

FPGA Carry Chain Adder (1A)

-
-

Copyright (c) 2010 -- 2020 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

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

This document was produced by using OpenOffice and Octave.

FPGA Carry Chain Cell



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

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

when $\bar{p}_i = 1$, 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

FPGA Carry Chain Cell



Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded 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

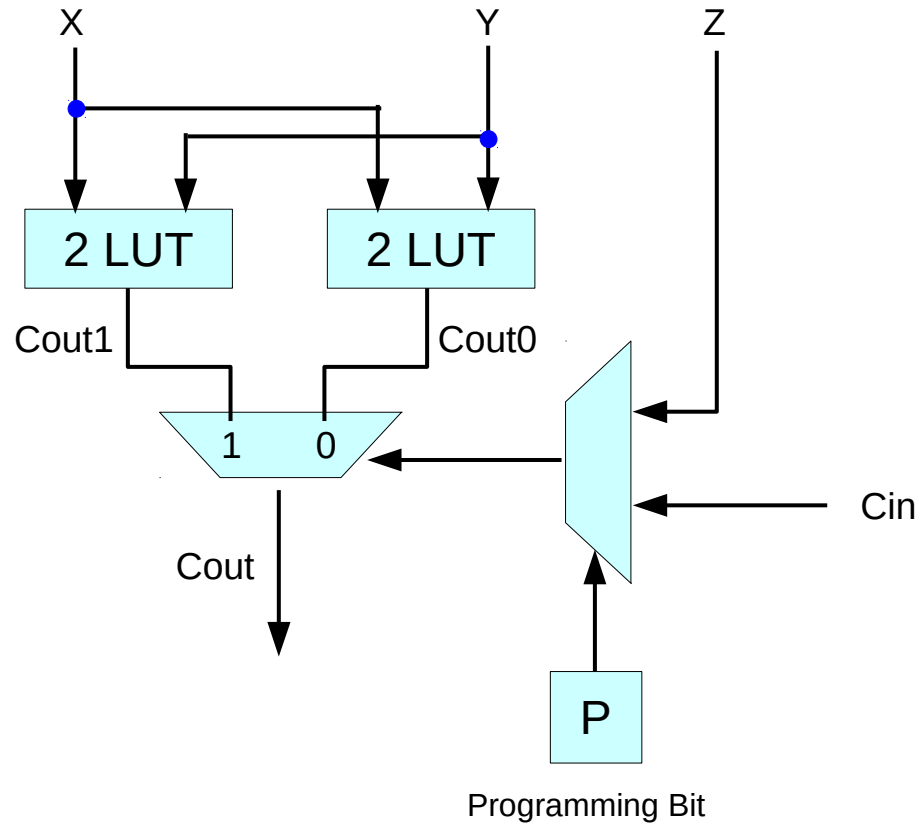
The carry chain is made up of multiplexers belonging to adjacent configurable blocks

the lookup table is used for implementing the exclusive or function

$$p(i) = x(i) \text{ xor } y(i)$$

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

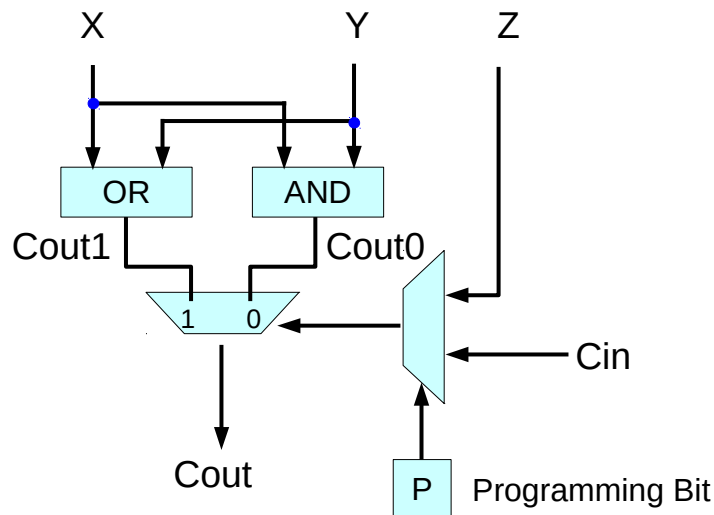
Cout = (X + Y) Cin + X Y $\overline{\text{Cin}}$

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



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

$$\text{Cout}(X, Y, 1) = \text{Cout1} = X + Y$$

$$\text{Cout}(X, Y, 0) = \text{Cout0} = X Y$$

$$\text{Cout1} = X + Y \text{ when Cin}=1$$

$$\text{Cout0} = X Y \text{ when Cin}=0$$

$$\text{Cout1} = P' \text{ Cin} \dots P' = \text{relaxed } P$$

$$\text{Cout0} = G \overline{\text{Cin}}$$

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

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

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

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

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

$$P \text{ Cin} + G$$

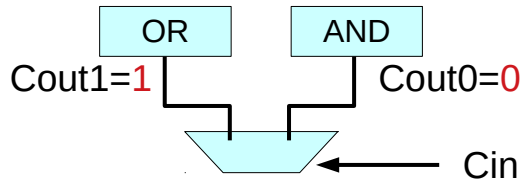
$$\text{Cin} (X + Y) + \overline{\text{Cin}} X Y$$

$$\text{Cin } P' + \overline{\text{Cin}} G$$

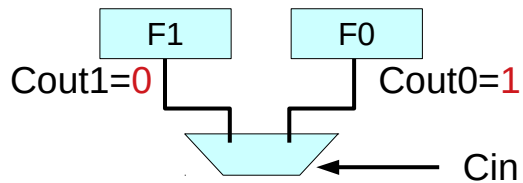
$$\dots P' : \text{relaxed } P$$

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

FPGA Carry Chain Cell



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



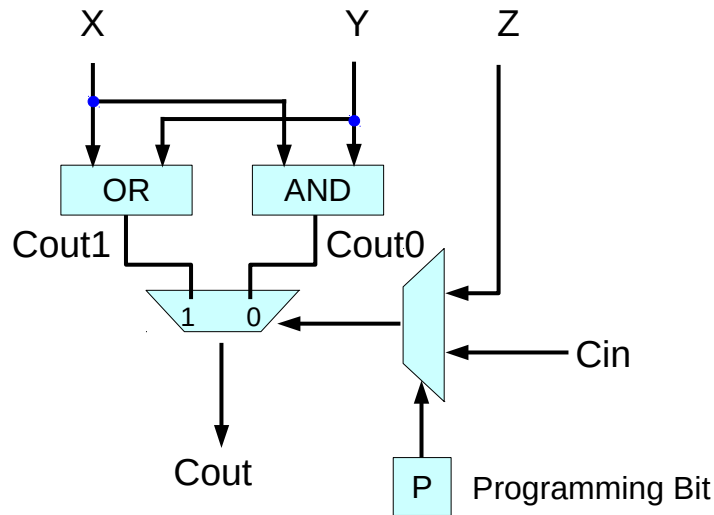
Cout1=0 when Cin=1
 Cout0=1 when Cin=0
 Cout = $\overline{\text{Cin}}$

