FPGA Carry Select Adder (1B)

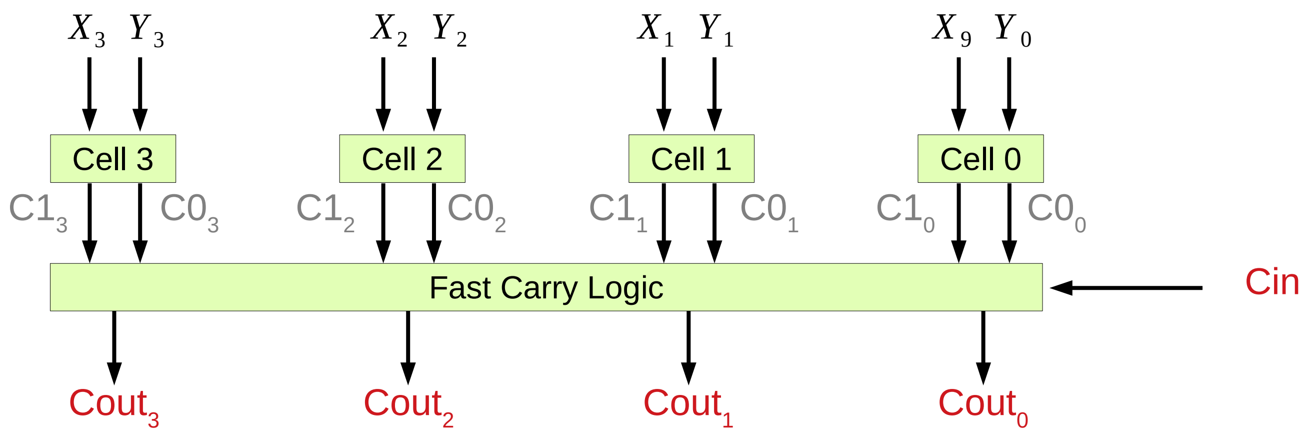
- Carry Select
-

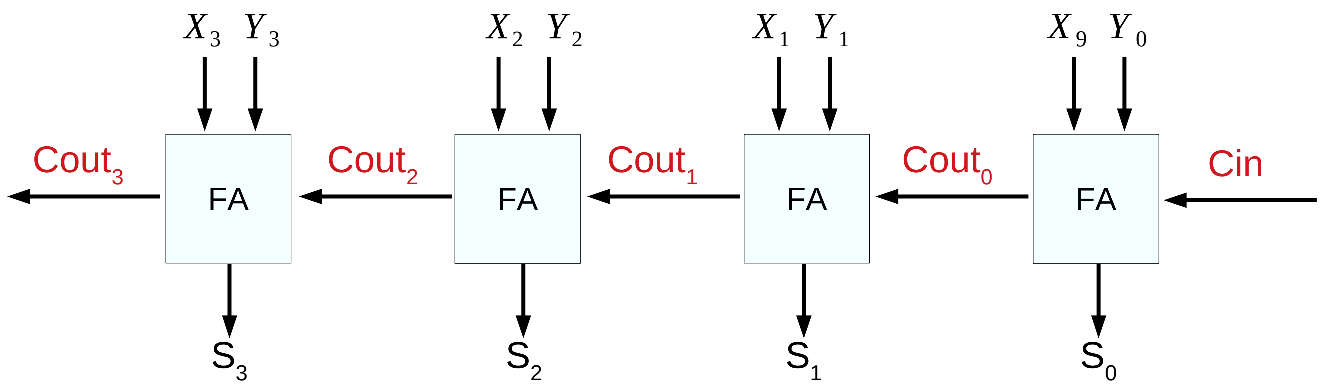
Fast Carry Logic

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

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

Fast Carry Logic v.s. Ripple Carry Logic





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

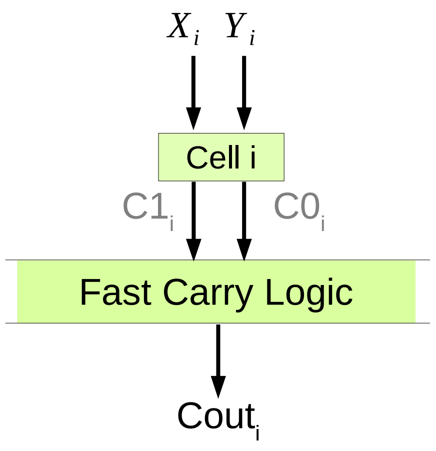
Carry Select Equation

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 .. 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**.



$$C1 = X + Y$$
$$C0 = X \cdot Y$$

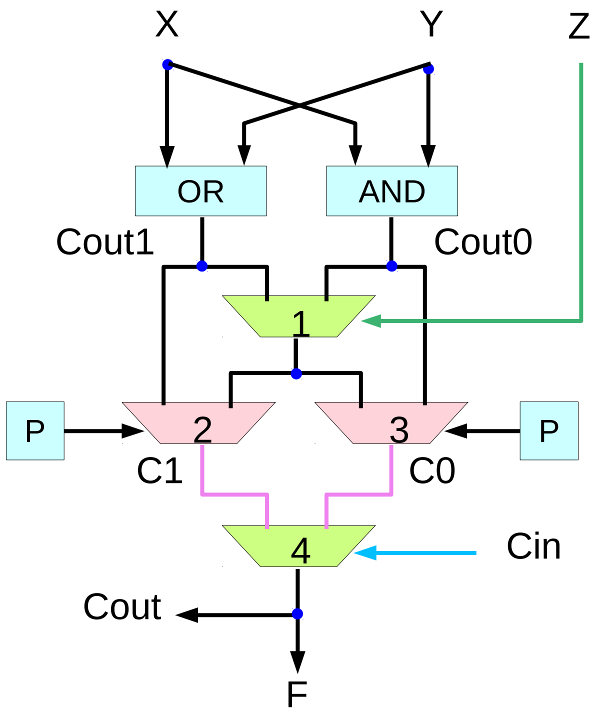
$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

