Designing with programmable macro logic

INTRODUCTION TO PROGRAMMABLE MACRO LOGIC DESIGN CONCEPTS

Programmable Macro Logic (PML), an extension of the Programmable Logic Array (PLA) concept combines a programming or fuse array with an array of wide input NAND gates wherein each gate folds back upon itself and all other such NAND gates. This is called a foldback NAND structure and its basic elements have been outlined previously (Cavlan[1], Wong[2], Gheissari and Safari[3]).

The choice of an internal NAND logic cell is appropriate because the cell is functionally complete, requiring but a single cell type to generate any Boolean function. A cell within the PLHS501 may be configured to accommodate from one to 32 inputs from the outside world, and up to 72 inputs from within the chip. Because the user can select either direct or inverted input variables, and either a direct or complemented output, the NAND function can generate, with a single pass through the programming array, the basic four logic functions of AND, OR, NAND, NOR. all these basic functions, can be extremely wide, of course (see Figure 1). This convenient structure allows efficient exploitation of all widely used minimization techniques (Karnaugh Maps, Quine-McClusky, Boolean Algebra, etc.).

The obvious extensions to additional combinational functions for decoding, multiplexing and general Boolean functions is straightforward. Adding feedback to the system expands the range of realizable functions to include sequential as well as combinational functions. Figure 2 illustrates the basic arrangement of the PLHS501.

Because of the large number of inputs each NAND gate has available, logic functions that require several levels of conventional 4 or 8 input gates may be able to be reduced to 1 or 2 levels. However, it is important to realize that unlike AND-OR PLD architectures, more than 2 levels of logic may be implemented in the PLHS501 without wasting output or input pins. Up to 72 levels of logic may be implemented due to each of the 72 foldback NAND gates.

So far, the concept of a “macro” is still not evident. Two ways for the generation of a macro exist—namely, hard and soft. Borrowing from the concept in computer programming wherein a section of code (called a macro) is repeated every time its use is required, we can establish subfunctions which can be repeated each time required. The user defined or soft macro can be one which will generate a function by fused interconnect. When a fixed design function is provided, it is a hard macro. This may be an optimized structure like a flip-flop or an adder, or some other function which is generated on the foundation, by the manufacturer. Soft macros are seldom optimized or precisely consistent, but hard macros are both optimized and unalterable.

When a user function for a particular use is isolated, defined and repetition of the function is required, special software constructs are provided which will allow it to be defined at a higher performance and functional density, and an array of choices which contain optimized functions or hard macros will be offered in successor chips. In particular, the PML2552 and PML2852 include an array of flip-flops for state machine design.

Optimizing combinational functions in PML consists largely in making choices and trade-offs. For single output logic functions, the choice is obvious from the truth table. If a particular function’s truth table has fewer entries that are logical zeroes than logical ones, product of sums should be chosen and the appropriate OR-AND structure generated. Otherwise, the usual sum of products should be chosen, minimizing as usual, before dropping into the two level AND-OR structure (using the NAND-NAND realization). Combining the availability of inversion at the input and output of the chip, the NAND-NAND structure can perform either the OR-AND or the AND-OR rendition of a function with equal logic levels. The designer needs only to choose the optimal rendition to suit his needs (see Table 1). Truth tables with 50% ones can use either version at the designers whim unless other uses arise.

PERFORMANCE

The PLHS501 (Figure 2) is a high speed, oxide isolated, vertically fused PML device containing 72 internal NAND functions which are combined with 24 dedicated outputs. A large collection of applications, both combinational and sequential, may be configured using this part which looks roughly like a small, user definable gate array. For the sake of clarity, worst case passing a signal from an input, making one pass through the NAND array (output terms) and exiting an output takes around 25 nanoseconds with each incremental pass through the NAND foldback array taking about 8 nanoseconds.

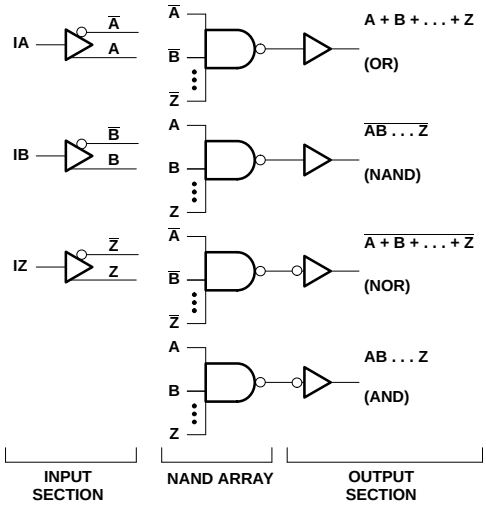


Figure 1. PML Basic Functions

Designing with programmable macro logic

Table 1. Example Demonstration

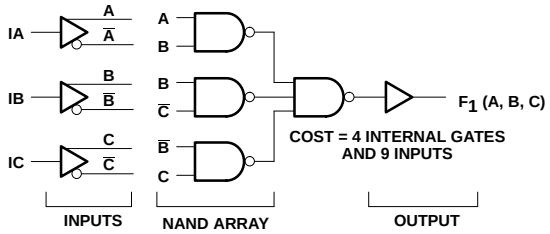
F1 (A, B, C) = $\overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}C + AB\overline{C} + ABC$

