Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher:	Robert Dick
Office:	L477 Tech
Email:	dickrp@northwestern.edu
Phone:	847–467–2298

NORTHWESTERN UNIVERSITY

Today's topics

- Can use today's class for Q&A
- PALs/PLAs
- Review, Q&A on MOS transistors
- Multiplexers, Demultiplexers
- Transmission gates
- Perl/Python

Lab two

- Lab two is more substantial than lab one
- If you find a problem and figure out the solution, yourself, please send me an email or post to the newsgroup
- Don't think you'll be able to fully understand the effects all the SIS commands, options, and sequences will have
- If you have a basic understanding of how to get reasonably good results with the software, that's good

Midterm exam

Suggesting 21 or 23 October

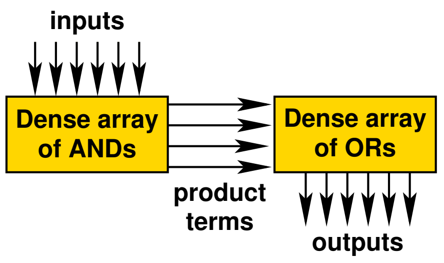
Programmable arrays of logic gates

- We have considered implementing Boolean functions using discrete logic gates
  - NOT, AND, OR, NAND, NOR, XOR, and XNOR
- Can arrange AND and OR gates (or NAND and NOR gates) into a general array structure
- Program array to implement logic functions
- Two popular variants
  - Programmable logic arrays (PLA) and programmable array logic (PAL)

PALs and PLAs

- Pre-fabricated building block of many AND and OR (or NAND and NOR) gates
- "Personalized" (programmed) by making or breaking connections among the gates

SOP programmable array block diagram



PLAs efficiency

- PLAs can share terms – Share product terms
- Consider the following set of functions

$f_0 = a + \overline{b}\,\overline{c}$

$f_1 = a\overline{c} + ab$

$f_2 = \overline{b}\,\overline{c} + ab$

$f_3 = \overline{b}\,c + a$

Personality matrix

Product	Input			Output			
term	a	b	c	f_0	f_1	f_2	f_3
ab	1	1	X	0	1	1	0
$\overline{b}\,c$	X	0	1	0	0	0	1
$a\overline{c}$	1	X	1	0	1	0	0
$\overline{b}\,\overline{c}$	X	0	0	1	0	1	0
a	1	X	X	1	0	0	1

PLA programming

All connections available

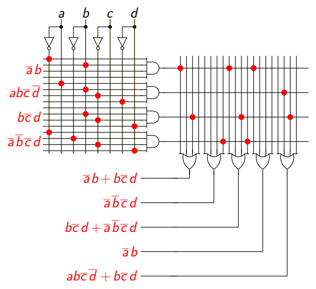
All exist

Some removed

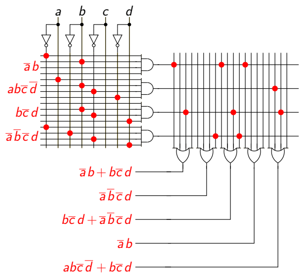
None exist

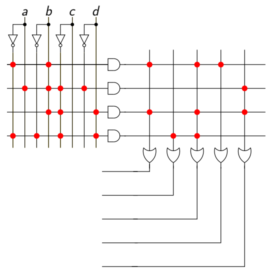
Connections made

PLA programming

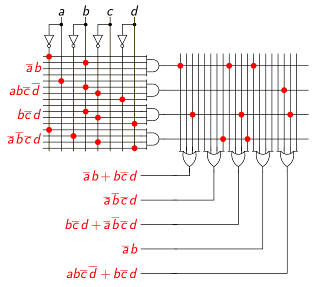


PLA diagram shorthand





Shorthand – Draw subset of wires



PAL/PLA differences

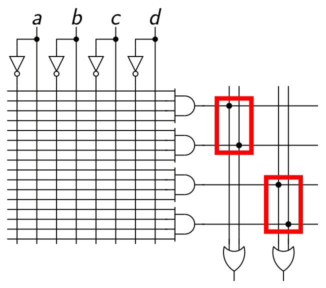
PAL

- Only the AND array is programmable
- A column of the OR array only has access to a subset of the product terms
- Generally, no sharing of product terms

