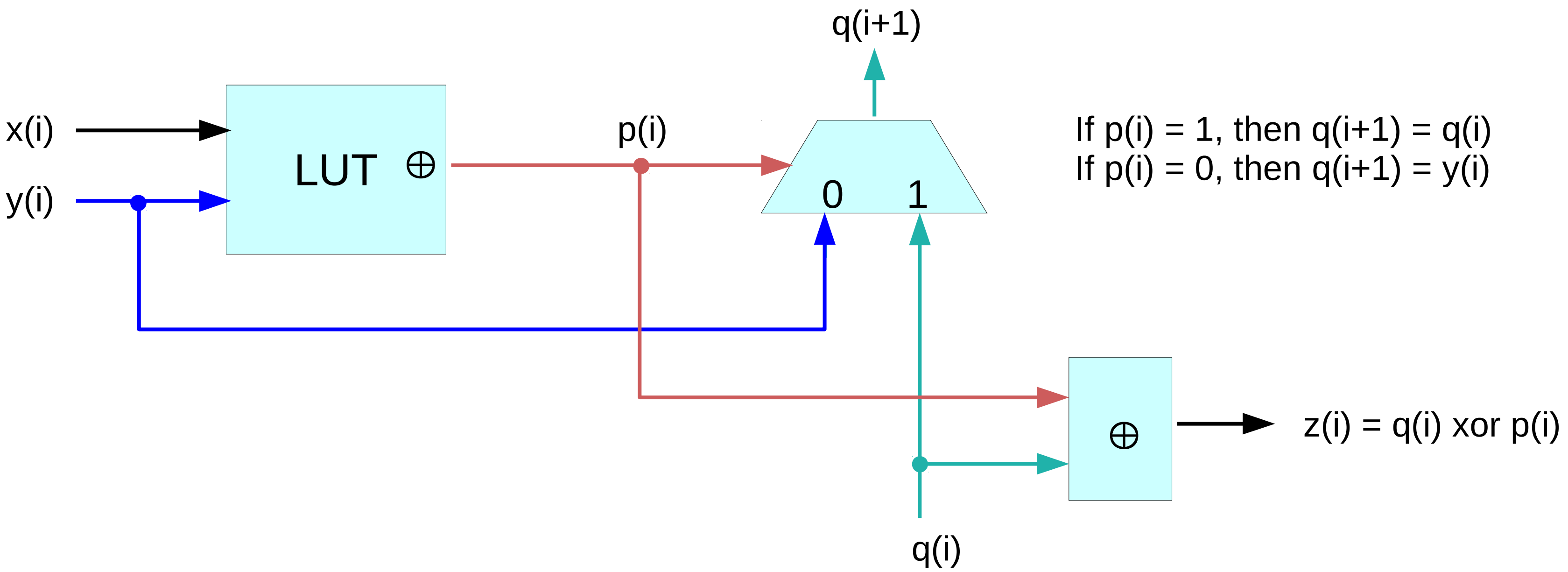
FPGA Carry Chain Adder (1A)

-
-

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

This document was produced by using OpenOffice and Octave.

FPGA Carry Chain Cell



If p(i) = 1, then q(i+1) = q(i)
If p(i) = 0, then q(i+1) = y(i)

$$s_i = (a_i \oplus b_i) \oplus c_i = p_i \oplus c_i$$

$$c_{i+1} = (a_i \cdot b_i) + (a_i \oplus b_i)\, c_i = \overline{p}_i \cdot g_i + p_i \cdot c_i = \overline{p}_i \cdot a_i + p_i \cdot c_i = \overline{p}_i \cdot b_i + p_i \cdot c_i$$

when $\overline{p}_i = 1$, *then* $a_i = b_i$

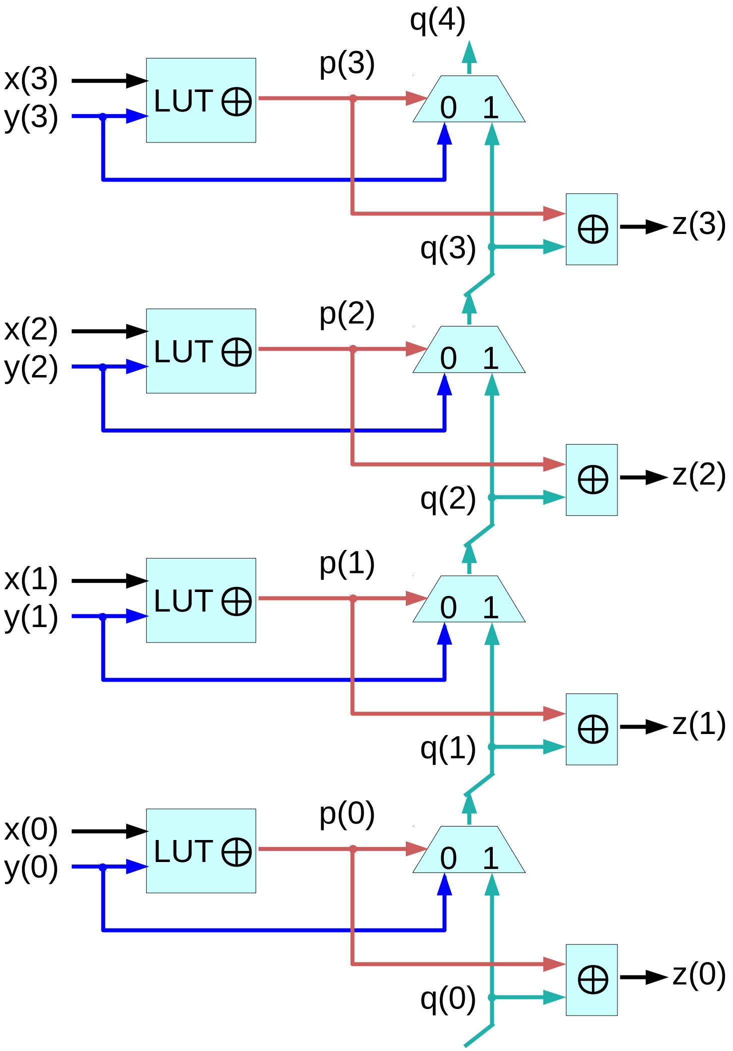
when $g_i = 1$, *then* $a_i = b_i = 1$

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

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

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

FPGA Carry Chain Cell



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

FPGA Carry Chain

FPGAs generally contain dedicated computation resources
for generating fast adders

The Virtex family programmable arrays include
logic gates (**XOR**) and **multiplexers** that along with the
general purpose **lookup tables** allow one to build effective carry-chain adders

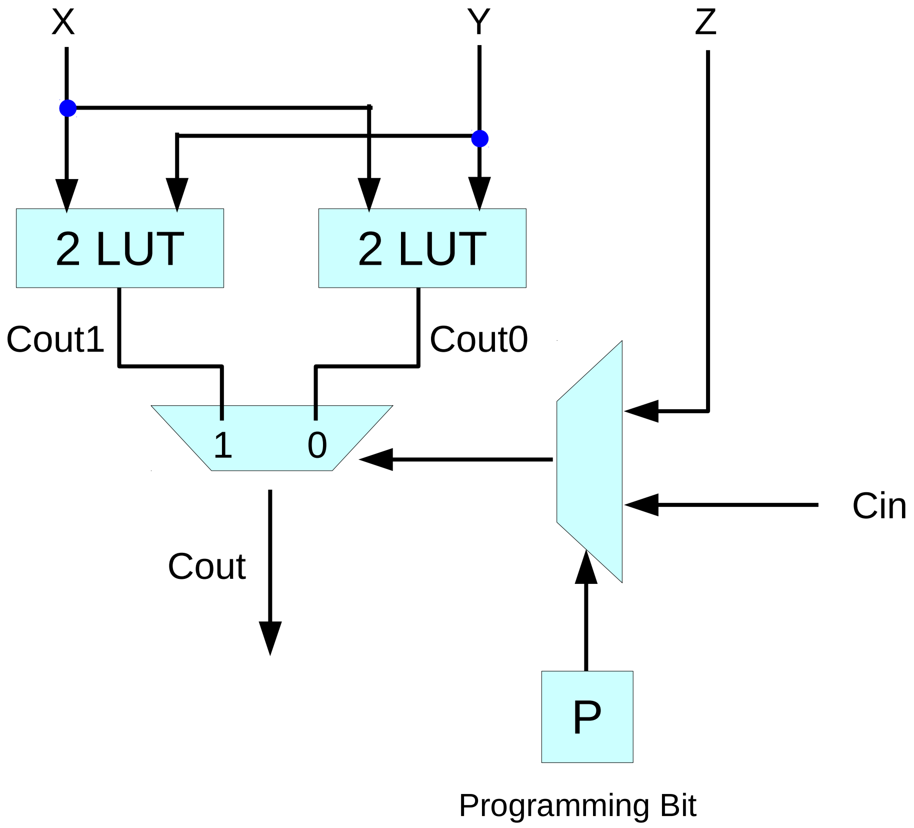
The carry chain is made up of multiplexers
belonging to adjacent configurable blocks

the lookup table is used for implementing the exclusive or function

p(i) = x(i) xor y(i)

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

FPGA Carry Chain Cell



Cout1, Cout2 : functions of X, Y, Cin

Cout1 = X+Y when Cin=1
Cout0 = X Y when Cin=0

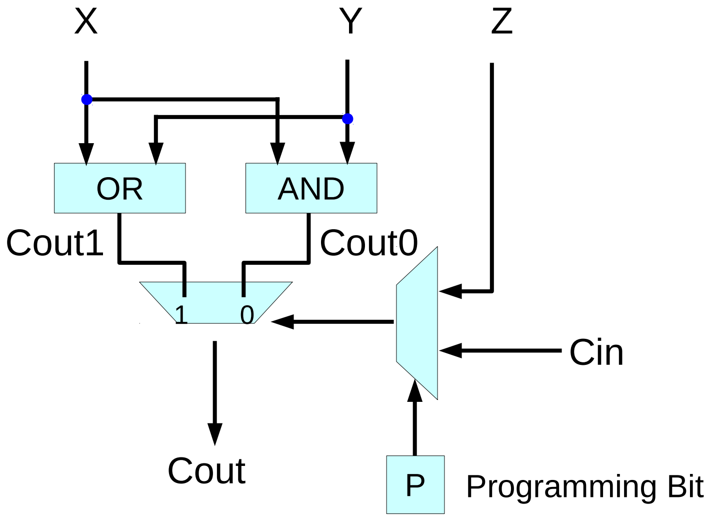
Cout = (X + Y) Cin + X Y $\overline{\text{Cin}}$

Cout = P’ Cin + G $\overline{\text{Cin}}$ … P’ = relaxed P

Cout1	Cout0	Cout	Name
0	0	0	Kill
0	1	$\overline{\text{Cin}}$	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



		Cin	$\overline{Cin}$	
X	Y	Cout1	Cout0	
0	0	0	0	$\overline{X}\ \overline{Y}$
0	1	1	0	$\overline{X}\ Y$
1	0	1	0	$X\ \overline{Y}$
1	1	1	1	$X\ Y$

Cout : functions of X, Y, Cin

Cout(X, Y, 1) = Cout1 = X + Y
Cout(X, Y, 0) = Cout0 = X Y

Cout1 = X + Y when Cin=1
Cout0 = XY when Cin=0

Cout1 = P’ Cin … P’ = relaxed P
Cout0 = G $\overline{\text{Cin}}$

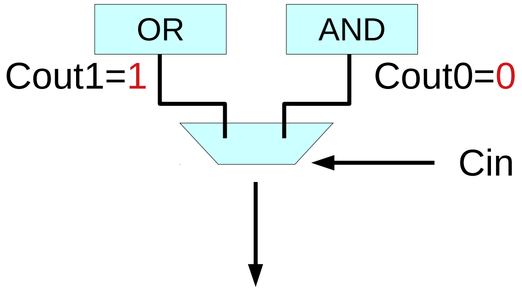
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

If Cin, then Cout = ($\overline{X}$ Y + X $\overline{Y}$ + X Y)
If $\overline{\text{Cin}}$, then Cout = X Y

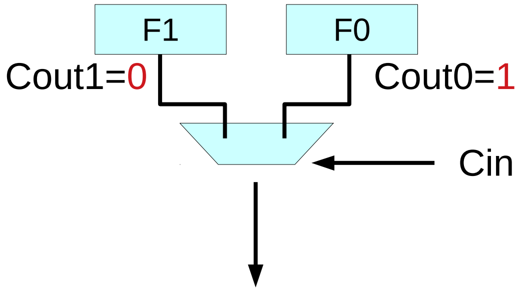
Cin (X + Y) + $\overline{\text{Cin}}$ X Y
Cin ($\overline{X}$ Y + X $\overline{Y}$ + X Y) + $\overline{\text{Cin}}$ X Y
Cin ($\overline{X}$ Y + X $\overline{Y}$) + (Cin + $\overline{\text{Cin}}$) X Y
P Cin + G

