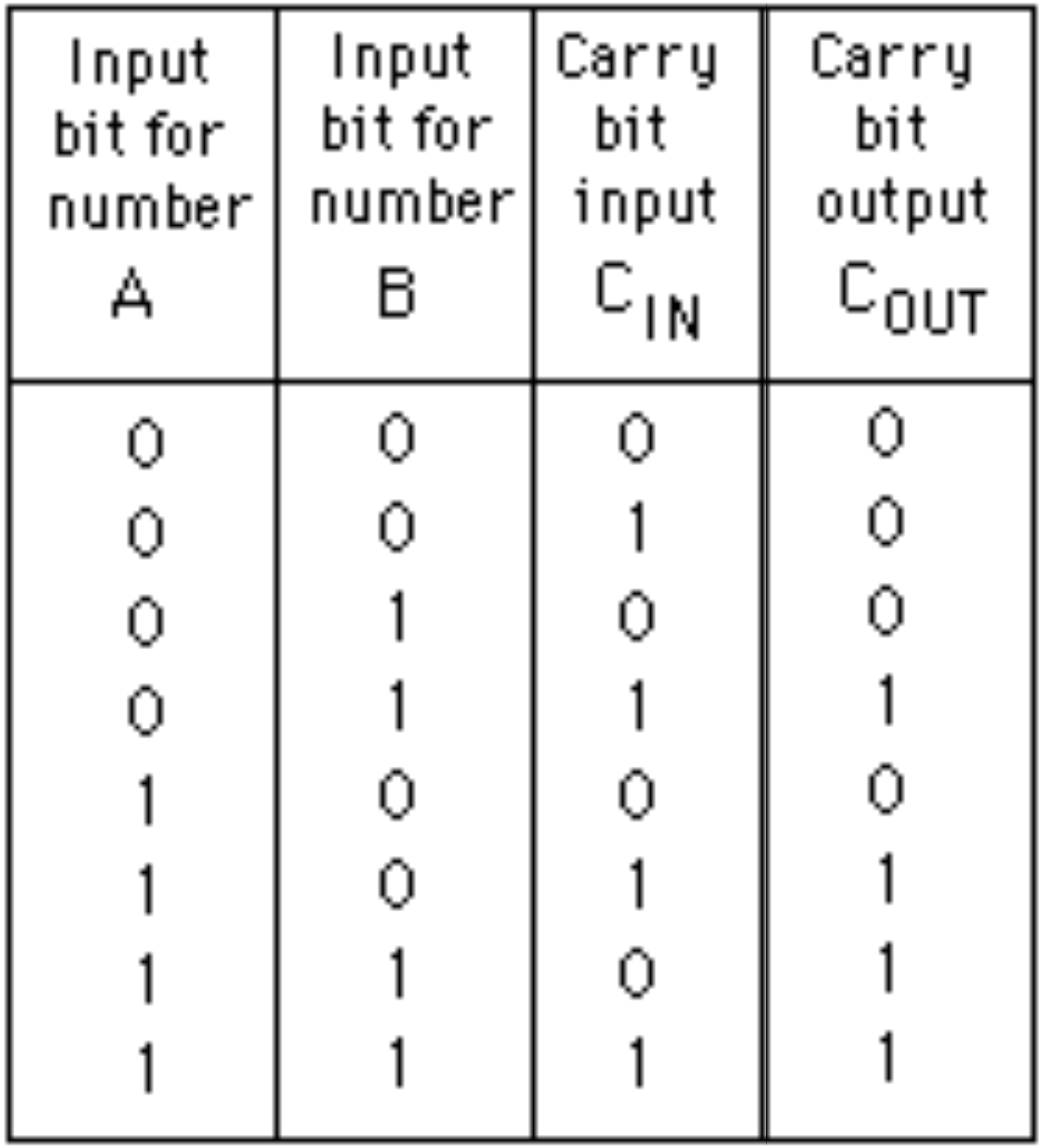
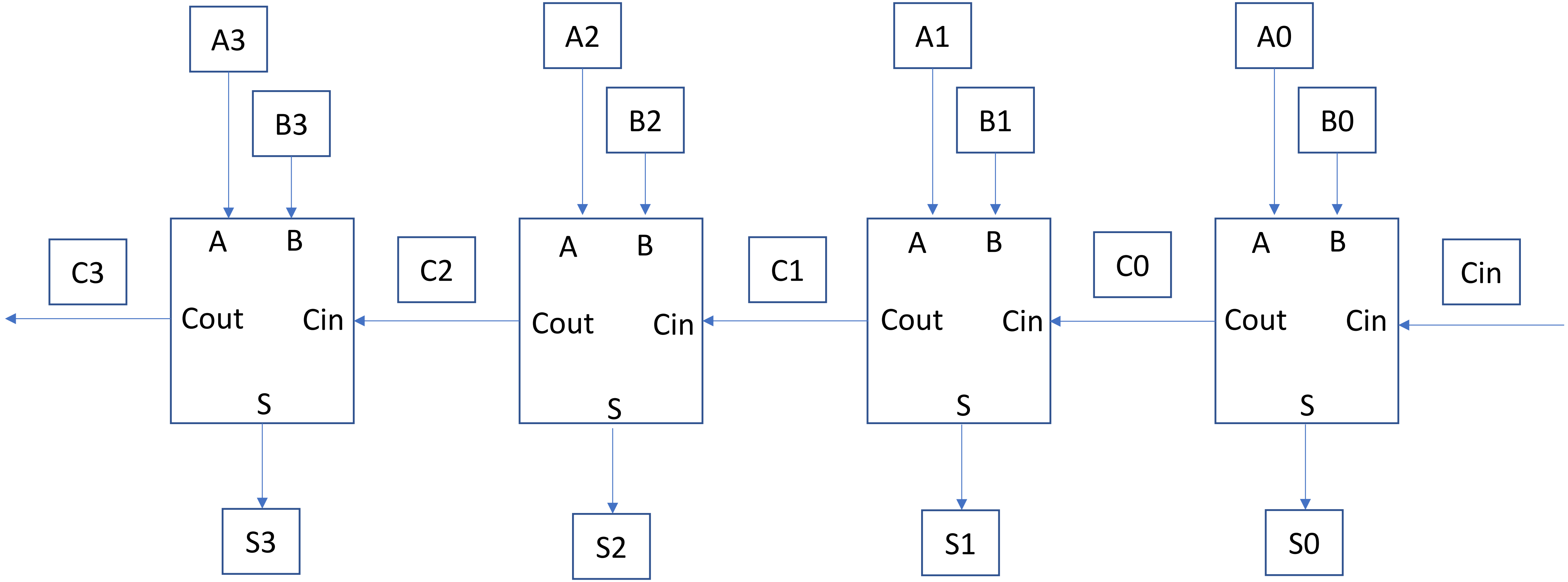
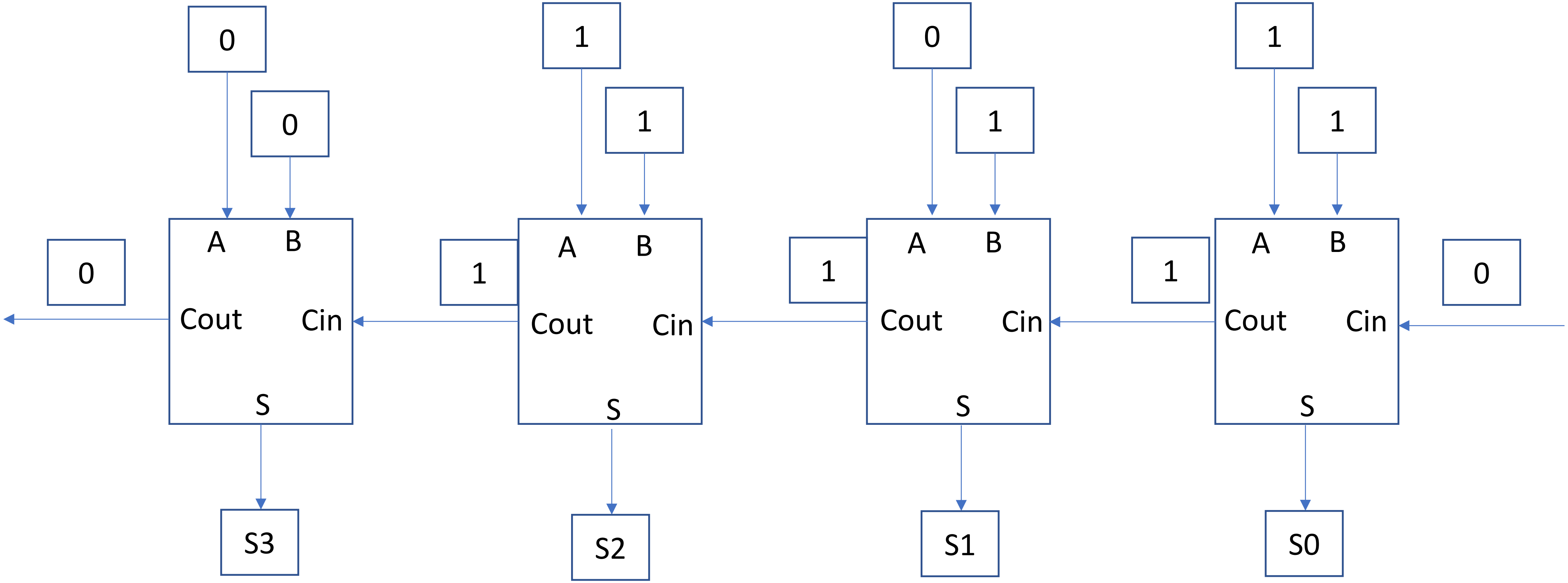
Carry Lookahead Adder

