Interactive adder exercises

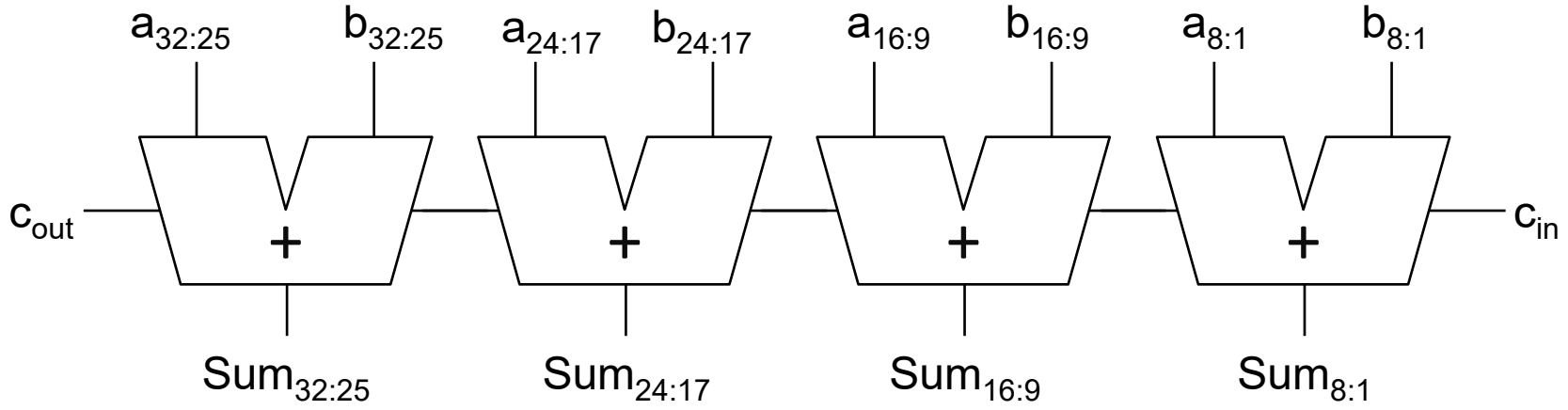
Studion

Lecture 2

2017

Purpose of todays lecture

- To learn how to design a 32-bit adder/subtractor where all sum cells have received their incoming carry signals after only 5 AND-OR delay units! Will be synthesized in Methods course.
- The 5 units delay is to be compared to the delay of a ripple-carry adder: 31 AND-OR units delay, see figure below.
- Or a ripple-carry lookahead adder with 17 AND-OR unit delays



Background

We assume that you have completed the first ripple-carry adder in the excel template!

	ADD/SUBTRACT			ADD=0			A=	-100	<<<<<	**ENTER TWO NUMBERS**							
	CONTROL SIGNAL	1		SUB=1			B=	-120	<<<<<	-128<NUMBER<128							
							SUM=	20										
SUM result converted back to decimal:						20	Both sums are equal?			YES	OVERFLOW?			NO				
	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF		
	a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1		
	1	0	0	1	0	1	1	1	1	0	1	1	0	1	0	1		
←	1		1		1		1		1		1		1		1		1	←CIN
	0		0		0		1		0		1		0		0		SUM LOGIC	
	SUM8		SUM7		SUM6		SUM5		SUM4		SUM3		SUM2		SUM1			
	FF		FF		FF		FF		FF		FF		FF		FF			

Ripple-carry timing

- And that you have made yourself familiar with the timing of such a ripple-carry adder.