Cin (X + Y) + $\overline{\text{Cin}}$ X Y
Cin P’ + $\overline{\text{Cin}}$ G … P’ : relaxed P

FPGA Carry Chain Cell



Cout1=1 when Cin=1
Cout0=0 when Cin=0
Cout = Cin



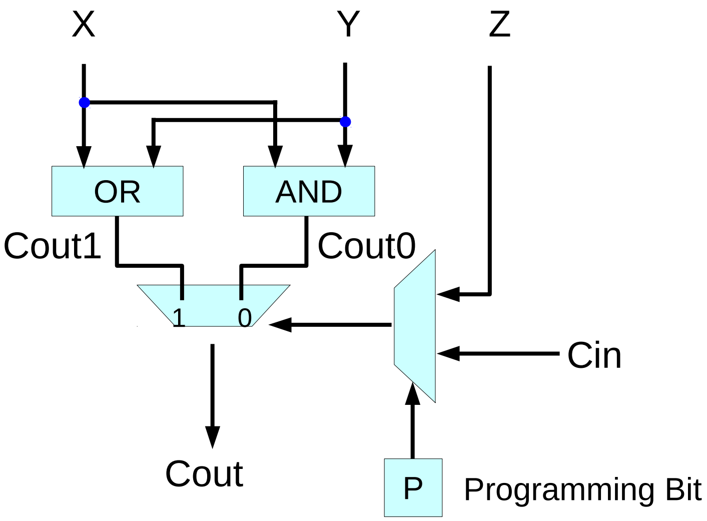
Cout1=0 when Cin=1
Cout0=1 when Cin=0
Cout = $\overline{Cin}$

Cout0	Cout1	Cout	Name
0	0	0	Kill
0	1	Cin	Propagate
1	0	$\overline{Cin}$	Inverse Propagate
1	1	1	Generate

Cout1	Cout0	Cout	Name
0	0	0	Kill
0	1	$\overline{Cin}$	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Carry Chain



		Cin	$\overline{\text{Cin}}$	
X	Y	Cout1	Cout0	
0	0	0	0	$\overline{X}\,\overline{Y}$
0	1	1	0	$\overline{X}\,Y$
1	0	1	0	$X\,\overline{Y}$
1	1	1	1	$X\,Y$

Cout1	Cout0	Cout	Name
0	0	0	Kill
0	1	$\overline{\text{Cin}}$	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

Carry Out

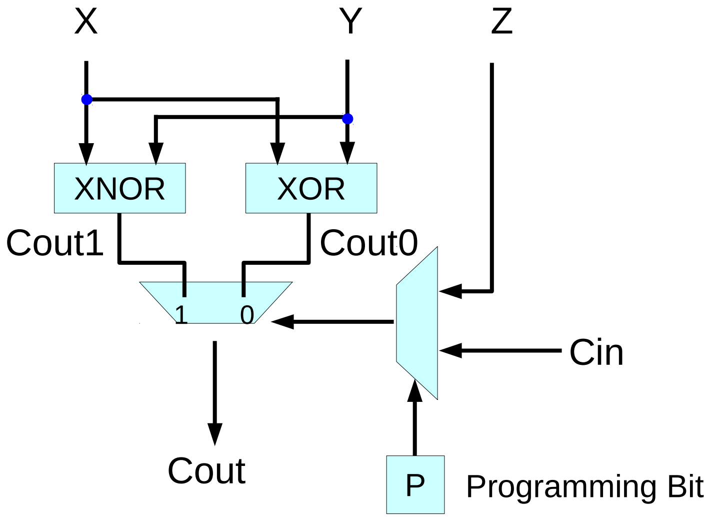
X	Y	Cin	
0	0	Cin	Cin
0	1	Cin	$\overline{\text{Cin}}$
1	0	Cin	$\overline{\text{Cin}}$
1	1	Cin	Cin

Cout1=1 when Cin=1
Cout0=0 when Cin=0
Cout = Cin propage

Cout1=0 when Cin=1
Cout0=1 when Cin=0
Cout = $\overline{\text{Cin}}$ inverse propagate

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Parity Checker



		Cin	$\overline{Cin}$	
X	Y	Cout1	Cout0	
0	0	1	0	$\overline{X}\ \overline{Y}$
0	1	0	1	$\overline{X}\ Y$
1	0	0	1	$X\ \overline{Y}$
1	1	1	0	$X\ Y$

Cout1	Cout0	Cout	Name
0	0	0	Kill
0	1	$\overline{Cin}$	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

Computing Parity

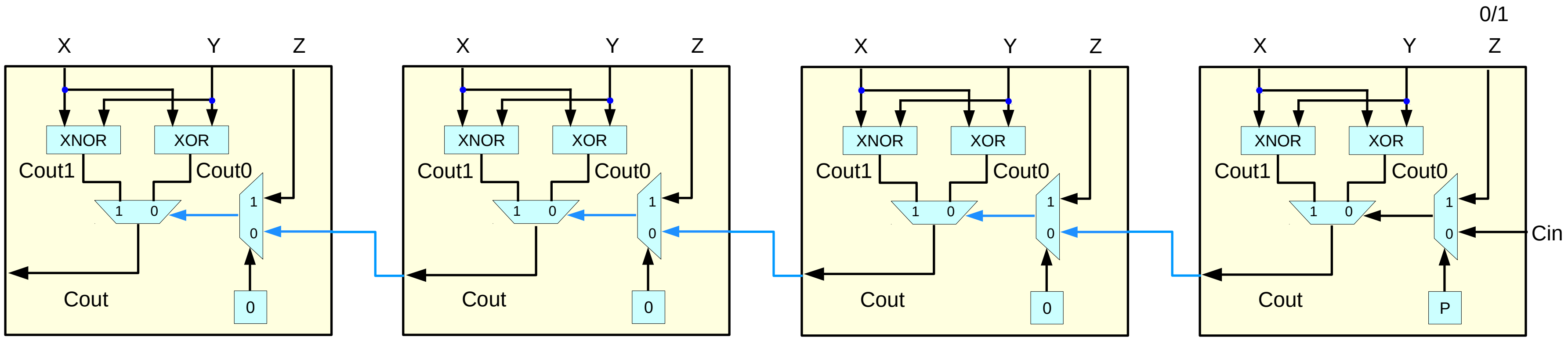
X ⊕ Y ⊕ Cin	
0 ⊕ 0 ⊕ Cin	Cin
0 ⊕ 1 ⊕ Cin	$\overline{Cin}$
1 ⊕ 0 ⊕ Cin	$\overline{Cin}$
1 ⊕ 1 ⊕ Cin	Cin

Cout1=1 when Cin=1
Cout0=0 when Cin=0
Cout = Cin propagate

Cout1=0 when Cin=1
Cout0=1 when Cin=0
Cout = $\overline{Cin}$ inverse propagate

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

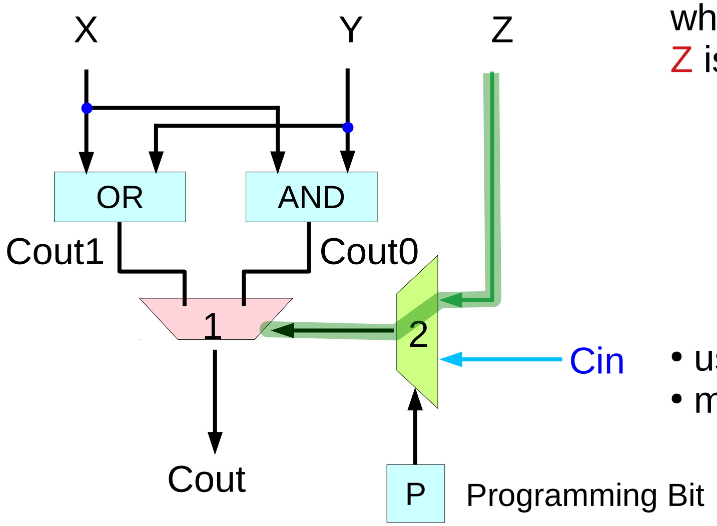
Ripple Carry Chain



the **first cell** in the chain

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



when Cin is ignored,
Z is routed to mux1

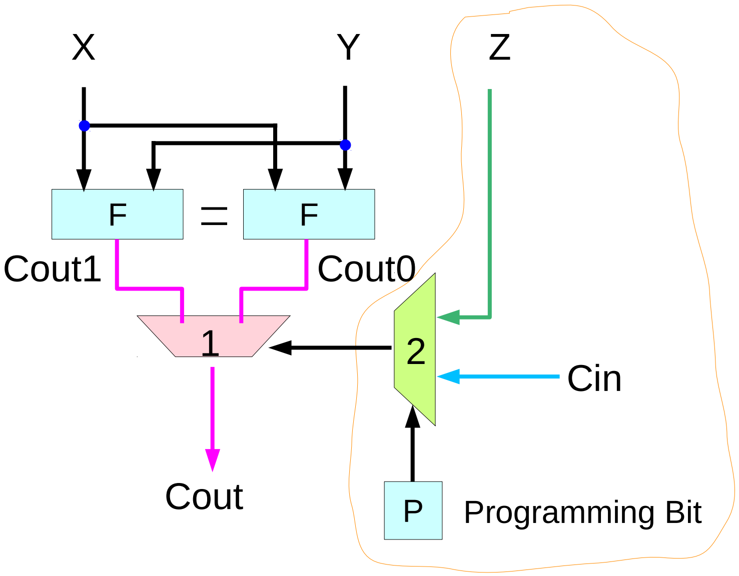
- used in combined adder/subtractors
- must be ignored, otherwise

the logic cells - resources to compute a function
the exact location of logic cells depends on the user.
a user can start or end a carry computation
at any place in an fpga.

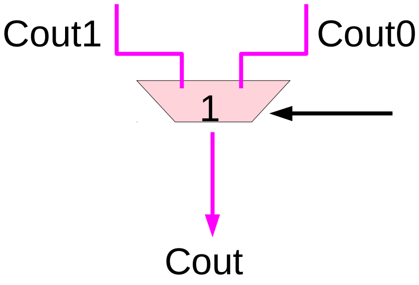
But in many carry computations,
the first cell has only 2 inputs,
and forcing the carry chain
to wait for the arrival of an additional,
unnecessary input Z will only needlessly
slow down the circuit's computation.

