NAND3_X1

Databook Build Date: Thursday Feb 17 15:07 2011

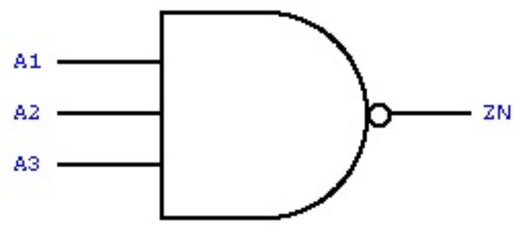
Conditions for characterization library **NangateOpenCellLibrary**, corner **NangateOpenCellLibrary_typical_typical**: Vdd= **1.10**V, Tj= **25.0** deg. C .

Additional corners: NangateOpenCellLibrary [NangateOpenCellLibrary_slow_slow], NangateOpenCellLibrary [NangateOpenCellLibrary_fast_fast], NangateOpenCellLibrary [NangateOpenCellLibrary_worst_low_worst_low], NangateOpenCellLibrary [NangateOpenCellLibrary_low_temp_low_temp].

Output transition is defined from **30%** to **70%** (rising) and from **70%** to **30%** (falling) output voltage.

Propagation delay is measured from **50%** (input rise) or **50%** (input fall) to **50%** (output rise) or **50%** (output fall).

Strength	1
Cell Area	1.064 um^2
Equation	ZN = "!((A1 & A2) & A3)"
Type	Combinational
Input	A1, A2, A3
Output	ZN
PG Pins	VDD (primary_power), VSS (primary_ground)



State Table			
A1	**A2**	**A3**	**ZN**
L	-	-	H
H	H	H	L
-	L	-	H
-	-	L	H

Propagation Delay [ns]					
Input Transition [ns]		0.0012		0.1985	
Load Capacitance [fF]		0.3656	58.3649	0.3656	58.3649
A1 to ZN	fall	0.01	0.18	0.02	0.26
	rise	0.01	0.15	0.04	0.25
A2 to ZN	fall	0.01	0.18	0.03	0.25
	rise	0.01	0.15	0.05	0.25
A3 to ZN	fall	0.01	0.19	0.02	0.23
	rise	0.01	0.15	0.06	0.26

Output Transition [ns]					
Input Transition [ns]		0.0012		0.1985	
Load Capacitance [fF]		0.3656	58.3649	0.3656	58.3649
A1 to ZN	fall	0.01	0.15	0.04	0.16
	rise	0.01	0.14	0.04	0.15
A2 to ZN	fall	0.01	0.15	0.03	0.16
	rise	0.01	0.14	0.03	0.15
A3 to ZN	fall	0.01	0.15	0.03	0.16
	rise	0.01	0.14	0.03	0.15

Capacitance [fF]	
A1	1.5903
A2	1.6212
A3	1.6504

Leakage [nW]
18.10

Dynamic Power Consumption [uW/GHz]					
Input Transition [ns]		0.0012		0.1985	
Load Capacitance [fF]		0.3656	58.3649	0.3656	58.3649
A1 to ZN	fall	0.52	0.56	3.28	1.11
	rise	2.47	2.46	5.19	3.45
A2 to ZN	fall	0.53	0.56	2.42	0.82
	rise	3.20	3.01	5.85	4.37
A3 to ZN	fall	0.53	0.56	2.25	0.80
	rise	3.77	3.83	6.61	4.88

NANGATE NAND3_X2

Databook Build Date: Thursday Feb 17 15:07 2011

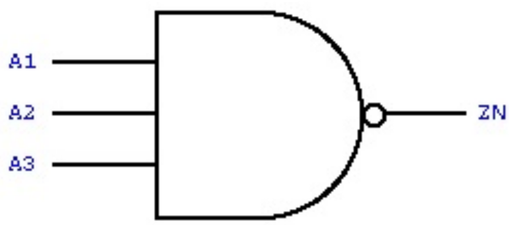
Conditions for characterization library **NangateOpenCellLibrary**, corner **NangateOpenCellLibrary_typical_typical**: Vdd= **1.10**V, Tj= **25.0** deg. C .