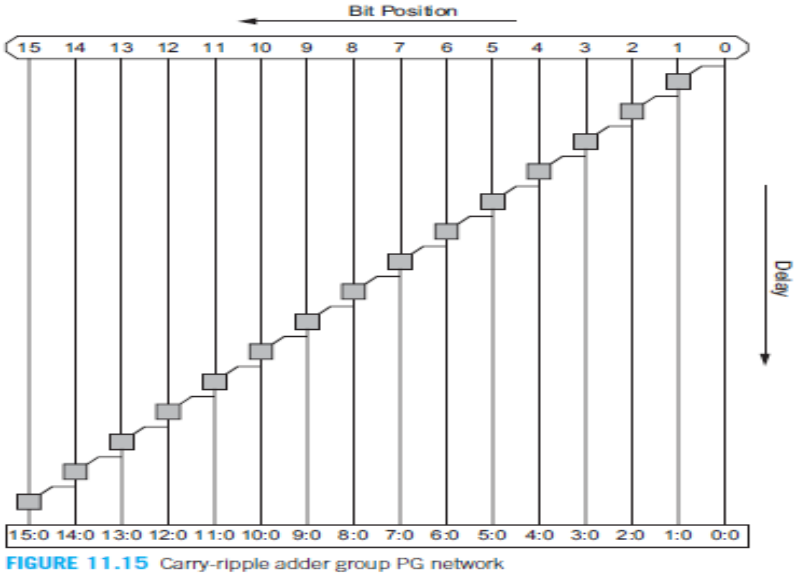
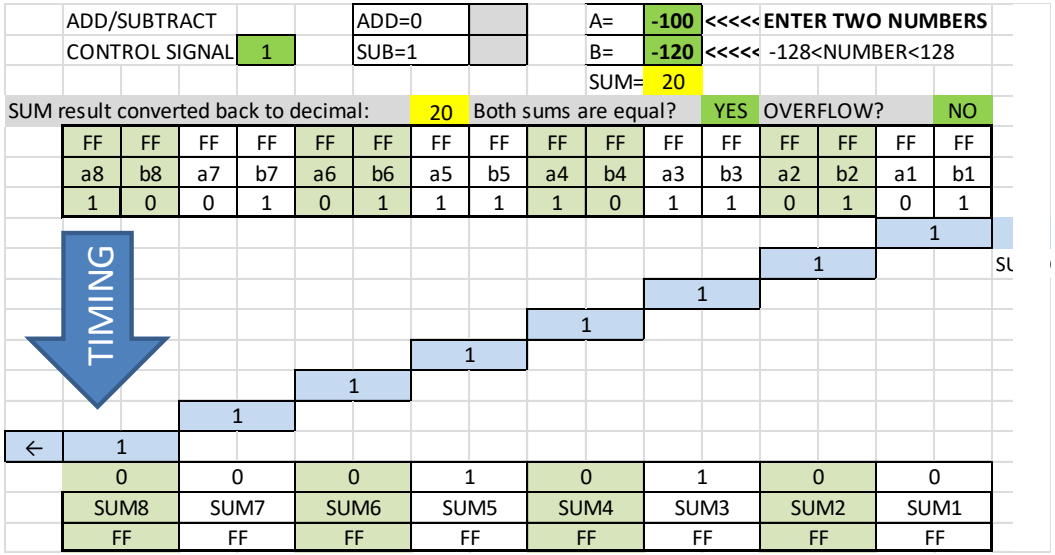


FIGURE 11.15 Carry-ripple adder group PG network



Propagate/generate setup

- We have redesigned adder with G and P setup to get a less complex carry cell!
- The delay time for setting up the bit-G and bit-P signals is denoted t_{pg} in the textbook!

	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF		
	a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1		
	1	1	1	0	1	0	1	1	1	1	1	1	1	0	0	0		
	G8	P8	G7	P7	G6	P6	G5	P5	G4	P4	G3	P3	G2	P2	G1	P1		
	1	0	0	1	0	1	1	0	1	0	1	0	0	1	0	0		
	to be left blank until exercise #3!																	
←	1		1		1		1		1		1		0		0		0	←CIN
	1		0		0		1		1		0		1		0		SUM LOGIC	
	SUM8		SUM7		SUM6		SUM5		SUM4		SUM3		SUM2		SUM1			
	FF		FF		FF		FF		FF		FF		FF		FF			