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



when Cin is ignored,
Z can also be ignored
by having the same LUTs



the **first cell** in the chain

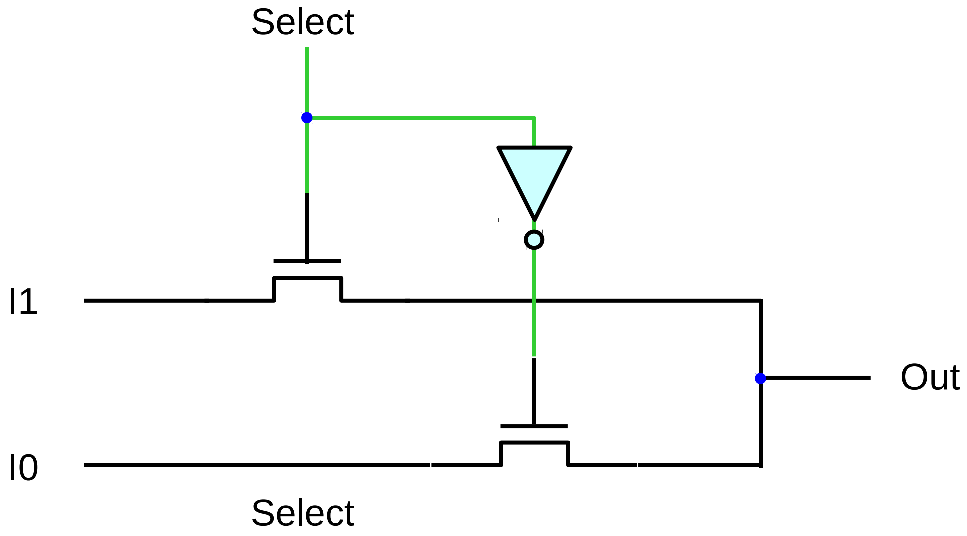
the same LUTs

the same output
regardless of Z and Cin

Cout1 = Cout0 = Cout
regardless of the select

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Ripple Carry Chain



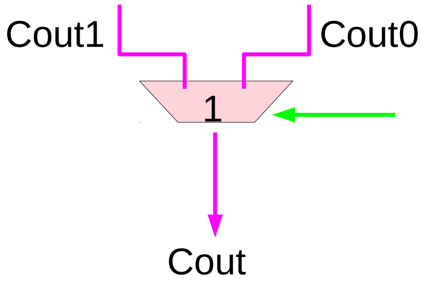


fig1b shows an implementation of a mux that does not obey this requirement

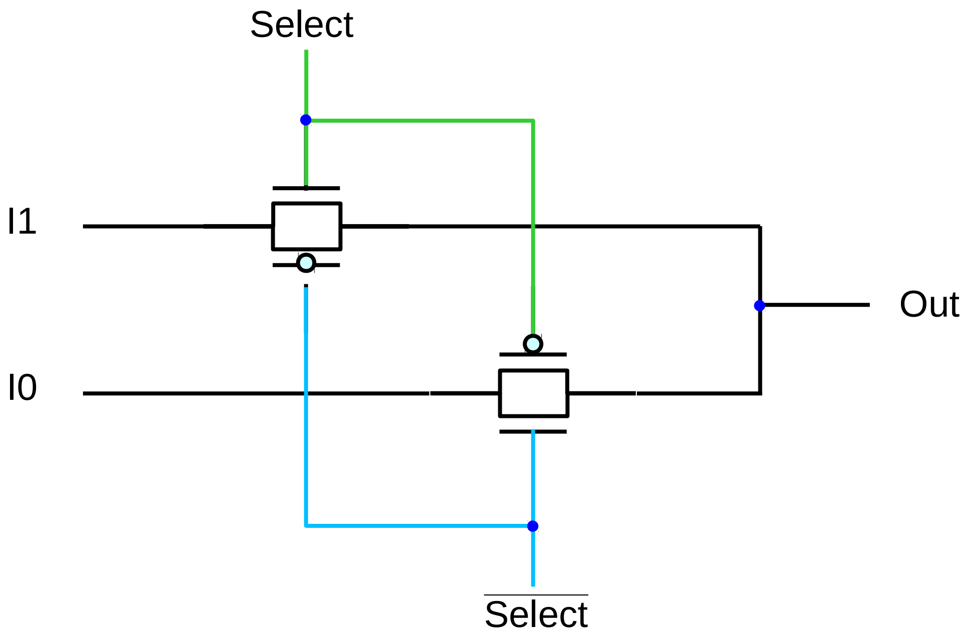
since the carry chain is part of an fpga,
the input to this mux could be connected to some unused logic in another row which is generating unknown values.

if that unused logic had multiple transitions which caused the signal to change quicker than the gate could react,
then it is possible that **the select signal** to this mux could be stuck midway between true and false (2.5V for 5V CMOS)

in this case, it will not be able to pass a true value from the input to the output
and thus will not function properly for this application.

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Ripple Carry Chain



however a mux built with
both n-transistor and p-transistor pass gates
will operate properly for this case

assume this mux implementation will be used

tristate driver based muxes could be used,
which restore signal drive and cut series RC chains

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Unit Gate Delay Model

All simple gate of two or three inputs
that are directly implementable
in one logic level in CMOS
are considered to have a delay of one.

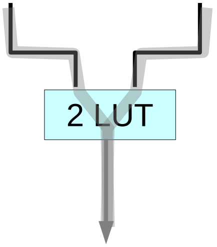
All other gate must be implemented by such gates,
and have the delay of the underlying circuit.

Delay of one

- inverters and
- 2 to 3 input NAND
- 2 to 3 input NOR gates

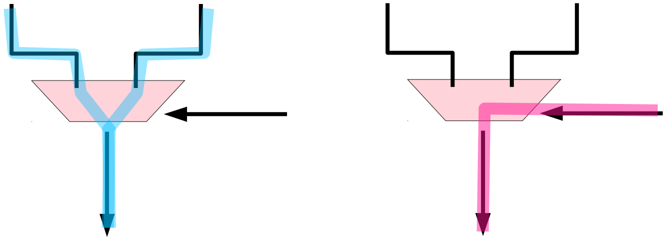
Delay of zero (constant delay)

- the delay of the 2-LUTs,
- any routing leading to them,



Delay of **0**

A 2:1 mux has a delay of one
from the I0 or I1 inputs to the output,
But has a delay of two
from the select input to the output
due to the Inverter delay

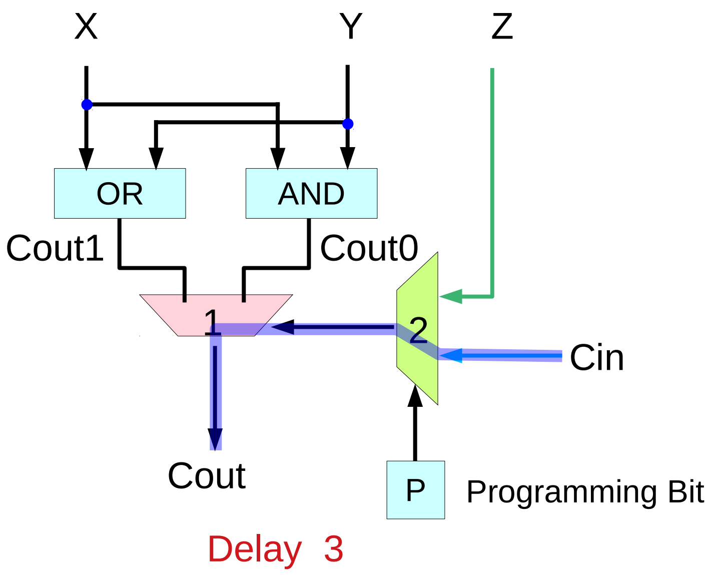


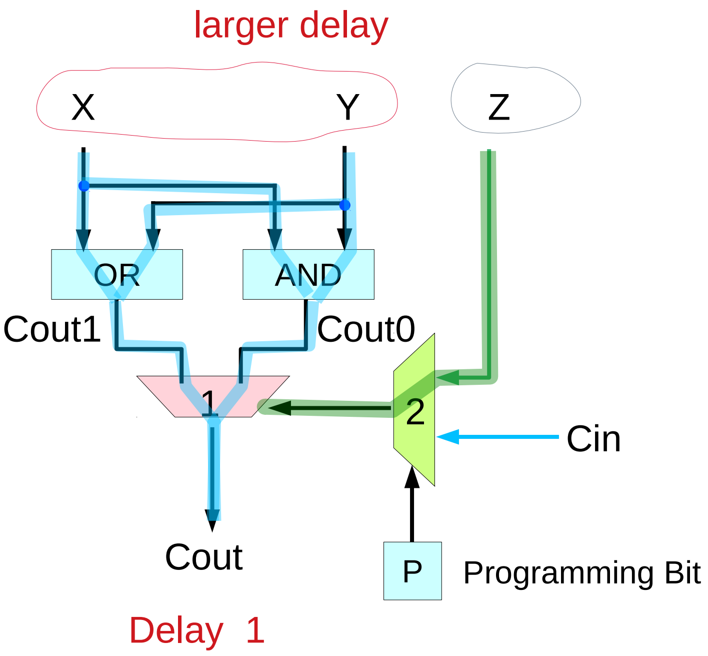
Delay of **1**

Delay of **2**

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell





Significantly slower
two muxes on the carry chain in each cell

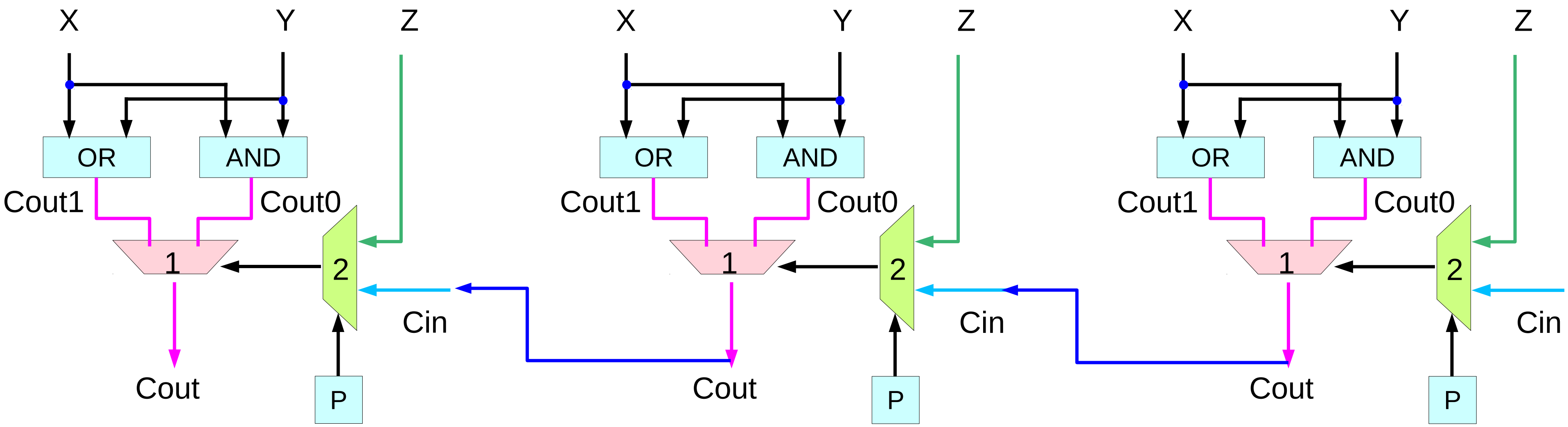
Delay 1 for first cell
Delay 3 for each additional cell in the carry chain
delay 1 for mux2
delays 2 for mux1

Overall 3n-2 for an n-cell carry chain

The critical path comes from the 2-LUTs
and not from the input Z
since the delay through the 2-LUTs
will be larger than through mux 2 in the first cell

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



delay of 3

delay of 3

delay of 1

delay of 3n-2 for an n-bit ripple carry chain

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell

to optimize a ripple carry chain structure for use in FPGAs
while this provides some performance gain over the basis ripple carry scheme
found in many current FPGAs, it is still much slower than what is done in custom logic
There have been tremendous amounts of work done on developing alternative
carry chain scheme that overcome the linear delay growth of ripple carry adders
Although these techniques have not yet been applied to FPGAs,
demonstrate how these advanced adder techniques can be integrated into reconfigurable logic

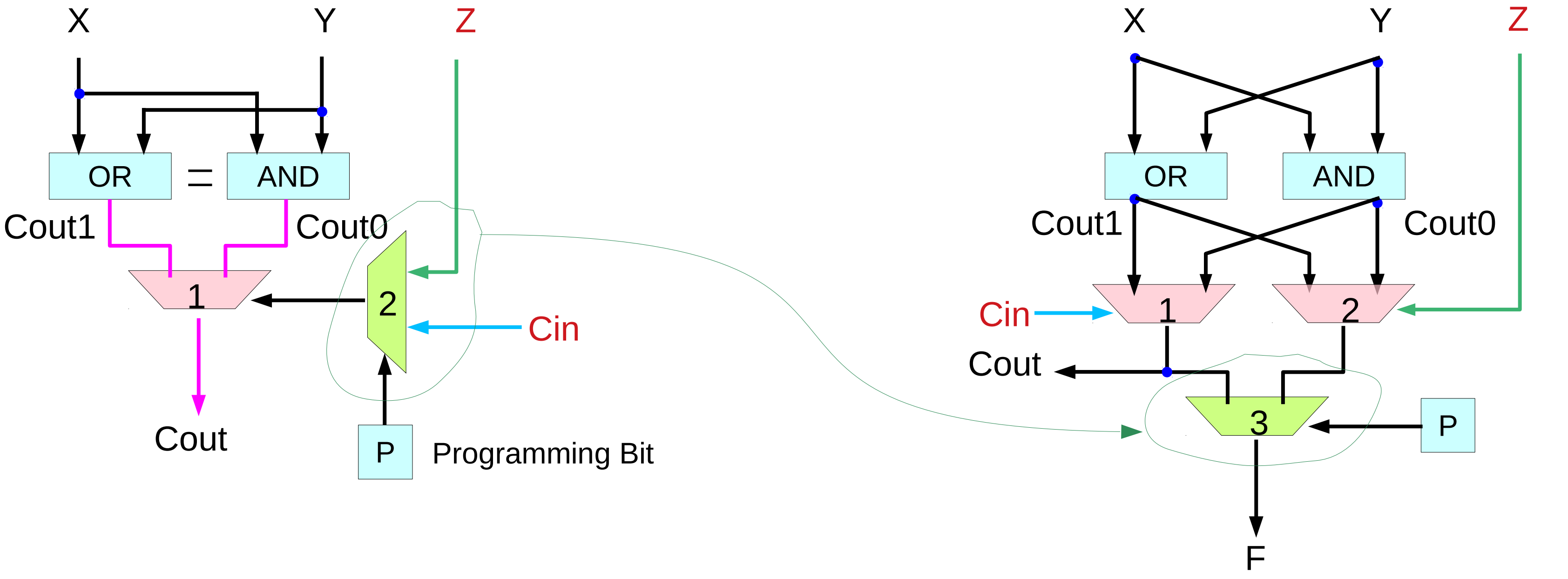
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design A

to reduce the delay of the ripple carry chain

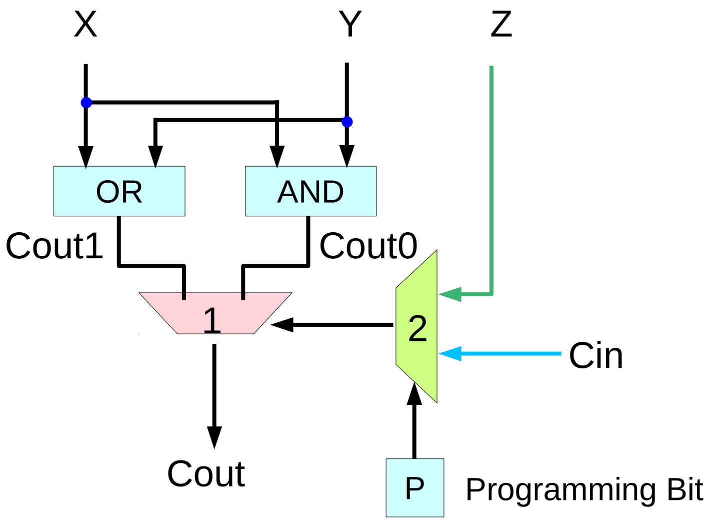
- remove mux2 from the carry path.
- no need to choose between Cin and Z for the select line to the output mux1

