Introduction to Computer Engineering – EECS 203

http://ziyang.eecs.northwestern.edu/~dickrp/eecs203/

Instructor:	Robert Dick	TA:	Neal Oza
Office:	L477 Tech	Office:	Tech. Inst. L375
Email:	dickrp@northwestern.edu	Phone:	847-467-0033
Phone:	847–467–2298	Email:	nealoza@u.northwestern.edu
		TT:	David Bild
		Office:	Tech. Inst. L470
		Phone:	847-491-2083
		Email:	d-bild@northwestern.edu

NORTHWESTERN UNIVERSITY

Encoders

- Assume you have n one-bit signals
- Only one signal can be 1 at a time
- How many states can you be in?
- How many signals are required to encode all those states?

Encoder example

Pressed (i_2, i_1, i_0)	Code (o_1, o_0)
000	00
001	01
010	10
011	XX
100	11
101	XX
110	XX
111	XX

Implementation?

Priority encoder

What if we want the highest-order high signal to dominate?

Pressed (i_3, i_2, i_1)	Code (o_1, o_0)
000	00
001	01
010	10
011	10
100	11
101	11
110	11
111	11

What impact on implementation efficiency?

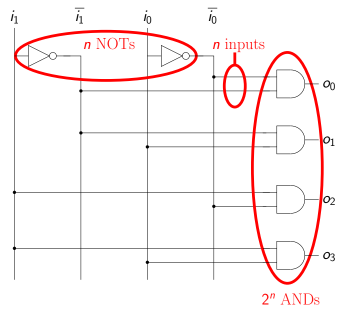
Decoders

Need to map back from encoded signal to state

Pressed (i_1, i_0)	Code (o_3, o_2, o_1, o_0)
00	0001
01	0010
10	0100
11	1000

o_0 isn't always used. Why?
Most straightforward implementation?

Straight-forward decoder implementation



Decoder implementation efficiency

- n NOTs
- n^2 n-input ANDS
- $\mathcal{O}\left(n^2\right)$
- Can't do this for large number of inputs!
- Instead, decompose into multi-level implementation

Multilevel decoder implementation

Starting point

$$o_0 = \overline{i_2}\,\overline{i_1}\,\overline{i_0}$$
$$o_1 = \overline{i_2}\,\overline{i_1}\,i_0$$
$$o_2 = \overline{i_2}\,i_1\overline{i_0}$$
$$o_3 = \overline{i_2}\,i_1 i_0$$
$$o_4 = i_2\overline{i_1}\,\overline{i_0}$$
$$o_5 = i_2\overline{i_1}\,i_0$$
$$o_6 = i_2 i_1\overline{i_0}$$
$$o_7 = i_2 i_1 i_0$$

Multilevel decoder implementation

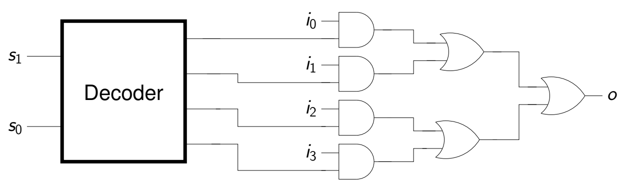
$$o_0 = \overline{i_2}\,(\overline{i_1}\,\overline{i_0})$$
$$o_1 = \overline{i_2}\,(\overline{i_1}\,i_0)$$
$$o_2 = \overline{i_2}\,(i_1\,\overline{i_0})$$
$$o_3 = \overline{i_2}\,(i_1\,i_0)$$
$$o_4 = i_2(\overline{i_1}\,\overline{i_0})$$
$$o_5 = i_2(\overline{i_1}\,i_0)$$
$$o_6 = i_2(i_1\,\overline{i_0})$$
$$o_7 = i_2(i_1\,i_0)$$

Reuse terms! Schematic?

Multiplexers or selectors

- Routes one of 2^n inputs to one output
- n control lines
- Can implement with logic gates

Logic gate MUX



However, there is another way. . .