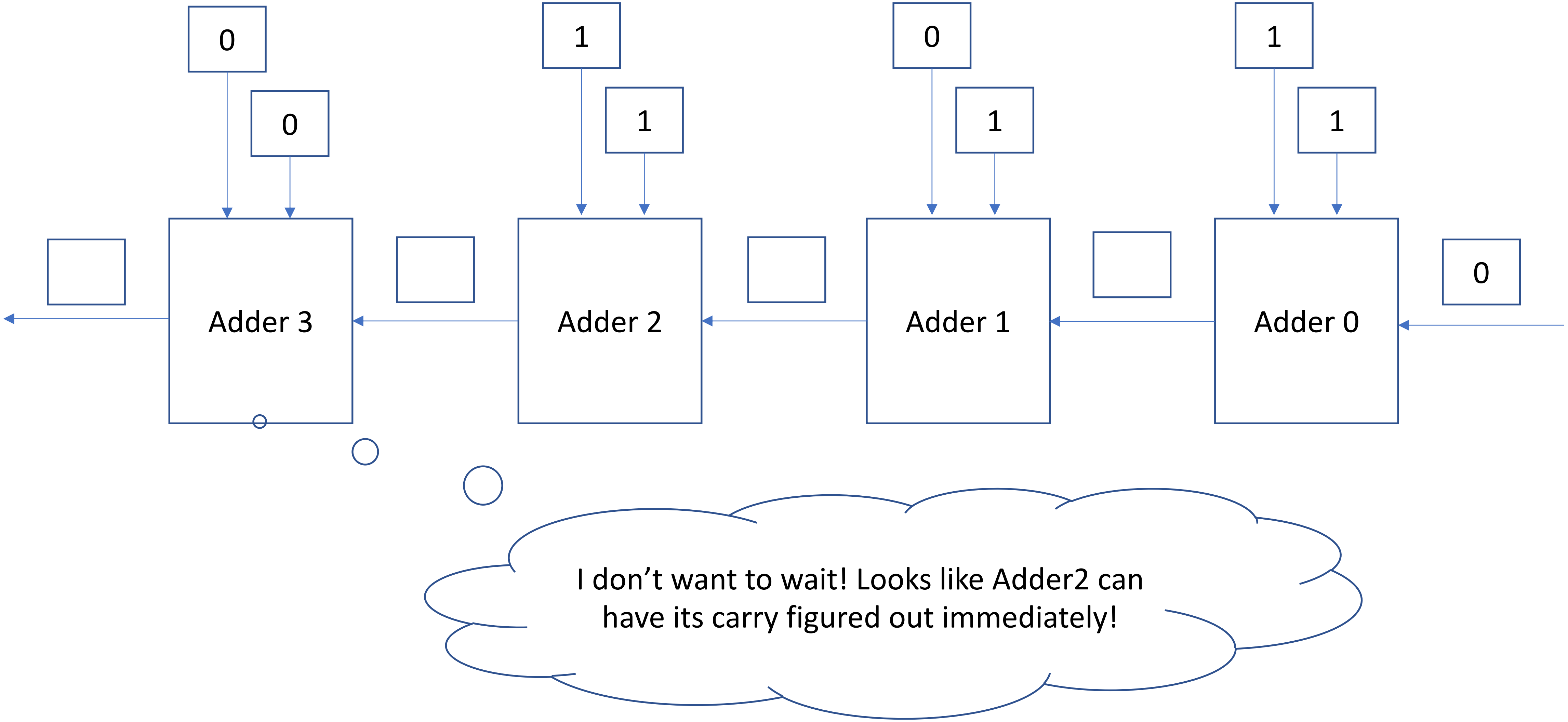


Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Carry bit output C_{OUT}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

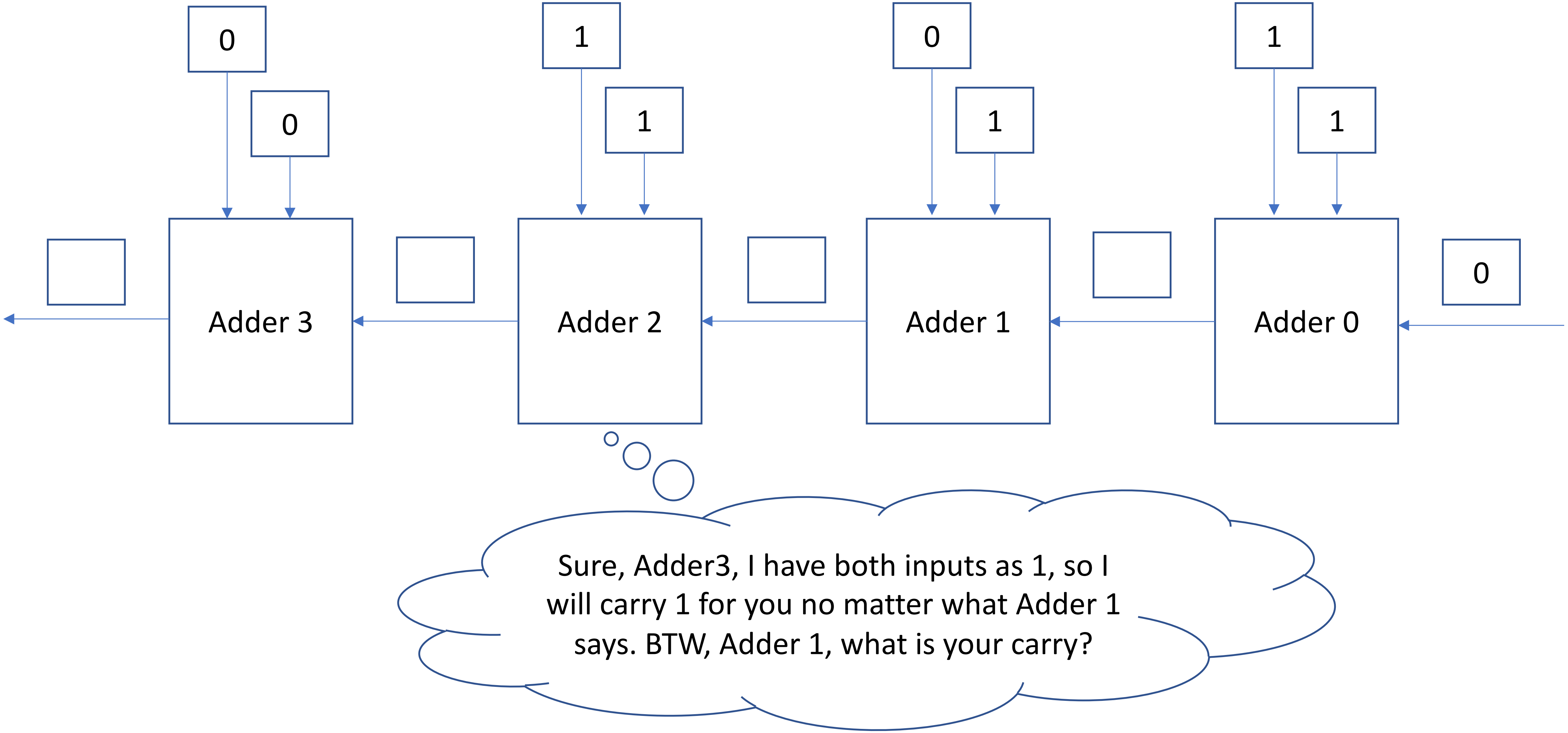

A3
A2
A1
A0
B3
B2
B1
B0
A
B
A
B
A
B
A
B
C3
Cout
Cin
C2
Cout
Cin
C1
Cout
Cin
C0
Cout
Cin
Cin
S
S
S
S
S3
S2
S1
S0

Compute 0101b + 0111b

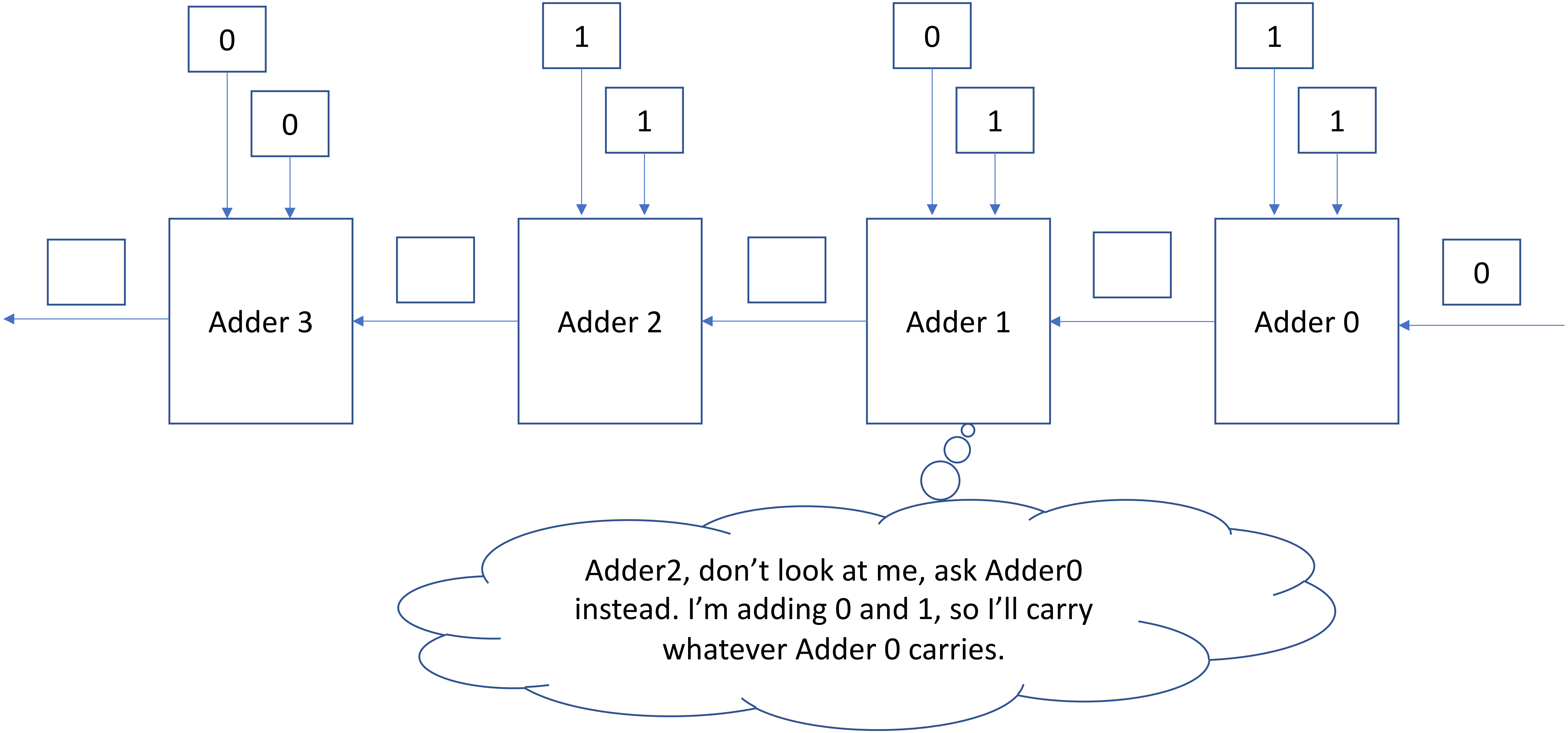




I don’t want to wait! Looks like Adder2 can have its carry figured out immediately!