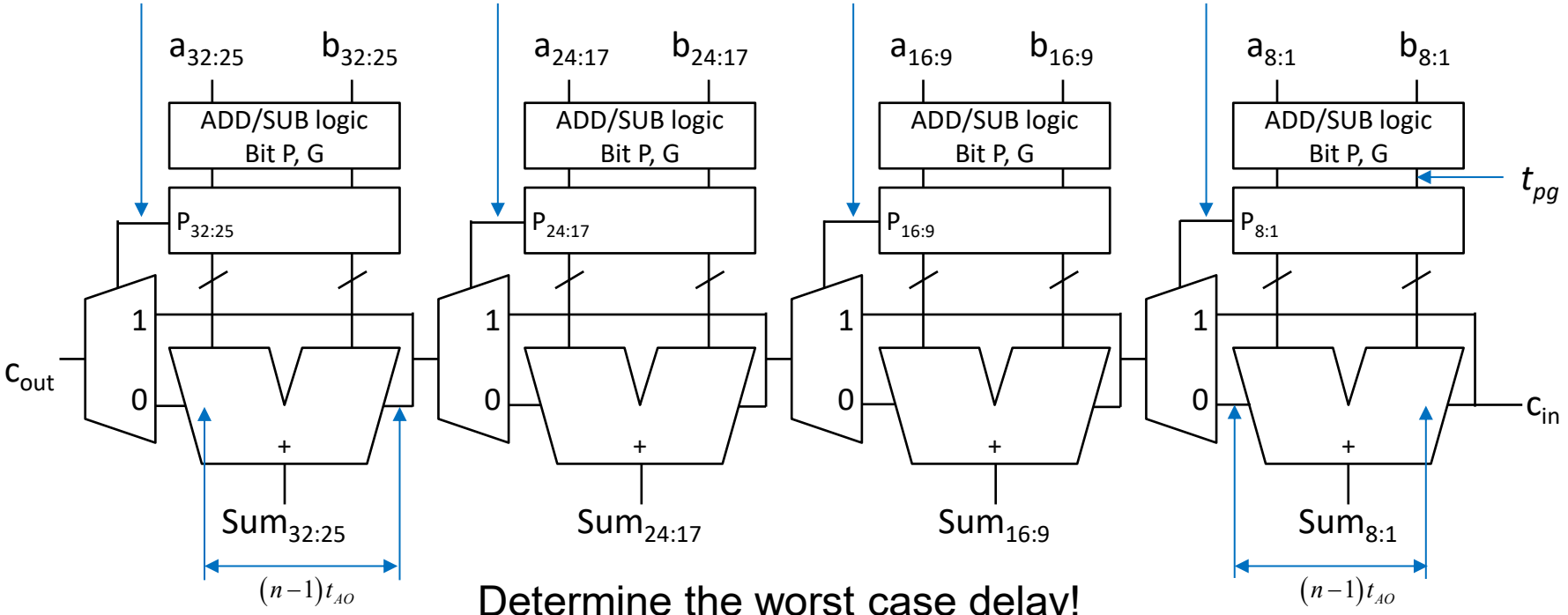
Propagate/generate setup

- We have redesigned the 8-bit adder block with G and P setup logic to get a less complex carry cell! Use AO21 gate!
- After that we added a block-propagate output. (n=8)
- Block propagate is available after another delay of (n-1)t_{AND}.

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

32-bit carry skip adder

- Identify worst-case propagation delay for 32-bit adder!
- For N-bit adder built with *k* *n*-bit blocks!



Determine the worst case delay!

