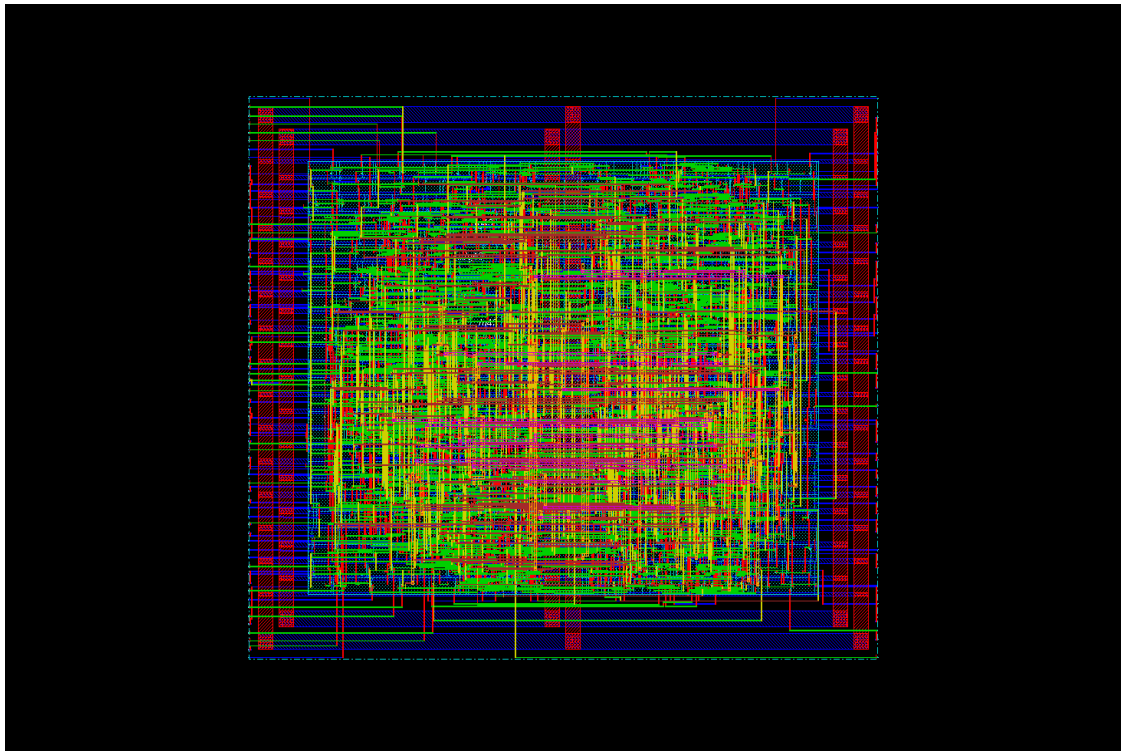
Adder design

going further . . .

exploring new design approaches

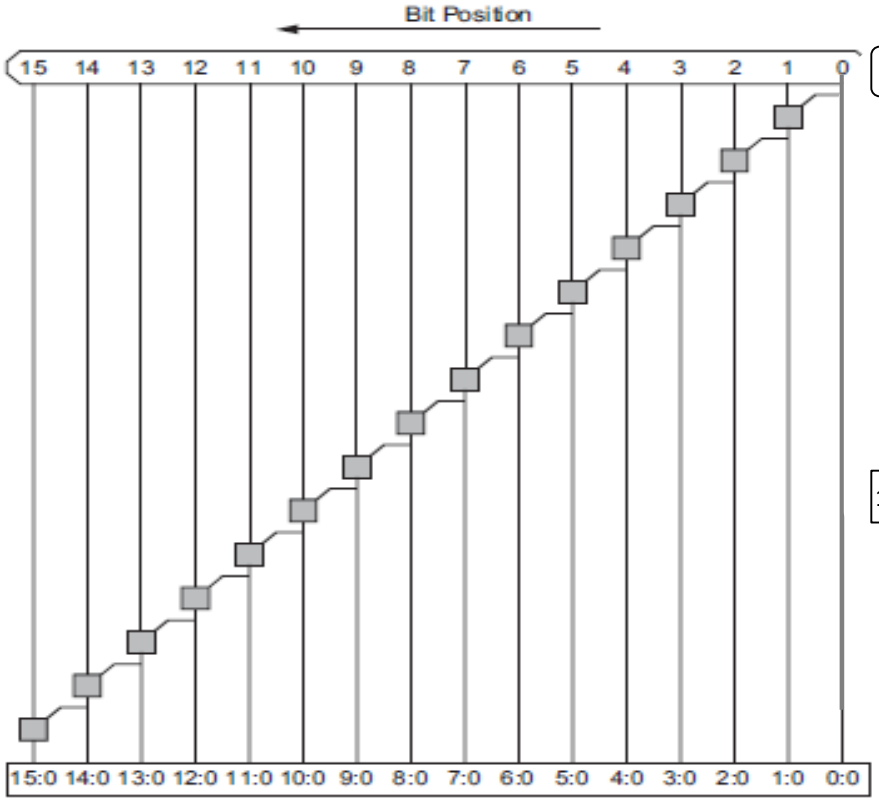
Purpose of this exercise

- Apply our CMOS circuit design skills for designing larger blocks, like adders
- Go through a step-by-step process for improving performance wrt timing constraints
- This exercise is a preparation for a larger design task in next quarter:
  - **A 32 bit ”super-fast” prefix binary-tree Sklansky adder**