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

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

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

Carry Chain



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

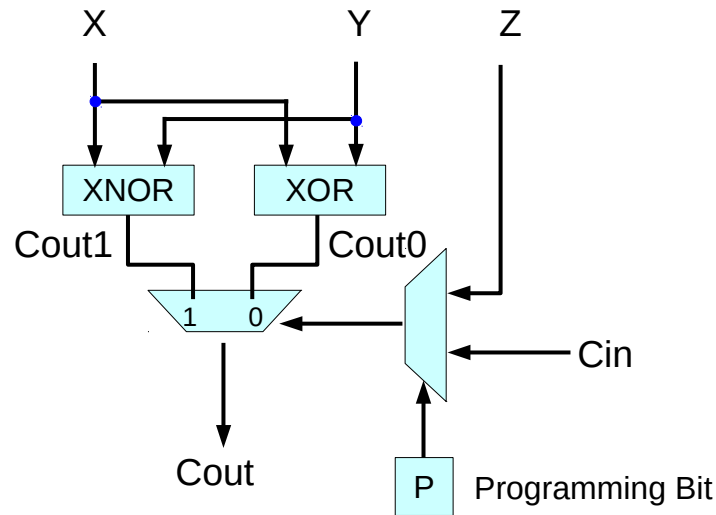
Carry Out

X	Y	Cin	Cout
0	0	Cin	$\overline{\text{Cin}}$
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 propagate

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

Parity Checker



Computing Parity

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

X	Y	Cin	\overline{Cin}	
		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

Cout1=1 when Cin=1

Cout0=0 when Cin=0

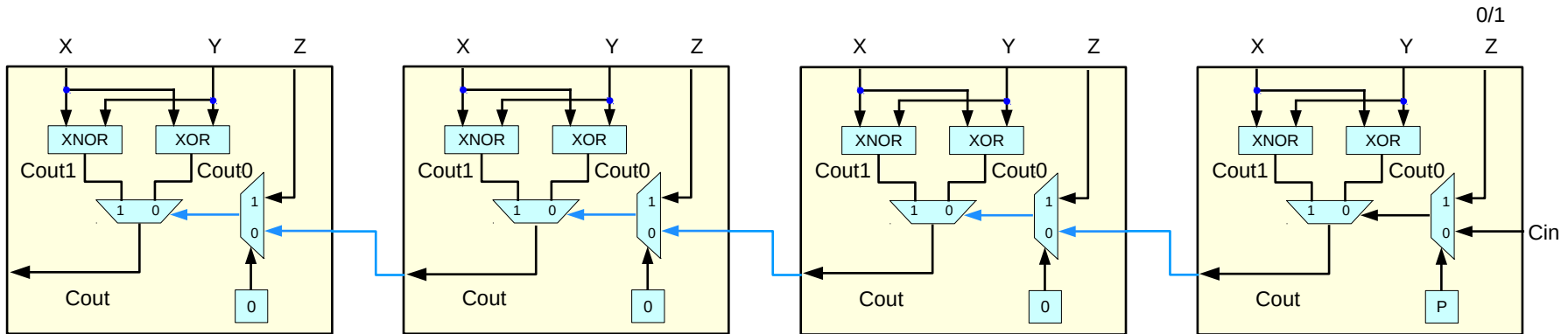
Cout = Cin propagate

Cout1=0 when Cin=1

Cout0=1 when Cin=0

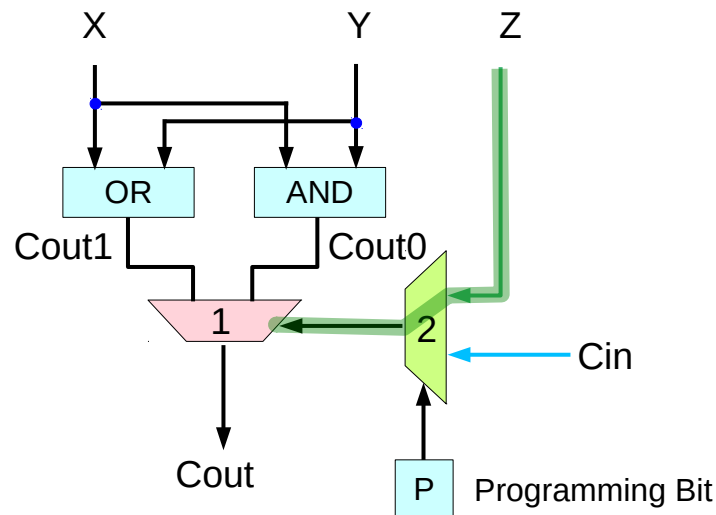
Cout = \overline{Cin} inverse propagate

Ripple Carry Chain



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

FPGA Carry Chain Cell



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.

to start a carry chain, the **first cell** in the chain
must be programmed to **ignore** the **Cin** signal

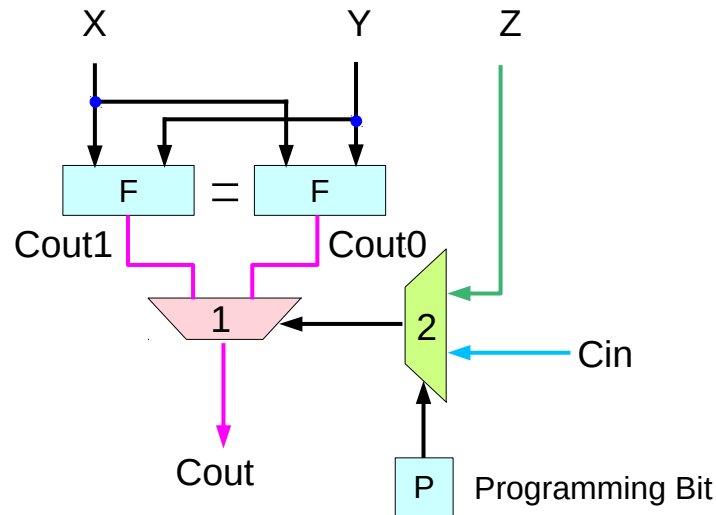
program **mux2** in the cell to route input **Z**
to **mux1** instead of **Cin**

When it is desired to have a carry input
to the first cell of the chain
(implementing combined adder/subtractors)

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 will only needlessly
slow down the circuit's computation.

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

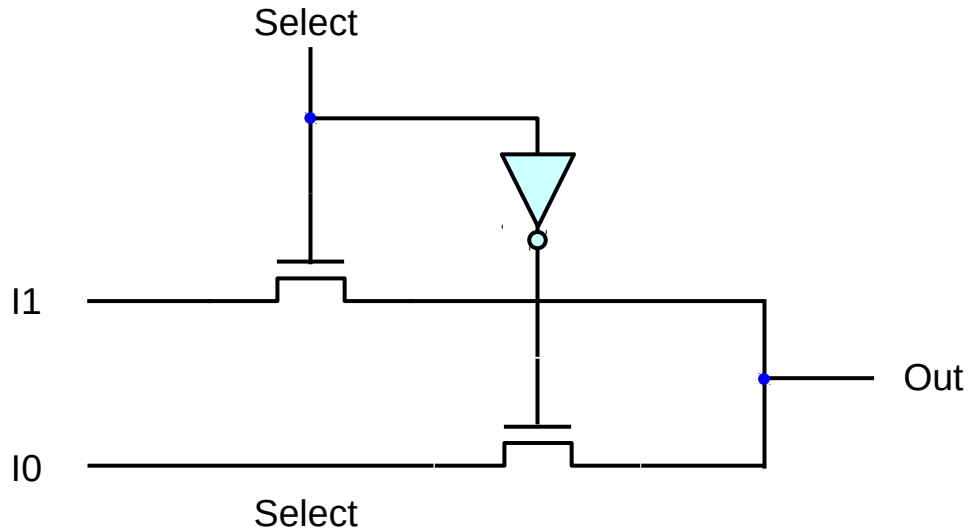
FPGA Carry Chain Cell



The first cell with only 2 inputs
thus either 2-LUT in the cell could compute this value.
if we program both 2-LUTs with the same function,
the output will be forced to the proper value
regardless of the input, and thus either the Cin or the Z
can be routed to mux1 without changing the computation

