

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 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

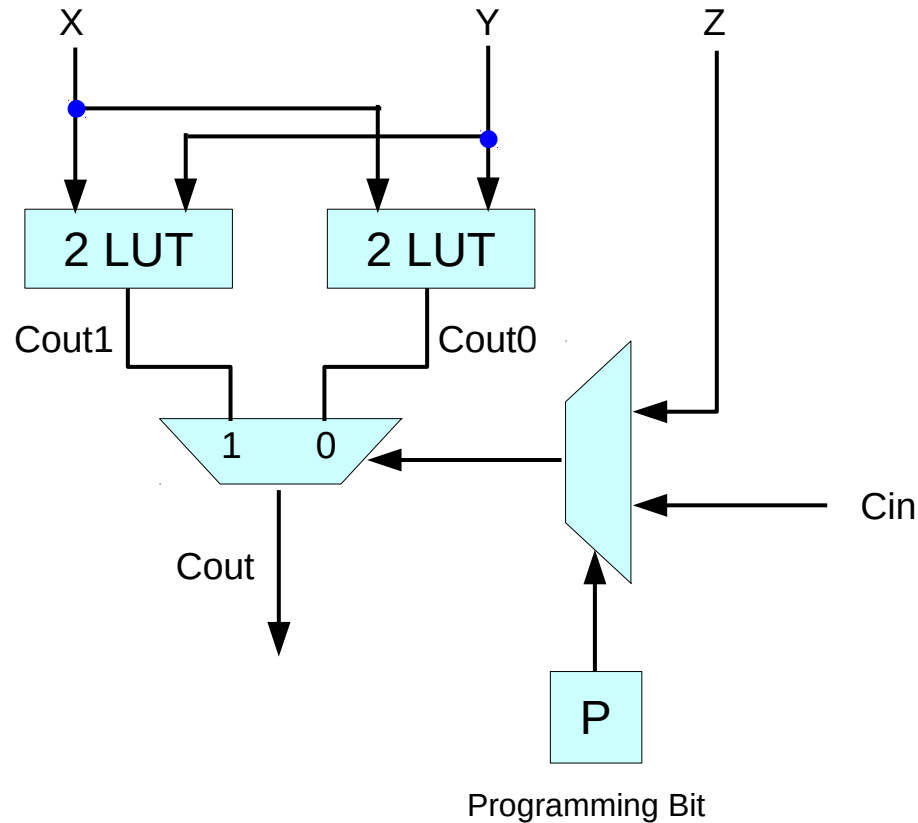
The carry chain is made up of multiplexers belonging to adjacent configurable blocks

the lookup table is used for implementing the exclusive or function

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

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

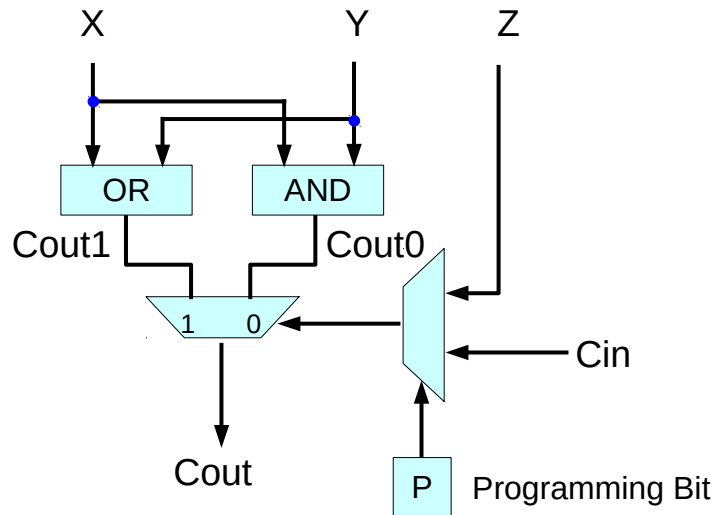
Cout = (X + Y) Cin + X Y $\overline{\text{Cin}}$

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

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

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

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

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

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

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

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

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

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

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

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

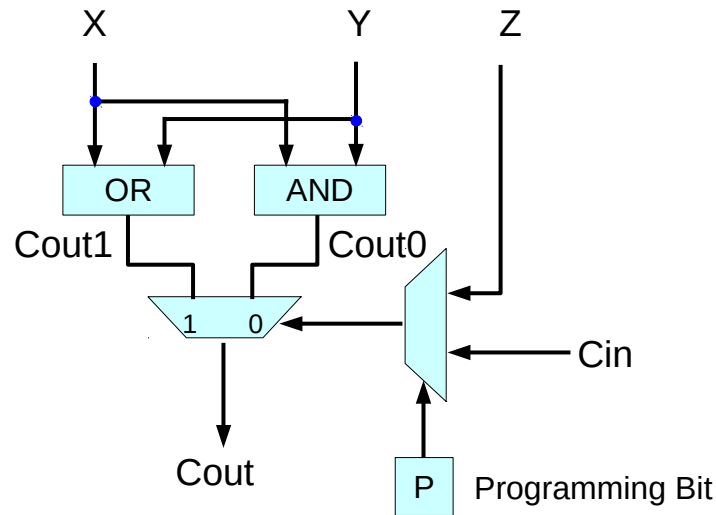
$$\text{Cin} (X + Y) + \overline{\text{Cin}} X Y$$

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

... P' : relaxed P

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

FPGA Carry Chain Cell

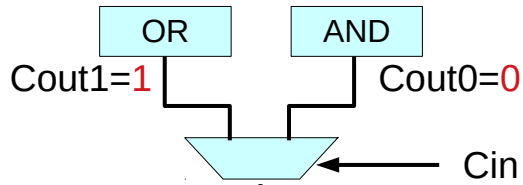


X	Y	Cin	$\overline{\text{Cin}}$	$\overline{X} \overline{Y}$
		Cout1	Cout0	
0	0	0	0	$\overline{X} \overline{Y}$
0	1	1	0	$\overline{X} Y$
1	0	1	0	$X \overline{Y}$
1	1	1	1	$X Y$

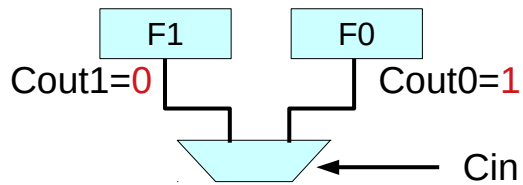
X	Y	Cin	Cout
0	0	0	Cout0
0	1	0	Cout0
1	0	0	Cout0
1	1	0	Cout0
0	0	1	Cout1
0	1	1	Cout1
1	0	1	Cout1
1	1	1	Cout1

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 $Cin=0$
 $Cout = Cin$



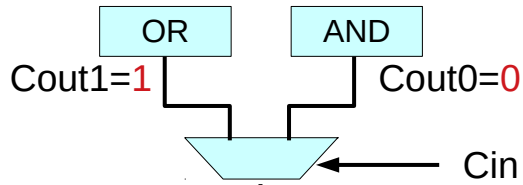
$Cout1=0$ when $Cin=1$
 $Cout0=1$ when $Cin=0$
 $Cout = \overline{Cin}$

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

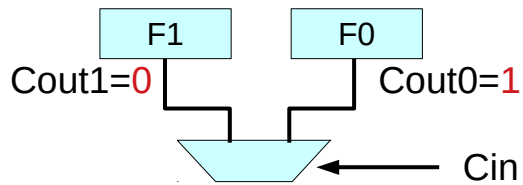
Cout1	Cout0	Cout	Name
0	0	0	Kill
0	1	\overline{Cin}	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

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

FPGA Carry Chain Cell



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



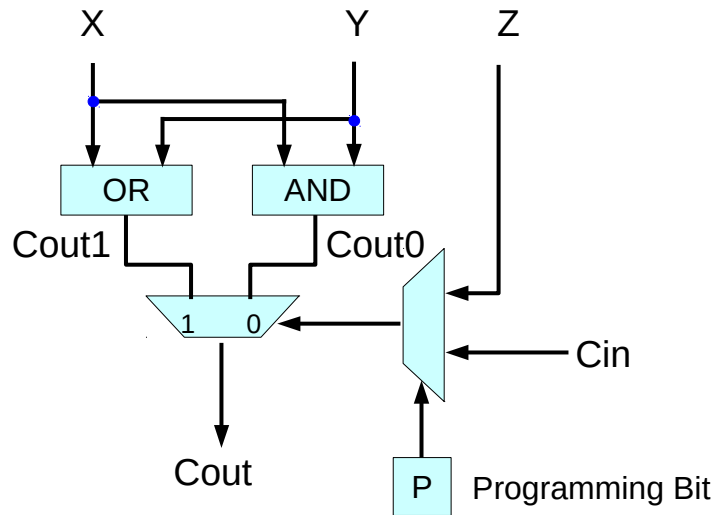
Cout1=0 when Cin=1
 Cout0=1 when Cin=0
 Cout = $\overline{\text{Cin}}$

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

X	Y	Cin	Cout		Cout1	Cout0
0	0	0	0	Cout0	0	0
0	1	0	0	Cout0	1	0
1	0	0	0	Cout0	1	0
1	1	0	1	Cout0	1	1
0	0	1	0	Cout1	0	0
0	1	1	1	Cout1	1	0
1	0	1	1	Cout1	1	0
1	1	1	1	Cout1	1	1

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

Carry Chain



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

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

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

Cout1=1 when Cin=1

Cout0=0 when Cin=0

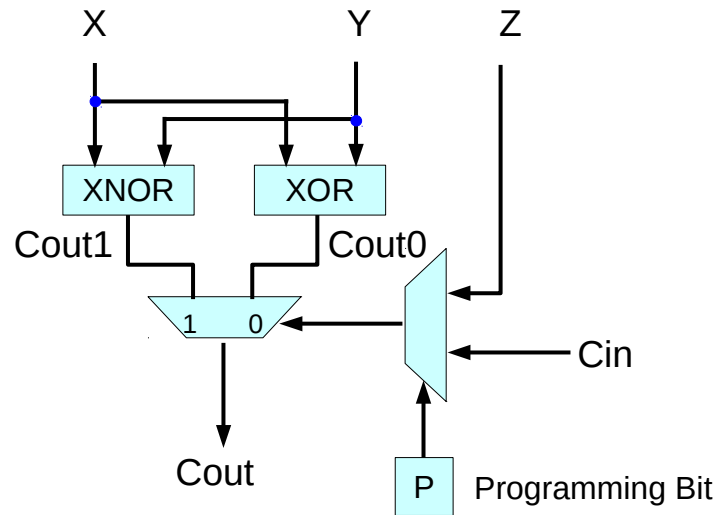
Cout = Cin propagate

Cout1=0 when Cin=1

Cout0=1 when Cin=0

Cout = $\overline{\text{Cin}}$ 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

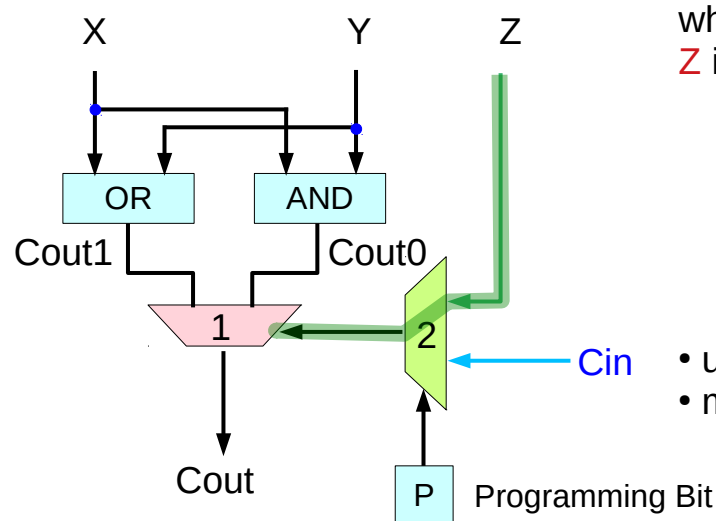
Ripple Carry Chain



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**

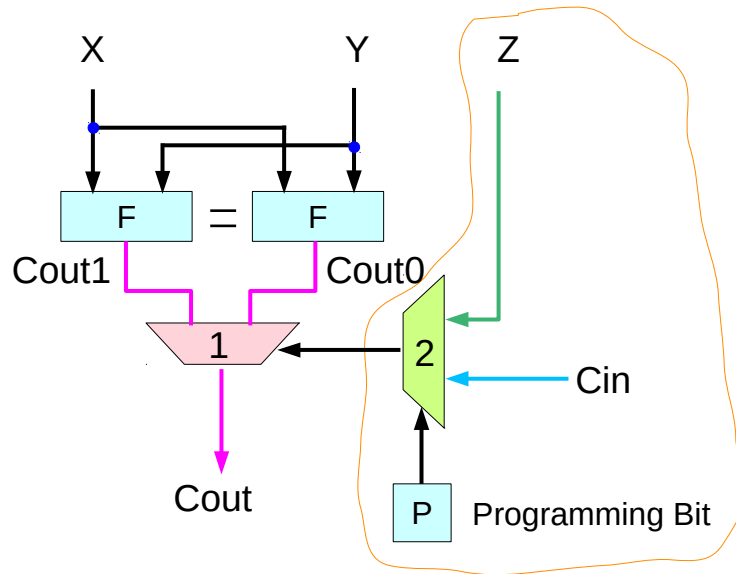
- used in **combined adder/subtractors**
- must be ignored, otherwise

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.

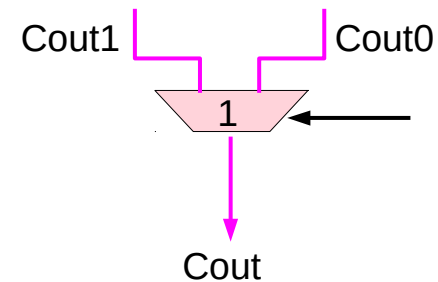
But in many carry computations,
the first cell has only 2 inputs,
and forcing the carry chain
to wait for the arrival of an additional,
unnecessary input **Z** will only needlessly
slow down the circuit's computation.

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

Ripple Carry Chain

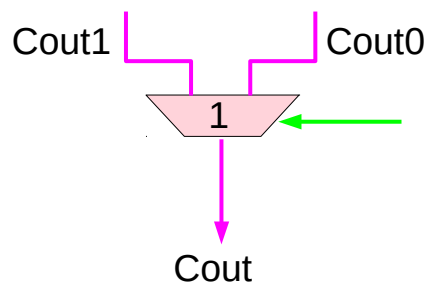
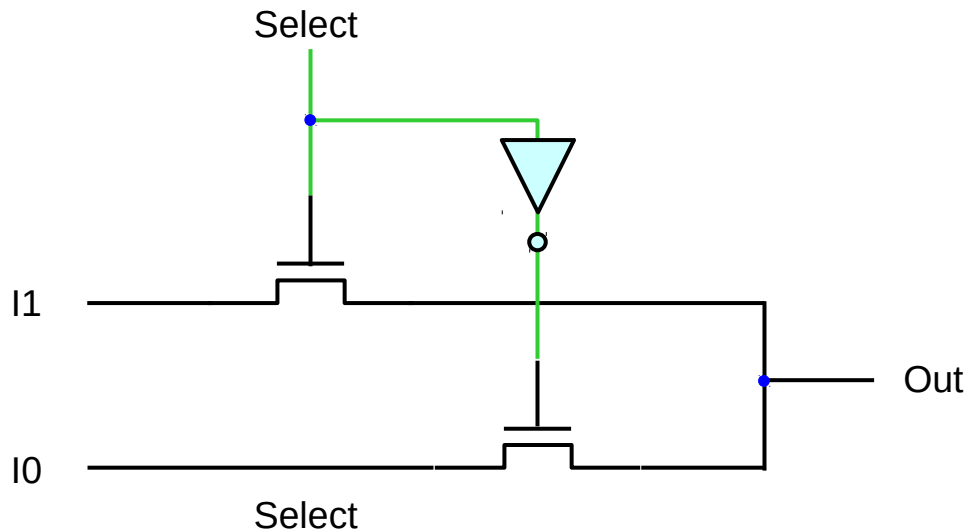


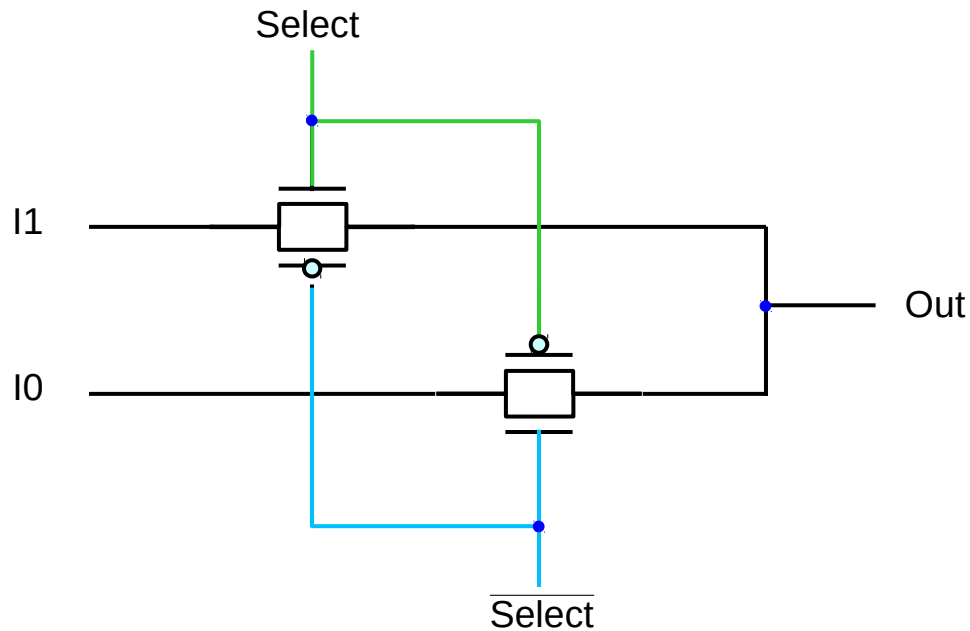
fig1b shows an implementation of a mux that does not obey this requirement

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.

