ESE534: Computer Organization

Day 3: January 23, 2012
Arithmetic

Work preclass exercise

Last Time

- Boolean logic ⇒ computing **any** finite function
- Saw gates…and a few properties of logic

Today

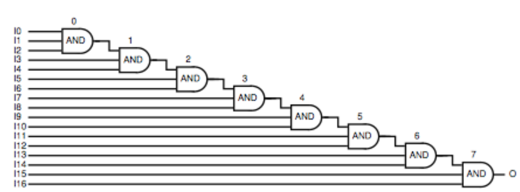
- Addition
  - organization
  - design space
  - parallel prefix

Why?

- Start getting a handle on
  - Complexity
    - Area and time
    - Area-time tradeoffs
  - Parallelism
  - Regularity
- Arithmetic underlies much computation
  - grounds out complexity

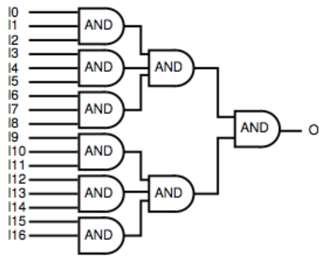
Preclass

Circuit 1

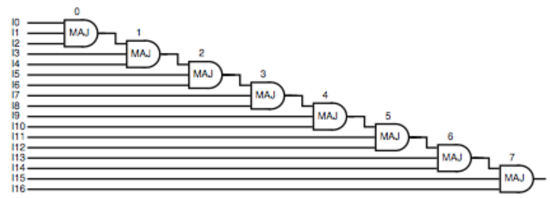


- Can the delay be reduced?
- How?
- To what?

Tree Reduce AND

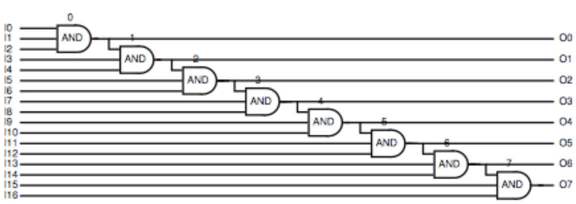


Circuit 2



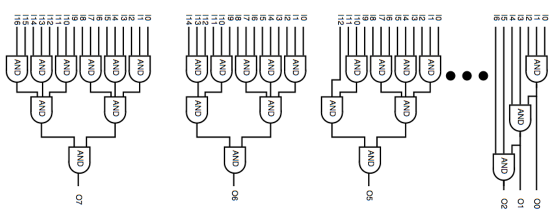
- Can the delay be reduced?

Circuit 3



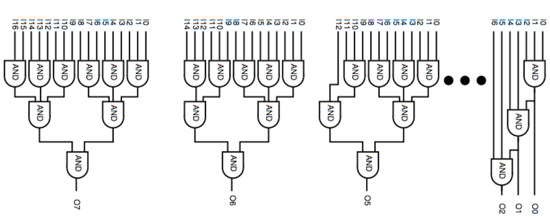
- Can the delay be reduced?

Brute Force Multi-Output AND



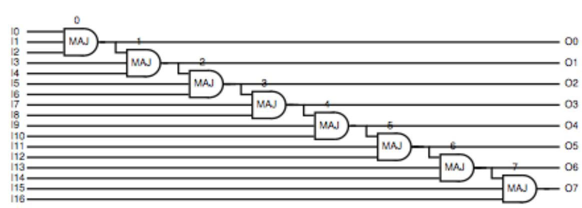
- How big?
- ~38 here
- … in general about $N^2/2$

Brute Force Multi-Output AND



- Can we do better?

Circuit 4



- Can the delay be reduced?