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

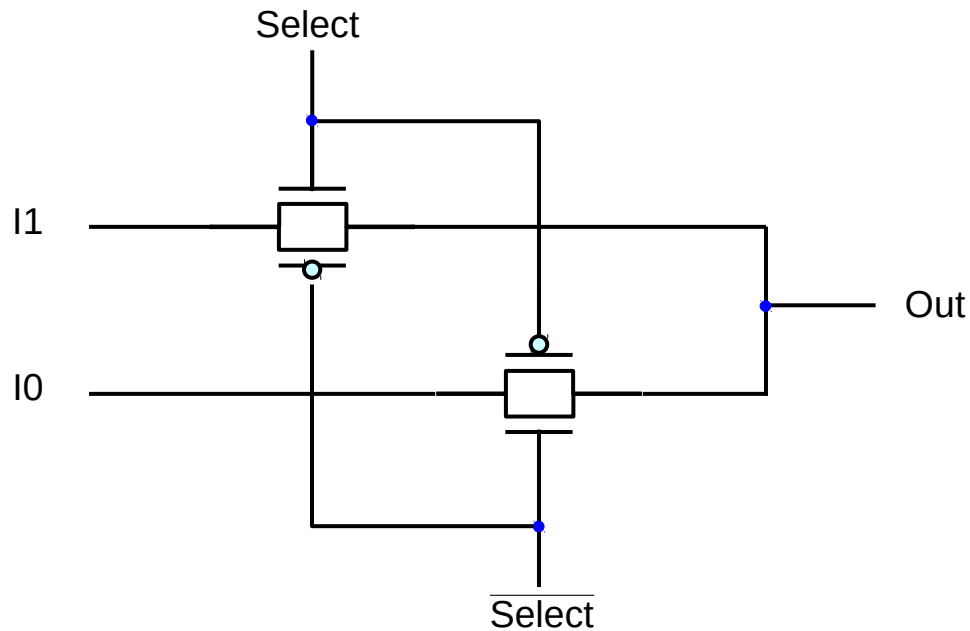
Ripple Carry Chain



however, this is only true if mux1 is implemented such that if the two inputs to the mux are the same, the output of the mux is identical to the inputs regardless of the state of the select line

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 ow 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.

Ripple Carry Chain



however a mux built like that in fig1c, with both n-transistor and p-transistor pass gates will operate properly for this case. thus, we will assume throughout any other mux implementation with the same property could be used (including tristate driver based muxes which an restore signal drive and cut series RC chains)

FPGA Carry Chain Cell

however, this is only true if mux1 is implemented such that if the two inputs to the mux are the same, the output of the mux is identical to the inputs regardless of the state of the select line
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.

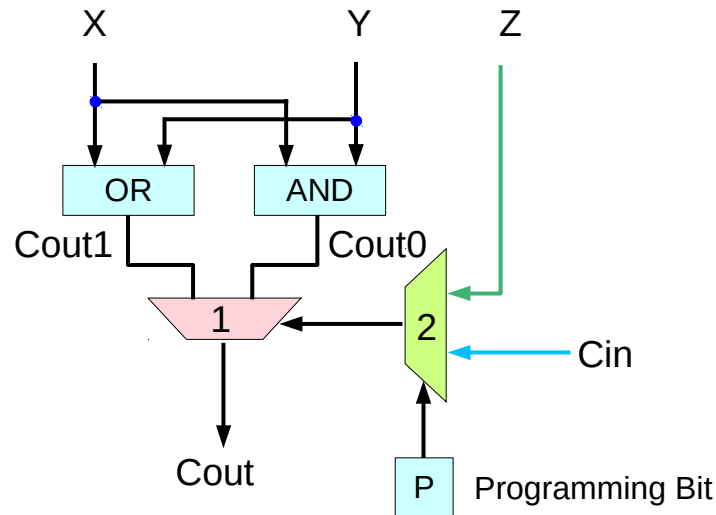
however a mux built like that in fig1c, with both n-transistor and p-transistor pass gates will operate properly for this case.
thus, we will assume throughout any other mux implementation with the same property could be used (including tristate driver based muxes which are restore signal driven and cut series RC chains)

FPGA Carry Chain Cell

in an fpga, the cells represent resources that can be used to compute arbitrary functions. however, the location of functions within this structure is completely up to the user. thus, a user may decide to start or end a carry computation at any place in the array. in order to start a carry chain we must program the first cell in the carry chain to ignore the Cin signal. one easy way to do this is to program mux2 in the cell to route input Z to mux1 instead of Cin for situations where one wishes to have carry input to the first stage of an adder (which is usual for implementing combined adder/subtractors as well as other circuits) this is the right solution. however, in other cases this may not be possible. the first stage in many carry computations is only a 2-input function, and forcing the carry chain to wait for the arrival of an additional, unnecessary input will only needlessly slow down the circuit's computation. this is not necessary in these circuits, the first stage is only a 2-input function. thus, either 2-LUT in the cell could compute this value. if we program both 2-LUTs with same function, the output will be forced to the proper value regardless of the input, and thus either the Cin or the Z signal can be routed to mux1 without changing the computation.

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

FPGA Carry Chain Cell

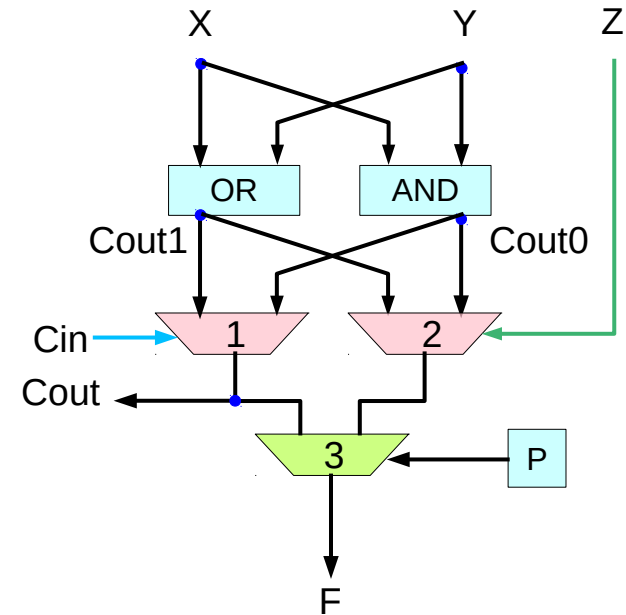
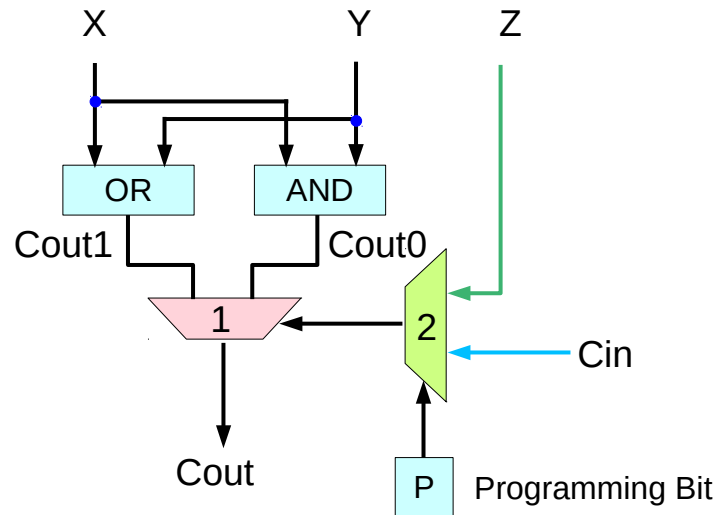


