Lecture 5

Logical Effort
Using LE on a Decoder

Mark Horowitz
Computer Systems Laboratory
Stanford University
horowitz@stanford.edu

Overview

- **Reading**
  - Harris, Logical Effort talk slides
    - Read book if you want more information
  - For sizing, there is nothing that good in either book
    - Chandrakasan 14.3 describes some issues
  - Amrutur, Fast Low Power Decoders
    - Section I, II cover this lecture, Section III covers Lecture 6

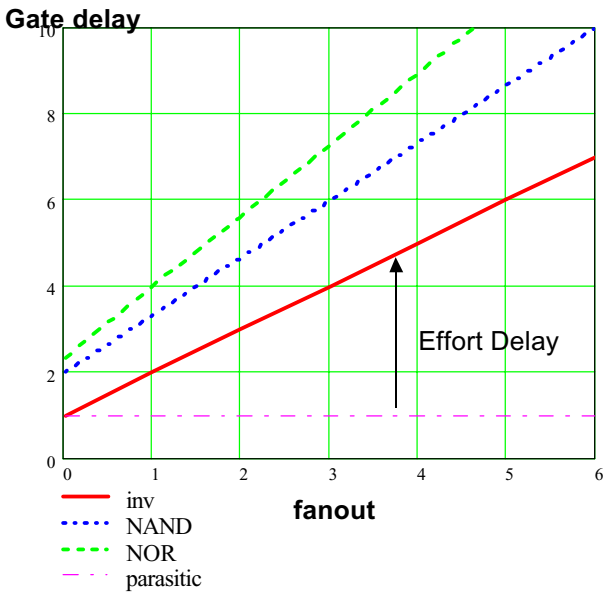
- **Overview**

  Having setup the basic optimization problems, we will next develop a formalism for doing sizing with real gates. This formalism is called logical effort. To get some practice using this method we will apply it to the memory decoder we talked about in Lecture 3. This lecture will use logical effort to optimize the performance of a decoder, and in the process motivate some ‘creative’ ways to build CMOS logic gates.

Logical Effort Formalism

- Let us express delays in terms of τ_{inv}
  - Delay* = Delay/ τ_{inv}
- Delay of logic gate has two components:
  - Delay* = EffortDelay + ParasiticDelay
- Effort delay again has two components:
  - EffortDelay = LogicalEffort * ElectricalEffort
- ElectricalEffort is just fanout
  - ElectricalEffort = Cload/Cin
- LogicalEffort describes the relative ability of gate topology to deliver current for a given input capacitance
  - LogicalEffort = τ_{gate} / τ_{inv}

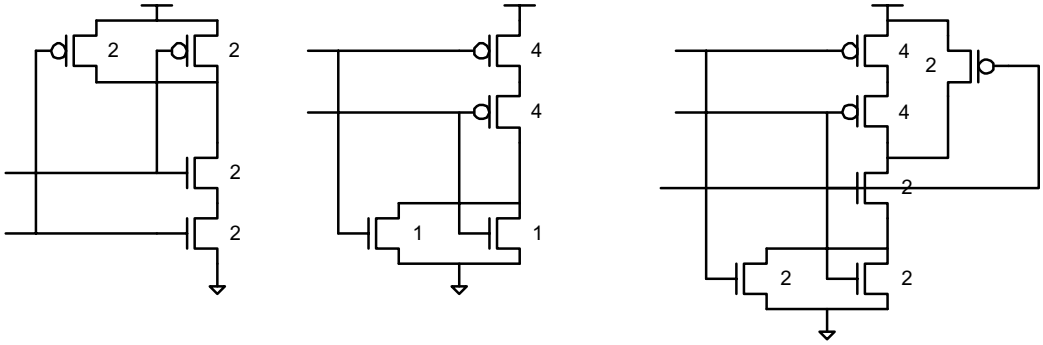
Logical Effort's View of Gate Delays



- Graphical model
  - Slope is logical effort
  - Y intercept is parasitic delay
- More complex gates
  - Have larger LE
  - Have larger parasitics
- Logical Effort
  - $\frac{Cin_{gate} * Rdrive_{gate}}{Cin_{inv} * Rdrive_{inv}}$

Calculating Logical Effort for a Gate

- Build the gates to have the same drive strength as a 2x pMOS, 1x nMOS inverter. The numbers on each transistor is relative to the 1x nMOS transistor in the inverter. The Cin of inverter is 3x.