Sure, Adder3, I have both inputs as 1, so I will carry 1 for you no matter what Adder 1 says. BTW, Adder 1, what is your carry?



Adder2, don’t look at me, ask Adder0 instead. I’m adding 0 and 1, so I’ll carry whatever Adder 0 carries.

What does that imply?

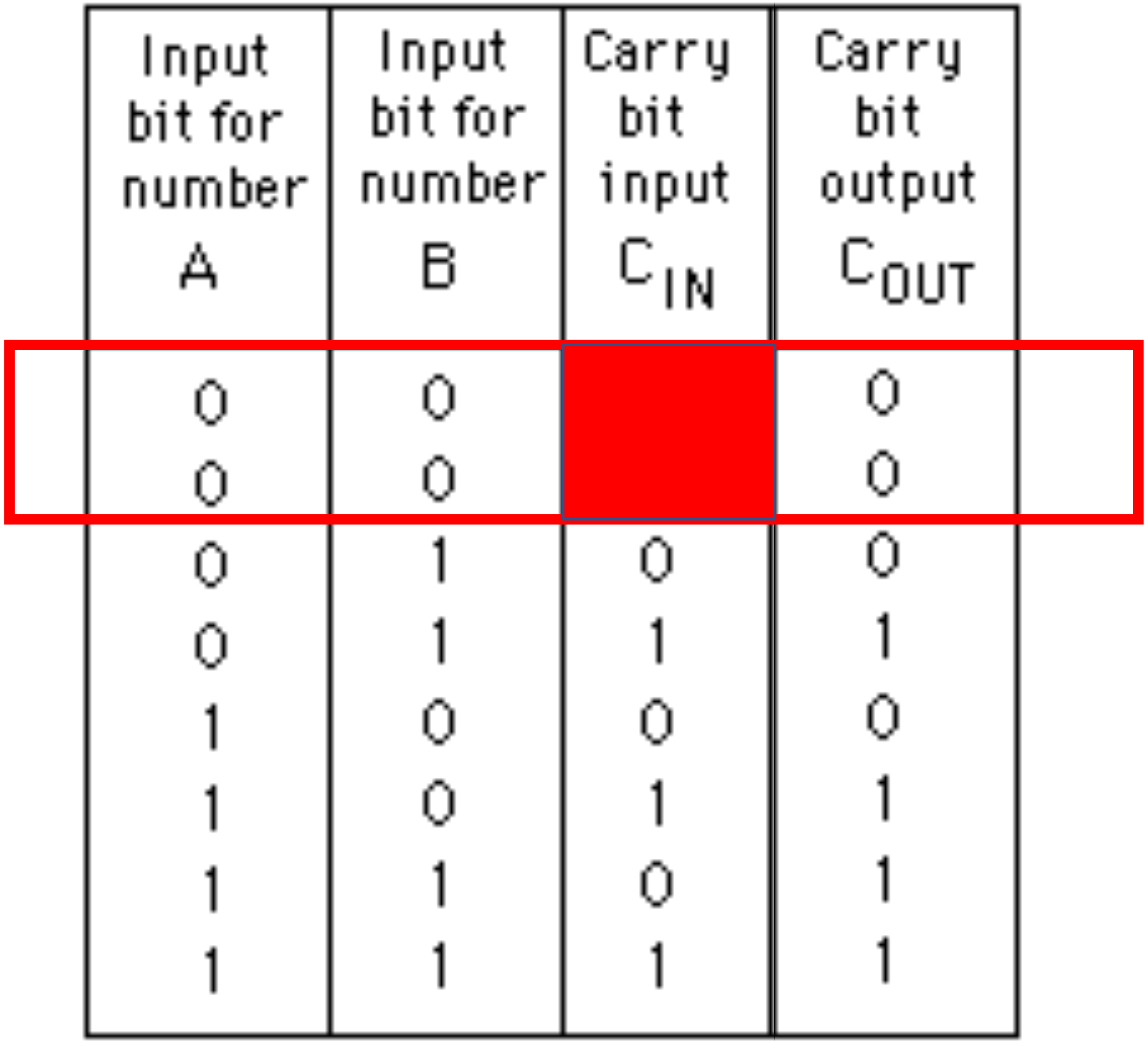
- “Sure, Adder3, I have both inputs as 1, so I will carry 1 for you no matter what Adder 1 says.”

- An adder can generate carry if both its input are 1s. This property cuts the long-chained carry dependency into pieces.
- We call it a “Generate” signal. G in short.
- G = A and B

What does that imply?

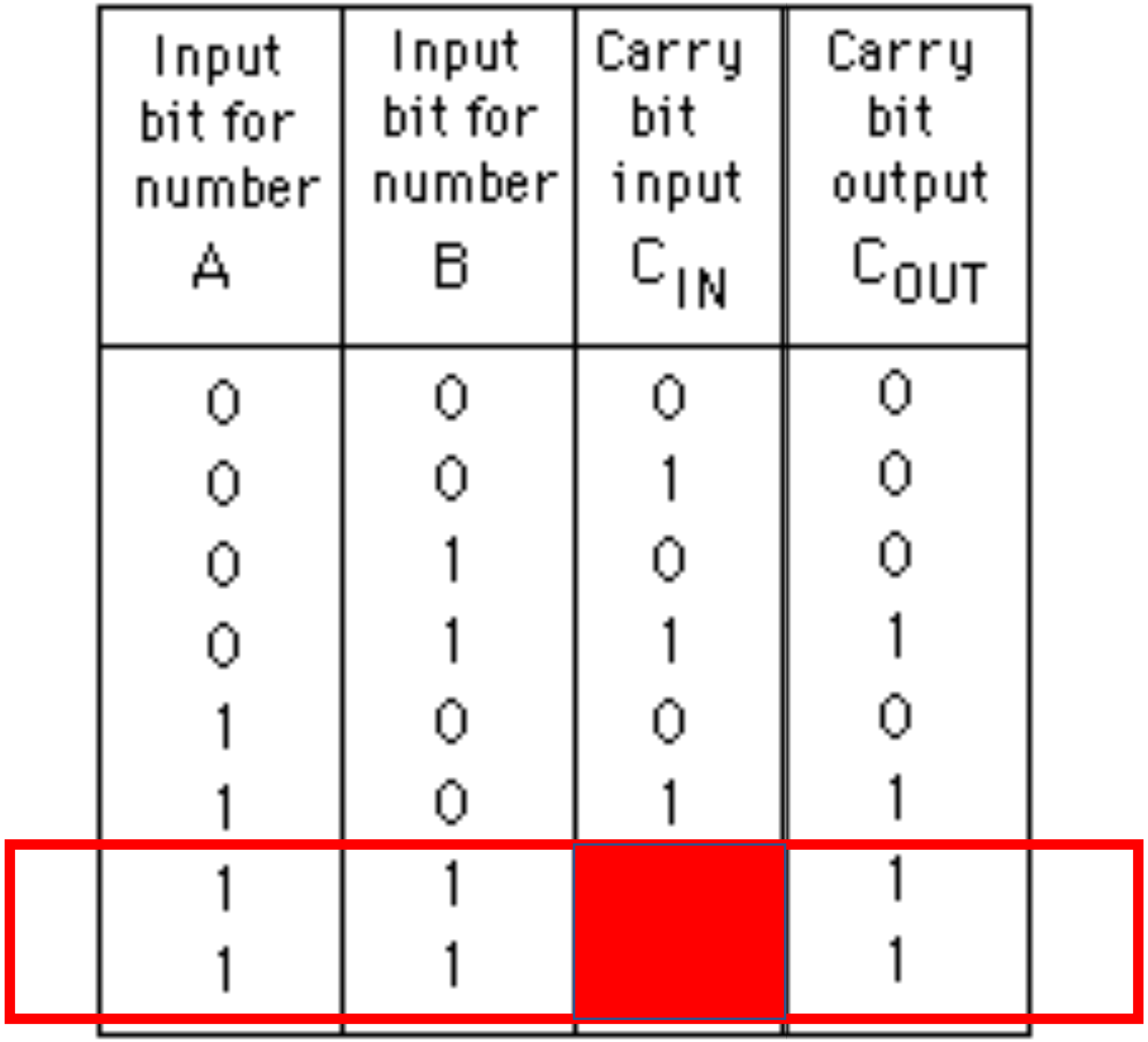
- “Adder2, don’t look at me, ask Adder0 instead. I’m adding 0 and 1, so I’ll carry whatever Adder 0 carries.”

- If an adder is adding a 0 and a 1, its Cout will be exactly same as its Cin. We can directly connect its Cin signal directly to its Cout in this case, so that the adder becomes transparent (eliminated) in the dependency chain.
- We call this a “Propagate” signal. P in short.
- P = A xor B



