Carry Chain Adder (1A)

-
-

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

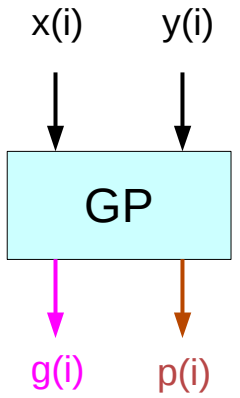
GP Cell

x(i), y(i) : ($\log_2$B)-bit number

Generate	$G_i = a_i \cdot b_i$
Propagate	$P_i = a_i \oplus b_i$

Generate	g(i) = 1	If x(i) + y(i) > (B – 1)
	0	otherwise

Propagate	p(i) = 1	If x(i) + y(i) = (B – 1)
	0	otherwise



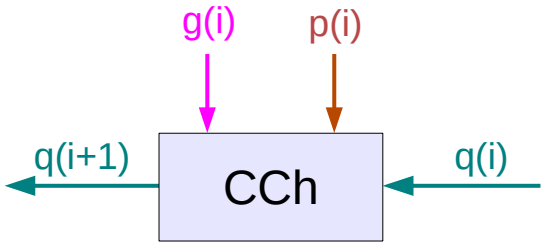
Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Carry Chain Cell (1)

q(i+1), q(i) : 1-bit number

$$c_{out} = G_i + P_i c_i$$

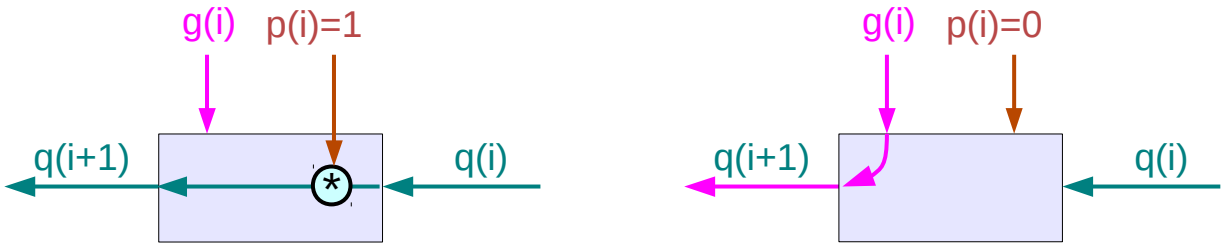
q(i+1)	= q(i)	when p(i) = 1	Propagate
	= g(i)	otherwise	Generate



Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Carry Chain Cell (2)

q(i+1)	= q(i)	when p(i) = 1	Propagate
	= g(i)	otherwise	Generate

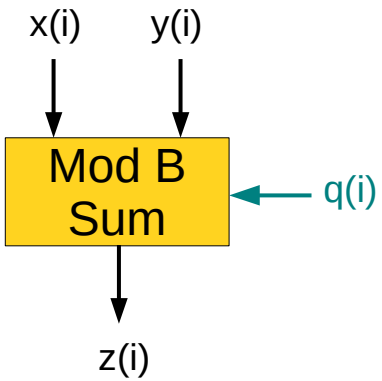


Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

$$c_{out} = G_i + P_i c_i$$

Mod B Sum Cell

$$z(i) = (x(i) + y(i) + q(i)) \bmod B$$



Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

4-ary Carry Chain Addition Example

Generate	g(i) = 1	If x(i) + y(i) > 3
	0	otherwise

Propagate	p(i) = 1	If x(i) + y(i) = 3
	0	otherwise

q(i+1)	= q(i)	when	p(i) = 1	Propagate
	= g(i)	otherwise		Generate

z(i) = (x(i) + y(i) +q(i)) mod 4

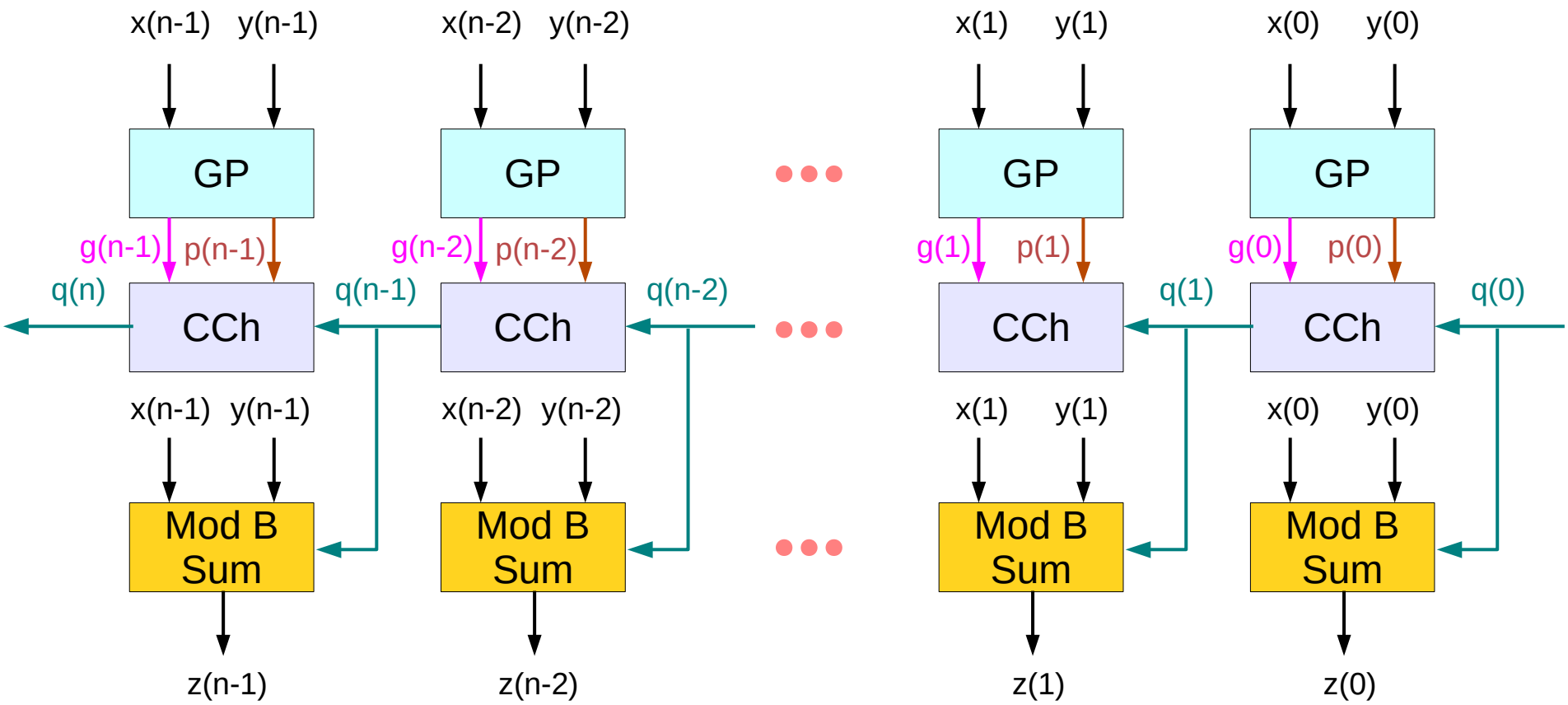
	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

p(i)	0	1	2	3
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

g(i)	0	1	2	3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Carry Chain Adder



Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Carry Chain Addition

```
-- computation of the generation and propagation conditions
for i in 0..n-1 loop
    g(i) := g(x(i), y(i));
    p(i) := p(x(i), y(i));
end loop

-- carry computation
q(0) := c_in;
for i in 0..n-1 loop
  if p(i)=1 then q(i+1):=q(i); else q(i+1):=g(i); end if;
end loop

-- sum computation
for i in 0..n-1 loop
    z(i) := (x(i)+y(i)+q(i)) mod B
end loop;
z(n) := q(n);
```

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Computation Decomposition

the first iteraton includes **2.n B-ary** operations
computation of g(i) and p(i) that could be executed in parallel

```
g(i) := g(x(i), y(i));
p(i) := p(x(i), y(i));
```

The second iteration is made up of **n** iteration steps
that must be used executed sequentially
as q(i+1) is a function of q(i)
consists of **binary** operation only

```
if p(i)=1 then q(i+1):=q(i); else q(i+1):=g(i); end if;
```

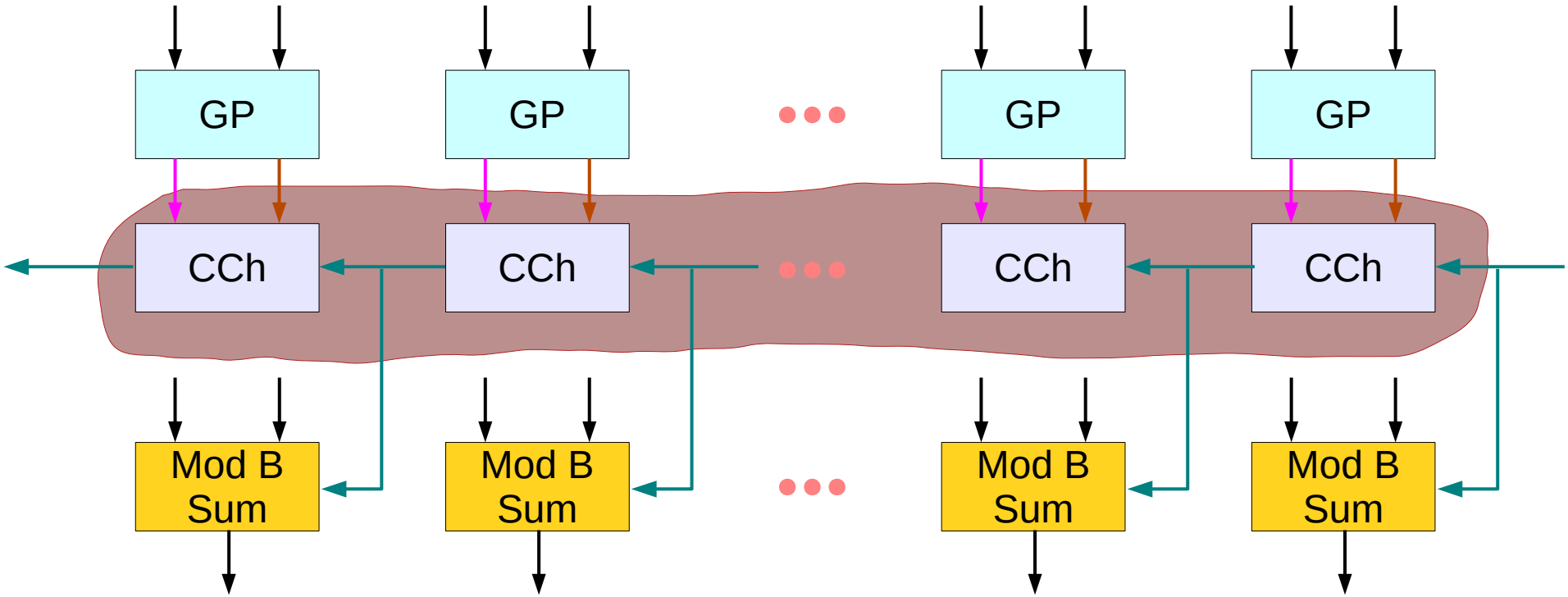
the last iteration includes **n B-ary** operations
computation of z(i) that could be executed in parallel

```
z(i) := (x(i)+y(i)+q(i)) mod B
```

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Sequential and concurrent computations

Splits the operations into **concurrent B-ary** ones (1st and 3rd iterations)
And **sequential binary** ones (2nd iterations)



Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

B-ary n-digit Carry Chain Adder VHDL Code

```
q(0) <= c_in;

iterative_step for i in 0 to n-1 generate
    p(i) <= '1' when x(i)+y(i) = B-1 else '0';
    g(i) <= '1' when x(i)+y(i) > B-1 else '0';
    with p(i) select q(i+1) <= q(i) when '1', g(i) when others;
    z(i) <= (x(i)+y(i)+ conv_integer(q(i))) mod B;
end generate;

c_out <= q(n);
```

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

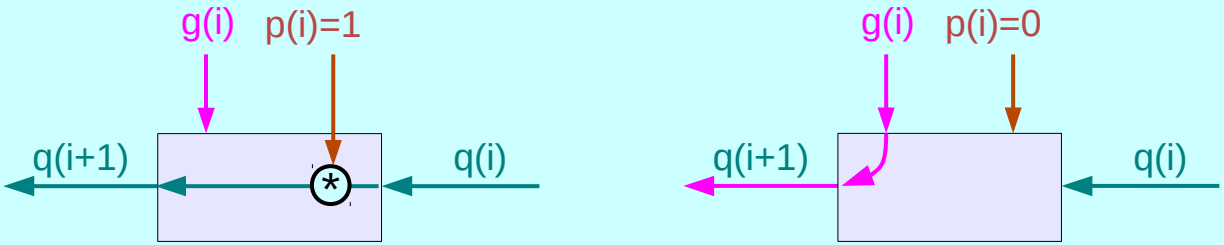
Computation Substitution

The **sequential binary** operations are the same whatever base **B** is

However, the expected computation time can be reduced
by the substitution of the relatively complex instruction

```
if x(i)+y(i)+q(i)>B-1 then q(i+1):=1 else q(i+1):=0 end if;
```

```
if p(i)=1 then q(i+1):= q(i) else q(i+1):=g(i); end if;
```



Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Relaxing g(i)

```
if p(i)=1 then q(i+1):= q(i) else q(i+1):=g(i); end if;
```

the corresponding Boolean equation

q(i+1) = p(i).q(i) V not(p(i)).g(i)

the generate function g(a,b) can be relaxed as bellows

g(a,b) = 1	if a + b > B-1 ….. when p(i) = 0 …. not(p(i))=1
g(a,b) = 0	if a + b < B-1 ….. when p(i) = 0 …. not(p(i))=1
g(a,b) = 1/0 dont care	if a + b = B-1 ….. when p(i) = 1 …. not(p(i))=0

the original generate and propagate function

g(a,b) = 1	if a + b > B-1,
g(a,b) = 0	otherwise
p(a,b) = 1	if a + b = B-1,
p(a,b) = 0	otherwise

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Relaxed g(i) Examples

	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

p(i)	0	1	2	3
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

g(i)	0	1	2	3
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1

Original g(i) **: 4-ary**

	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

p(i)	0	1	2	3
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

g(i)	0	1	2	3
0	0	0	0	X
1	0	0	X	1
2	0	X	1	1
3	X	1	1	1

Relaxed g(i) **: 4-ary**

	0	1
0	0	1
1	1	2

p(i)	0	1
0	0	1
1	1	0

g(i)	0	1
0	0	X
1	X	1

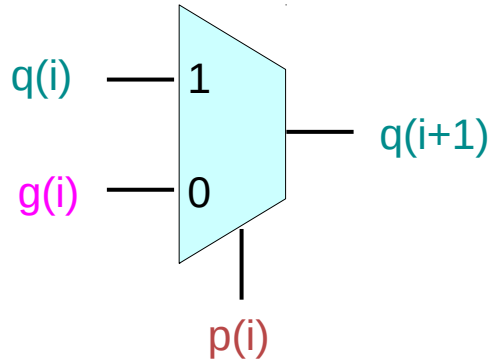
Relaxed g(i) **: binary**

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Multiplexer Carry Chain

```
if p(i)=1 then q(i+1):= q(i) else q(i+1):=g(i); end if;
```

q(i+1) = p(i).q(i) V not(p(i)).g(i)



	0	1
0	0	1
1	1	2

p(i)	0	1
0	0	1
1	1	0

g(i)	0	1
0	0	X
1	X	1

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Manchester Carry Chain

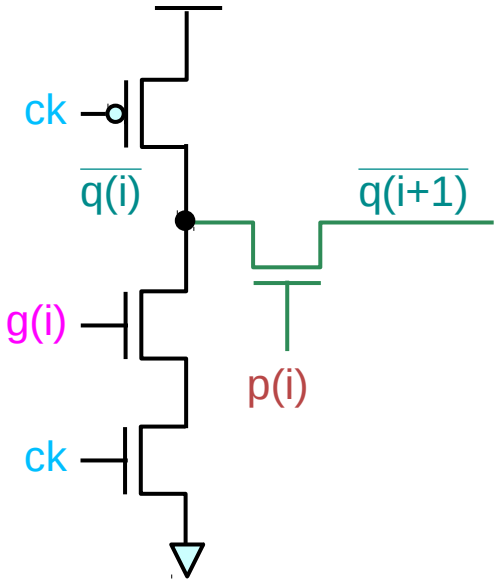
the output node $\overline{q(i+1)}$ is precharged,
when the synchronization signal is equal to 0 (ck=0),

when ck=1, the output node is discharged
if either p(i) =1 and the preceding node $\overline{q(i)}$
has been discharged, or
if g(i)=1.