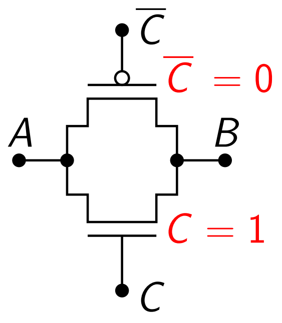
MUX functional table

C	Z
0	I_0
1	I_1

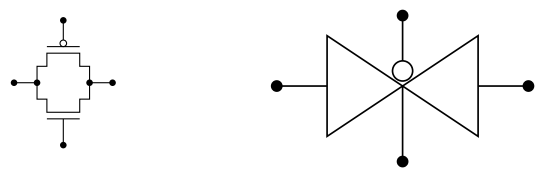
MUX truth table

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

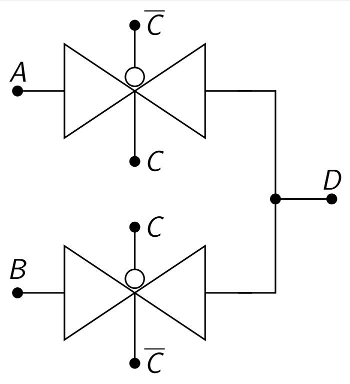
Review: CMOS transmission gate (TG)



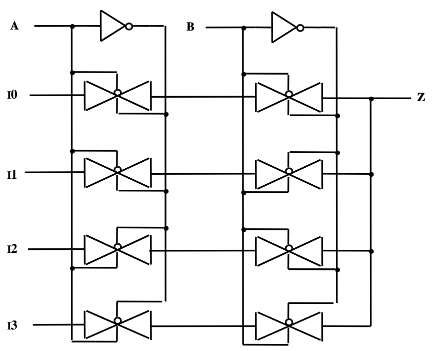
Review: Other TG diagram



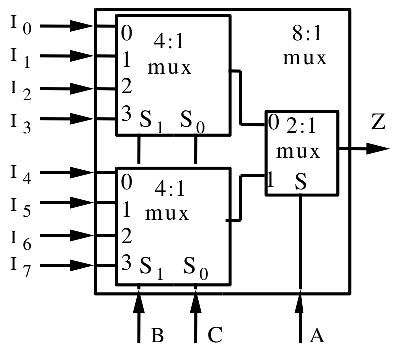
MUX



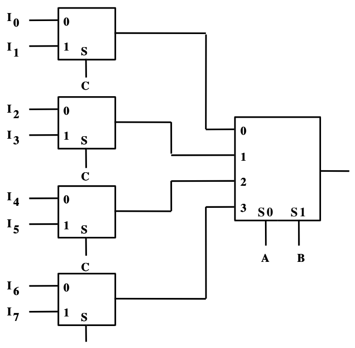
MUX using TGs



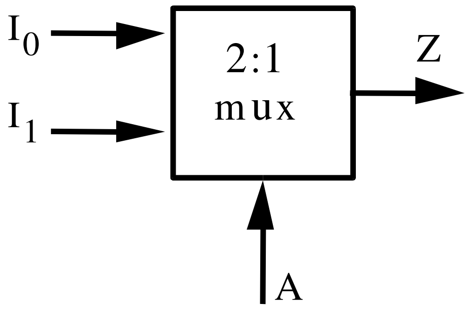
Hierarchical MUX implementation



Alternative hierarchical MUX implementation

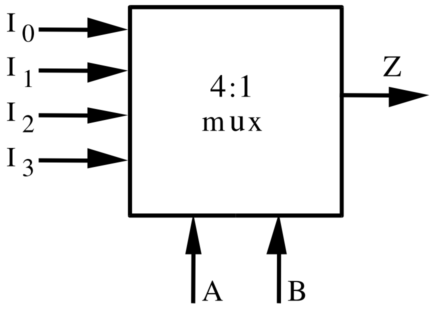


MUX examples



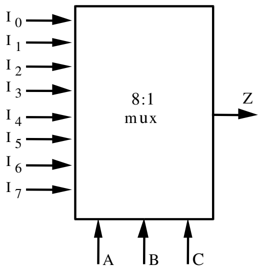
$$Z = \overline{A}\, I_0 + A I_1$$

MUX examples



$$Z = \overline{A}\,\overline{B}\, I_0 + \overline{A}\, B I_1 + A\overline{B}\, I_2 + A B I_3$$