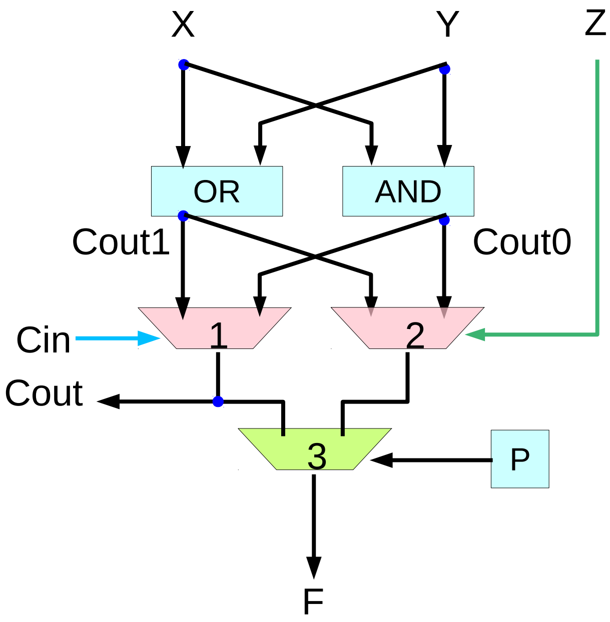
- two separate muxes, mux1 and mux2, controlled by Cin and Z, respectively.
- the circuit chooses between these outputs with mux3.



High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design A





- not logically equivalent
- no longer use the Z input in the first cell since Z is only attached to mux2 and mux2 does not lead to the carry cells
- no longer use the Cin input in the first cell since Cin is only attached to mux1 and mux1 does not lead to the carry cells

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

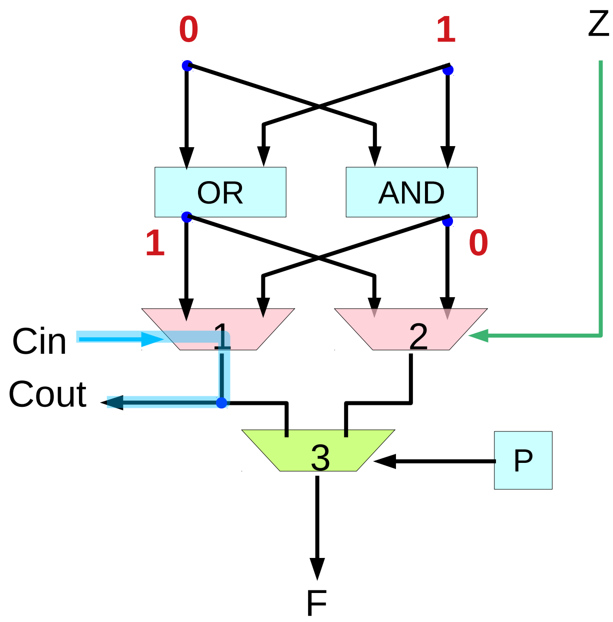
Design A

on the other hand, in order to implement
a n-bit carry chain with a carry input,

the design of fig 2a
requires an additional cell
at the beginning of the chain
to bring in this input,
resulting in a delay of 2(n+1)=2n+2,

which is lower than that of the design in fig2b
thus, the design of fig 2b is the preferred ripple carry
design among those presented so far

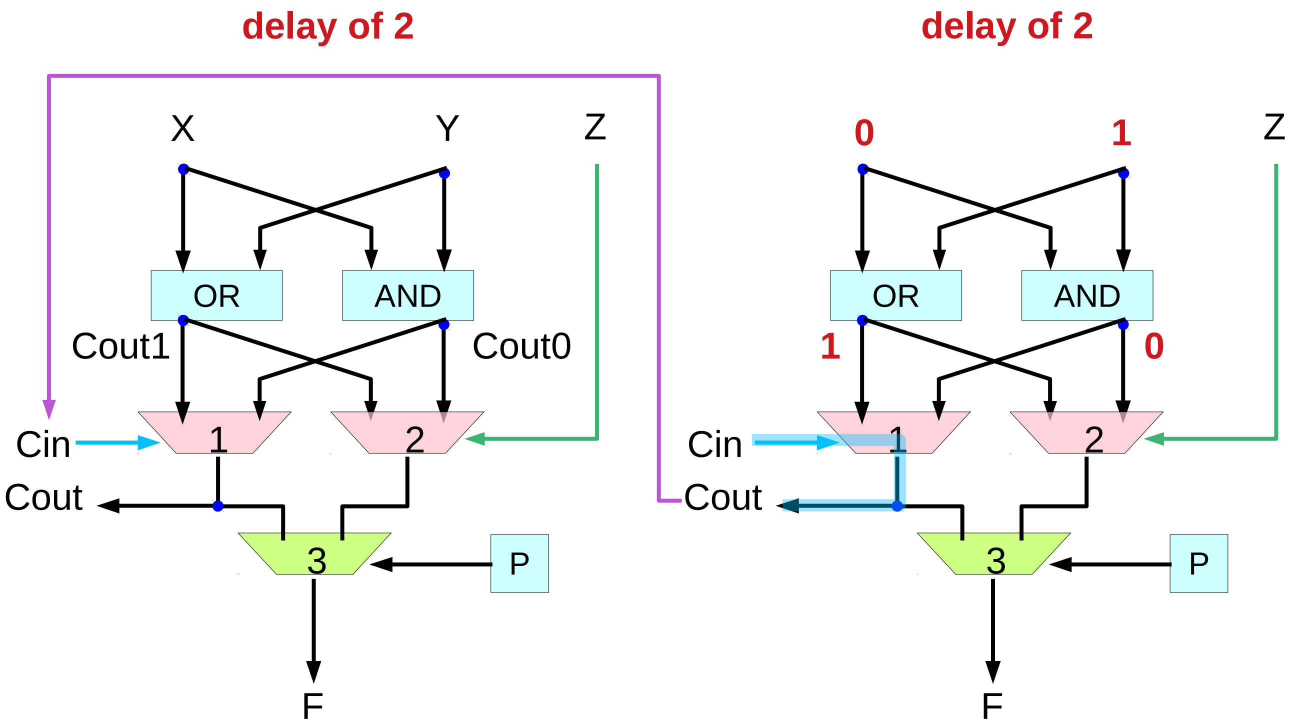
delay of 2



an additional cell
for generating Cin

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

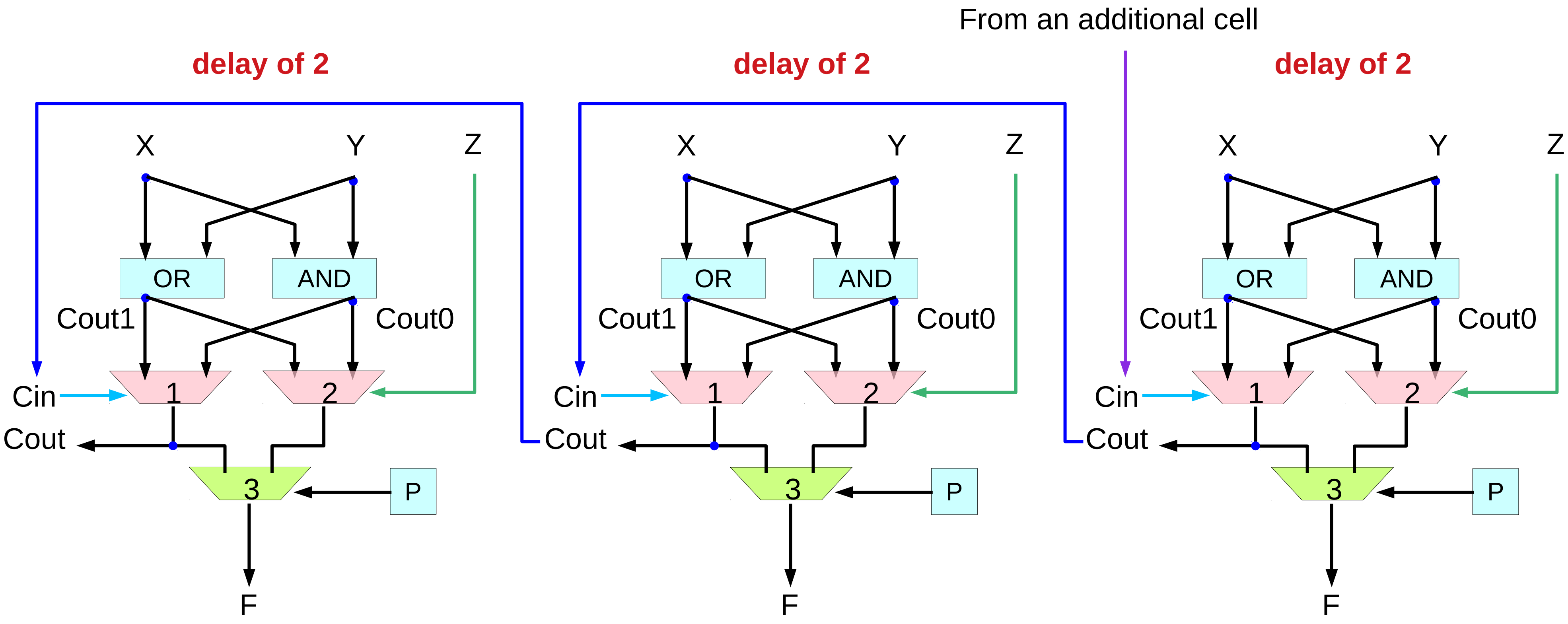
Design A



an additional cell
for generating Cin

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design A



(2 for mux1, 1 for mux3)

delay of 2(n+1) for an n-bit ripple carry chain

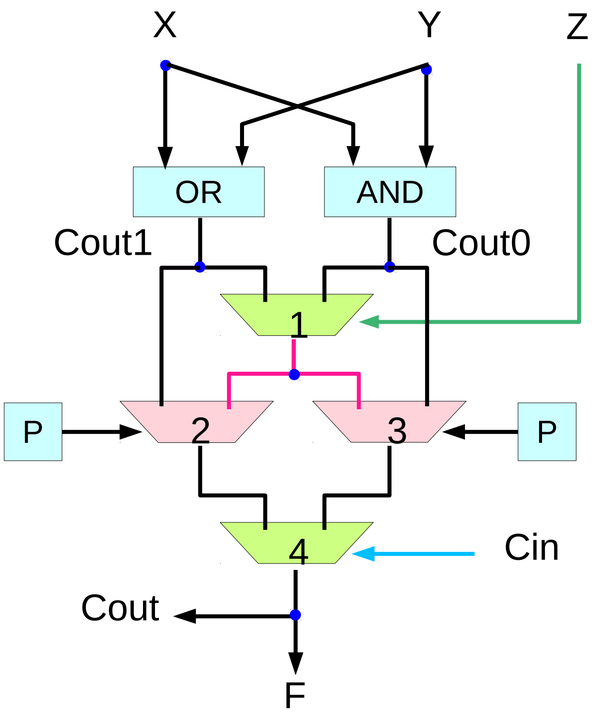
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design B

thus, although this design is
1 gate delay slower than that of fig 2a,
it provides the ability to have a carry input
to the first cell in a carry chain,
something that is important in many computations.