In order that it works properly
g(i) and p(i) should not be equal to 1 simultaneously
so that the definition of g(i) cannot be relaxed
as in the preceding case

p(i)	0	1
0	0	1
1	1	0

g(i)	0	1
0	0	0
1	0	1



Manchester Carry Chain

Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Faster Adders

Bottle neck : any stage needs information regarding all preceding carry bits
to be able to compute its own sum and carry-out bits

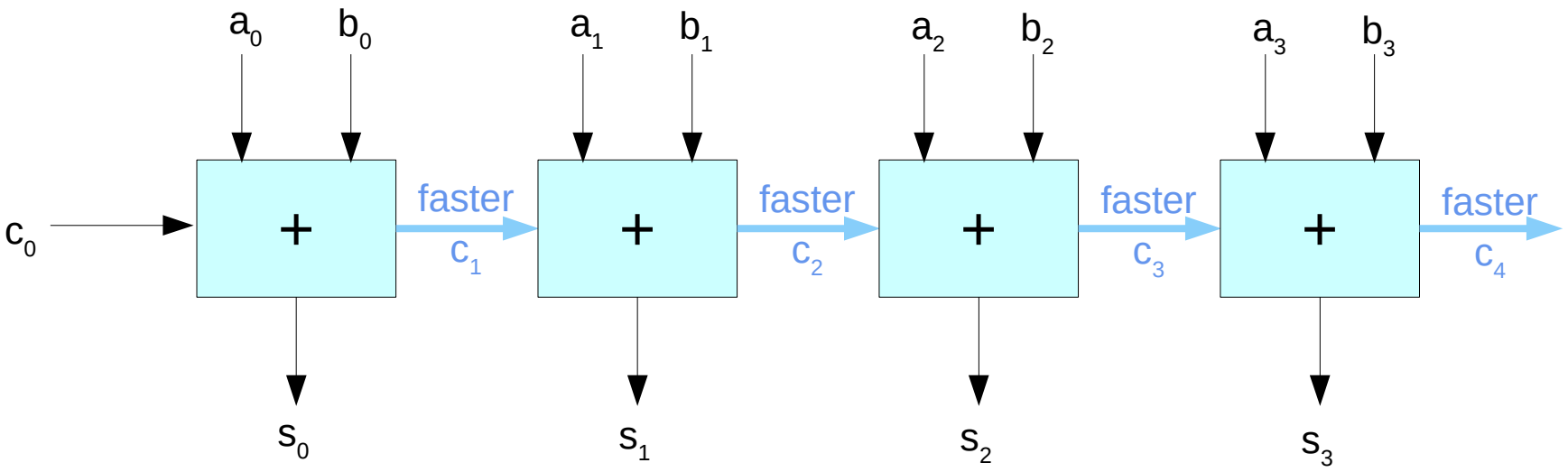
Faster Adders

1) **faster carry propagation** Manchester Carry Chain Adder
reduction of the time required
for the carry signal to propagate to the cell

2) **faster carry generation** Carry Lookahead Adder
local computation of the carry, without having to wait for the carry-out
produced by preceding stages

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

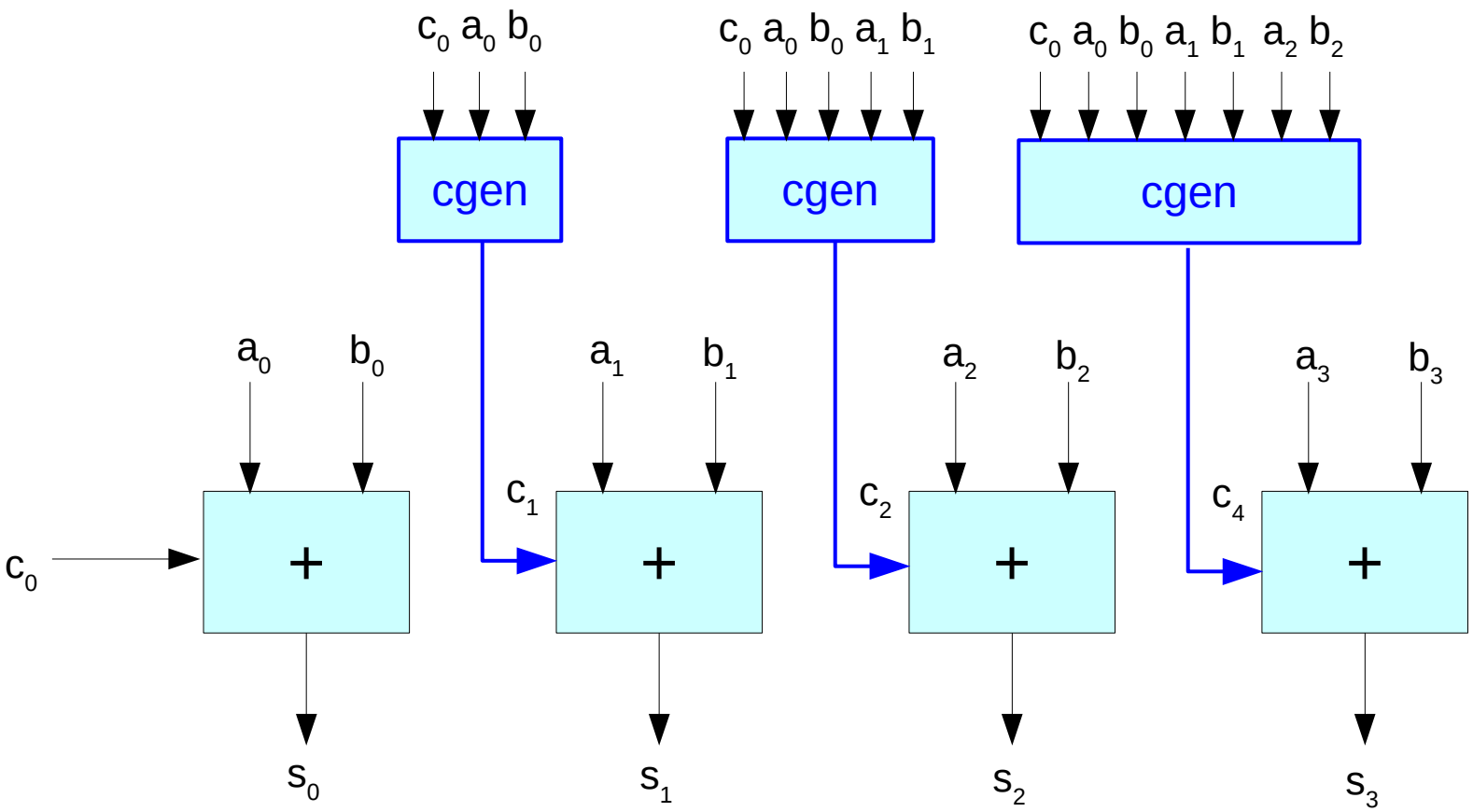
Faster Carry Propagation



Reduces the time needed for the carry to propagate through the cells

Digital Electronics and Design with VHDL,V, A< Pedroni

Faster Carry Generation



Each stage computes its own carry-in bit

Digital Electronics and Design with VHDL,V, A< Pedroni

Manchester Carry-Chain Adder

Manchester Carry-Chain Adder
--- faster carry propagation
carry propagate adder in which
the delay through the carry cells is reduced

Static or Dynamic Circuits

Thanks to the parameter G and P,
The delay is just one gate-delay

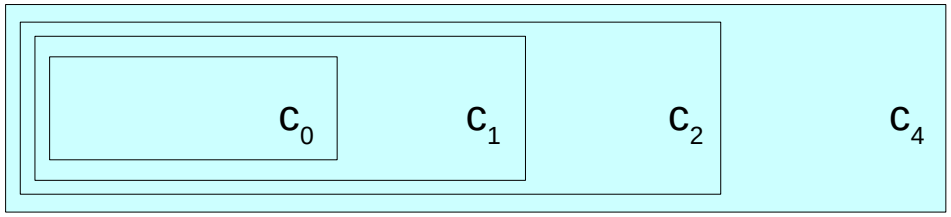
Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Manchester Carry Chain – Shared Logic

The **Manchester carry chain** is
a variation of the **carry-lookahead adder**
that uses **shared logic** to lower the transistor count.

the logic for **generating** each carry contains
all of the logic used to **generate the previous carries**.

A Manchester carry chain
generates the **intermediate carries**
by tapping off nodes in the gate that calculates
the **most significant carry value**.

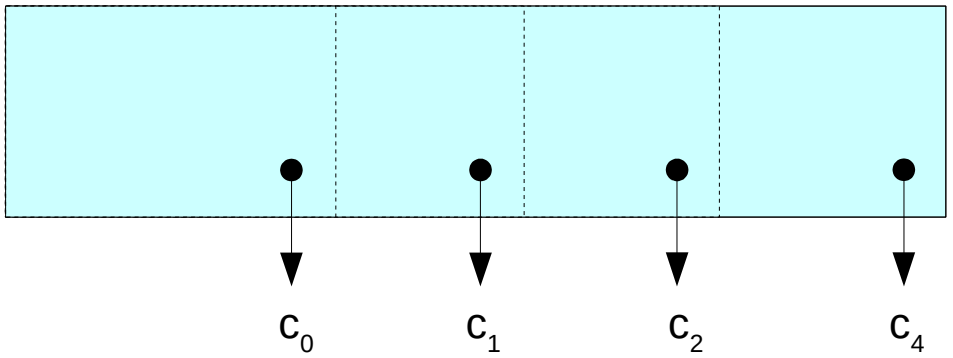


https://en.wikipedia.org/wiki/Carry-lookahead_adder

Manchester Carry Chain – tap to internal nodes

The Manchester adder stage improves
on the carry-lookahead implementation
by using a single $\mathbf{C_3}$ circuit

$\mathbf{C_2}$, $\mathbf{C_1}$, $\mathbf{C_0}$, are tapped to the internal nodes
of the single $\mathbf{C_3}$ circuit



Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

In addition to four Manchester stages,
the adder requires four PG generator blocks
One representative implementation

Four SUM generate blocks an XNOR gate complete the adder
This worst case propagation time can be improved
by bypassing the four stages if all carry propagate signals are true

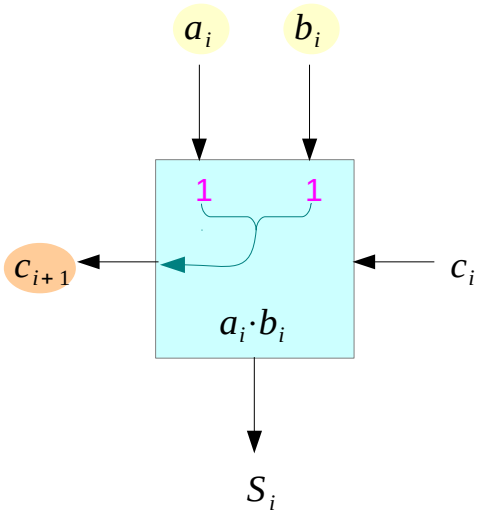
The optimum number of cascaded stages
may be calculated for a given technology by simulation
A final implementation of a 4-bit Manchester adder

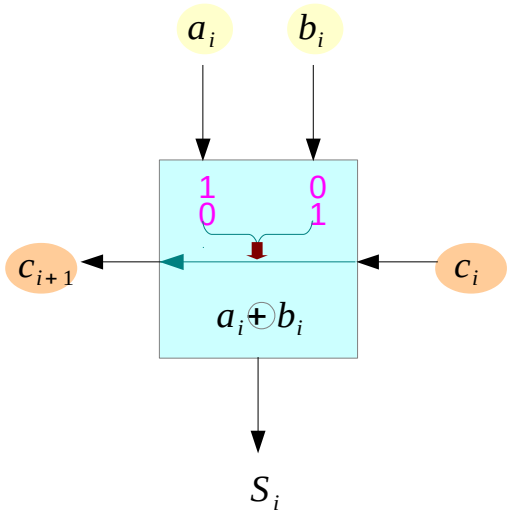
Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

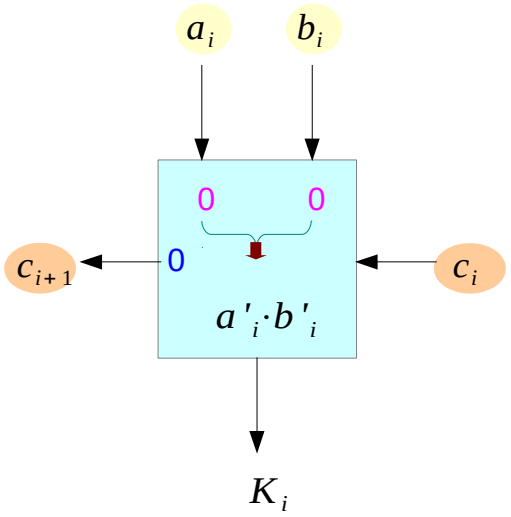
G, P, and K

Generate	$G_i = a_i \cdot b_i$
Propagate	$P_i = a_i \oplus b_i$
Kill	$K_i = \bar{a}_i \cdot \bar{b}_i$

$$c_{out} = G_i + P_i c_i$$



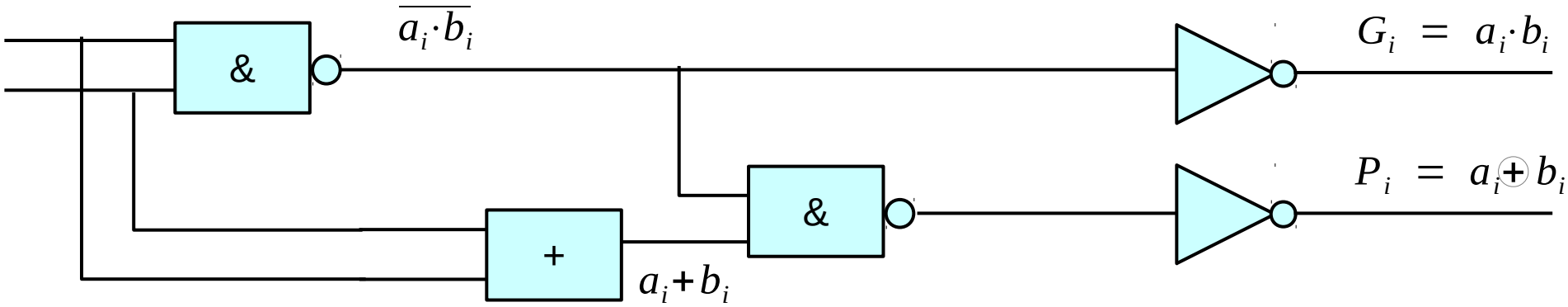




PG Circuits

Generate $G_i = a_i \cdot b_i$

Propagate $P_i = a_i \oplus b_i$



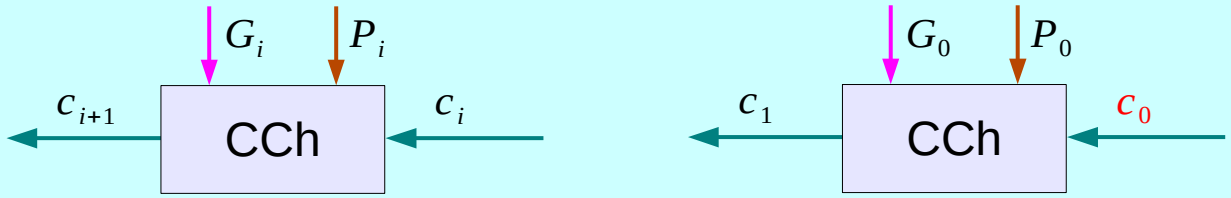
$\overline{(a_i+b_i)\overline{(a_i \cdot b_i)}}$ $\quad (a_i+b_i)\overline{(a_i \cdot b_i)}$