Significantly slower
Two muxes on the carry chain in each cell
Delay 1 for first cell
Delay 3 for each additional cell in the carry chain
1 delay for mux 2 and 2 delays for mux 1
Overall $2n-2$ for an n -cell carry chain

The critical path comes from the 2-LUTs and not 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

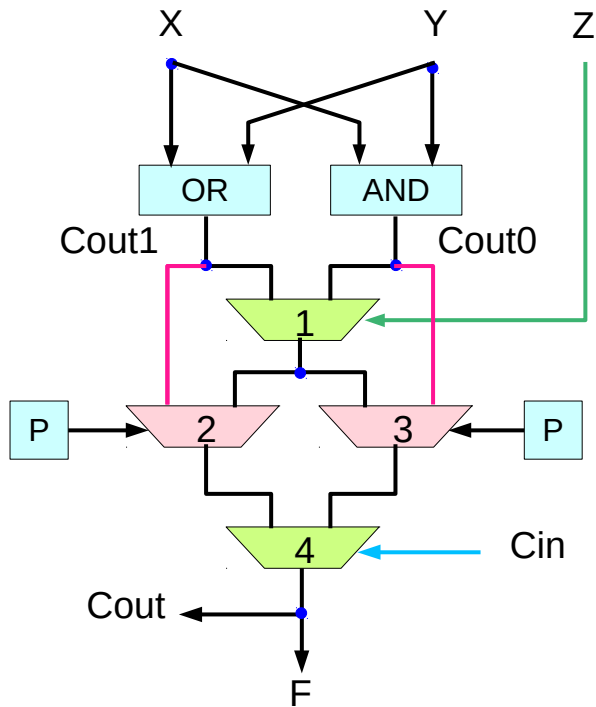
FPGA Carry Chain Cell



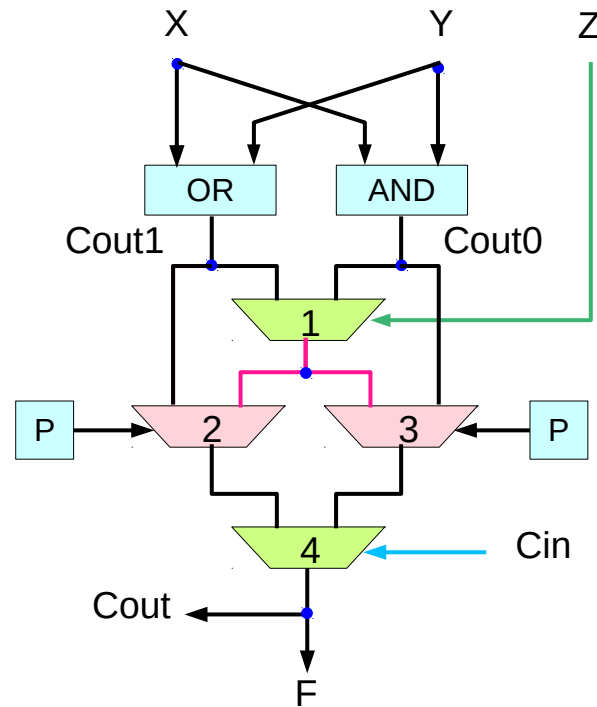
- not logically equivalent
- no longer use the Z input in the first cell since Z is only attached to mux2 and mux 2 does not lead to the carry cells

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

FPGA Carry Chain Cell



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.

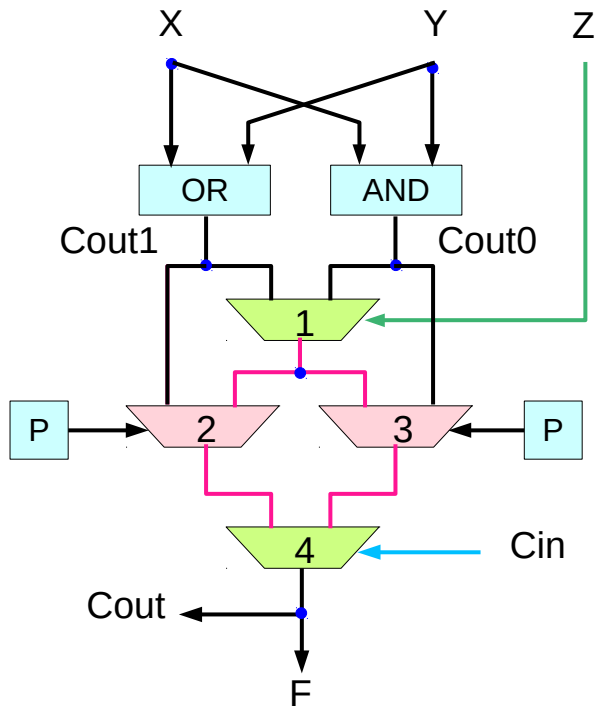


For the first cell in a carry chain
with a carry input (provided by input Z),
mux2 and mux3 both pass the value from mux1

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

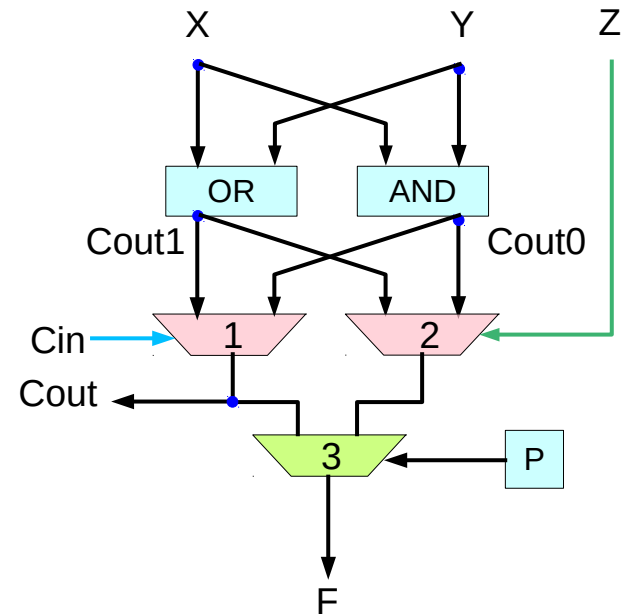
FPGA Carry Chain Cell



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

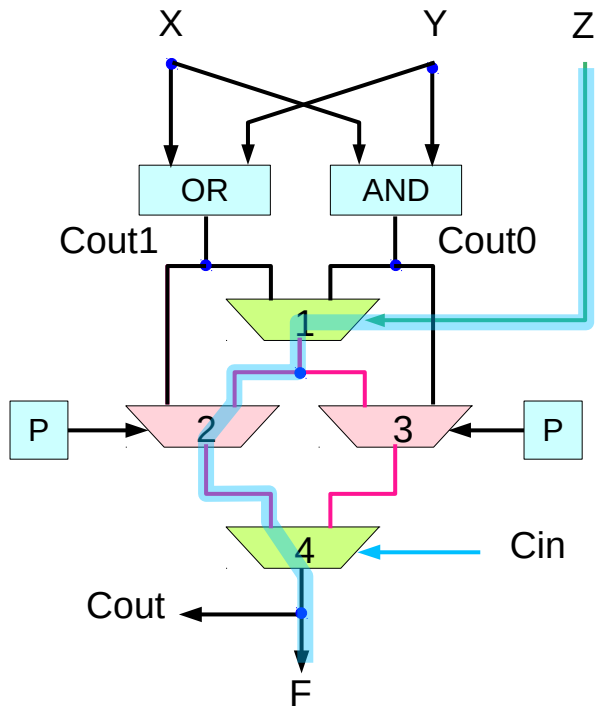
When mux2 and mux3 pass the value from mux1 (Cout1 and Cout2 respectively) the circuit is configured to continue the carry chain