PLA

- A column has access to any desired product terms
- Can share product terms

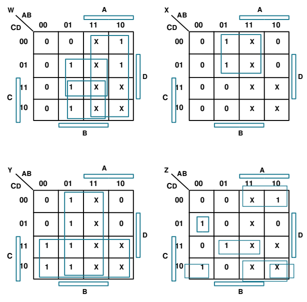
PAL/PLA differences



BCD-Gray code converter

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

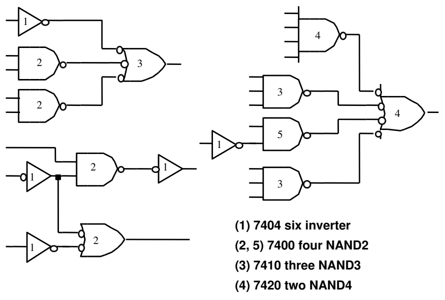
BCD-Gray code converter



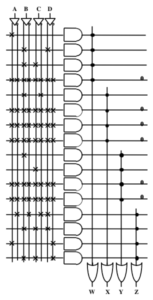
Minimized BCD-Gray functions

$$W = A + BD + BC$$
$$X = B\overline{C}$$
$$Y = B + C$$
$$Z = \overline{A}\,\overline{B}\,\overline{C}\,D + BCD + A\overline{D} + \overline{B}\,C\,\overline{D}$$

BCD-Gray discrete logic



BCD-Gray PAL



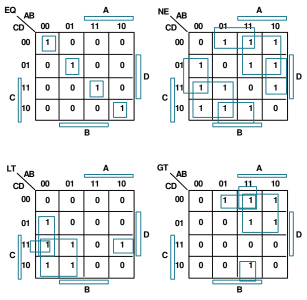
Comparator example

Determine whether a the first two-bit number (AB) is

- Equal to (EQ),
- Not equal to (NE),
- Less than (LT),
- Or greater than (GT)

a second two-bit number (CD)

Comparator Karnaugh map



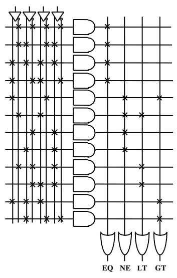
EQ CD \ AB	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

NE CD \ AB	00	01	11	10
00	0	1	1	1
01	1	0	1	1
11	1	1	0	1
10	1	1	1	0

LT CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

GT CD \ AB	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

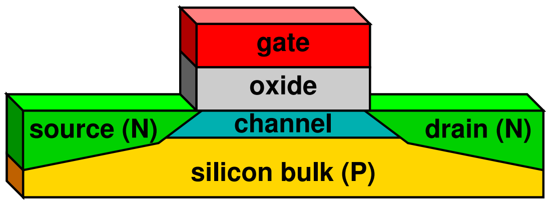
Comparator PLA



Transistors

- Basic device in NMOS and PMOS (CMOS) technologies
- Can be used to construct any logic gate

NMOS transistor



NMOS transistor

Metal–oxide semiconductor (MOS)

- Then, it was polysilicon–oxide semiconductor
- Now, it is MOS again

P-type bulk silicon doped with positively charged ions

N-type diffusion regions doped with negatively charged ions

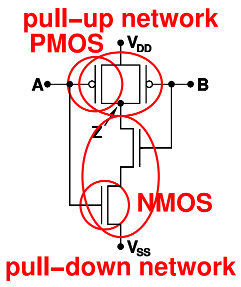
Gate can be used to pull a few electrons near the oxide

Forms channel region, conduction from source to drain starts

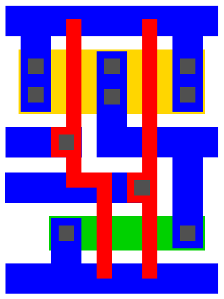
CMOS

- NMOS turns on when the gate is high
- PMOS just like NMOS, with N and P regions swapped
- PMOS turns on when the gate is low
- NMOS good at conducting low (0s)
- PMOS good at conducting high (1s)
- Use NMOS and PMOS transistors together to build circuits
  - Complementary metal oxide silicon (CMOS)