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

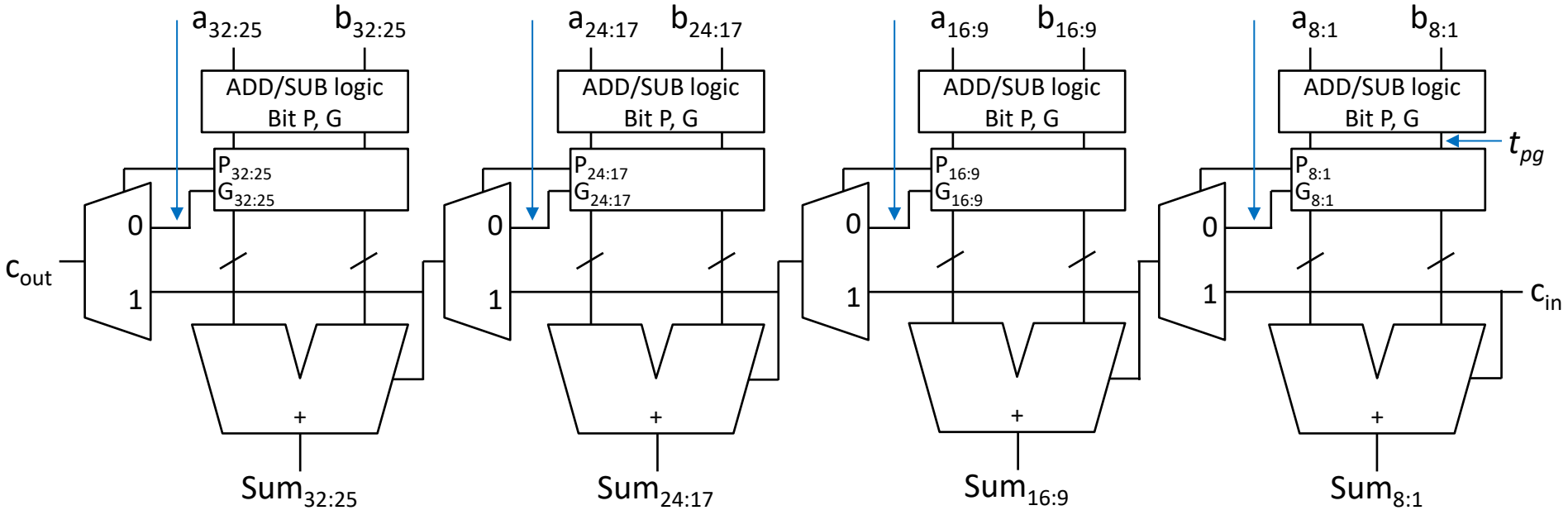
Carry lookahead adders

- In the next step, we added a block generate output!
- Block-G is available $t_{pg(n)}=(n\text{-}1)t_{AO}$ after bit-P and bit-G setup.
- Again, build a 32-bit adder with 4 instances of this 8-bit block

	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF		
	a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1		
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1		
	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	1	0		
	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0		
←	1		1		1		1		1		1		1		1		0	←CIN
	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			

32-bit carry-lookahead adder

DELAY = $t_{pg(n)}$ = (n-1)*t_{AO} where n=8



DELAY through k MUXES = (k-1)*t_{mux} where k=4

Determine the worst case delay!