Addition

Example: Bit Level Addition

- Addition
  - Base 2 example
  - Work together

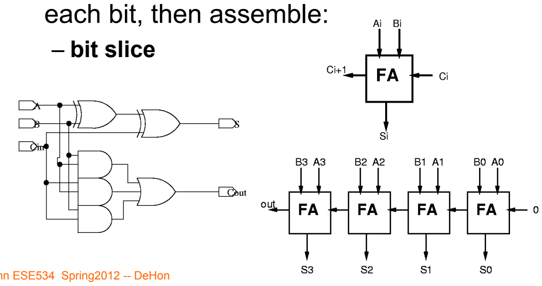
C: 11011010000
A: 01101101010
B: 01100101100
S: 11010010110

Addition Base 2

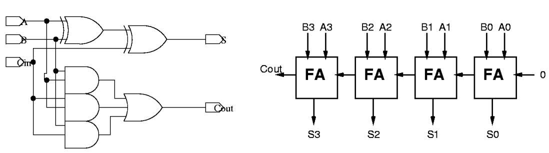
- $A = a_{n-1}*2^{(n-1)}+a_{n-2}*2^{(n-2)}+\ldots a_1*2^1+ a_0*2^0$
  $= \Sigma (a_i*2^i)$
- S=A+B
- What is the function for s_i … $carry_i$?
- s_i= $carry_i$ **xor** a_i **xor** b_i
- $carry_i = (a_{i-1} + b_{i-1} + carry_{i-1})\geq 2$
  $= a_{i-1}*b_{i-1}+a_{i-1}*carry_{i-1} + b_{i-1}*carry_{i-1}$
  $= MAJ(a_{i-1},b_{i-1},carry_{i-1})$

Ripple Carry Addition

- Shown operation of each bit
- Often convenient to define logic for each bit, then assemble:
  - **bit slice**



Ripple Carry Analysis

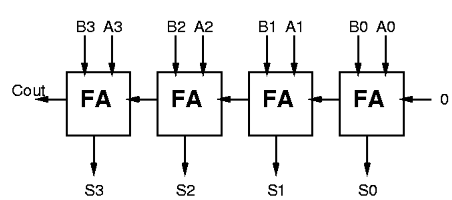


What is area and delay for N-bit RA adder? [unit delay gates]

- Area: O(N) [6n]
- Delay: O(N) [2n]

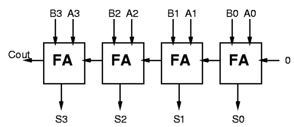
Can we do better?

- Lower delay?



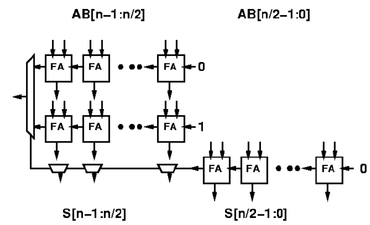
Important Observation

- Do we have to wait for the carry to show up to begin doing useful work?
  - We do have to know the carry to get the right answer.
  - How many values can the carry take on?



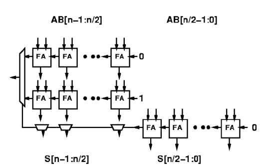
Idea

- Compute both possible values and select correct result when we know the answer



Preliminary Analysis

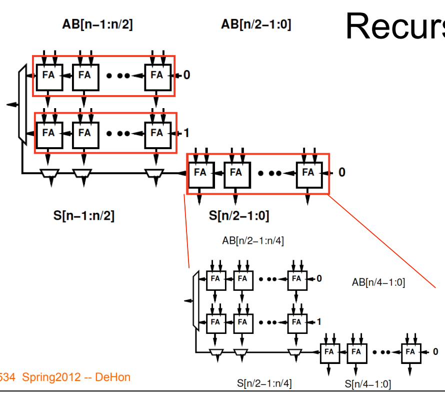
- Delay(RA) --Delay Ripple Adder
- Delay(RA(n)) = k*n [k=2 this example]
- Delay(RA(n)) = 2*(k*n/2)=2*DRA(n/2)
- Delay(P2A) -- Delay Predictive Adder
- Delay(P2A)=DRA(n/2)+D(mux2)
- …almost half the delay!