A	B	C	f1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

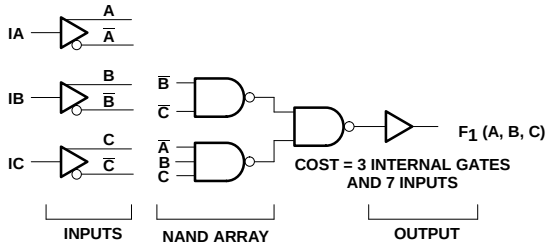
C \ AB	00	01	11	10
0	0	1	1	0
1	1	0	1	1

The optimal choice would be to generate the zero entries

If we group on the <u>one</u> entries we shall get: $AB + \overline{B}C + B\overline{C}$



If we group on the <u>zero</u> entries we get instead: $F_1 = (\overline{B} + \overline{C})(\overline{A} + B + C)$



Designing with programmable macro logic

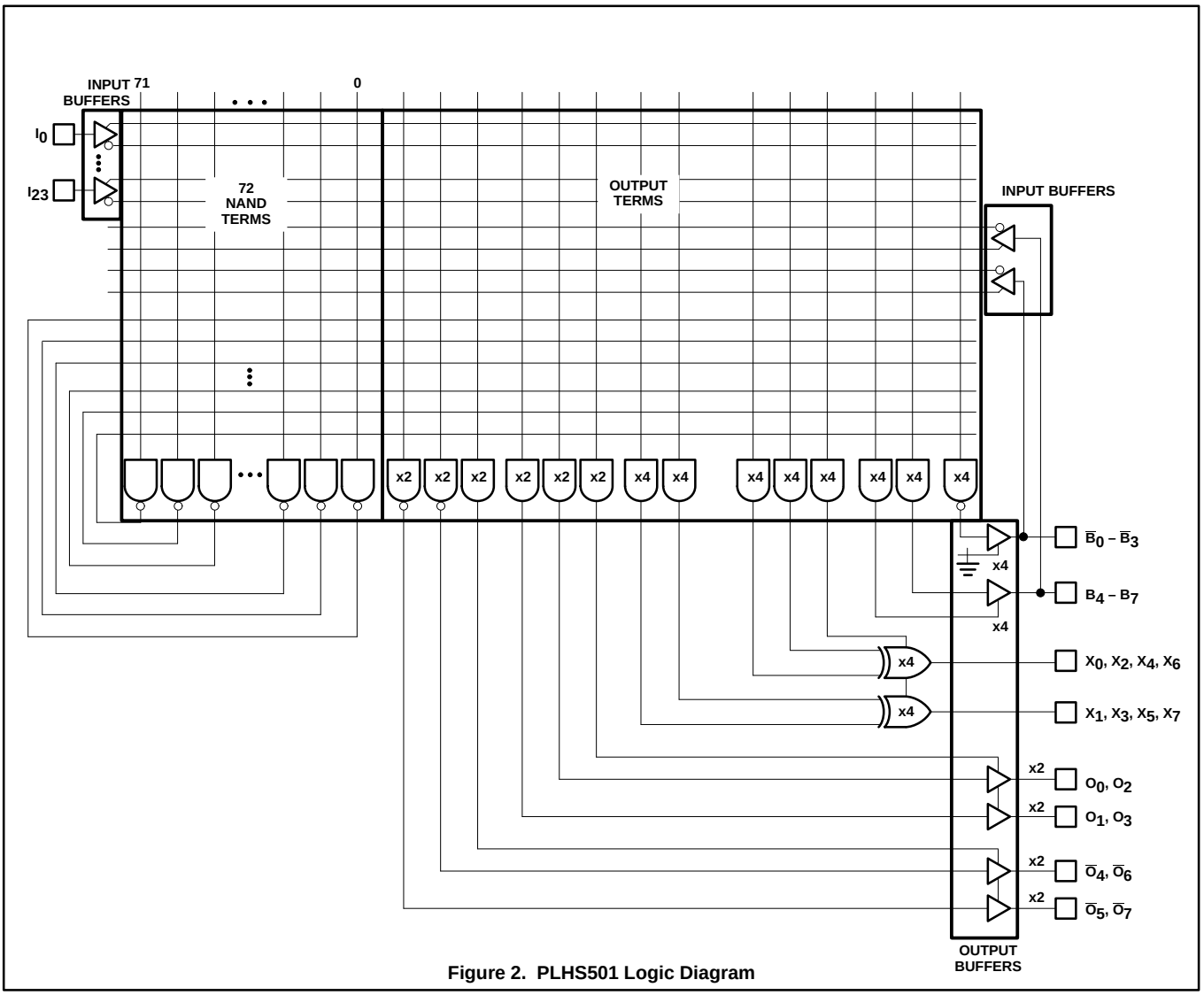


Figure 2. PLHS501 Logic Diagram

Designing with programmable macro logic

The data sheet first lists some maximum propagation delays from an input, through a NAND output term and out through various output gates. Secondly, it lists maximum propagation delays from an input, through a NAND foldback term, through a NAND output term and out through the different output gates.

It is intriguing that subtracting one from the other yields a NAND foldback gate delay of 5 to 6ns when the worst case gate delay of an internal foldback gate is listed as 8ns. This is due to the fact that a gate has less of a delay when its output is falling (t_{PHL}) than when its output is rising (t_{PLH}). When passing a signal through two NAND gates one gate will have less of a delay than the other, and since the individual rise and fall delays are not specified, this causes the apparent discrepancy between the two delays.