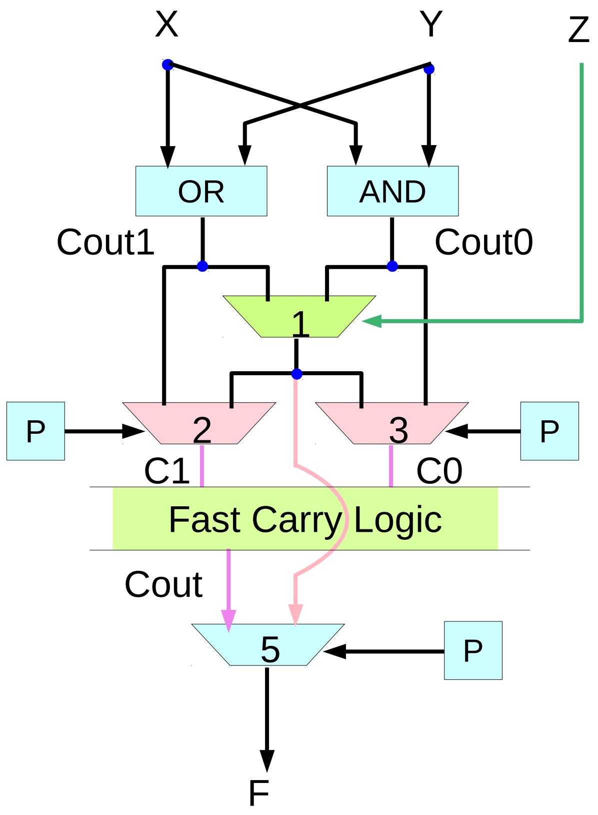
True Cin **False Cin**

X	Y	C1	C0	
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	$X\,Y$

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

Carry Chain Adders

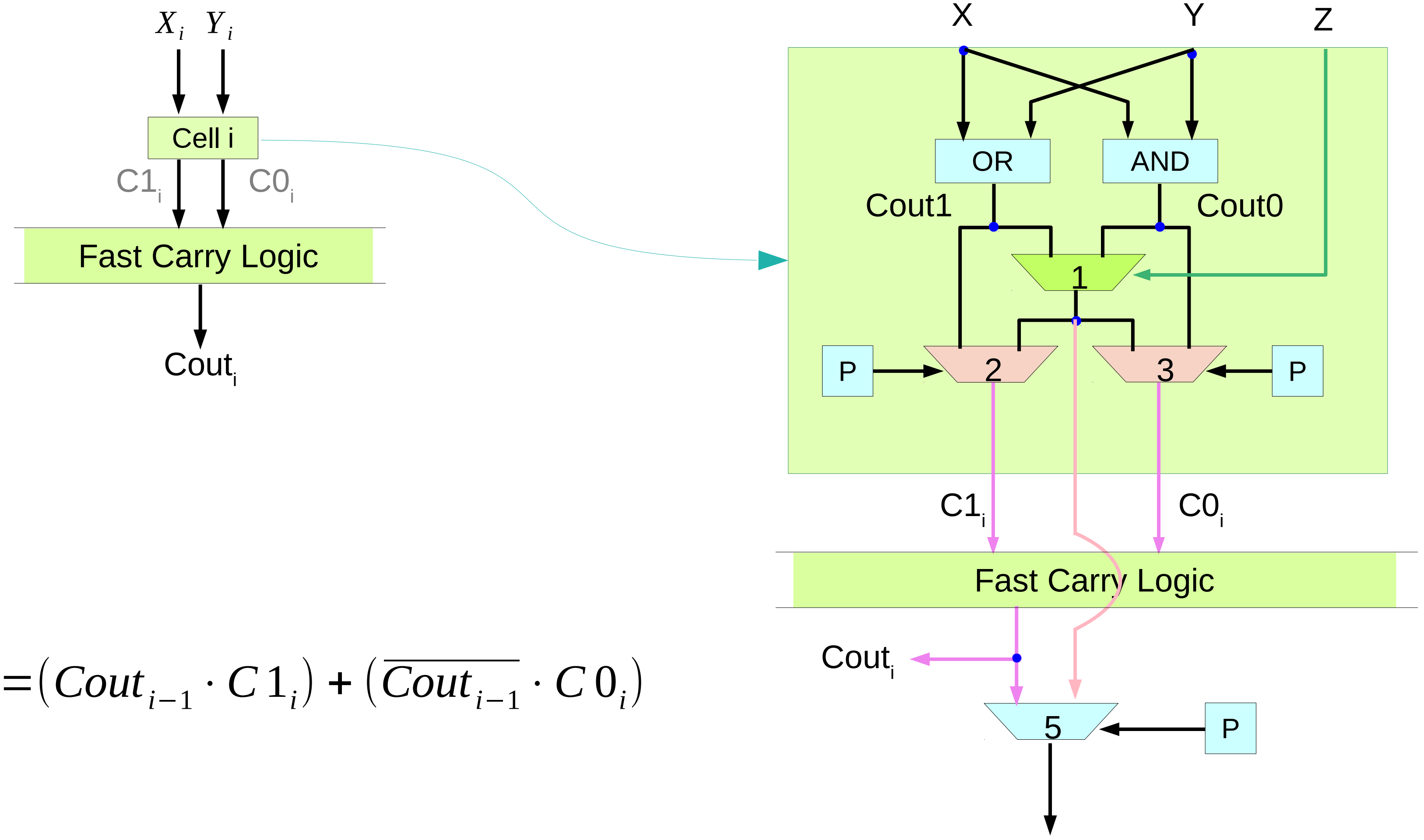




$$Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i) \qquad Cout_i = (Cout_{i-1} \cdot C1_i) + (\overline{Cout_{i-1}} \cdot C0_i)$$