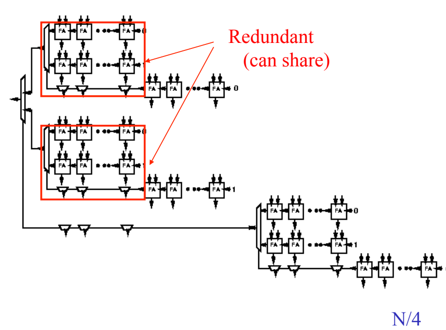
Recurse

- If something works once, do it again.
- Use the predictive adder to implement the first half of the addition

Recurse



Recurse



Recurse

- If something works once, do it again.
- Use the predictive adder to implement the first half of the addition
- Delay(P4A(n))=
  Delay(RA(n/4)) + D(mux2) + D(mux2)
- Delay(P4A(n))=Delay(RA(n/4))+2*D(mux2)

Recurse

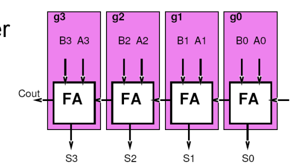
- By know we realize we've been using the wrong recursion
  - should be using the Predictive Adder in the recursion
- Delay(PA(n)) = Delay(PA(n/2)) + D(mux2)
- Every time cut in half…?
- How many times cut in half?
- Delay(PA(n))=$\log_2$(n)*D(mux2)+C
  - C = Delay(PA(1))
    - if use FA for PA(1), then C=2

Another Way

(Parallel Prefix)

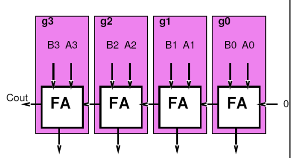
CLA

- Think about each adder bit as a computing a function on the carry in
  - C[i]=g(c[i-1])
  - Particular function f will depend on a[i], b[i]
  - g=f(a,b)



Functions

- What are the functions g(c[i-1])?
  - g(c)=carry(a=0,b=0,c)
  - g(c)=carry(a=1,b=0,c)
  - g(c)=carry(a=0,b=1,c)
  - g(c)=carry(a=1,b=1,c)

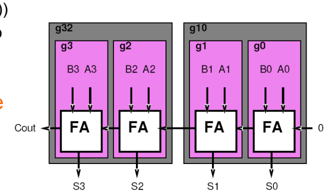


Functions

- What are the functions g(c[i-1])?
  - g(x)=1 Generate
    - a[i]=b[i]=1
  - g(x)=x Propagate
    - a[i] xor b[i]=1
  - g(x)=0 Squash
    - a[i]=b[i]=0

Combining

- Want to combine functions
  - Compute $c[i]=g_i(g_{i-1}(c[i-2]))$
  - Compute compose of two functions
- What functions will the compose of two of these functions be?
  - Same as before
    - Propagate, generate, squash



Compose Rules (LSB MSB)

- GG
- GP
- GS
- PG
- PP
- PS

- SG
- SP
- SS

[work on board]

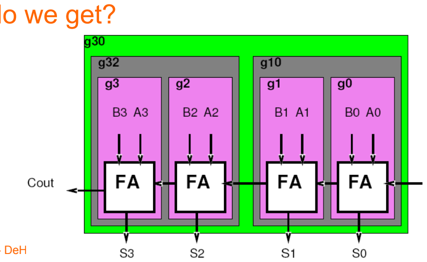
Compose Rules (LSB MSB)

- GG = G
- GP = G
- GS = S
- PG = G
- PP = P
- PS = S

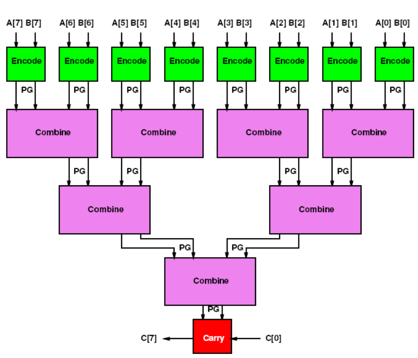
- SG = G
- SP = S
- SS = S

Combining

- Do it again…
- Combine g[i-3,i-2] and g[i-1,i]
- What do we get?