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

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.

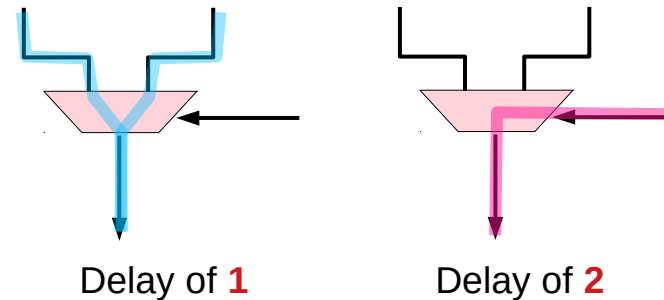
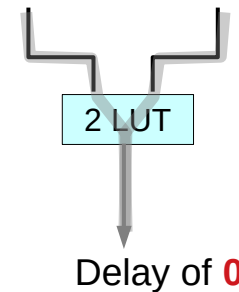
Delay of one

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

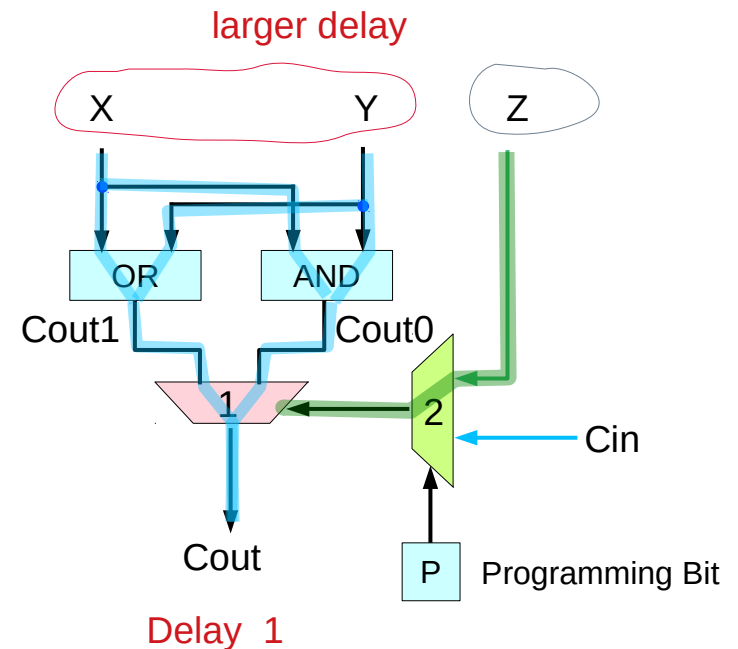
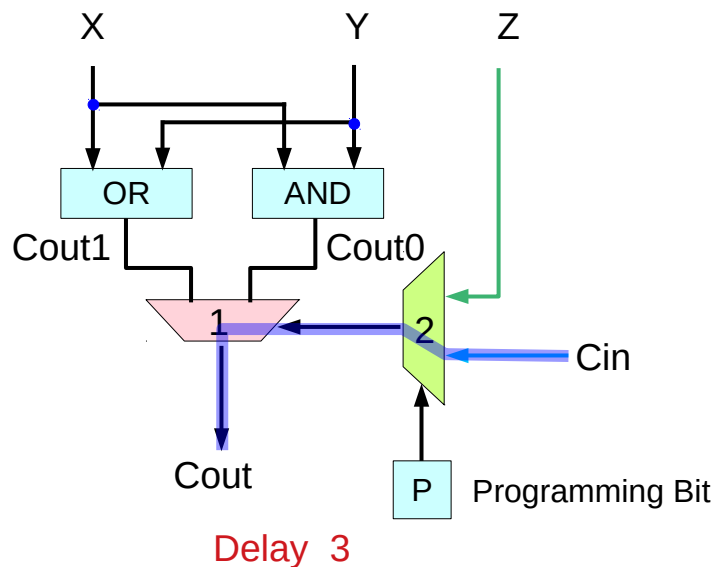
A **2:1 mux** has a **delay of one** from the I0 or I1 inputs to the output, But has a **delay of two** from the select input to the output due to the Inverter delay

Delay of zero (constant delay)

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



FPGA Carry Chain Cell



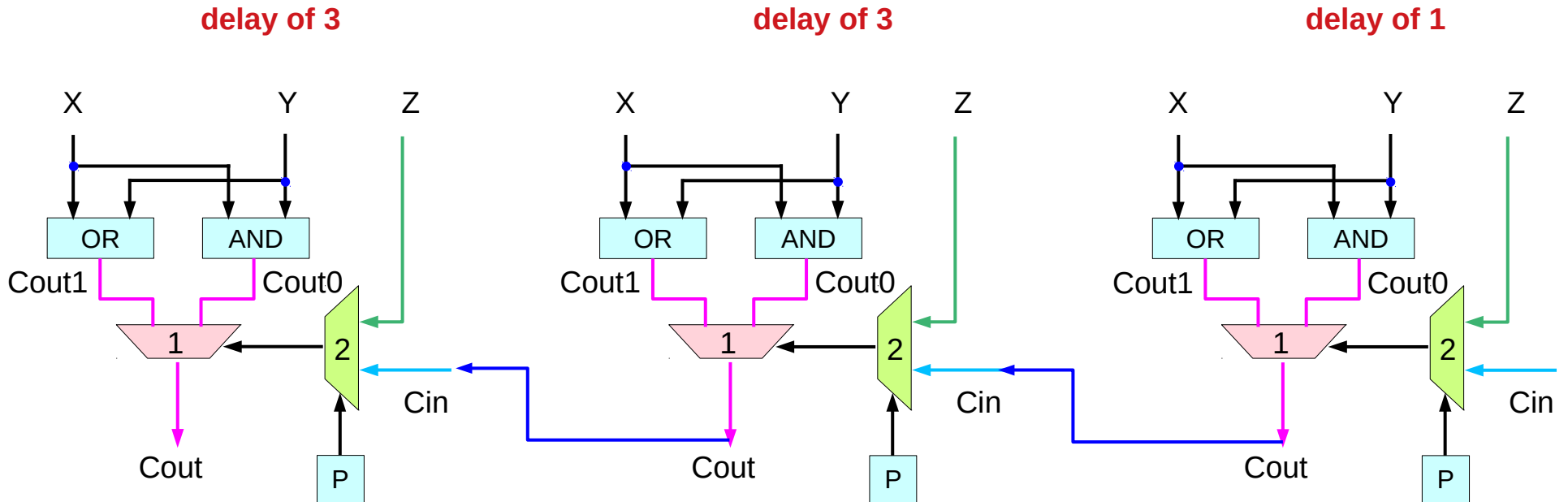
Significantly slower
 two muxes on the carry chain in each cell

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

FPGA Carry Chain Cell



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

the linear delay growth of ripple carry adders

optimize a ripple carry chain structure for use in FPGAs

while this provides some performance gain
over the basic ripple carry scheme
found in many current FPGAs,

still much slower than what is done in custom logic

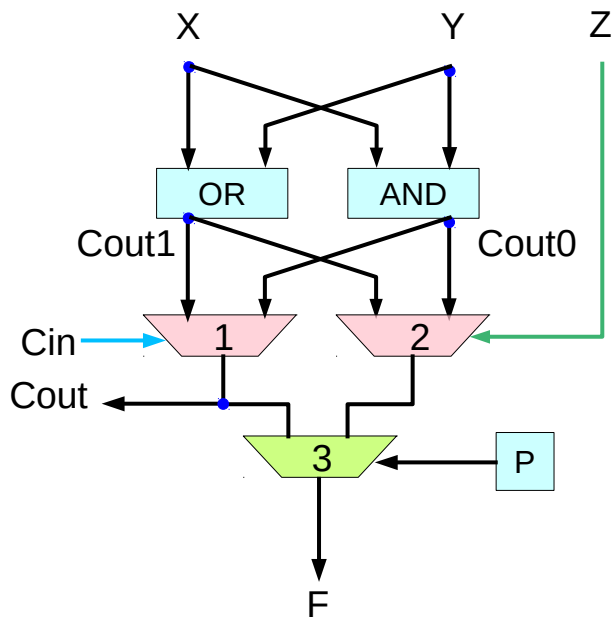
advanced adder techniques in custom logic
can be integrated into reconfigurable logic

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

FPGA Carry Chain Cell

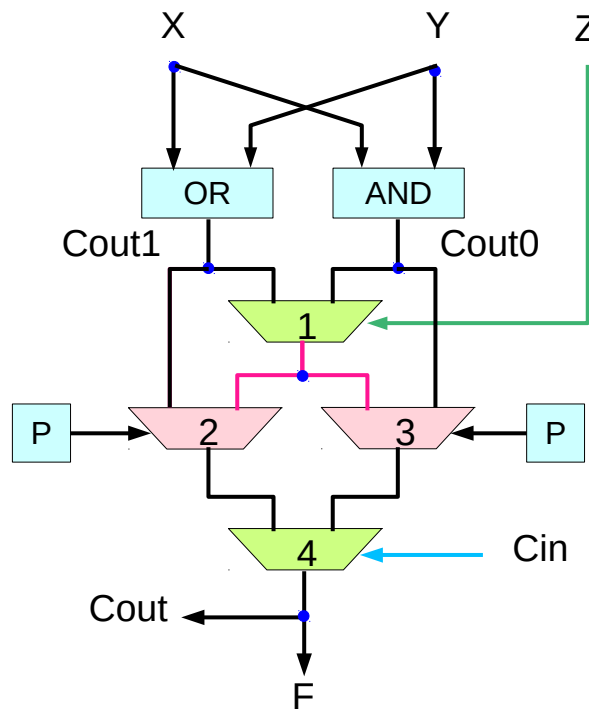
Design A

$$2n / 2n+2$$



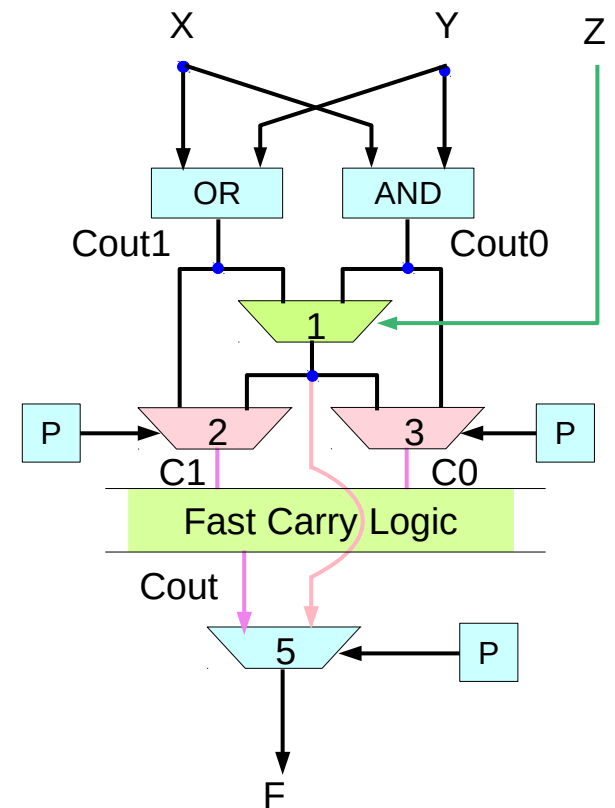
Design B

$$2n / 2n+1$$



Design C

$$2n+2$$



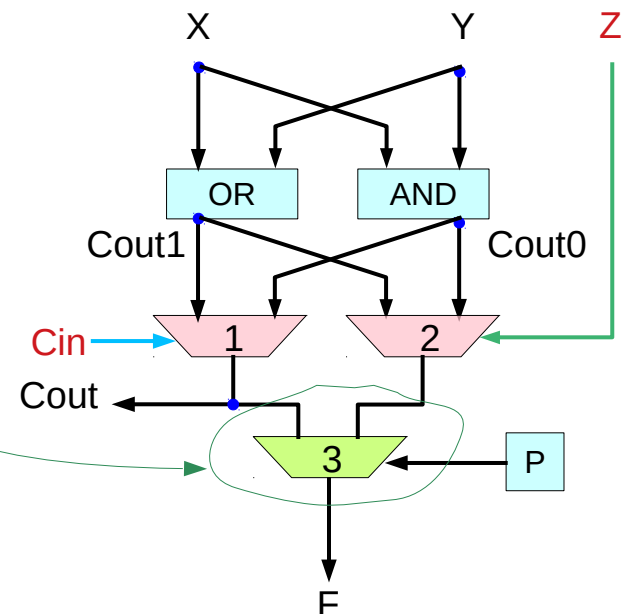
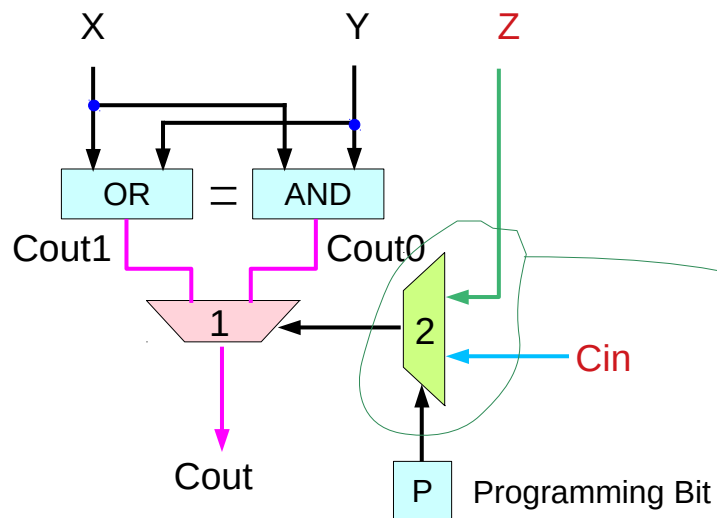
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design A

to reduce the delay of the ripple carry chain

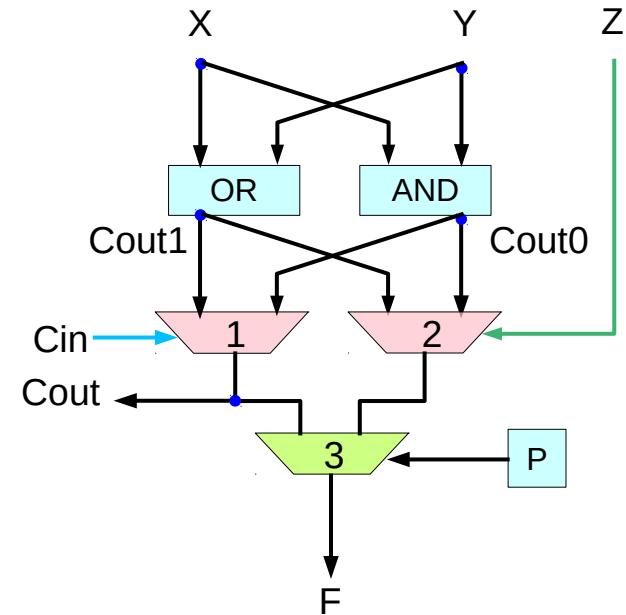
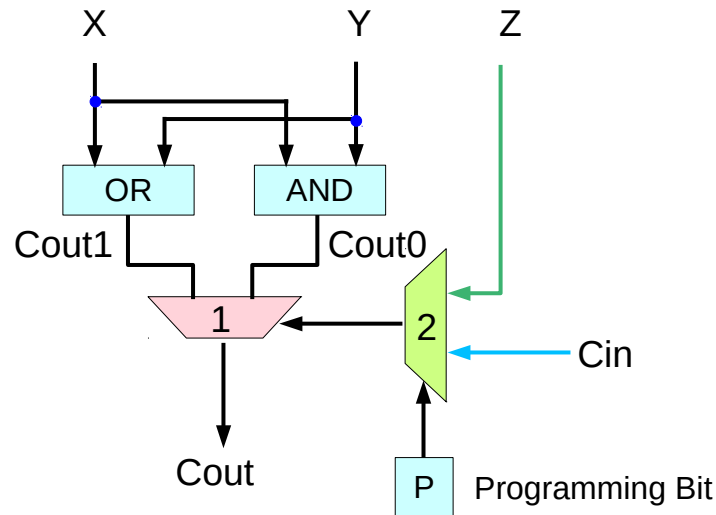
- remove **mux2** from the carry path.
- no need to choose between **Cin** and **Z** for the select line to the output **mux1**