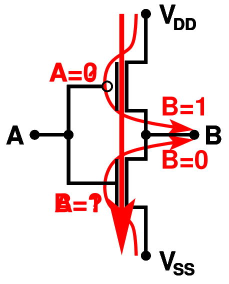
CMOS NAND gate



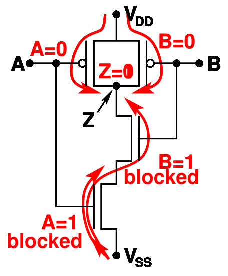
CMOS NAND gate layout



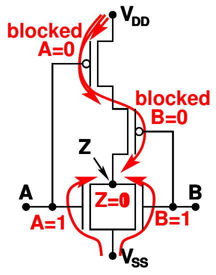
CMOS inverter operation



NAND operation



NOR operation



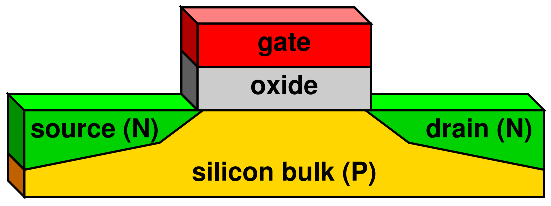
CMOS inefficient for ANDs/ORs

- Recall that NMOS transmits low values easily. . .
- . . . transmits high values poorly
- PMOS transmits high values easily. . .
- . . . transmits low values poorly

CMOS inefficient for ANDs/ORs

- V_T, or threshold voltage, is commonly 0.7 V
- NMOS conducts when $V_{GS} > V_T$
- PMOS conducts when $V_{GS} < -V_T$
- What happens if an NMOS transistor's source is high?
- Or a PMOS transistor's source is low?
- Alternatively, if one states that $V_{TN} = 0.7$ V and $V_{TP} = -0.7$ V then NMOS conducts when $V_{GS} > V_{TN}$ and PMOS conducts when $V_{GS} < V_{TP}$

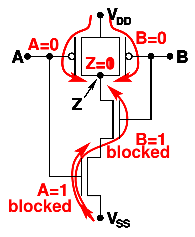
NMOS transistor



CMOS inefficient for ANDs/ORs

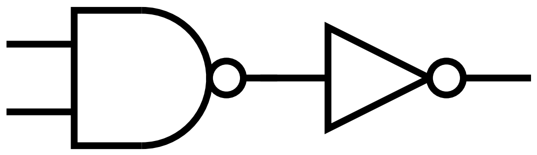
- If an NMOS transistor's input were V_{DD} (high), for $V_{GS} > V_{TN}$, the gate would require a higher voltage than V_{DD}
- If an PMOS transistor's input were V_{SS} (low), for $V_{GS} < V_{TP}$, the gate would require a lower voltage than V_{SS}

Implications of using CMOS



NAND/NOR easy to build in CMOS

Implications of using CMOS



AND/OR requires more area, power, time

CMOS transmission gates (switches)

NMOS is good at transmitting 0s

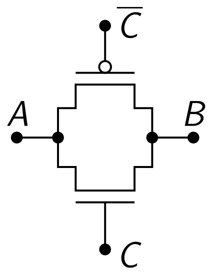
Bad at transmitting 1s

PMOS is good at transmitting 1s

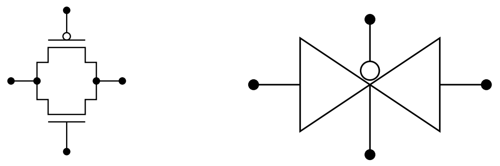
Bad at transmitting 0s

To build a switch, use both: CMOS

CMOS transmission gate (TG)