Functionally equivalent



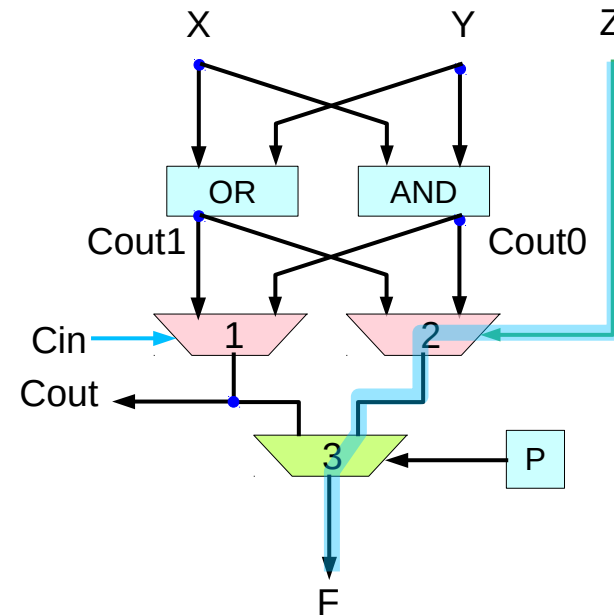
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

FPGA Carry Chain Cell



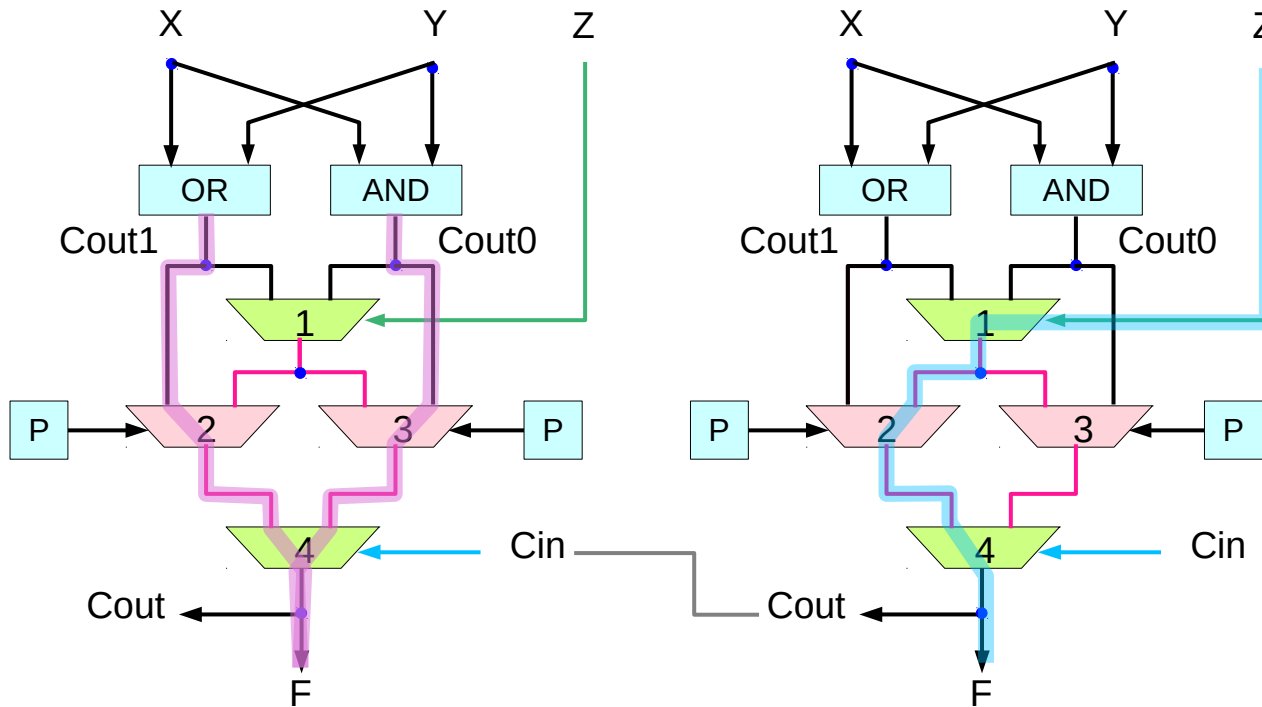
a delay of 3 in the first cell
 (1 in mux1, 1 in mux2, 1 in mux4)
 2 in all other cells in the carry chain
 an total delay of $2n+1$ for an n -bit carry chain

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

FPGA Carry Chain Cell



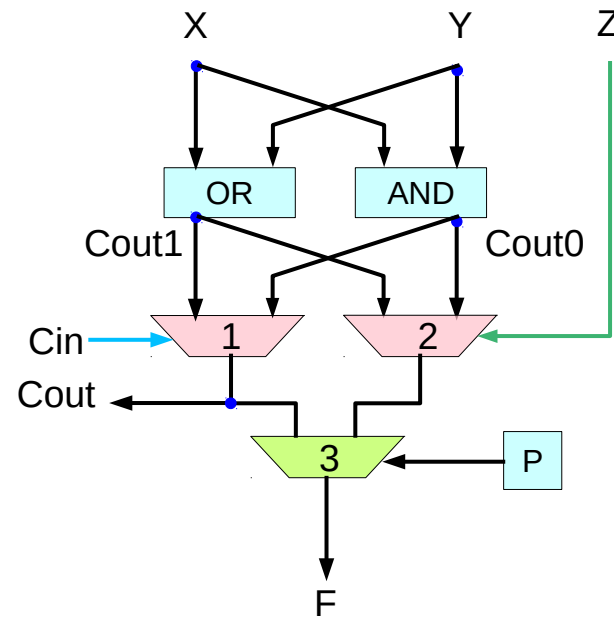
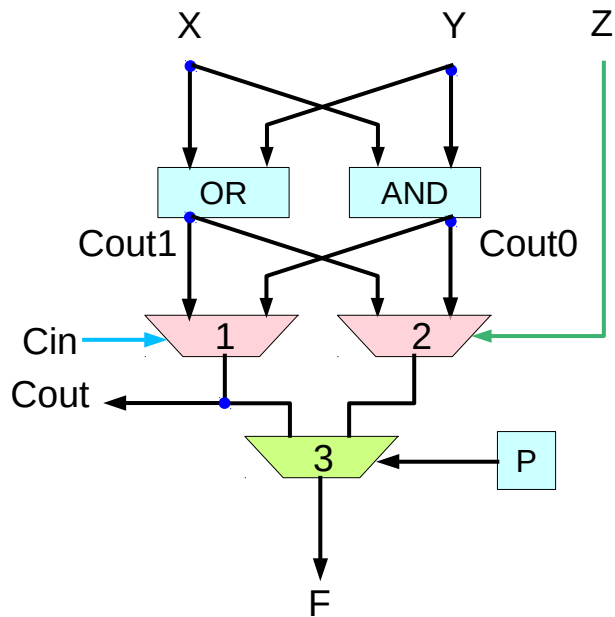
Also, for carry computations that do not need this feature, the first cell in a carry chain built from fig 2b can be configured to bypass mux1, reducing the overall delay to $2n$, which is identical to that of fig2a.

