BROWN
School of Engineering

DIGITAL ELECTRONICS SYSTEM DESIGN

FALL 2019

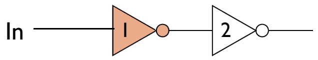
PROF. IRIS BAHAR

NOVEMBER 13, 2019

LECTURE 19: FANOUT SIZING

NMOS/PMOS RATIO

- So far we have sized the PMOS and NMOS so that the R_{eq} values match (i.e., β = $(W/L_p)/(W/L_n)$ = W_p/W_n = 2 to 2.8)
  - Symmetric VTC
  - Equal high-to-low and low-to-high propagation delays
- If speed is the only concern, reduce the width of the PMOS device!
  - Widening the PMOS degrades the t_{pHL} due to larger intrinsic capacitance
- What does this imply if we want to minimize delay for an inverter?



NMOS/PMOS RATIO

- We define propagation delay as:

$$t_p = \frac{t_{pHL}+t_{pLH}}{2} = 0.69C_L\left(\frac{R_{eqn}+R_{eqp}/\beta}{2}\right)$$

- And define

$$C_L = (C_{dp1}+C_{dn1})+(C_{gp2}+C_{gn2})+C_W$$

$$C_L \approx (1+\beta)(C_{dn1}+C_{gn2})+C_W$$

- So we have

$$t_p = 0.345((1+\beta)(C_{dn1}+C_{gn2})+C_w)(R_{eqn}+R_{eqp}/\beta)$$

$$t_p = 0.345\big((1+\beta)(C_{dn1}+C_{gn2})+C_w\big)R_{eqn}\left(1+\frac{r}{\beta}\right), \quad \text{where } r = \frac{R_{eqp}}{R_{eqn}}$$

- Now, optimize t_p with respect to β…

NMOS/PMOS RATIO

- Given the equation for t_p:

$$t_p = 0.345\big((1+\beta)(C_{dn1}+C_{gn2})+C_w\big)R_{eqn}\left(1+\frac{r}{\beta}\right)$$

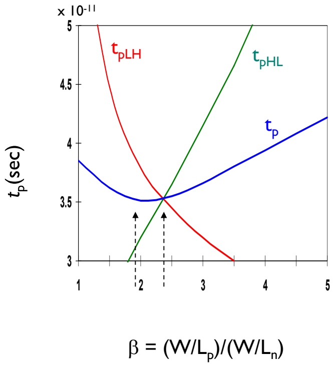
- Minimize t_p as a function of β…
- Compute the optimal value of β by setting $\partial t_p/\partial\beta = 0$

$$\beta_{opt} = \sqrt{r\left(1+\frac{C_W}{C_{dn1}+C_{gn2}}\right)}$$

Where r=R_{eqp}/R_{eqn}=resistance ratio for identically sized PMOS, NMOS

$$\beta_{opt} = \sqrt{r} \quad \text{when } C_W \text{ is negligible}$$

PMOS/NMOS RATIO EFFECTS



$\beta = (W/L_p)/(W/L_n)$

β of 2.4 gives symmetrical response

β of 1.6 to 1.9 gives optimal performance

DEVICE SIZING FOR PERFORMANCE

- Divide capacitive load, C_L, into
  - C_{int} : intrinsic → diffusion
  - C_{ext} : extrinsic → fanout (gate-channel cap and wiring)

$$t_p = 0.69\, R_{eq}\, C_{int}\, (1 + C_{ext}/C_{int}) = t_{p0}\, (1 + C_{ext}/C_{int})$$

  - $t_{p0} = 0.69\, R_{eq}\, C_{int}$ is the intrinsic (unloaded) delay of the gate
- Widening both PMOS and NMOS by a factor S reduces R_{eq} by an identical factor ($R_{eq} = R_{ref}/S$), but raises the intrinsic capacitance by the same factor ($C_{int} = SC_{iref}$)

$$t_p = 0.69\, R_{ref}\, C_{iref}\, (1 + C_{ext}/(SC_{iref})) = t_{p0}(1 + C_{ext}/(SC_{iref}))$$

DEVICE SIZING FOR PERFORMANCE

$$t_p = 0.69\, R_{ref}\, C_{iref}\, (1 + C_{ext}/(SC_{iref})) = t_{p0}(1 + C_{ext}/(SC_{iref}))$$

What can we conclude from this?