Other TG diagram



Multiplexer (MUX) definitions

- Also called *selectors*
- 2^n inputs
- n control lines
- One output

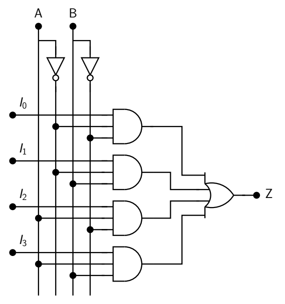
MUX functional table

C	Z
0	I_0
1	I_1

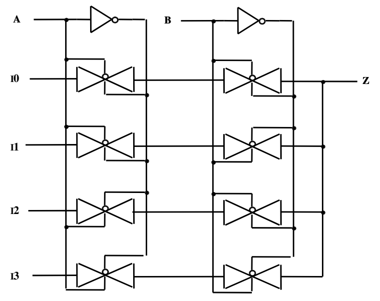
MUX truth table

I_1	I_0	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

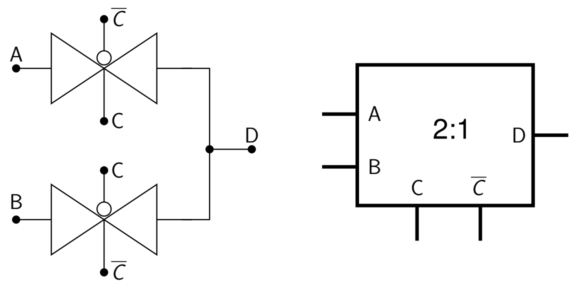
MUX using logic gates



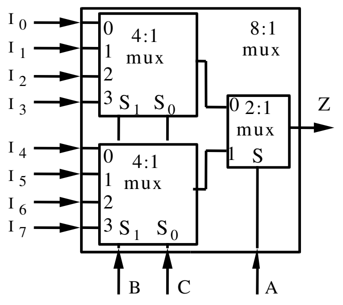
MUX using TGs



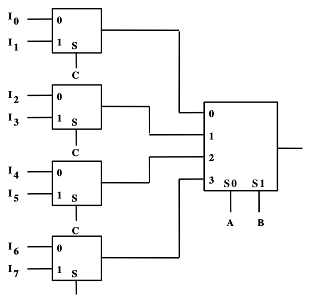
MUX



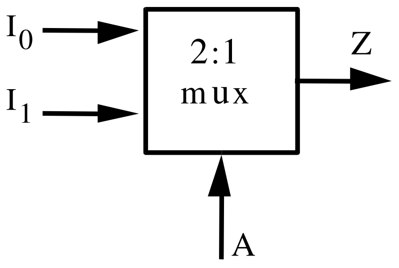
Hierarchical MUX implementation



Alternative hierarchical MUX implementation

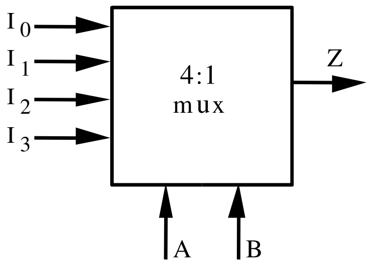


MUX examples



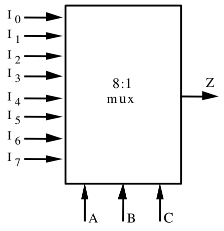
$$Z = \overline{A}\, I_0 + A I_1$$

MUX examples



$$Z = \overline{A}\,\overline{B}\, I_0 + \overline{A}\, B I_1 + A\overline{B}\, I_2 + ABI_3$$

MUX examples



$$Z = \overline{A}\,\overline{B}\,\overline{C}\, I_0 + \overline{A}\,\overline{B}\, CI_1 + \overline{A}\, B\overline{C}\, I_2 + \overline{A}\, BCI_3 + \\ A\overline{B}\,\overline{C}\, I_4 + A\overline{B}\, CI_5 + AB\overline{C}\, I_6 + ABCI_7$$

MUX properties

- A $2^n : 1$ MUX can implement any function of n variables
- A $2^{n-1} : 1$ can also be used
  - Use remaining variable as an input to the MUX

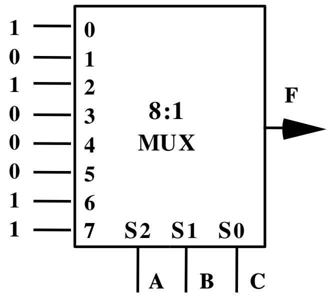
MUX example

$$F(A, B, C) = \sum(0, 2, 6, 7) \\ = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\, B\overline{C} + AB\overline{C} + ABC$$

Truth table

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Lookup table implementation



MUX example

$$F(A, B, C) = \sum(0, 2, 6, 7) \\ = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\, B\overline{C} + AB\overline{C} + ABC$$