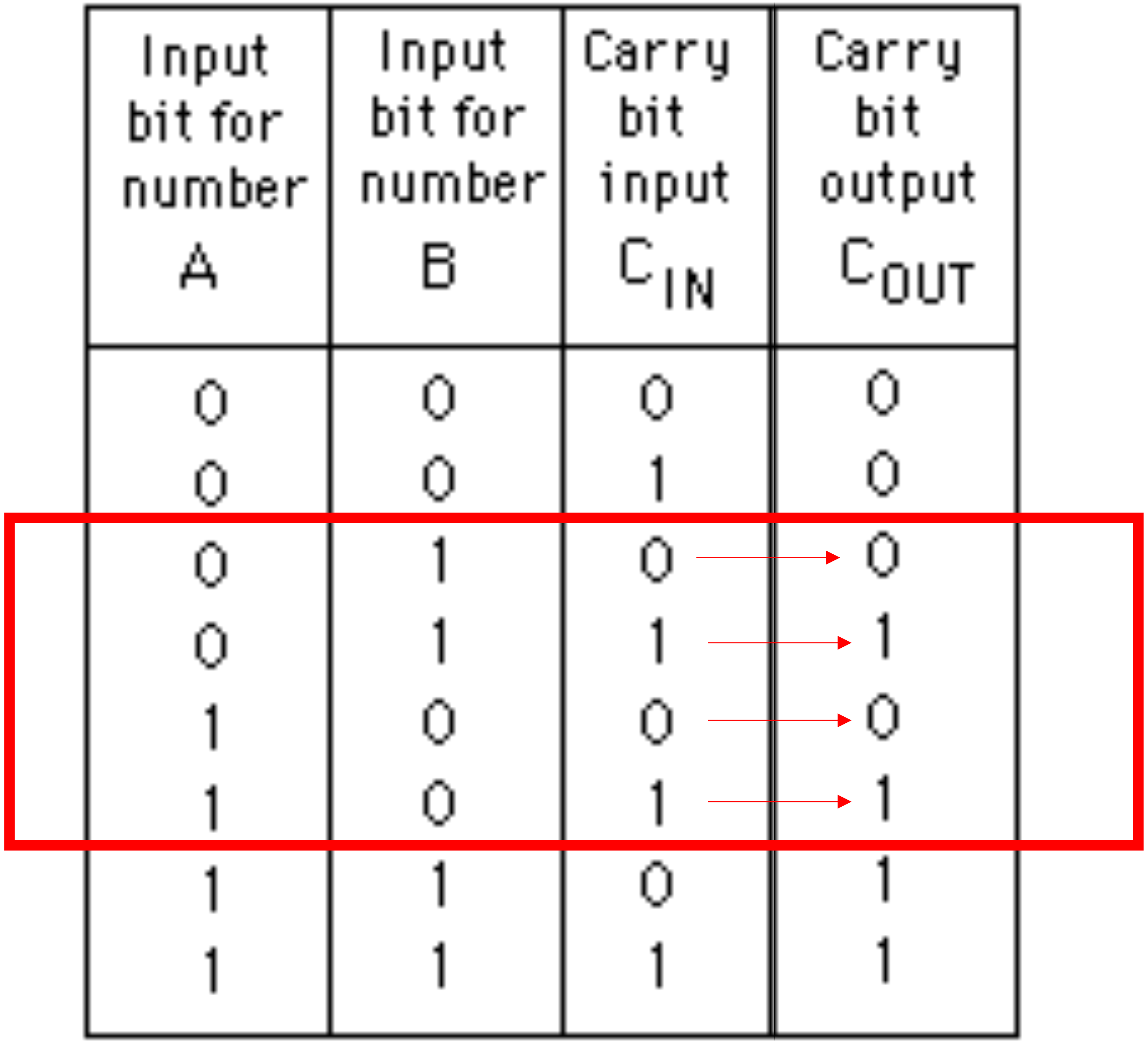
Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Carry bit output C_{OUT}
0	0		0
0	0		0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A=B=0
P = 0
G = 0



Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Carry bit output C_{OUT}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	[redacted]	1
1	1	[redacted]	1

A=B=1
P = 0
G = 1



Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Carry bit output C_{OUT}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A != B
P = 1
G = 0

P & G

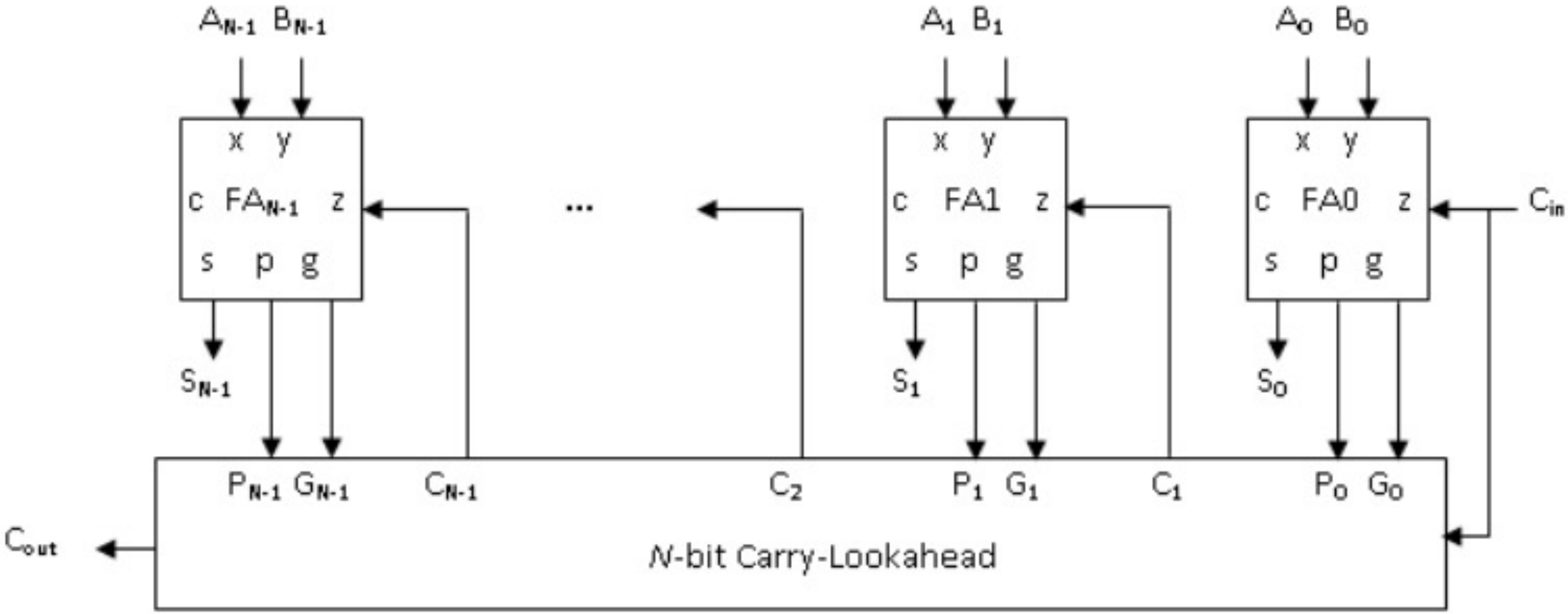
PG	Meaning
00 Generate, cuts propagation	Adder sees **0 + 0 + Cin**, knows **Cout = 0** for sure.
01	Adders sees **1 + 1 + Cin**, knows **Cout = 1** for sure.
10 Propagate, transparent in propagation	Propagate Cin to Cout. G becomes don’t care.
11	-- Unreachable because A & B, A xor B can’t be both 1

Recursive formula for carry

- Consider bit i in a N bit adder.
- Your carry C_i = $G_i + P_i C_{i-1}$
- C_i =$G_i + P_i(G_{i-1} + \ P_{i-1}C_{i-2})$
- If you keep substitute C_k with C_{k-1}, you eventually reach a formula where C_i is some function of $G_{0..i}$ and $P_{0..i}$ and C_{-1}, which is equal to C_{in}
- You are “looking ahead” all the carries, finding your way through all units that are “propagating”, until you find someone that’s “generating” or you reach original C_{in}
- All adders no longer have to wait for signal propagation delay because all P & Gs are computed in parallel and made available to all adders by a Lookahead logic circuit.

Example

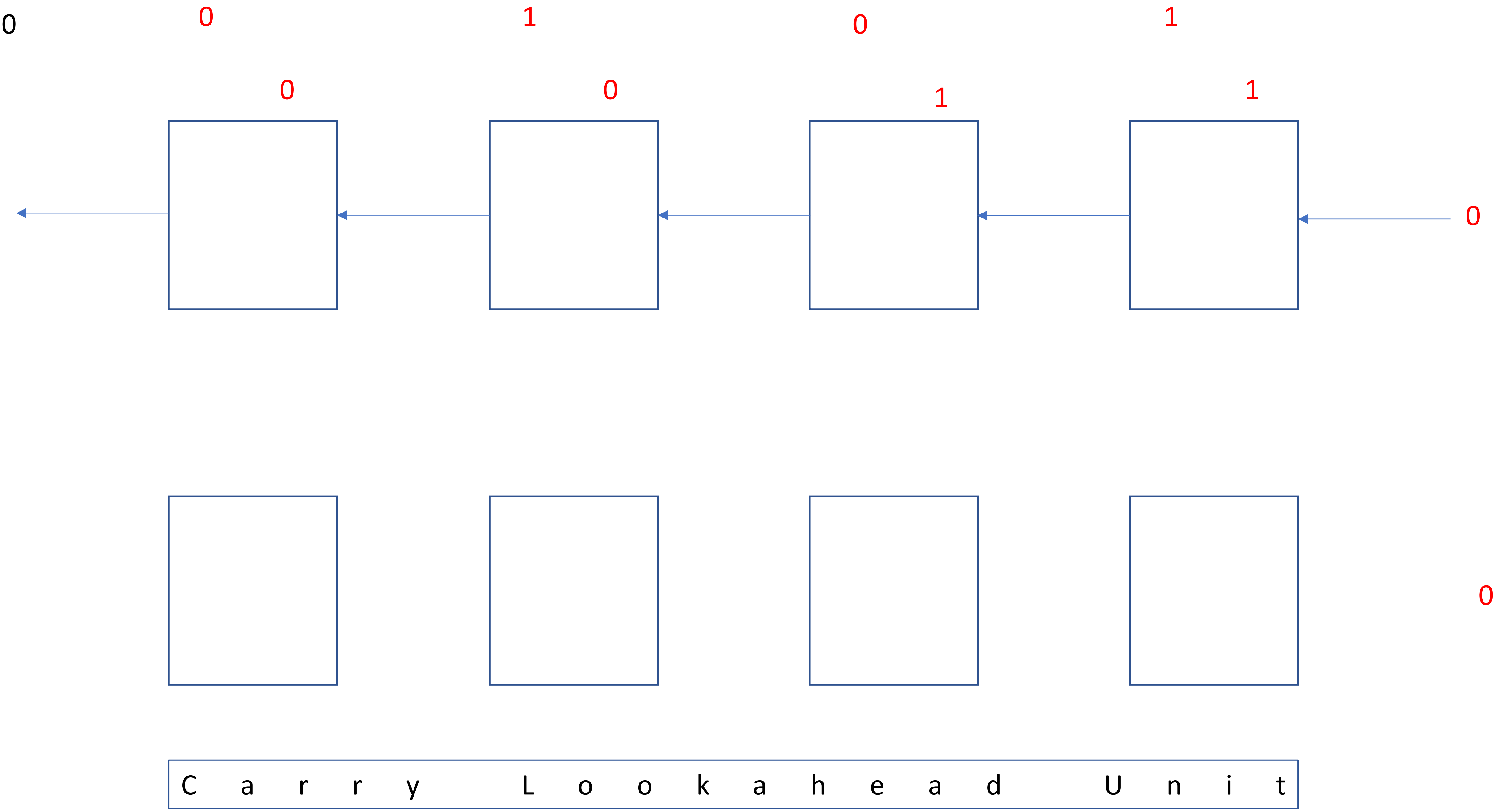
C_3 = $G_3 + P_3(G_2{+}P_2(G_1 + P_1(G_0 + P_0C_{in})))$