- 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

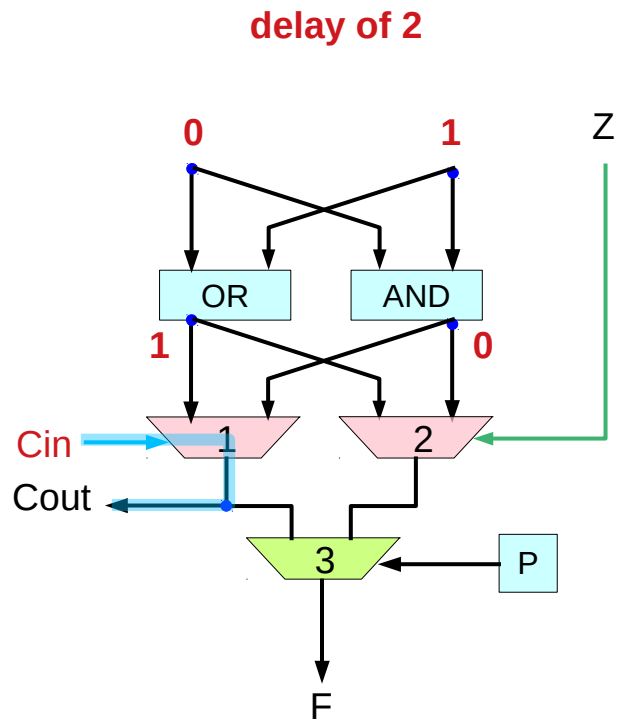
Design A



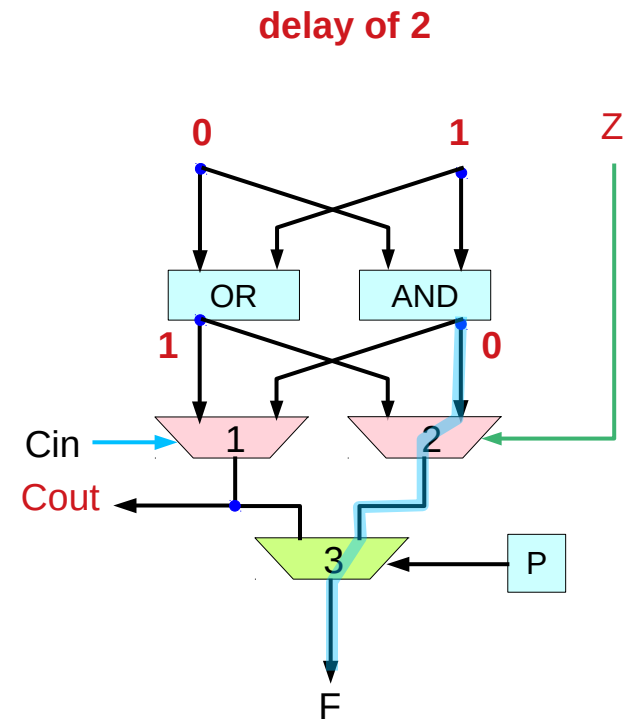
- not logically equivalent
- the **Z** input in the first cell cannot be used
 - **Z** is only attached to **mux2**
 - **mux2** does not lead to the carry cells
 - not connected to **Cout**

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

Design A



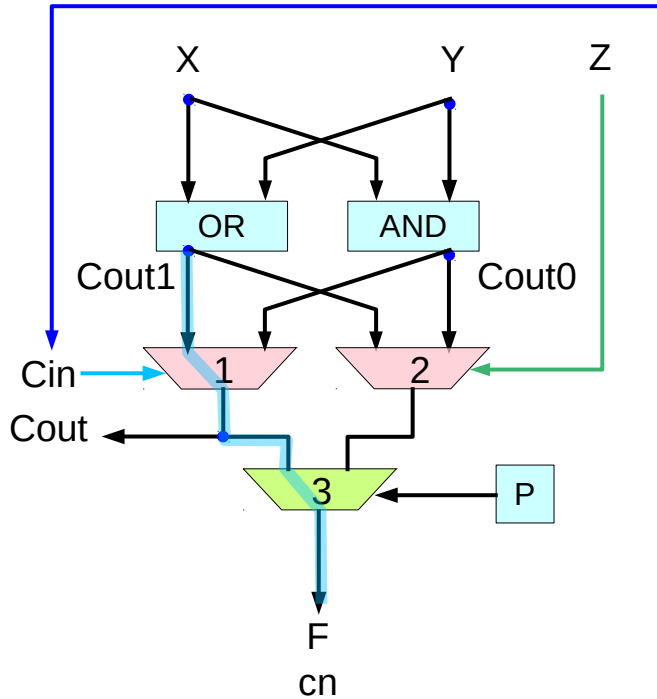
an additional cell
for generating Cin



- need an additional cell to use Z as a carry input

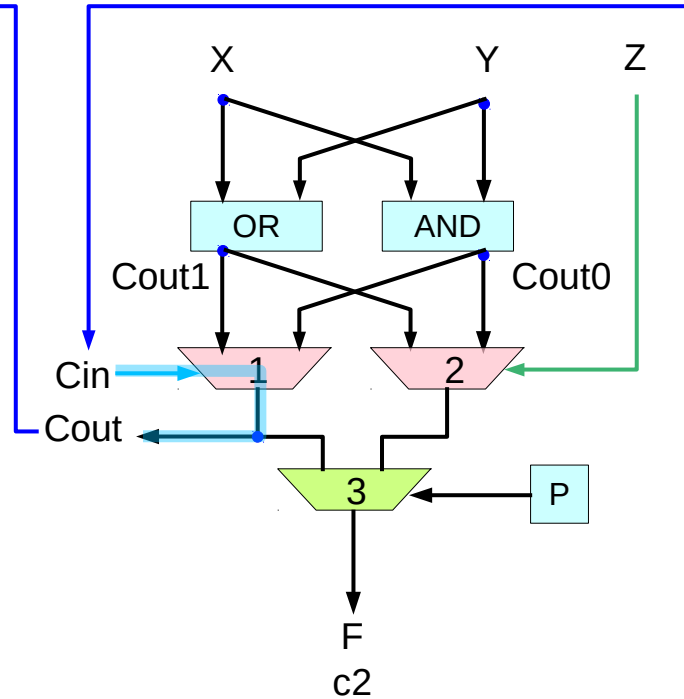
Design A

delay of 3



(2 for mux1, 1 for mux3)

delay of 2



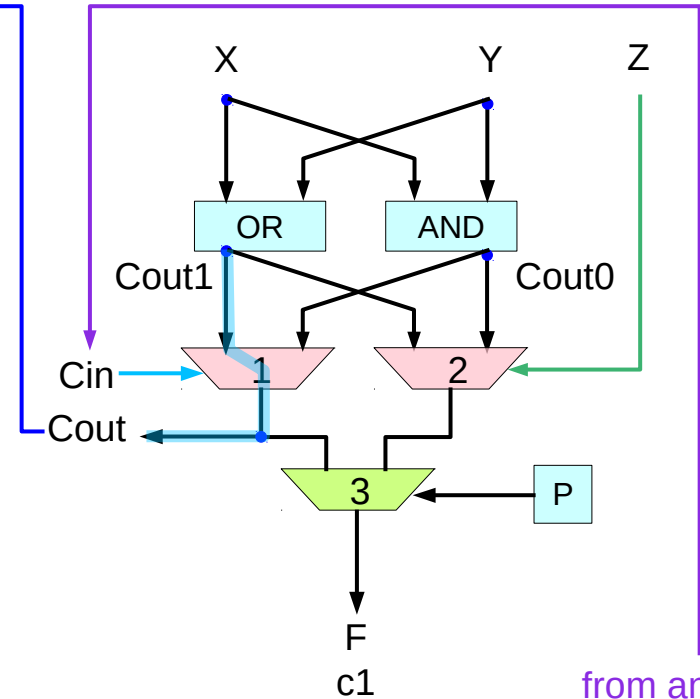
50% faster circuit than the original design

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

delay of $2(n+1)$

the first cell

delay of 1

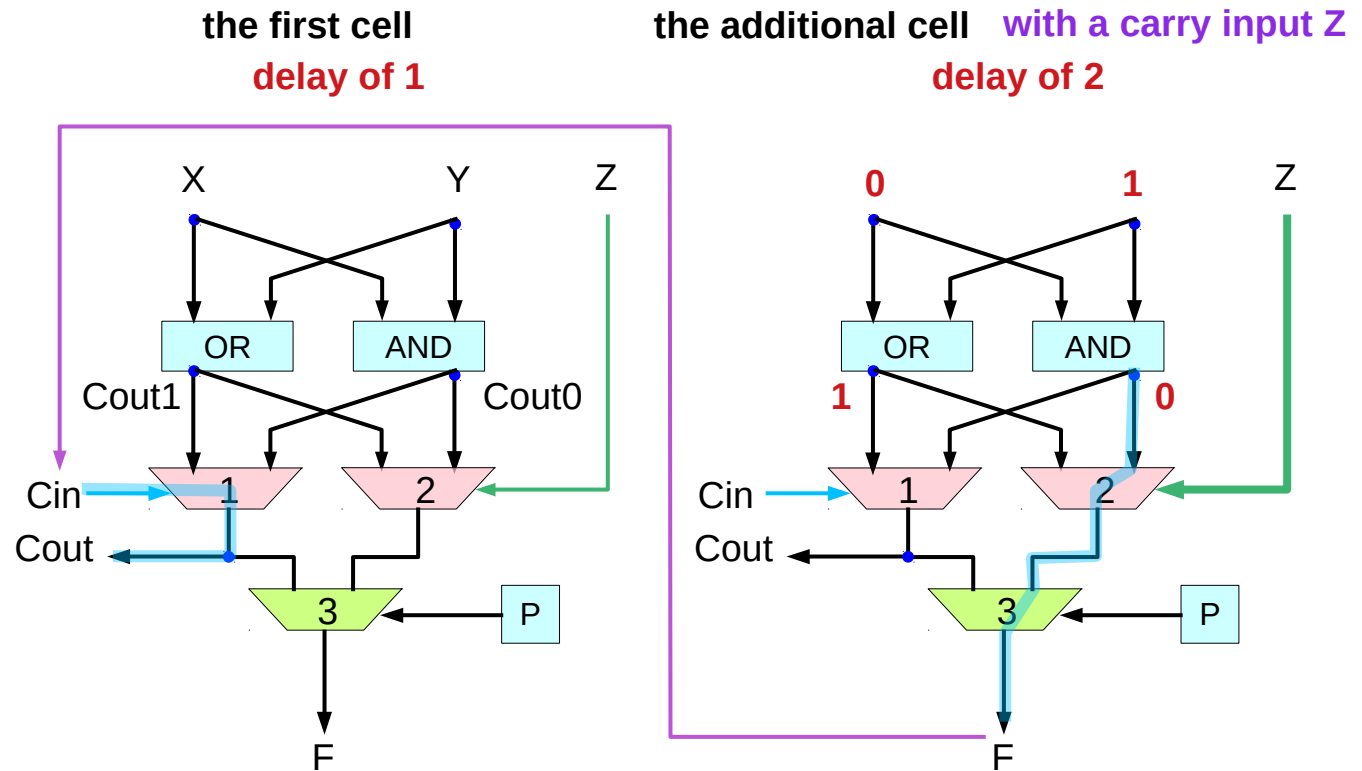


from an additional cell

without a carry input Z

with a carry input Z

Design A

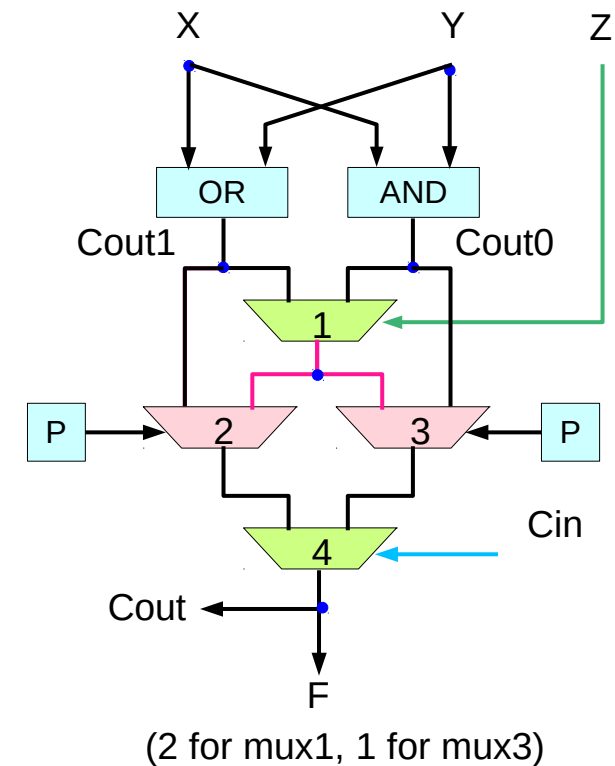


delay of $2(n+1)$ for an n -bit ripple carry chain with a carry input

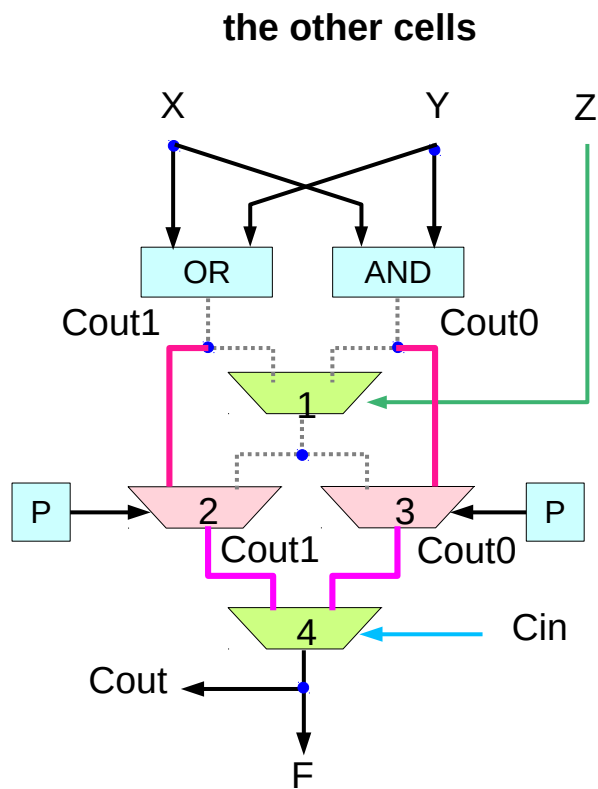
Design B

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.

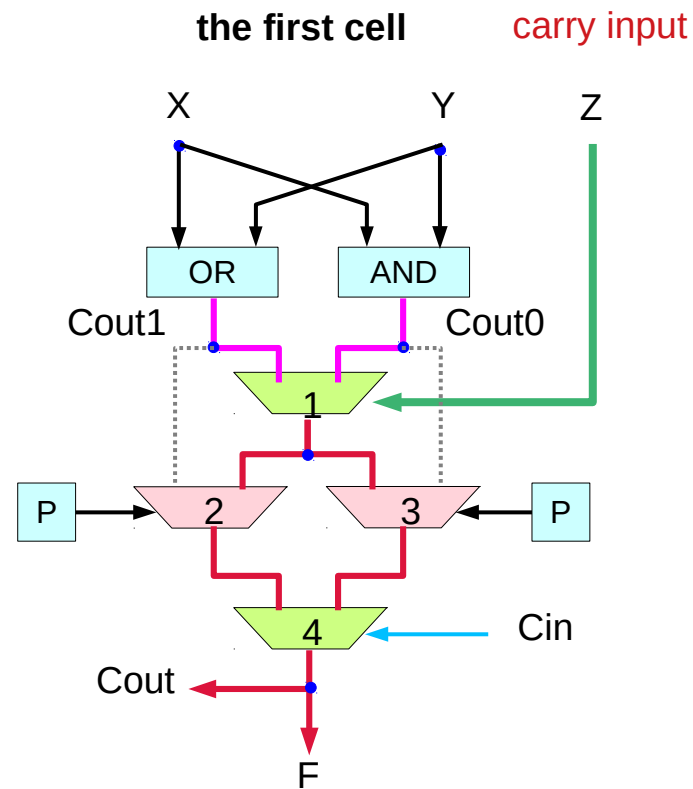
Also, for carry computations that do not need this feature, without a **carry input** the first cell in a **carry chain** built from fig 2b can be configured to bypass **mux1**, reducing the overall delay to $2n$, which is identical to that of fig2a.



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.