$\overline{(a_i+b_i)}+(a_i \cdot b_i)$ $\quad (a_i+b_i)\cdot(\overline{a_i}+\overline{b_i})$

$\overline{a_i} \cdot \overline{b_i} + a_i \cdot b_i$ $\quad \overline{a_i} \cdot b_i + a_i \cdot \overline{b_i}$

Carry Equations (1)

$$c_{i+1} = G_i + P_i c_i$$



c_i : carry in of the i-th stage

$$c_1 = G_0 + P_0 c_0$$
$$c_2 = G_1 + P_1 c_1$$
$$c_3 = G_2 + P_2 c_2$$
$$c_4 = G_3 + P_3 c_3$$

Carry Chain Adder

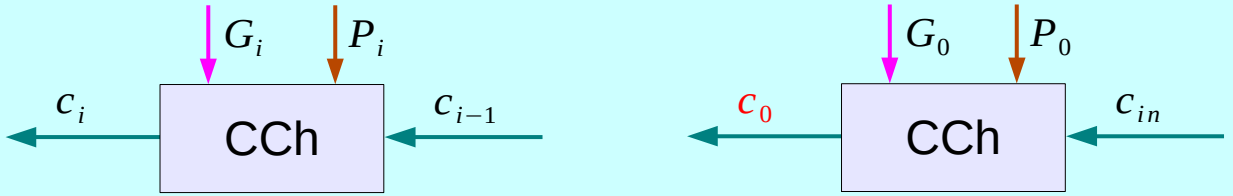
$$c_1 = G_0 + P_0 c_0$$
$$c_2 = G_1 + P_1 \left[G_0 + P_0 c_0 \right]$$
$$c_3 = G_2 + P_2 \left[G_1 + P_1 \left[G_0 + P_0 c_0 \right] \right]$$
$$c_4 = G_3 + P_3 \left[G_2 + P_2 \left[G_1 + P_1 \left[G_0 + P_0 c_0 \right] \right] \right]$$

Carry Lookahead Adder

$$G_0 + P_0 c_0 = c_1$$
$$G_1 + P_1 G_0 + P_1 P_0 c_0 = c_2$$
$$G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0 = c_3$$
$$G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0 = c_4$$

Carry Equations (2)

$$c_i = G_i + P_i c_{i-1}$$



c_i : carry out of the i-th stage

$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 c_0$$
$$c_2 = G_2 + P_2 c_1$$
$$c_3 = G_3 + P_3 c_2$$

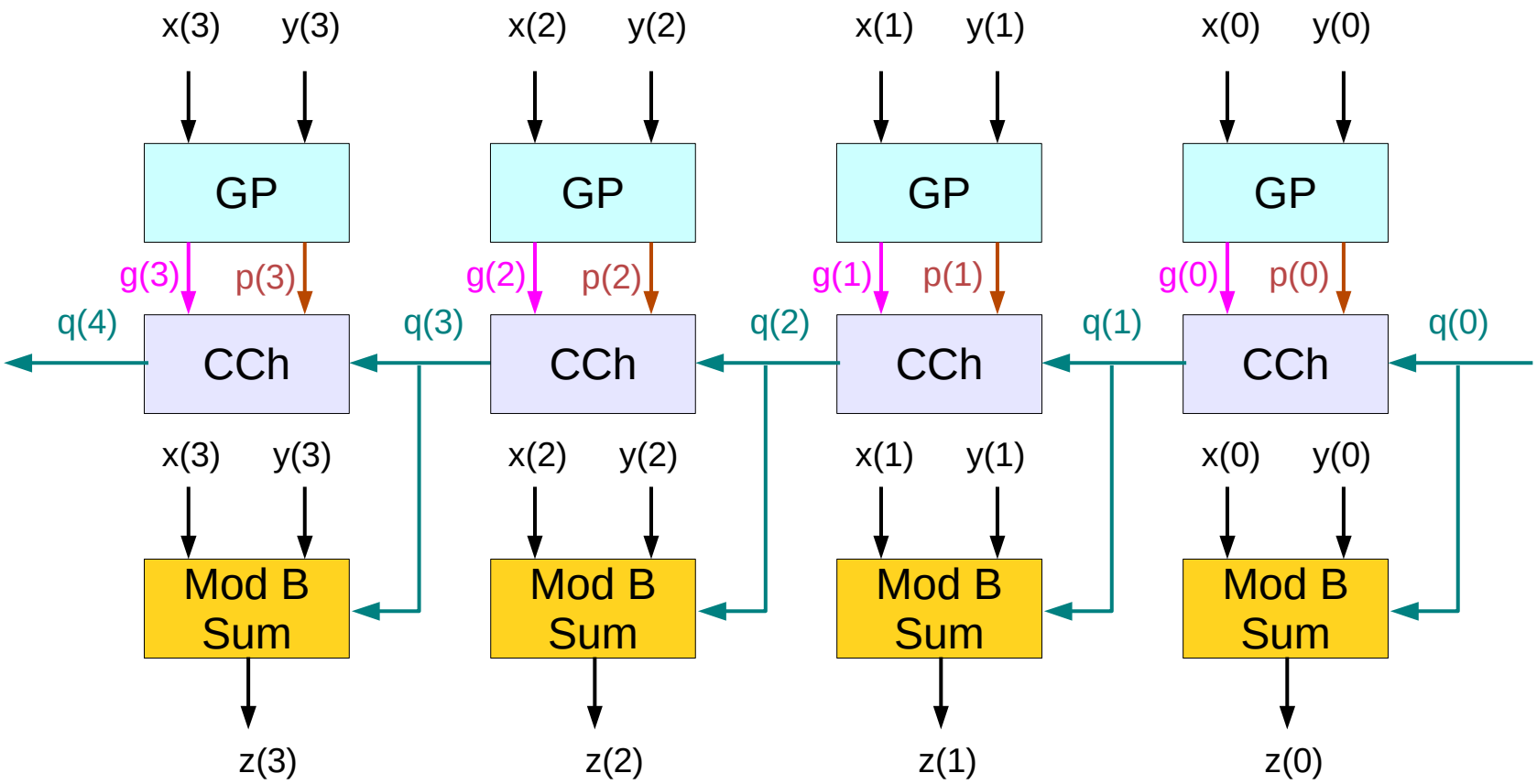
Carry Chain Adder

$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 \left[G_0 + P_0 c_{in} \right]$$
$$c_2 = G_2 + P_2 \left[G_1 + P_1 \left[G_0 + P_0 c_{in} \right] \right]$$
$$c_3 = G_3 + P_3 \left[G_2 + P_2 \left[G_1 + P_1 \left[G_0 + P_0 c_{in} \right] \right] \right]$$

Carry Lookahead Adder

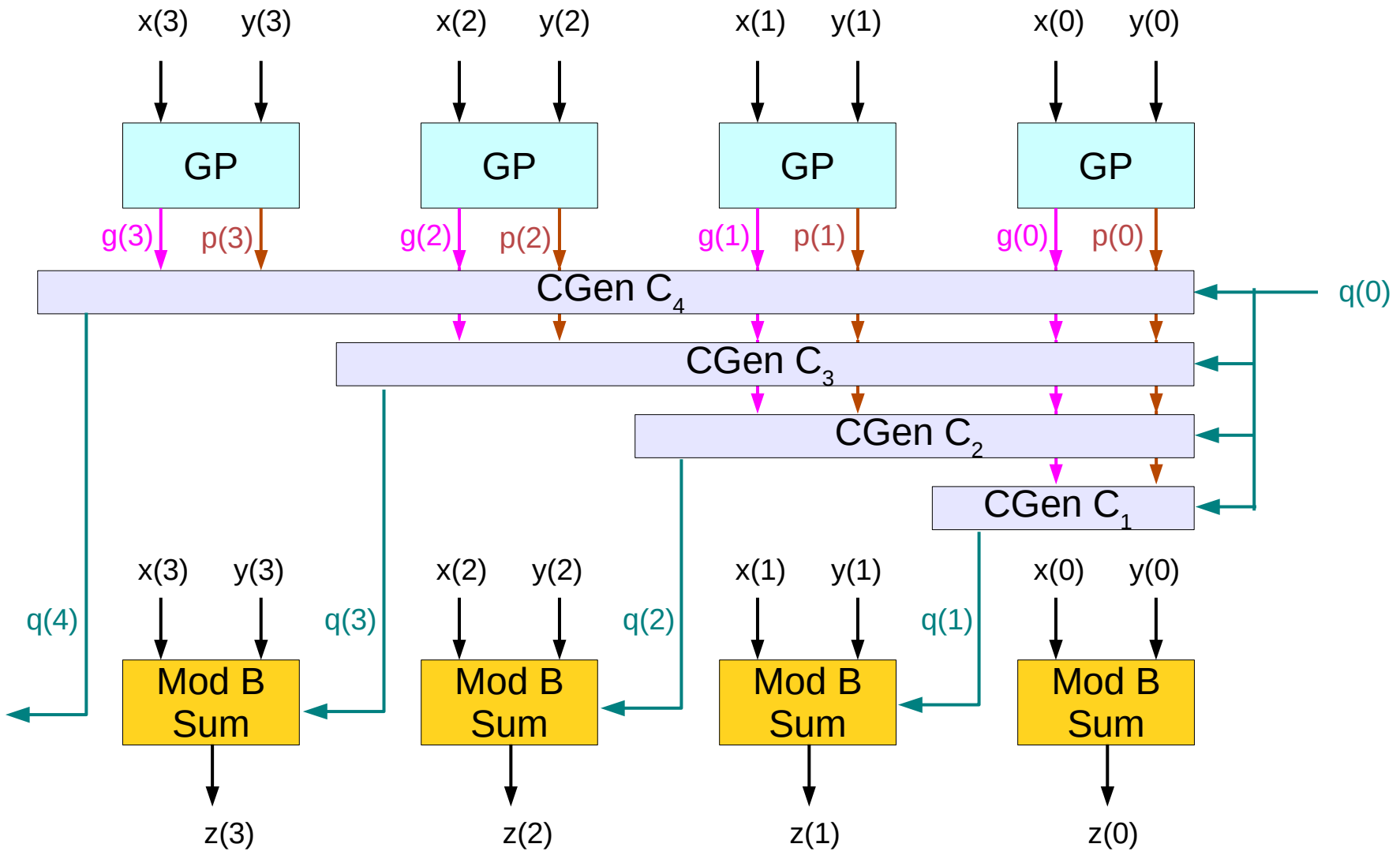
$$G_0 + P_0 c_{in} = c_0$$
$$G_1 + P_1 G_0 + P_1 P_0 c_{in} = c_1$$
$$G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_{in} = c_2$$
$$G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_{in} = c_3$$

Carry Chain Adder



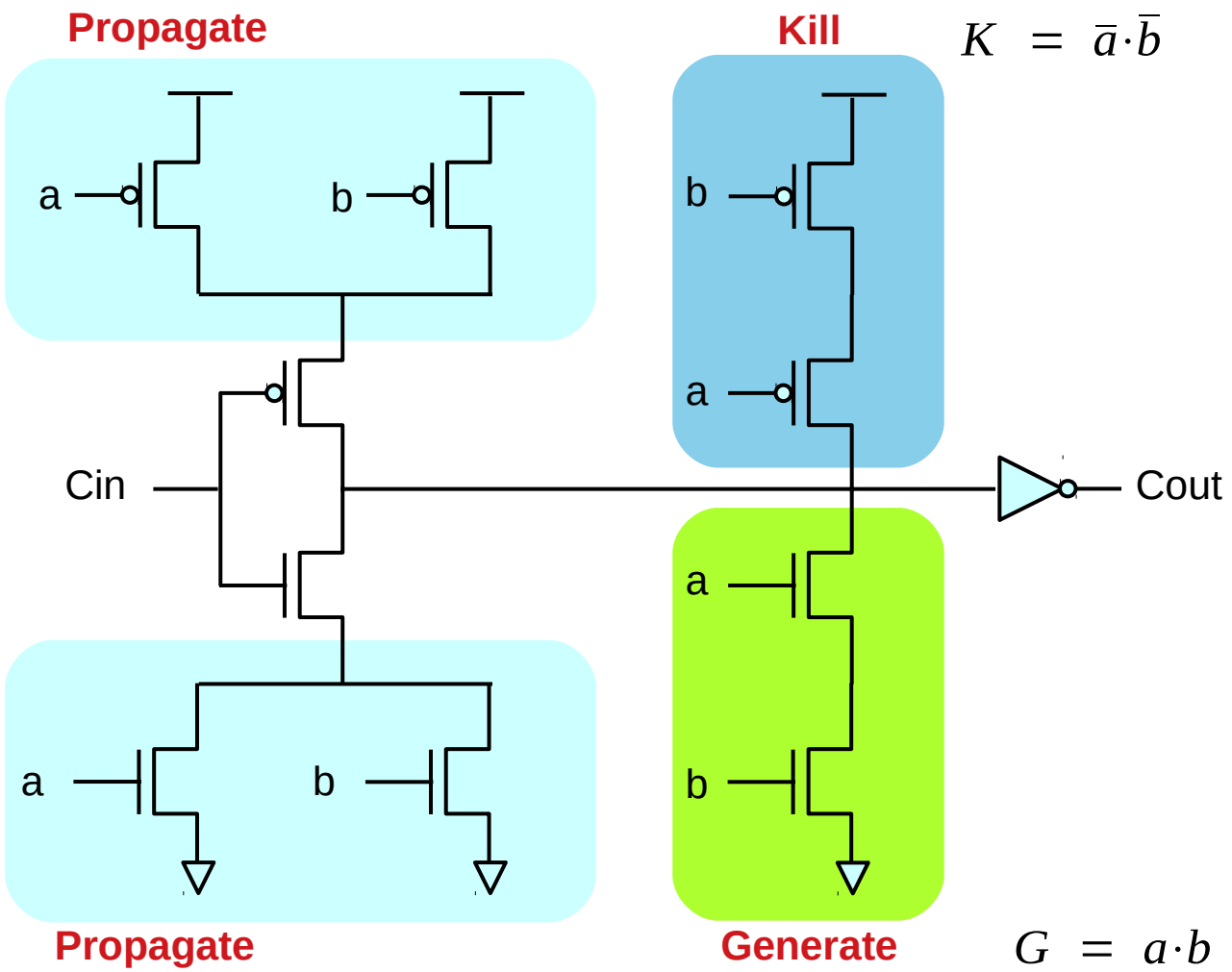
Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