- t_{p0} is independent of the sizing of the gate; *with no load the drive of the gate is totally offset by the increased capacitance*
- Any S sufficiently larger than (C_{ext}/C_{int}) yields the best performance gains with least area impact

IMPACT OF FANOUT ON DELAY

$$t_p = t_{p0}\, (1 + C_{ext}/C_{int})$$

- Extrinsic capacitance, C_{ext} , is a function of the gates being driven by the gate under question (i.e. the fanout)

  larger fanout ⇨ larger external load.
- Re-express the intrinsic capacitance (C_{int}) in terms of input gate capacitance:

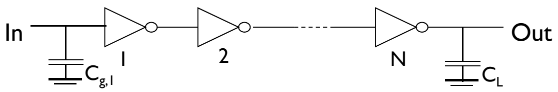
$$C_{int} = \gamma C_g\,, \quad \text{where } \gamma \approx 1$$

$$t_p = t_{p0}\, (1 + C_{ext}/\, \gamma C_g) = t_{p0}\, (1 + f/\gamma)$$

$f = C_{ext}/C_g$ is the effective fanout

INVERTER CHAIN

- Goal is to minimize the delay through an inverter chain



- The delay of the j^{th} inverter stage is

  $t_{p,j} = t_{p0}\,(1 + C_{g,j+1}/(\gamma C_{g,j})) = t_{p0}(1 + f_j/\,\gamma)$

  and $t_p = t_{p,1} + t_{p,2} + \ldots + t_{p,N}$

  so $t_p = \sum t_{p,j} = t_{p0} \sum\,(1 + C_{g,j+1}/(\gamma C_{g,j}))$

- If C_L and $C_{g,1}$ are given, we have 2 different optimizations
  - How should the inverters be sized to minimize delay?
  - What is the optimal number of inverters to minimize the delay?

SIZING THE INVERTERS IN THE CHAIN

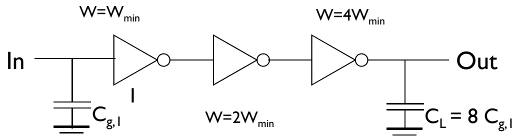
- After a bit of calculus, we find that for minimum delay:

  $C_{g,j+1}/C_{g,j} = C_{g,j}/C_{g,j-1}$ for j=2…N
- What does this imply?
  - All gates have the same effective fanout, f
  - Each gate should be scaled up by the same factor w.r.t. its preceding gate
- What is the effective fanout for a gate given C_L and $C_{g,1}$?
  - With a bit of algebra and inductive reasoning we find that:

  $$f = \sqrt[N]{C_L / C_{g,1}} = \sqrt[N]{F}$$
  - $F = C_L/C_{g,0}$ is the overall effective fanout
- What is the minimum delay through the chain?

  $$t_p = N t_{p0}(1 + \sqrt[N]{F} / \gamma)$$

EXAMPLE: INVERTER CHAIN SIZING



- $C_L/C_{g,1}$ has to be evenly distributed over N = 3 inverters

  $F = C_L/C_{g,1} = 8/1$

  $$f = \sqrt[3]{8} = 2$$
- Assuming L remains unchanged for all inverters, f is obtained by adjusting W relative to the previous stage (i.e., scale up by factor of 2 relative to the previous gate).

OPTIMAL NUMBER OF INVERTERS

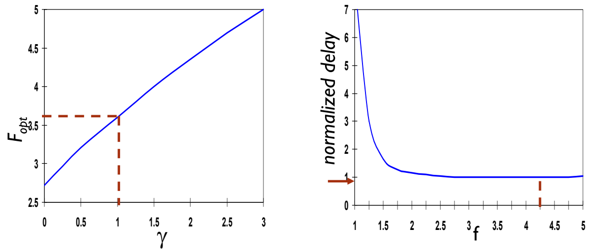
- What is the optimal value for *N* given F? (where F = f^N)
  - if the number of stages is too large, the intrinsic delay of the stages dominates
  - if the number of stages is too small, the effective fan-out of each stage dominates

  $$t_p = N t_{p0}(1 + \sqrt[N]{F} / \gamma)$$

  $$\partial t_p / \partial N = \gamma + \sqrt[N]{F} - \frac{\sqrt[N]{F} \ln F}{N} = 0$$

  $$\rightarrow f = e^{(1+\gamma/f)}$$
- For γ = 0 (ignoring self-loading) *N = ln (F)* and the effective-fan out (tapering factor) becomes f = e = 2.718
- For γ = 1 (the typical case) the optimum effective fan-out can be solved numerically and turns out to be close to 3.6

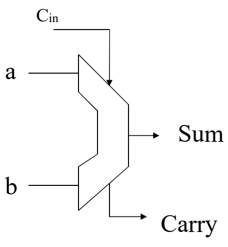
OPTIMUM EFFECTIVE FANOUT



- Too many stages has a substantial negative impact on delay
- Choosing f slightly larger than optimum has little effect on delay and reduces the number of stages (and area).
  - Common practice to use f = 4 (for γ = 1)

Fanout of 4 (FO4) rule of thumb delay metric is based on this result

FULL ADDER

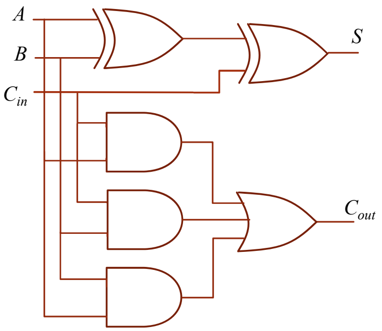


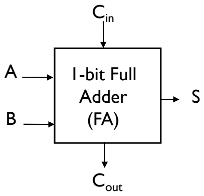
Truth Table

Id	a	b	c_{in}	carry	sum
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

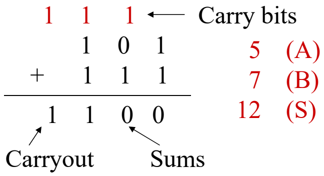
- How do you express sum and carry as Boolean functions?