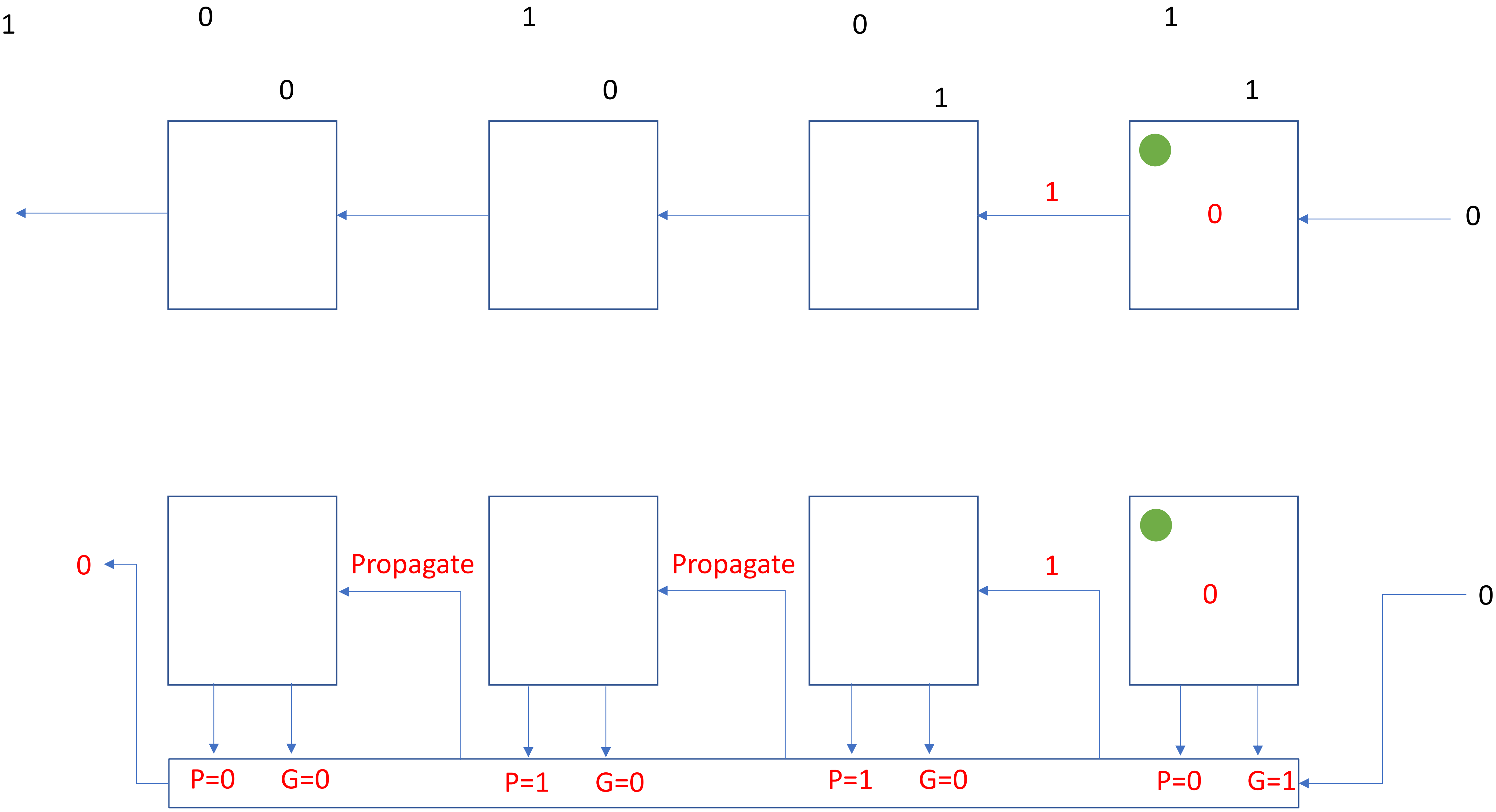
A_{N-1} B_{N-1}
x y
c FA_{N-1} z
s p g
S_{N-1}
...
A_1 B_1
x y
c FA1 z
s p g
S_1
A_0 B_0
x y
c FA0 z
s p g
S_0
C_{in}
P_{N-1} G_{N-1} C_{N-1} C_2 P_1 G_1 C_1 P_0 G_0
C_{out}
N-bit Carry-Lookahead

- Assume all combinational logic (Propagate, Generate, Sum and Carry) take same amount of time dt (for simplicity, not in reality!)
- Every dt, new produced values are marked red, done adders are marked with green circle