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

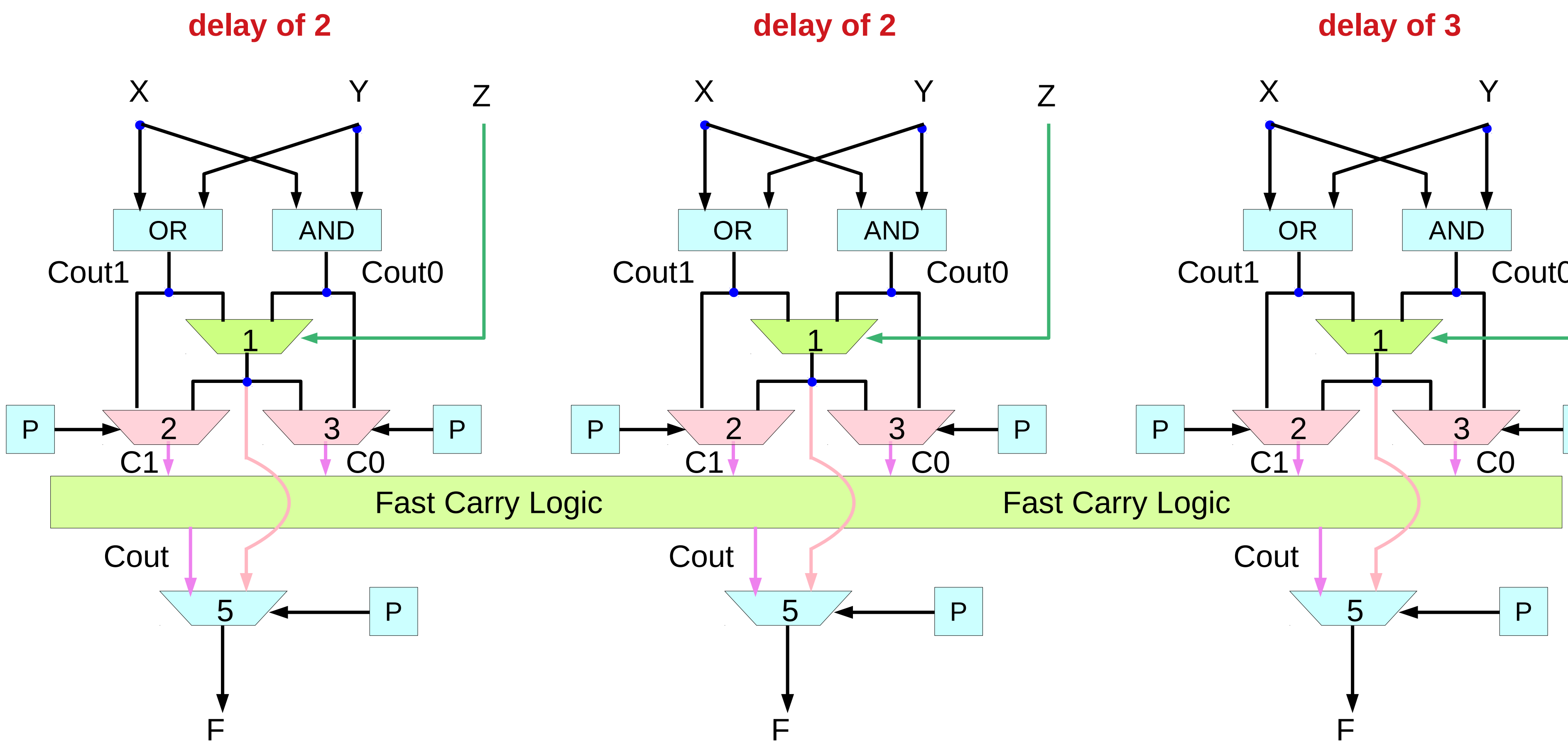
FPGA Carry Chain Cell



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

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

Carry Chain Adders with Fast Carry Logic

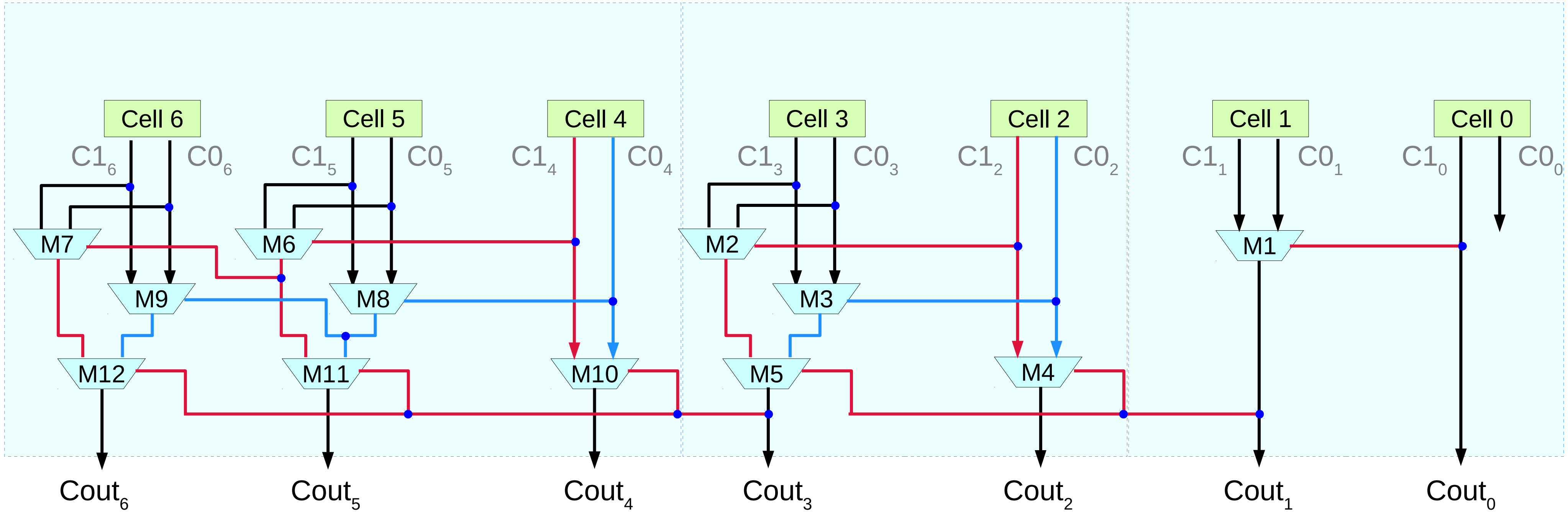


(1 for mux1, 1 for mux2, 1 in mux4)

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

Fast Carry Logic using Carry Select Structure



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

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

Cout using C1, C0, Cin

X	Y	C1	C0	
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	$X\,Y$