Figure 3, Figure 4, Figure 5 and Figure 6 show graphically the timing paths listed in the PLHS501 data sheet.

PLHS501 TIMING

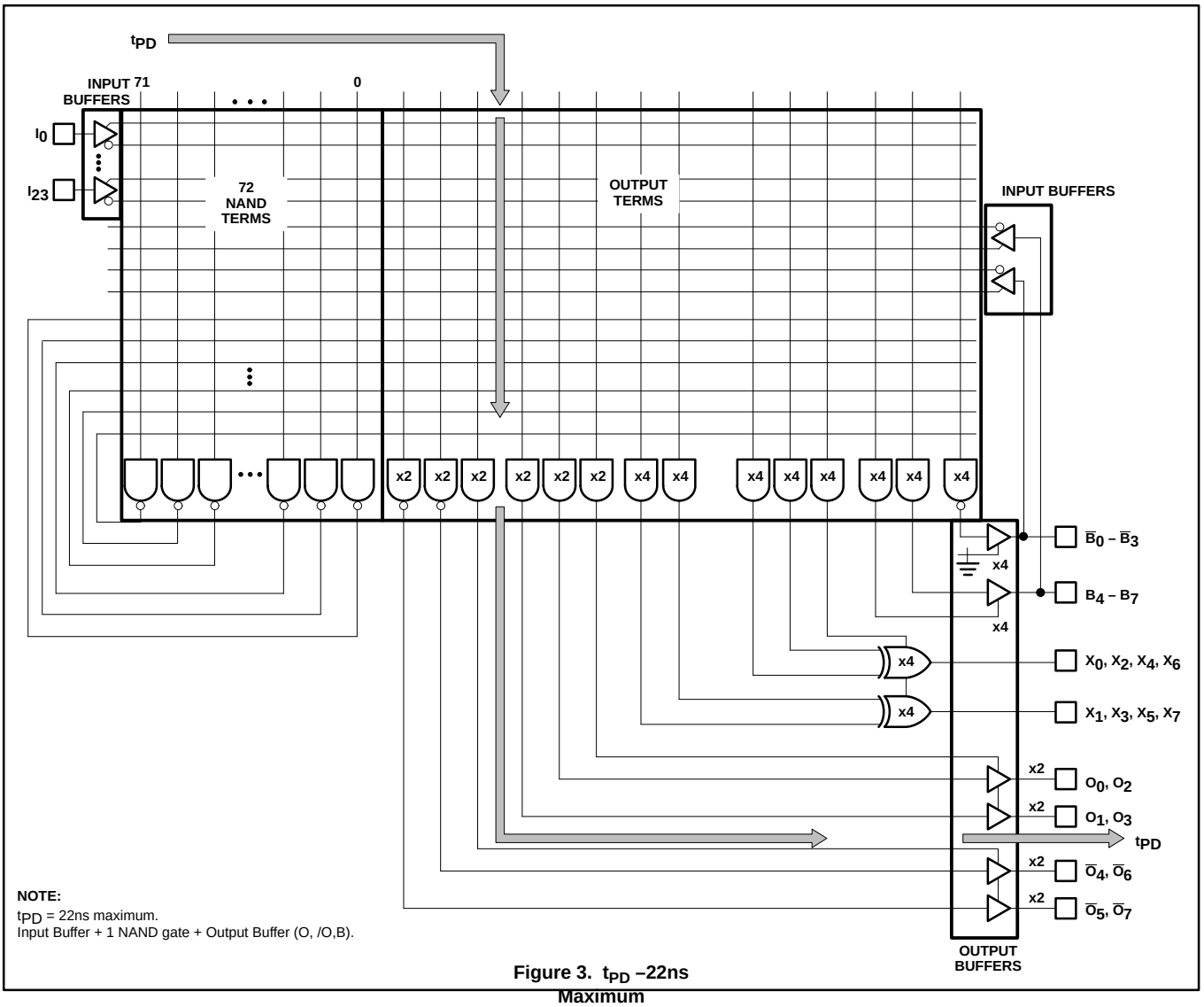


Figure 3. t_{PD} –22ns Maximum

Designing with programmable macro logic

PLHS501 TIMING (Continued)