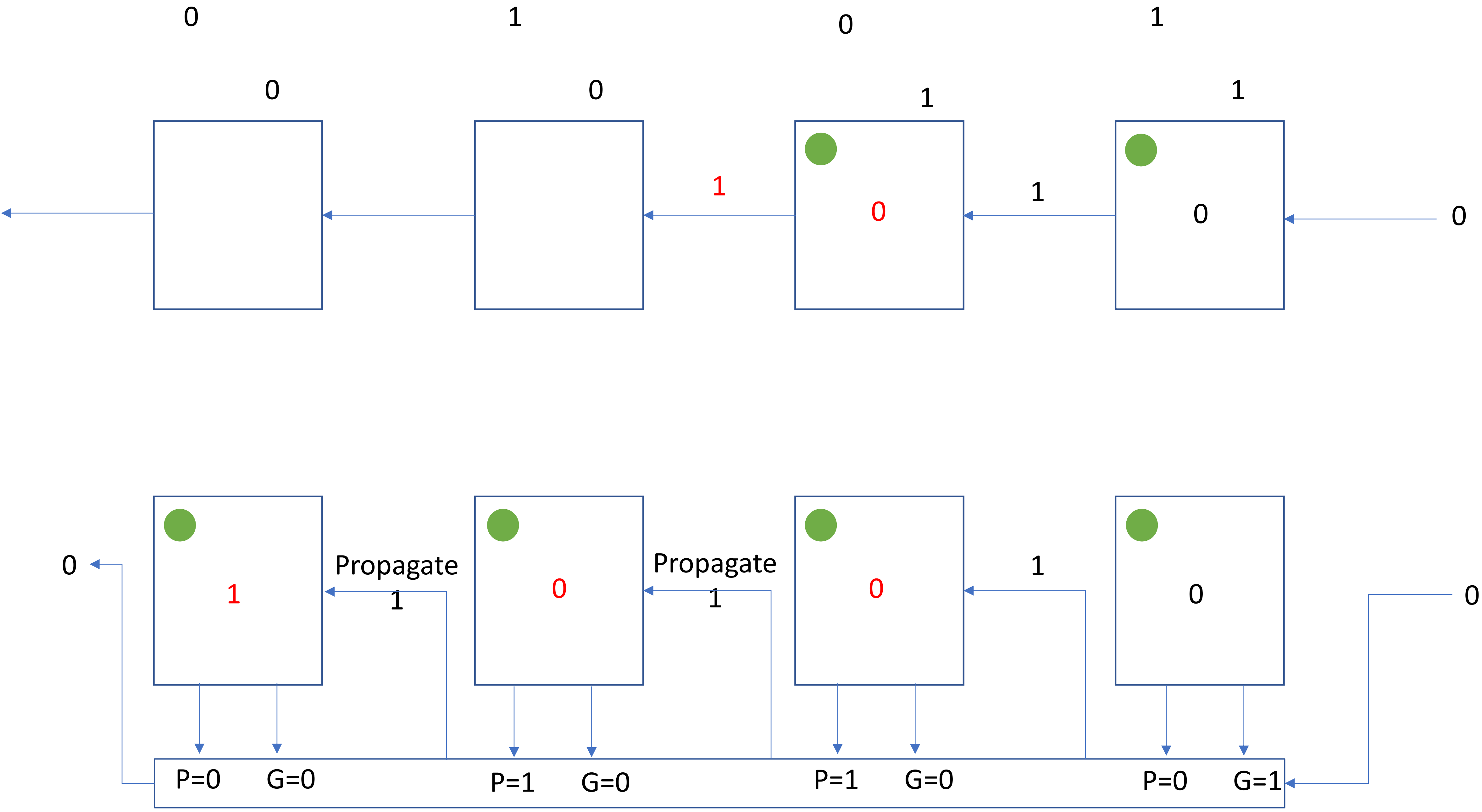
T=0



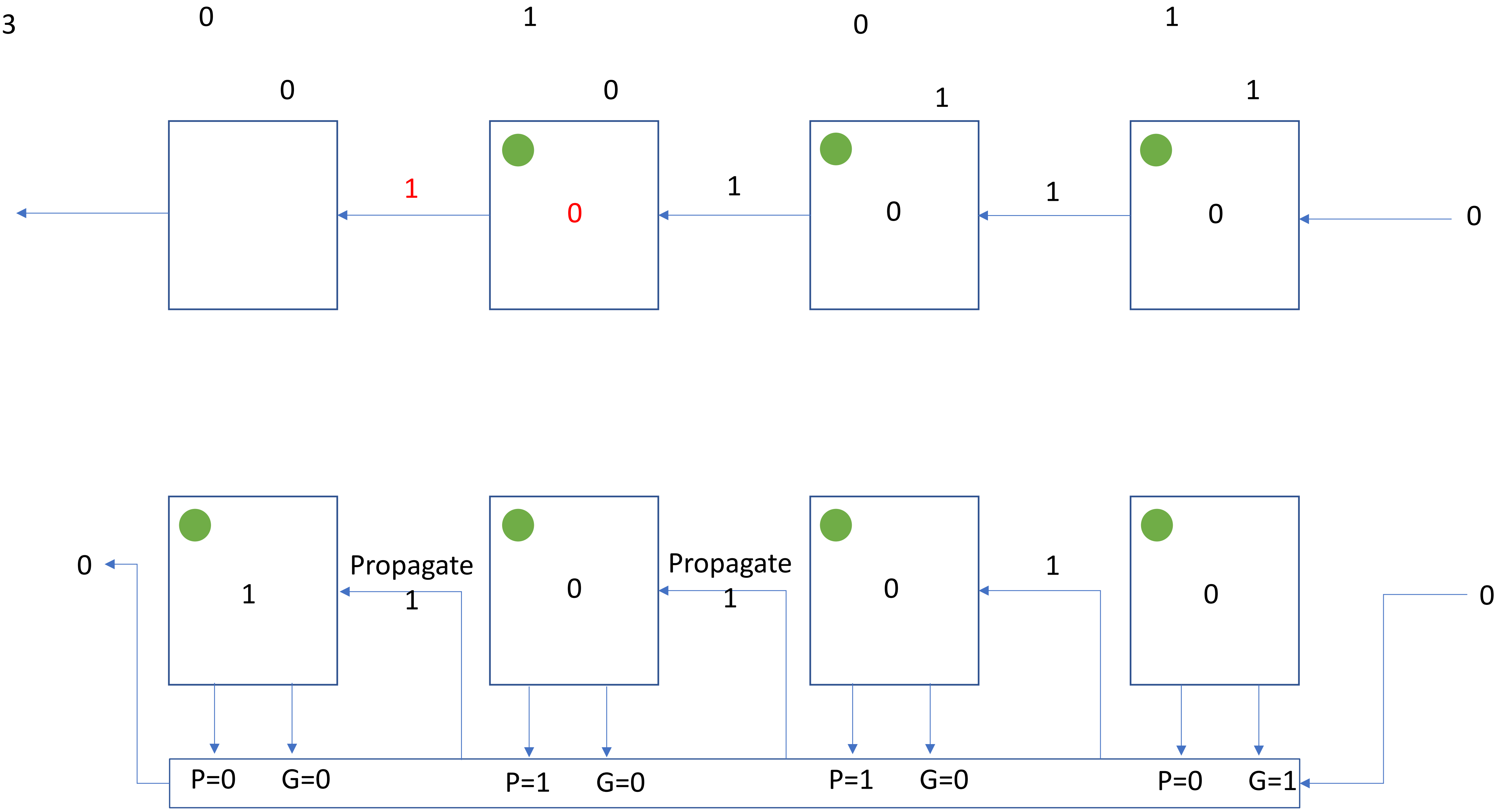
T=1

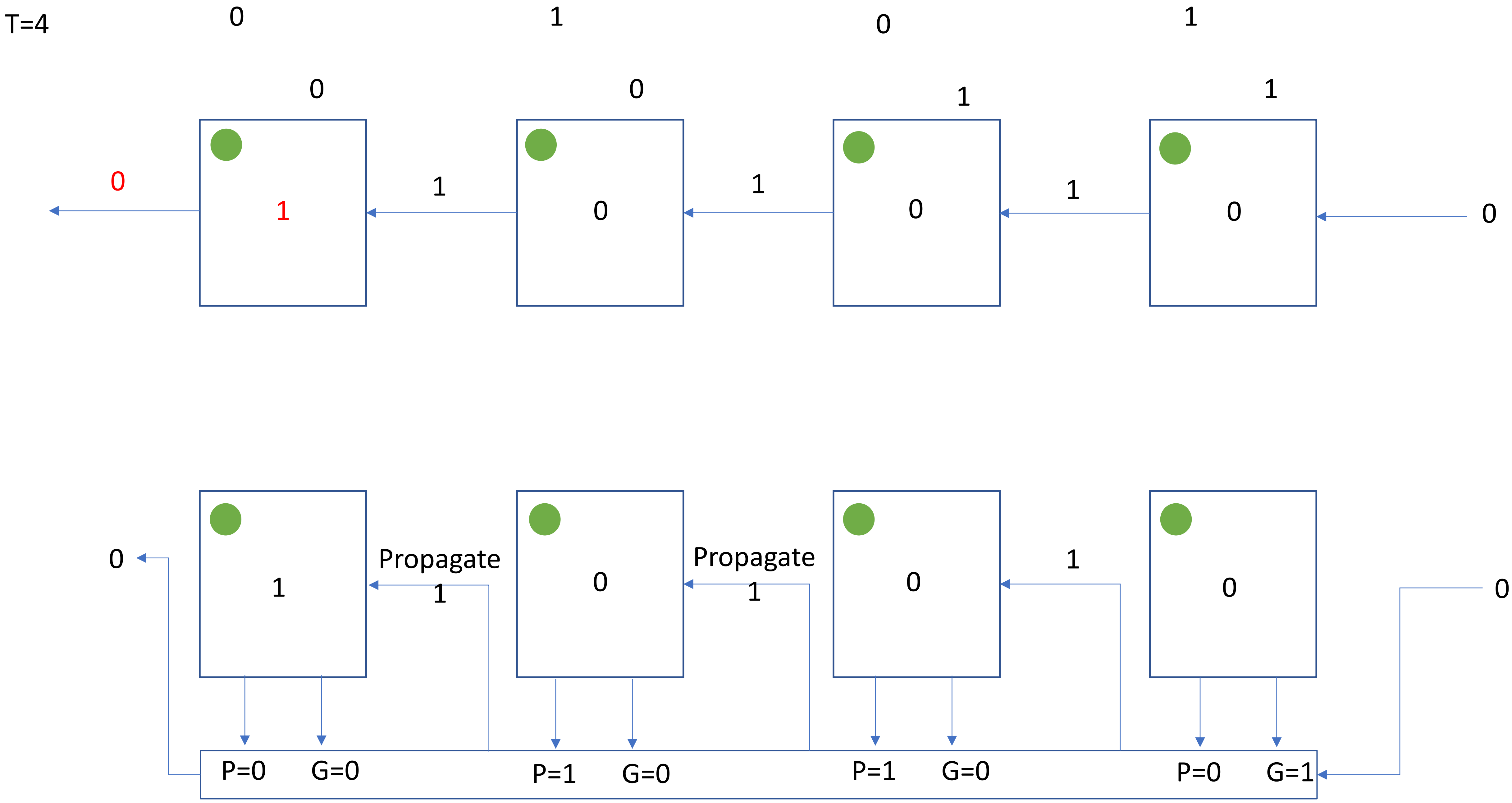


T=2

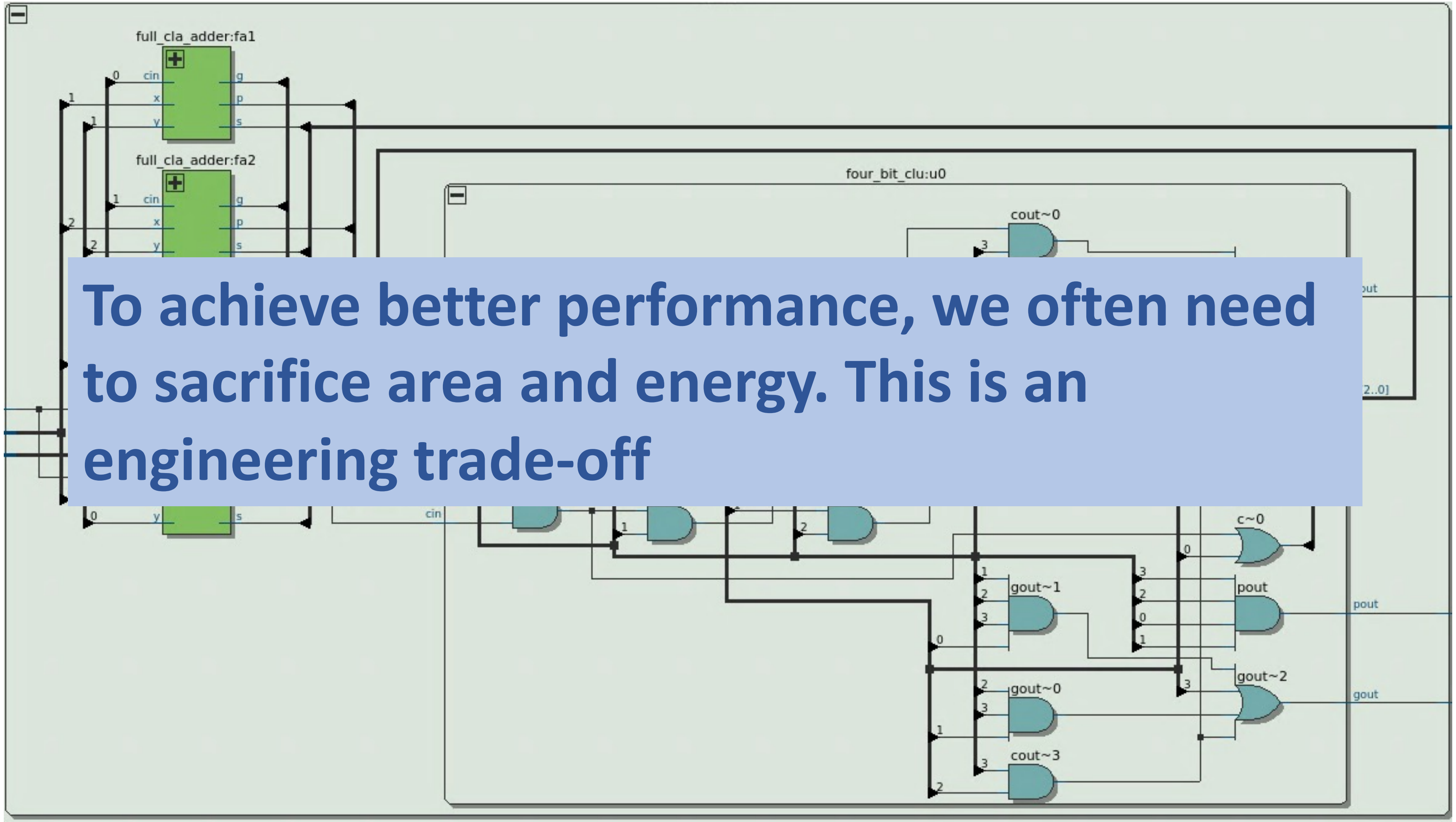

0
1
0
1
0
0
1
1
1
0
1
0
0
0
Propagate
1
1
0
Propagate
1
0
1
0
0
P=0
G=0
P=1
G=0
P=1
G=0
P=0
G=1