in order to implement a n -bit carry chain with a carry in, 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

FPGA Carry Chain Cell

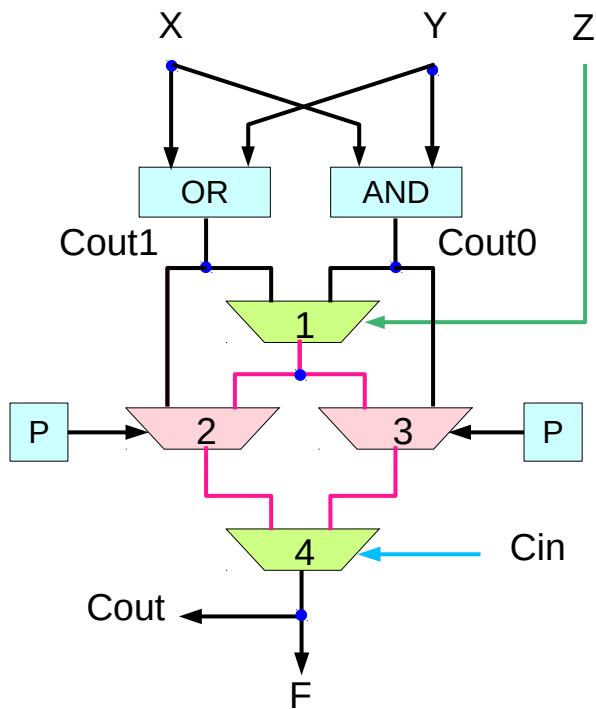


in order to implement a n -bit carry chain with a carry input the design of fig 2a requires an additional cell at the beginning of the chain to bring in this input, resulting in a delay of $2(n+1)=2n+2$, which is lower than that of the design in fig2b

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

FPGA Carry Chain Cell



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

t_1 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 preferred
ripple carry design among those presented so far

FPGA Carry Chain Cell

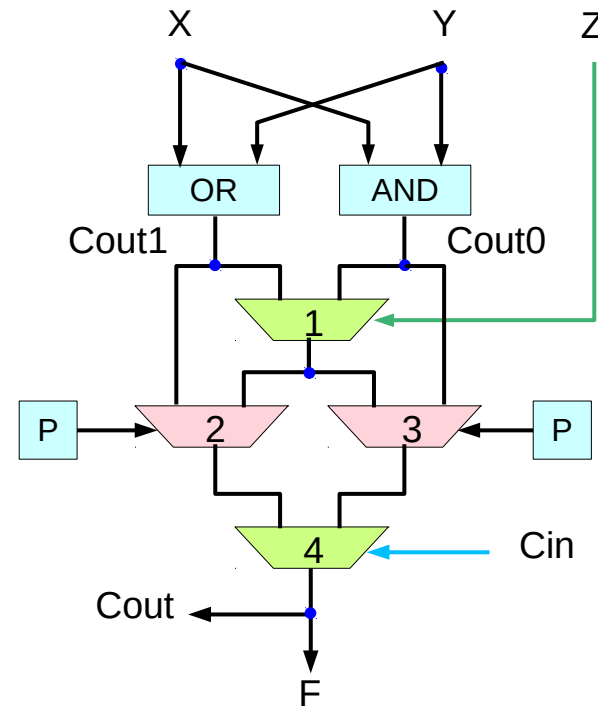
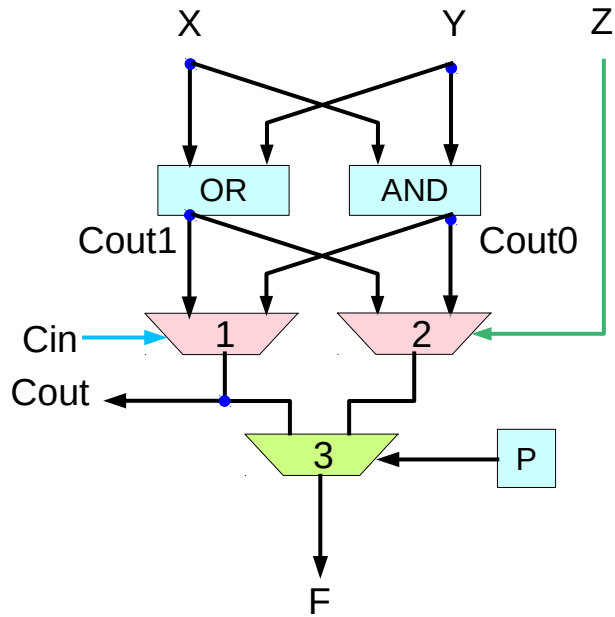
However, carry chains built from this design have a delay of 3 in the first cell (1 in mux1, 1 in mux2, 1 in mux4) and 2 in all other cells in the carry chain, yielding an overall delay of $2n+1$ for an n -bit carry chain.

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.

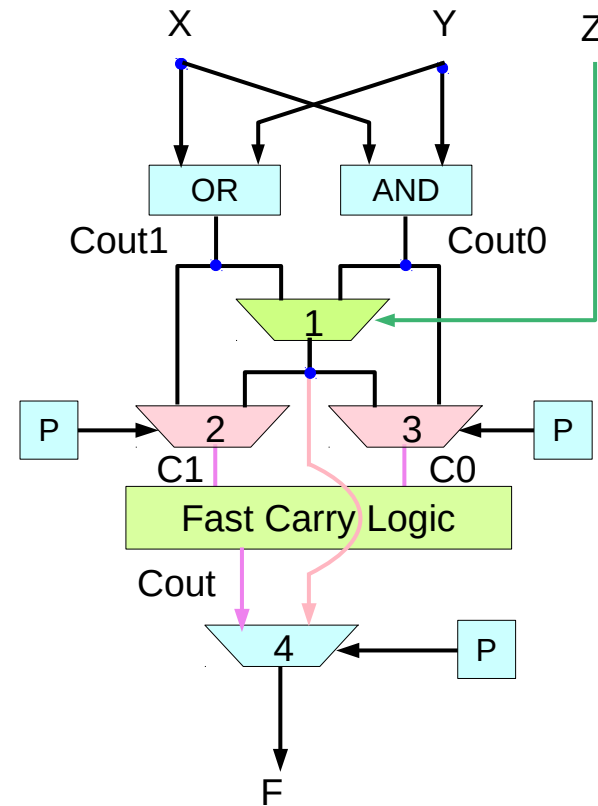
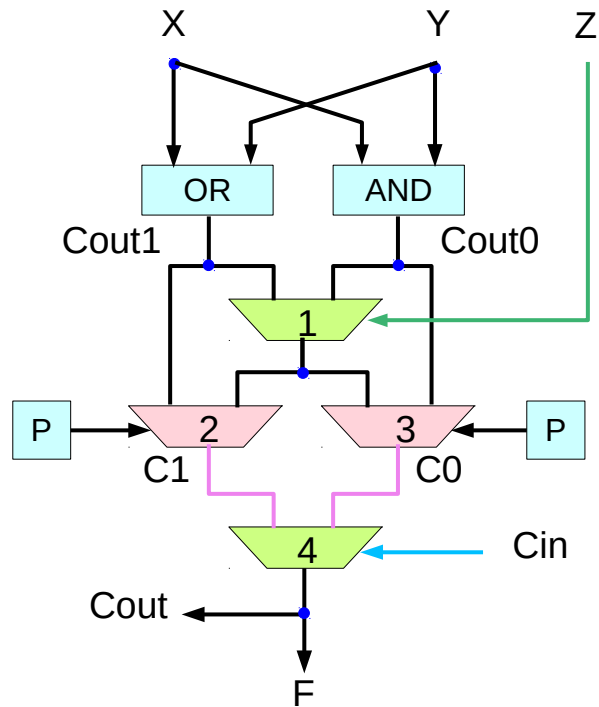
on the other hand, in order to implement a n -bit carry chain with a carry input, the design of fig 2a requires an additional cell at the beginning of the chain to bring in this input, resulting in a delay of $2(n+1)=2n+2$, which is lower than that of the design in fig2b thus, the design of fig 2b is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