Carry Lookahead Adder



Synthesis of Arithmetic Circuits: FPGA, ASIC and Ebedded Systems, J-P Deschamps et al

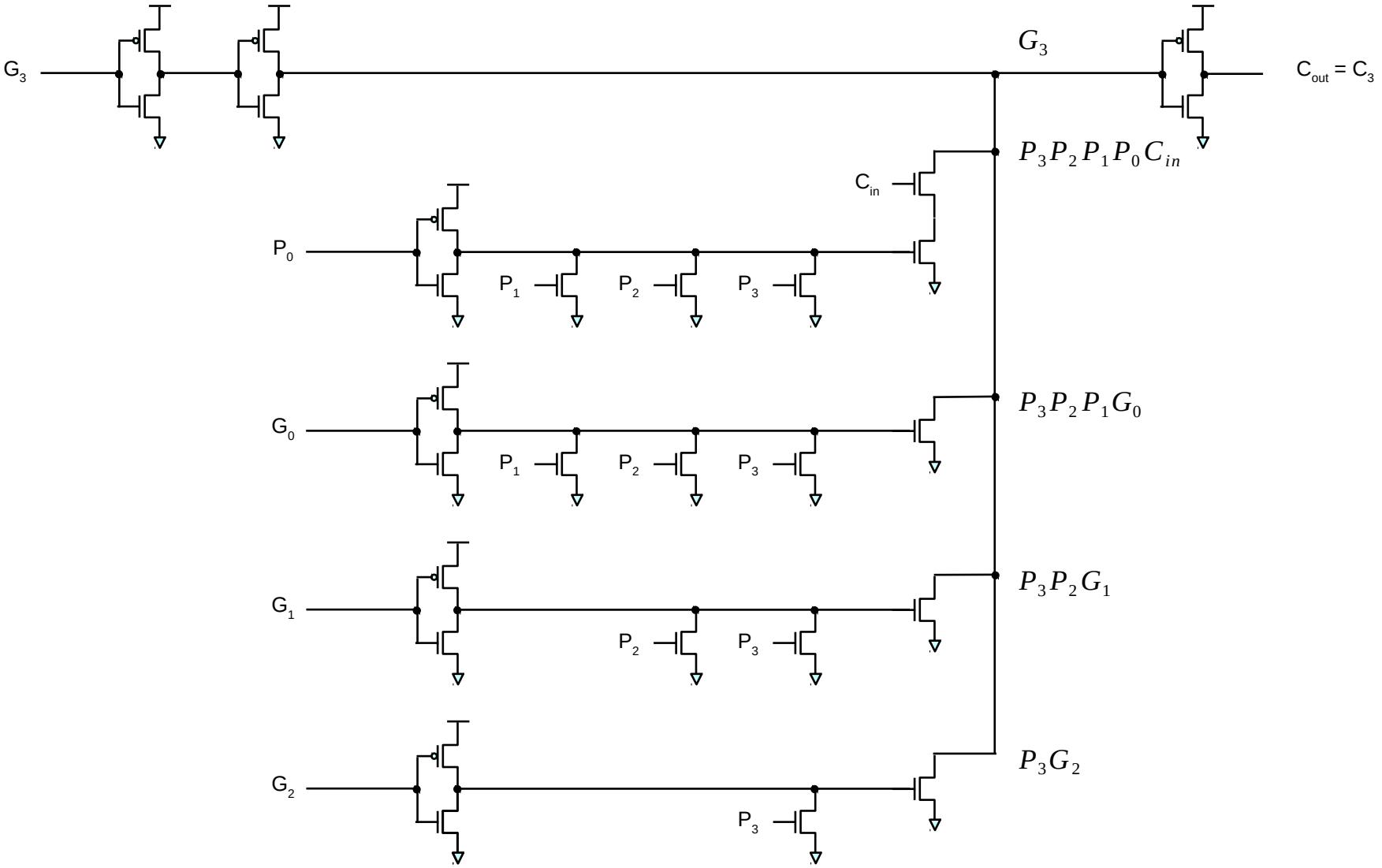
Carry section of FA



Digital Electronics and Design with VHDL,V, A< Pedroni

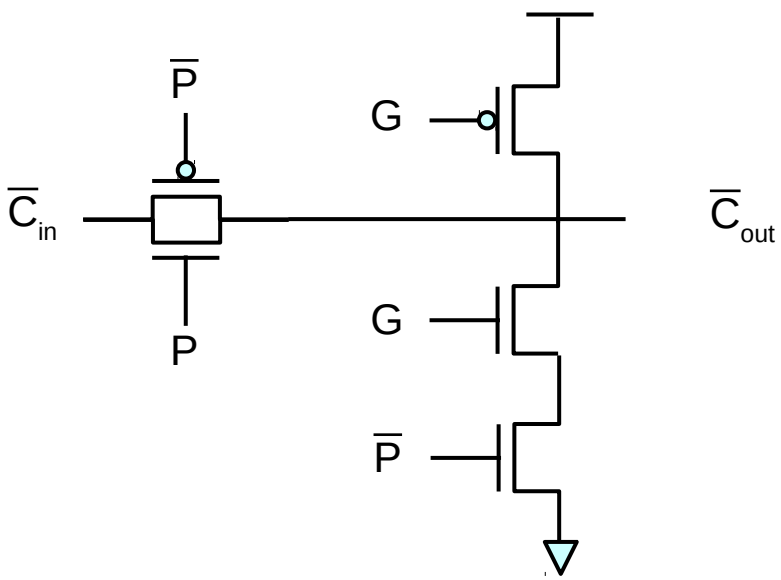
Static Carry Circuit

$$G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_{in} = c_3$$



Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Static Carry Circuit – using G, P



P cannot be relaxed

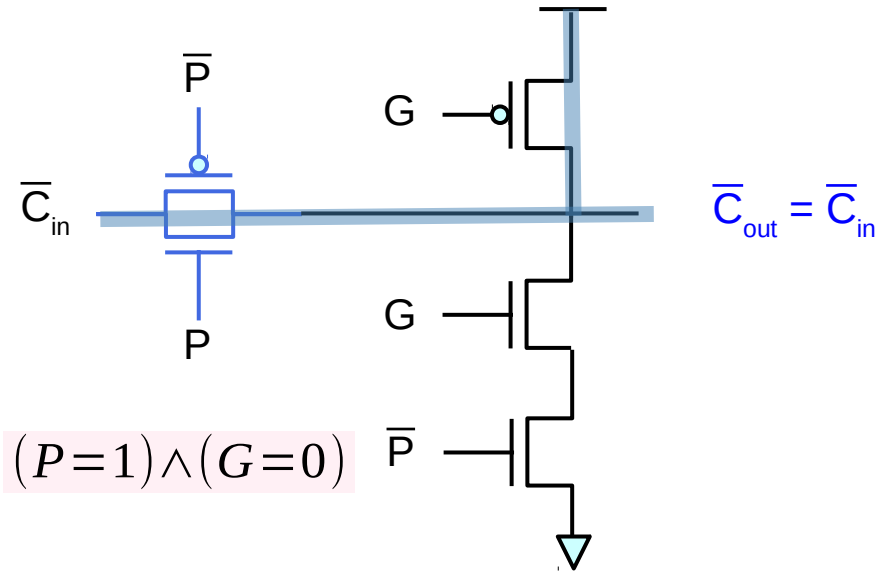
$$P = a \oplus b$$

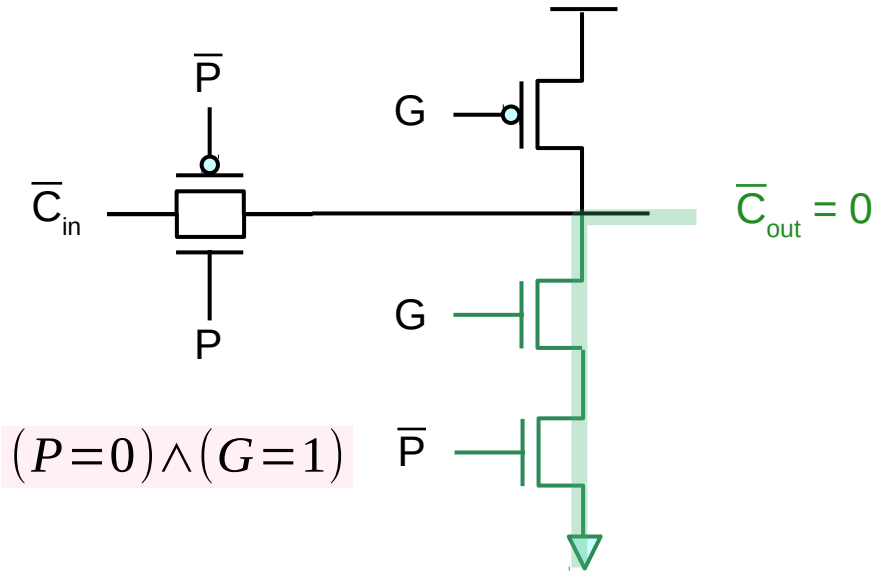
~~$P = a + b$~~

$$G = a \cdot b$$

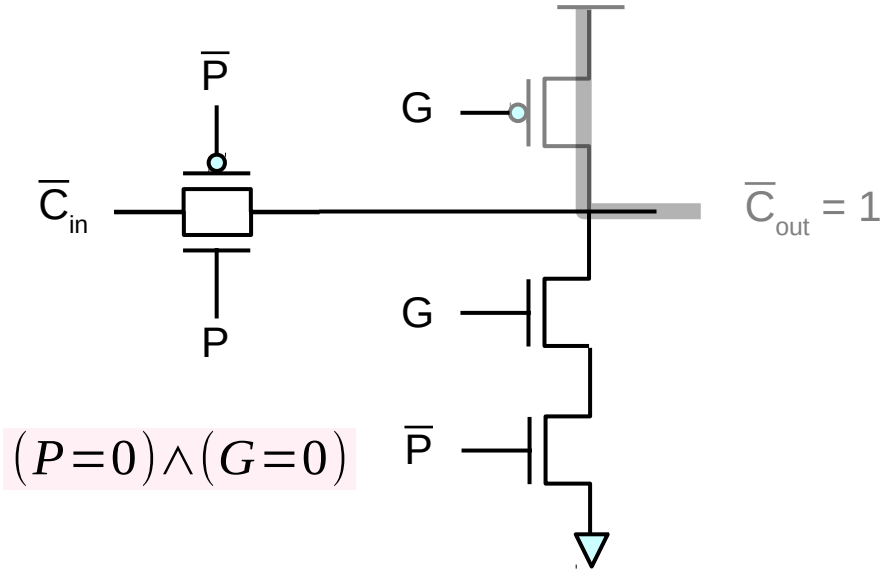
Digital Electronics and Design with VHDL,V, A< Pedroni