T=3

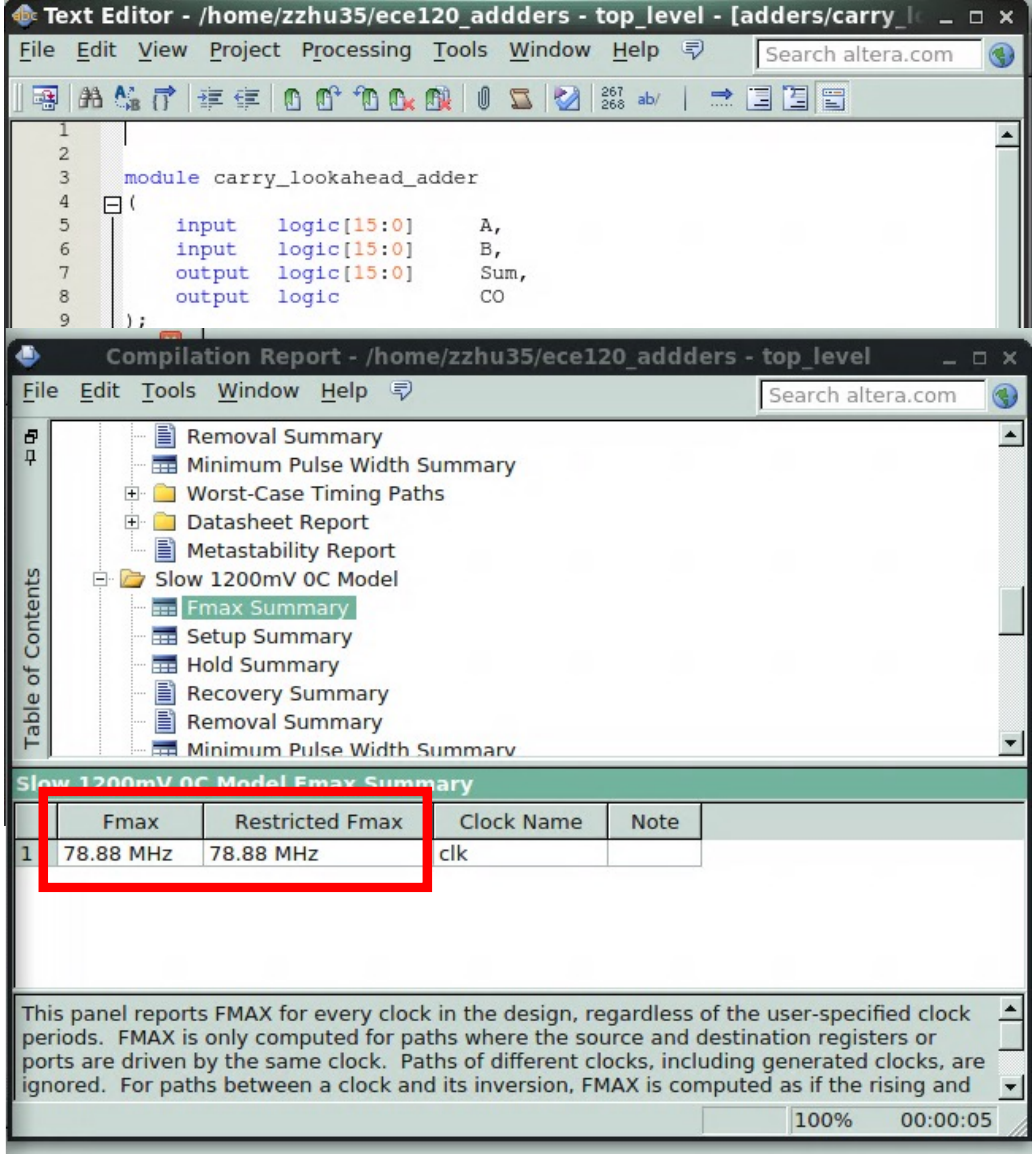

0
1
0
1
0
0
1
1
1
0
1
0
1
0
0
0
0
1
Propagate
1
0
Propagate
1
0
1
0
0
P=0
G=0
P=1
G=0
P=1
G=0
P=0
G=1


T=4
0
1
0
1
0
0
1
1
0
1
1
0
0
1
1
0
0
0
0
Propagate
1
0
Propagate
1
0
1
0
0
P=0
G=0
P=1
G=0
P=1
G=0
P=0
G=1

4-bit Carry Lookahead Unit at a glance



To achieve better performance, we often need to sacrifice area and energy. This is an engineering trade-off


Text Editor - /home/zzhu35/ece120_addders - top_level - [adders/carry_l
File Edit View Project Processing Tools Window Help
Search altera.com
Compilation Report - /home/zzhu35/ece120_addders - top_level
File Edit Tools Window Help
Search altera.com
Table of Contents
Removal Summary
Minimum Pulse Width Summary
Worst-Case Timing Paths
Datasheet Report
Metastability Report
Slow 1200mV 0C Model
Fmax Summary
Setup Summary
Hold Summary
Recovery Summary
Removal Summary
Minimum Pulse Width Summary
100%
00:00:05

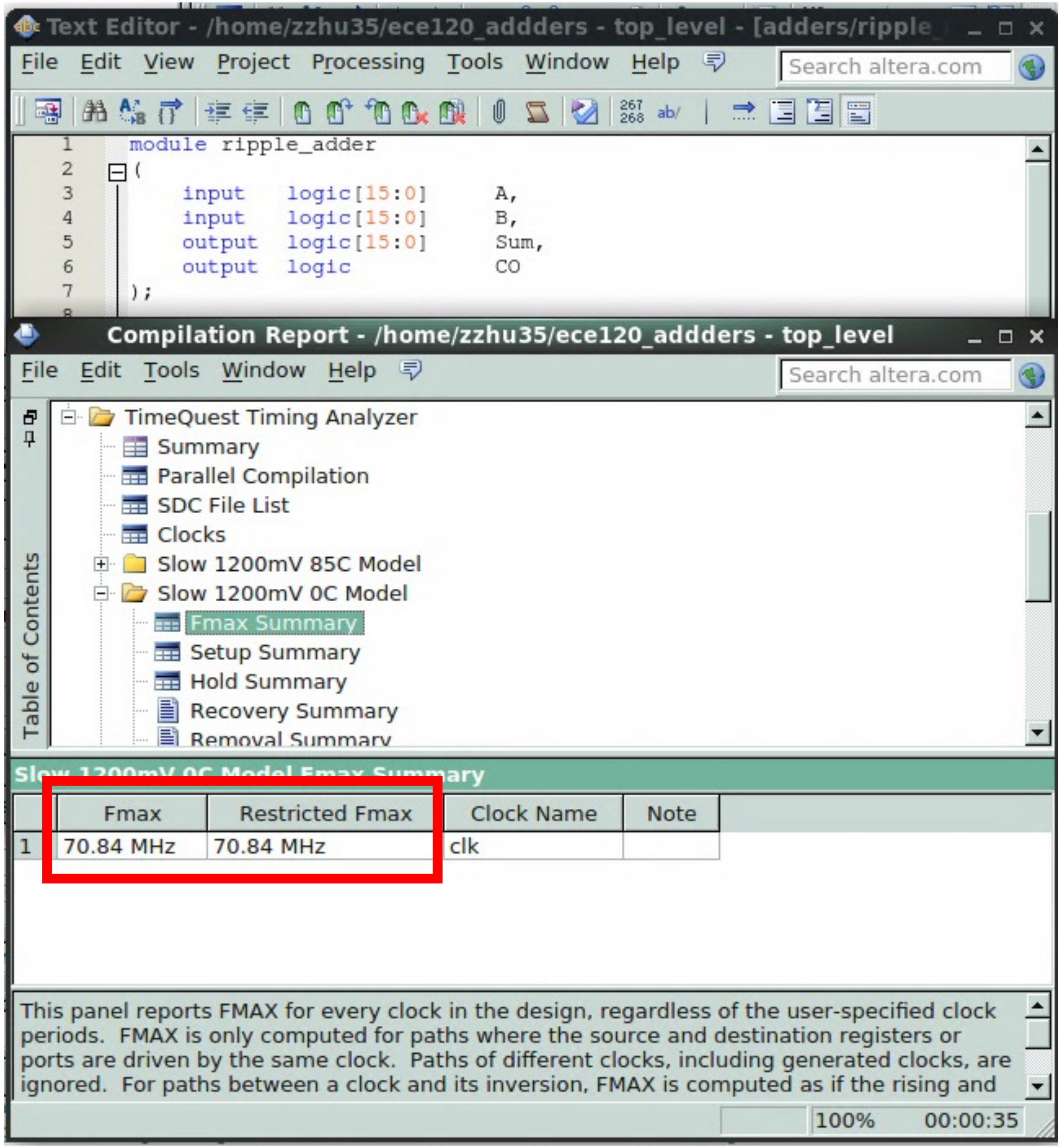
```

module carry_lookahead_adder
(
    input   logic[15:0]     A,
    input   logic[15:0]     B,
    output  logic[15:0]     Sum,
    output  logic           CO
);
```

Slow 1200mV 0C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	78.88 MHz	78.88 MHz	clk	

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and


Text Editor - /home/zzhu35/ece120_addders - top_level - [adders/ripple_
File Edit View Project Processing Tools Window Help
Search altera.com
Compilation Report - /home/zzhu35/ece120_addders - top_level
File Edit Tools Window Help
Search altera.com
Table of Contents
TimeQuest Timing Analyzer
Summary
Parallel Compilation
SDC File List
Clocks
Slow 1200mV 85C Model
Slow 1200mV 0C Model
Fmax Summary
Setup Summary
Hold Summary
Recovery Summary
Removal Summary
100%
00:00:35