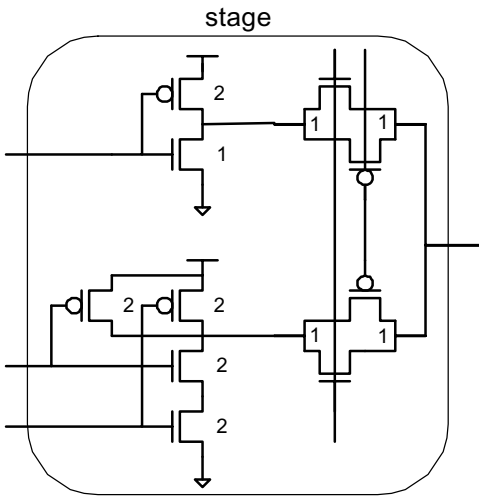
- LE = 4/3 LE=5/3 LE=2; 4/3
- Note that the logical effort of all inputs does not always match

Warm-ups with Logical Effort

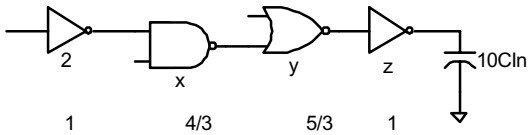
- Frequency of an N stage ring oscillator
  - LE=1, FO=1, γ=1; delay = 2 per inverter
  - Delay through N inverters is 2*N,
  - Frequency is 1/(4N) since it takes 2N time to change from a high to a low, and another 2N to change from a low to a high
- Delay of a inverter with a fanout of 4
  - LE=1, FO=4, γ=1; delay = 5 per inverter
- Normally we will report delays in terms of FO4 inverter delays
  - Roughly equal to normalized delay divided by 5

Logical Effort for Transmission Gates

- Need to consider the stage
  - TG and drive gates
- Rdrive
  - Set by path to Vdd/Gnd
  - Resistance of logic gate + TG
  - In this example it is 2x a 2/1 inverter from all paths
- Cin
  - Depends on input
  - 3 for inv, 4 for NAND, 2 TG
- LE
  - 2 for inv, 8/3 for NAND, 4/3 TG



Using Logical Effort to Size Gates

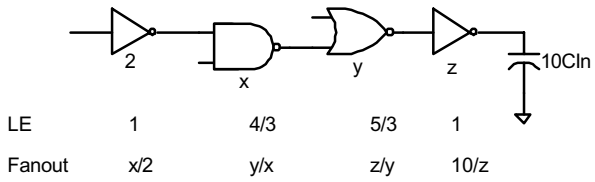


LE	1	4/3	5/3	1
Fanout	x/2	y/x	z/y	10/z

- Know the effort delay is the same per stage
  - Call this the effective fanout, (EF) since it is the delay for an inverter with this fanout
- Need to find the total effective fanout for the chain
  - This is the product of LE * Fanout of every stage
  - =Product of LEs *fanout of chain
- In example:
  - Total LE = 2.2, Fanout = 5
  - Effective Fanout = 11

$$\prod_j LE_j \cdot \frac{C_{in_{j+1}}}{C_{in_j}} = \frac{C_{load}}{C_{in_1}} \cdot \left(\prod_j LE_j \right)$$

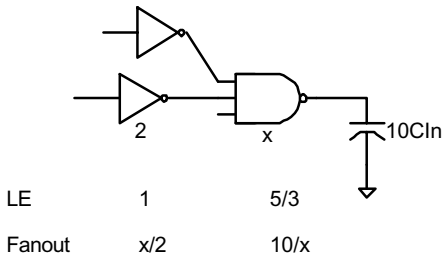
Sizing Gates



- Given we know the total EF
  - EF for each stage must be $EF^{1/N}$
  - Makes the EF of each stage the same
  - Matches the required total
- Given the EF of each stage
  - Fanout is EF/LE of that stage
- In the example the EF of each stage is $1.8 = 11^{1/4}$
  - This is too small. We have too many stages