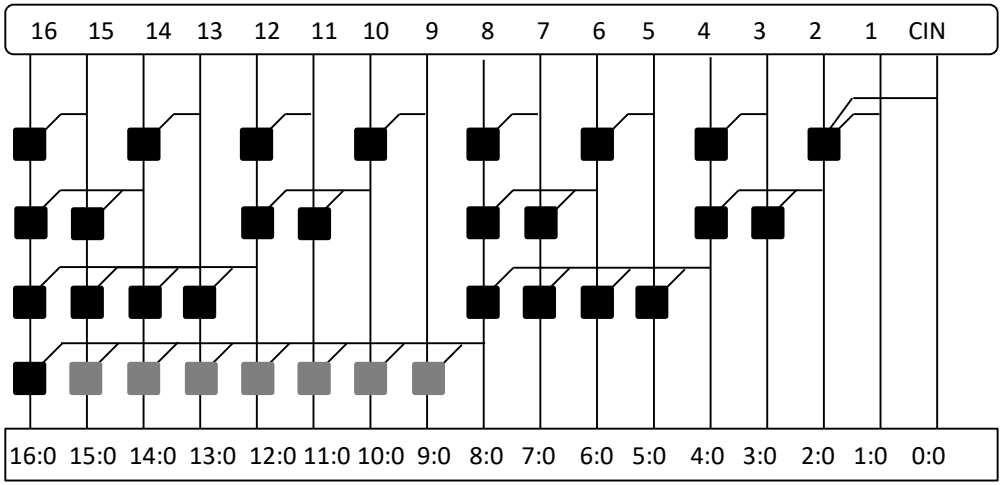


Purpose of this exercise

- Understanding Adder PG networks



Carry-ripple adder group PG network

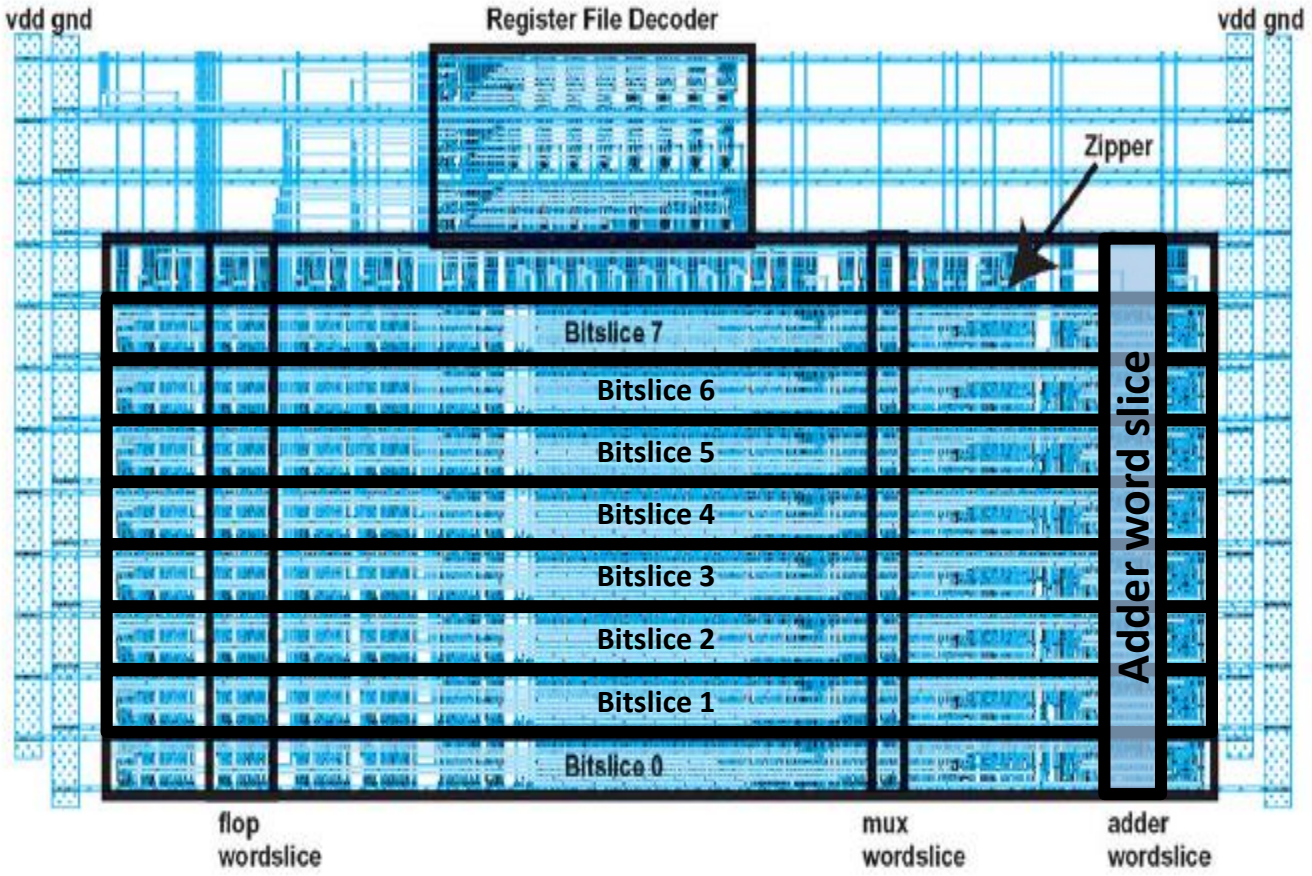


Sklansky type tree adder PG network

Outline

- Interactive lecture where we play around designing adders in Excel
- Why Excel?
  - Cell based like standard-cell design
  - Excel logic functions can be used to validate adder functionality
  - Unit delay model can be used to illustrate propagation delay
- Types of adders
  - Ripple-carry adder
  - Carry generate and propagate prefix setup logic to improve speed
  - Carry-skip adders
  - Carry-lookahead adders
  - Prefix tree adders, like Sklansky adders, Ladner-Fischer, Brent-Kung adders

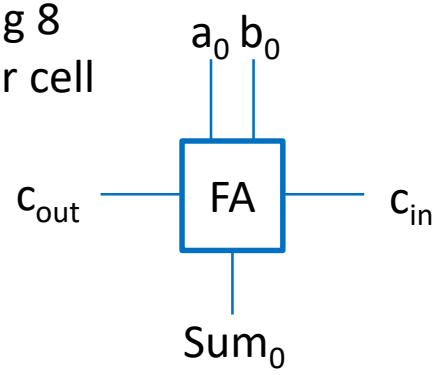
Datapath adder word slice

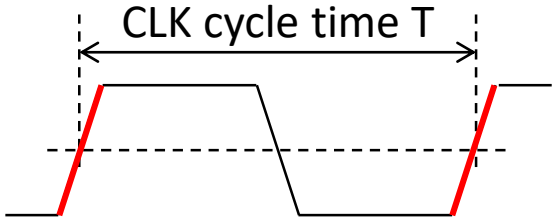


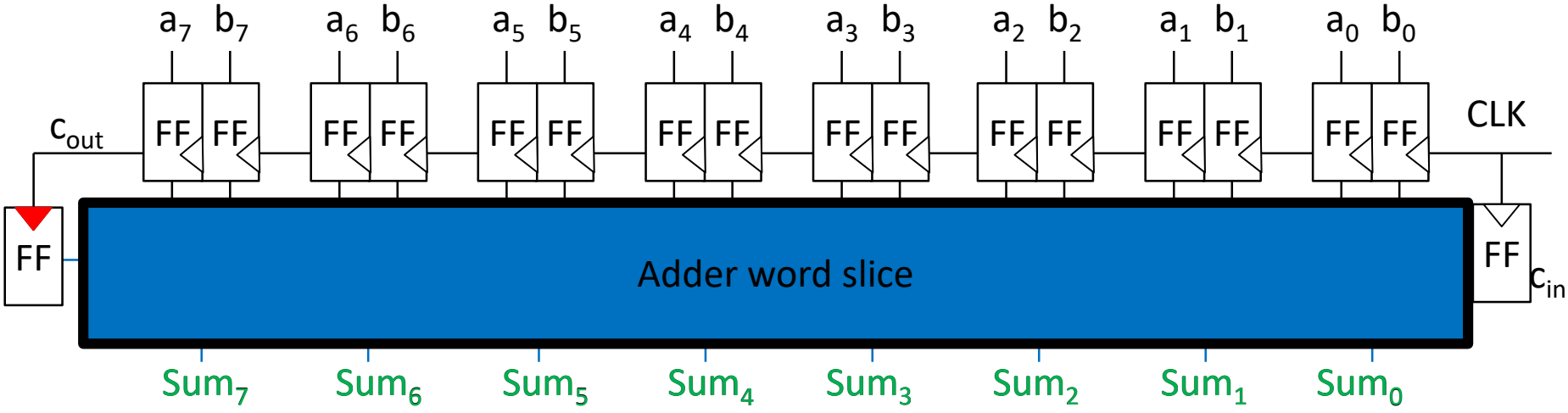
MIPS datapath layout with adder ”word slice”

