BROWN
School of Engineering

# DIGITAL ELECTRONICS SYSTEM DESIGN

**FALL 2019**

**PROF. IRIS BAHAR**

NOVEMBER 11, 2019

LECTURE 18: ADDING SIGNED NUMBERS & MINIMIZING DELAY

## SEMINAR SPEAKER THIS TUESDAY

**Hannah Chung**
Co-founder & Head of Design of Sproutel
Tuesday, November 12
B&H 190, 12-1pm

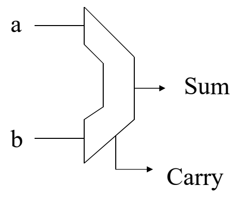
**From Mechanical Engineering to Designer --- The Sproutel Story**

Hannah will share her story of finding her voice as a designer and engineer as she pursues her passion for designing for good in healthcare

## UPDATES TO LAB HOURS THIS WEEK

- Monica has to cancel her lab hours Tuesday 1-4pm. She will schedule makeup hours later this week. Stay tuned.

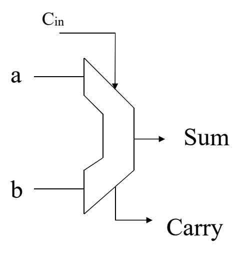
## HALF ADDER



Truth Table

| a | b | carry | sum |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

## FULL ADDER

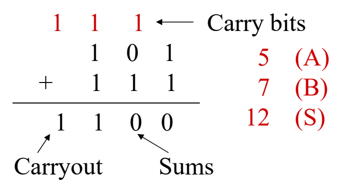


Truth Table

| 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 |

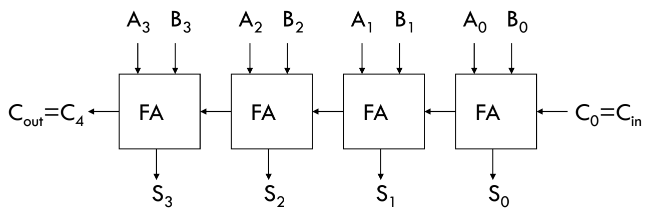
- How do you express sum and carry as Boolean functions?

## BUILDING A BINARY ADDER



- Inputs & outputs for the ith 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

## WHAT ABOUT NEGATIVE NUMBERS?

- So far we have just considered unsigned numbers when converting from base 10 to binary.
- What about negative numbers and how do we add two signed numbers in binary?
- 3 ways of representing signed numbers:
  - Signed magnitude
  - 1's complement
  - 2's complement

## SIGNED MAGNITUDE

- The Most Significant Bit (MSB) is the sign bit: 0 → positive, 1 → negative
- The rest of the bits define the magnitude
- Need to *know how many bits are available* to represent a number!
- Example: $(2)_{10} = (0010)_2 = (0\ 010)_{S\&M}$
  $(-2)_{10} = (1\ 010)_{S\&M}$
- Makes adding and subtracting a pain
  - Can't just add them regularly
- Also, two representations for zero (+0 and -0)

## SIGNED MAGNITUDE ADDITION

$(1)_{10}$ → 0001
+ $(5)_{10}$ → + 0101
---------- ---------
$(6)_{10}$ 0110 (both positive, so a positive result)

$(-2)_{10}$ → 1010
+ $(-4)_{10}$ → + 1100
---------- ---------
$(-6)_{10}$ 1110 (both negative, so keep the negative sign)

$(4)_{10}$ → 0100 (larger number – smaller number)
+ $(-3)_{10}$ → + 1011 (keep the sign of the larger number)
---------- ---------
$(1)_{10}$ 0001 (signs are different → subtract smaller from larger number, keep sign of larger number)

- Need a comparator to supplement adder/subtractor

## 1'S COMPLEMENT

- To negate a number, complement (invert, flip) each bit
- Example: $(4)_{10} = (0100)_2 = (0100)_{1\text{'s comp}}$
  $(-4)_{10} = (1011)_{1\text{'s comp}}$
- Like sign and magnitude, the high bit indicates the sign of the number
- What about adding and subtracting?