Also, for carry computations
that do not need this feature, the first cell in a carry
chain built from fig 2b can be configured
to bypass mux1, reducing the overall delay to 2n,
which is identical to that of fig2a.

on the other hand, in order to implement
a n-bit carry chain with a carry input,
the design of fig 2a requires an additional cell
at the beginning of the chain to bring in this input,
resulting in a delay of 2(n+1)=2n+2,
which is lower than that of the design in fig2b
thus, the design of fig 2b is the preferred
ripple carry design among those presented so far

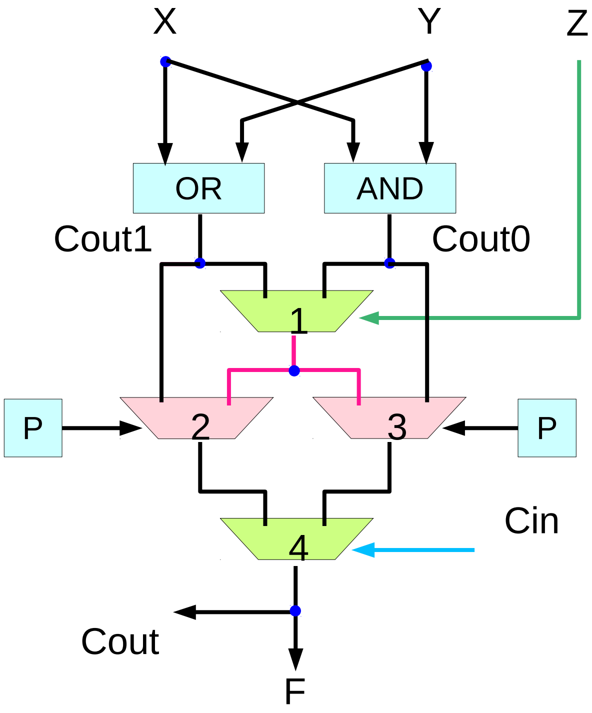


High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design B

although this design is 1 gate delay slower than that of fig 2a,
it provides the ability to have a carry input
to the first cell in a carry chain,
something that is important in many computations.

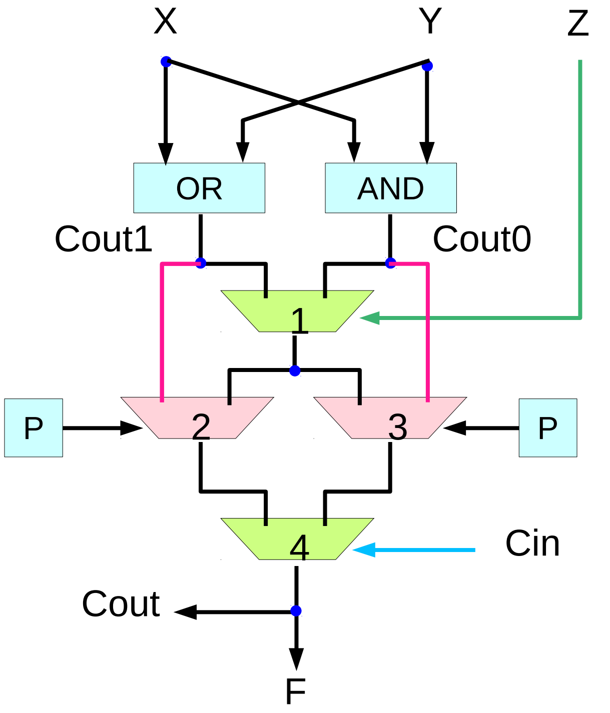
Also, for carry computations that do not need this feature,
without a carry input
the first cell in a carry chain built from fig 2b
can be configured to bypass mux1,
reducing the overall delay to 2n,
which is identical to that of fig2a.



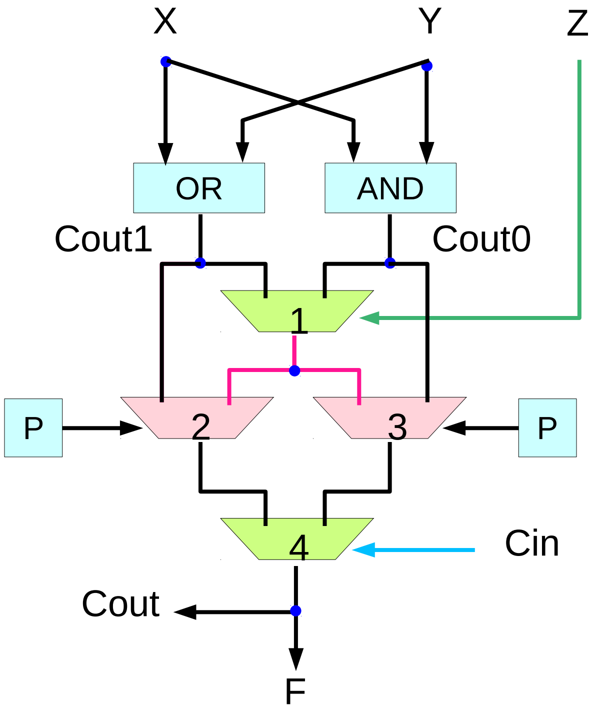
(2 for mux1, 1 for mux3)

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design B



for cells in the middle of a carry chain
mux2 passes Cout1
mux3 passes Cout0
mux4 receives Cout1 and Cout0
provides a standard ripple carry path.

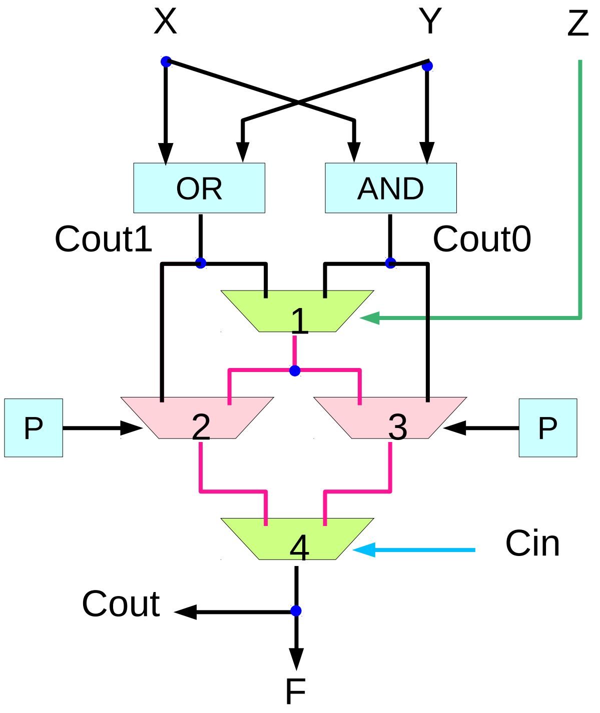


For the first cell in a carry chain
with a carry input (provided by input Z),
mux2 and mux3 both pass the value from mux1

the two main inputs to mux4 are identical
the output of mux4 (Cout) will be the same
as the output of mux1 (ignoring Cin)

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

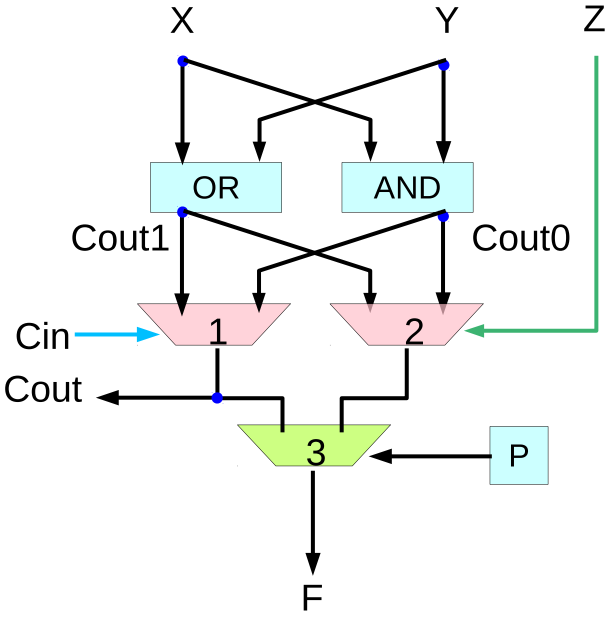
Design B



mux1's main inputs are driven
by two 2-LUTs (OR, AND) controlled by X and Y
mux1 forms a 3-LUT with the other 2-LUTs

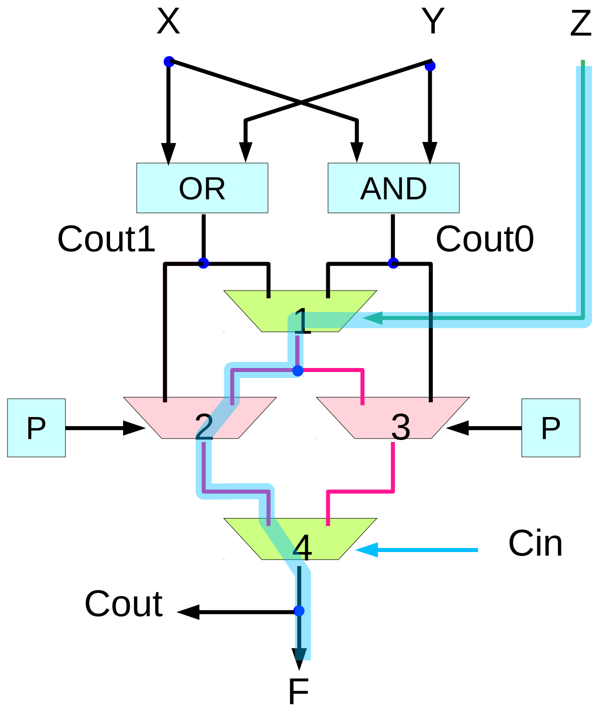
When mux2 and mux3 pass the value from mux1
(Cout1 and Cout2 respectively)
the circuit is configured to continue the carry chain

Functionally equivalent



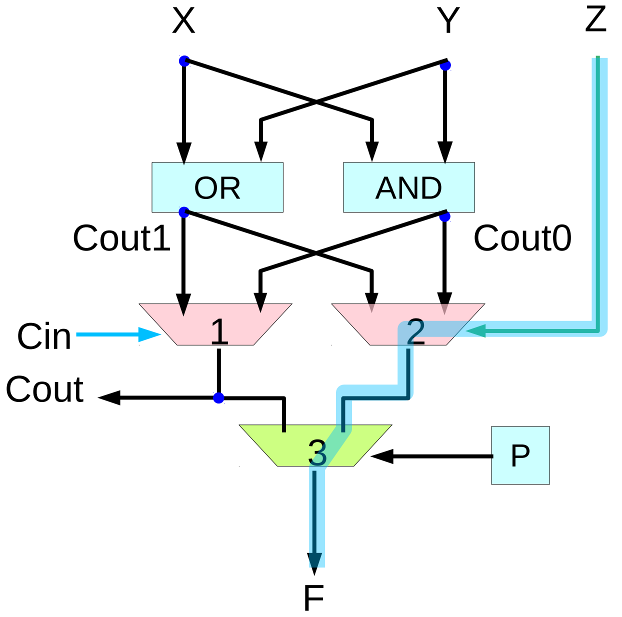
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design B



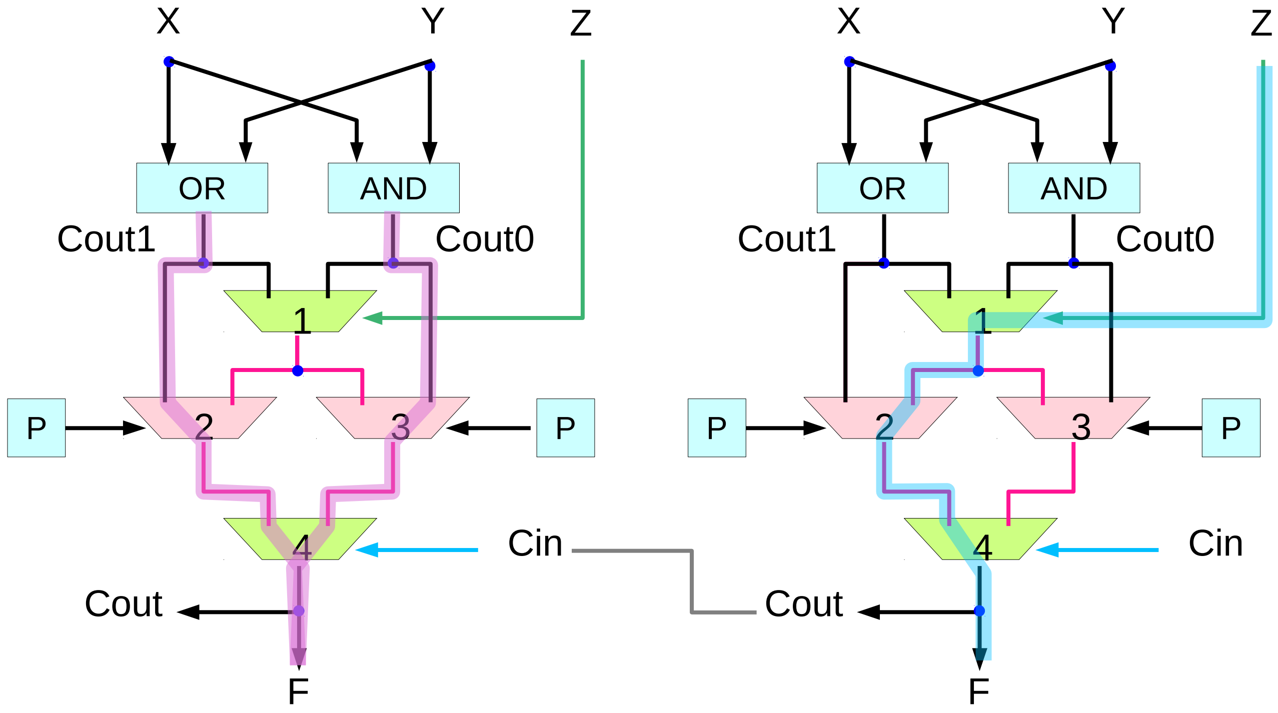
a delay of 3 in the first cell
(1 in mux1, 1 in mux2, 1 in mux4)
2 in all other cells in the carry chain
an total delay of 2n+1 for an n-bit carry chain