$C1 = X + Y$

$C0 = X \cdot Y$

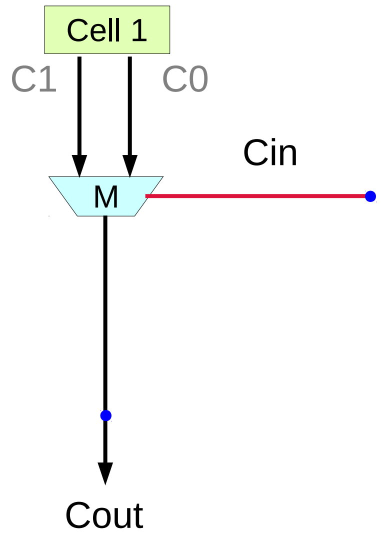
C1	C0	Name	
0	0	0	Kill
0	1	$\overline{Cin}$	Inverse Propagate
1	0	Cin	Propagate
1	1	1	Generate

$$Cout = (Cin \cdot C1) + (\overline{Cin} \cdot C0)$$

$$(Cin \cdot C1) = Cin \cdot (\overline{X}\,Y + X\,\overline{Y} + X\,Y) \rightarrow \textit{propagate Cin}$$

$$(\overline{Cin} \cdot C0) = \overline{Cin} \cdot X\,Y \rightarrow \textit{generate a new carry}$$

X	Y	Cin	Cout
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



$$= (Cin \cdot C1) + (\overline{Cin} \cdot C0)$$

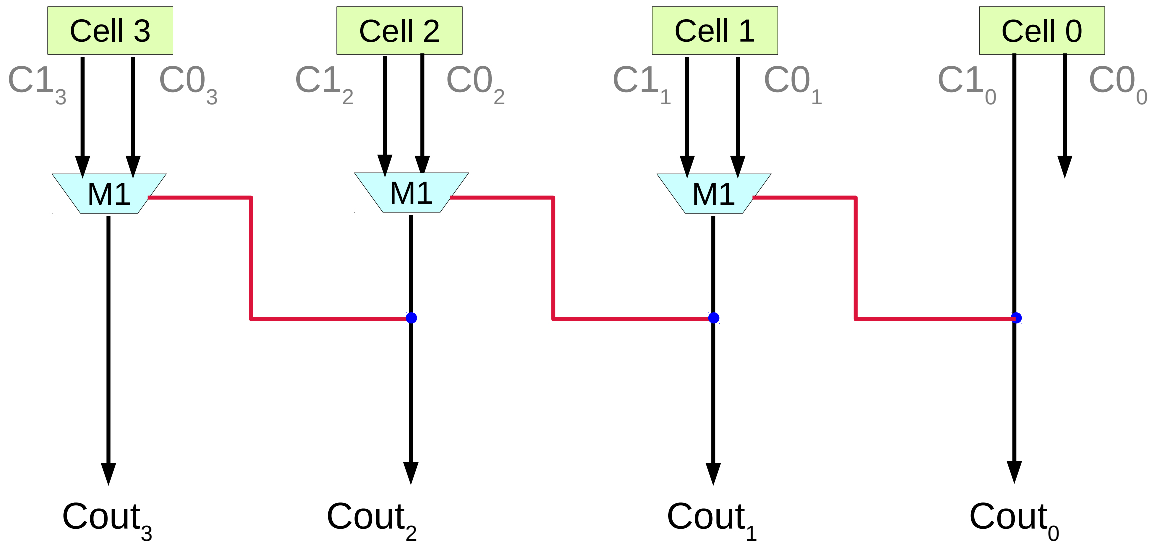
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Ripple Carry Structure

