EVEN_OC=EVEN_OCI;
   ODD_OC.OE = T2*T3*/OEN;
   EVEN_OC.OE = T0*T1*/OEN;
```

Figure 48. PARITET PLHS501 .BEE File

16-Bit Comparator

This example “compare”, implements, a 16-bit comparator over 32 input bits. The design generates outputs for conditions representing the classic “EQUAL”, “AGTB” (A>B) and BGTA (B>A). The long, triangularized equation for T42 suggests a clever editing approach to accurately enter a relatively long design equation into SNAP.

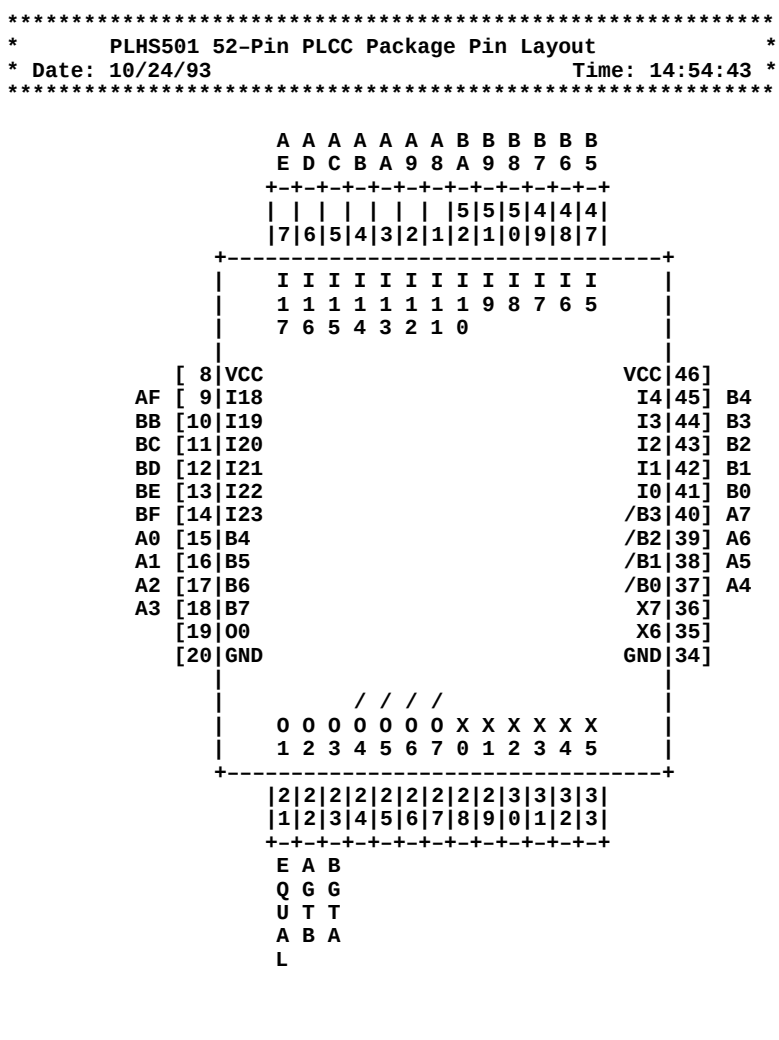


Figure 49. PLHS501 Pinlist for 16–Bit Comparator

```
"FILENAME:PARITET.EQN
          16 BIT COMPARATOR WITH THREE OUTPUTS:
          EQUAL,AGTB (A>B), AND BGTA (B>A)"

@PINLIST

A[9..0] I;
AA      I;   AD  I;
AB      I;   AE  I;
AC      I;   AF  I;

B[9..0] I;
BA      I;   BD  I;
BB      I;   BE  I;
BC      I;   BF  I;

EQUAL   O;
AGTB    O;
BGTA    O;

@LOGIC EQUATION
   T1=/(AF*/BF);    T2=/(/AF*BF);
   T3=/(AE*/BE);    T4=/(/AE*BE);
   T5=/(AD*/BD);    T6=/(/AD*BD);
   T7=/(AC*/BC);    T8=/(/AC*BC);
   T9=/(AB*/BB);    T10=/(/AB*BB);
   T11=/(AA*/BA);   T12=/(/AA*BA);
   T13=/(A9*/B9);   T14=/(/A9*B9);
   T15=/(A8*/B8);   T16=/(/A8*B8);
   T17=/(A7*/B7);   T18=/(/A7*B7);
   T19=/(A6*/B6);   T20=/(/A6*B6);
   T21=/(A5*/B5);   T22=/(/A5*B5);
   T23=/(A4*/B4);   T24=/(/A4*B4);
   T25=/(A3*/B3);   T26=/(/A3*B3);
   T27=/(A2*/B2);   T28=/(/A2*B2);
   T29=/(A1*/B2);   T30=/(A1*B1);
   T31=/(A0*/B0);   T32=/(/A0*B0);

   T41=T1*T2*T3*T4*T5*T6*T7*T8*T9*T10*T11*T12*T13*T14*T15*T16*T17*
     T18*T19*T20*T21*T22*T23*T24*T25*T26*T27*T28*T29*T30*T31*T32;
   T42=                                                          /T1+
                                                               /T3*T2+
                                                            /T5*T4*T2+
                                                         /T7*T6*T4*T2+
                                                      /T9*T8*T6*T4*T2+
                                                  /T11*T10*T8*T6*T4*T2+
                                              /T13*T12*T10*T8*T6*T4*T2+
                                          /T15*T14*T12*T10*T8*T6*T4*T2+
                                      /T17*T16*T14*T12*T10*T8*T6*T4*T2+
                                  /T19*T18*T16*T14*T12*T10*T8*T6*T4*T2+
                              /T21*T20*T18*T16*T14*T12*T10*T8*T6*T4*T2+
                          /T23*T22*T20*T18*T16*T14*T12*T10*T8*T6*T4*T2+
                      /T25*T24*T22*T20*T18*T16*T14*T12*T10*T8*T6*T4*T2+
                  /T27*T26*T24*T22*T20*T18*T16*T14*T12*T10*T8*T6*T4*T2+
              /T29*T28*T26*T24*T22*T20*T18*T16*T14*T12*T10*T8*T6*T4*T2+
        /T31*T30*T28 *T26*T24*T22*T20*T18*T16*T14*T12*T10*T8*T6*T4*T2;

   EQUAL=T41;
   AGTB=T42;
   BGTA=/(T41+T42);
```

Figure 50. Compare PLHS501 .BEE File