Static Carry Circuit – using G, P



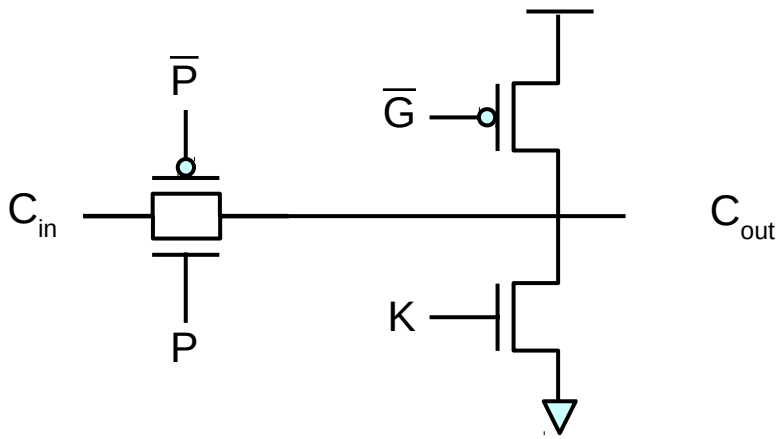


$$P = a \oplus b$$
$$G = a \cdot b$$



Digital Electronics and Design with VHDL,V, A< Pedroni

Static Carry Circuit – using G, P, K



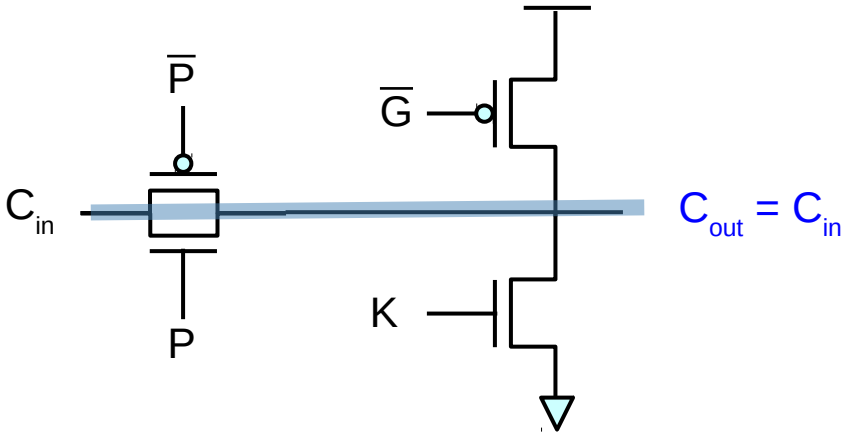
$$P = a \oplus b$$

$$G = a \cdot b$$

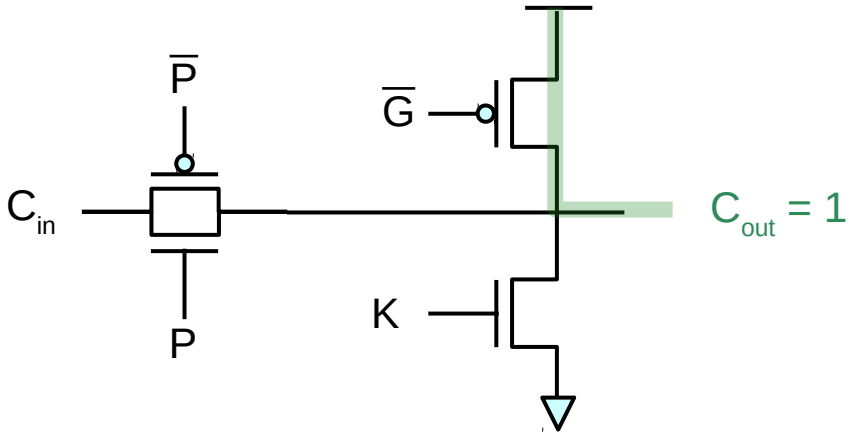
$$K = \bar{a} \cdot \bar{b}$$

Digital Electronics and Design with VHDL,V, A< Pedroni

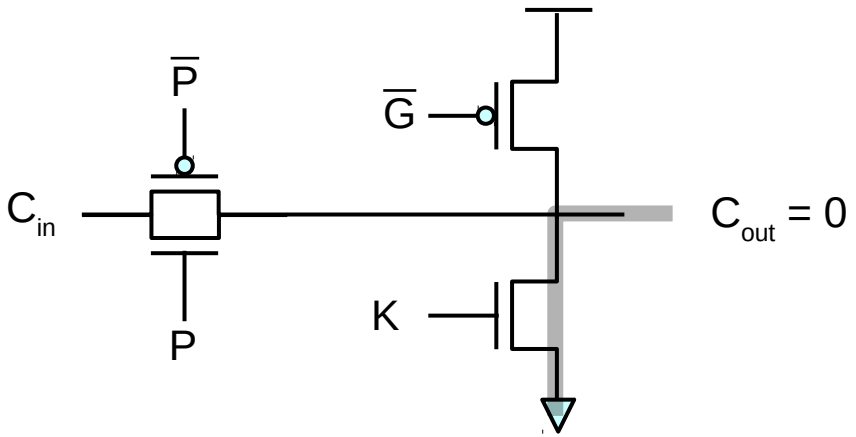
Static Carry Circuit – using G, P, K



$$P = a \oplus b$$



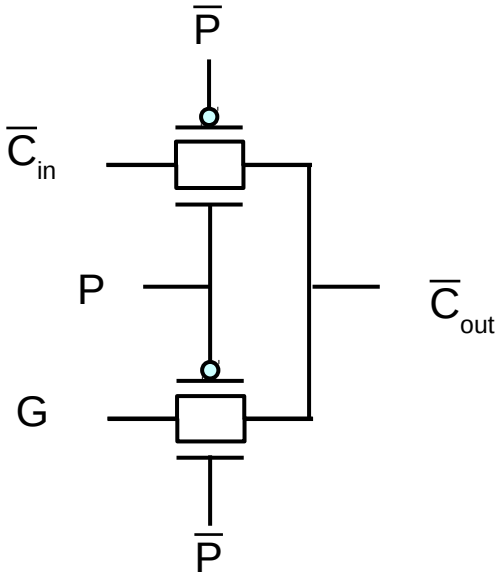
$$G = a \cdot b$$

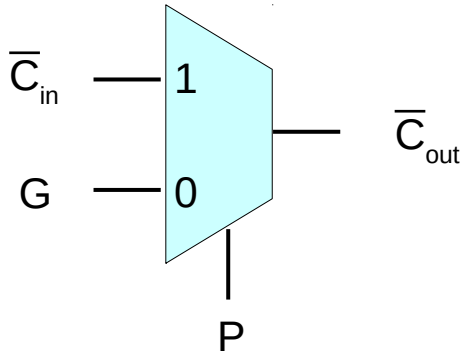


$$K = \bar{a} \cdot \bar{b}$$

Digital Electronics and Design with VHDL,V, A< Pedroni

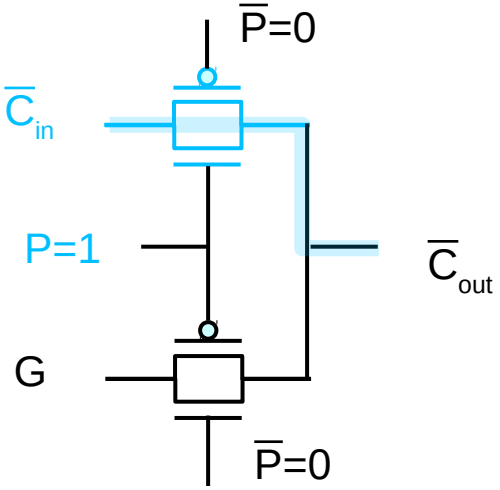
Static Carry Circuit – using Multiplexer

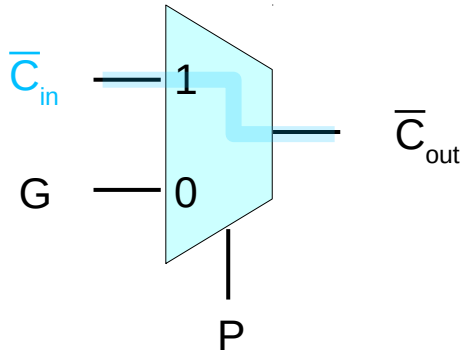


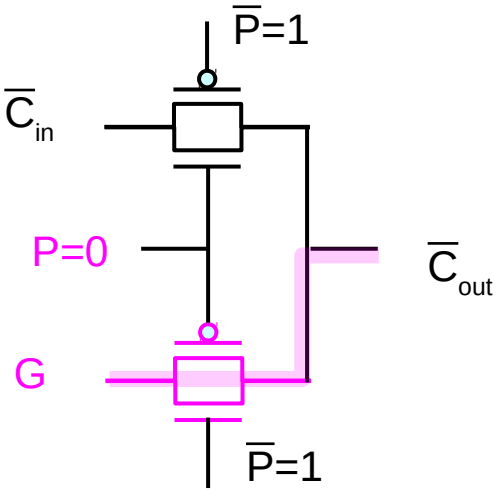


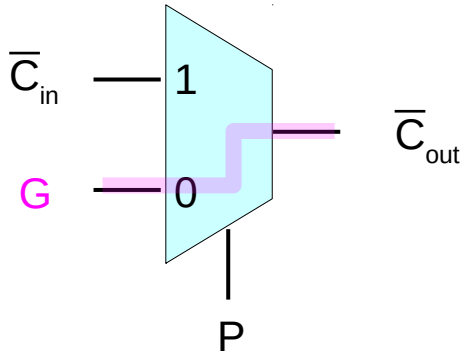
Digital Electronics and Design with VHDL,V, A< Pedroni

Static Carry Circuit – using Multiplexer



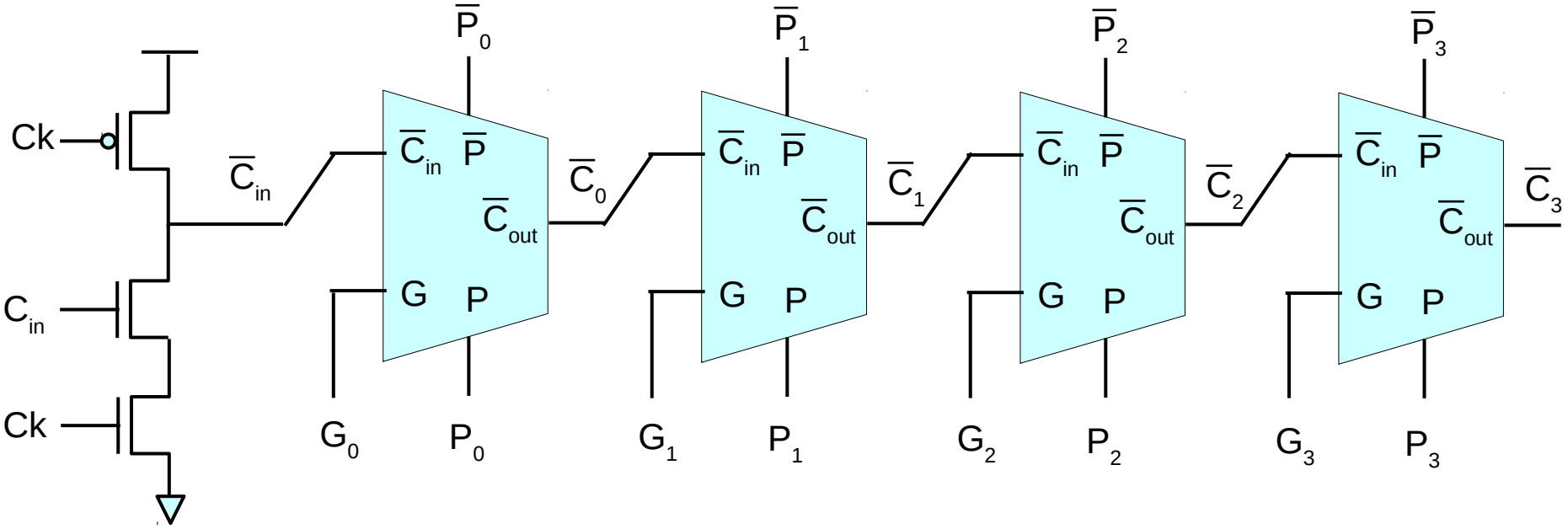


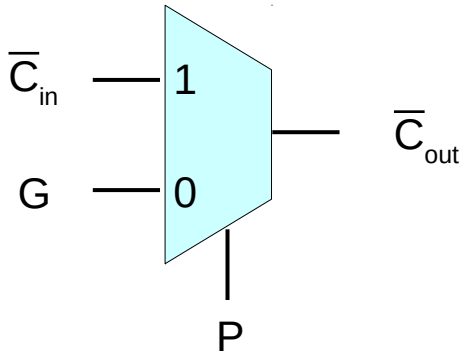


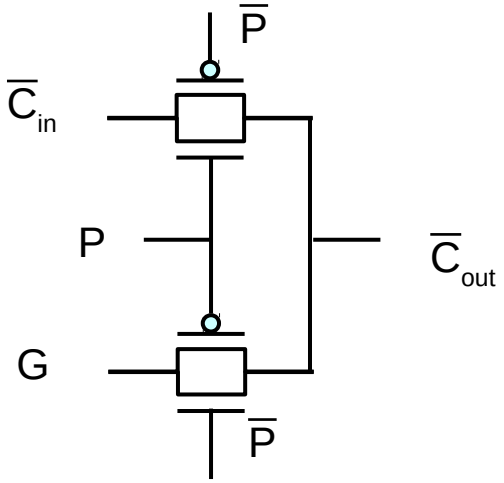


Digital Electronics and Design with VHDL,V, A< Pedroni

Static Carry Circuit – using multiplexers

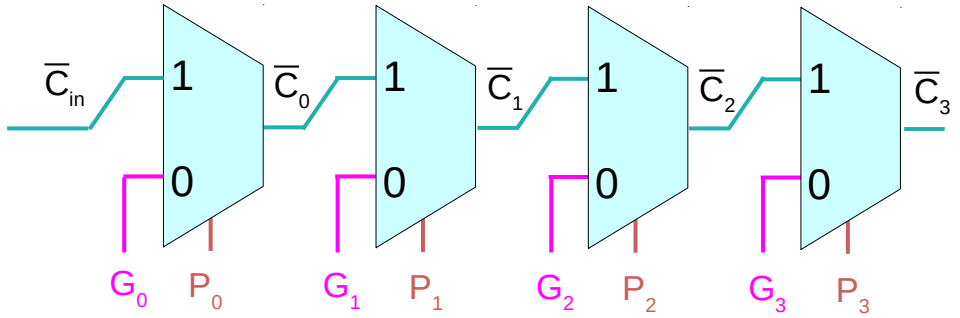


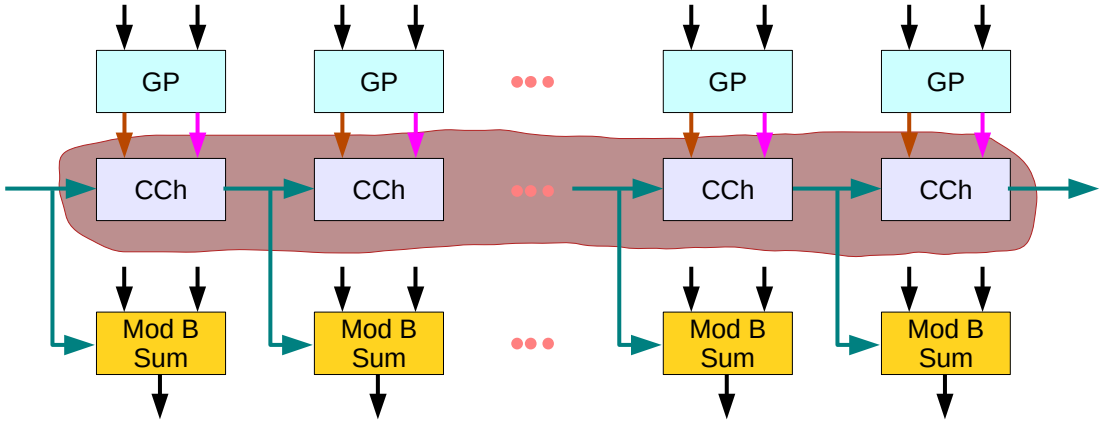




Digital Electronics and Design with VHDL,V, A< Pedroni

Static Carry Circuit – using multiplexers





Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

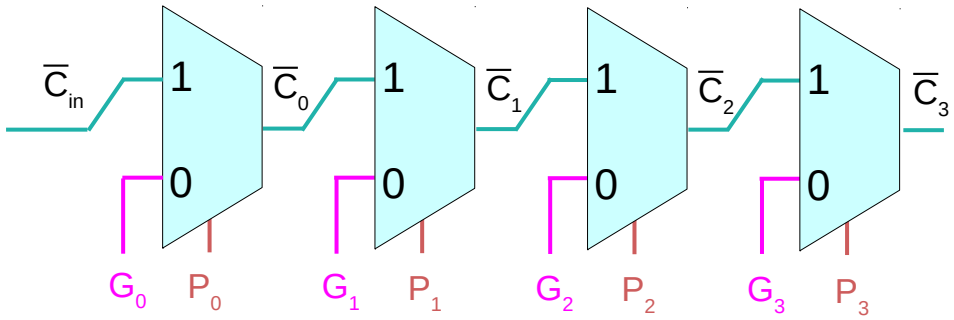
Static Carry Circuit – using multiplexers

A multiplexer-based 4-bit adder

cascading four such stages
supplying P_i, G_i

This is commonly called
a **Manchester carry adder**

not **Manchester carry chain adder**

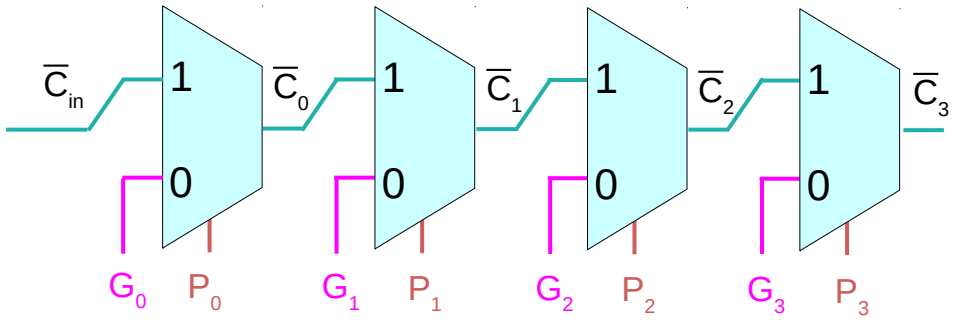


There is some similarity with the domino carry circuit (nMos)
Manchester carry chain adder

However, the intermediate carry gates are no longer needed, (G_i, P_i)
Because the carry values are available in a distributed fashion

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Static Carry Circuit – using multiplexers



The 4-bit adder is chosen to reduce
the number of series-propagate transistors
Which improves the speed

Note that if all propagate signals are true,
and CI is high , six series n-transistors
Pull the output node lo in the case of the dynamic gate
While five transistors re in series in the static gate

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Manchester Carry Chain – Dynamic Logic

However, not all logic families have these **internal nodes**,
CMOS being a major example.