Additional corners: NangateOpenCellLibrary [NangateOpenCellLibrary_slow_slow], NangateOpenCellLibrary [NangateOpenCellLibrary_fast_fast], NangateOpenCellLibrary [NangateOpenCellLibrary_worst_low_worst_low], NangateOpenCellLibrary [NangateOpenCellLibrary_low_temp_low_temp].

Output transition is defined from **30%** to **70%** (rising) and from **70%** to **30%** (falling) output voltage.

Propagation delay is measured from **50%** (input rise) or **50%** (input fall) to **50%** (output rise) or **50%** (output fall).

Strength	2
Cell Area	1.862 um^2
Equation	ZN = "!((A1 & A2) & A3)"
Type	Combinational
Input	A1, A2, A3
Output	ZN
PG Pins	VDD (primary_power), VSS (primary_ground)



State Table			
A1	**A2**	**A3**	**ZN**
L	-	-	H
H	H	H	L
-	L	-	H
-	-	L	H

Propagation Delay [ns]					
Input Transition [ns]		0.0012		0.1985	
Load Capacitance [fF]		0.3656	116.272	0.3656	116.272
A1 to ZN	fall	0.01	0.18	0.02	0.26
	rise	0.01	0.15	0.04	0.25
A2 to ZN	fall	0.01	0.18	0.03	0.25
	rise	0.01	0.15	0.05	0.25
A3 to ZN	fall	0.01	0.19	0.02	0.23
	rise	0.01	0.15	0.06	0.26

Output Transition [ns]					
Input Transition [ns]		0.0012		0.1985	
Load Capacitance [fF]		0.3656	116.272	0.3656	116.272
A1 to ZN	fall	0.01	0.15	0.04	0.16
	rise	0.01	0.14	0.04	0.15
A2 to ZN	fall	0.01	0.15	0.03	0.16
	rise	0.01	0.14	0.03	0.15
A3 to ZN	fall	0.01	0.15	0.03	0.16
	rise	0.01	0.14	0.03	0.15