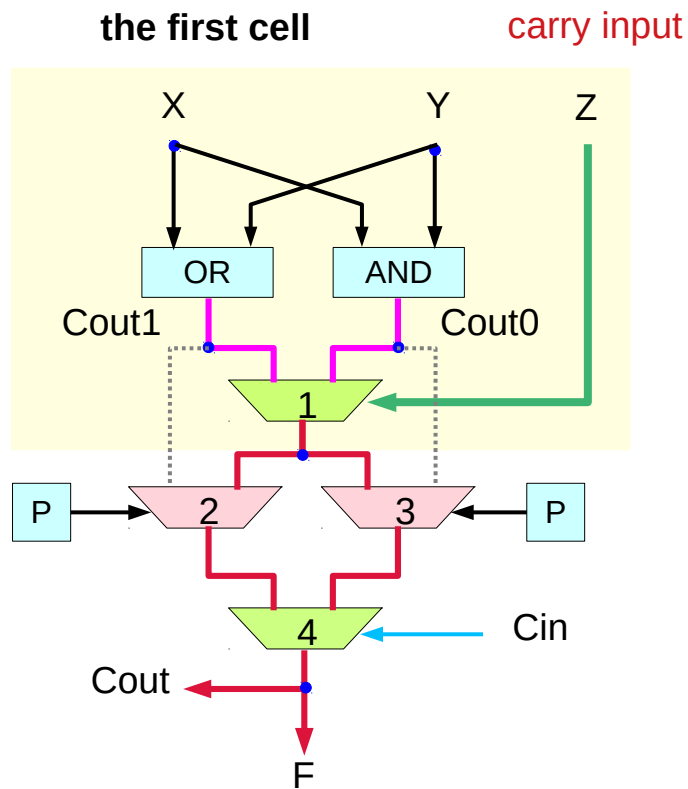


For the first cell in a carry chain
with a carry input (provided by input Z),
mux2 and mux3 both pass the value from mux1

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

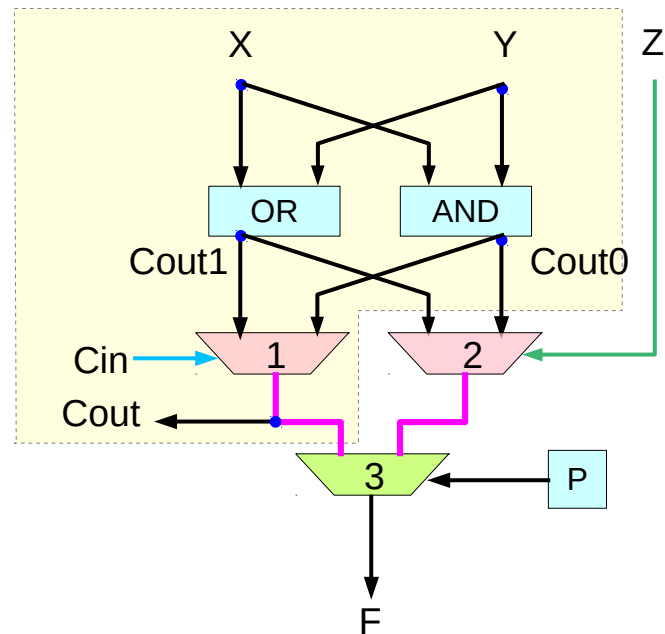
Design B



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

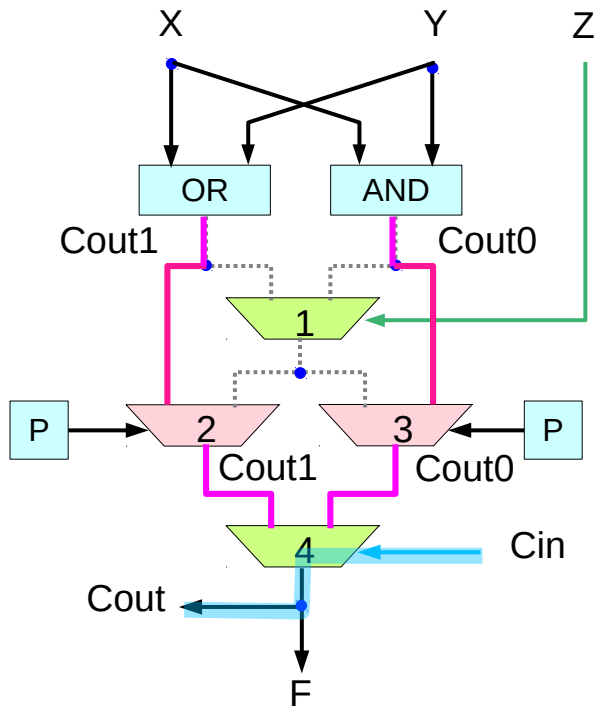
Functionally equivalent



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

Design B

delay of 2 the other cells

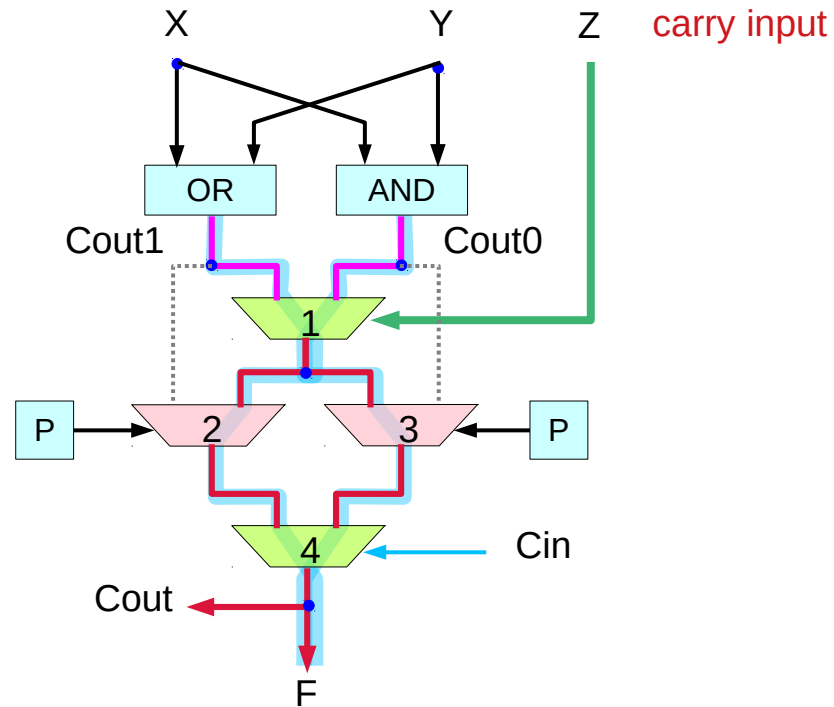


A delay of 2 in all other cells
except the first cell in the carry chain

an total delay of $2n+1$ for an n-bit carry chain
 when a carry input to the first cell is enabled

1 gate delay slower than that of fig 2a,

delay of 3 the first cell with a carry input

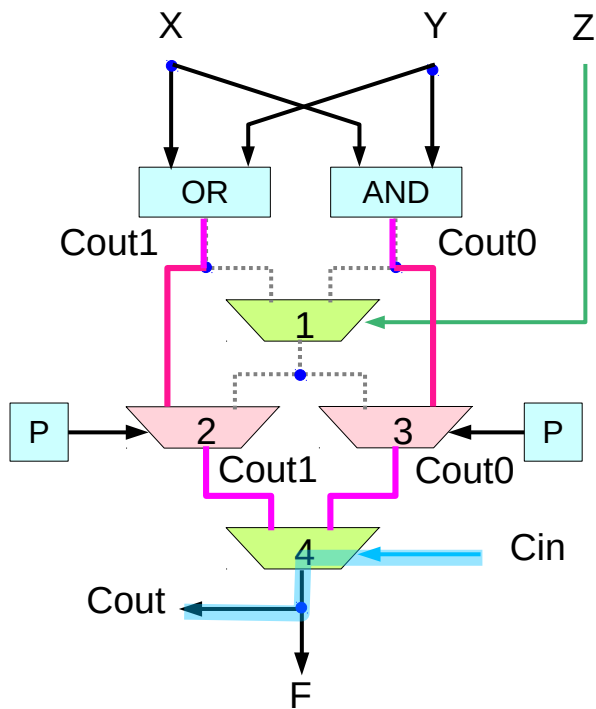


a delay of 3 in the first cell
 1 in mux1, 1 in mux2, 1 in mux4

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

Design B

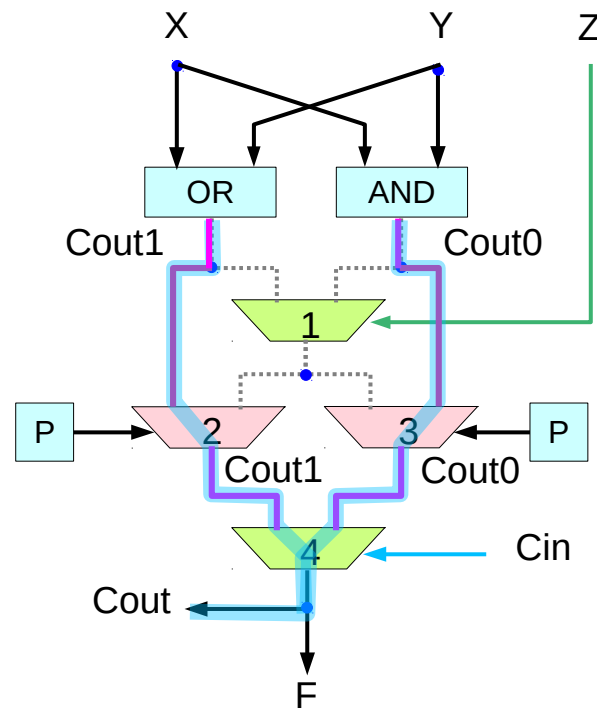
delay of 2 the other cells



A delay of 2 in all other cells
except the first cell in the carry chain

an total delay of **2n** for an n-bit carry chain
when a carry input to the first cell is **disabled**

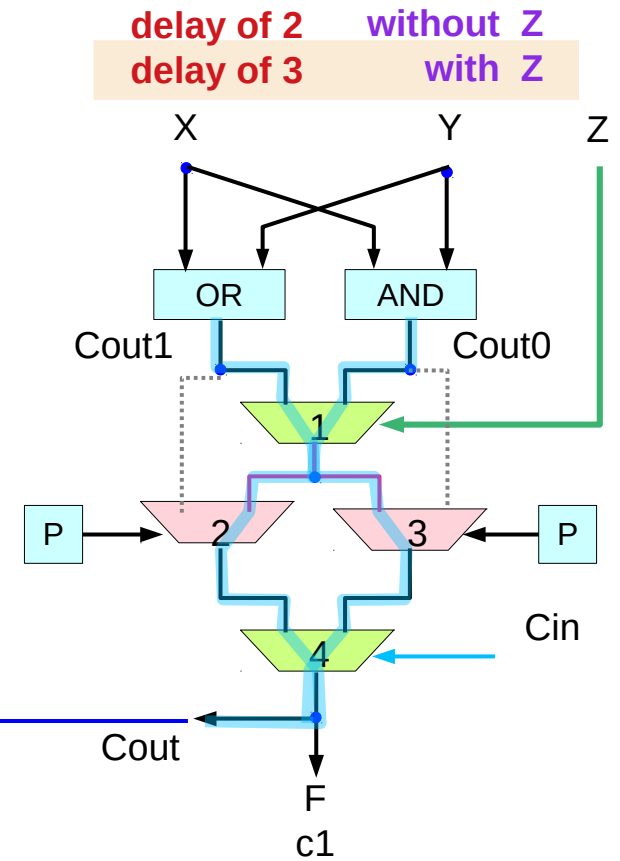
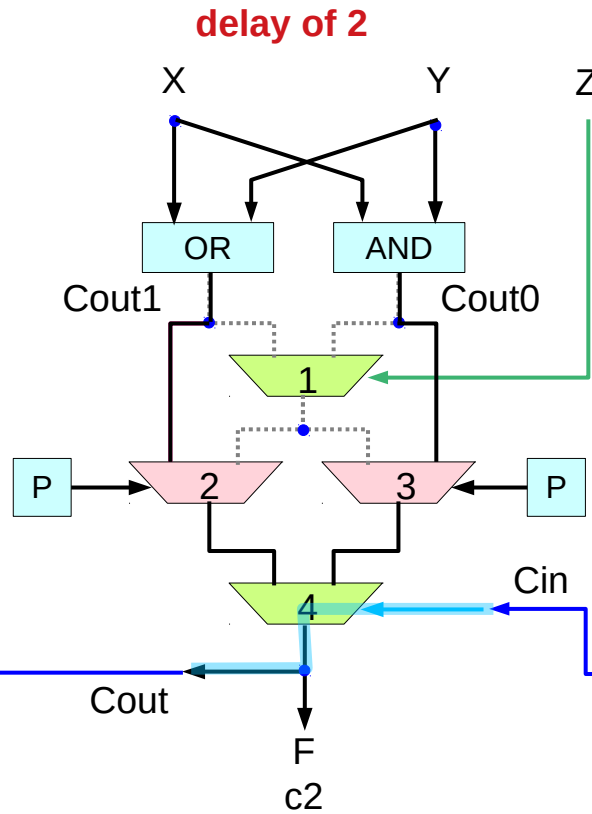
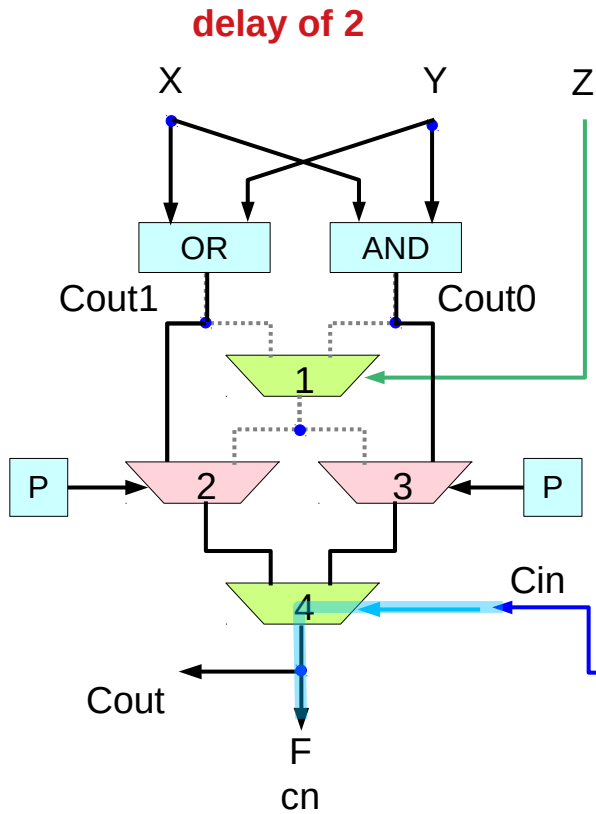
delay of 2 the first cell without a carry input



a delay of 2 in the first cell
when a carry input is not used

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

Design B



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

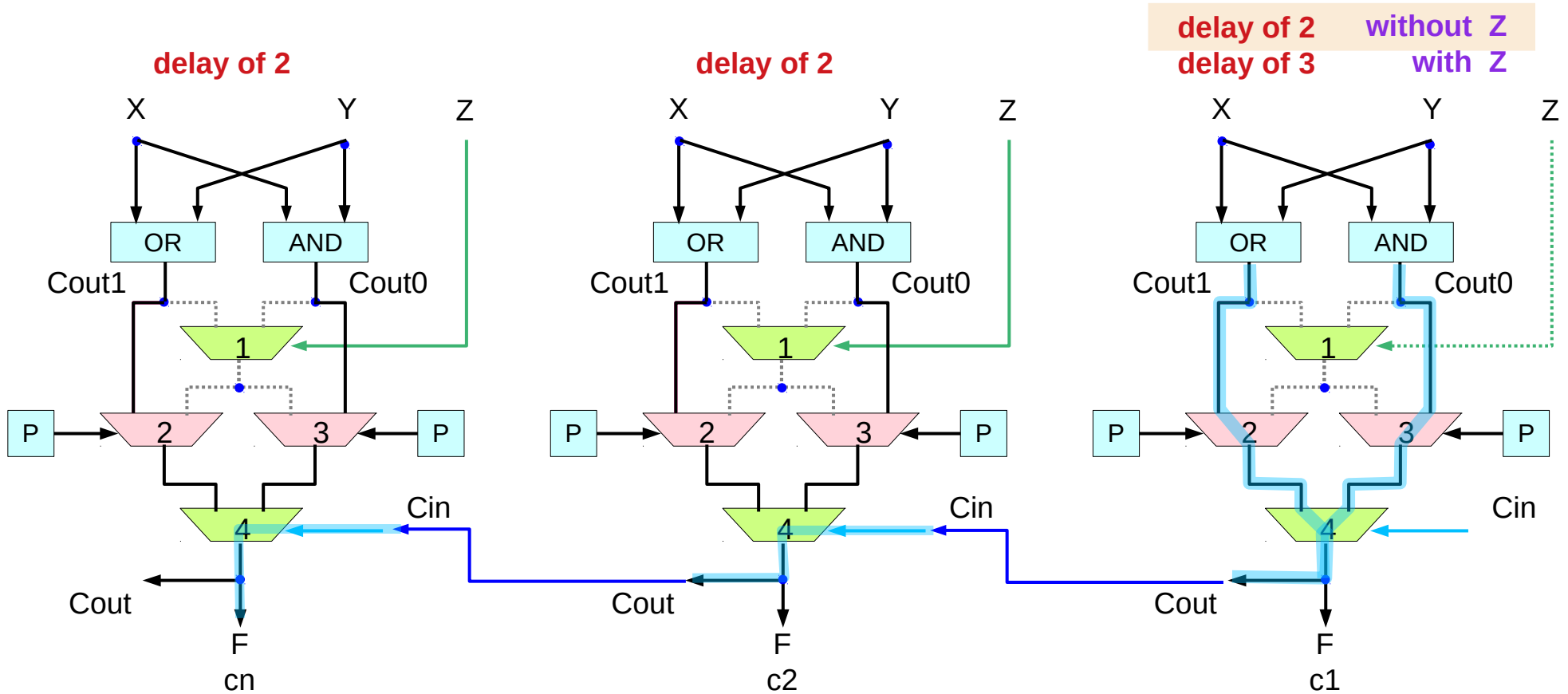
without a carry input Z

delay of $2n+1$

with a carry input Z

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

Design B



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

without a carry input Z

delay of $2n+1$

with a carry input Z

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

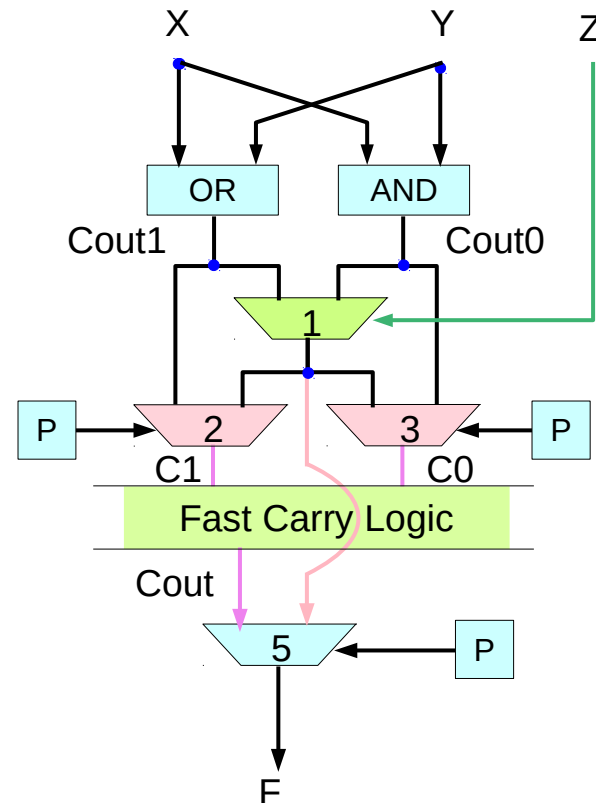
Design C (1)

various high performance carry chains can be developed based on the carry cell of Design C

very similar to Design B 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 **3 LUT**, using inputs X, Y, and Z **mux5** allows us to bypass the carry chain by selecting the output of **mux1**