Therefore,

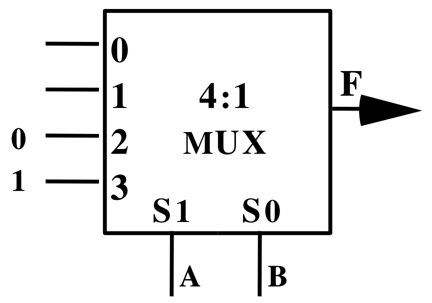
$$\overline{A}\,\overline{B} \rightarrow F = \overline{C}$$
$$\overline{A}\, B \rightarrow F = \overline{C}$$
$$A\overline{B} \rightarrow F = 0$$
$$AB \rightarrow F = 1$$

Truth table

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$F=\overline{C}$

Lookup table implementation



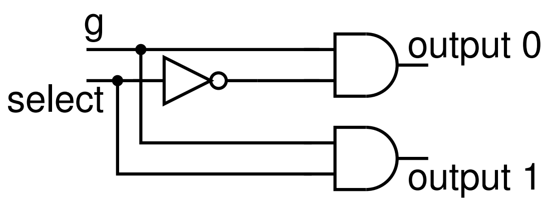
Demultiplexer (DMUX) definitions

- Closely related to *decoders*
- n control signals
- Single data input can be routed to one of 2^n outputs

Decoders vs. demultiplexers

- Decoders have n inputs and 2^n outputs.
- They activate only the output indicated by the binary input value.
- Demultiplexers have one input, 2^n select lines, and 2^n outputs.
- They route the input to the output indicated by the binary select value, and inactivate the other outputs.
- In practice, decoders have an *output enable* input.
- If you treat a decoder output enable as a demultiplexer input and treat the decoder inputs as demultiplexer select lines, the two are equivalent.
- In practice decoders and demultiplexers are interchangeable.

Active high 1:2 decoder



Perl/Python

- Lab two has a tedious portion
- You'll need to lookup gates in a library
- I wrote a perl and python script for you to accelerate this process
  - . . . and serve as examples
- You'll still need to do this manually once
- Can use my scripts afterward
  - Read and understand it

Perl/Python

- A genius colleague is an ASIC (Application-Specific Integrated Circuit) design engineer at PMC-Sierra
- He glues together standard cells to make high-performance special-purpose circuits
- When he talks about perl, his eyes get all watery
- Why do digital design engineers get so excited about a system administrator's scripting language?

Commercial CAD tool flows are often a mess

- Different tools that don't quite work together
  - Translation
- Tools that don't quite finish the job
  - Pre-post processing
- Poor support for complex testing, e.g., comparing a high-level language model's behavior with circuit simulation
  - IO processing and command scripting
- Perl (python, etc.) allow quick (although sometimes inelegant) solutions to these problems

Perl code example motivation – Library

```
GATE    "1310:physical"        16      O=!1A;
PIN     * INV 1 999 1 .2 1 .2

GATE    "1120:physical"        24      O=!(1A+1B);
PIN     * INV 1 999 1 .2 1 .2

GATE    "1130:physical"        32      O=!(1A+1B+1C);
PIN     * INV 1 999 1 .2 1 .2

GATE    "1220:physical"        24      O=!(1A*1B);
PIN     * INV 1 999 1 .2 1 .2
```

Perl code example motivation – Gates

```
[348]      2310:physical   40.00
{cout}     1970:physical   56.00
[345]      1310:physical   16.00
[364]      1310:physical   16.00
{sum}      1860:physical   40.00
```

Perl code example

```
# Make sure number of args is correct
if (scalar @ARGV != 2) {
    die "Usage: lookup-gate.perl " .
        "[library] [file]\n";
}

my $lib = $ARGV[0];
my $file = $ARGV[1];

my %lab_op = ();
```

Perl code example

```
# Read in library
open LIB, "< $lib";
while (<LIB>) {
    if (m/^GATE\s+"([^"]+)"\s+\d+\s+(.+)$/) {
# Put the data into a hash map
        my ($label, $op) = ($1, $2);
        $lab_op{$label} = $op;
    }
}
close LIB;
```

Perl code example

```
# Loop on input file
open FILE, "< $file";
while (<FILE>) {
# Chop up the line on whitespace
    my @ln = split ’ ’, $_;
    if (scalar(@ln) == 3) {
# Grab the node and label
        my ($node, $lab) = @ln;
```