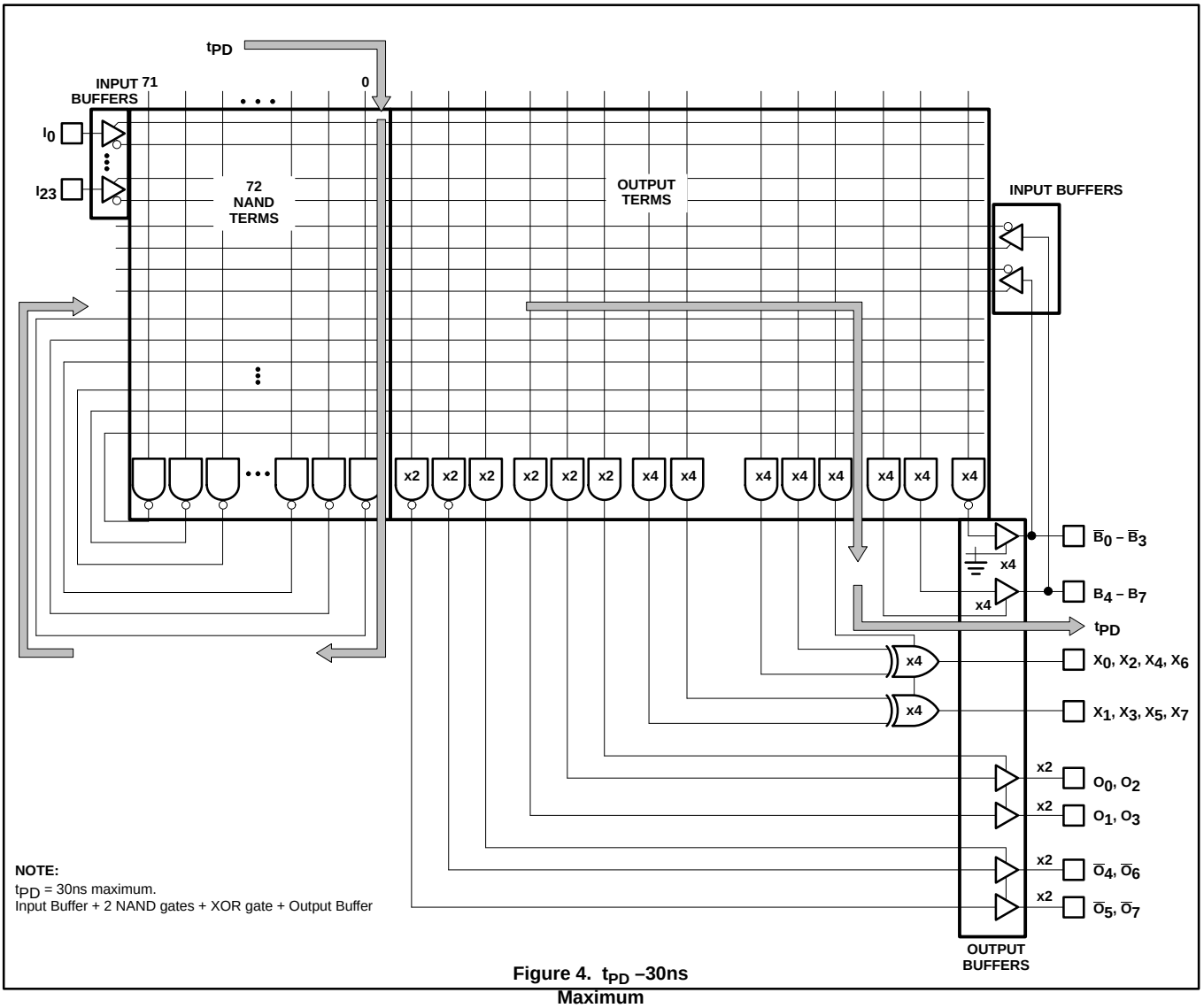


Figure 4. t_{PD} –30ns Maximum

Designing with programmable macro logic

PLHS501 TIMING (Continued)