Design C (1)

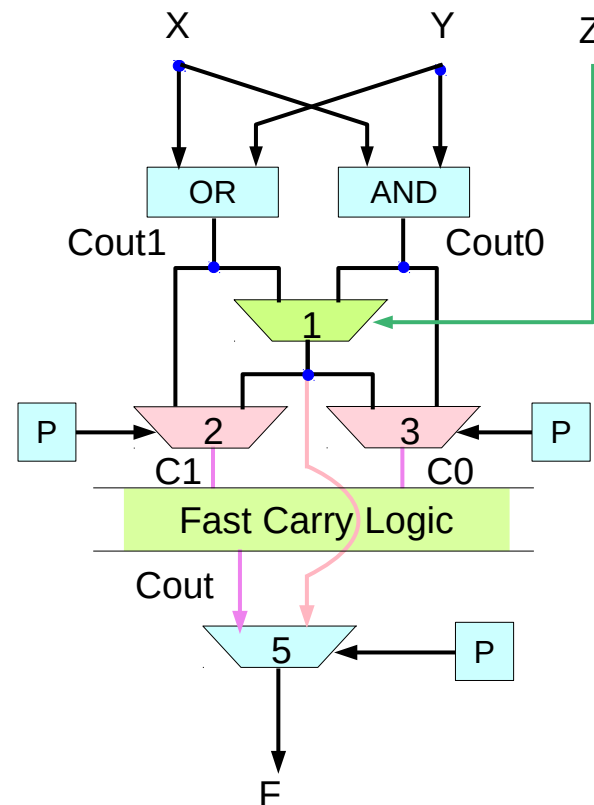
The important thing to realize about the logic of Design C is that any logic that can compute the value

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

where i is the position of the cell within the carry chain, can provide the functionality necessary to support the needs of FPGA computations

thus, the fast carry logic unit can contain any logic structure implementing this (including Brent-Kung), Variable Bit, and Ripple Carry.

Note that because of the needs and requirements of carry chains for FPGAs, we will have to develop new circuits, inspired by the standard adder structures, but which are more appropriate for FPGAs



Design C (2)

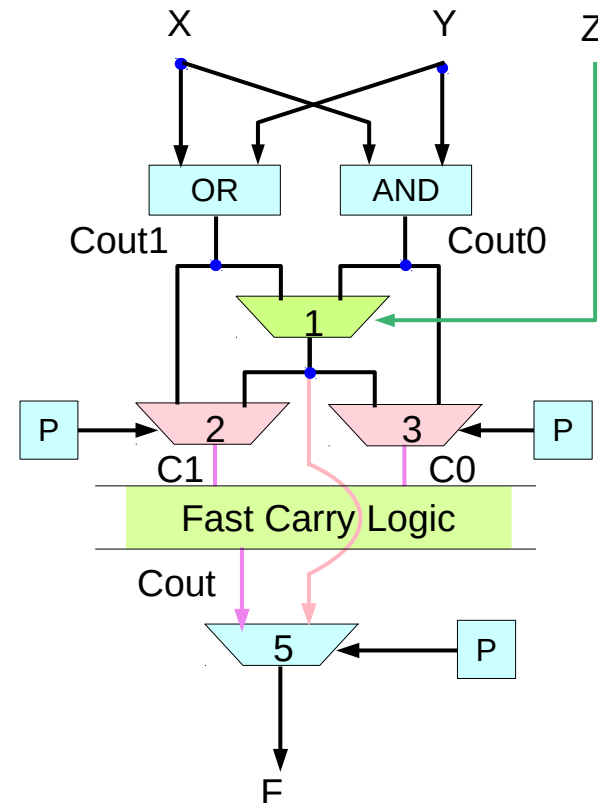
the main difference is to support all states

- Generate
- Propagate
- Kill
- Inverse Propagate

These 4 states are encoded on signals **C1** and **C0**

Also, while standard adders are concerned only with the maximum delay through an entire **n-bit adder** structure, the delay concerns for FPGAs are more complicated

Specifically, when an **n-bit carry chain** is built into the architecture of an FPGA it does not represent an actual computation, but only the potential for a computation.

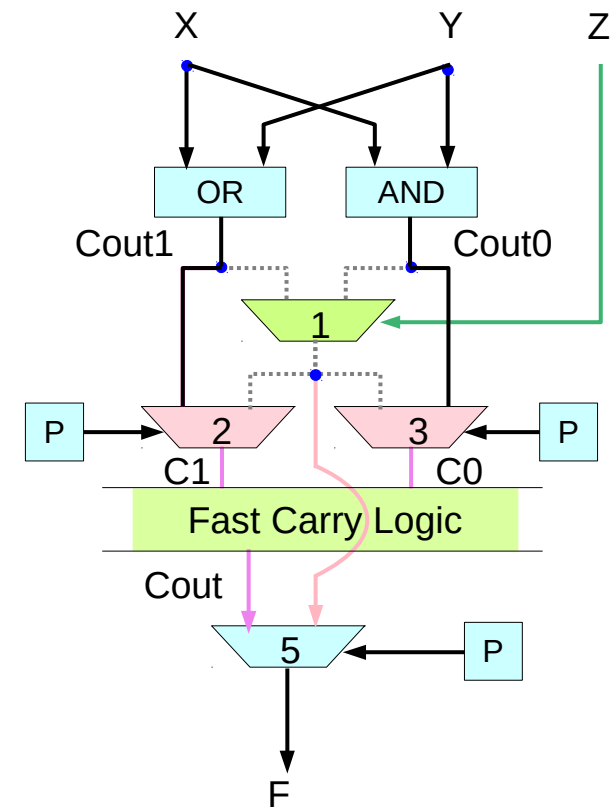
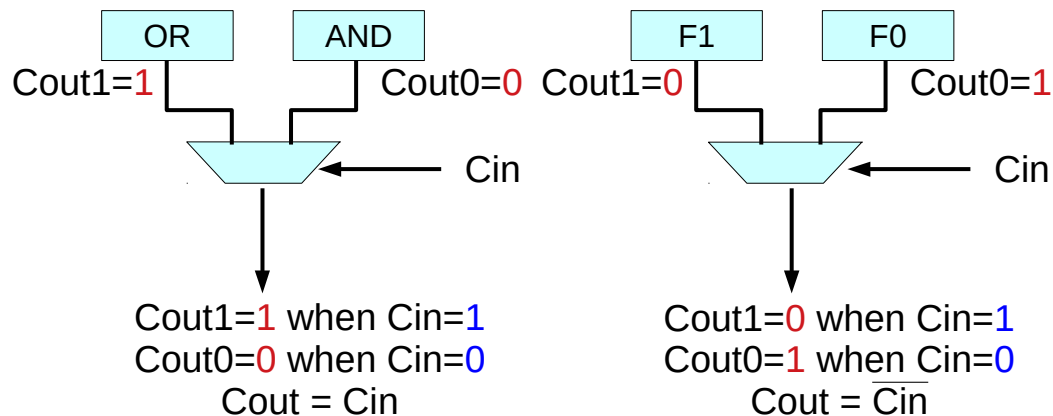


Design C (2)

X	Y	Cin	Cout0	Cout1
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

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

C1	C0	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

Design C (2)

X	Y	C1	C0	
0	0	0	0	$\bar{X}\bar{Y}$
0	1	1	0	$\bar{X}Y$
1	0	1	0	$X\bar{Y}$
1	1	1	1	XY

$$C1_i = X_i + Y_i$$

$$C0_i = X_i \cdot Y_i$$

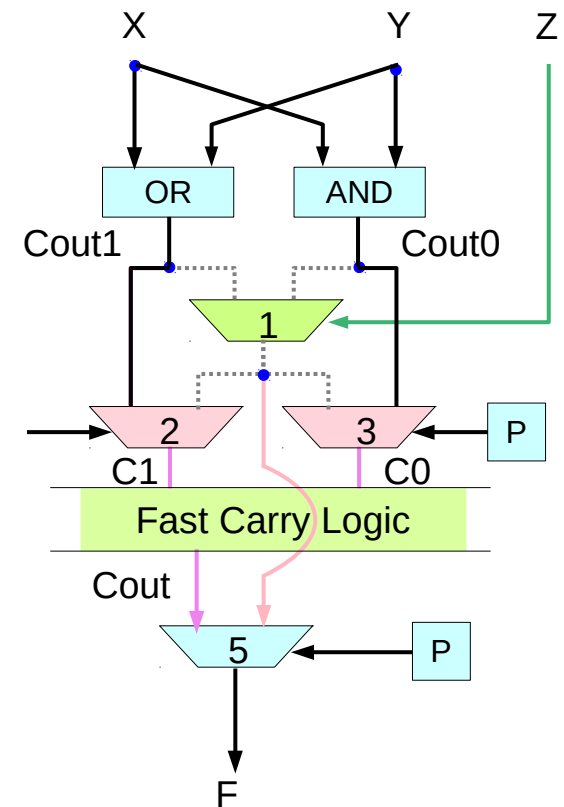
C1	C0		Name
0	0	0	Kill
0	1	\bar{Cin}	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

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

$$(Cout_{i-1} \cdot C1_i) = Cout_{i-1} \cdot (\bar{X}Y + X\bar{Y} + XY)$$

$$(\overline{Cout_{i-1}} \cdot C0_i) = \overline{Cout_{i-1}} \cdot XY$$

X	Y	Cout _i	Cout _{i+1}
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1



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

Generate and propagate

$$C1 = \bar{X}Y + X\bar{Y} + XY$$

X	Y	C1
0	0	0
0	1	1
1	0	1
1	1	1

$$C0 = XY$$

X	Y	C0
0	0	0
0	1	0
1	0	0
1	1	1

$$C1 = \bar{X}Y + X\bar{Y} + XY$$

$$C0 = XY$$

$$\bar{C1} = \overline{(\bar{X}Y) + (X\bar{Y}) + (XY)} = \bar{X}\bar{Y}$$

$$\bar{C0} = \bar{X} + \bar{Y} = \bar{X}Y + X\bar{Y} + \bar{X}\bar{Y}$$

$$\bar{C1} = \bar{X}\bar{Y}$$

X	Y	$\bar{C1}$
0	0	0
0	1	1
1	0	1
1	1	1

$$\bar{C0} = \bar{X}Y + X\bar{Y} + \bar{X}\bar{Y}$$

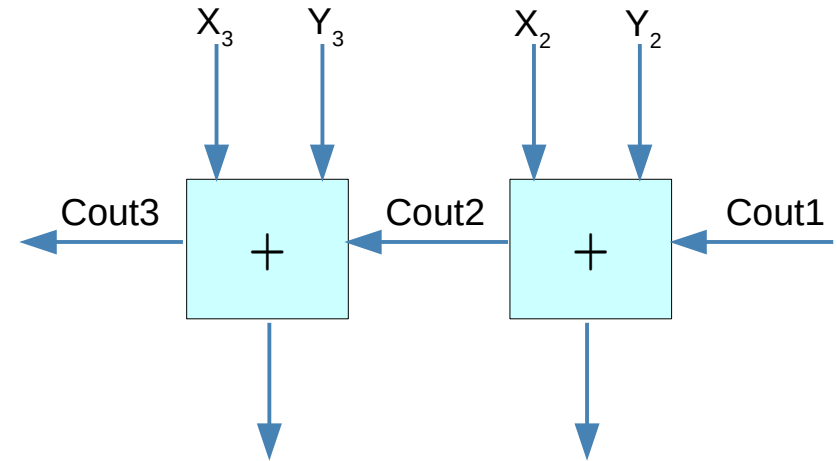
X	Y	$\bar{C0}$
0	0	0
0	1	1
1	0	1
1	1	1

$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \bar{C1}_2)) + (\bar{Cout}_1 \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \bar{C0}_2))$$

$$= (Cout_1 \cdot (C1_3 \cdot (\bar{X}_2Y_2 + X_2\bar{Y}_2 + X_2Y_2) + C0_3 \cdot \bar{X}_2\bar{Y}_2)) + (\bar{Cout}_1 \cdot (C1_3 \cdot X_2Y_2 + C0_3 \cdot (\bar{X}_2Y_2 + X_2\bar{Y}_2 + \bar{X}_2\bar{Y}_2)))$$

Cout3 in terms of Cout1

X_3	Y_3	X_2	Y_2	Cout2	Cout3	Cout3
0	0	0	0	0	0	0
0	0	0	1	Cout1	0	0
0	0	1	0	Cout1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	Cou2	0
0	1	0	1	Cout1	Cou2	Cout1
0	1	1	0	Cout1	Cou2	Cout1
0	1	1	1	1	Cou2	1
1	0	0	0	0	Cou2	0
1	0	0	1	Cout1	Cou2	Cout1
1	0	1	0	Cout1	Cou2	Cout1
1	0	1	1	1	Cou2	1
1	1	0	0	0	1	1
1	1	0	1	Cout1	1	1
1	1	1	0	Cout1	1	1
1	1	1	1	1	1	1



$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \overline{C1_2})) + (\overline{Cout_1} \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \overline{C0_2}))$$

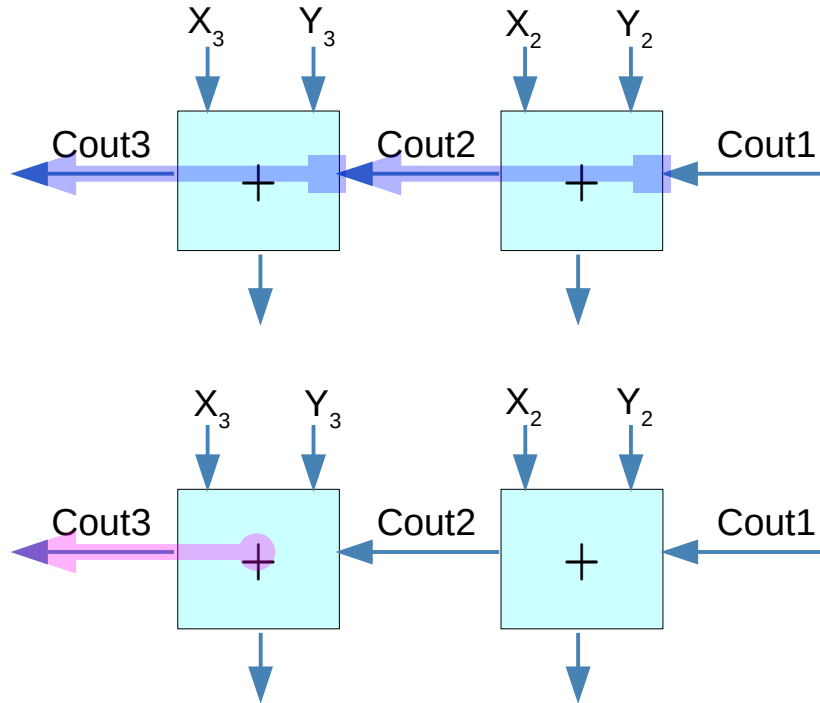
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Cout3 in terms of Generate and Propagate