Perl code example

```
# Look it up in the hash map and
# display the results
        print "Node $node implemented with " .
            "gate $lab_op{$lab}\n";
    }
}
close FILE;
```

Perl code output

```
Node [348] implemented with gate O=!(1A*1B+!1A*!1B);
Node {cout} implemented with gate O=1A*1B+2C*2D;
Node [345] implemented with gate O=!1A;
Node [364] implemented with gate O=!1A;
Node {sum} implemented with gate O=!((1A+1B)*(2C+2D));
```

Python code example

```
# Make sure number of args is correct
if len(sys.argv) < 3:
  print ’Usage: lookup-gate.py [library] [file]’
  sys.exit(0)
lib, file = sys.argv[1:]
lab_op = dict();
```

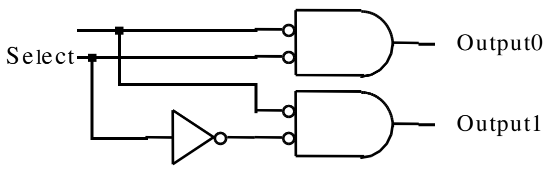
Python code example

```
# Read in library
fl = open(lib)
for ln in fl.readlines():
  m = re.match('^GATE\s+"([^"]+)"\s+\d+\s+(.+)$', ln)
  if m:
# Put the data into a hash map
    label, op = m.group(1, 2)
    lab_op[label] = op
fl.close()
```

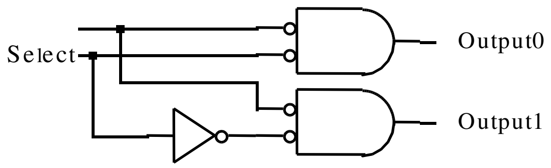
Python code example

```
# Loop on input file
fl = open(file)
for ln in fl.readlines():
# Chop up the line on whitespace
  ln_ar = ln.split()
  if len(ln_ar) == 3:
# Grab the node and label
    node, lab = ln_ar[:2]
# Look it up in the hash map and display the results
    print 'Node %s implemented with gate %s' % (node, lab_op
fl.close()
```

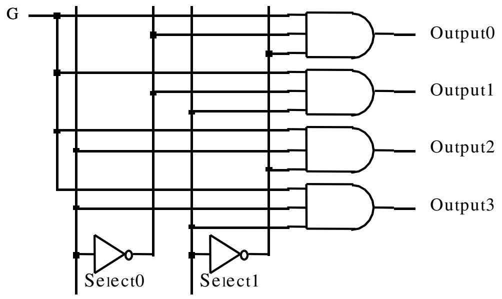
Back to implementation technologies
What is this?



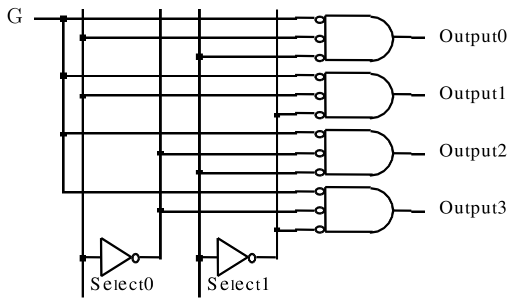
What is this?



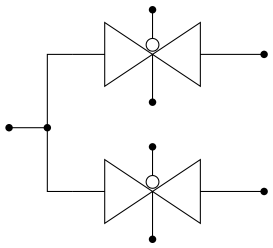
Active-high 2:4 decoder/demultiplexer



Active-low 2:4 decoder/demultiplexer

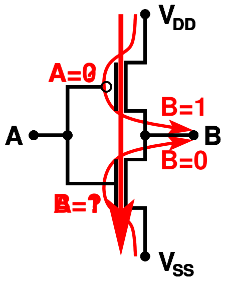


Dangers when implementing with TGs

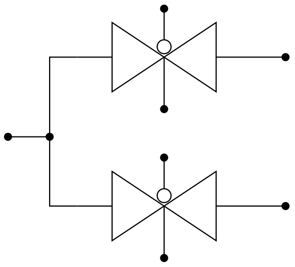


What if an output is not connected to any input?