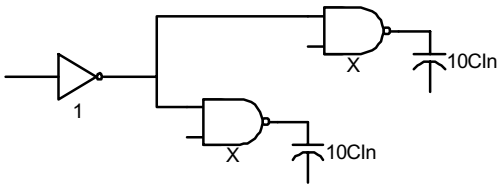
Reducing Number of Logic Stages

- Reformulate logic
  - If you have too low EF, use more complex gates, with fewer stages
  - Often need to use AND-OR-Invert gates or OR-AND-Invert gates
  - EF = 8.33, with two stages
  - EF per stage is 2.9 which is quite reasonable
    - X = 2*2.9 = 5.8



Note: this was the wrong optimization if the top input was critical. Since we have slowed down this path. Make sure you are sizing the critical path.

Branching Factors



- In some circuits there is fan-out in the conventional sense
  - One gate drives a number of gates
- If all gates are on the critical path
  - $EF_{stage} = LE * (C_{j+1}/C_j) * B$ (Branching factor)
- Total EF for a chain is then the product of LE and B for all the gates on the chain. In the example = 4/3*2*10
  - Electrical fanout per stage = $EF_{stage}/(LE*B)$
  - So X in the example would be around 2.6 = ½ * √27

Logical Effort Summary

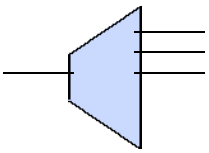
- Estimate the path effort
  - EF = Π LE * Π B *Cload/Cin
- Estimate the optimal number of stages
  - $N^* = \log_4(EF)$
- Estimate the minimum delay
  - 4N* + Parasitic delay
- Determine the actual number and type of gates
  - Fit the required logic in N stages, where N is close to N*
  - This may change slightly the path EF
- Determine the stage effort
  - New $EF^{1/N} = f$
- Working from either end, determine gate sizes
  - Cin = Cout*LE*B/f

Some Definitions

Term	Stage expression	Path expression
Logical effort	g (seeTable 1)	$G = \prod g_i$
Electrical effort	$h = \frac{C_{out}}{C_{in}}$	$H = \frac{C_{out\ (path)}}{C_{in\ (path)}}$
Branching effort	n/a	$B = \prod b_i$
Effort	$f = gh$	$F = GBH$
Effort delay	f	$D_F = \sum f_i$
Number of stages	1	N
Parasitic delay	p (seeTable 2)	$P = \sum p_i$
Delay	$d = f + p$	$D = D_F + P$

Decoder Review

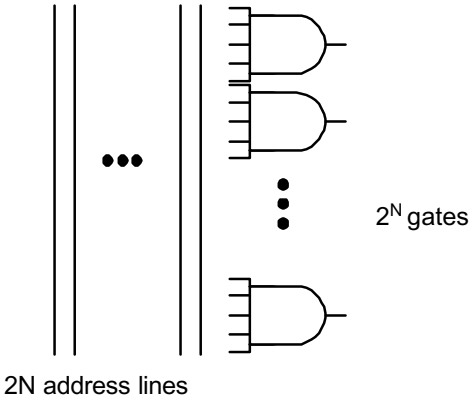
Decoder has two main jobs:



- Logic function
  - Using N address bits
  - Needs to select 1 of 2^N wordlines
  - This means the logical effort of the chain will be larger than 1
    - Equal to LE of an N input AND gate
- Act as a buffer chain
  - The address line has a large fanout
    - Each address line ultimately needs to drive every AND gate
    - A0 drives ½ of the decoders and A0_b drives the other ½
  - The wordline capacitance can be large
    - It has 2^M cells on it, and a large wire capacitance
  - Total fanout is proportional to the size of the memory

Decoder Logic

- N address
  - 2N lines
  - A0, A0_b both needed
- Each decoder an AND gate
  - Large fanin
  - Must drive wordline

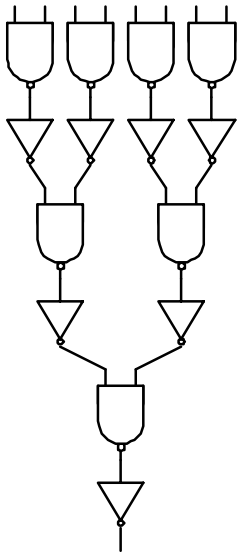


Example for Class

- Assume we are building a 256x256 memory (8KB memory)
- Assume that $C_{add} = 4^* C_{cell}$
- Total fanout on each address input is then 2^{14}
- We still need to build the decoder too!
- What is the minimum logical effort for 8 input AND gate?
  - Since fanout is large don't need to worry about # of stages
  - We will need lots of inverters anyhow
- Many possibilities
  - 2 NAND -Inv trees
  - 2 NAND - 2 NOR trees
  - 4 NAND - Inv tree
  - Etc