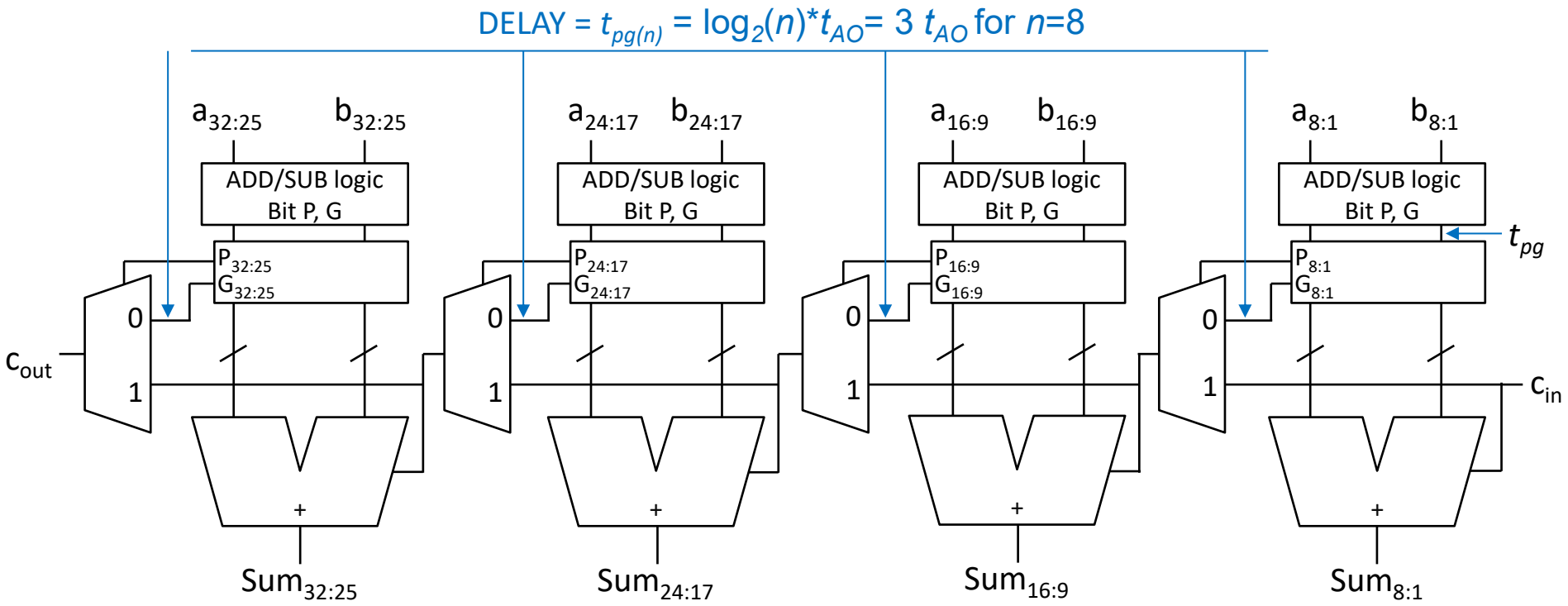
$$t_{cla} = t_{pg} + t_{pg(n)} + (n-1)t_{AO} + (k-1)t_{mux} + t_{XOR}$$

Carry lookahead adders

- Today we are going to reduce the time for obtaining block-P and block-G by obtaining them from a binary tree.
- Then delay will increase as $\log_2(N)$ for an N-bit adder

	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	
	a8	b8	a7	b7	a6	b6	a5	b5	a4	b4	a3	b3	a2	b2	a1	b1	
	1	0	1	0	0	1	1	0	1	0	1	0	1	0	1	1	
	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	1	0	
	0	1			0	1			0	1			1	0			
	0	1							1	0							
←	1	0															
	1		1		1		1		1		1		1		1		0 ←CIN
	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		