X_3	Y_3	X_2	Y_2	$C1_3$	$C0_3$	$C1_2$	$C0_2$	$C1_3 C1_2$	$C0_3 \overline{C1_2}$	$C1_3 C0_2$	$C0_3 \overline{C0_2}$	Cout3
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0
0	0	1	1	0	0	1	1	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	1	0	1	0	1	0	0	0	Cout1
0	1	1	0	1	0	1	0	1	0	0	0	Cout1
0	1	1	1	1	0	1	1	1	0	1	0	1
1	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	0	1	0	0	0	Cout1
1	0	1	0	1	0	1	0	1	0	0	0	Cout1
1	0	1	1	1	0	1	1	1	0	1	0	1
1	1	0	0	1	1	0	0	0	1	0	1	1
1	1	0	1	1	1	1	0	1	0	0	1	1
1	1	1	0	1	1	1	0	1	0	0	1	1
1	1	1	1	1	1	1	1	1	0	1	0	1

$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \overline{C1_2})) + (\overline{Cout_1} \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \overline{C0_2}))$$

When Cout1 = 1



$$C 1_3 \cdot C 1_2 \cdot Cout_1$$

prop

prop

$$\overline{X}_3 Y_3$$

$$\overline{X}_2 Y_2$$

$$X_3 \overline{Y}_3$$

$$X_2 \overline{Y}_2$$

$$X_3 Y_3$$

$$X_2 Y_2$$

$$C 0_3 \cdot \overline{C 1_2} \cdot Cout_1$$

gen

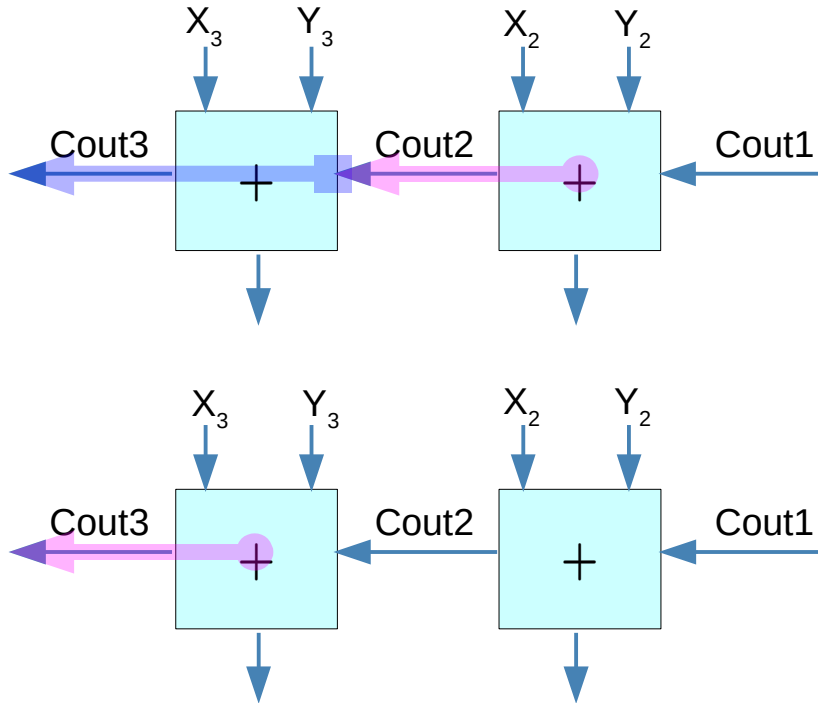
$\overline{\text{prop}}$

$$X_3 Y_3$$

$$\overline{X}_2 \overline{Y}_2$$

$$Cout_3 = (Cout_1 \cdot (C 1_3 \cdot C 1_2 + C 0_3 \cdot \overline{C 1_2})) + (\overline{Cout_1} \cdot (C 1_3 \cdot C 0_2 + C 0_3 \cdot \overline{C 0_2}))$$

When Cout1 = 0



$$C1_3 \cdot C0_2 \cdot \overline{Cout_1}$$

prop

gen

$$\overline{X_3}Y_3$$

$$X_2Y_2$$

$$X_3\overline{Y_3}$$

$$X_3Y_3$$

$$C0_3 \cdot \overline{C0_2} \cdot \overline{Cout_1}$$

gen

gen

$$X_3Y_3$$

$$\overline{X_2}Y_2$$

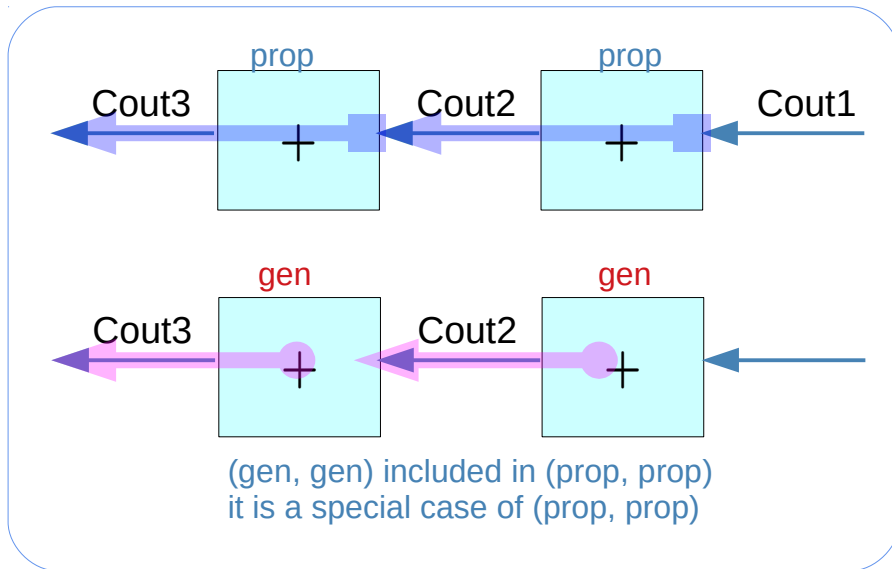
$$X_2\overline{Y_2}$$

$$X_2Y_2$$

$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \overline{C1_2})) + (\overline{Cout_1} \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \overline{C0_2}))$$

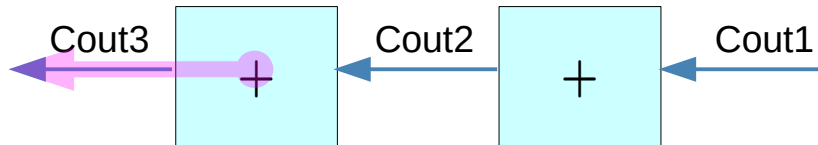
$$(C1_3 C1_2 + C0_3 \overline{C1_2})Cout_1 + (C1_3 C0_2 + C0_3 \overline{C0_2})\overline{Cout_1}$$

When Cout1 = 1



$$C 1_3 \cdot C 1_2 \cdot Cout_1$$

prop	prop
$\overline{X}_3 Y_3$	$\overline{X}_2 Y_2$
$X_3 \overline{Y}_3$	$X_2 \overline{Y}_2$
$X_3 Y_3$	$X_2 Y_2$

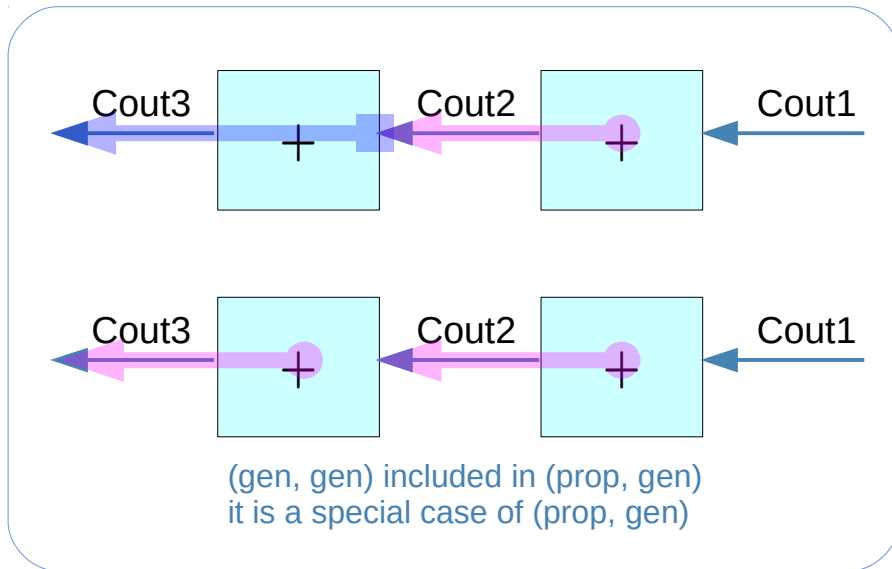


$$C 0_3 \cdot \overline{C 1_2} \cdot Cout_1$$

gen	prop
$X_3 Y_3$	$\overline{X_2 Y_2}$

$$Cout_3 = (Cout_1 \cdot (C 1_3 \cdot C 1_2 + C 0_3 \cdot \overline{C 1_2})) + (\overline{Cout_1} \cdot (C 1_3 \cdot C 0_2 + C 0_3 \cdot \overline{C 0_2}))$$

When Cout1 = 0



$$C1_3 \cdot C0_2 \cdot \overline{Cout_1}$$

prop

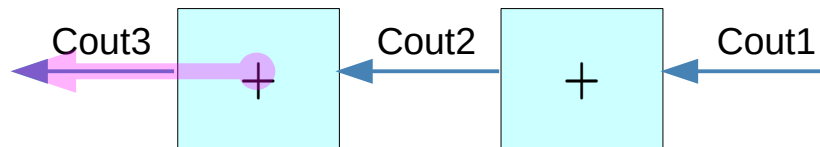
gen

$$\overline{X_3}Y_3$$

$$X_2Y_2$$

$$X_3\overline{Y_3}$$

$$X_3Y_3$$



$$C0_3 \cdot \overline{C0_2} \cdot \overline{Cout_1}$$

gen

gen

$$X_3Y_3$$

$$\overline{X_2}Y_2$$

$$X_2\overline{Y_2}$$

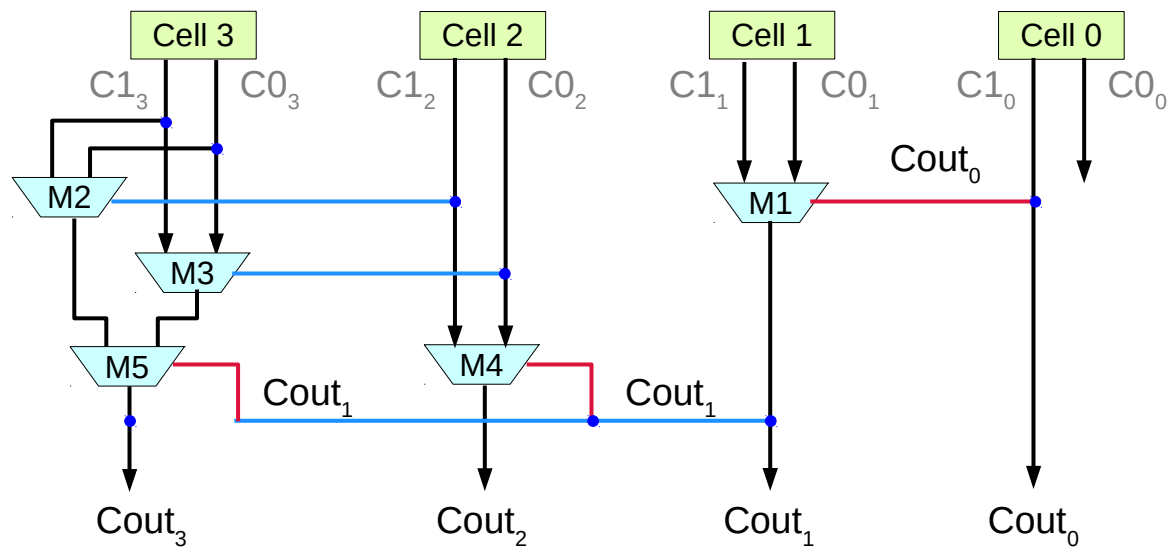
$$X_2Y_2$$

$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \overline{C1_2})) + (\overline{Cout_1} \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \overline{C0_2}))$$

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

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

Design C - Carry Select (1)



$$C1 = \bar{X}Y + X\bar{Y} + XY \quad C0 = XY$$

$$\bar{C1} = \bar{X}\bar{Y}$$

$$\bar{C0} = \bar{X}Y + X\bar{Y} + \bar{X}\bar{Y}$$

$$Cout_2 = (Cout_1 \cdot C1_2) + (\bar{Cout}_1 \cdot C0_2) \quad Cout_1 = (Cout_0 \cdot C1_1) + (\bar{Cout}_0 \cdot C0_1)$$

$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \bar{C1}_2)) + (\bar{Cout}_1 \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \bar{C0}_2)) = (Cout_1 \cdot (C1_3 \cdot (\bar{X}_2Y_2 + X_2\bar{Y}_2 + X_2Y_2) + C0_3 \cdot \bar{X}_2\bar{Y}_2)) + (\bar{Cout}_1 \cdot (C1_3 \cdot X_2Y_2 + C0_3 \cdot (\bar{X}_2Y_2 + X_2\bar{Y}_2 + \bar{X}_2\bar{Y}_2)))$$

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

Design C - Carry Select (1)

$$C1 = \bar{X}Y + X\bar{Y} + XY$$

$$C0 = XY$$

$$\bar{C1} = \bar{X}\bar{Y}$$

$$\bar{C0} = \bar{X}Y + X\bar{Y} + \bar{X}\bar{Y}$$

C1	C0	Name
0	0	0 Kill
0	1	\bar{Cin} Inverse Propagate
1	0	Cin Propagate
1	1	1 Generate

$$Cout_1 = (Cout_0 \cdot C1_1) + (\bar{Cout}_0 \cdot C0_1)$$

$$Cout_2 = (Cout_1 \cdot C1_2) + (\bar{Cout}_1 \cdot C0_2)$$

$$Cout_3 = (Cout_2 \cdot C1_3) + (\bar{Cout}_2 \cdot C0_3)$$

$$Cout_3 = (Cout_1 \cdot (C1_3 \cdot C1_2 + C0_3 \cdot \bar{C1}_2)) + (\bar{Cout}_1 \cdot (C1_3 \cdot C0_2 + C0_3 \cdot \bar{C0}_2))$$

$$= Cout_1 \cdot [(\bar{X}Y + X\bar{Y} + XY)_3 \cdot (\bar{X}Y + X\bar{Y} + XY)_2 + (XY)_3 \cdot (XY)_2] + \bar{Cout}_1 \cdot [(\bar{X}Y + X\bar{Y} + XY)_3 \cdot (XY)_2 + (XY)_3 \cdot (\bar{X}Y + X\bar{Y} + \bar{X}\bar{Y})_2]$$

$$= (Cout_1 \cdot (C1_3 \cdot (\bar{X}_2Y_2 + X_2\bar{Y}_2 + X_2Y_2) + C0_3 \cdot \bar{X}_2\bar{Y}_2)) + (\bar{Cout}_1 \cdot (C1_3 \cdot X_2Y_2 + C0_3 \cdot (\bar{X}_2Y_2 + X_2\bar{Y}_2 + \bar{X}_2\bar{Y}_2)))$$

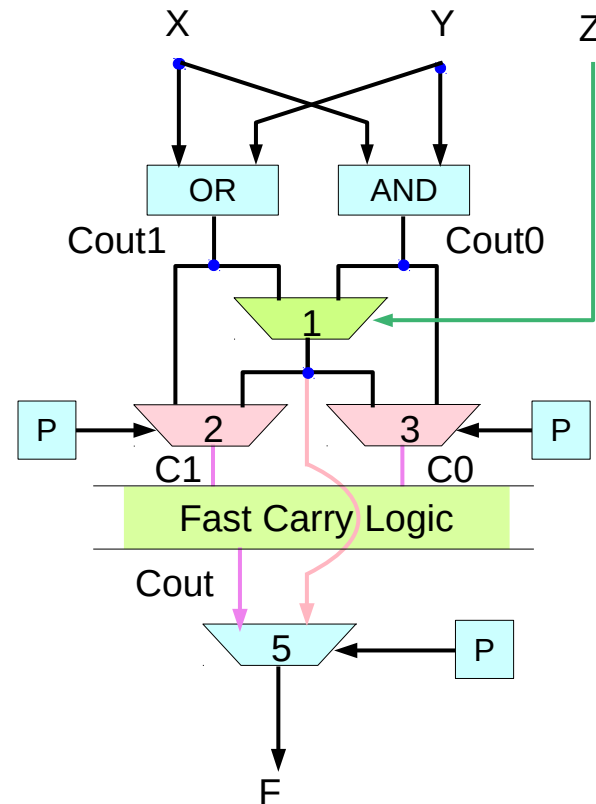
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design C (3)

A carry chain resource may span the entire height of a **column** in the FPGA, but a mapping to the logic may use only a small portion of this chain, with the carry logic in the mapping starting and ending at arbitrary points in the **column**

concerned with not just the **carry delay** from the first to the last position in a carry chain, but must consider the delay for a **carry computation** beginning at any point within this **column**.