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

$$Cout = (Cin \cdot C1) + (\overline{Cin} \cdot C0)$$

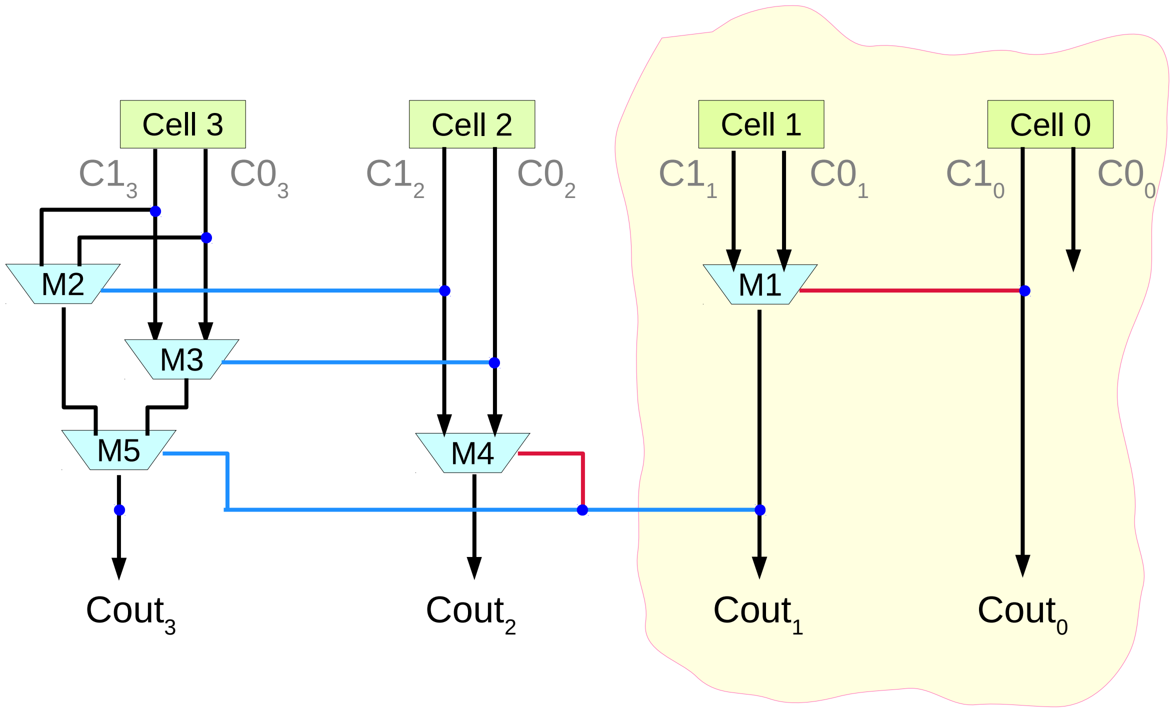


High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

The 1st Carry Select Structure

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



$(C1_1)Cout_0 + (\overline{C0_1})\overline{Cout_0}$

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

The 2nd Carry Select Structure (1)

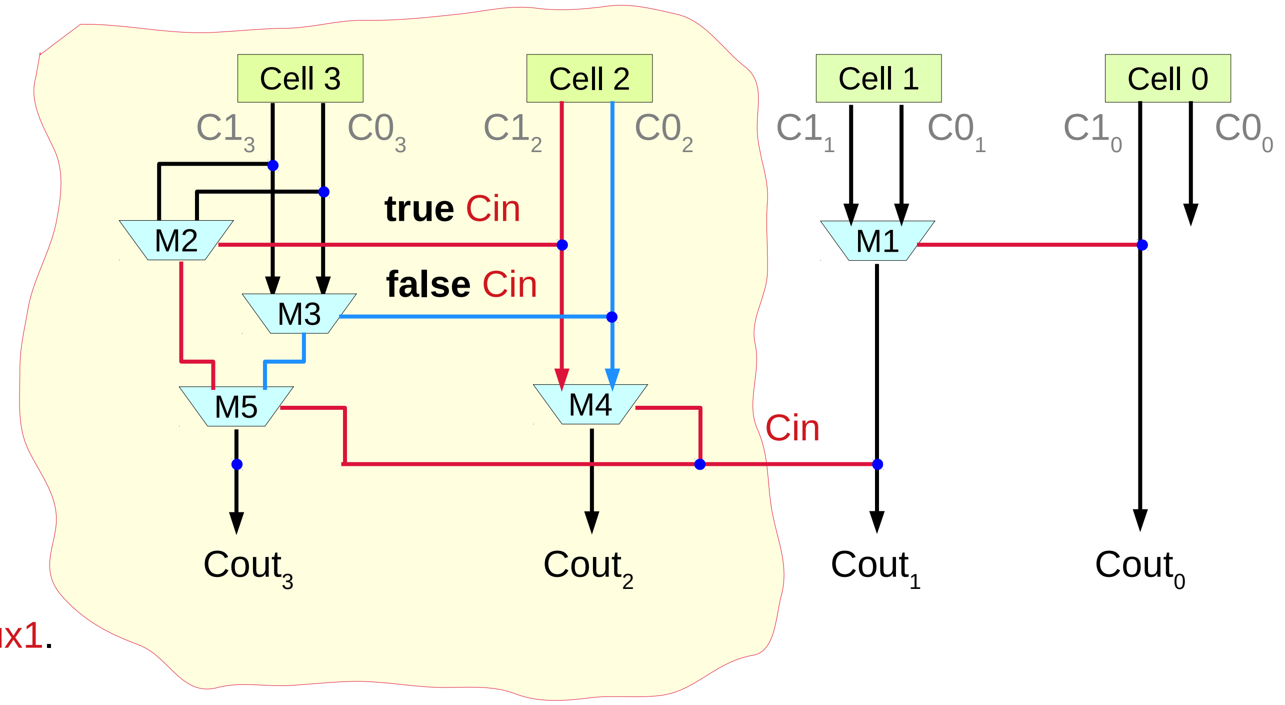
A **Carry Select carry chain** structure
for use in FPGAs