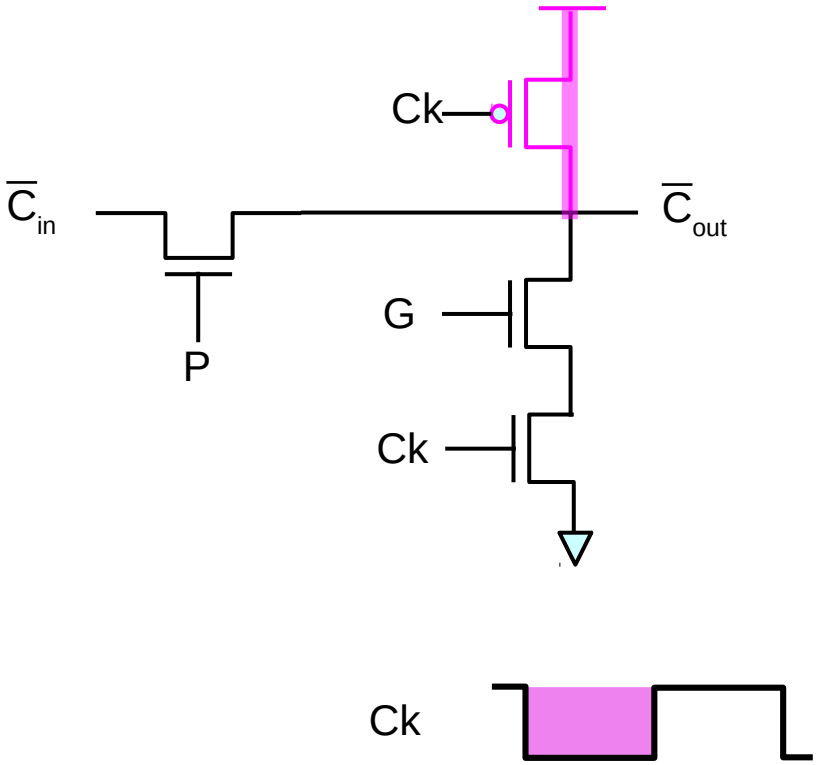
Dynamic logic can support shared logic,
as can transmission gate logic.

One of the major downsides of the Manchester carry chain
is that the **capacitive load** of all of these outputs,
together with the resistance of the transistors causes
the **propagation delay** to increase much *more quickly*
than a regular carry lookahead.

A Manchester-carry-chain section generally
doesn't exceed 4 bits.

https://en.wikipedia.org/wiki/Carry-lookahead_adder

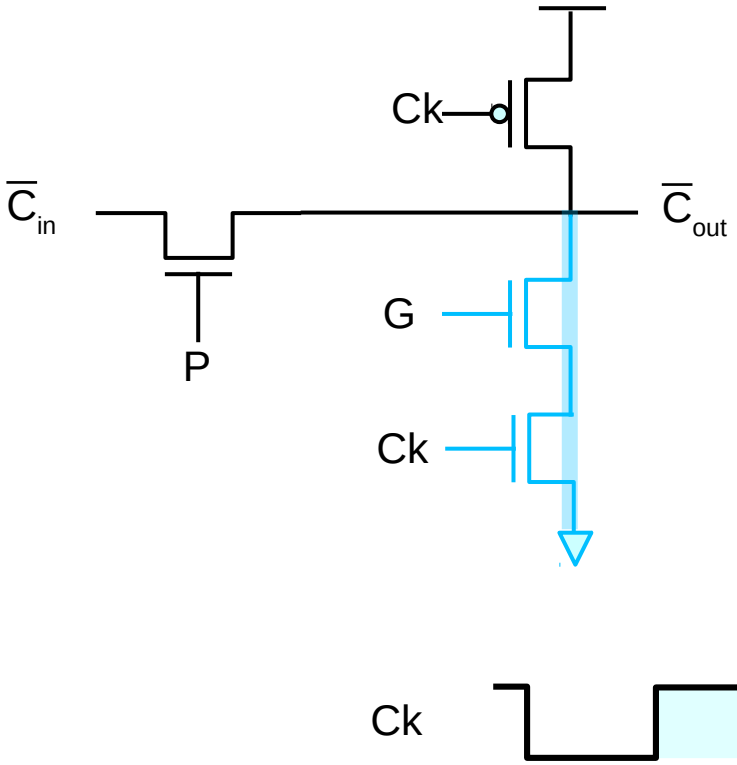
Dynamic Carry Circuit – using G, P



When **CLK** is **low**, the output node
is precharged by the **pull-up** transistor

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Dynamic Carry Circuit – using G, P

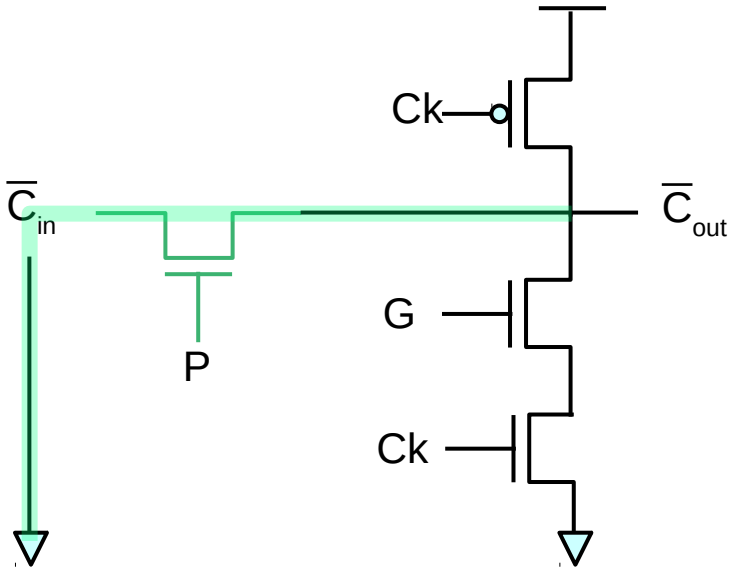


When **CLK** goes **high**,
the n **pull-down** transistor turns on
If carry generate **G=AB** is **true**,
then the output node **discharge**

$$c_i = G_i + P_i c_{i-1}$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Dynamic Carry Circuit – using G, P



When **CLK** goes **high**,
the n **pull-down** transistor turns on
If carry propagate **P=A+B** is **true**,
then a **previous carry** may be coupled
to the output node,
conditionally **discharging** it

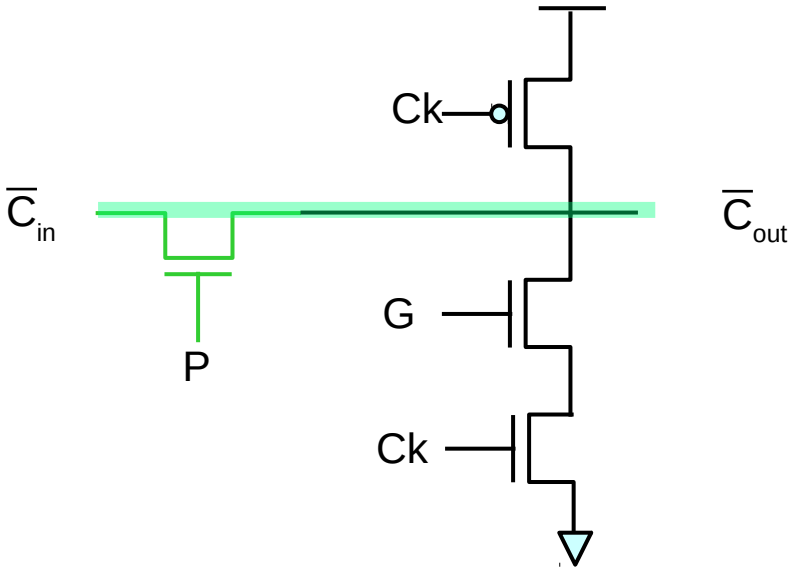
Note that in this circuit CARRY is actually propagated

$$c_i = G_i + P_i c_{i-1}$$

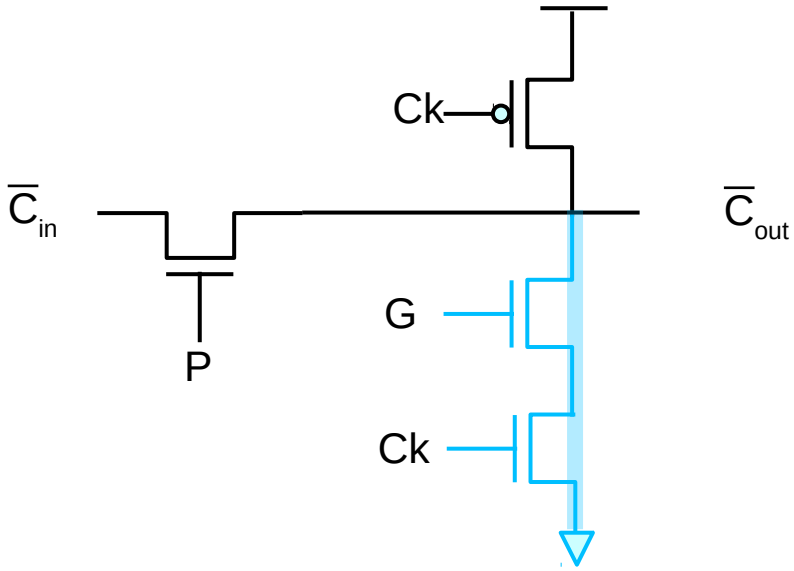
Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Dynamic Carry Circuit – using G, P

$$c_i = G_i + P_i c_{i-1}$$



$$c_i = G_i + P_i c_{i-1}$$



This requires P must not be relaxed

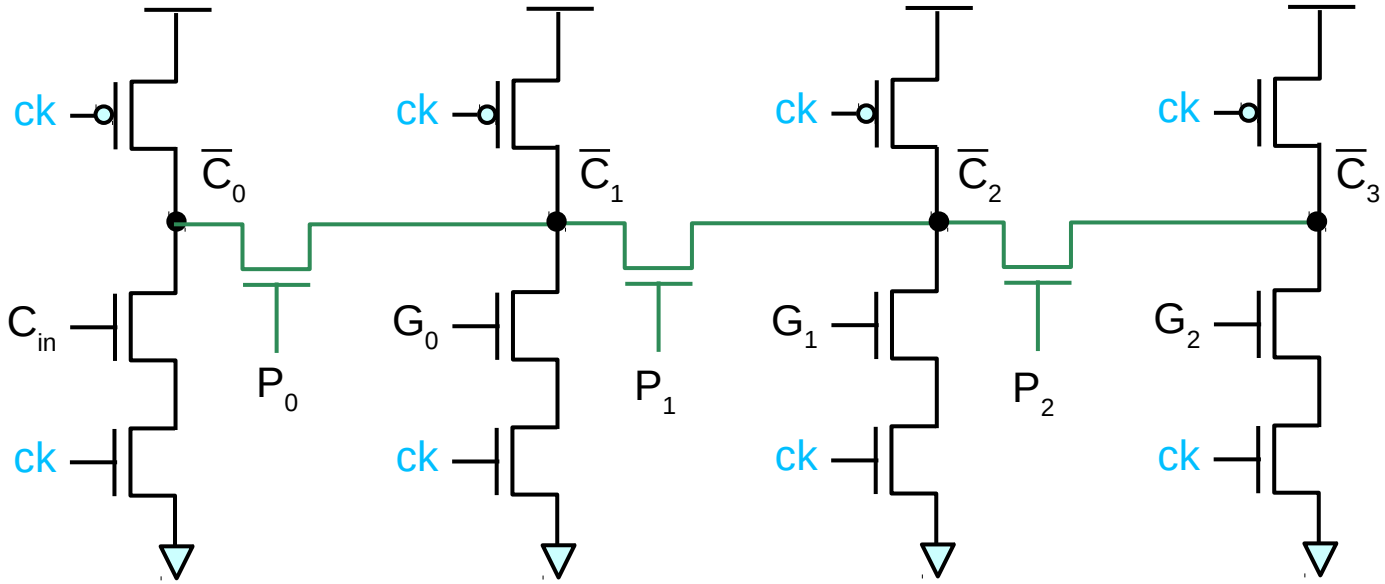
$$P = a \oplus b$$

p(i)	0	1
0	0	1
1	1	0

g(i)	0	1
0	0	0
1	0	1

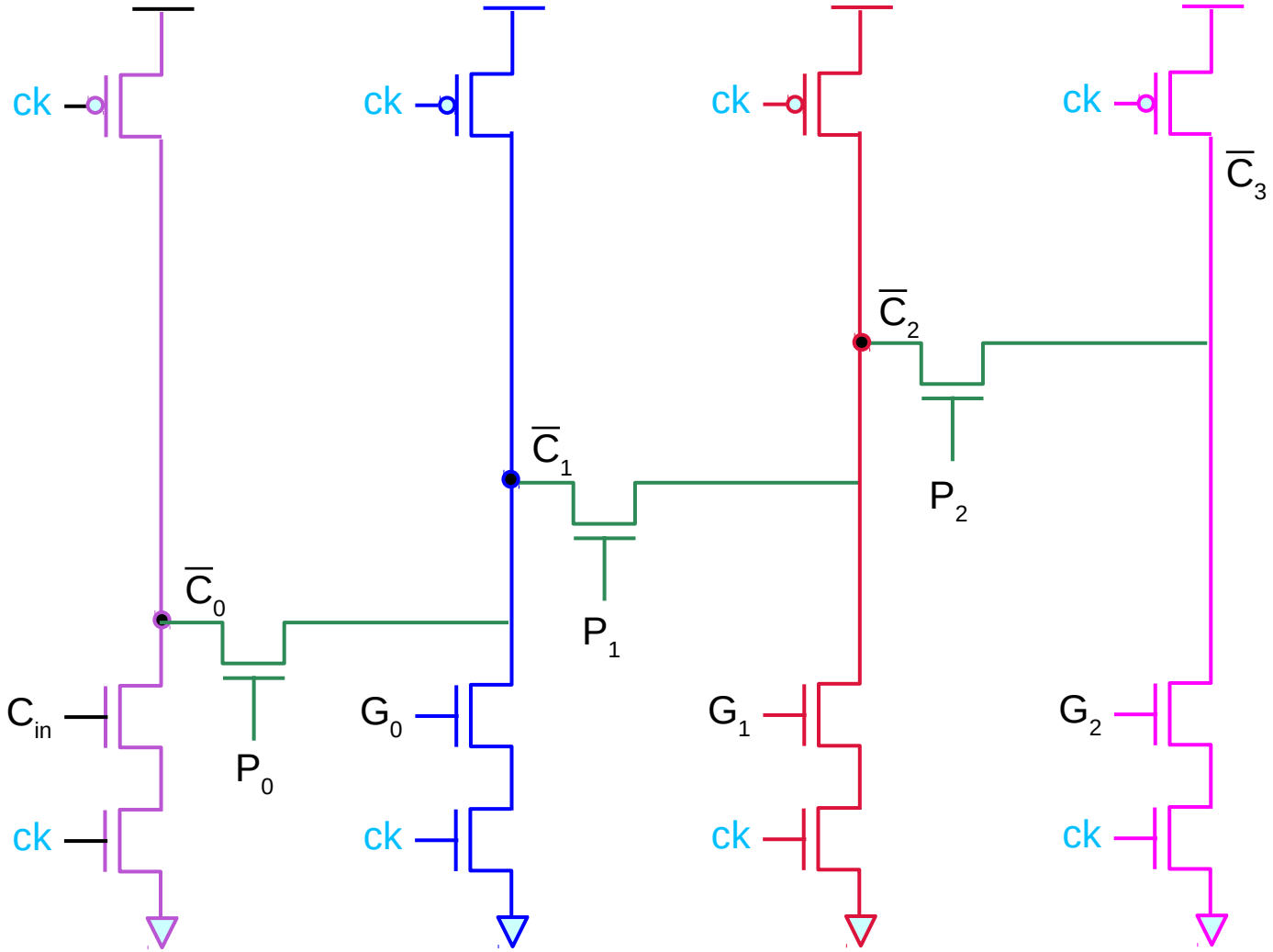
Digital Electronics and Design with VHDL,V, A< Pedroni