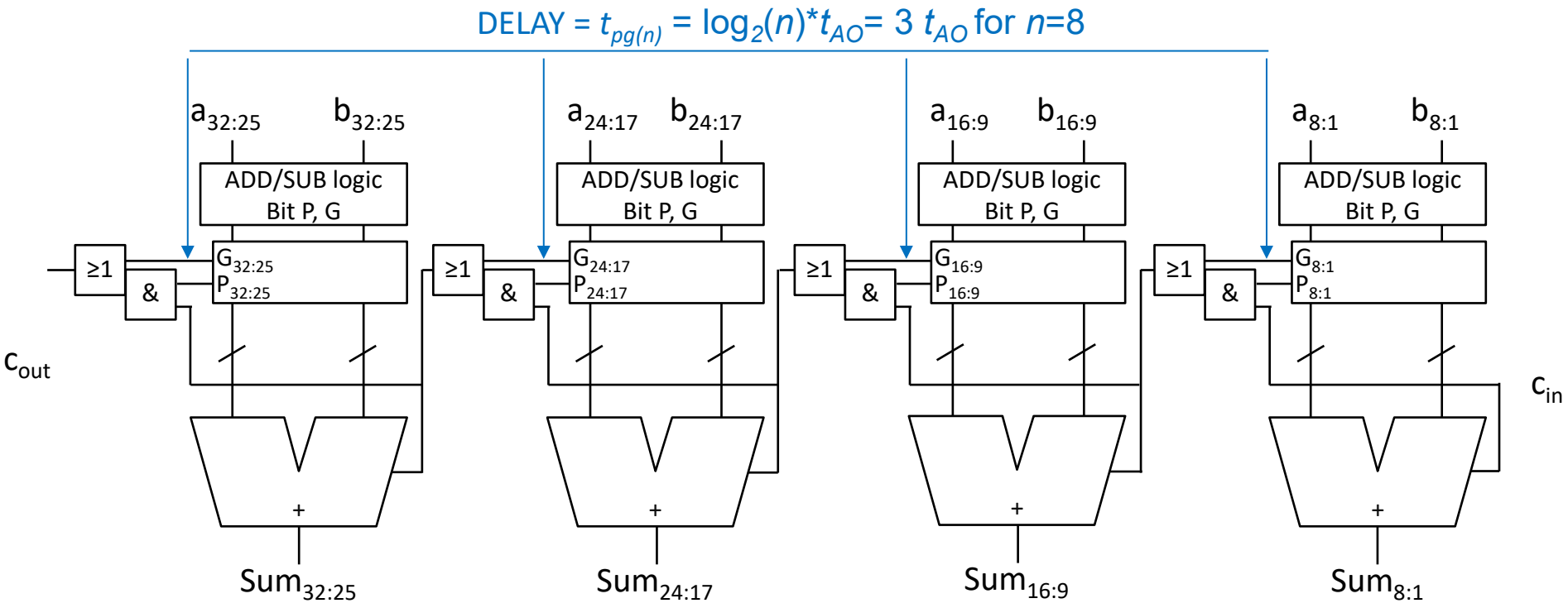
Carry-lookahead adder



Determine the worst case delay!

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

Carry-lookahead adder



Determine the worst case delay!

$$t_{cla} = t_{pg} + t_{pg(n)} + \left[\left(n-1\right)+\left(k-1\right)\right]t_{AO} + t_{XOR}$$

Sklansky adder

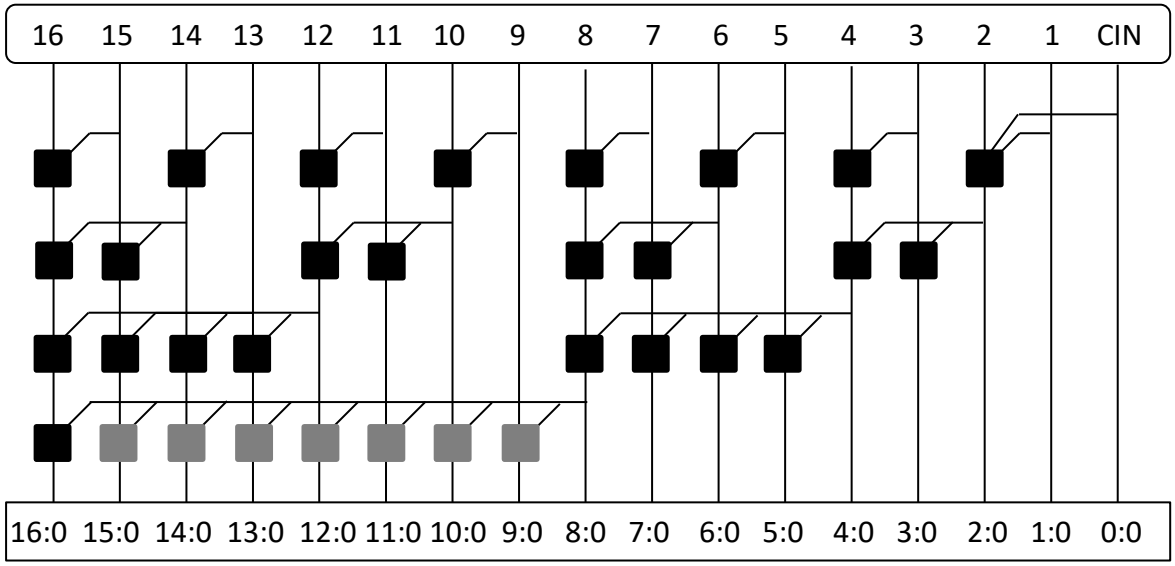


FIGURE 11.29 Tree adder PG networks

Determine the worst case delay!