1 gate delay slower than that of fig 2a,
a carry input to the first cell is enabled



High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

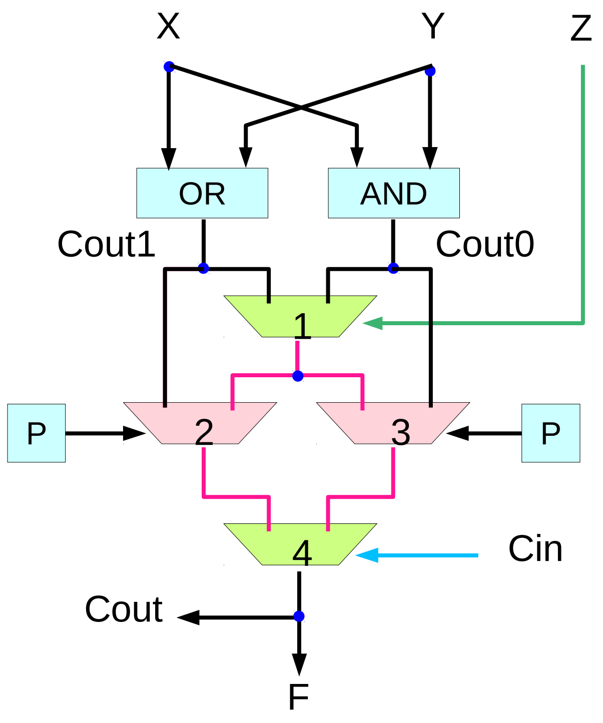
Design B



Also, for carry computations that do not need this feature,
the first cell in a carry chain built from fig 2b
can be configured to bypass mux1,
reducing the overall delay to 2n,
which is identical to that of fig2a.

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design B



a delay of 3 in the first cell
(1 in mux1, 1 in mux2, 1 in mux4)
2 in all other cells in the carry chain
an total delay of 2n+1 for an n-bit carry chain

t1 gate delay slower than that of fig 2a,
a carry input to the first cell is enabled

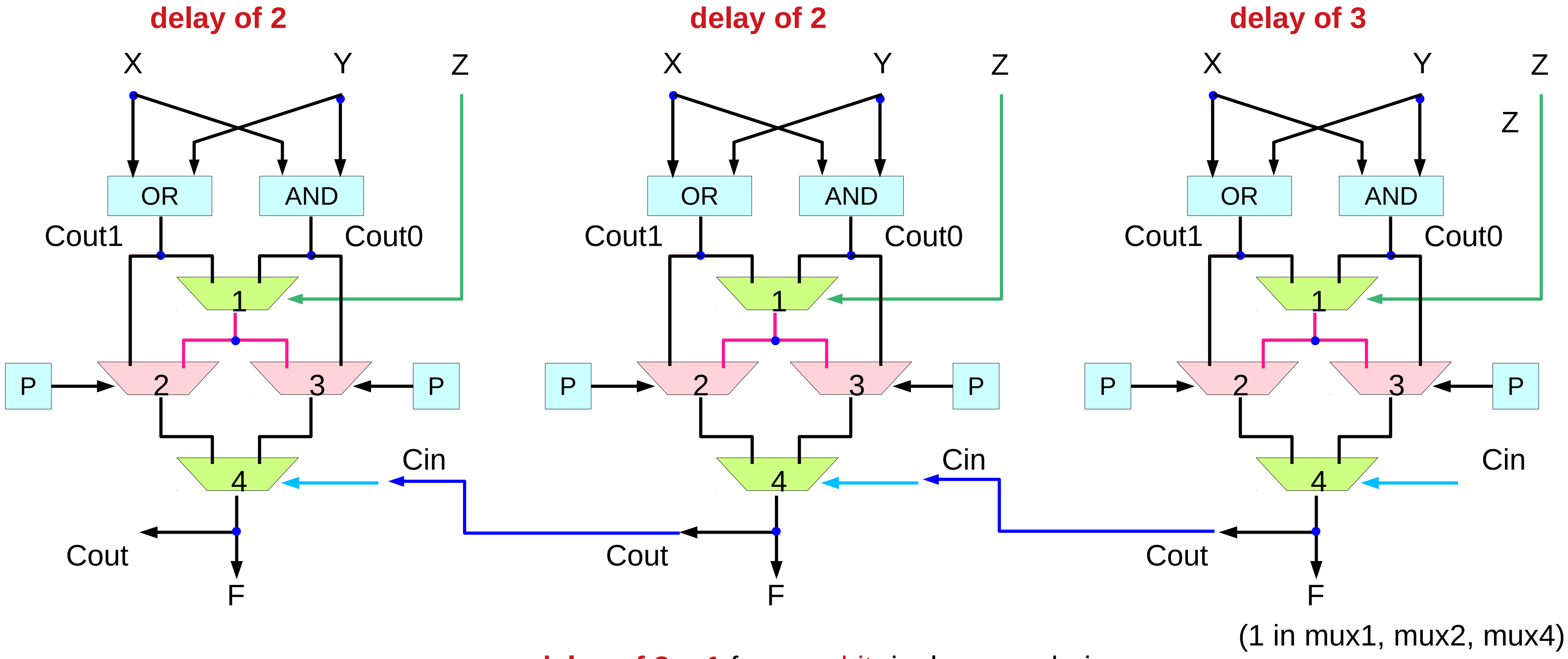
Also, for carry computations that do not need this feature,
the first cell in a carry chain built from fig 2b
can be configured to bypass mux1,
reducing the overall delay to 2n,
which is identical to that of fig2a.

in order to implement a n-bit carry chain with a carry input,
the design of fig 2a requires an additional cell
at the beginning of the chain to bring in this input,
resulting in a delay of 2(n+1)=2n+2,
which is lower than that of the design in fig2b

thus, the design of fig 2b is the preferrred
ripple carry design among those presented so far

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design B



delay of 2n+1 for an n-bit ripple carry chain

50% faster circuit that the original design

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

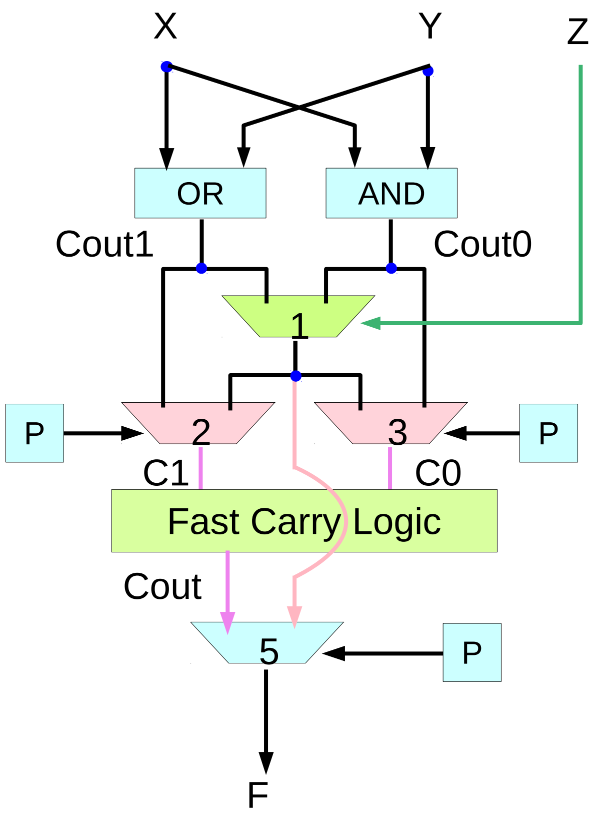
Design C

various high performance carry chains
can be developed based on
the carry cell of fig 2c

this cell is very similar to that of fig 2b,
except that the actual carry chain (mux4)
has been replaced by an abstract fast carry logic unit
and mux5 has been added

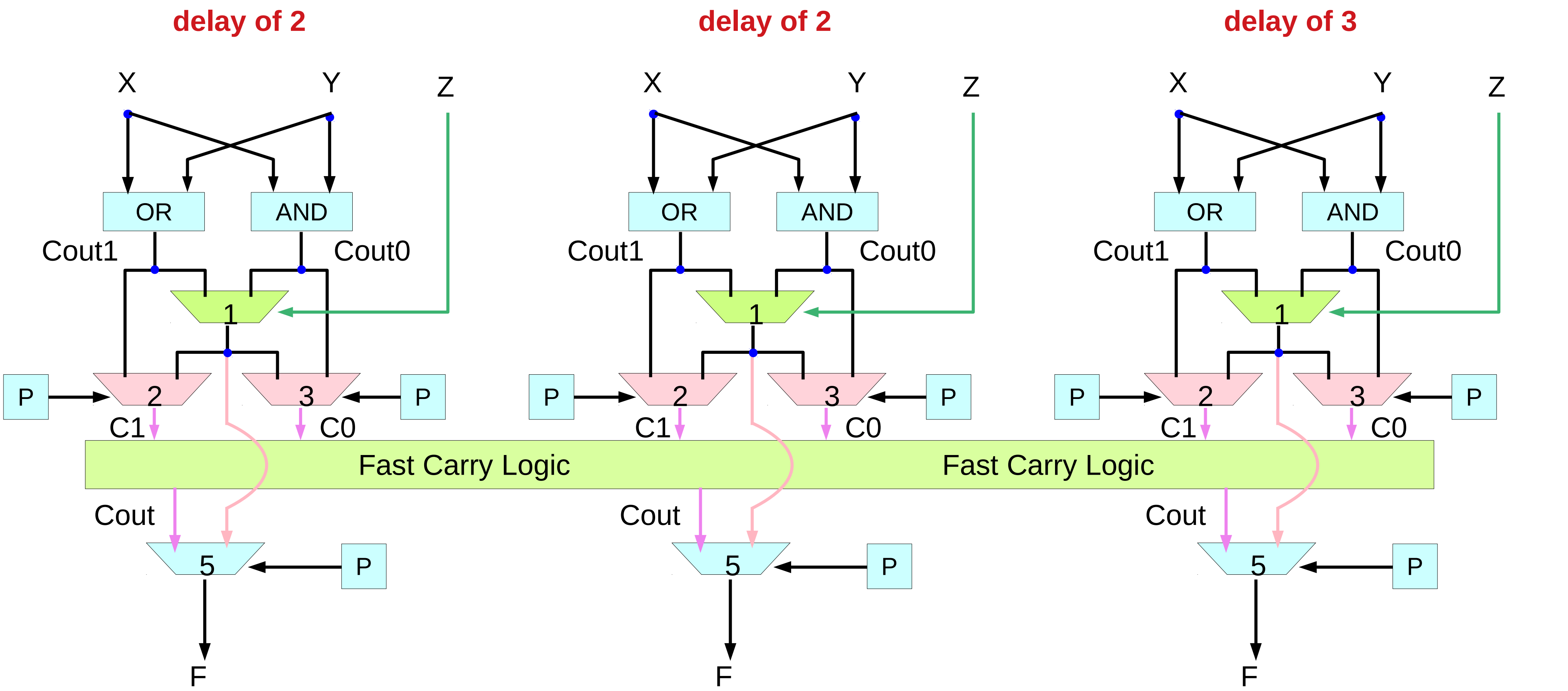
this extra mux5 is present because
although some of our faster carry chains will
have much faster carry propagation
for long carry chains,
they incur significant delay
for non-carry computations

thus, when the cell is used as
a simple normal 3LUT,
using inputs X, Y, and Z
mux5 allows us to bypass the carry chain
by selecting the output of mux1