Optimal Static Decoder

- Logical Effort of building 4 input AND
  - Two 2 NAND gates is 4/3*4/3 = 1.8
  - One 4 NAND is 2
  - 2-input NAND/2-input NOR is 4/3*5/3 = 2.2
- Which is best?
  - Depends on the number of levels of logic you need
  - If you need lots of gates, 2-input gates are often the best
- Using 2-input NAND gates
  - An 8-input gate will take 6 levels of gates
    - 8 to 4 outputs, 4 to 2 outputs, 2 to 1 output



Number of Stages of Logic

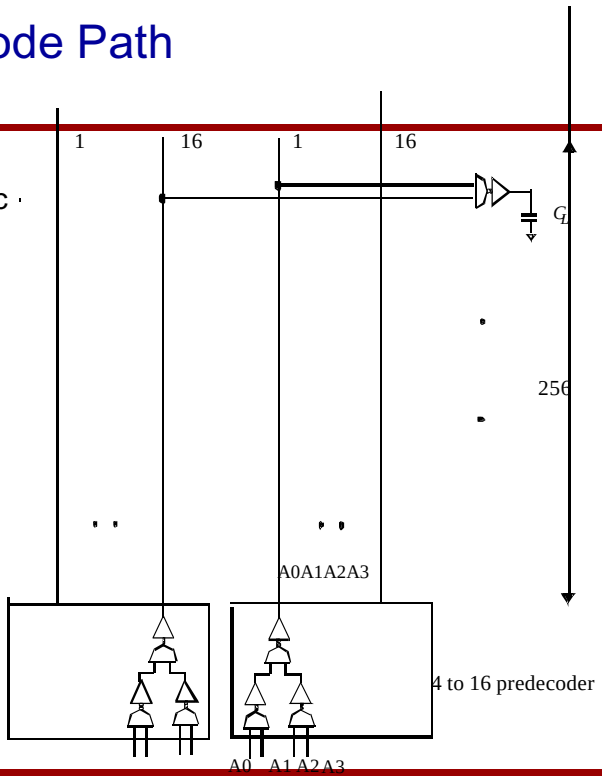
- For our decoder (assuming 2 input NAND gates)
  - The decoder has a total effort of
    - 2.4 (which is 4/3 cubed) * FO (which is 2^{14})
    - Note that using other gates would not change this result very much, so don't sweat which gate you are going to use when you figure out the total effective fanout.
  - For a effective fanout of around 4 per stage
    - This design would need around 7.5 stages which is $\text{Log}_4(2^{15})$
- But how is it going to be put together?
  - How do we size the individual gates?
  - Which wires do we run up the decoder?

Predecode Options

- Two basic choices
  - Can do a 2-4 predecode in 4 groups, with a 4 to 1 final gate
    - Final gate has two level of and gates
    - Uses only 16 address wires running across the decoder
    - Final gates are larger
  - Can do a 4-16 predecode in 2 groups, with a 2-1 final gate
    - Uses 32 address lines running across the decoder
    - Final gates are smaller
- Generally doing a larger predecode is better for two reasons
  - More levels of logic before the wire capacitance
  - Less capacitance switches each cycle (lower power)

Decode Path

- By using a 4-16 predecode we have move more stages of logic before the long wire
  - This decreases its effect on the circuit, since it naturally give us more stages of buffers before driving the wire
  - Otherwise we would need more stages in the predecode, just to drive the wire