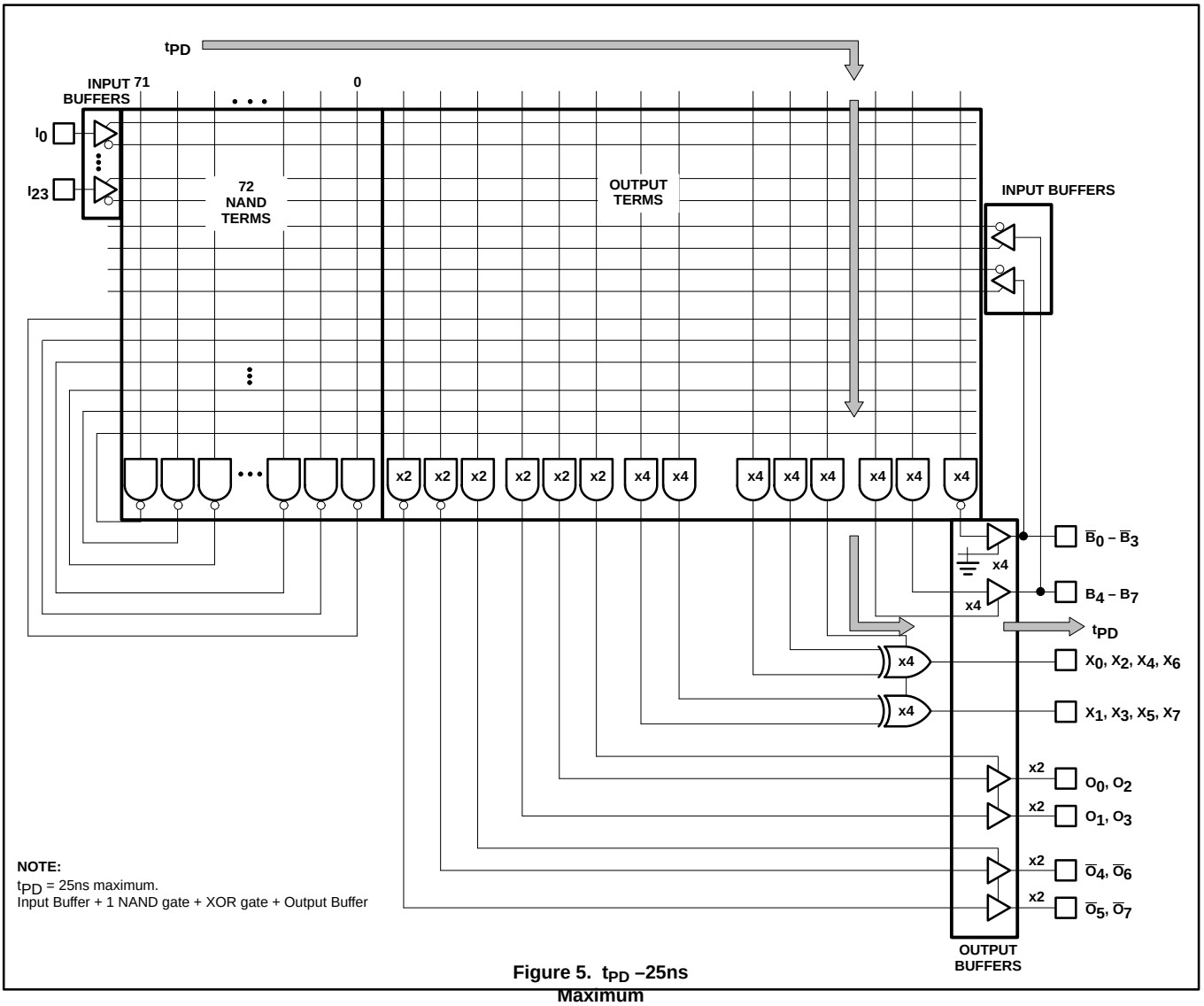


Figure 5. t_{PD} –25ns Maximum

Designing with programmable macro logic

PLHS501 TIMING (Continued)

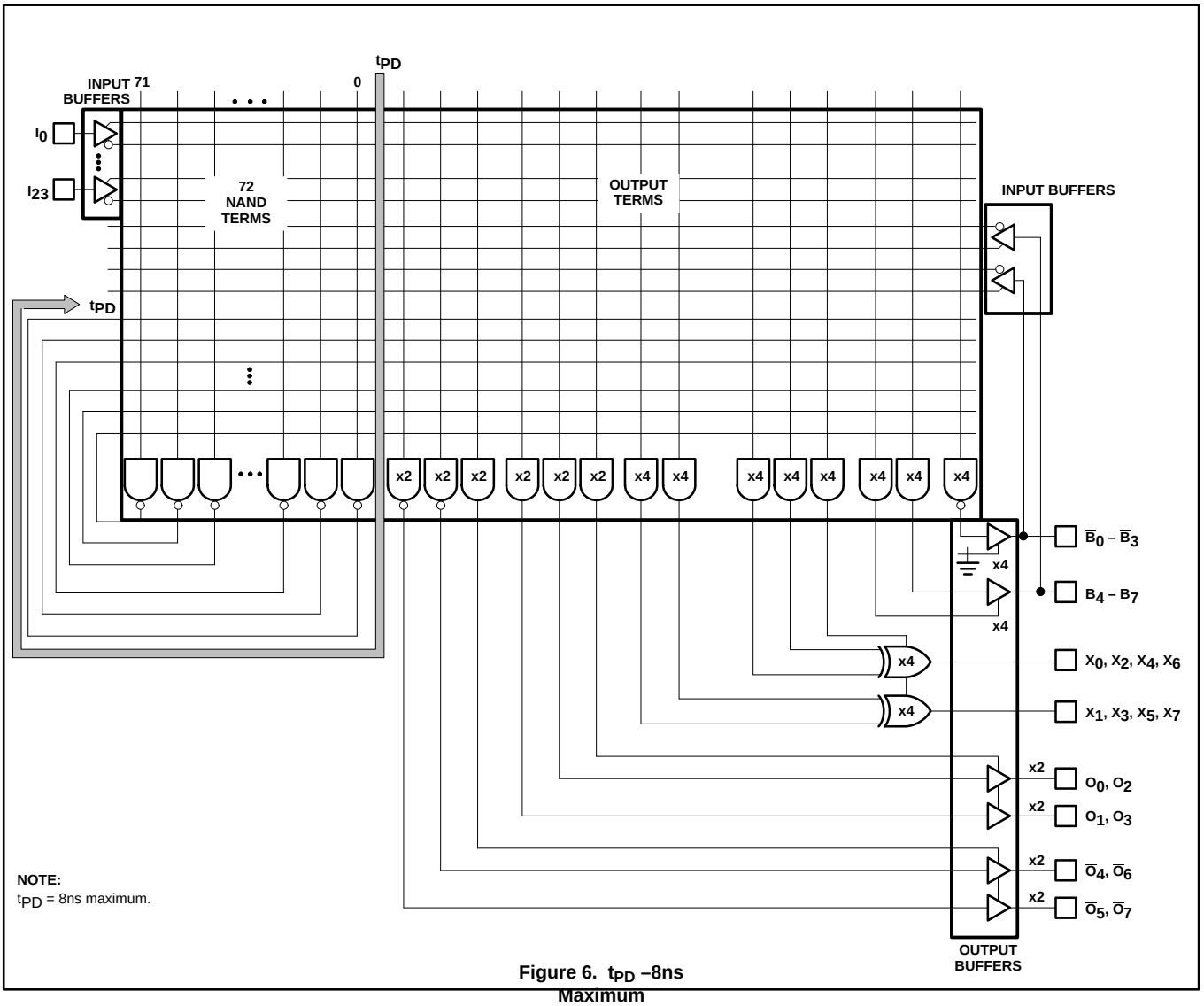


Figure 6. t_{PD} –8ns Maximum

Designing with programmable macro logic

NAND GATE FLIP-FLOPS

Various types of flip-flops and latches may be constructed using the NAND gate building blocks of the PLHS501. A typical 7474 type of edge-triggered D flip-flop requires 6 NAND gates as shown in Figure 7.