MUX examples



$$Z = \overline{A}\,\overline{B}\,\overline{C}\, I_0 + \overline{A}\,\overline{B}\, C I_1 + \overline{A}\, B\overline{C}\, I_2 + \overline{A}\, B C I_3 +$$
$$A\overline{B}\,\overline{C}\, I_4 + A\overline{B}\, C I_5 + A B\overline{C}\, I_6 + A B C I_7$$

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

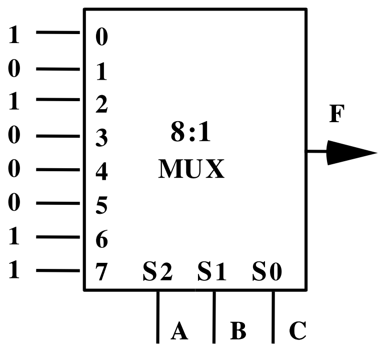
MUX example

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

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,

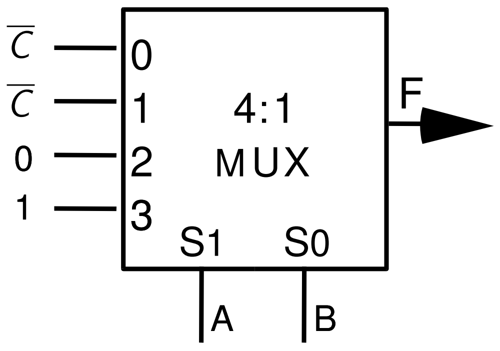
$$\overline{A}\,\overline{B} \rightarrow F = \overline{C}$$
$$\overline{A}\,B \rightarrow F = \overline{C}$$
$$A\overline{B} \rightarrow F = 0$$
$$AB \rightarrow F = 1$$