For example, even though the FPGA architecture may provide support for **carry chains** of up to **32 bits**, it must also efficiently support **8 bit** carry computations placed at any point within this carry chain resource



Design C (4)

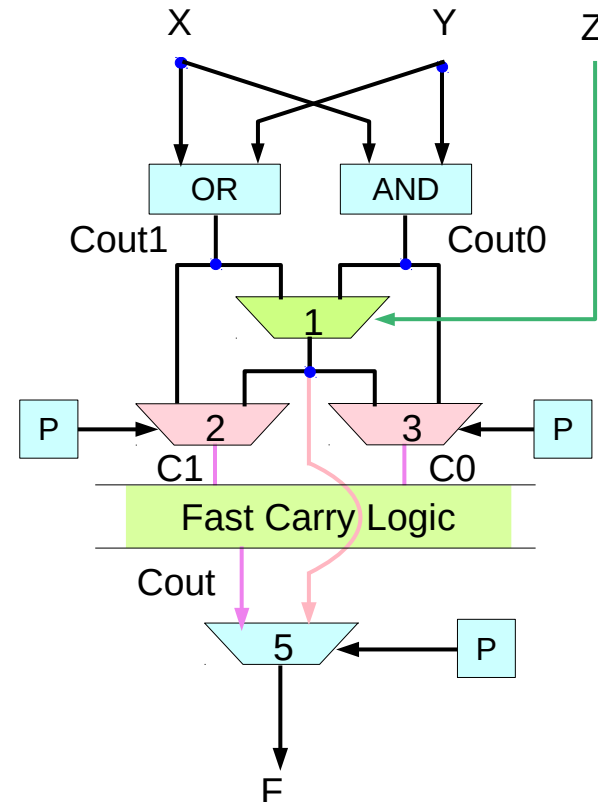
Carry Select

the problem with a **ripple carry** structure is that the computation of the Cout for bit position i cannot begin until after the computation has been completed in bit positions $0 \dots i-1$

A carry select structure overcomes this limitation

the main observation is that for any bit position, the only information it received from the previous bit positions is its Cin signal, which can be either **true** or **false**.

In a **carry select adder** the **carry chain** is broken at a specific **column**, and two **separate additions** occur



Design C (5)

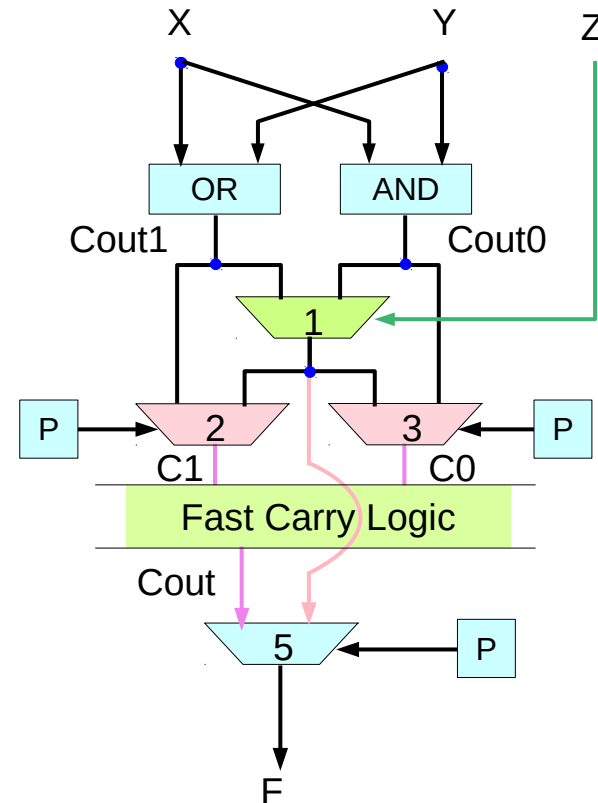
one assuming the **Cin** signal is **true**,
the other assuming it is **false**

These computations can take place **before**
the **previous columns** complete their operation
since they do not depend on the actual value of
the **Cin** signal

This **Cin** signal is instead used to determine
which adder's outputs should be used

if the **Cin** signal is **true**, the output of the following
stages comes from the adder that assumed
that the **Cin** would be **true**

likewise, a **false Cin** chooses the other adder's output

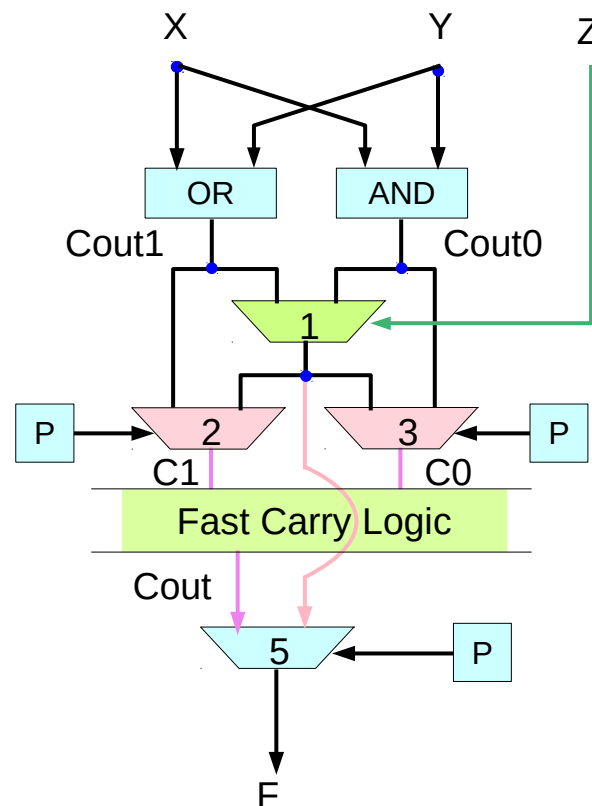


Design C (6)

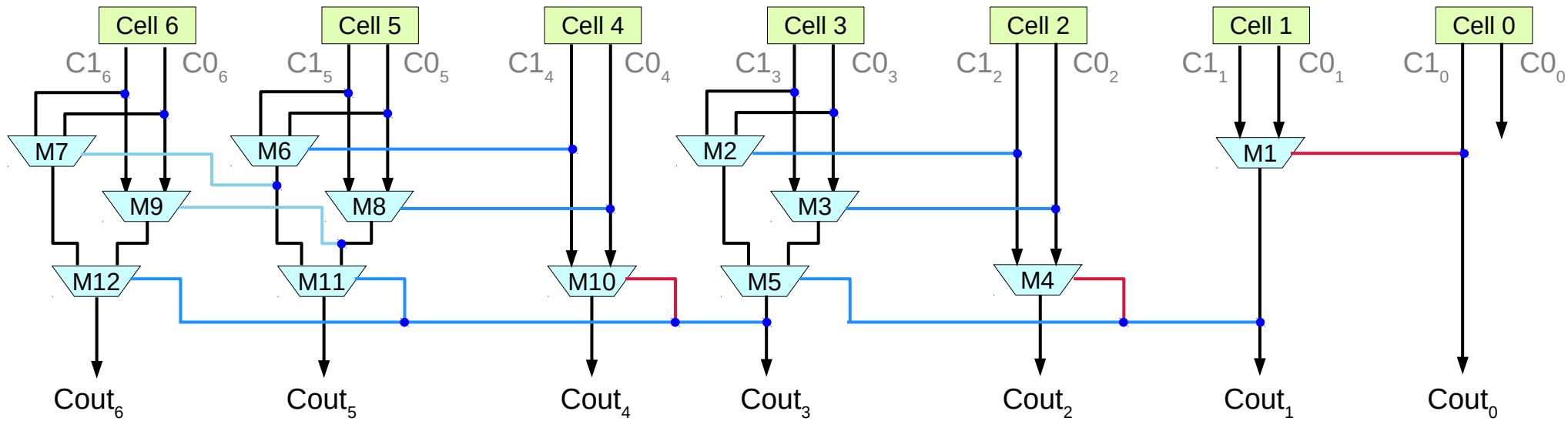
This splitting of the **carry chain** can be done multiple times, breaking the computation into several pairs of short adders with **output muxes** choosing which adder's output to select

the length of the adders and the breakpoint are carefully chosen such that the small adders finish computation just as their **Cin** signals become available

Short adders handle the low-order bits, and the adder length is increased further along the carry chain, since later computations have more time until their **Cin** signal is available



FPGA Carry Chain Cell



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

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

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

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

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

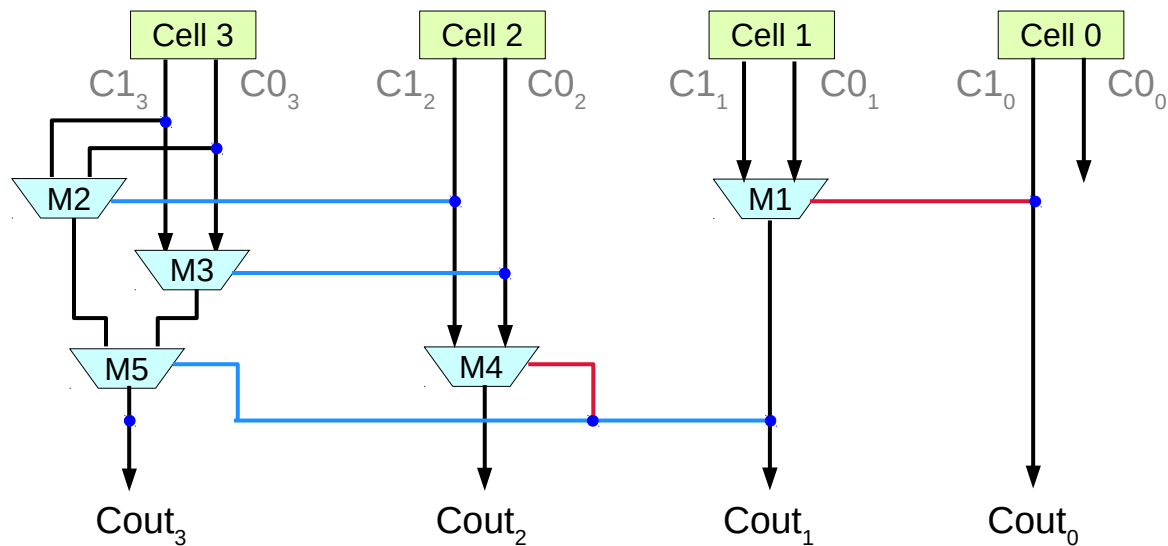
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Design C - Carry Select (1)

A **Carry Select** carry chain structure for use in FPGAs
the carry computation for the first two cells is performed with the simple **ripple-carry** structure implemented by **mux1**

For **cell2** and **cell3** we use two ripple carry adders, with one adder (implemented by **mux2**) assuming the **Cin** is **true**, and the other (**mux3**) assuming the **Cin** is **false**

Then **mux4** and **mux5** pick between these two adders' outputs based on the actual **Cin** coming from **mux1**.

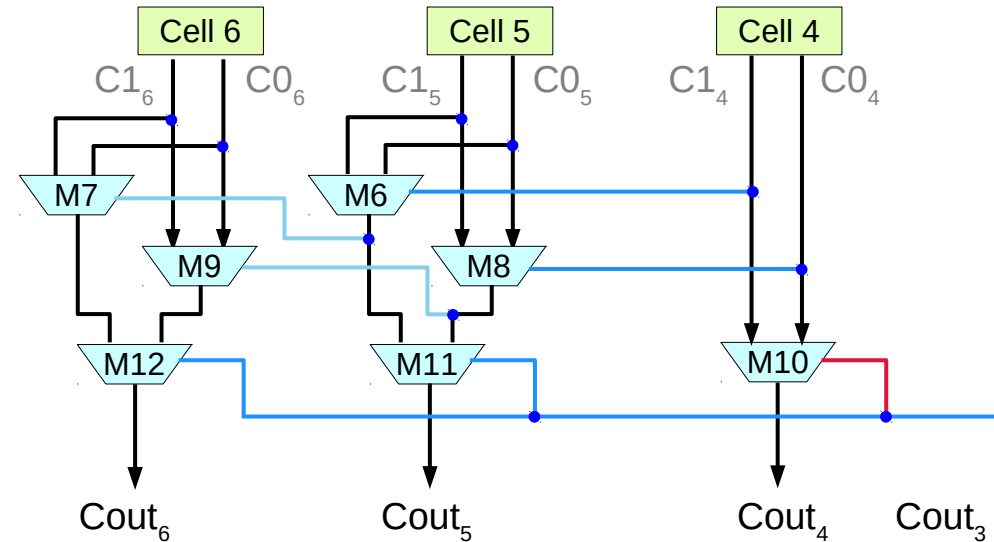


Design C - Carry Select (2)

Similarly, **cell4**, **cell5**, **cell6** have two ripple carry adders (**mux6** & **mux7** for a **Cin** of 1, **mux8** & **mux9** for a **Cin** of 0), with output muxes (**mux10**, **mux11**, **mux12**) deciding between the two based upon the actual **Cin** (from **mux5**).

Subsequent stages will continue to grow in length by one, with **cells7**, **cell8**, **cell9**, **cell10** in one block, **cell11**, **cell12**, **cell13**, **cell14**, **cell15** in another, and so on.

timing values showing the delay of the Carry Select carry chain relative to other carry chain will be presented later



Design C - Carry Select (3)

A Carry Select carry chain structure for use in FPGAs
The carry computation for the first two cells is performed with the simple ripple-carry structure implemented by mux1

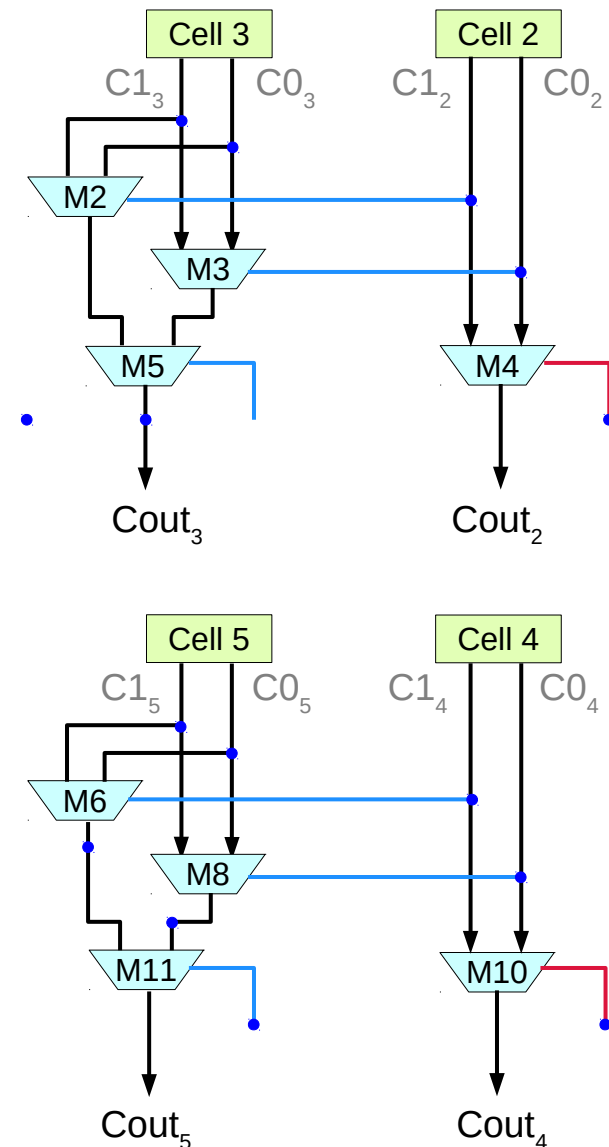
For cells 2 and 3 we use two ripple carry adders, with one adder (implemented by mux2) assuming the C_{in} is true, and the other (mux3) assuming the C_{in} is false

Then muxes 4 and 5 pick between these two adders' outputs based on the actual C_{in} coming from mux1.

Similarly, cells 4-6 have two ripple carry adders (mux6 & mux7 for a C_{in} of 1, mux8 & mux9 for a C_{in} of 0), with output muxes (muxes 10-12) deciding between the two based upon the actual C_{in} (from mux5).

