EECS 151/251A
Discussion 2

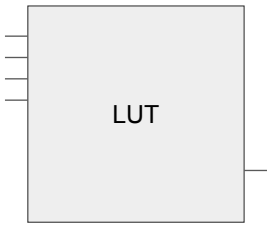
02/02/2018

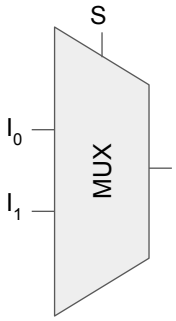
Agenda

- LUT as a MUX
- Boolean Algebra
- Timing
- FSM

LUT as a MUX

- What is a LUT?
- What is a MUX?
- How can you use a LUT as a MUX?
- Biggest MUX in a 5-LUT? 6-LUT?





Look-up table

				Output
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	

				Output
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

Outputs:

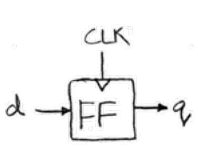
If lsb is sel; output =

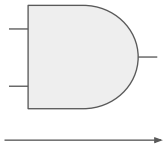
0
0
1
0
0
1
1
1

And repeated

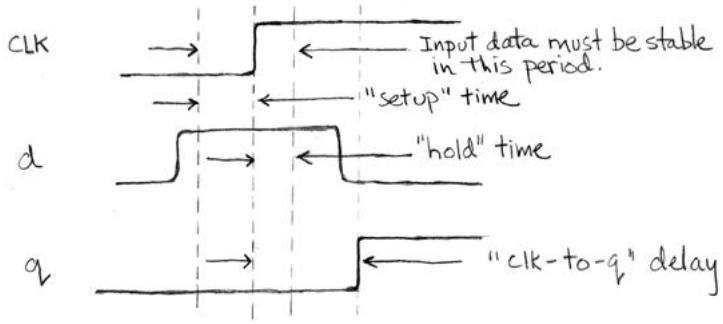
Timing

4 things we care about (for now)





Propagation delay



Combination logic propagation delay (max) - after this, the output stably represents the function applied to the input
Combination logic contamination delay (min prop delay) - after this, the previous output is no longer valid

Flip flops: set up time: time to keep input stable BEFORE clock edge
Hold time: time to keep input stable AFTER clock edge
Clock-to-Q: time until output of flip flop

Boolean Algebra

Basics

A.B	A AND B	1.1 = ? 1.0 = ?
A + B	A OR B	0 + 0 = ? 1 + 0 = ? 1 + 1 = ?
A + A = ? A + 0 = ?	A.A = ? A.0 = ?	

De Morgan's laws

~(AB) = ~A OR ~B
~A.~B = ?

Not a coincidence that the notation suggests behaviour in the manner we're used to with integers.

1.1 = 1
1 + 0 = 1
1 + 1 = 2 but in a

~(AB) = ~A OR ~B
~A.~B = ~~(~A.~B) = ~(A OR B)

Boolean Algebra

Basics

A.B	A AND B	1.1 = 1 1.0 = 0
A + B	A OR B	0 + 0 = 1 1 + 0 = 1 1 + 1 = 1 in $F_2/Z_2/GF(2)$
A + A = A A + 0 = A	A.A = A A.0 = 0	

De Morgan's laws

~(AB) = ~A OR ~B
~A.~B = ~(A OR B)

A NAND B = (NOT A) OR (NOT B))
(NOT A) AND (NOT B) = A NOR B

Not a coincidence that the notation suggests behaviour in the manner we're used to with integers.

1.1 = 1
1 + 0 = 1
1 + 1 = 2 but in a

~(AB) = ~A OR ~B
~A.~B = ~~(~A.~B) = ~(A OR B)

Truth Tables

- Quickly prove to you equivalence of expressions
- Canonically describe an expression (exhaustive proof)
- Require discipline and rigour
- Are your friend

If you ever get confused...

Simplification

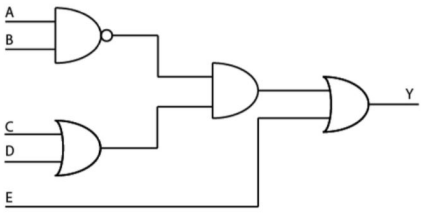
X = ~A.~B + ~A.B + A.B

~(A + ~B) = ~A.B

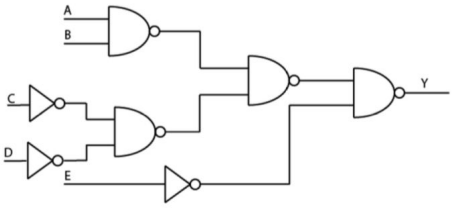
X = ~(~(~(A + ~B) + (AB))(A + B));

~A.~B + ~A.B + A.B
= ~A.~B + ~A.B + A.B + ~A.B
= ~A(~B + B) + B(A + ~A)
= ~A(1) + B(1)
= ~A + B

A) Convert the circuit below to use only 2-input NAND gates and inverters.



A) Using DeMorgan's Law, the NAND only circuit is:



De morgan's laws:
~(AB) = ~A OR ~B
~A.~B = ~~(~A.~B) = ~(A OR B)

In this circuit

C OR D = ~~(C OR D) = ~(~C.~D) <- nand with inverters

I (for intermediate) 0 (the top one) AND I1
I0.I1 = ~~(I0.I1)

I3 + E = ~(~(I3 + E)) = ~(~I3.~E) <- nand with inverters on inputs again
But if we do this step last, we see that the output just needs to be inverted anyway, so we can just turn the original and into a nand.

Think of the algebra you're used to. How do you simplify and solve, or expand, or factor, to get it to look like something you want?

Don't get hung up thinking through what it all means intuitively. It'll probably slow you down. Just do it a few times.

B) Configure a 2-to-1 multiplexer in order to implement the following functions:

(a) $F = XY$
(b) $F = X\bar{Y}$
(c) $F = X + Y$

You can only use one multiplexer per function. You are given the input signals X and Y (but NOT their complements) and you may use the supply and ground as inputs (i.e. 1 and 0).

B) Assume that the inputs of the multiplexer are In0, In1, the output is Out and the select signal is S, i.e. the multiplexer is implementing the function $Out = In0 \cdot \bar{S} + In1 \cdot S$. The following table shows how the multiplexer needs to be configured for each function:

Function	In0	In1	S	Out
(a)	0	X	Y	XY
(b)	X	0	Y	$X\bar{Y}$
(c)	Y	1	X	$X + Y$

Finite State Machines

Moore vs Mealy?

Examples?

Moore= outputs depend only on state
Mealy = outputs depend on state and other inputs

What are simple examples?

- Traffic lights

References

- Homework 2, EECS 151/251A Spring 2017
- Wikipedia