Capacitance [fF]	
A1	2.9778
A2	3.2864
A3	3.5589

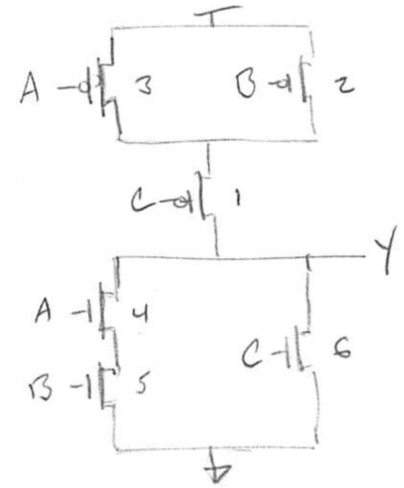
Leakage [nW]
36.21

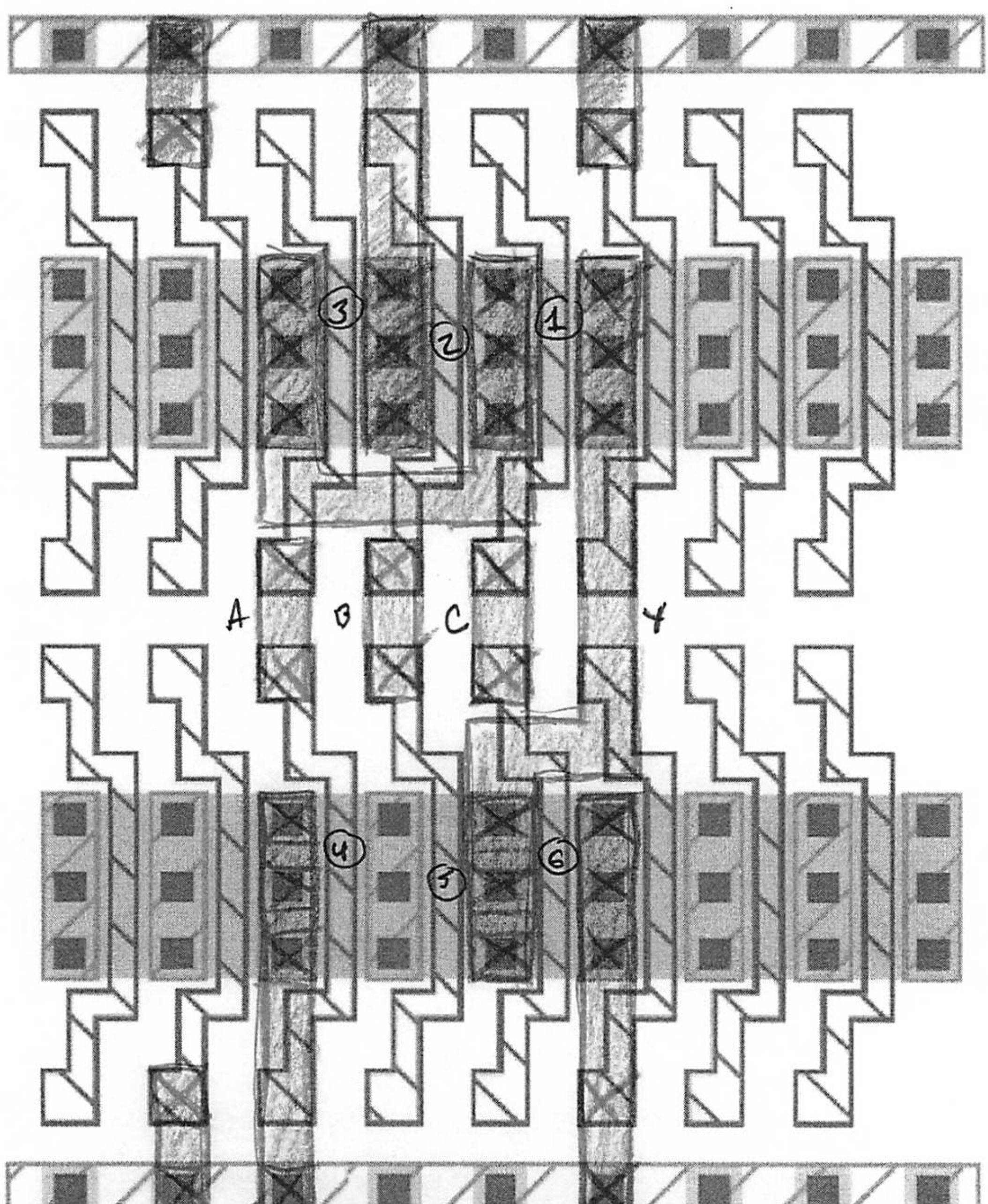
Dynamic Power Consumption [uW/GHz]					
Input Transition [ns]		0.0012		0.1985	
Load Capacitance [fF]		0.3656	116.272	0.3656	116.272
A1 to ZN	fall	1.24	1.31	6.74	2.40
	rise	5.11	5.07	10.57	7.05
A2 to ZN	fall	1.24	1.31	5.03	1.83
	rise	6.57	6.85	11.85	8.89
A3 to ZN	fall	1.24	1.31	4.68	1.78
	rise	7.72	8.20	13.41	10.20

ACTIVITY

USE A SEA OF GATES APPROACH TO IMPLEMENT THE FOLLOWING LOGIC FUNCTION

$$F = \overline{(A \cdot B) + C}$$





Programming a PROM (slide 34)