For cell2 and cell3 we use
two **ripple carry adders**,

with one adder (mux2)
assuming the Cin is **true**,

and the other (mux3)
assuming the Cin is **false**

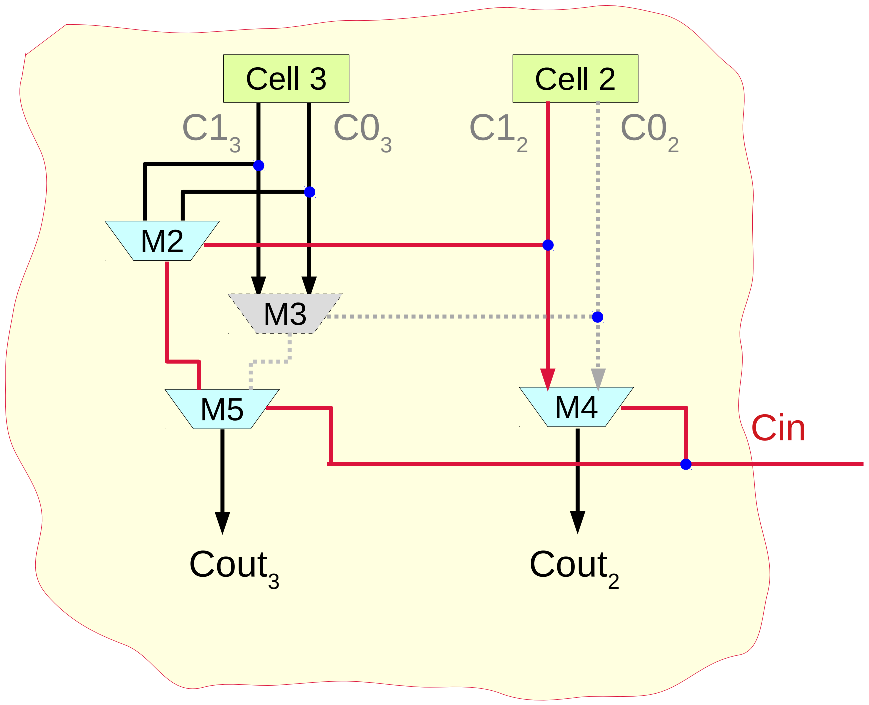
Then mux4 and mux5 pick
between these two adders' outputs
based on the actual Cin coming from mux1.



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

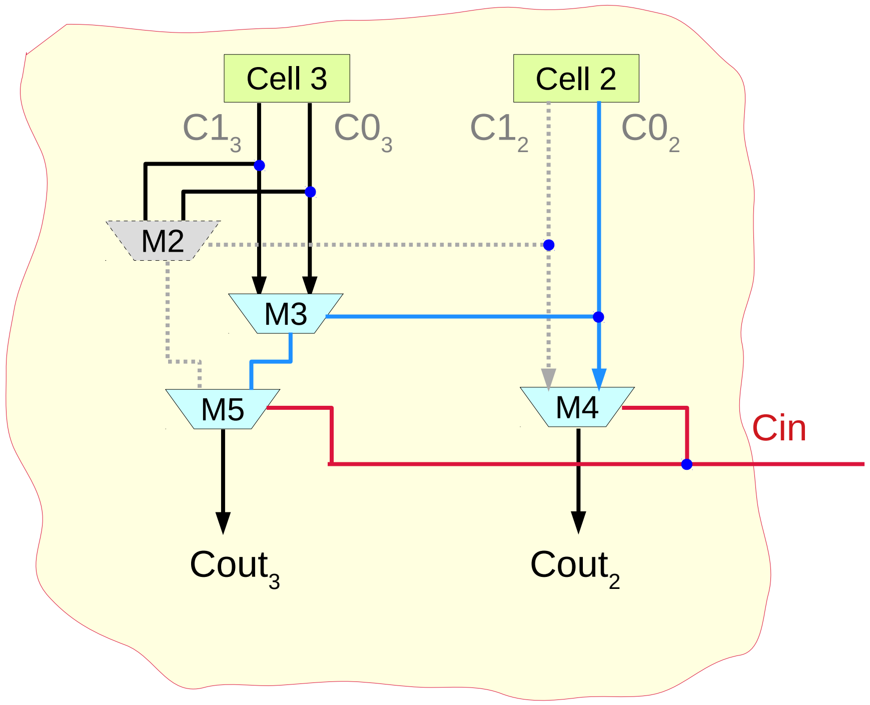
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