Dynamic Manchester Carry-Chain Adder



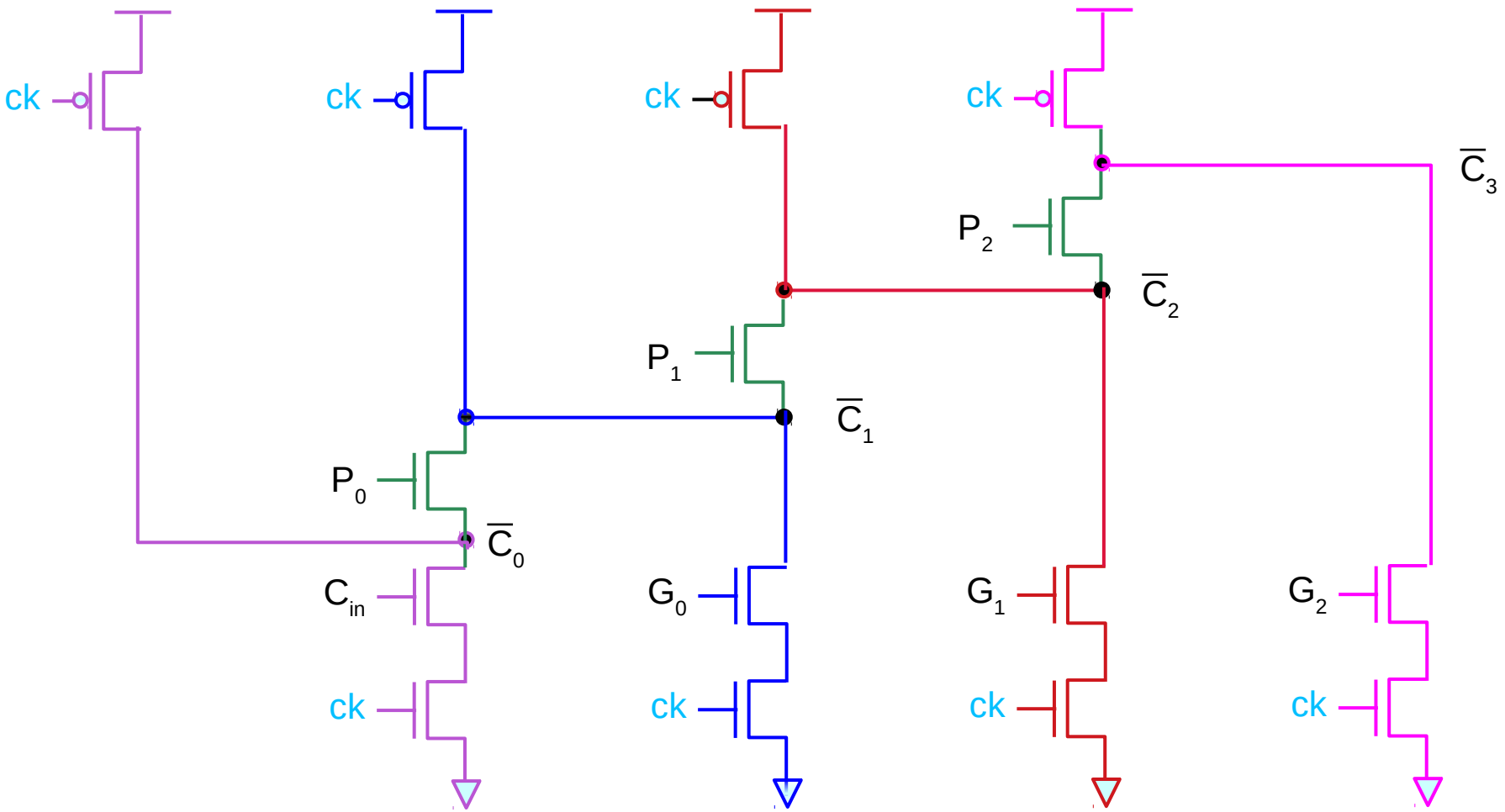
Digital Electronics and Design with VHDL,V, A< Pedroni

Other representation I



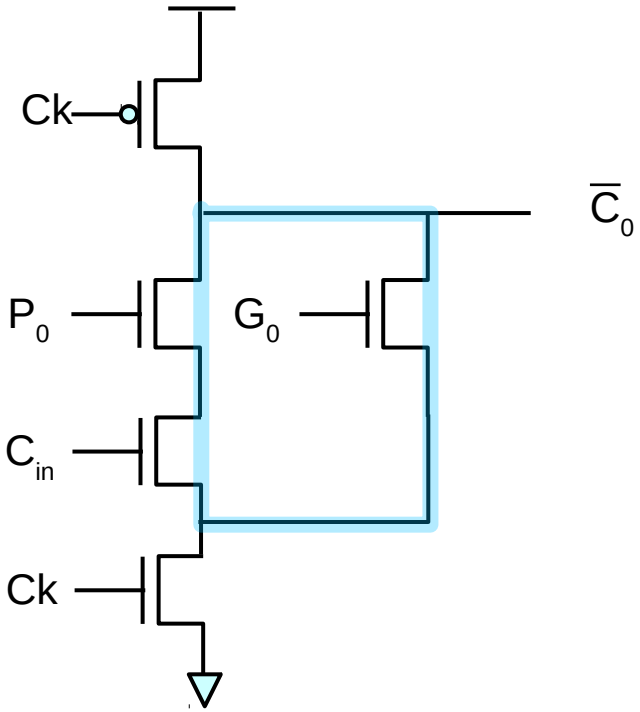
Digital Electronics and Design with VHDL,V, A< Pedroni

Other representation II



Digital Electronics and Design with VHDL,V, A< Pedroni

Dynamic Carry Circuit – C_0

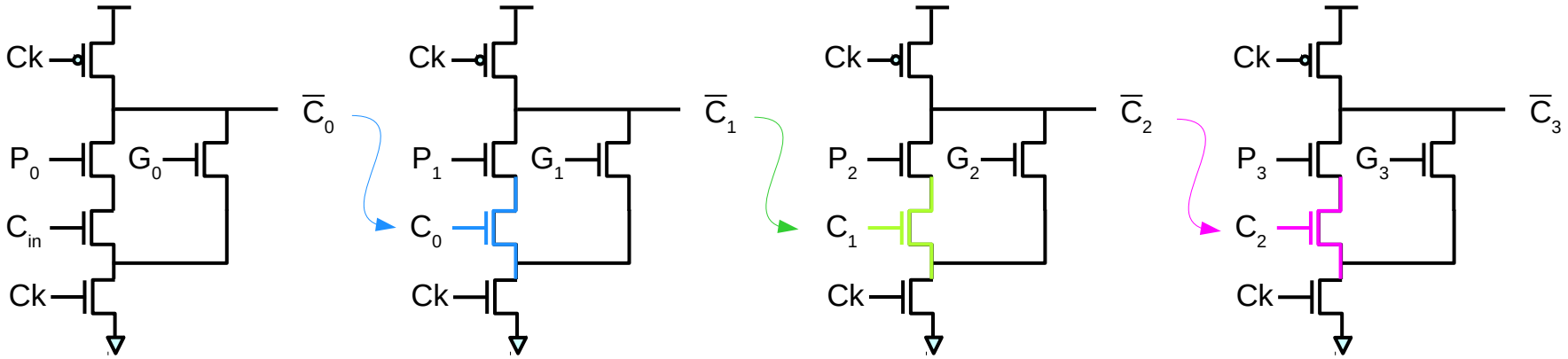


$$c_0 = G_0 + P_0 c_{in}$$

$$c_{out} = P c_i + G$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

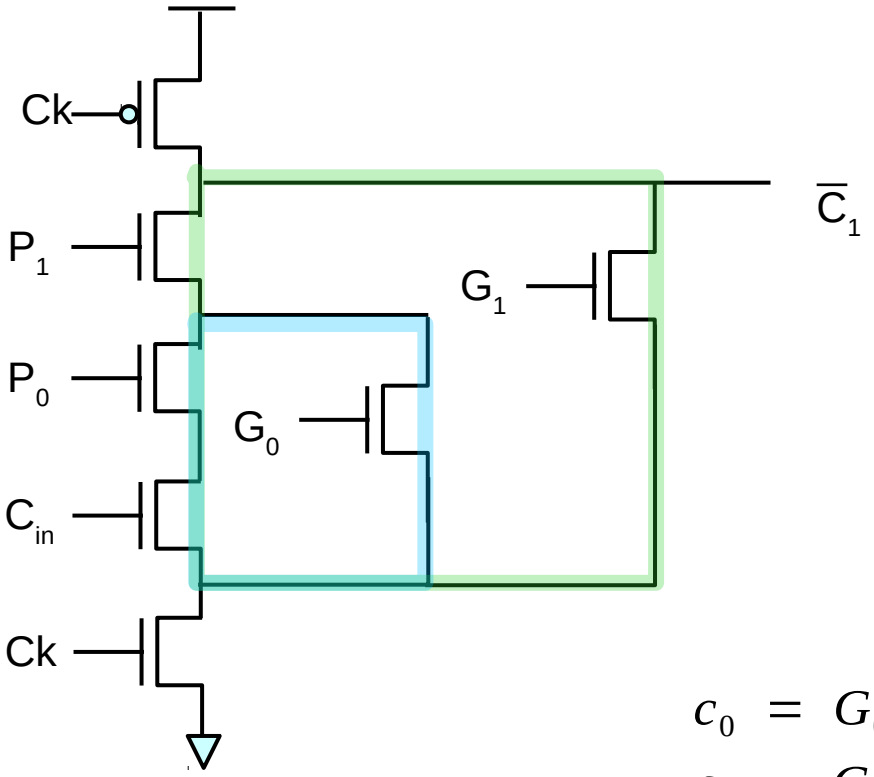
Dynamic Carry Circuit – C_0, C_1, C_2, C_3

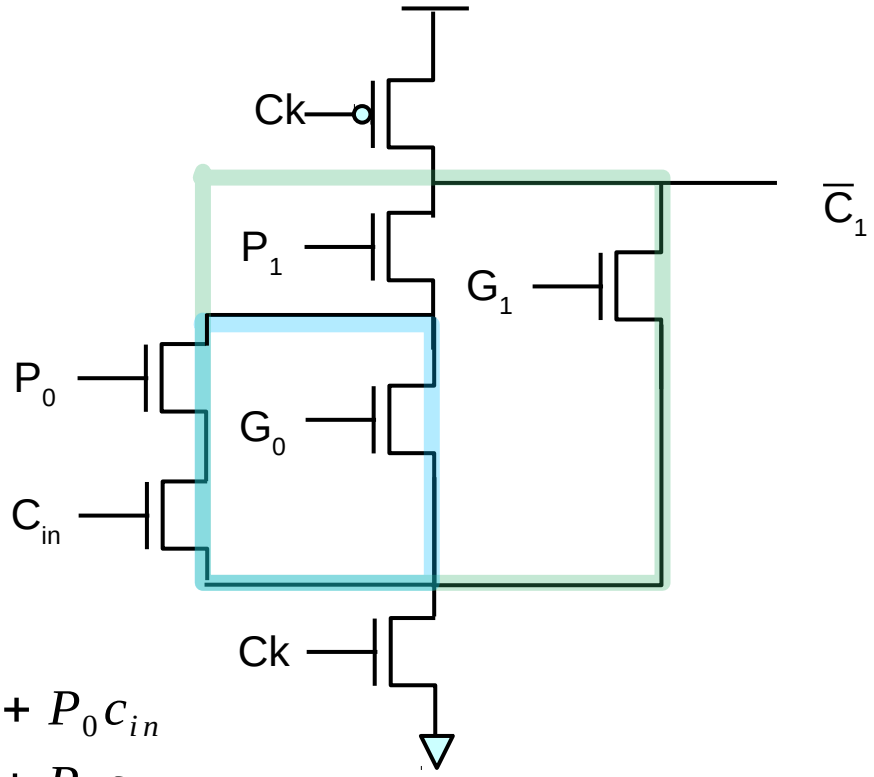


$$c_0 = G_0 + P_0 c_{in} \qquad c_1 = G_1 + P_1 c_0 \qquad c_2 = G_2 + P_2 c_1 \qquad c_3 = G_3 + P_3 c_2$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

Dynamic Carry Circuit – C_1

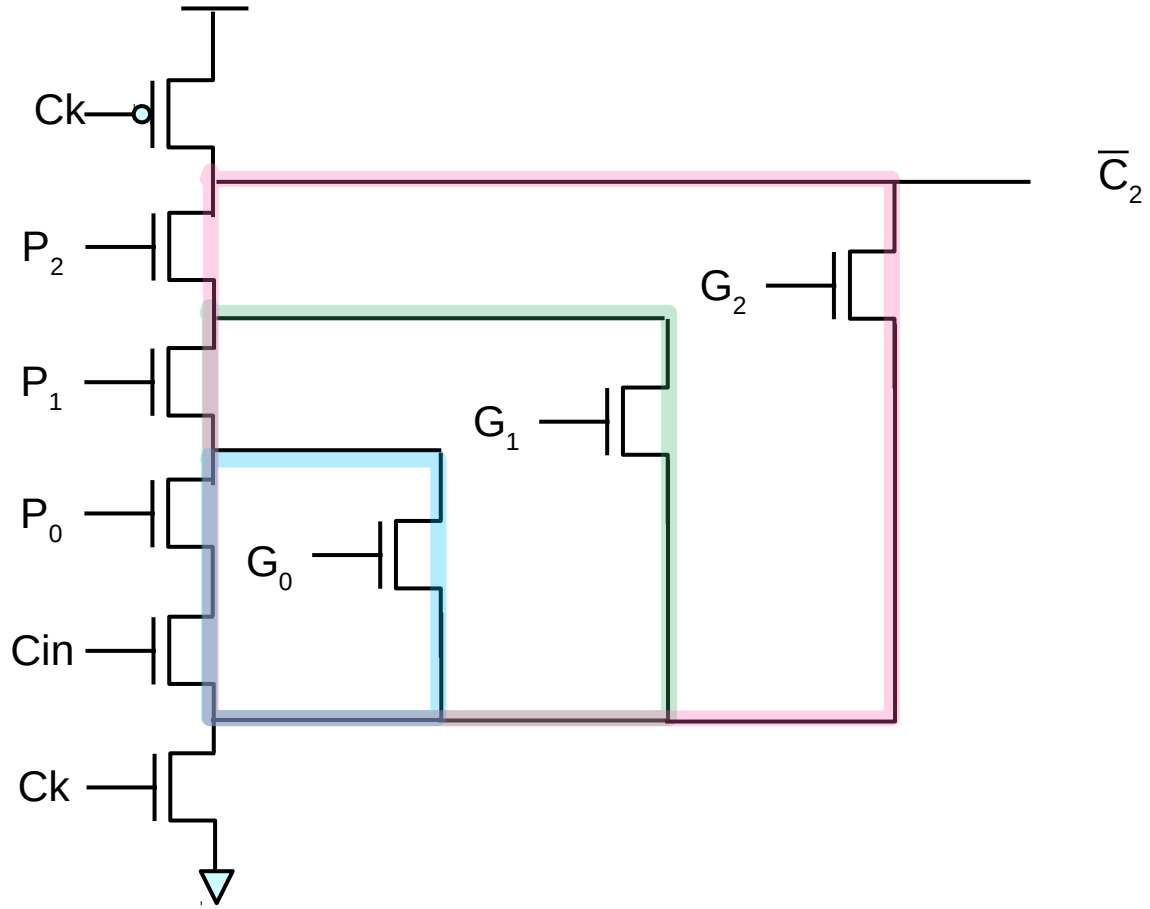




$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 c_0$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