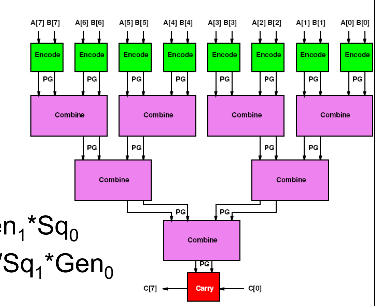
Reduce Tree



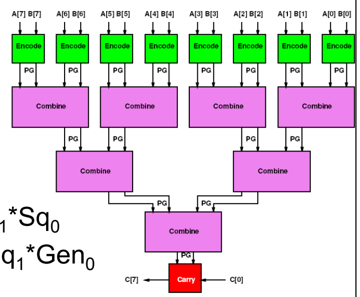
Reduce Tree

- Sq=/A*/B
- Gen=A*B

- Sq_{out}=Sq_1+/Gen_1*Sq_0
- Gen_{out}=Gen_1+/Sq_1*Gen_0



Reduce Tree



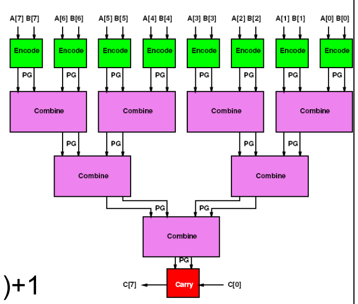
- Sq=/A*/B
- Gen=A*B
- $Sq_{out}=Sq_1+/Gen_1*Sq_0$
- $Gen_{out}=Gen_1+/Sq_1*Gen_0$
- Delay and Area?

Reduce Tree

- Sq=/A*/B
- Gen=A*B
- $Sq_{out}=Sq_1+/Gen_1*Sq_0$
- $Gen_{out}=Gen_1+/Sq_1*Gen_0$

- A(Encode)=2
- D(Encode)=1
- A(Combine)=4
- D(Combine)=2
- A(Carry)=2
- D(Carry)=1

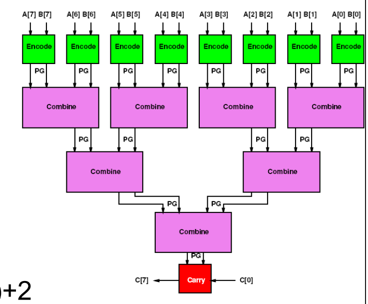
Reduce Tree: Delay?



- D(Encode)=1
- D(Combine)=2
- D(Carry)=1

Delay = $1+2\log_2(N)+1$

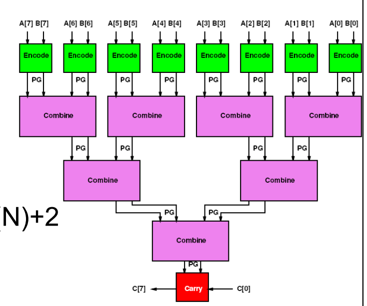
Reduce Tree: Area?



- A(Encode)=2
- A(Combine)=4
- A(Carry)=2

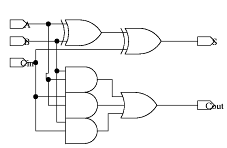
Area= 2N+4(N-1)+2

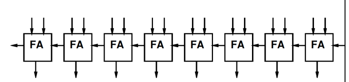
Reduce Tree: Area & Delay

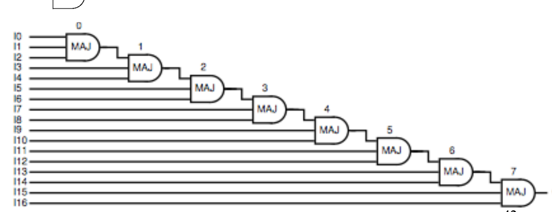


- Area(N) = 6N-2
- Delay(N) = $2\log_2(N)+2$

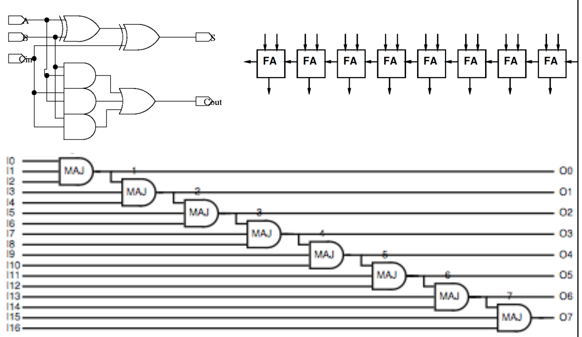
How Relate?