FPGA Carry Chain Cell



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

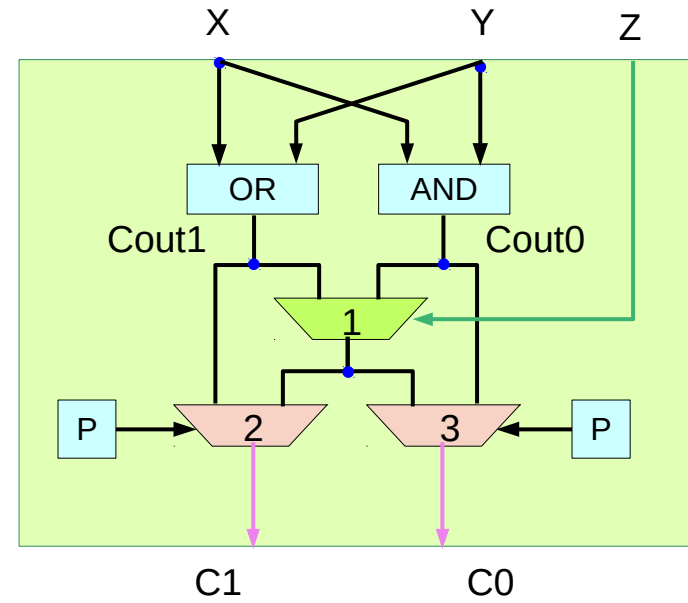
FPGA Carry Chain Cell



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

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

FPGA Carry Chain Cell



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

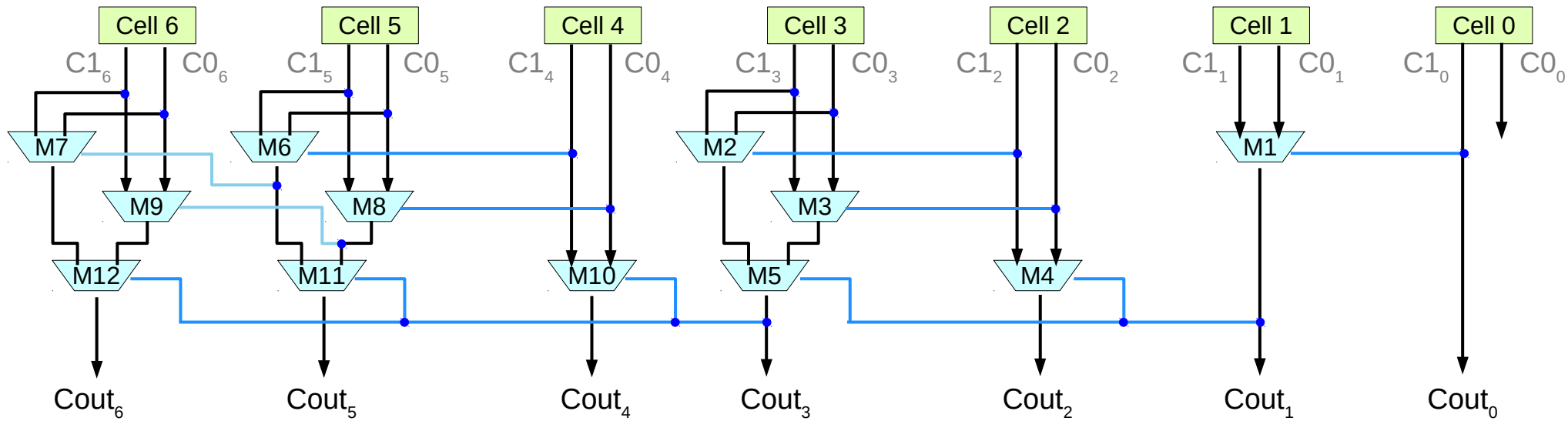
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Fast Carry Logc