THE FULL ADDER





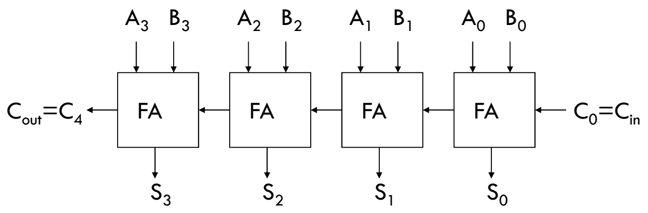
BUILDING A BINARY ADDER



```
  1  1  1  ← Carry bits
     1  0  1      5  (A)
 +   1  1  1      7  (B)
 ___________
   1  1  0  0    12  (S)
 Carryout    Sums
```

- Inputs & outputs for the i^{th} bit position
  - Inputs: A_i, B_i, and C_i (carry-in)
  - Outputs: S_i (sum) and C_{i+1} (carry out)
- Carry out of a bit position is the carry in for the next bit position

THE RIPPLE CARRY ADDER



- The carry out of one stage ripples to the carry in of the next

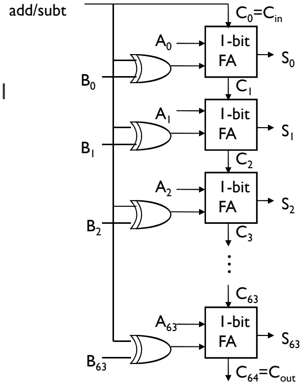
2'S COMPLEMENT SUBTRACTION

- Negate second operand and add:

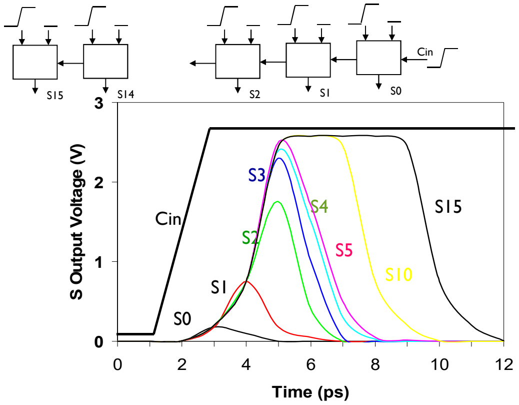
```
  01101000     (104)
– 00010001     (17)

  01101000     (104)
– 11101111     (-17)
  01010111     (87)
```

A 64-BIT ADDER/SUBTRACTOR

- Ripple Carry Adder (RCA) built out of 64 Full Adders
- Subtraction – complement all subtrahend bits (xor gates) and set low order carry-in
- RCA
  - Simple logic, so low (area) cost
  - Slow: (*O*(*N*) for N bits) and lots of glitching



GLITCHING IN A RIPPLE CARRY ADDER



FAST CARRY CHAIN DESIGN

- The key to fast addition is a low latency carry network
- What matters is whether in a given position a carry is
  - generated $G_i = A_i \,\&\, B_i \;= A_iB_i$
  - propagated $P_i = A_i \oplus B_i$
  - annihilated (killed) $K_i = !A_i \,\&\, !B_i$
- Giving a carry recurrence of

$$\boxed{C_{i+1} = G_i + P_iC_i}$$

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1\,P_0C_0$

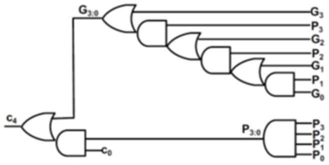
$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0\,C_0$

$C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0\,C_0$

CARRY OUT LOGIC FOR 4-BIT ADDER

- Ps and Gs are computed at time 0
- $C_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0\,C_0$

$$= \underbrace{G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))}_{G_{3:0}\text{: carry generation from bits 3-0}} + \underbrace{(P_3P_2P_1P_0)}_{P_{3:0}\text{: carry propagation from bits 3-0}} C_0$$



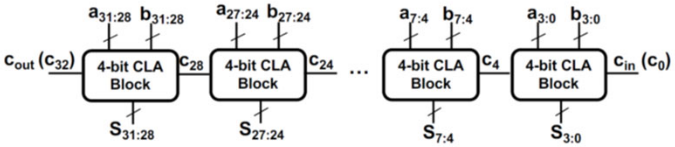
$G_i = A_i \,\&\, B_i$

$P_i = A_i \oplus B_i$

P_i can also be expressed as

$P_i = A_i + B_i$

32-BIT CARRY LOOKAHEAD ADDER WITH 4-BIT RIPPLE CARRY ADDERS



- Carry out logic gets more complicated beyond 4 bits
- CLAs are often implemented as 4-bit modules and instantiated in a hierarchical way to realize wider adders