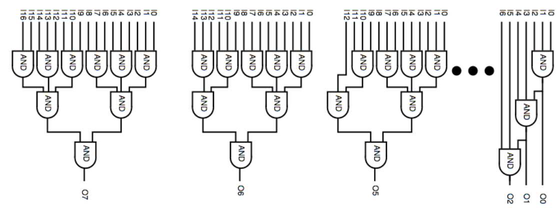
Need



Need PG[i:0] forall i

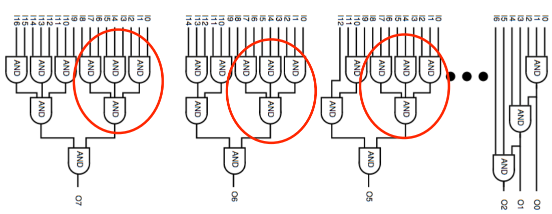
Intermediates

- Can we compute intermediates efficiently?



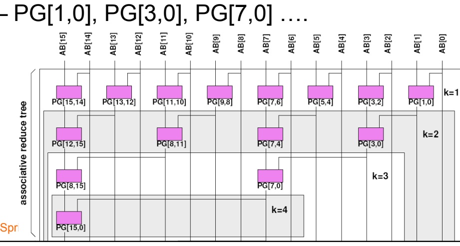
Intermediates

- Share common terms



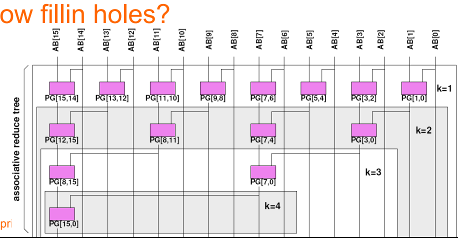
Reduce Tree

- While computing PG[N,0]
  compute many PG[I,j]'s
  – PG[1,0], PG[3,0], PG[7,0] ….



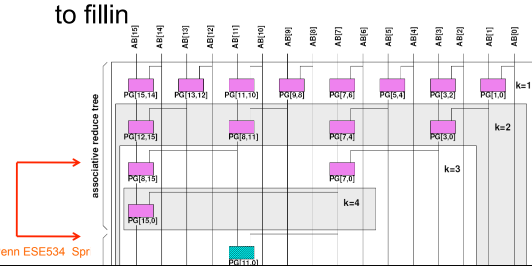
Prefix Tree

- While computing PG[N,0] only get
  – PG[2^n-1,0]
- How fillin holes?



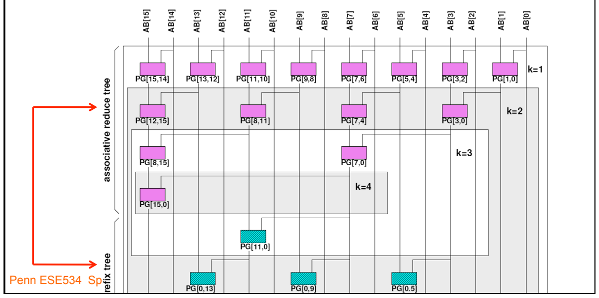
Prefix Tree

- Look at Symmetric stage (with respect to middle=PG[N,0] stage) and combine to fillin