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

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

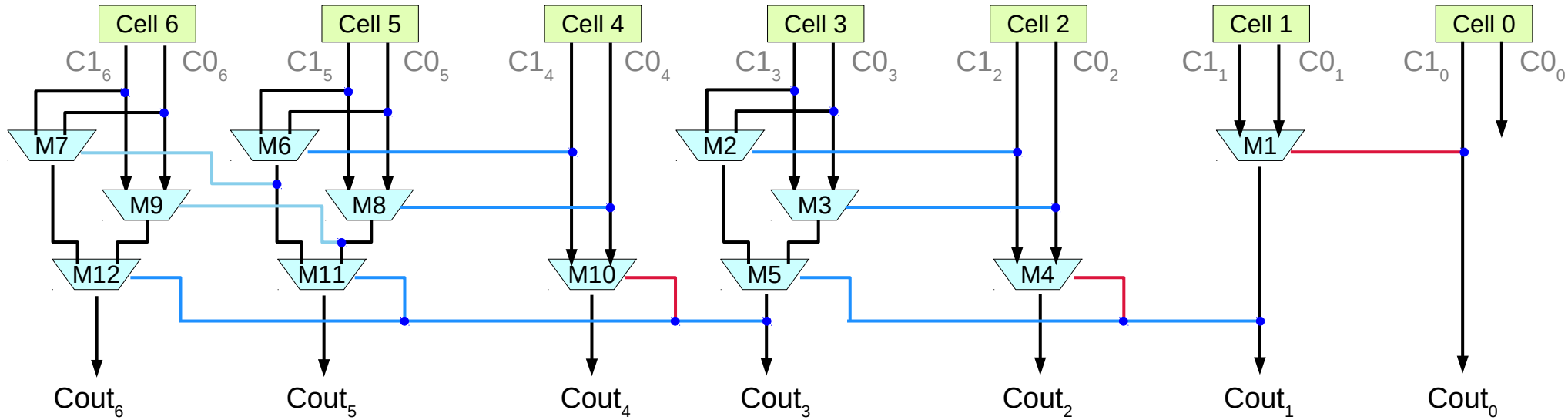
FPGA Carry Chain Cell



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

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

FPGA Carry Chain Cell



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

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

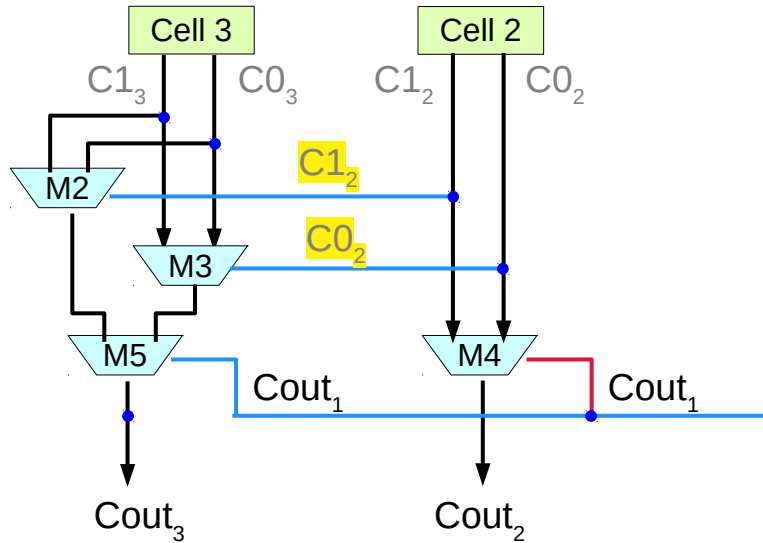
$$Cout_{i+1} = (Cout_i \cdot C1_{i+1}) + (\overline{Cout_i} \cdot C0_{i+1})$$

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

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

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

FPGA Carry Chain Cell



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

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

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

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

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

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

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

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

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

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

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

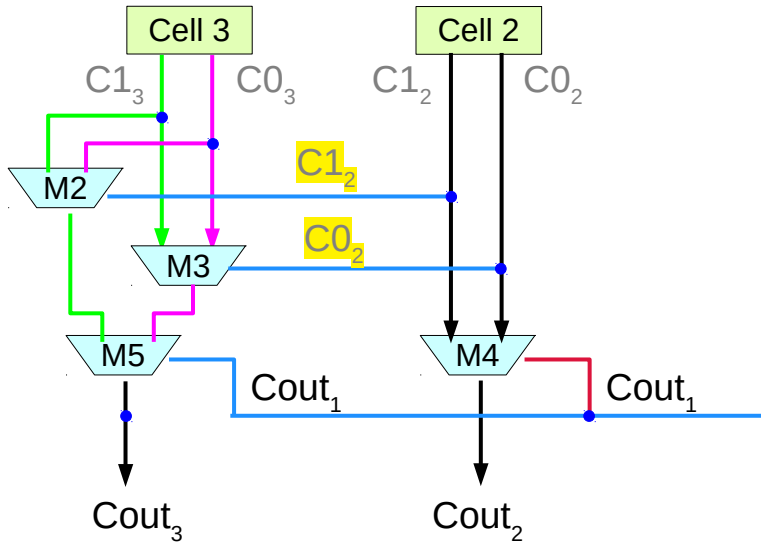
$$= (\overline{C1_2} \overline{Cout_1} + \overline{C0_2} \overline{Cout_1}) \cdot C0_3$$

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

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

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

FPGA Carry Chain Cell

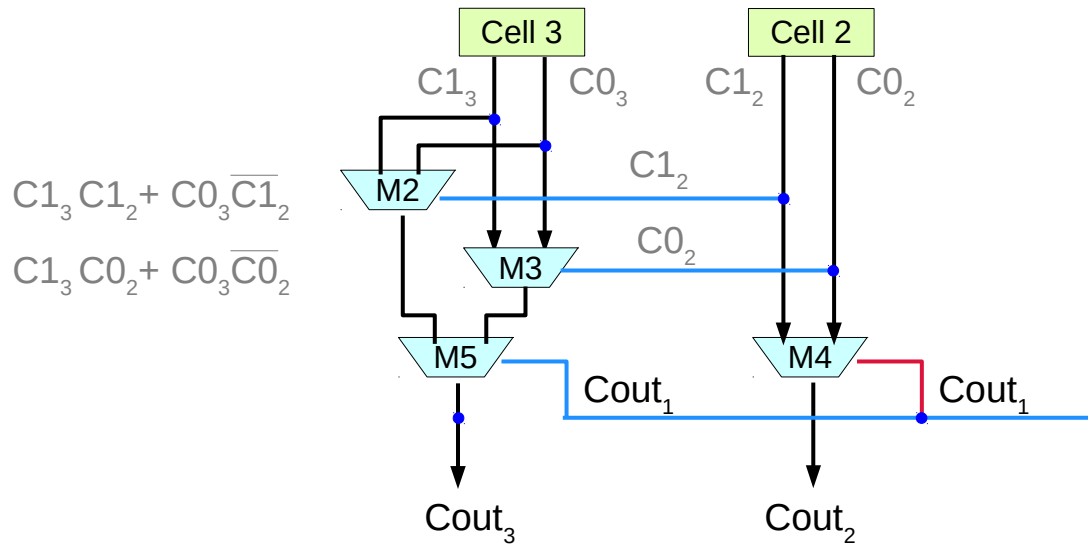


$$\begin{aligned}
 &= (\overline{Cout_1}Cout_1 + \overline{C1_2}Cout_1 + \overline{Cout_1}C0_2 + \overline{C1_2}C0_2) \cdot C0_3 \\
 &= (\overline{C1_2}Cout_1 + \overline{C0_2}Cout_1) \cdot C0_3 \\
 &= (C0_3\overline{C1_2}Cout_1 + C0_3\overline{C0_2}Cout_1)
 \end{aligned}$$

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

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

FPGA Carry Chain Cell



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

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

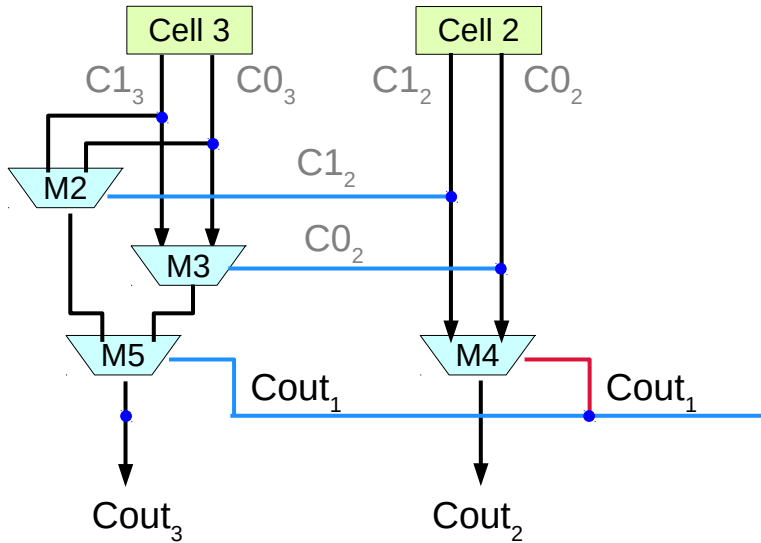
$$(C1_3 C1_2 + C0_3 \overline{C1_2}) Cout_1 + (C1_3 C0_2 + C0_3 \overline{C0_2}) \overline{Cout_1}$$

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

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

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

FPGA Carry Chain Cell



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

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

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

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