Iterative logic arrays: FULL ADDER

Design adder by using 8 instances of bit adder cell

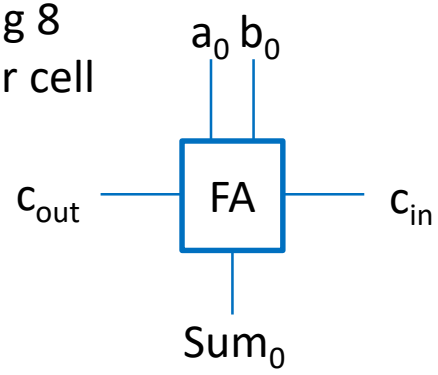


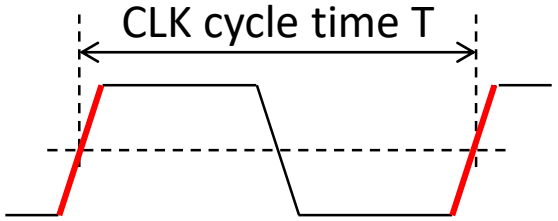


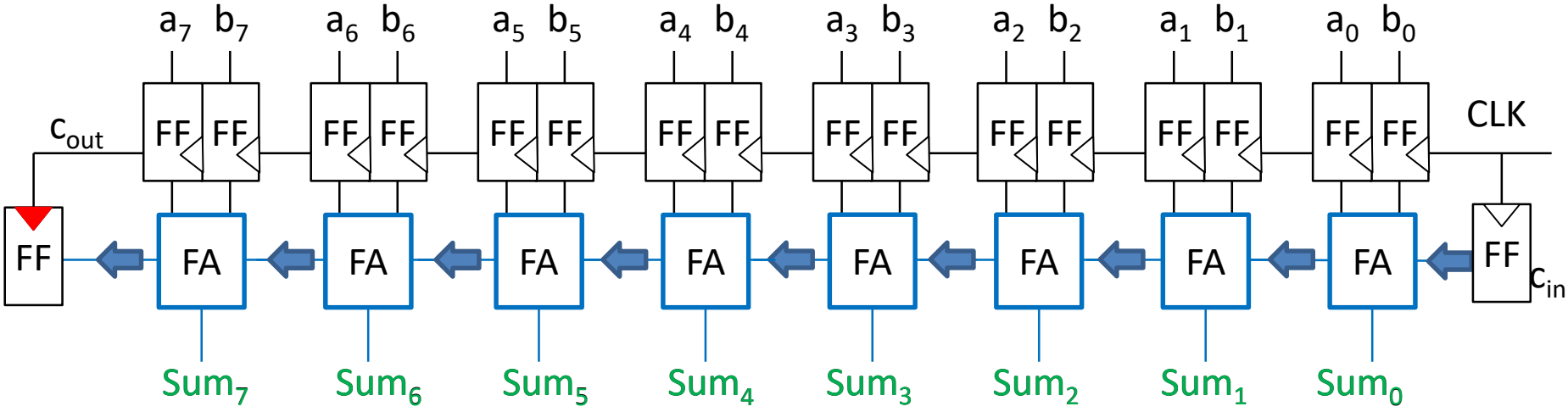


Iterative logic arrays: FULL ADDER

Design adder by using 8 instances of bit adder cell







Boolean truth table

You can check your Boolean expression by inserting it to the Boolean truth table first!
SUM expression already inserted as a hint on how to write logic in Excel!
SUM=OR(AND(A;B;CIN);AND(NOT(COUT);OR(A;B;CIN)))*1
(A, B, CIN, COUT to be replaced by work sheet cell coordinates)

BOOLEAN TRUTH TABLE						**LOGIC FORMULAS**		
	A	B	CIN	COUT	SUM		**COUT**	**SUM**
	0	0	0	0	0		0	0
	0	0	1	0	1		0	1
	0	1	0	0	1		0	1
	0	1	1	1	0		1	0
	1	0	0	0	1		0	1
	1	0	1	1	0		1	0
	1	1	0	1	0		1	0
	1	1	1	1	1		1	1

Ripple-carry adder design in Excel

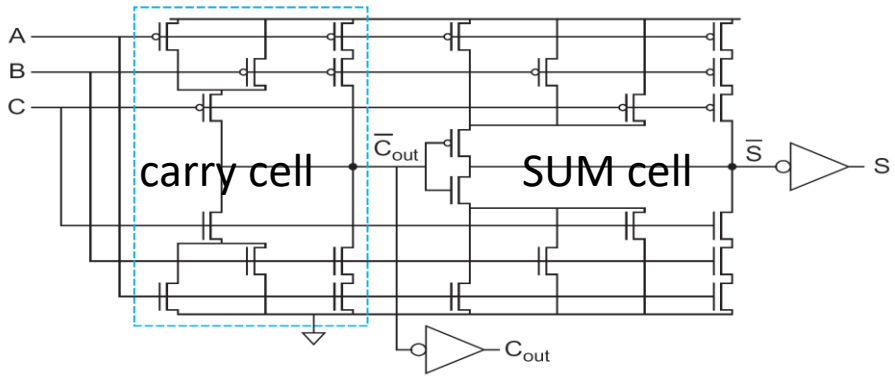
This is a geometrical representation of a ripple-carry adder.
Each row in in Excel work sheet corresponds to 2.6 μm in ST 65 nm CMOS cell library.
Next task: implement your carry cell using the adder template where the SUM cell logic has already been implemented.

←	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	←	CLK
	a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1	↓	DATA IN
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	↓	
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	↓	ADD/SUB LOGIC
←	0		0		0		0		0		0		0		0		0	CIN
	0		0		1		0		0		0		0		1			SUM LOGIC
	SUM8		SUM7		SUM6		SUM5		SUM4		SUM3		SUM2		SUM1			
	FF		FF		FF		FF		FF		FF		FF		FF		←	← CLK

YOUR TASK IS
1) TO WRITE CARRY LOGIC EXPRESSION
INTO LSB CARRY CELL USING EXCEL AND/OR/NOT LOGIC FORMULA, AND
2) CLICK-AND-DRAG TO GET 8 INSTANCES OF CARRY CELL
CARRY LOGIC ”=(LOGIC EXPRESSION)*1”

Text book full adder block solution

SUM LOGIC ”=OR(AND(A;B;C);AND(IN;OR(A;B;C)))*1”



(b)

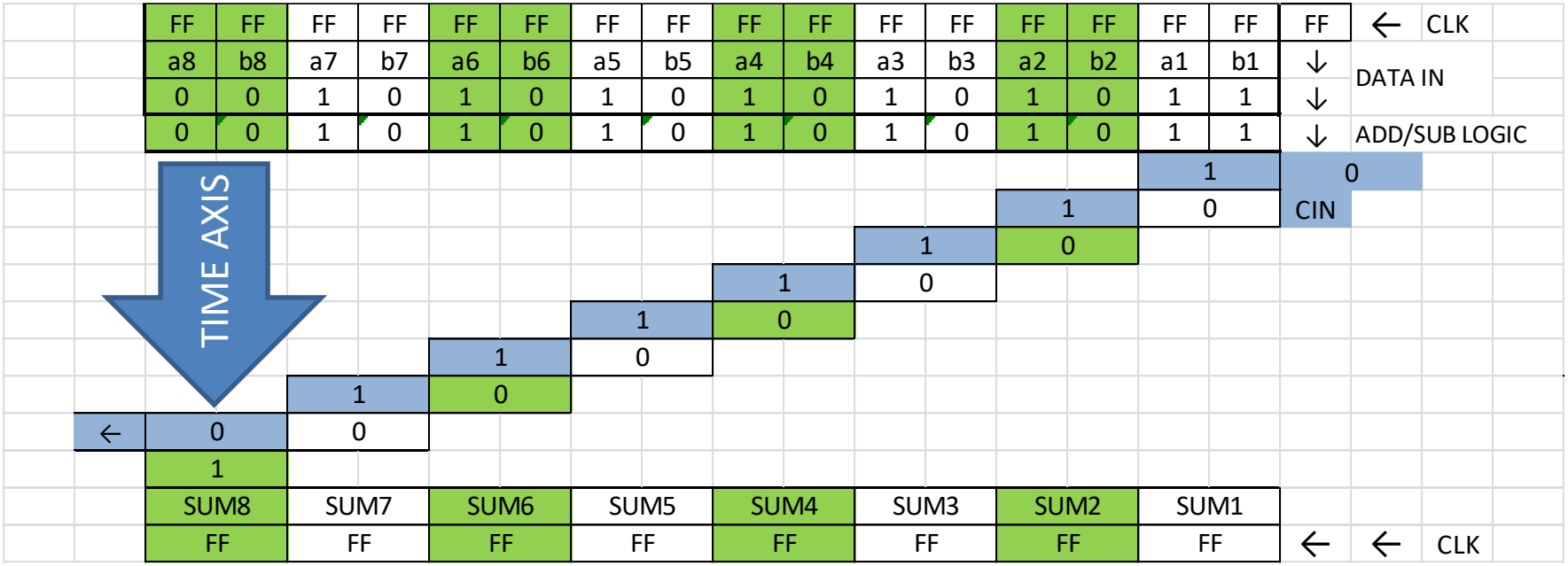
Ripple-carry adder design in Excel

	ADD/SUBTRACT			ADD=0	CIN=?	A=	32	<<<<<	ENTER TWO NUMBERS										
	CONTROL SIGNAL	0		SUB=1	CIN=?	B=	1	<<<<<	-128<NUMBER<128										
						SUM=	33												
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	←	CLK
	a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1	↓		DATA IN
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	↓		
	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	↓		ADD/SUB LOGIC
←	0		0		0		0		0		0		0		0		0		CIN
	0		0		1		0		0		0		0		1				SUM LOGIC
	SUM8		SUM7		SUM6		SUM5		SUM4		SUM3		SUM2		SUM1				
	FF		FF		FF		FF		FF		FF		FF		FF		←	←	CLK
	SUM converted back to decimal:				33		Both sums are equal?			YES									
							OVERFLOW?			NO									

Next task: Validate your design by checking its functionality!
Does it work also for subtraction?