The 2nd Carry Select Structure (2)



true Cin

$(C1\,C1_2 + C0_3\overline{C1_2})Cin$

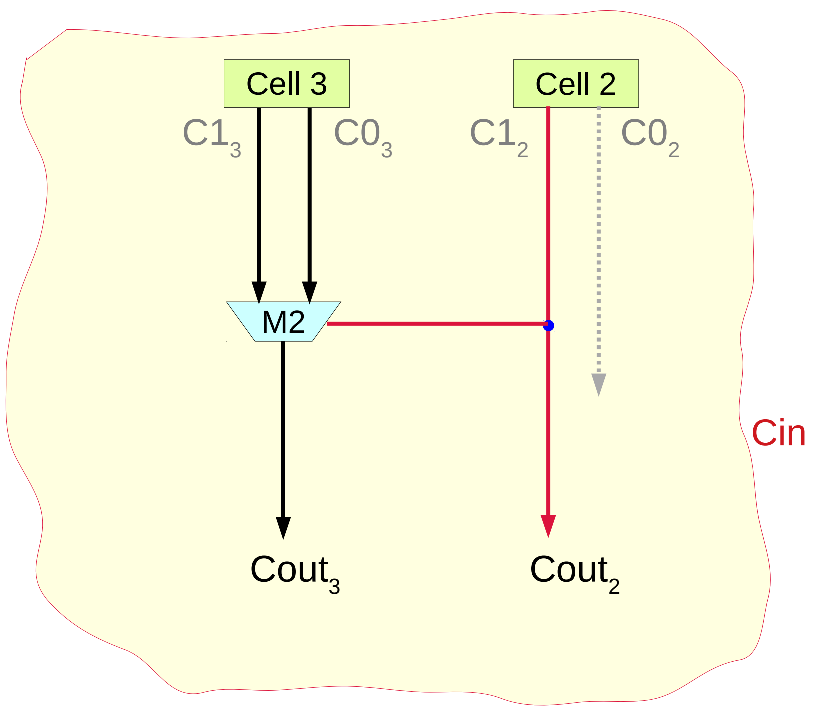


false Cin

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

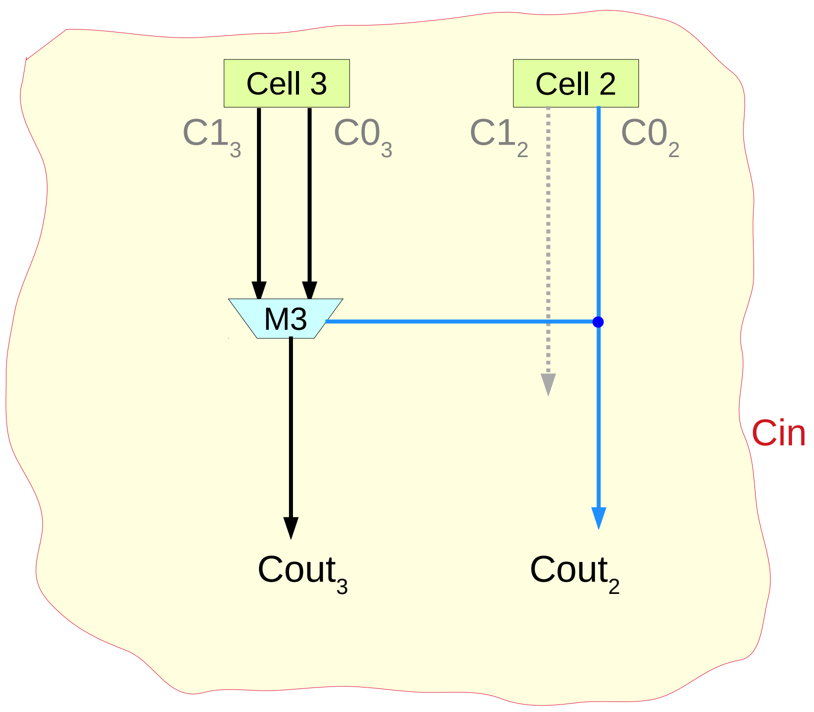
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