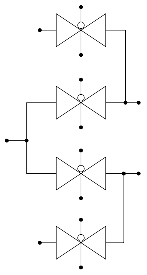
Consider undriven inverter inputs



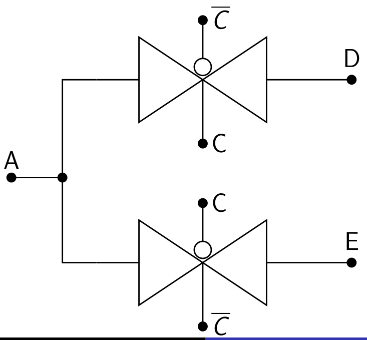
Dangers when implementing with TGs



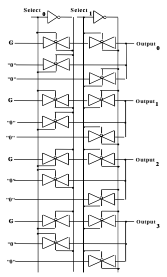
Set all outputs



Demultiplexer

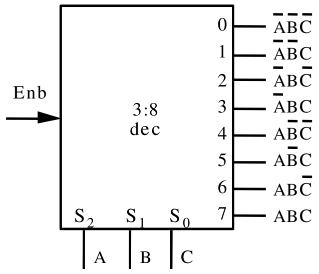


TG decoder/demultiplexer implementation



Consider alternative paths

Demultiplexers as building blocks

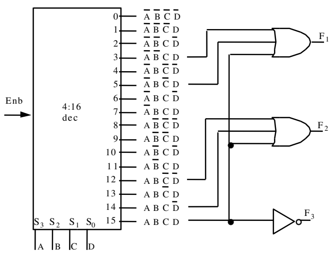

Enb
3:8 dec
0 $\overline{A}\overline{B}\overline{C}$
1 $\overline{A}\overline{B}C$
2 $\overline{A}B\overline{C}$
3 $\overline{A}BC$
4 $A\overline{B}\overline{C}$
5 $A\overline{B}C$
6 $AB\overline{C}$
7 ABC
S_2 S_1 S_0
A B C

Generate minterm based on control signals

Example function

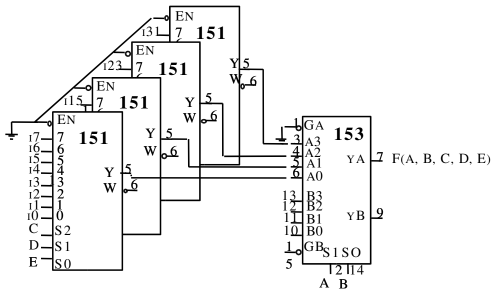
$$F_1 = \overline{A}\,\overline{B}\,CD + \overline{A}\,B\,\overline{C}\,D + ABCD$$
$$F_2 = ABC\,\overline{D} + ABC = ABC\,\overline{D} + ABC\overline{D} + ABCD$$
$$F_3 = \overline{A} + \overline{B} + \overline{C} + \overline{D} = \overline{ABCD}$$

Demultiplexers as building blocks

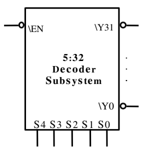


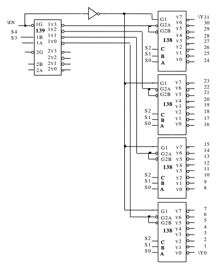
If active-low, use NAND gates

Implementation of 32:1 MUX

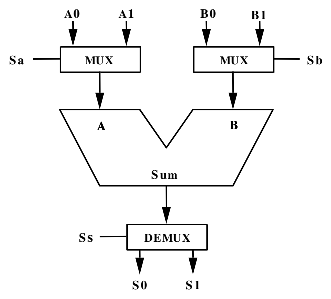


1:32 demultiplexer





Multiple I/O circuit



Summary

- Q&A
- PALs/PLAs
- Review, Q&A on MOS transistors
- Multiplexers, Demultiplexers
- Transmission gates
- Perl/Python

Homework

Recommended reading

- M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals*. Prentice-Hall, NJ, fourth edition, 2008
- Chapters 3 and 4

Lab two

Espresso and SIS logic minimization

Next lecture

- ROMs
- Multilevel logic minimization
- Review: NAND/NOR implementation