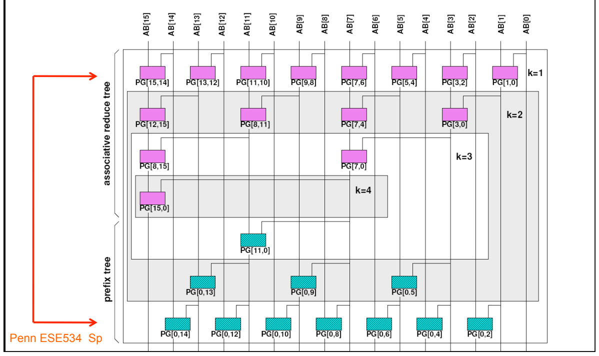
Prefix Tree

•



Penn ESE534 Sp

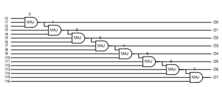
Prefix Tree

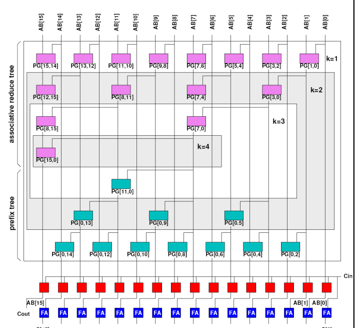


Penn ESE534 Sp

Prefix Tree

- Bring in Carry and compute each intermediate carry-in

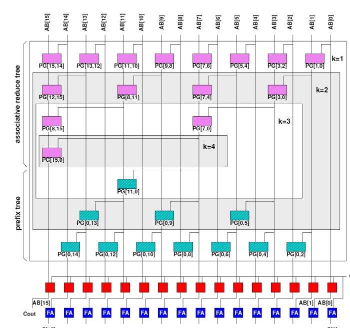




Penn ESE534 Spring2012 -- DeHon

Prefix Tree

- Note: prefix-tree is same size as reduce tree
  - Always matched same number of elements in symmetric stage

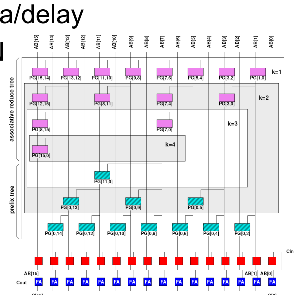


Penn ESE534 Spring2012 -- DeHon

Parallel Prefix
Area and Delay?

- Roughly twice the area/delay
- Area= 2N+4N+4N+2N
  = 10N
- Delay = $4\log_2(N)+2$

- Conclude:
  can add in log time with linear area.



Penn ESE534 Spring2012 -- DeHon

Parallel Prefix

- Important **Pattern**
- Applicable any time operation is *associative*
  - Or can be made assoc. as in MAJ case
- Examples of associative functions?
  - Non-associative?
- Function Composition is always associative

54

Penn ESE534 Spring2012 -- DeHon

Note: Constants Matter

- Watch the constants
- Asymptotically this Carry-Lookahead Adder (CLA) is great
- For small adders can be smaller with
  - fast ripple carry
  - larger combining than 2-ary tree
  - mix of techniques
- …will depend on the technology primitives and cost functions

Two's Complement

- positive numbers in binary
- negative numbers
  - subtract 1 and invert
  - (or invert and add 1)

Two's Complement

- 2 = 010
- 1 = 001
- 0 = 000
- -1 = 111
- -2 = 110

Addition of Negative Numbers?

- …just works

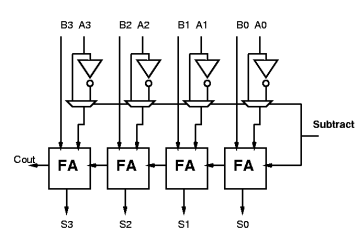
A: 111	A: 110	A: 111	A: 111
B: 001	B: 001	B: 010	B: 110
S: 000	S: 111	S: 001	S: 101

Subtraction

- Negate the subtracted input and use adder
  - which is:
    - invert input and add 1
    - works for both positive and negative input
      - 001 → 110 +1 = 111
      - 111 → 000 +1 = 001
      - 000 → 111 +1 = 000
      - 010 → 101 +1 = 110
      - 110 → 001 +1 = 010

Subtraction (add/sub)

- **Note:** you can use the "unused" carry input at the LSB to perform the "add 1"



Overflow?

A: 111
B: 001
S: 000

A: 110
B: 001
S: 111

A: 111
B: 010
S: 001

A: 111
B: 110
S: 101

A: 001
B: 001
S: 010

A: 011
B: 001
S: 100

A: 111
B: 100
S: 011

- Overflow=(A.s==B.s)*(A.s!=S.s)

Admin

- HW2 out today
- Reading for Wednesday online

Big Ideas [MSB Ideas]

- Can build arithmetic out of logic

Big Ideas [MSB-1 Ideas]

- Associativity
- Parallel Prefix
- Can perform addition
  - in log time
  - with linear area