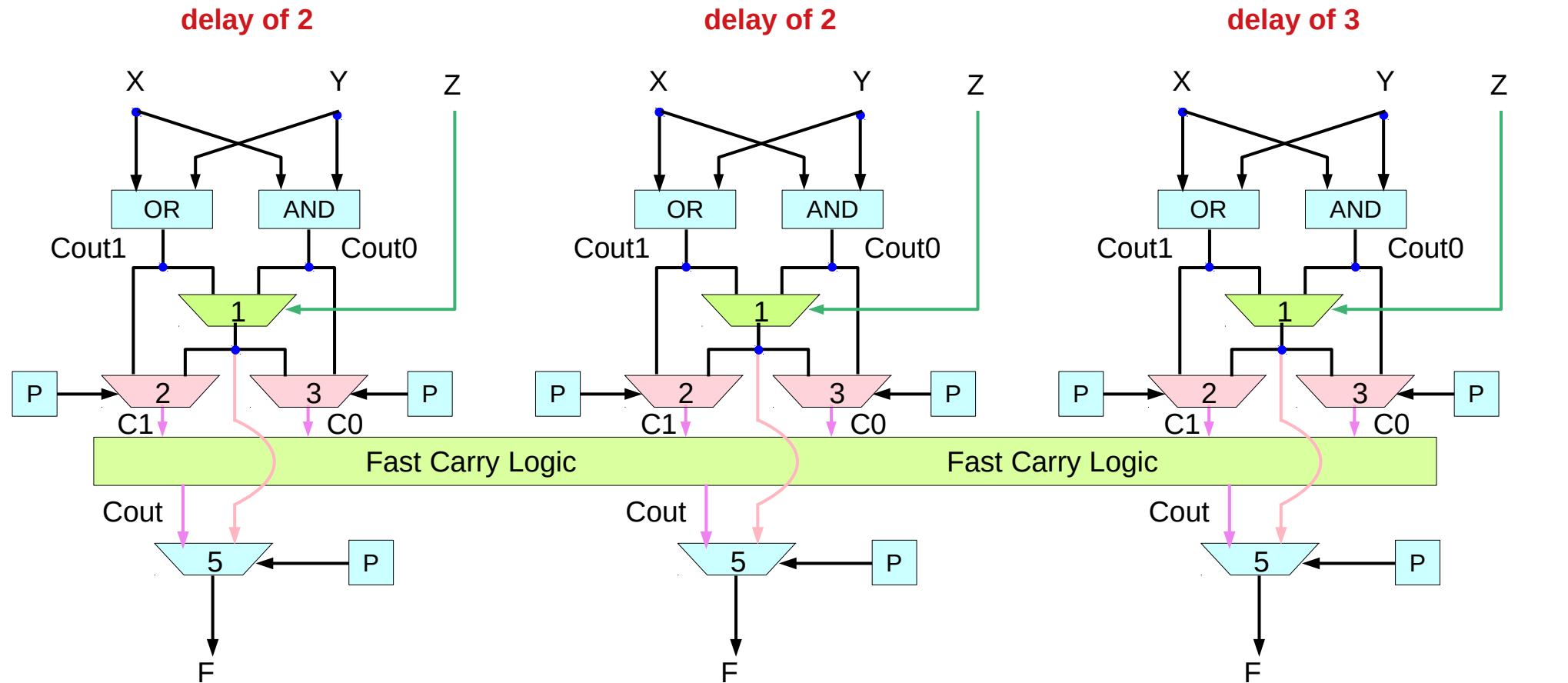
Subsequent stages will continue to grow in length by one, with cells 7-10 in one block, cells 11-15 in another, and so on.

timing values showing the delay of the Carry Select carry chain relative to other carry chain will be presented later



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

Design C

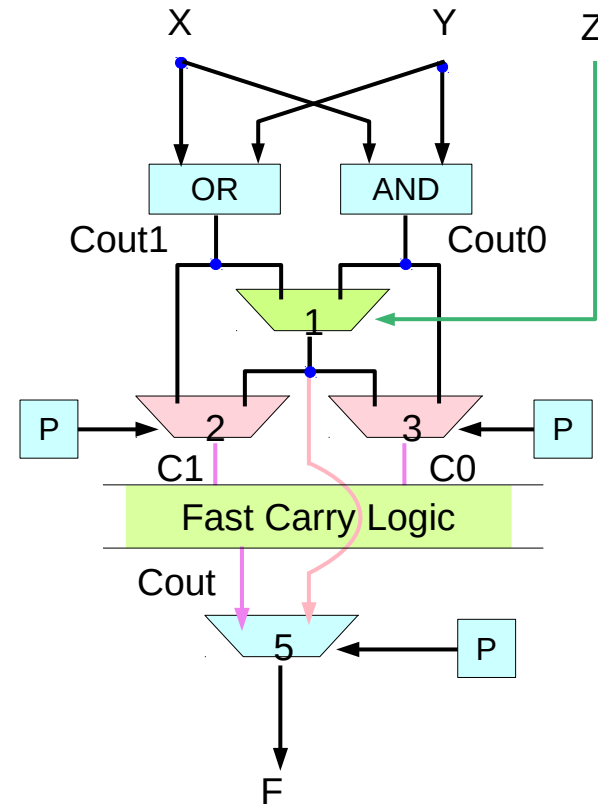
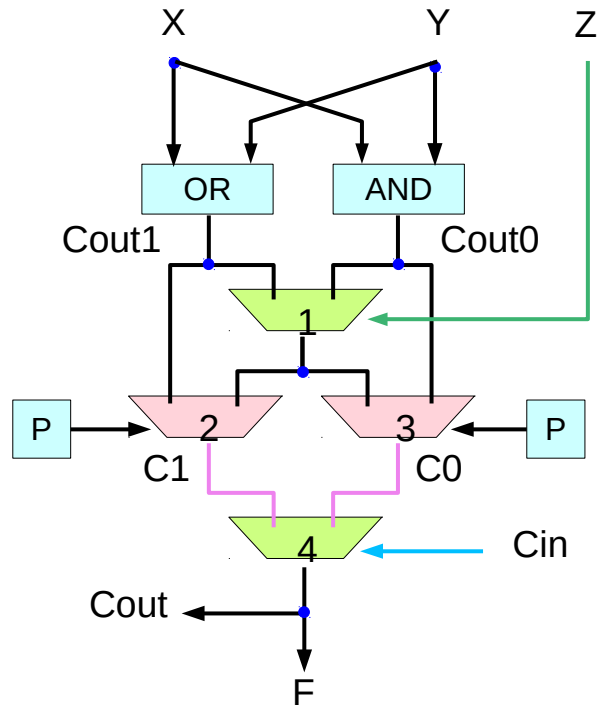


(1 for mux1, 1 for mux2, 1 in mux4)

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

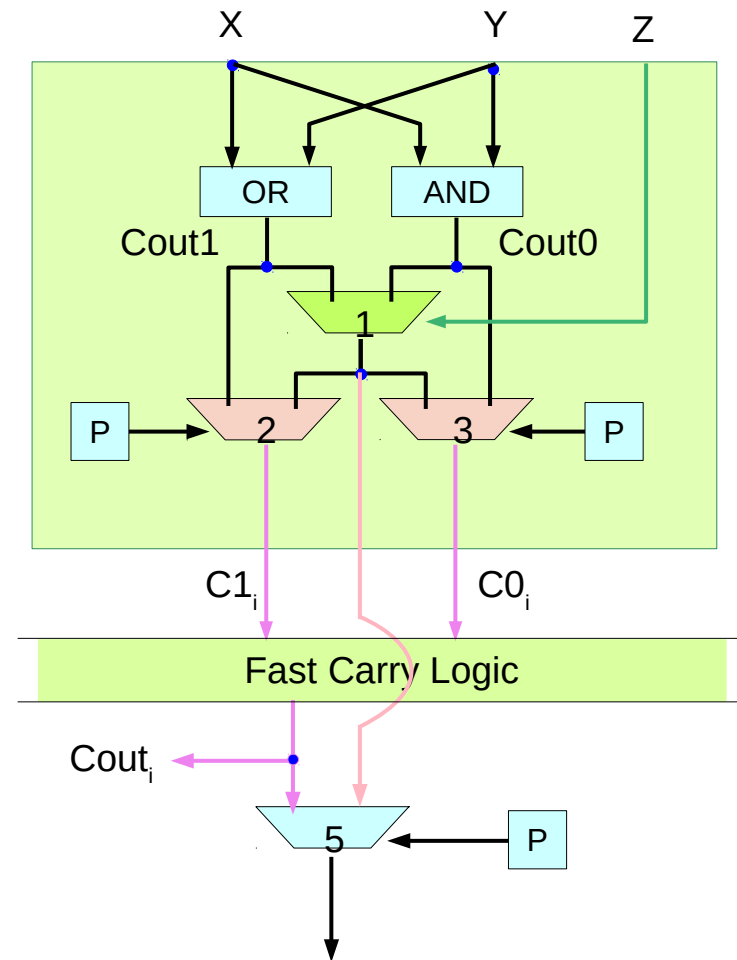
FPGA Carry Chain Cell



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

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

FPGA Carry Chain Cell



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

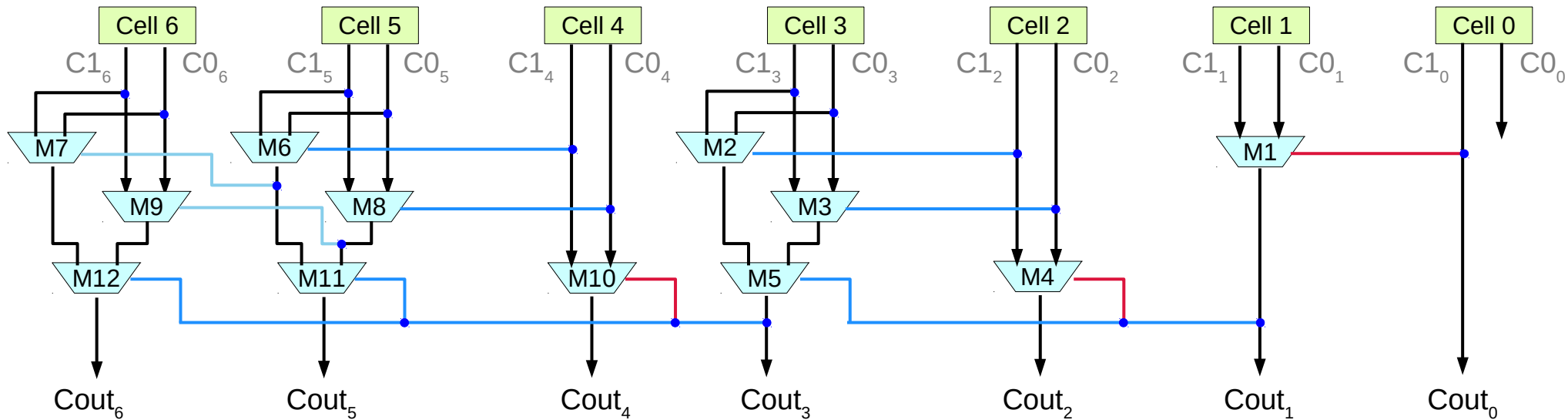
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

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



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

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

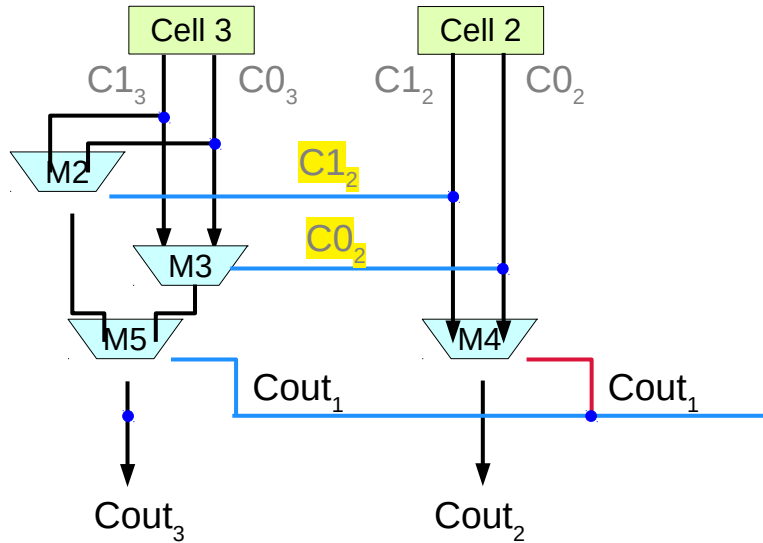
$$Cout_{i+1} = (Cout_i \cdot C1_{i+1}) + (\overline{Cout_i} \cdot C0_{i+1})$$

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

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

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

FPGA Carry Chain Cell



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

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

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

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

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

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

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

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

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

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

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

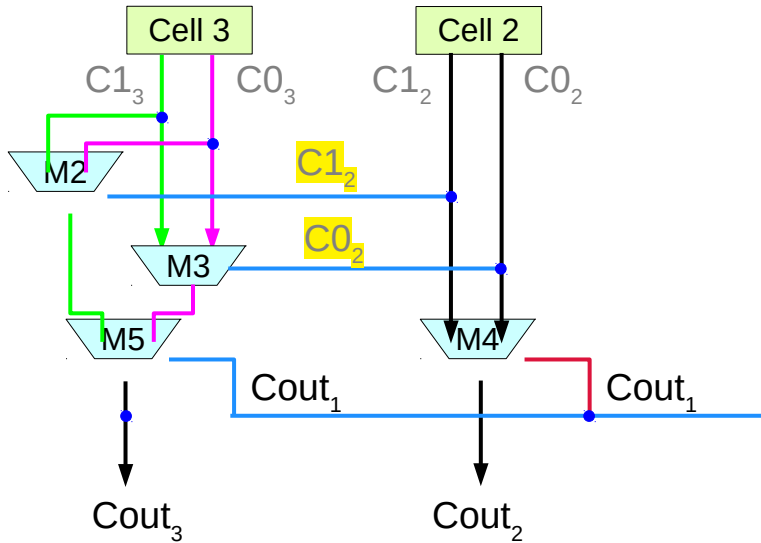
$$= (\overline{C1_2} \overline{Cout_1} + \overline{C0_2} \overline{Cout_1}) \cdot C0_3$$

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

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

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

FPGA Carry Chain Cell

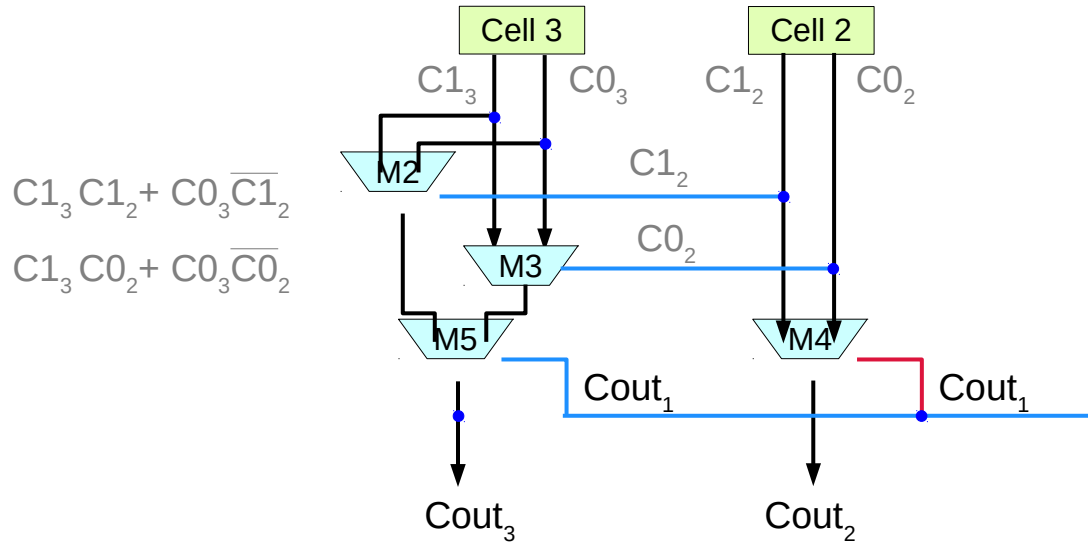


$$\begin{aligned}
 &= (\overline{Cout_1}Cout_1 + \overline{C1_2}Cout_1 + \overline{Cout_1}C0_2 + \overline{C1_2}C0_2) \cdot C0_3 \\
 &= (\overline{C1_2}Cout_1 + \overline{C0_2}Cout_1) \cdot C0_3 \\
 &= (C0_3\overline{C1_2}Cout_1 + C0_3\overline{C0_2}Cout_1)
 \end{aligned}$$

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

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

FPGA Carry Chain Cell



$$C_1 C_2 + C_0 \overline{C_1}$$

$$C_1 C_0 + C_0 \overline{C_0}$$

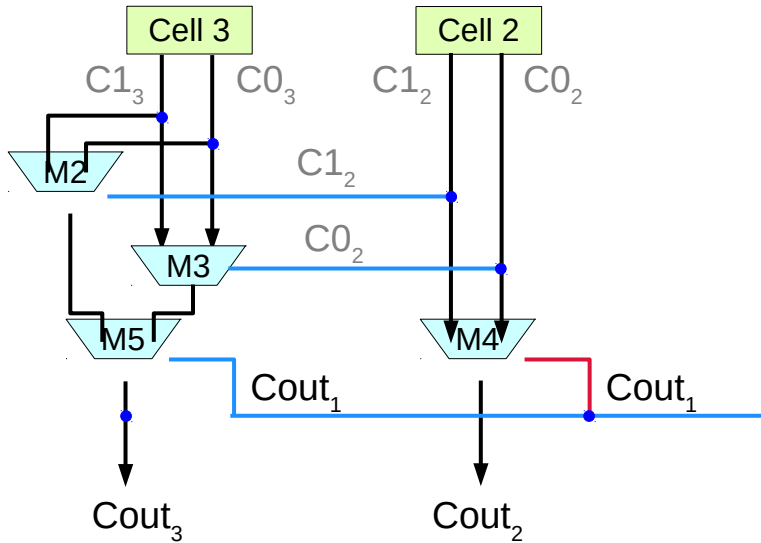
$$(C_1 C_2 + C_0 \overline{C_1}) Cout_1 + (C_1 C_0 + C_0 \overline{C_0}) \overline{Cout_1}$$

$$= C_1 \cdot (C_2 Cout_1 + C_0 \overline{Cout_1})$$

$$+ C_0 \cdot (\overline{C_1} Cout_1 + \overline{C_0} \overline{Cout_1})$$

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., “Synthesis of Arithmetic Circuits”, 2006