First manipulate function so each term in sum-of-products includes all three inputs

$$f_0 = x_0 x_1 + \bar{x}_2$$

SAME

$$= x_2 x_1 x_0 + \bar{x}_2 x_1 x_0 + \bar{x}_2 x_1 x_0 + \bar{x}_2 \bar{x}_1 x_0 + \bar{x}_2 x_1 \bar{x}_0 + \bar{x}_2 \bar{x}_1 \bar{x}_0$$

$$= x_2 x_1 x_0 + \bar{x}_2 x_1 x_0 + \bar{x}_2 \bar{x}_1 x_0 + \bar{x}_2 x_1 \bar{x}_0 + \bar{x}_2 \bar{x}_1 \bar{x}_0$$

each term corresponds to one connection made in the PROM (need 5 connections)

SAME

$$f_1 = x_2 x_1 x_0 + \bar{x}_2 x_1 x_0 + \bar{x}_2 x_1 \bar{x}_0 + \bar{x}_2 \bar{x}_1 x_0 + \bar{x}_2 \bar{x}_1 \bar{x}_0 + x_2 x_1 \bar{x}_0 + \bar{x}_2 x_1 \bar{x}_0$$

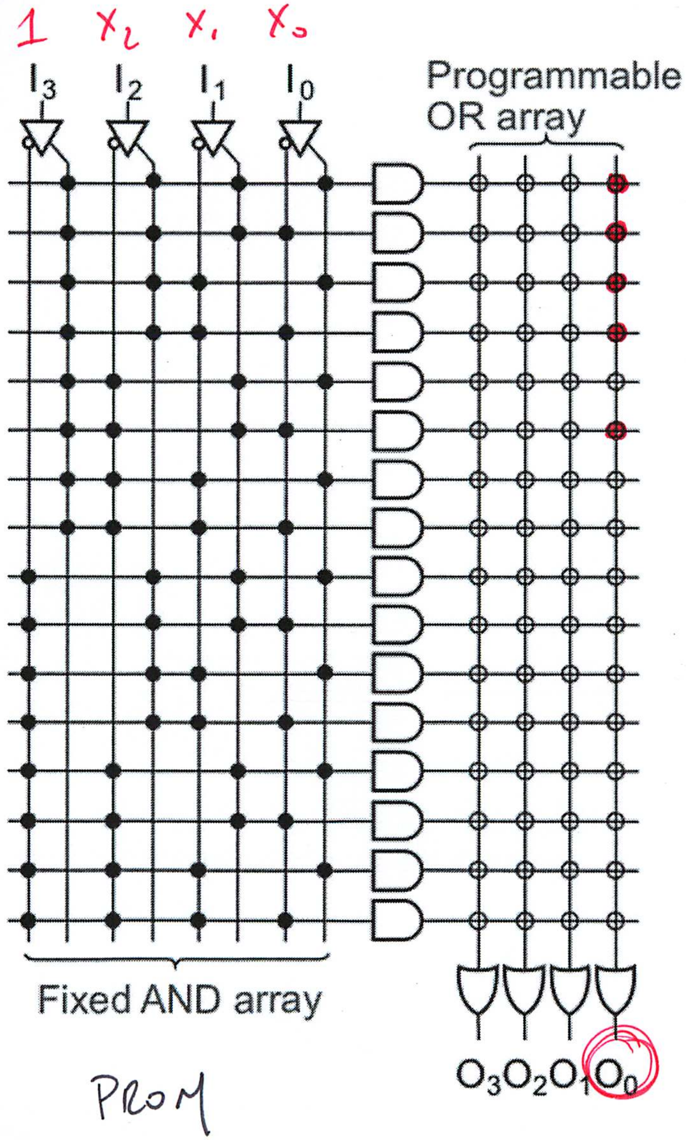
$$= \underbrace{x_2 x_1 x_0 + \bar{x}_2 x_1 x_0 + \bar{x}_2 x_1 \bar{x}_0 + \bar{x}_2 \bar{x}_1 x_0 + \bar{x}_2 \bar{x}_1 \bar{x}_0}_{\text{SAME AS } f_0} + x_2 x_1 \bar{x}_0$$