Ripple-carry adder design in Excel

Teacher demo: illustrating the carry-ripple delay by introducing a timing axis, and by letting each cell row height represent a unit delay. CUT-AND-PASTE exercise. Each row in Excel is a timing representation using unit-delay model.



Ripple-carry adder design in Excel

Teacher demo: illustrating the carry-ripple delay by introducing a timing axis, and by letting each cell row height represent a unit delay. CUT-AND-PASTE exercise. Each row in Excel is a timing representation using unit-delay model.

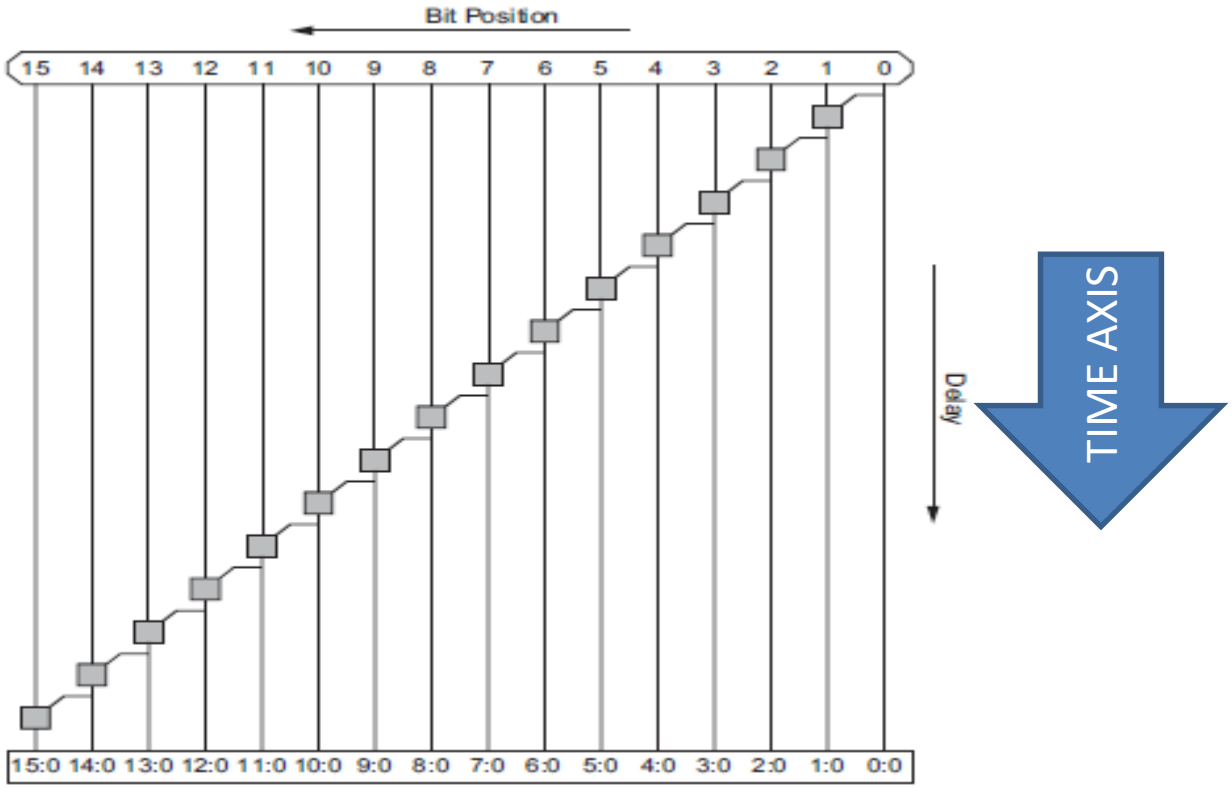
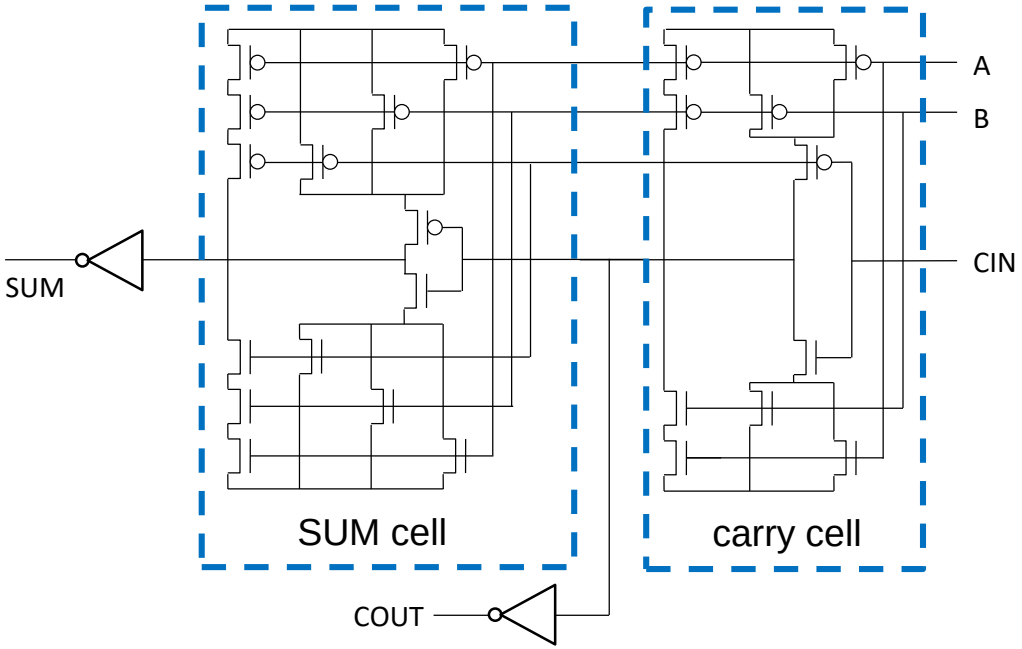


FIGURE 11.15 Carry-ripple adder group PG network

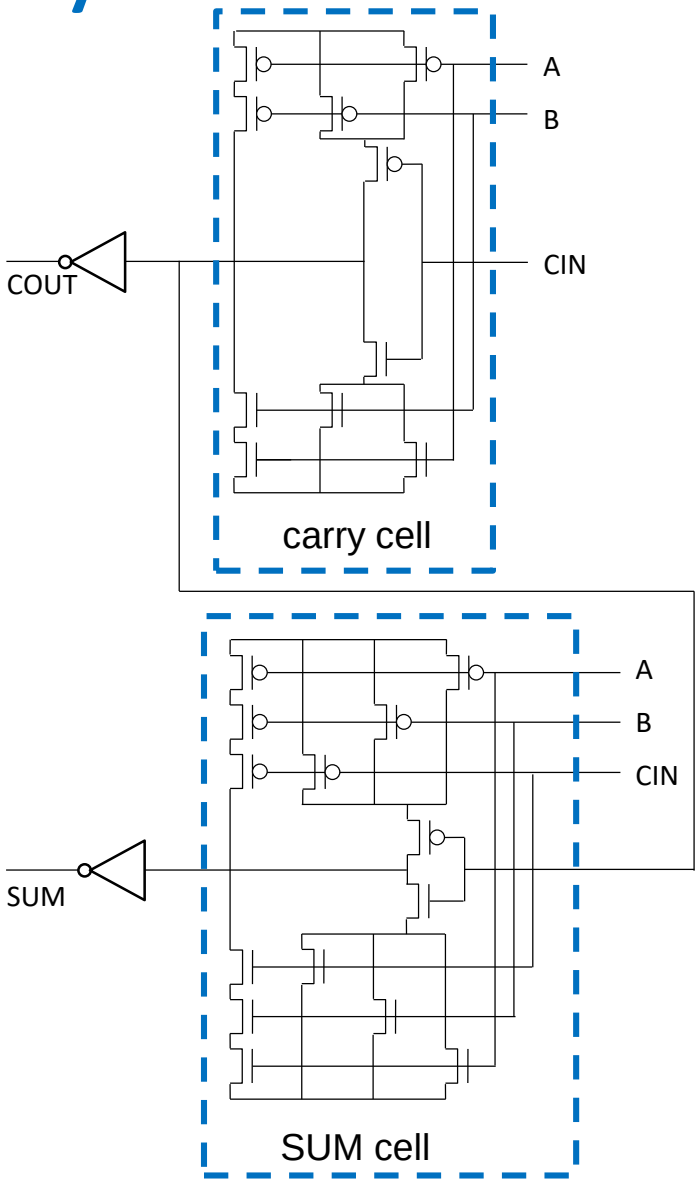
Full textbook solution as proposed in chapter 10 on adders