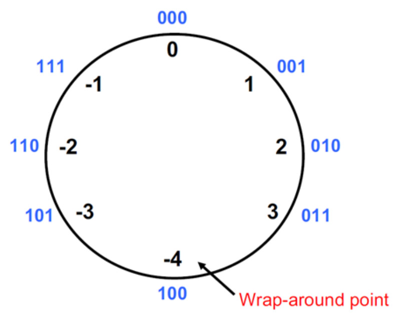
## 1'S COMPLEMENT ADD/SUBTRACT

$(-2)_{10}$ → 1101
+ $(-4)_{10}$ → + 1011
---------- ---------
$(-6)_{10}$ 11000 -- not right, $(-6)_{10} = (1001)_{1\text{'s comp}}$
+ 1 → add $C_{out}$ back to LSB
---------
1001 -- now it works

$(4)_{10}$ → 0100
+ $(-3)_{10}$ → + 1100
---------- ---------
$(1)_{10}$ 10000 -- not right, add $C_{out}$ back to LSB
+ 1
---------
0001 -- now it works

- Better than sign and magnitude (can subtract by adding the negative)
- But requires 2 addition operations (need to conditionally add $C_{out}$)

## ANOTHER ENCODING FOR BINARY



## 2'S COMPLEMENT REPRESENTATION

- MSB has weight $-2^{n-1}$
- Range of an n-bit number is $-2^{n-1}$ through $2^{n-1}-1$
  - Smallest negative number ($-2^{n-1}$) has no positive counterpart

| $-2^2$ | $-2^1$ | $-2^0$ | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | -4 |
| 1 | 0 | 1 | -3 |
| 1 | 1 | 0 | -2 |
| 1 | 1 | 1 | -1 |

## 2'S COMPLEMENT

- To negate in 2's complement, complement (flip) each bit and then add 1
- Example: Represent $(-5)_{10}$ in 2's complement using 4 bits

  $(5)_{10} = (0101)_{2\text{'s comp}}$
  ```
     1010
  +     1
  --------
     1011
  ```
  $1011 \rightarrow (-5)_{10} = (1011)_{2\text{'s comp}}$
- Like sign and magnitude, the MSB indicates the sign of the number
- Sign extension: Pad the high bits with the value of the MSB

  Example: $(-6)_{10} = (1010)_2 = (111010)_2$
- Range: for *n* bits: $[-2^{n-1}, (2^{n-1}-1)]$ → 1 more neg. than pos. number

## 2'S COMPLEMENT ADDITION

- Add numbers as you would for unsigned addition
- Examples with 4 bits:

```
    4   →      0100
-   4   →   +  1100   (subtract by negating 2nd number & adding)
  ------       ------
    0   →     ~~1~~0000  (ignore carry out since signs were different)

    5   →      0101
+   6   →   +  0110
  ------      -------
    11         1011   -- overflow (two positives, got a negative)
```

- 2's complement has one representation for 0 and arithmetic is easier
- It's the most commonly used negative number representation

## SIGNED BINARY NUMBERS

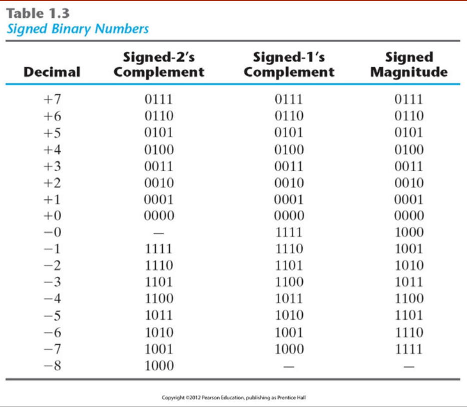


Table 1.3
*Signed Binary Numbers*

| Decimal | Signed-2's Complement | Signed-1's Complement | Signed Magnitude |
|---|---|---|---|
| +7 | 0111 | 0111 | 0111 |
| +6 | 0110 | 0110 | 0110 |
| +5 | 0101 | 0101 | 0101 |
| +4 | 0100 | 0100 | 0100 |
| +3 | 0011 | 0011 | 0011 |
| +2 | 0010 | 0010 | 0010 |
| +1 | 0001 | 0001 | 0001 |
| +0 | 0000 | 0000 | 0000 |
| −0 | — | 1111 | 1000 |
| −1 | 1111 | 1110 | 1001 |
| −2 | 1110 | 1101 | 1010 |
| −3 | 1101 | 1100 | 1011 |
| −4 | 1100 | 1011 | 1100 |
| −5 | 1011 | 1010 | 1101 |
| −6 | 1010 | 1001 | 1110 |
| −7 | 1001 | 1000 | 1111 |
| −8 | 1000 | — | — |

## GRAY CODES

- Represent decimal numbers 0-8 in binary such that only bit changes value as you count up/down
- Why would such an encoding be advantageous?

| Decimal | Gray code |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0011 |
| 3 | 0010 |
| 4 | 0110 |
| 5 | 0111 |
| 6 | 0101 |
| 7 | 0100 |
| 8 | 1100 |
| 9 | 1101 |