$$t_{tree} = t_{pg} + t_{pg(n)} + t_{XOR}$$

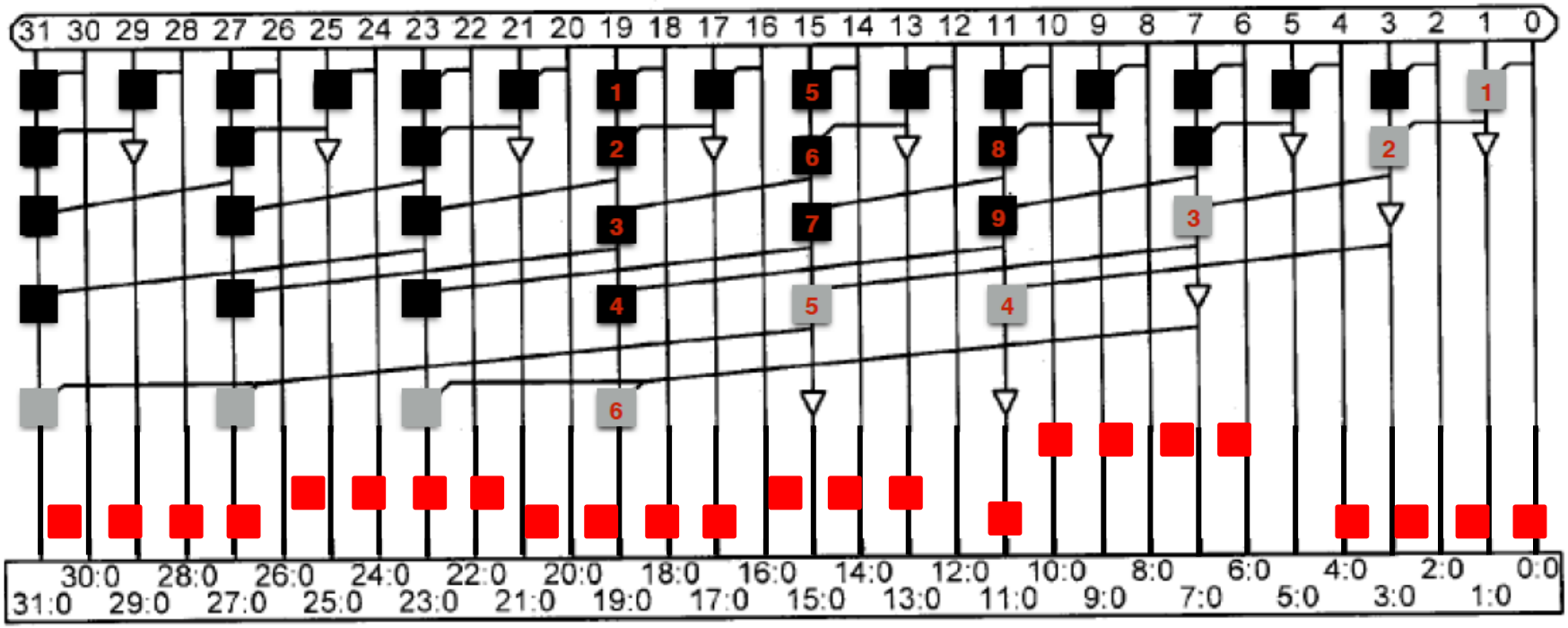
$$t_{pg(n)} = \lceil \log_2 N \rceil t_{AO}$$

Summary

After this interactive lecture you will know about

- Carry-skip adders, carry-lookahead adders and prefix-tree adders like Sklansky adders, etc.
- You can identify the worst-case propagation delay for *N*-bit carry-skip and carry-lookahead adders built from *k* *n*-bit blocks
- You can identify the worst-case delay for prefix-tree adders like Sklansky adders.
- Using the same principles, you should also be able to do so for the unknown prefix-tree adder in home assignment 3.
- Using the same principles, you should also be able to do so in the written exam for some of the other prefix-tree adders like Brent-Kung and Ladner-Fischer.

Home ssignment 3 adder task



Q & A