Ripple-carry delay calculations



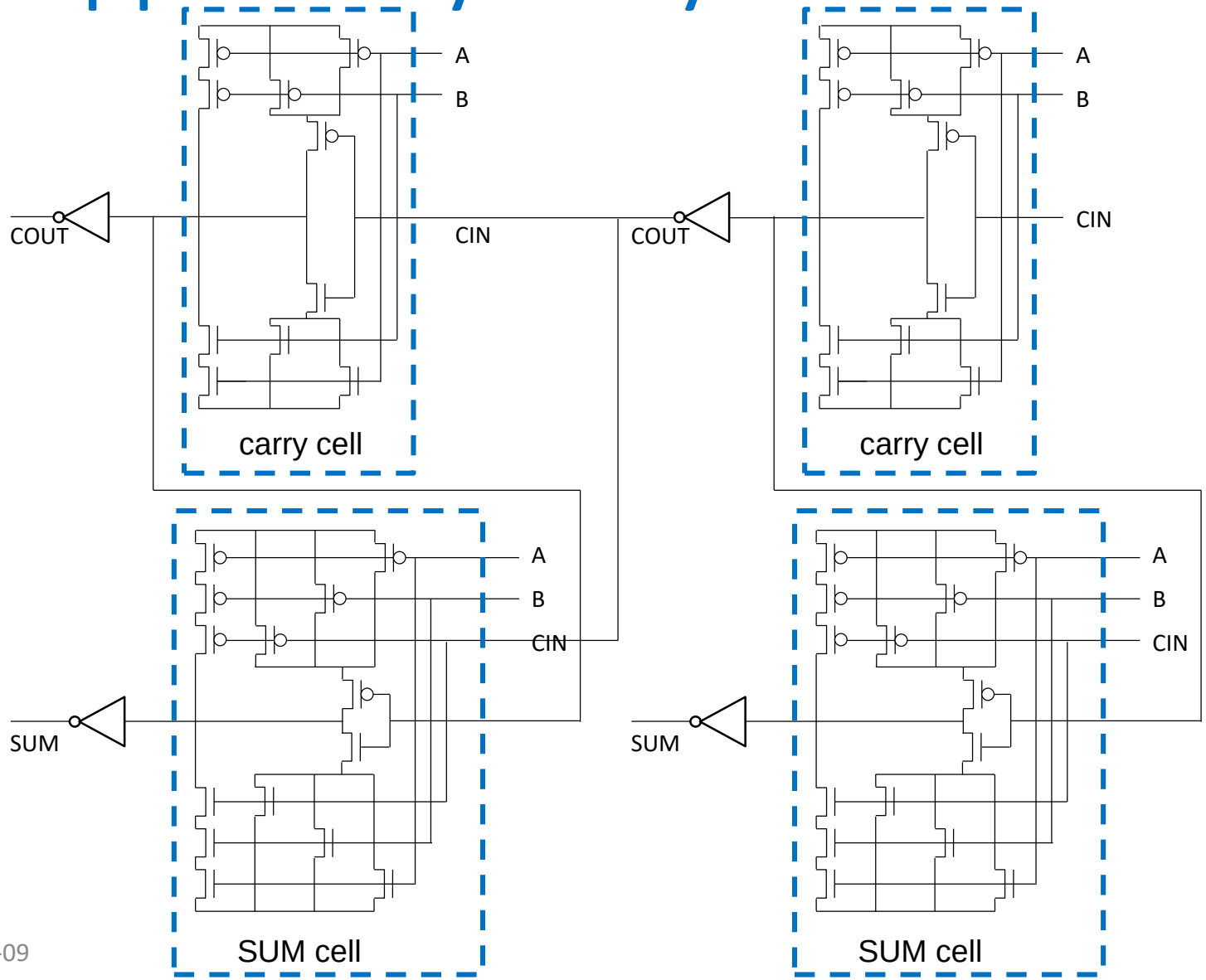
Redraw for identifying the critical timing path

Ripple-carry delay calculations

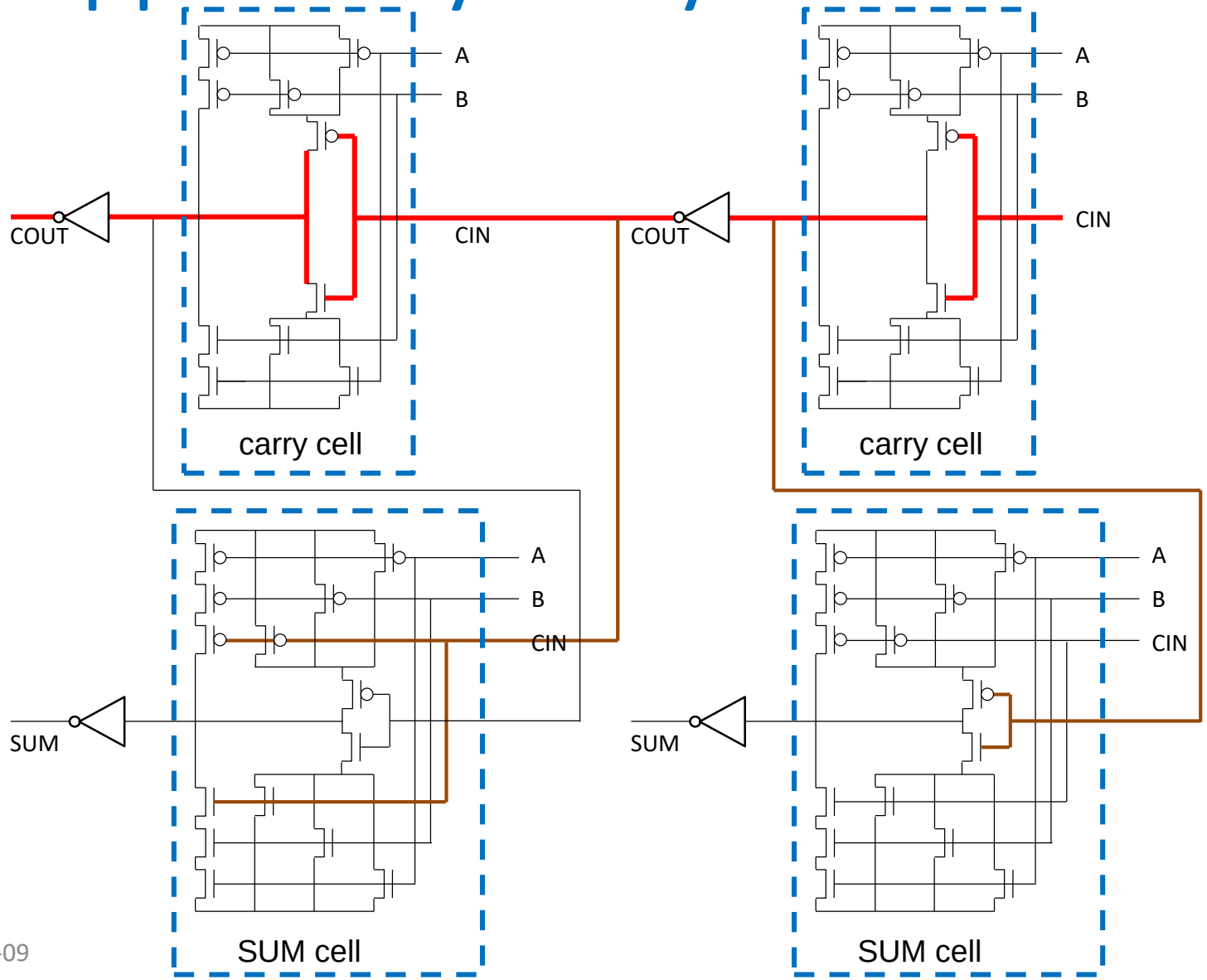


Add next bit cell for identifying the critical timing path

Ripple-carry delay calculations



Ripple-carry delay calculations

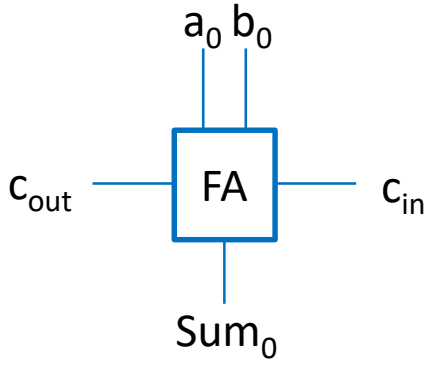


Improving the design

Finding faster carry cells!
Less complicated cell with smaller number of MOSFETs!
We are going to accomplish this by preparing the carry cell input signals!
As an example: SUM=XOR(A;B;CIN),
here P=XOR(A;B) can be prefixed while we are waiting for the carry!