## 2'S COMPLEMENT SUBTRACTION

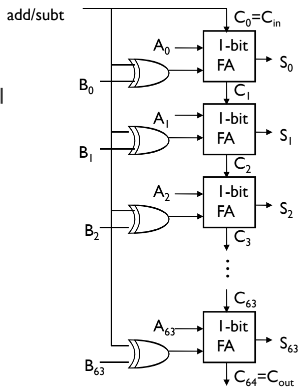
- Negate second operand and add:

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

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

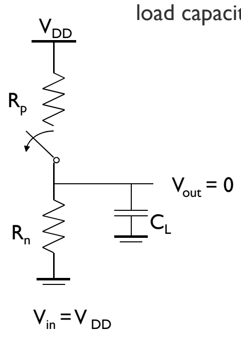
## A 64-BIT ADDER/SUBTRACTOR

- 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



## INVERTER PROPAGATION DELAY

- Propagation delay is proportional to the time-constant of the network formed by the pull-down resistor and the load capacitance.



$t_{pHL} = f(R_n, C_L)$

$$R_n = R_{eq} = \frac{3}{4}\frac{V_{DD}}{I_{DSAT}}$$

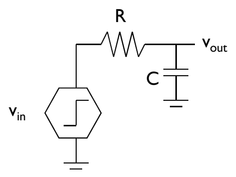
$$C_L = C_{int} + C_{ext} + C_{wire}$$

Want to have equal rise/fall delays

$\Rightarrow$ Make $R_n = R_p$

## MODELING PROPAGATION DELAY

- Model circuit as first-order RC network



$v_{out}(t) = (1 - e^{-t/\tau})V_{in}$

where $\tau = RC$

Time to reach 50% point is
$t = \ln(2)\ \tau = 0.69\ \tau$

Time to reach 90% point is
$t = \ln(9)\ \tau = 2.2\ \tau$

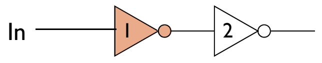
$$t_p = (t_{pHL} + t_{pLH})/2 = 0.69 C_L (R_n + R_p)/2$$

## HOW CAN WE IMPROVE PERFORMANCE?

- Reduce $R_n$, $R_p$
  - Increase W/L ratio of the transistor
    - most powerful and effective performance optimization tool for designer
    - But what happens to the intrinsic capacitance with larger W/L?
- Reduce $C_L$
  - Keep drain diffusions small
  - Limit interconnect capacitance
  - Limit fanout
- Increase $V_{DD}$
  - Trade off energy for performance
  - Increase $V_{DD}$ above a certain level yields minimal improvements
  - Reliability concerns enforce an upper bound on $V_{DD}$

## NMOS/PMOS RATIO

- So far we have sized the PMOS and NMOS so that the $R_{eq}$ values match (i.e., $\beta = (W/L_p)/(W/L_n) = W_p/W_n = 2$ 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:

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

- 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 $\beta$…

## NMOS/PMOS RATIO

- Given the equation for $t_p$:

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

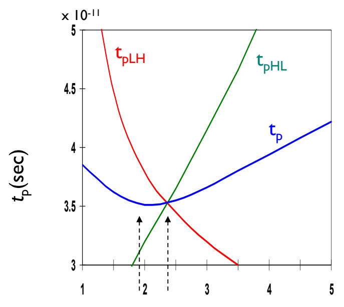
- Minimize $t_p$ as a function of $\beta$…
- Compute the optimal value of $\beta$ by setting $\partial t_p/\partial\beta = 0$

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

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



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

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

$\beta$ of 2.4 gives symmetrical response

$\beta$ 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 ($C_{int} = SC_{iref}$)

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

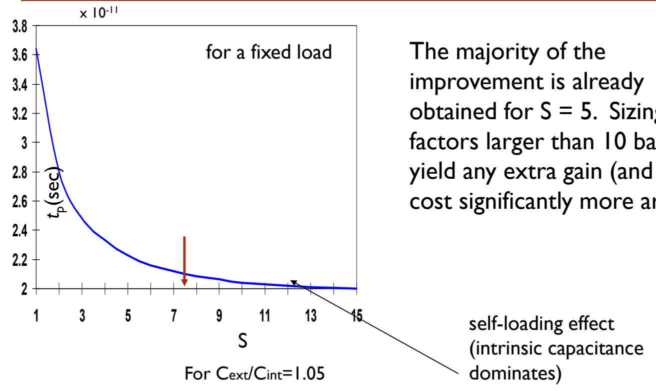
## DEVICE SIZING FOR PERFORMANCE

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

*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

## SIZING IMPACTS ON DELAY



The majority of the improvement is already obtained for S = 5. Sizing factors larger than 10 barely yield any extra gain (and cost significantly more area).