High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design C



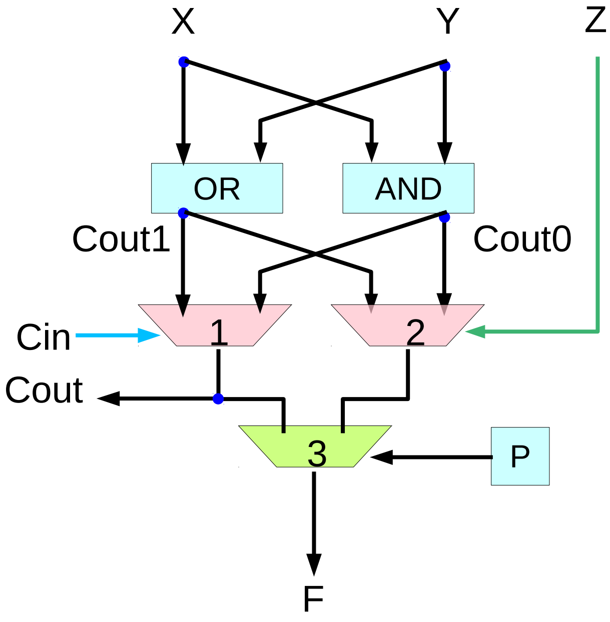
(1 for mux1, 1 for mux2, 1 in mux4)

delay of 2n+2 for an n-bit ripple carry chain

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

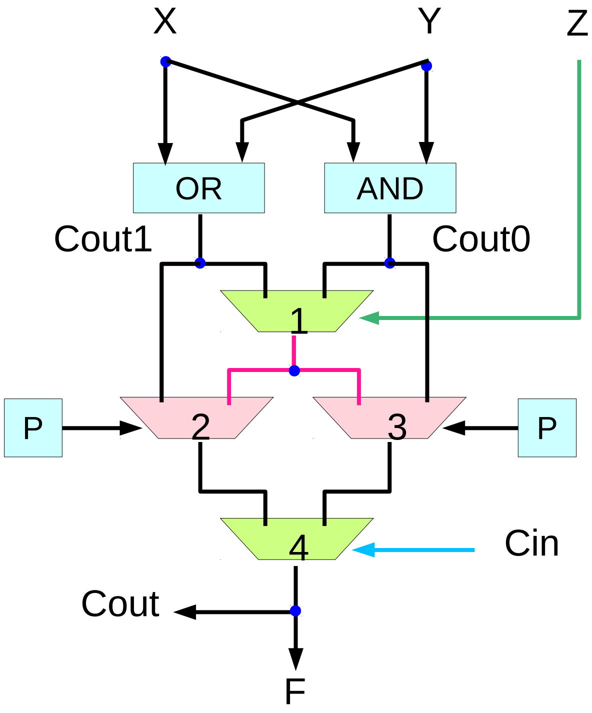
FPGA Carry Chain Cell

2n



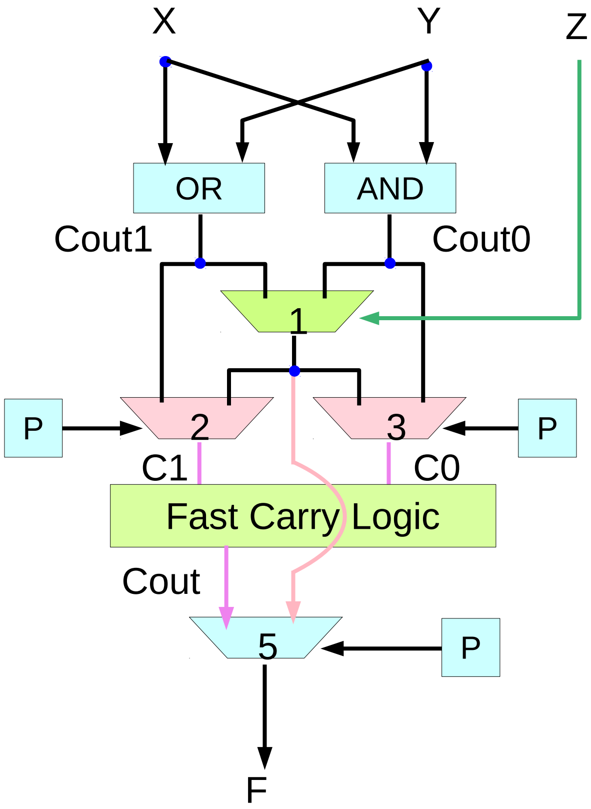
Design A

2n / 2n+1



Design B

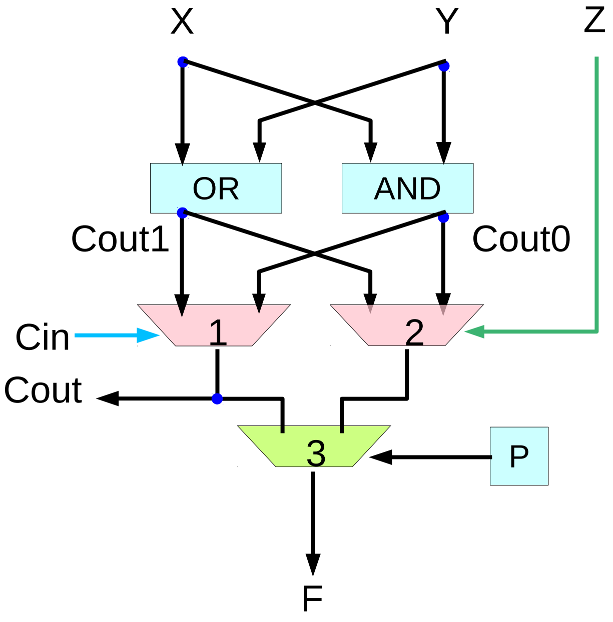
2n+2

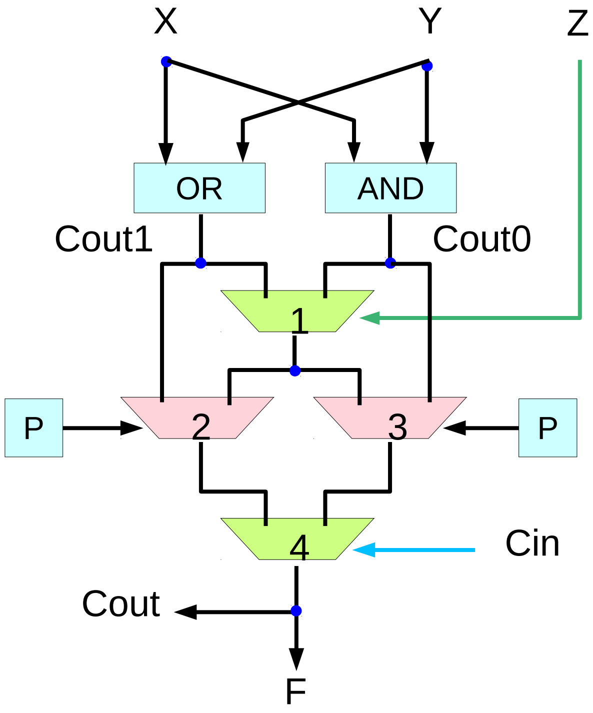


Design C

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

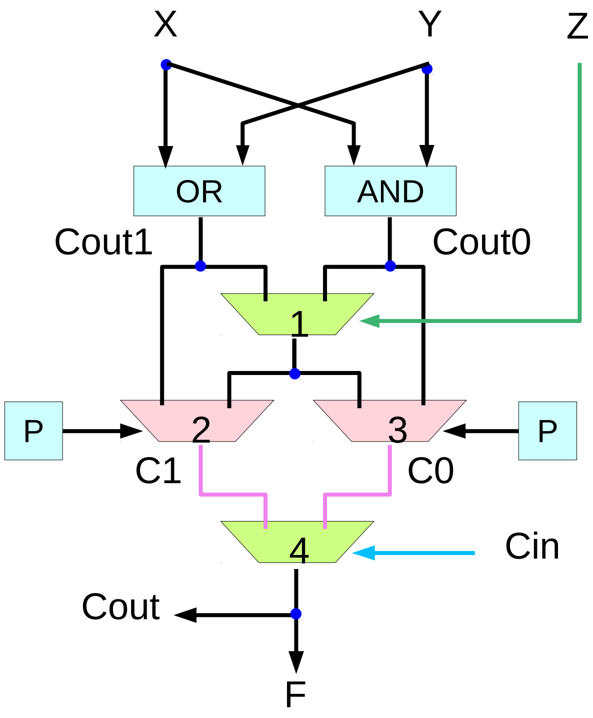
FPGA Carry Chain Cell

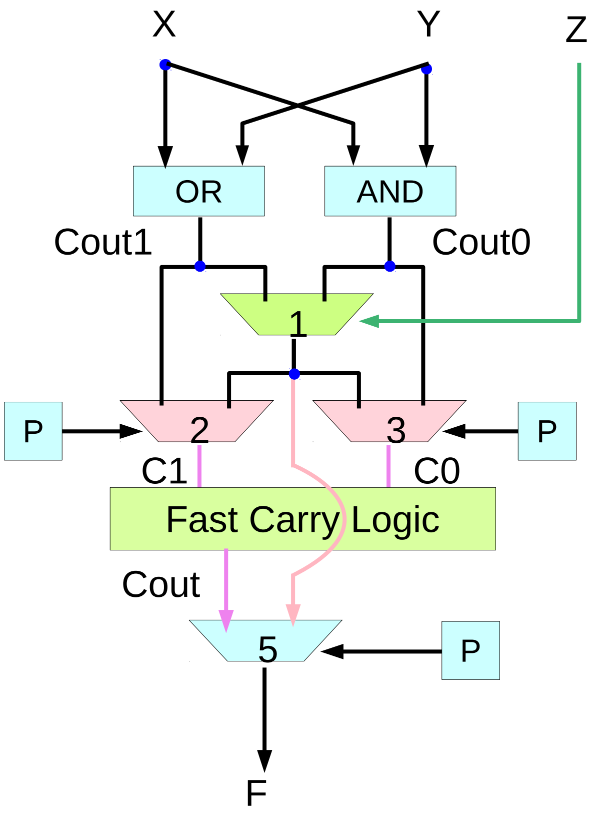




High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell

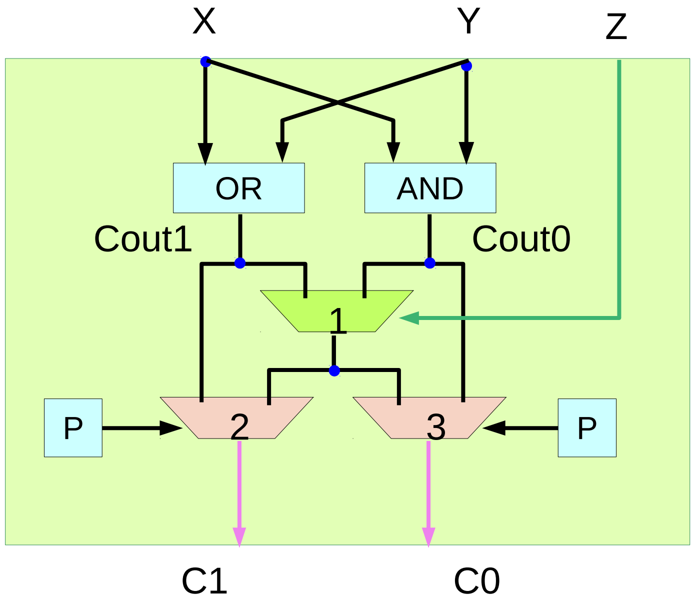




$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

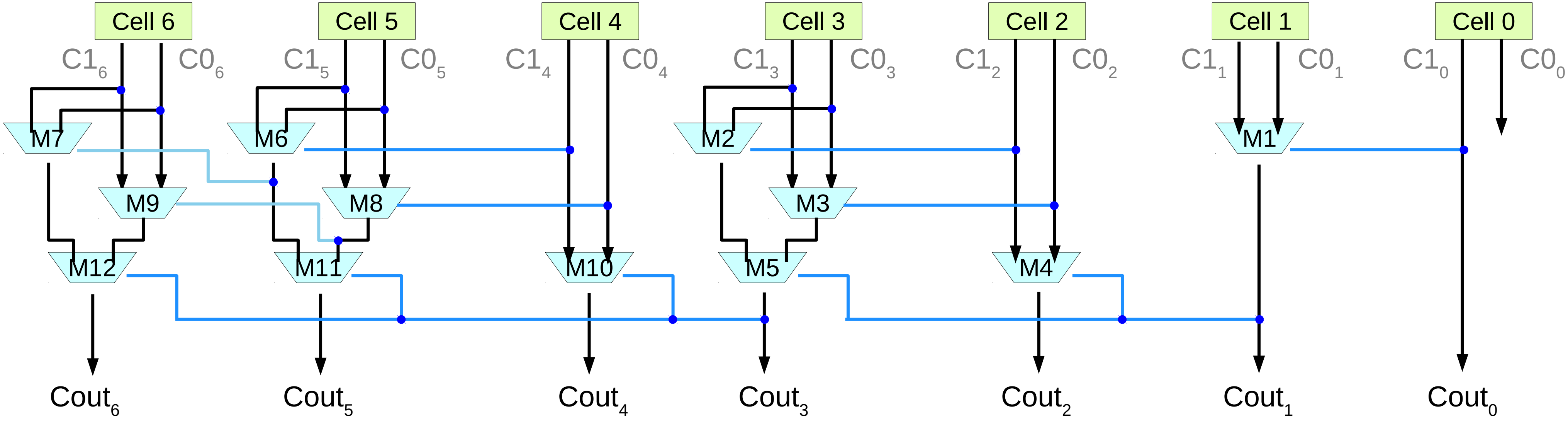
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Fast Carry Logc