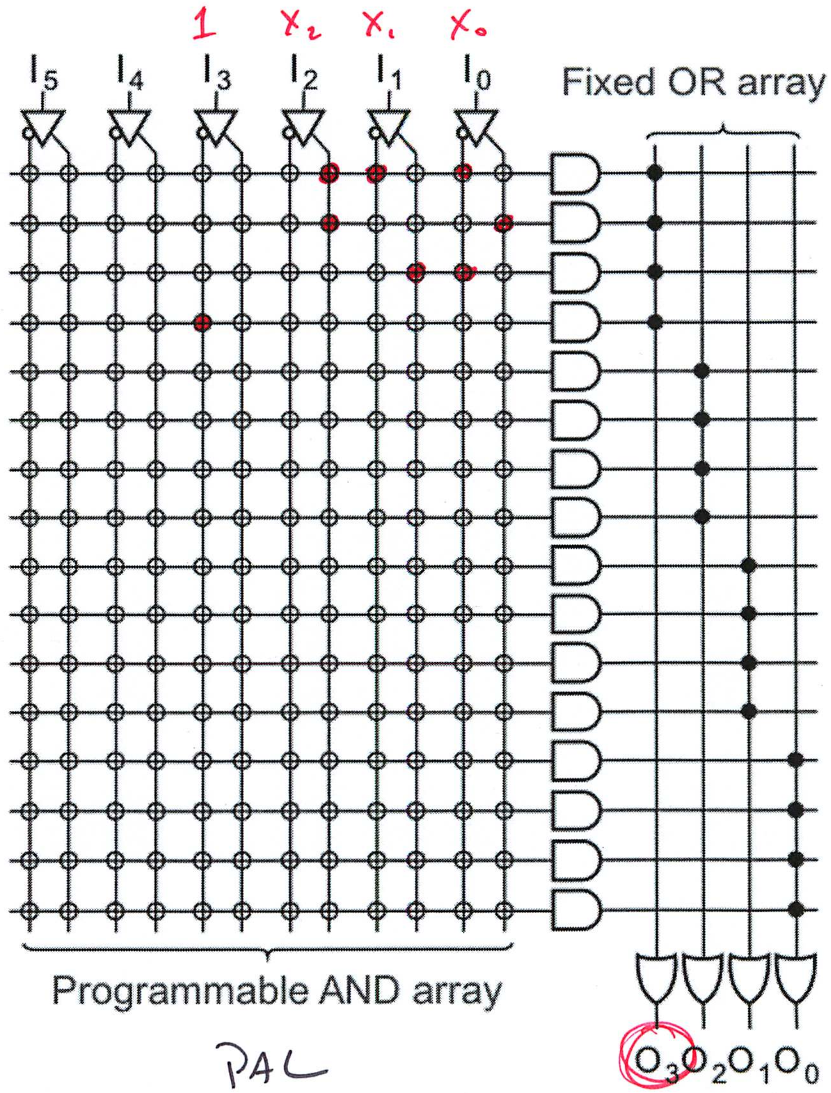
TOTAL CONNECTIONS = 6

need to add this connection

For PAL write in canonical product of sums form, no need to expand out each term because can leave specific inputs unconnected. Limited by fixed number of clauses (term) in NOR plane

PLA offers increased flexibility but lower density and performance compared to PROM or PAL





$$f = \bar{x}_0 \bar{x}_1 x_2 + x_0 x_2 + \bar{x}_0 x_1$$

```
==========================================================================
Follow up on PAL/PROM/PLA Activity
==========================================================================
```

The boolean equation we want to map to the PROM and PAL is:

f = x0'x1'x2 + x0x2 + x0'x1

For the PROM, we can expand this boolean equation into the canonical sum-of-minterms form. With three literals, a minterm is a product term (i.e., the AND of literals) of all three literals (i.e., the complemented or uncomplemented version of every literal is in the product term). So the sum-of-minterms for this boolean equation is:

f = x0'x1'x2 + x0x1x2 + x0x1'x2 + x0'x1x2 + x0'x1x2'

Each minterm corresponds to one row in the PROM. We simply connect the appropriate rows to a single OR gate in the OR plane. It is kind of like we are directly implementing the truth-table with the PROM. There is one row in the PROM for every row in the truth-table where the output is one.