No additional gates are required to implement asynchronous set and reset functions to the flip-flop. The equations necessary for SNAP to implement the D flip-flop are shown in Figure 8. However, please note that the equations of Figure 8 define a D flip-flop configured as a divide by 2 (i.e., QN is connected to the data input) whereas Figure 7 shows a general case. Also note that flip-flops with some additional features may be constructed without using more than the six NAND gates. This is possible because of the large number of inputs associated with each NAND gate. For instance, a flip-flop may be required to have a clock gated by one or more signals. Using the PLHS501, it may be implemented by adding additional input signal names to NAND gate equations of gates #2 and #3 of Figure 7. If the data input is to the AND of several signals, extra inputs to NAND gate #4 may be used. Or if additional set or reset lines are required, they may be added simply by using more of the inputs of each NAND gate connected to the main set or reset.

Figure 10 shows two simulations of the same flip-flop. The first one is at a little less than maximum frequency, for clarity in following the waveforms, and the second is at the maximum toggling frequency. For these simulations each NAND gate has a maximum t_{PHL} or t_{PLH} of 8ns (which is the gate delay of a NAND gate in the PLHS501's foldback array). First of all, it can be seen from these simulations that for proper simulation or testing of such a device a set or reset input is mandatory. Both Q and QN outputs are unknown not matter what the inputs do, until they are put into a known state by either a set or reset input. Secondly, various timing parameters such as propagation delay, as well as setup and hold times may be determined.

Therefore, performance of the flip-flop depends a great deal on which gates in the PLHS501 are used, either NAND gates in the foldback array or output NAND gates, connected to bidirectional pins. As a test of the simulation, a D flip-flop connected as a divide by 2 was constructed using only the foldback NAND terms (see Figure 8). An output NAND terms was used to invert the QN output and drive an output buffer. The only inputs were the clock and a reset. The data input to the flop was driven internally by the QN output. According to the simulation, it was possible to drive the clock at a frequency of 25MHz and this small circuit also functioned at that frequency.

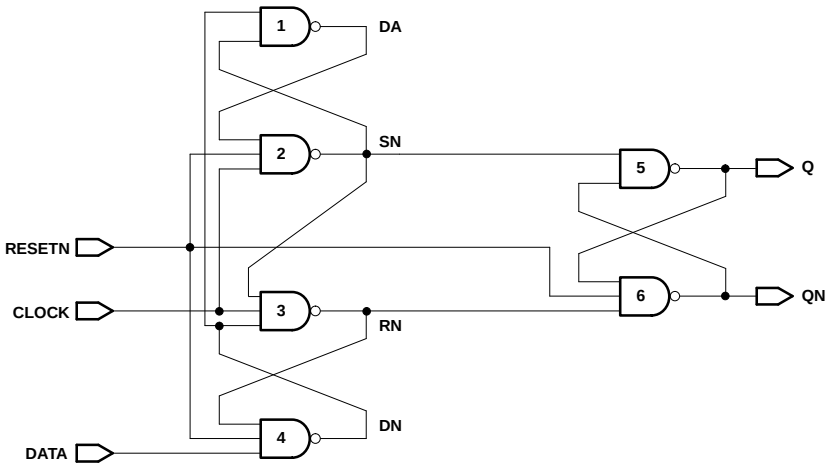


Figure 7. Edge-Triggered D Flip-Flop

Designing with programmable macro logic

```
************************************************************
*       PLHS501 52-Pin PLCC Package Pin Layout             *
* Date: 10/13/93                          Time: 16:42:21 *
************************************************************

                              R C
                              S L
                              T K
                         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         | | | | | | | |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]
                  [ 9|I18                              I4|45]
                  [10|I19                              I3|44]
                  [11|I20                              I2|43]
                  [12|I21                              I1|42]
                  [13|I22                              I0|41]
                  [14|I23                             /B3|40] OUT
                  [15|B4                              /B2|39]
                  [16|B5                              /B1|38]
                  [17|B6                              /B0|37]
                  [18|B7                               X7|36]
                  [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|2|3|3|3|3|
                         |1|2|3|4|5|6|7|8|9|0|1|2|3|
                         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

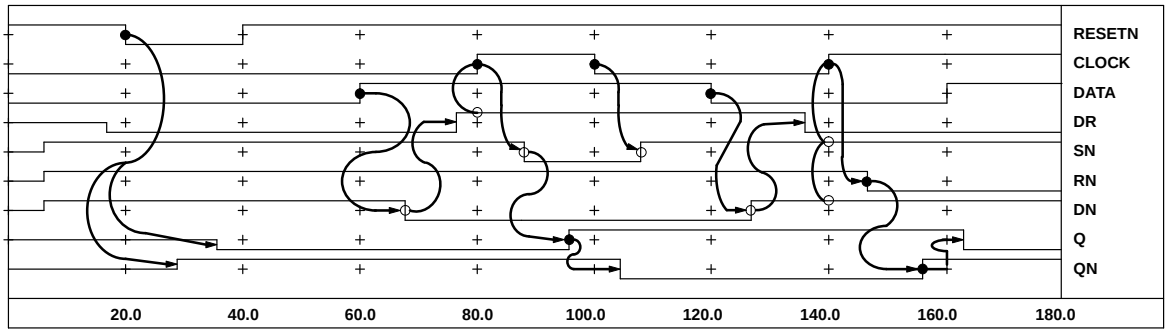
```
@PINLIST
clk i;
rst i;
out o;
@GROUPS
@TRUTHTABLE
@LOGIC EQUATIONS

flop.clk =  clk;
flop.d   = /flop;
flop.rst =  rst;
out      =  flop;