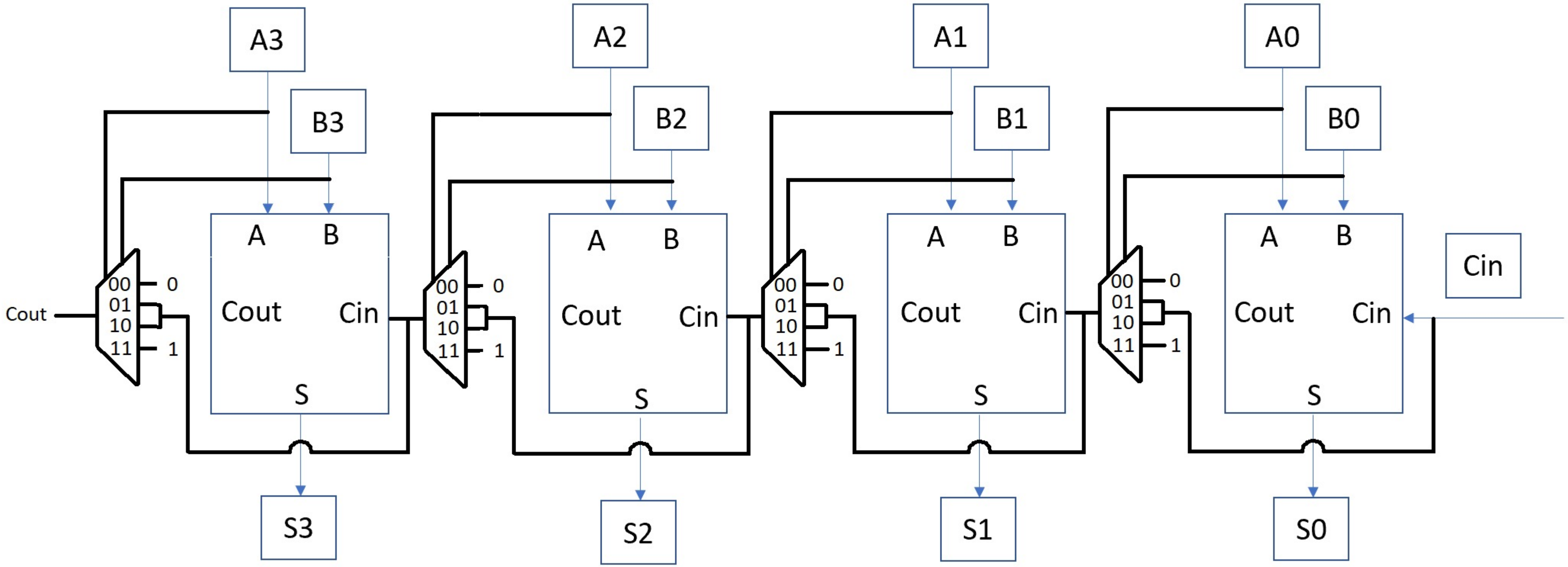
```
module ripple_adder
(
    input   logic[15:0]     A,
    input   logic[15:0]     B,
    output  logic[15:0]     Sum,
    output  logic           CO
);
```

Slow 1200mV 0C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	70.84 MHz	70.84 MHz	clk	

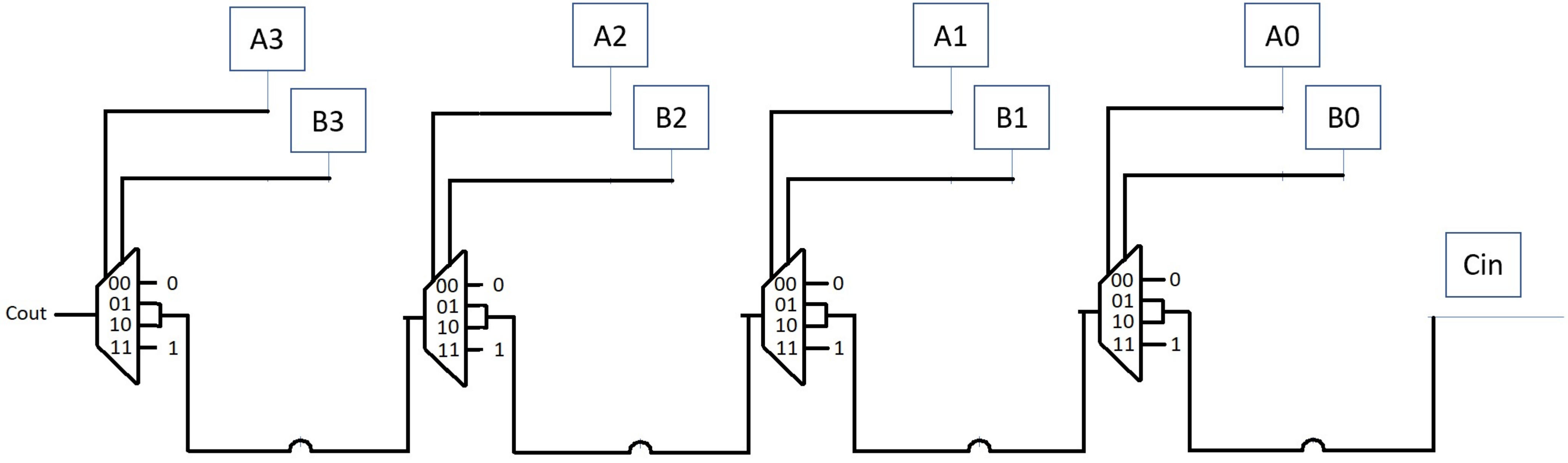
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and

Implementation of CLA



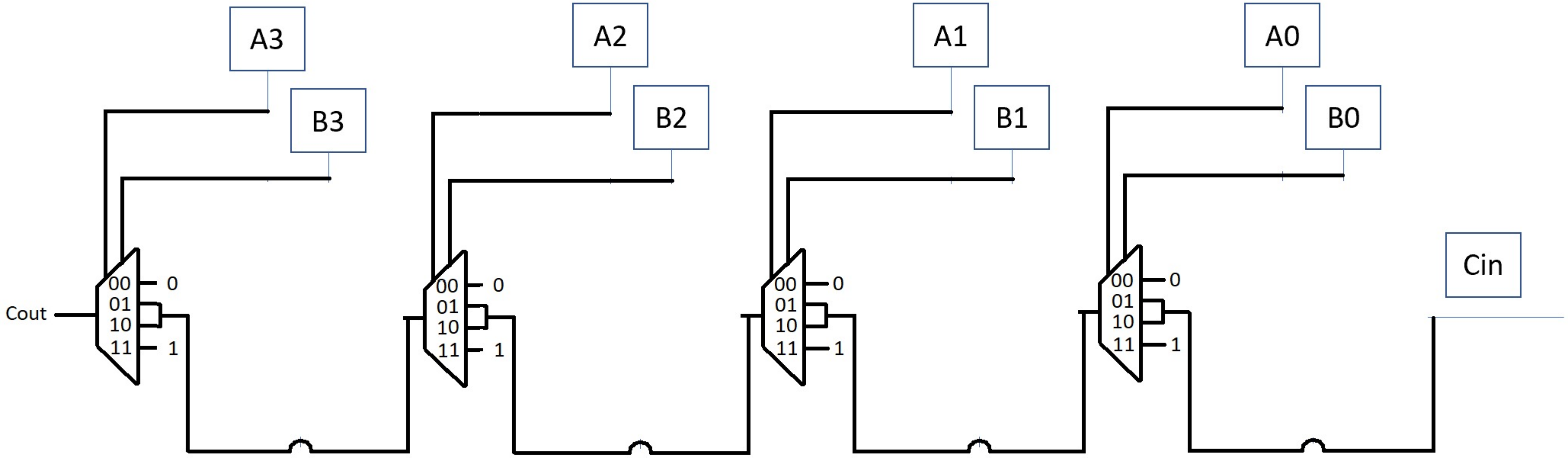
Assuming Mux does not have any delays

Implementation of CLA



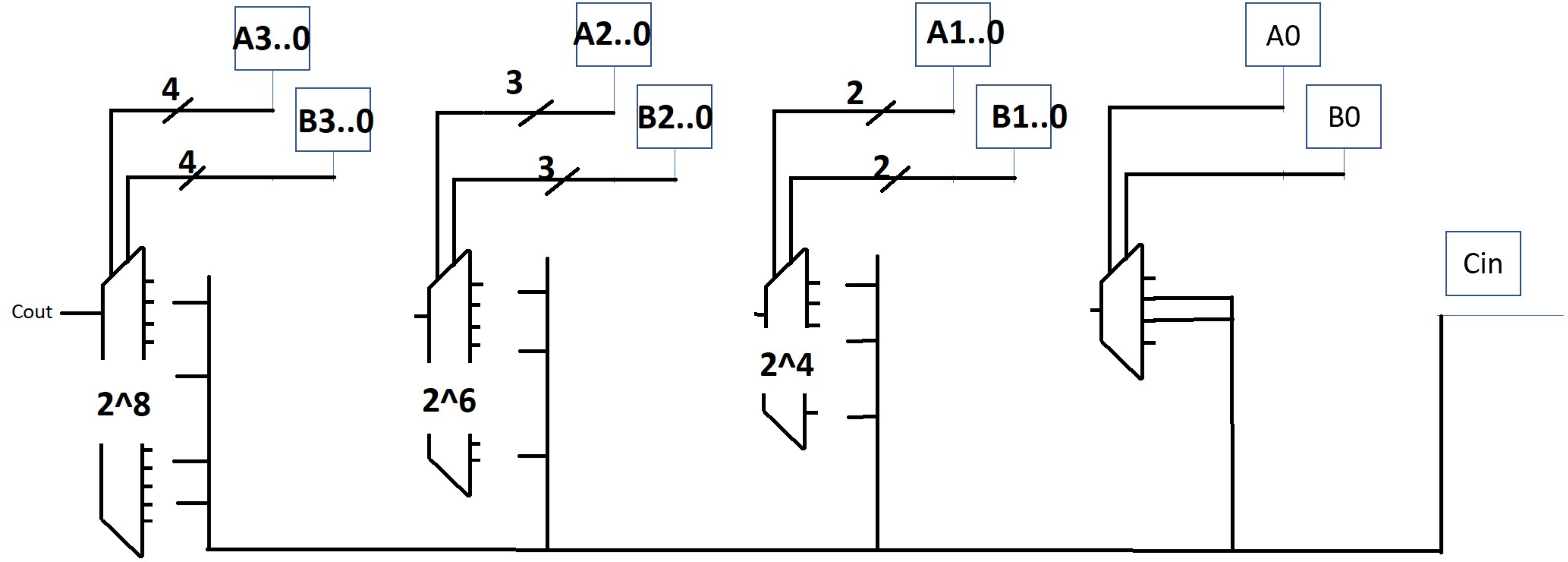
Essentially a Fan-in series of 4 4-input Mux

What if Mux has delay?



This implementation still causes propagation trouble

This circuit behaves exactly same as the “Carrylookahead Unit”. However, no hardware engineer will code 2^8 input mux. In fact, a lot of inputs in those big muxes are duplicates and/or don’t-cares. We can simplify the carry-MUX function into much more concise circuits with P&G signals using the recursive formula.


A3..0
B3..0
4
4
Cout
2^8
A2..0
B2..0
3
3
2^6
A1..0
B1..0
2
2
2^4
A0
B0
Cin