Go back to Boolean truth table

Design adder by using 8 instances of bit adder cell

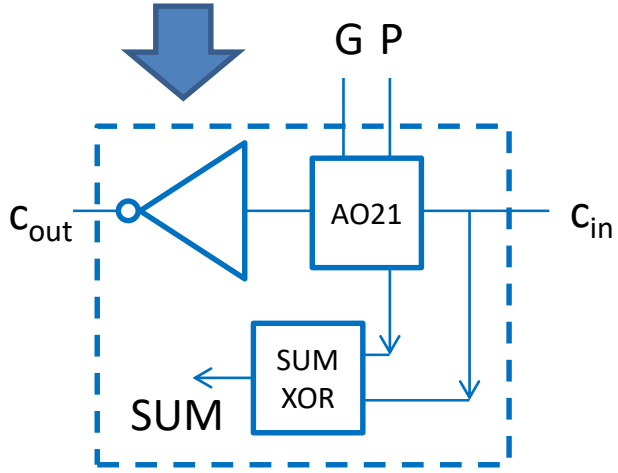


NOTE: COUT=Generate+Propagate*CIN

BOOLEAN TRUTH TABLE					
	A	B	CIN	COUT	SUM
	0	0	0	0	0
	0	0	1	0	1
	0	1	0	0	1
	0	1	1	1	0
	1	0	0	0	1
	1	0	1	1	0
	1	1	0	1	0
	1	1	1	1	1

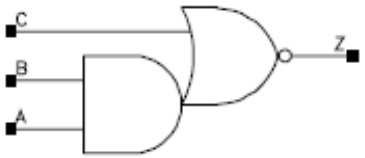
Divide and conquer: Divide full adder cell into carry and SUM cells!

G=AND(A;B)
P=XOR(A;B)



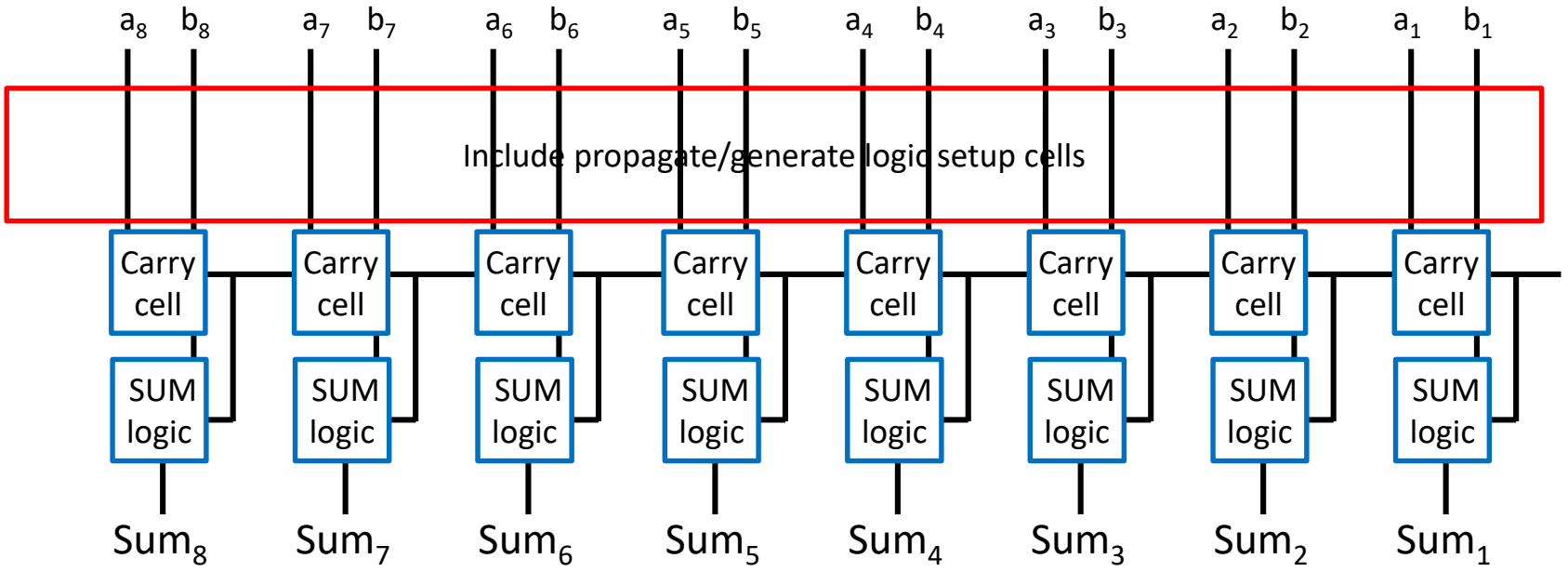
HS65_LS_AOI12

Logical Symbol

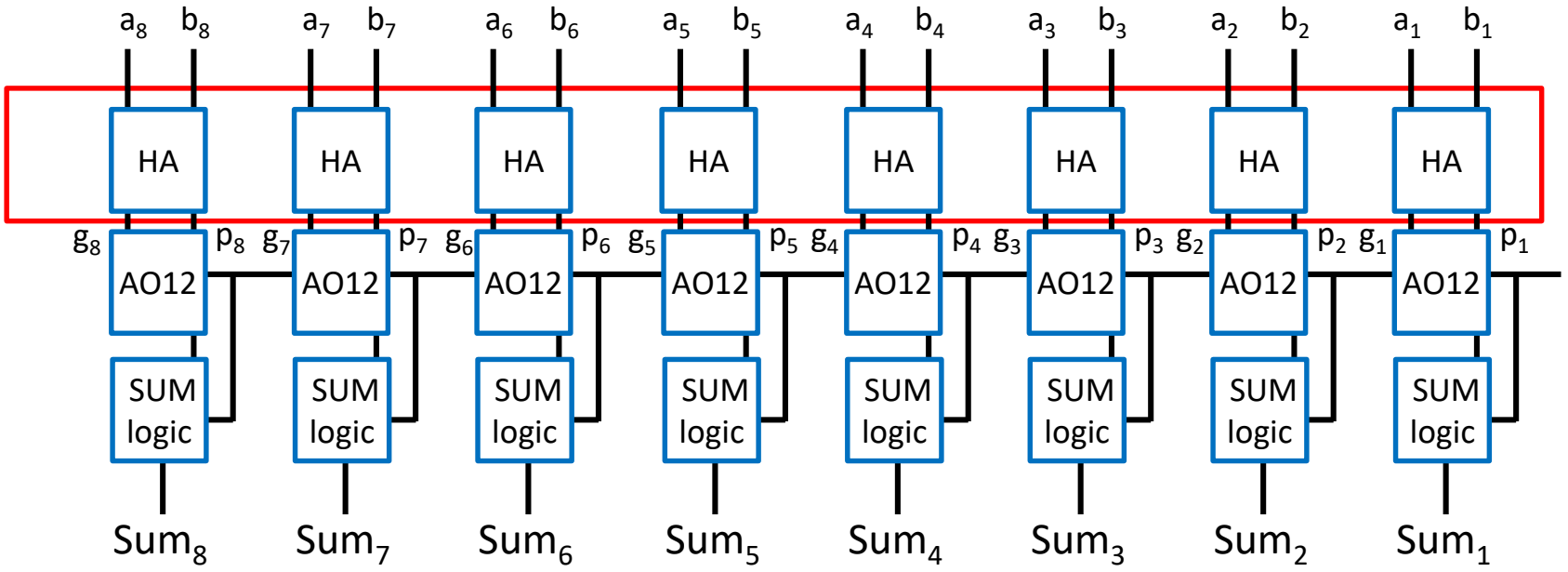


SUM=XOR(P;CIN)

Our adder design so far . . .