Truth table

A	B	C	F
0	0	0	1
0	0	1	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

F=$\overline{C}$

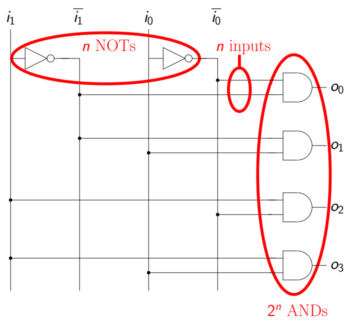
Lookup table implementation



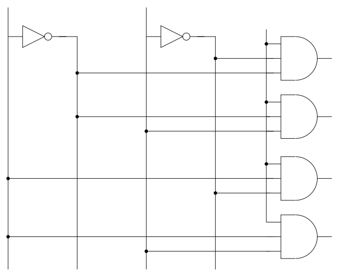
Demultiplexer (DMUX) definitions

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

Recall decoders

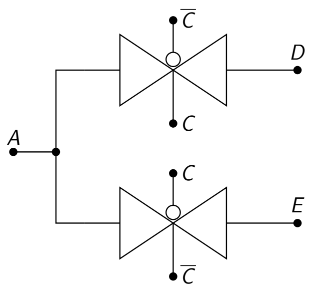


DMUXs similar to decoders

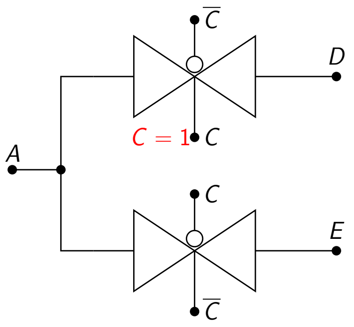


Use extra input to control output signal

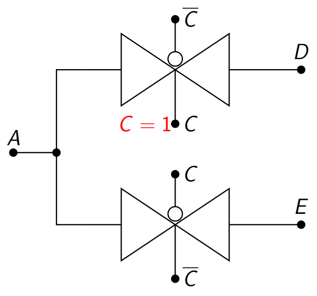
Demultiplexer



Dangers when implementing with TGs

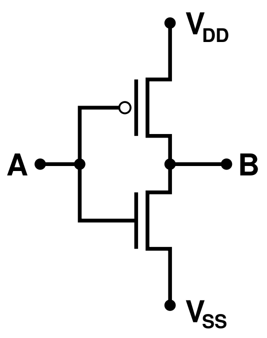


Dangers when implementing with TGs

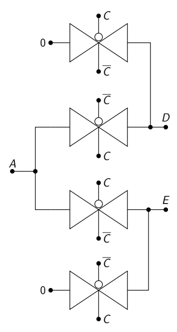


What if an output is not connected to any input?

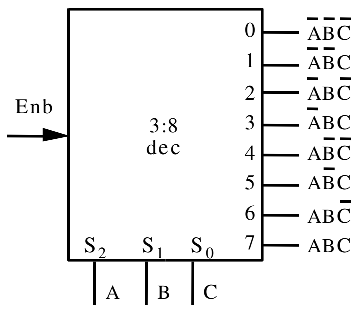
Review: Consider undriven inverter inputs



Set all outputs



Demultiplexers as building blocks

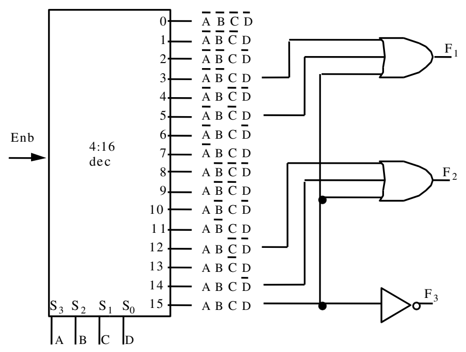

Enb
3:8
dec
0 $\overline{A}\,\overline{B}\,\overline{C}$
1 $\overline{A}\,\overline{B}C$
2 $\overline{A}B\overline{C}$
3 $\overline{A}BC$
4 $A\overline{B}\,\overline{C}$
5 $A\overline{B}C$
6 $AB\overline{C}$
7 ABC
S_2 S_1 S_0
A B C

Generate midterm based on control signals

Example function

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

Demultiplexers as building blocks


Enb
4:16
dec
0 $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$
1 $\overline{A}\,\overline{B}\,\overline{C}D$
2 $\overline{A}\,\overline{B}C\overline{D}$
3 $\overline{A}\,\overline{B}CD$
4 $\overline{A}B\overline{C}\,\overline{D}$
5 $\overline{A}B\overline{C}D$
6 $\overline{A}BC\overline{D}$
7 $\overline{A}BCD$
8 $A\overline{B}\,\overline{C}\,\overline{D}$
9 $A\overline{B}\,\overline{C}D$
10 $A\overline{B}C\overline{D}$
11 $A\overline{B}CD$
12 $AB\overline{C}\,\overline{D}$
13 $AB\overline{C}D$
14 $ABC\overline{D}$
15 $ABCD$
F_1
F_2
F_3
S_3 S_2 S_1 S_0
A B C D

Status

- CMOS
- Switch-based and gate-based design
- Two-level minimization
- Encoders
- Decoder
- Multiplexers
- Demultiplexers

Is anything still unclear? Then let's do some examples!

Lab three

- Requires error detection
- Read Section 1.4 in the book
- How to build an error injector, i.e., a conditional inverter?
- How to build a two-input parity gate?
- How to build a three-input parity gate from two-input parity gates?
- How to detect even number of ones?

Reading assignment

- M. Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals*. Prentice-Hall, NJ, fourth edition, 2008
- Sections 1.2–1.7

Computer geek culture reference

- Cliff Stoll. *The Cukoo's Egg*. Bantam Doubleday Dell Publishing Group, 1989