@INPUT VECTORS
@OUTPUT VECTORS
@STATE VECTORS
@TRANSITIONS
```

Figure 8. SNAP Listing

Designing with programmable macro logic

```
PLHS501
77666666666655555555544444444443333333333222222222211111111110000000000
10987654321098765432109876543210987654321098765432109876543210987654321O
IIIIIIIIIIIIIIIIIIIIIIIBBBBBBBB
22221111111111000000000000000000
32109876543210987654321076543210
--------------------------------   Col for   /B 0
.....................................................................A..
HH------------------------------   Col for    P 0
.....................................................................A...
-H------------------------------   Col for    P 1
.....................................................................A...A
H-------------------------------   Col for    P 2
.....................................................................A...A.
--------------------------------   Col for    P 3
.....................................................................A...A
H-------------------------------   Col for    P 4
.....................................................................AA.
--------------------------------   Col for    P 5
.....................................................................A.A
```

Figure 9. Partial PLHS501 Fusemap Showing Test Flip-Flop Fusing



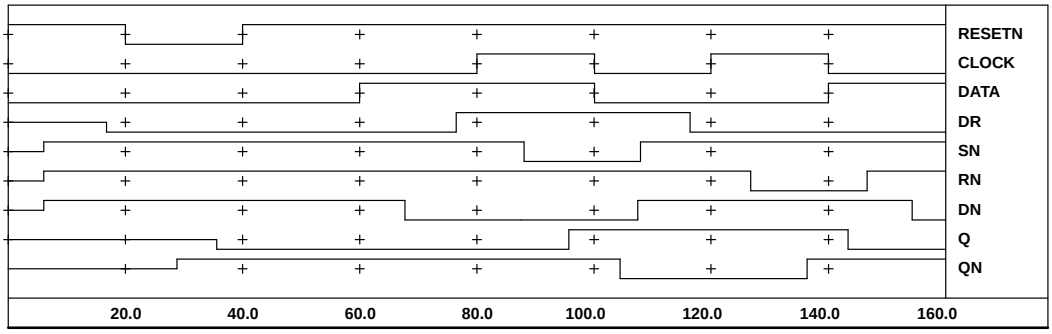


Figure 10. Waveforms of Test Flip-Flop

Designing with programmable macro logic

FUNCTIONAL FIT

In the late 1960's and early 1970's designers used SSI, MSI and small amounts of early LSI to generate logic solutions. Frustrated by the lack of wide input gates to accommodate a lot of product terms for two level solutions, they turned toward the budding ROM and PROM products. These devices relied on literally realizing a function by generating its truth table in silicon. The logic function had to have each logical one and zero realized distinctly as an entry for a particular combination of input variables, usually supplied on the address lines of the memory. Observing that many such truth tables were dense in ones or zeroes and sparse in the remainder, a cadre of initial manufacturers emerged with focus on supplying a programmable product with a few AND gates and OR gates which were versatile enough to compete against the ROM/PROM parts. The gimmick supplied these PLA manufacturers was to illustrate the functional equivalency of the PLA to the PROM by comparing the number of product terms (to be shortened to "p-terms") the PLA supplied and comparing this to the width and depth of available PROMs. P-terms became the "currency" of the PLA world and a designer only had to assess the equivalent number of Boolean product terms required by his function to determine whether a particular PLA was a suitable candidate for his design.

Almost in parallel, gate arrays became available. These provided an array of identical, fixed input gates (usually two input NANDs or NORs). These were generated in a regular fashion on substrate which has a fixed input/output pin arrangement. Also recognizing that all logic functions could be built from the appropriate two input gate, when interconnected correctly, manufacturers offered these devices to customers who required increased density.

The designer's responsibility was to generate what would ultimately be a metal interconnect pattern of his design. Special tools were required to allow an untrained system designer to do this successfully. Flop-flops, decoders, registers, adders, etc., could all be generated from the low level gate building blocks.

The currency of gate arrays became known as gate equivalent functions. That is with limited number of available gates on a substrate, the user needed to know precisely how many gates were used up, on a function by function basis, to generate each piece of his design. A D flip-flop requires about six gates, a D latch four, a 3 to 8 decoder takes about 14 gates and so forth. This allowed estimation regarding whether the function could conceivable be fit onto a particular substrate or not. Manufacturers had to offer multiple foundations to that a designer could be assured that his design would result in a working IC.