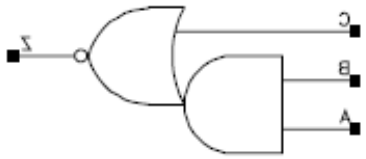


Using Propagate/Generate setup cells

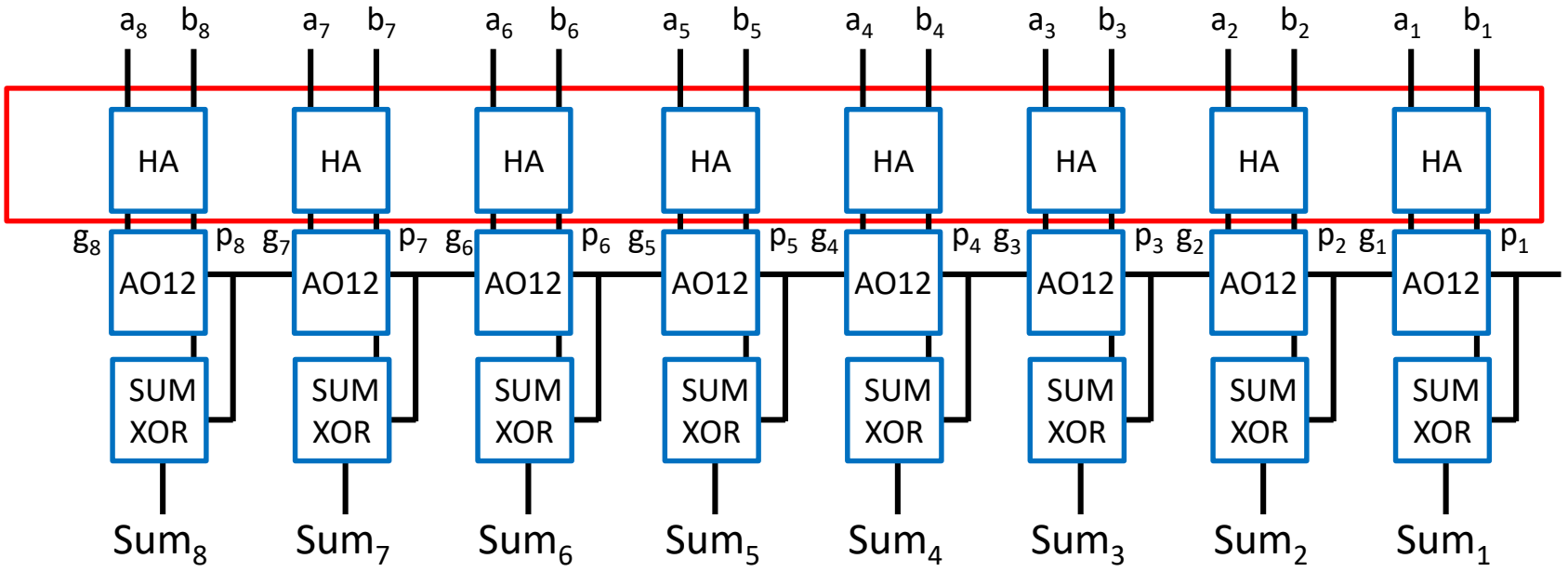


HS65_LS_AOI12

Logical symbol

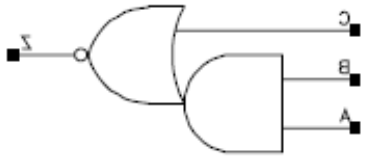


Using Propagate/Generate setup cells



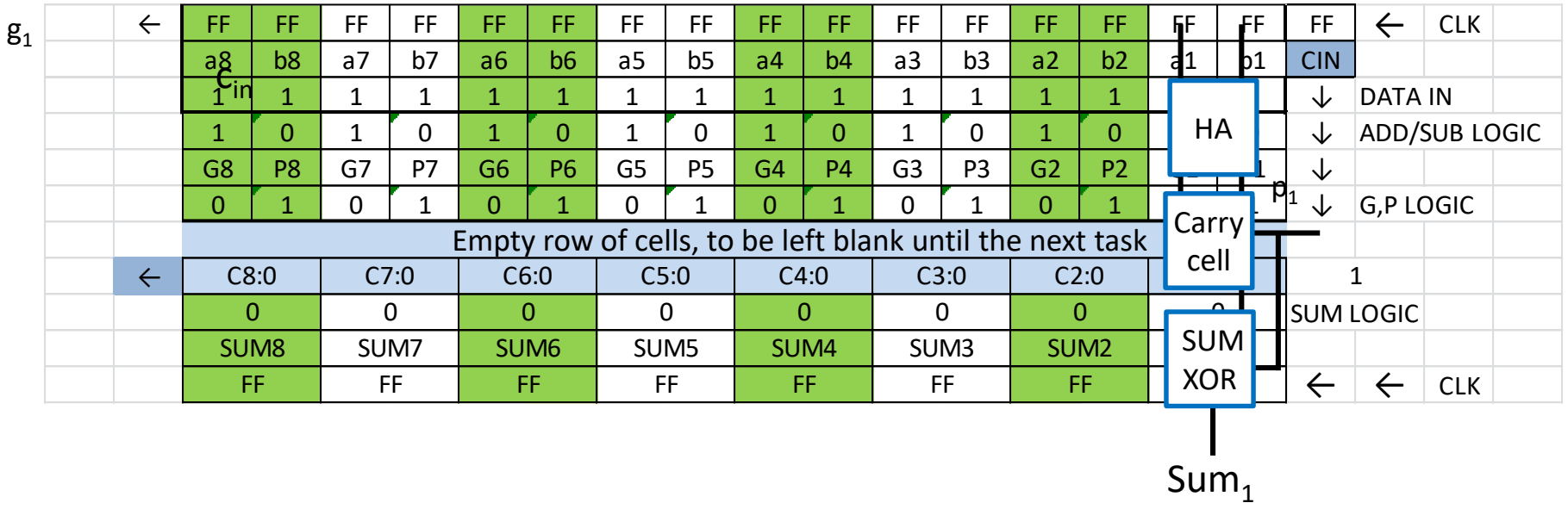
HS65_LS_AOI12

Logical symbol



Using Propagate/Generate setup cells

Fourth task: Enter G and P logic, enter carry AO21 logic and replace SUM cell logic with XOR logical function!



Note: P and G is never =1 at the same time! It is either propagate or generate, never both!
Therefore the AO logic expression in the C1:0 cells can be written very simple like =G1+P1*CIN!

Using Propagate/Generate setup cells

Fifth task: Let the combined propagate signal ripple through the adder array along with the carry signal

	←	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	←	CLK
		a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1	CIN	
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	↓	DATA IN
		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	↓	ADD/SUB LOGIC
		G8	P8	G7	P7	G6	P6	G5	P5	G4	P4	G3	P3	G2	P2	G1	P1	↓	
		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	↓	G,P LOGIC
			P8:1		P7:1		P6:1		P5:1		P4:1		P3:1		P2:1		P1:1		1
	←	C8:0		C7:0		C6:0		C5:0		C4:0		C3:0		C2:0		C1:0		1	
		0		0		0		0		0		0		0		0		SUM LOGIC	
		SUM8		SUM7		SUM6		SUM5		SUM4		SUM3		SUM2		SUM1			
		FF		FF		FF		FF		FF		FF		FF		FF		←	← CLK

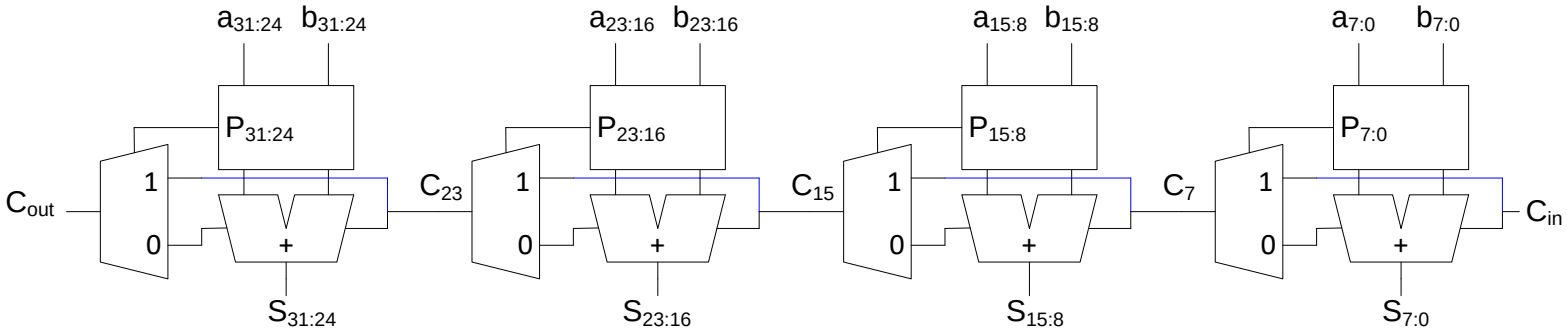
Note: P and G is never =1 at the same time! It is either propagate or generate, never both!
Therefore the AO logic expression in the C1:0 cells can be written very simple like =G1+P1*CIN!
Next improvement is letting the combined propagate signal ripple along with the carry signal to the array output.
Why: The carry-skip adder!