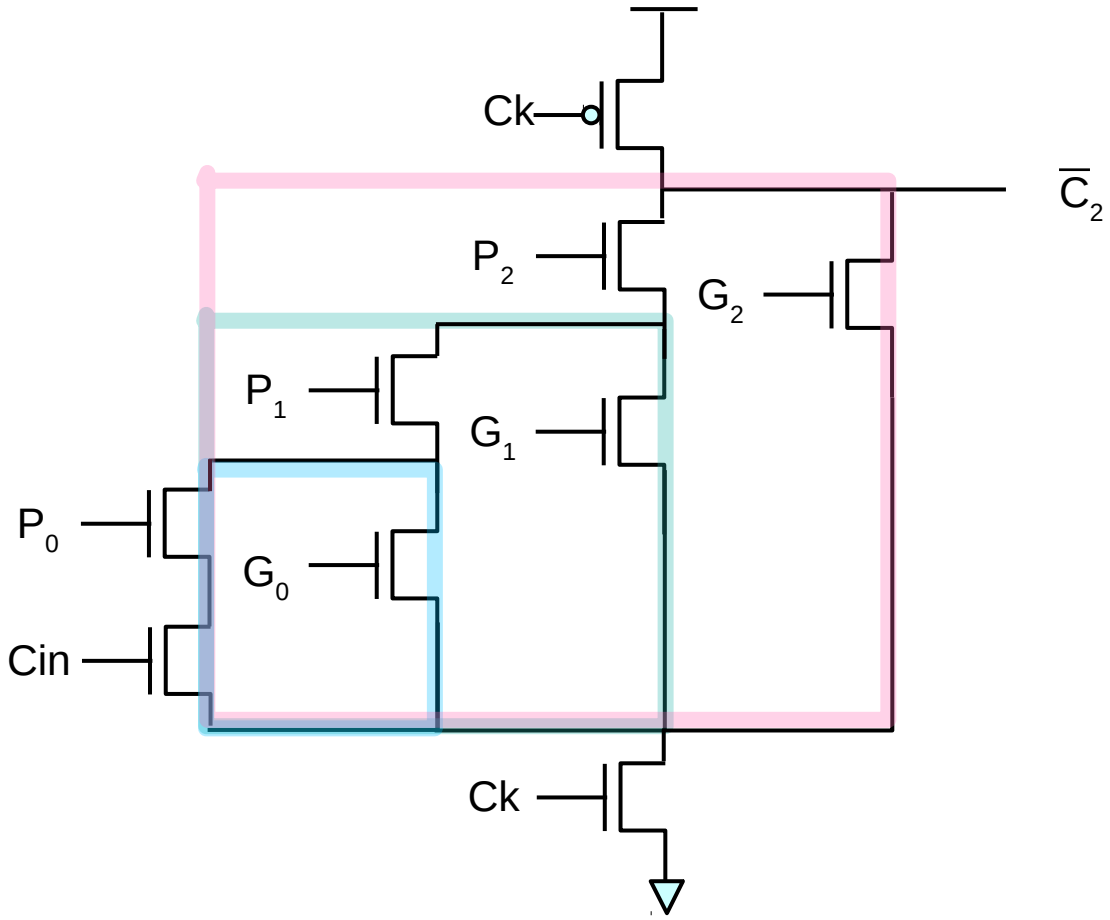
Dynamic Carry Circuit – C_2 (1)



$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 c_0$$
$$c_2 = G_2 + P_2 c_1$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

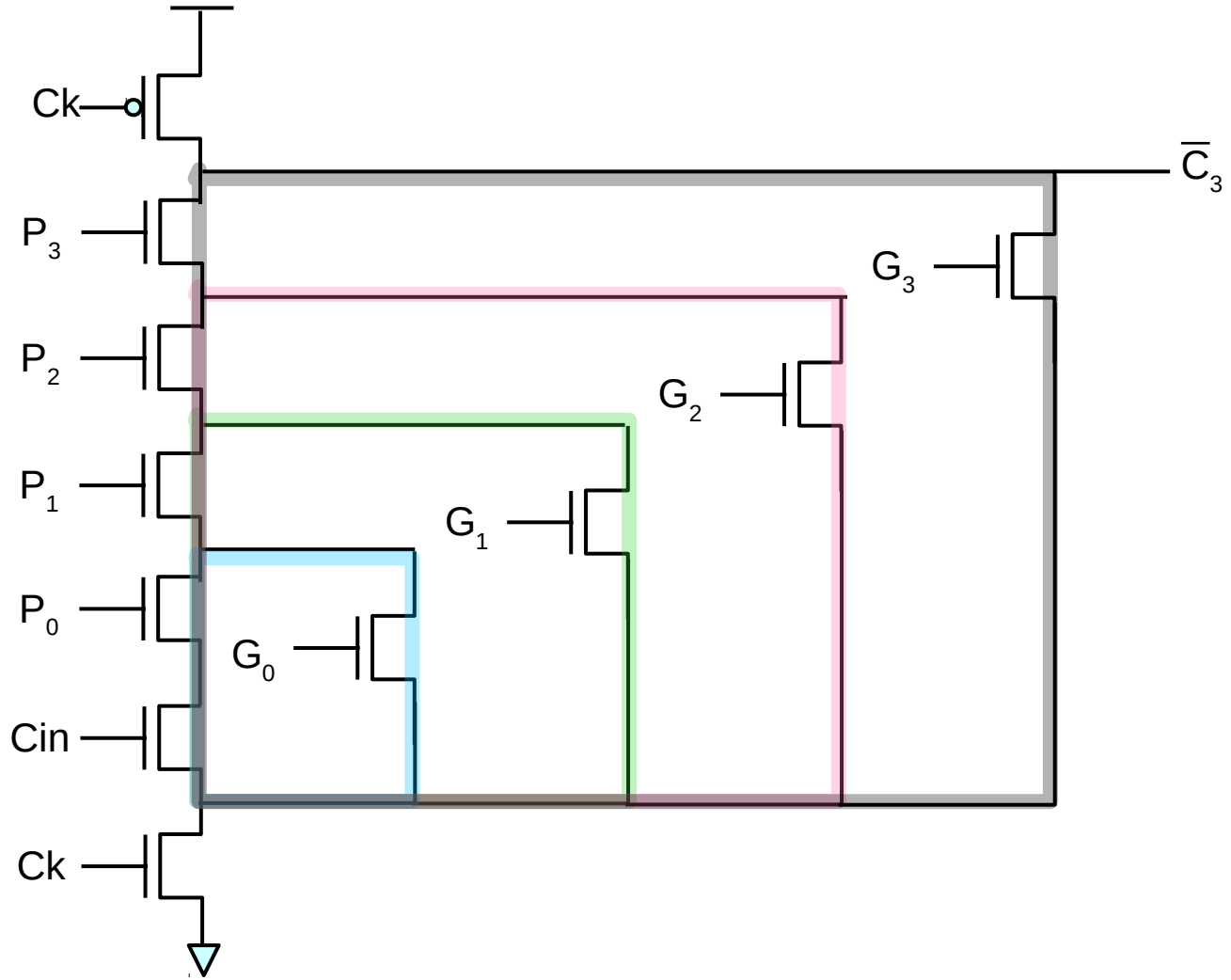
Dynamic Carry Circuit – C_2 (2)



$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 c_0$$
$$c_2 = G_2 + P_2 c_1$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

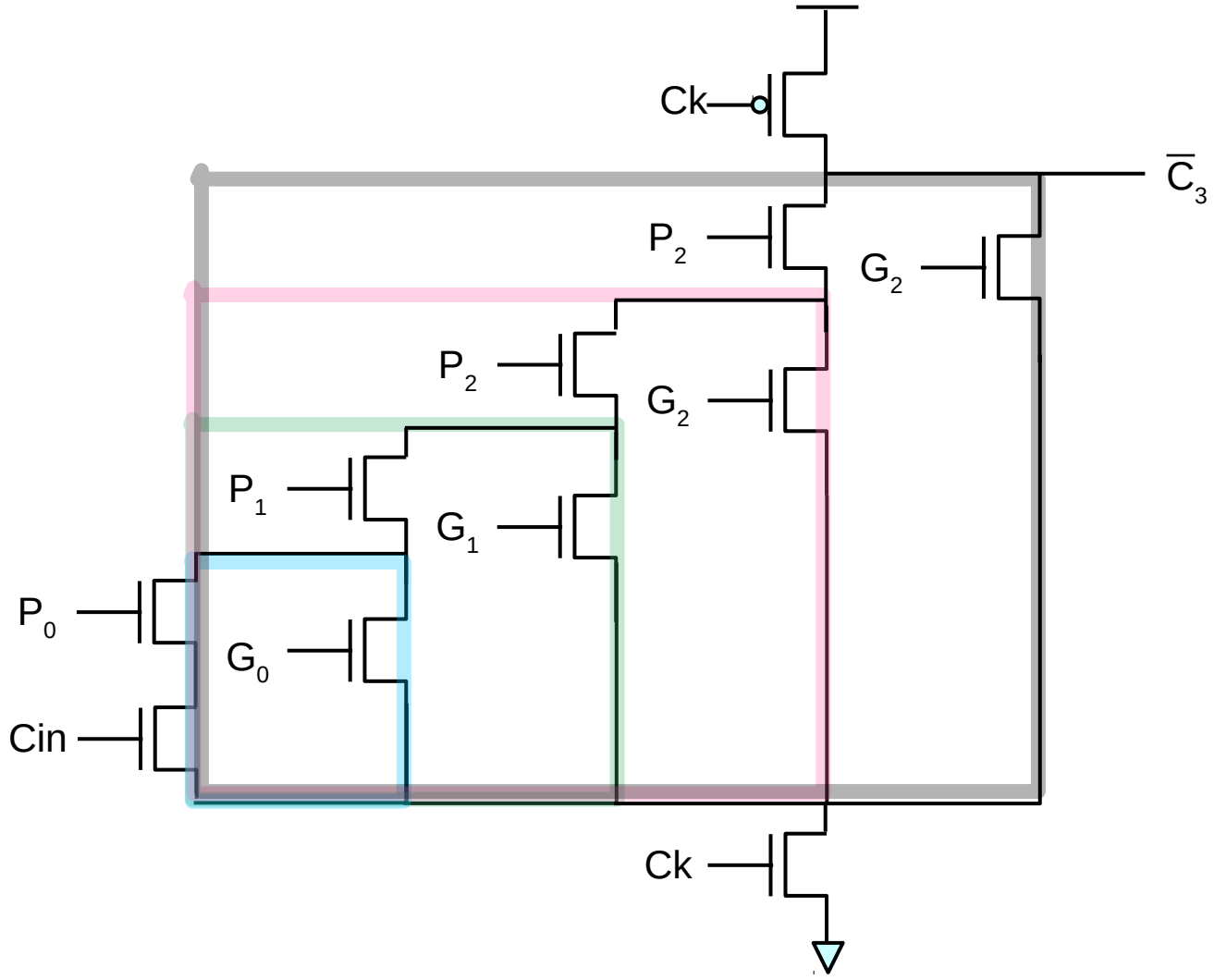
Dynamic Carry Circuit – C_3 (1)



$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 c_0$$
$$c_2 = G_2 + P_2 c_1$$
$$c_3 = G_3 + P_3 c_2$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

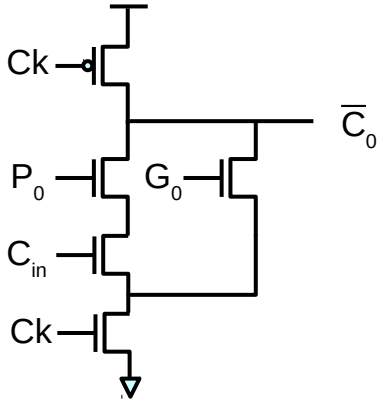
Dynamic Carry Circuit – C_3 (2)



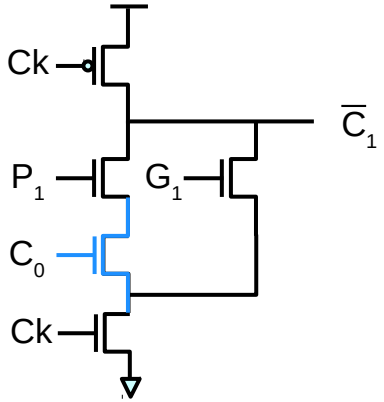
$$c_0 = G_0 + P_0 c_{in}$$
$$c_1 = G_1 + P_1 c_0$$
$$c_2 = G_2 + P_2 c_1$$
$$c_3 = G_3 + P_3 c_2$$

Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

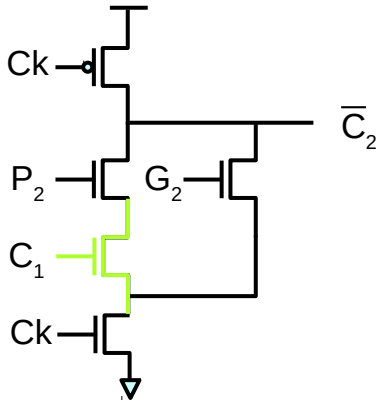
Dynamic Carry Circuit – C_3 (3)



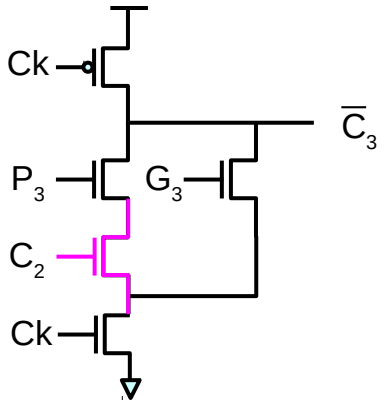
$$c_0 = G_0 + P_0 c_{in}$$



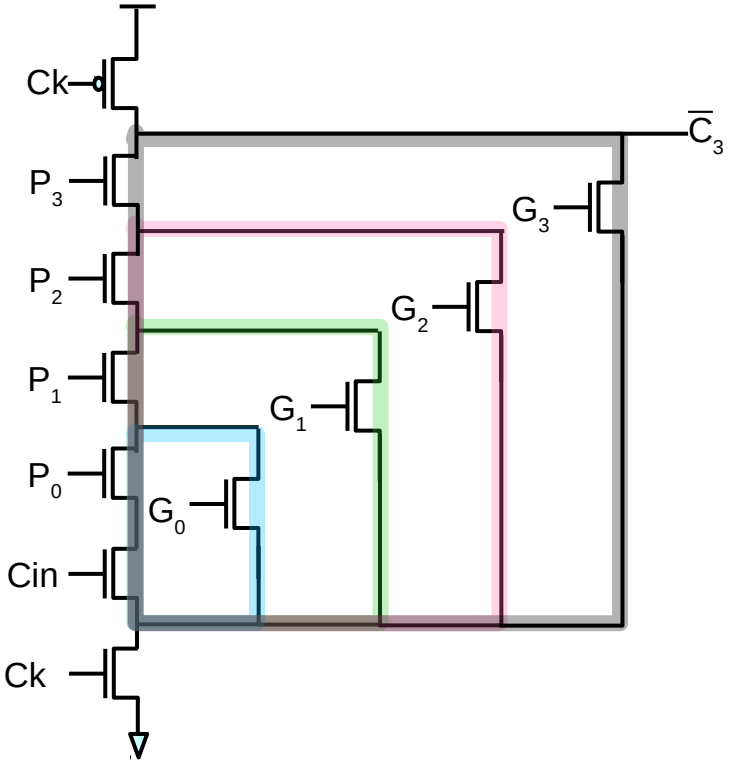
$$c_1 = G_1 + P_1 c_0$$



$$c_2 = G_2 + P_2 c_1$$



$$c_3 = G_3 + P_3 c_2$$



Principles of CMOS VLSI design – A Systems Perspective, N Weste, K Eshraghian

References

[1] http://en.wikipedia.org/
[2] J-P Deschamps,et. al., “Sunthesis of Arithmetic Circuits”, 2006