Carry Select Adder
Carry Lookahead Adder
    Brent-Kung
Variable Block
Ripple Carry Adder

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

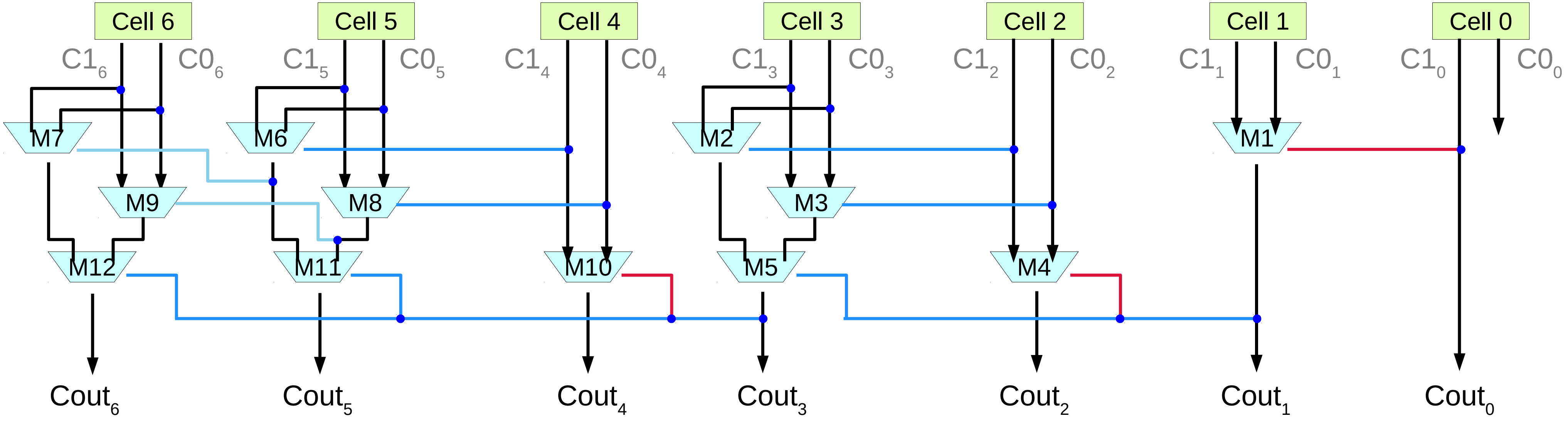
FPGA Carry Chain Cell



$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

$$Cout_1 = (Cout_0 \cdot C1_1) + (\overline{Cout_0} \cdot C0_1)$$

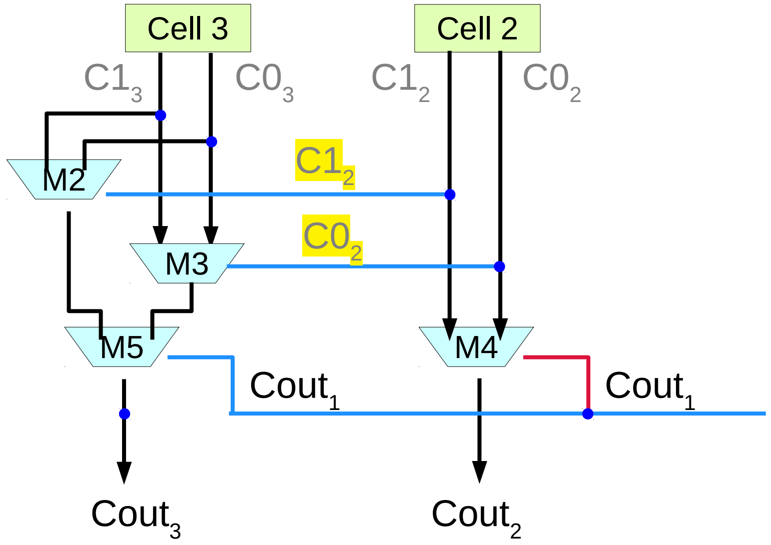
$$Cout_1 = (C1_0 \cdot C1_1) + (\overline{C1_0} \cdot C0_1)$$

$$Cout_{i+1} = (Cout_i \cdot C1_{i+1}) + (\overline{Cout_i} \cdot C0_{i+1})$$

$$Cout_{i+1} = ([(Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)] \cdot C1_{i+1}) + (\overline{[(Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)]} \cdot C0_{i+1})$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



$(C1_3\,C1_2 + C0_3\overline{C1_2})Cout_1 + (C1_3\,C0_2 + C0_3\overline{C0_2})\overline{Cout_1}$

$$Cout_i = (Cout_{i-1}\cdot C1_i) + (\overline{Cout_{i-1}}\cdot C0_i)$$

$$Cout_{i+1} = (Cout_i\cdot C1_{i+1}) + (\overline{Cout_i}\cdot C0_{i+1})$$

$$Cout_2 = (Cout_1\cdot C1_2) + (\overline{Cout_1}\cdot C0_2)$$

$$Cout_3 = (Cout_2\cdot C1_3) + (\overline{Cout_2}\cdot C0_3)$$

$$= (((Cout_1\cdot C1_2) + (\overline{Cout_1}\cdot C0_2))\cdot C1_3)$$

$$+ ((\overline{(Cout_1\cdot C1_2) + (\overline{Cout_1}\cdot C0_2)})\cdot C0_3)$$

$$(((Cout_1\cdot C1_2) + (\overline{Cout_1}\cdot C0_2))\cdot C1_3)$$

$$= (C1_3C1_2Cout_1 + C1_3C0_2\overline{Cout_1})$$

$$(((\overline{Cout_1\cdot C1_2})\cdot(\overline{\overline{Cout_1}\cdot C0_2}))\cdot C0_3)$$

$$= (((\overline{Cout_1} + \overline{C1_2})\cdot(Cout_1 + \overline{C0_2}))\cdot C0_3)$$

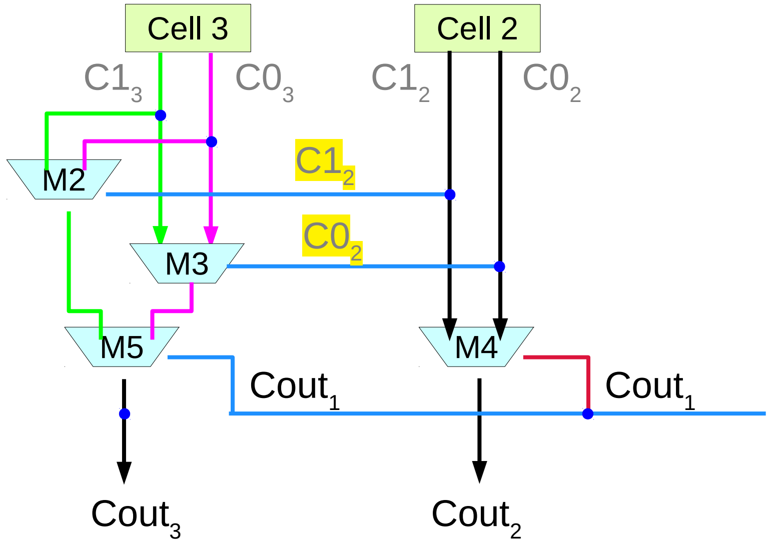
$$= (\overline{Cout_1}Cout_1 + \overline{C1_2}Cout_1 + \overline{Cout_1}\,\overline{C0_2} + \overline{C1_2}\,\overline{C0_2})\cdot C0_3$$

$$= (\overline{C1_2}Cout_1 + \overline{C0_2}\,\overline{Cout_1})\cdot C0_3$$

$$= (C0_3\overline{C1_2}Cout_1 + C0_3\overline{C0_2}\,\overline{Cout_1})$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



$$= \left(\cancel{\overline{Cout_1} Cout_1} + \overline{C1_2} Cout_1 + \overline{Cout_1}\,\overline{C0_2} + \overline{C1_2}\,\overline{C0_2}\right) \cdot C0_3$$

$$= \left(\overline{C1_2} Cout_1 + \overline{C0_2}\,\overline{Cout_1}\right) \cdot C0_3$$

$$= \left(C0_3 \overline{C1_2} Cout_1 + C0_3 \overline{C0_2}\,\overline{Cout_1}\right)$$

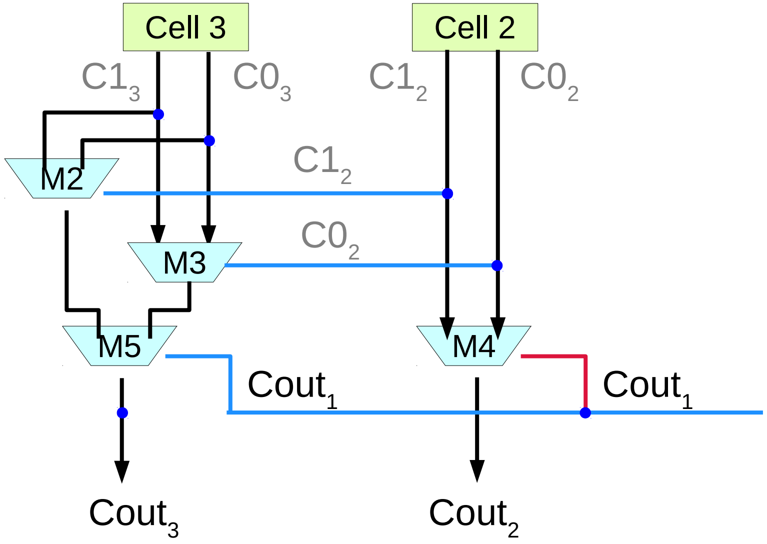
$$(C1_3\, C1_2 + C0_3 \overline{C1_2}) Cout_1 + (C1_3\, C0_2 + C0_3 \overline{C0_2}) \overline{Cout_1}$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell

$C1_3\,C1_2 + C0_3\,\overline{C1_2}$

$C1_3\,C0_2 + C0_3\,\overline{C0_2}$



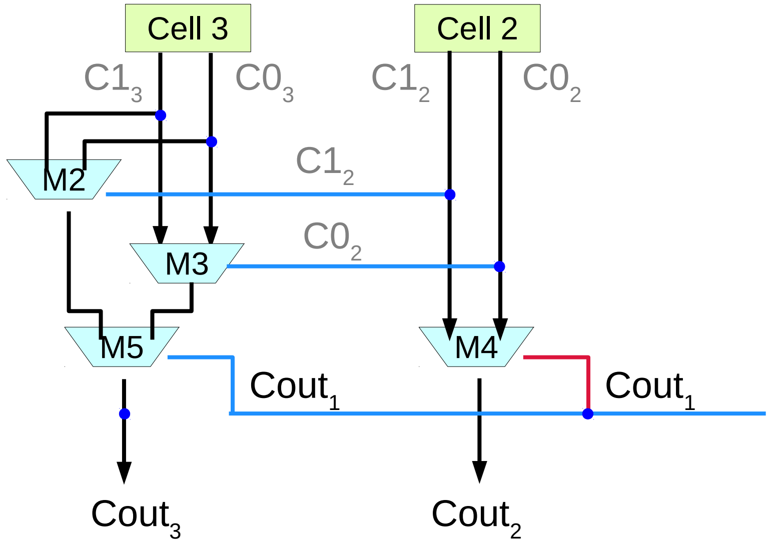
$(C1_3\,C1_2 + C0_3\,\overline{C1_2})Cout_1 + (C1_3\,C0_2 + C0_3\,\overline{C0_2})\overline{Cout_1}$

$$= C1_3 \cdot (C1_2\,Cout_1 + C0_2\,\overline{Cout_1})$$

$$+ C0_3 \cdot (\overline{C1_2}\,Cout_1 + \overline{C0_2}\,\overline{Cout_1})$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

$$Cout_{i+1} = (Cout_i \cdot C1_{i+1}) + (\overline{Cout_i} \cdot C0_{i+1})$$

$$Cout_{i+1} = ([(Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)] \cdot C1_{i+1})$$

$$+ (\overline{[(Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)]} \cdot C0_{i+1})$$

High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

References

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