For the PAL, we can leave the boolean equation in its original sum-of-products form. The original form is "sum-of-products" but not "sum-of-minterms" because each product term is not a minterm (i.e., each product term in the original boolean equation does not include all three inputs). Each product term in the original boolean equation corresponds to one row in the PAL. We simply connect the desired complement or uncomplemented version of the inputs we need to a row in the AND plane. Since there are three product terms in the original boolean equation, we need three rows. In general, there is no reason we couldn't map the sum-of-minterms form to a PAL as well; but this would require five rows and the PAL in this example only supports up to four inputs into a single OR gate so it wouldn't work for this specific PAL. By using a reduced version of the boolean logic equation we need less rows.

Note that the original boolean equation is actually _not_ a "minimal covering". A minimal covering is a sum-of-products boolean equation where each product term is a "prime implicant". A prime implicant is a product term that cannot be covered by a more general implicant (i.e., the product term cannot be reduced to include fewer literals). We can further minimize the original boolean logic equation. To see how we can use Karnaugh maps (which everyone learned about in ECE 2300). From the canonical sum-of-minterms form, we know there will be five ones in the K-map:

```
         x0x1
      -----------
      00 01 11 00
     +------------
 x2 0| 0  1  0  0
    1| 1  1  1  1
```

So we can cover the ones with two prime implicants (represented with a and b below):

```
         x0x1
      -----------
      00 01 11 00
     +------------
 x2 0| 0  a  0  0
    1| b  b  b  b
```

So the following is a minimal covering:

```
  f = x2 + x0'x1x2'
```

So given this minimal covering we can now implement the original boolean logic equation with just _two_ instead of _three_ rows in the PAL.

I also wanted to clarify how we might actually implement a PROM, PAL, PAL. The slides use an abstract representation with AND/OR gates, but a real implementation will _not_ use static CMOS AND/OR gates with so many inputs. PROM, PAL, and PLAs are usually implemented using a dynamic-logic style which is one reason these structures can be faster than logic synthesized out of static CMOS gates. Figure 12.78 in Weste & Harris (now included in the slides) illustrates how this works. We use dynamic NOR gates for both the AND and OR plane. We also need to add inverters to the input and outputs to make the two stages of NOR gates logically equivalent to just the AND/OR gates:

```
  --I>o--|NOR
         |NOR---|NOR
  --I>o--|NOR   |NOR
                |NOR--I>o--
  --I>o--|NOR   |NOR
         |NOR---|NOR
  --I>o--|NOR
```

So if we push the inverter bubbles into the first stage of NOR gates, they turn into AND gates and the final inverter means the second stage is really an OR gate.

So if you look at Figure 12.78 you can see how this works. Note that Figure 12.78 is not really for a "programmable" PLA but is for a PLA that is configured at design time, but it is the same idea as what we might use in a programmable PLA. We precharge the horizontal buses in the AND plane and then if any of the inputs are one they turn on the pull-down NMOS and cause the horizontal bus to discharge. This bus is directly connected to the gate of a pull-down NMOS in the OR plane. The vertical buses in the OR plane are also precharged and if the output from the AND plane is one it will turn on the pull-down NMOS in the OR plane and cause the vertical bus to discharge. Note that normally with dynamic logic we never directly connect the output of a dynamic node to the input of the next dynamic gate -- if we are not careful the horizontal buses could turn on the pull-down NMOS in the OR plane before the pull-down NMOS in the AND plane has had time to fully discharge the horizontal bus. This is why in domino logic we always have an inverter on the output of every dynamic gate. To avoid the delay of this inverter, Weste & Harris describe how we can use carefully crafted control signals to ensure that we don't precharge the vertical buses in the OR plane until the AND plane has fully evaluated (i.e., and horizontal buses that are supposed to be zero have fully discharged).