The classic method of estimating whether a logic function would fit into a PLA was to determine the number of I/O pads required and the number of product terms required to generate the logical function, then select the PLA. For a gate array, the required measure included the I/O pad arrangement but substituted the number of available gates to generate the logical function (usually by table lookup). In an attempt to reconcile the two measures, Hartman[4] has evolved a formula for his product line. A calculation using this method and developing an appropriate "exchange rate: is shown in Table 2 for the PLHS501 and PLHS502. An alternate method of generating an estimate is to consider the gate equivalent of generating, say for the PLHS501, a gate equivalent of the part in an optimistic functional configuration (72 occurrences of a 32 input NAND gate).

Figure 11 shows how this will result in over 2000 equivalent gates. Conversely, by stacking the NAND gates into D flip-flops, its least efficient function, the PLHS501 will have a gate equivalent of only about 100 gates.

The most rational method of assessing fit is to isolate functions and identify the correct configuration in terms of gates, to allow direct tally of the gates used, to generate the proposed configuration. Table 3 may assist in doing this analysis. Note that all basic gates require precisely one gate to generate the function. Also note the occurrence of functions in the table which could never be generated as standard ICs previously. The procedure is to tally the design against a total budget of 72 multiple input NAND gates.

Table 3 is illustrative only, and should by no means be taken as complete. It may be simply expanded by designing the proposed function with disregard to the usual restrictions on the number of inputs to a gate, realize the function as one, two, three, or more levels of interconnected logic and count the number of gate occurrences required. Special software has been provided to allow pyramided logic structures to be generated under the designer's control. These structures may, however, be no deeper than 72 levels for the PLHS501. Functions should be generated in accord with the guidelines mentioned before, for selecting an optimal 2 level logical solution.

It is an interesting observation that manufacturers of gate arrays and standard cell products which offer embedded PROMs, ROMs or RAMs have not successfully described these embedded functions in terms of equivalent gates, but rather resort to other means (such as divulging their relative area with respect to the area of a basic gate). There is, as yet, no standard in this arena.

Designing with programmable macro logic

Table 2. Equivalency Ratio

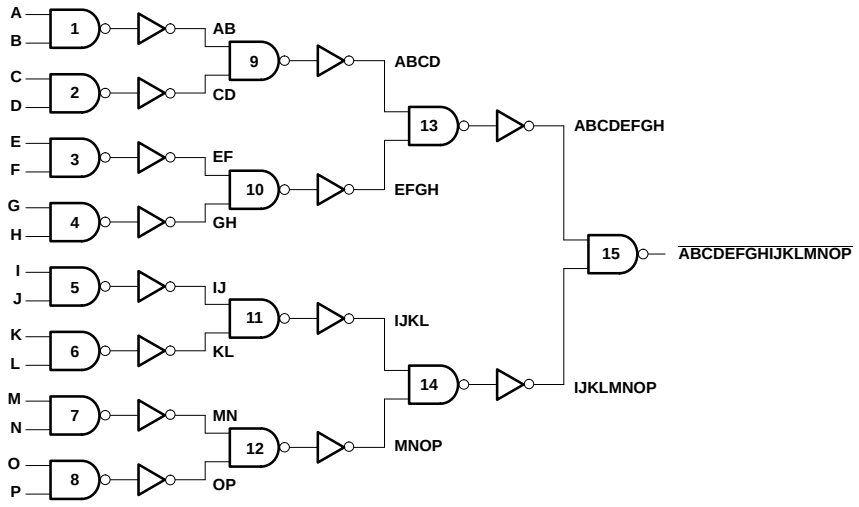
Hartman's method is based on a CMOS gate array equivalency wherein 4 transistors constitute a 2 input NAND or NOR gate, equal to one gate. Thus, his "exchange rate" is as follows:

E.R. = $4 \times$ # inputs
$+9 \times$ # FFs
$+7 \times$ # 3-State outputs
$+(15 \text{ to } 30) \times$ # OR outputs from the AND/OR array.

For the PLHS501: (using CMOS numbers which may be inappropriate)

E.R. = 4×32
$+9 \times 0$
$+7 \times 24$
$+(15 \text{ to } 30) \times 50\%$ of 72 feedbacks = 836 to 1376 gates

Being for two bipolar ICs, in this case, the method may be inappropriate, but may be taken as an estimating procedure.



NOTE:
Double this and add one for a 32 input NAND.

Figure 11. 16 Input NAND Formed from 2 Input Gates

Designing with programmable macro logic

Table 3. PLHS501 Gate Count Equivalents

FUNCTION	INTERNAL NAND EQUVAL-ENT	COMMENTS
Gates		
NANDs	1	For 1 to 32 input variables
ANDs	1	For 1 to 32 input variables
NORs	1	For 1 to 32 input variables
ORs	1	For 1 to 32 input variables
Decoders		
3-to-8	8	Inverted inputs available
4-to-16	16	Inverted inputs available
5-to-32	32	Inverted inputs available (24 chip outputs only)
Encoders		
8-to-3	15	Inverted inputs, 2 logic levels
16-to-4	32	Inverted inputs, 2 logic levels
32-to-5	41	Inverted inputs, 2 logic levels, factored solution.
Multiplexers		
4-to-1	5	Inverted inputs available
8-to-1	9	
16-to-1	17	
27-to-1	28	Can address only 27 external inputs - more if internal
Flip-Flops		
D-type Flip-Flop	6	With asynchronous S-R
T-type Flip-Flop	6	With asynchronous S-R
J-K-type Flip-Flop	10	With asynchronous S-R
Adders		
8-bit	45	Full carry-lookahead (four levels of logic)
Barrel Shifters		
8-bit	72	2 levels of logic
Latches		
D-latch	3	2 levels of logic with one shared gate