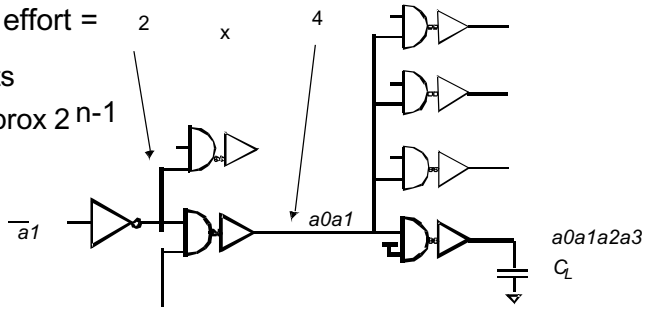
General Predecode

Consider the 4 to 16 decoder:

branching effort = 2 x 4

for n inputs

BE_p is approx 2^{n-1}



logical effort = 1 x $\frac{4}{3}$ x 1 x $\frac{4}{3}$ x 1

total path effort PE= $(LE_p)(BE_p)(FO_p)$

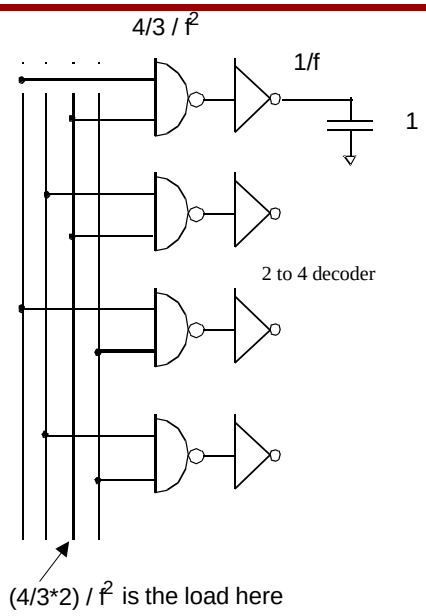
$=(4/3)^2(2)(4)C_L/C_{in}$

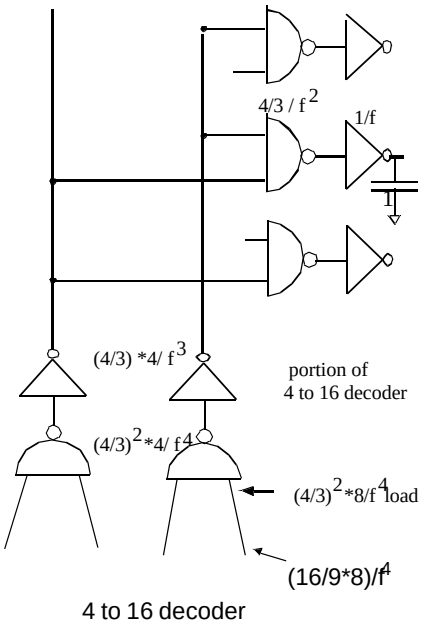
- This formula has A0 and A0_b as separate inputs
  - Otherwise branching effort would be 2^n

Logical Effort of Predecoder

This works back from the load on the predecoder for both a 2-4 predecoder, and a 4-16 predecode

If this gives you a input capacitance that is too high, you need to add extra buffer stages

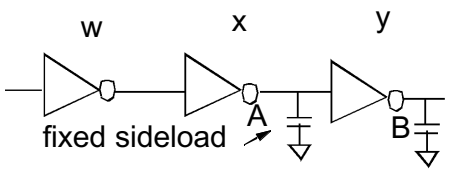




Logical Effort in Real Life

- There are fixed side loads you need to deal with
  - This is the wire capacitance of the predecoder outputs
  - Since these capacitances don't scale with sizing, they don't fit in nicely to the logical effort frame work
    - They also are caused by loading of non-critical gates
    - But they are not a huge issue either
- The optimal sizes are often pretty large
  - This will cause power and area issues
  - Often you need to back off a little to make things fit better

Handling Fixed Side Loads



If B=64 and A=0, then w=1, x=4, y=16 (easy)

If B=64 and A=8, things get more difficult.

- First solve the problem without the sideload: gives w=1, x=4, y=16
- Formally we can add the sideload and solve the problem again using the fact that x/w = (A+y)/x and y/x=B/y. These equations can be rewritten in an iterative form and then solved. But after iterating, this produces x=5 y=18. But increasing the size of 'y' makes a very small difference and adds power and area. It is probably not a good idea.
- So, if the side load is smaller than the gate load, we can size gates beyond the sideload using the standard approach, and size the gates before the sideload using the the sum of the sideload and the previous result.