Carry-Skip Adder

- Carry-ripple is slow through all N stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal

N-bit adder

n-bit blocks



k=N/n number of n-bit blocks

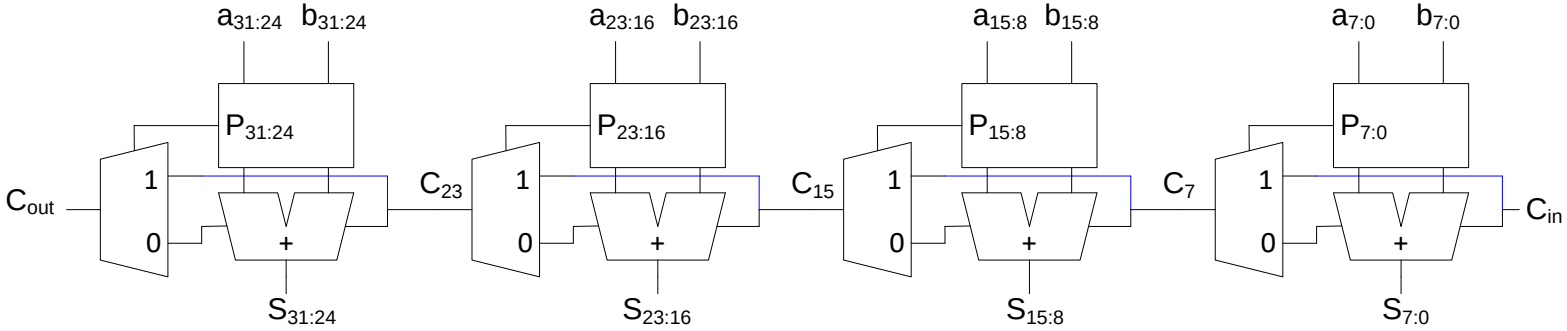
Verify the delay model given in eq. 10.13: $t_{skip} = t_{pg} + 2(n-1)t_{AO} + (k-1)t_{mux} + t_{XOR}$

Carry-Skip Adder

- Carry-ripple is slow through all N stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal

N-bit adder

n-bit blocks



k=N/n number of n-bit blocks

For 32-bit adder we had 31*t_{AO}; Now only t_{PG}+14t_{AO} + 3t_{MUX}
Delay is cut in half!

Using Propagate/Generate setup cells

		FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	←	CLK
		a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1	CIN	
		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	↓	DATA IN
		1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	↓	ADD/SUB LOGIC
		G8	P8	G7	P7	G6	P6	G5	P5	G4	P4	G3	P3	G2	P2	G1	P1	↓	
		0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	↓	G,P LOGIC
		G8:1	P8:1	G7:1	P7:1	G6:1	P6:1	G5:1	P5:1	G4:1	P4:1	G3:1	P3:1	G2:1	P2:1	G1:1	P1:1	↓	
	←			C7:0		C6:0		C5:0		C4:0		C3:0		C2:0		C1:0		1	
		0		0		0		0		0		0		0		0		SUM LOGIC	
		SUM8		SUM7		SUM6		SUM5		SUM4		SUM3		SUM2		SUM1			
		FF		FF		FF		FF		FF		FF		FF		FF		←	← CLK

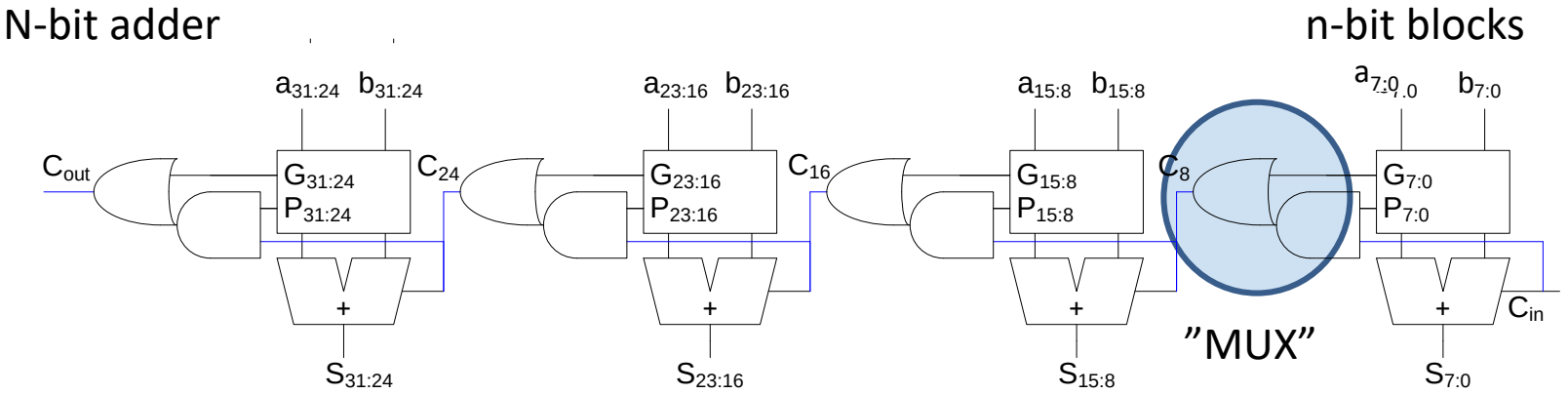
Once we realize that the carry arriving at the adder block output is actually a block generate signal, we are ready for the next improvement step!

THE CARRY-LOOKAHEAD ADDER

Why is that so? Because if $G_{8:1}$=1, a carry has been generated within the block!

Carry-Lookahead Adder

- Carry generate allows us to replace MUX with another AO21 gate.



k=N/n groups of n-bit blocks

Verify the delay model given in eq. 10.14: $t_{CLA} = t_{pg} + 2(n-1)t_{AO} + (k-1)t_{AO} + t_{XOR}$

Summary

We have

- introduced Microsoft Excel as a tool for studying adder designs
- introduced Excel timing graphs
- identified the generate/propagate behaviour of the addition
- built a new G and P based adder using AO21 cells for the carry
- built a 32-bit carry-skip adder with reduced propagation delay compared with the ripple-carry solution
- introduced the carry–lookahead concept using block G and P
- used a binary tree design to form the block P and G outputs

Next week: a close look at our G/P cell design name it the ”dot operator” and, see how it can be used to build pre-fix tree adders

To think about until next Tuesday

- Task 1: Try to appreciate the Excel adder design approach ☺
- Task 2: Verify the carry-skip propagation delay formula given in textbook equation 10.13.

$$t_{skip} = t_{pg} + 2(n-1)t_{AO} + (k-1)t_{mux} + t_{XOR}$$

- Task 3: Verify the carry-lookahead propagation delay formula given in textbook equation 10.14.

$$t_{CLA} = t_{pg} + t_{pg(n)} + (n-1)t_{AO} + (k-1)t_{AO} + t_{XOR}$$

- Task 4: Use the "dot operator" to show that the binary prefix-tree carry output is equal to the carry output from a ripple-carry array!

Example:
synthesized 32-bit ALU/adder layout