The 2nd Carry Select Structure (3)



true Cin

$(C1\,C1_2 + C0_3\overline{C1_2})Cin$



false Cin

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

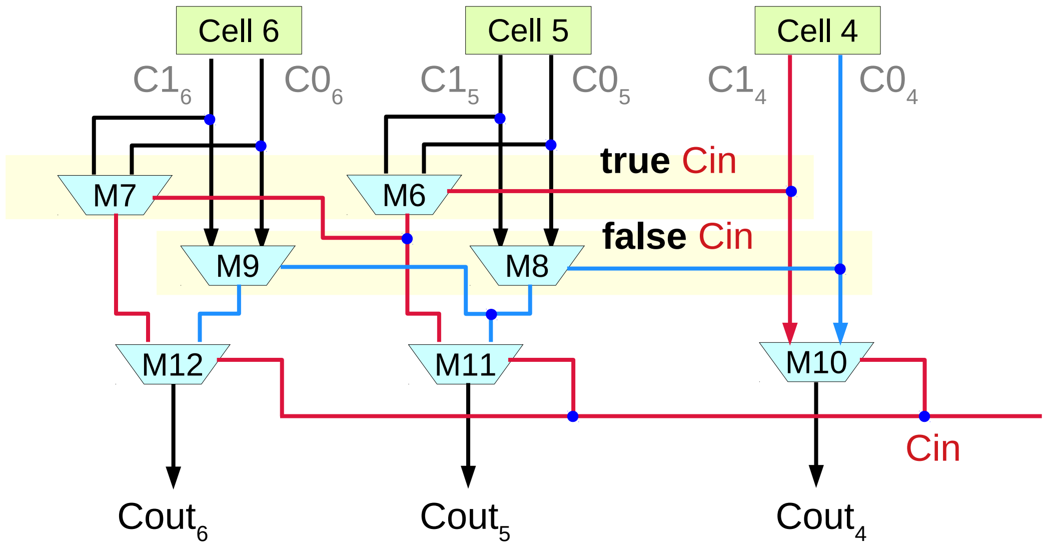
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

The 3rd Carry Select Structure (1)

Similarly, cell4, cell5, cell6 have

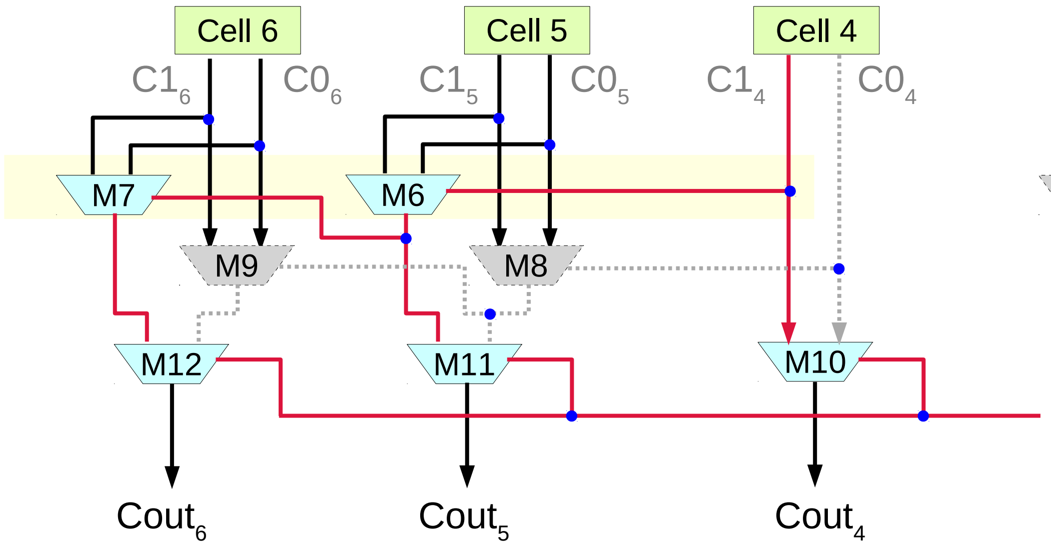
two ripple carry adders
mux6 & mux7 for a Cin of 1
mux8 & mux9 for a Cin of 0

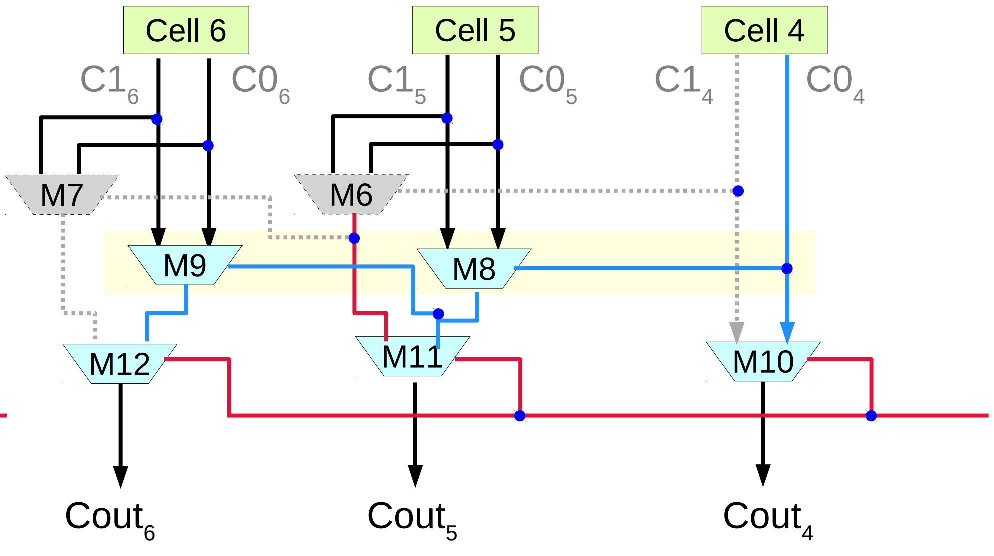
with output muxes (mux10, mux11, mux12)
deciding between the two
based upon the actual Cin (from mux5).



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

The 3rd Carry Select Structure (2)





true Cin

false Cin

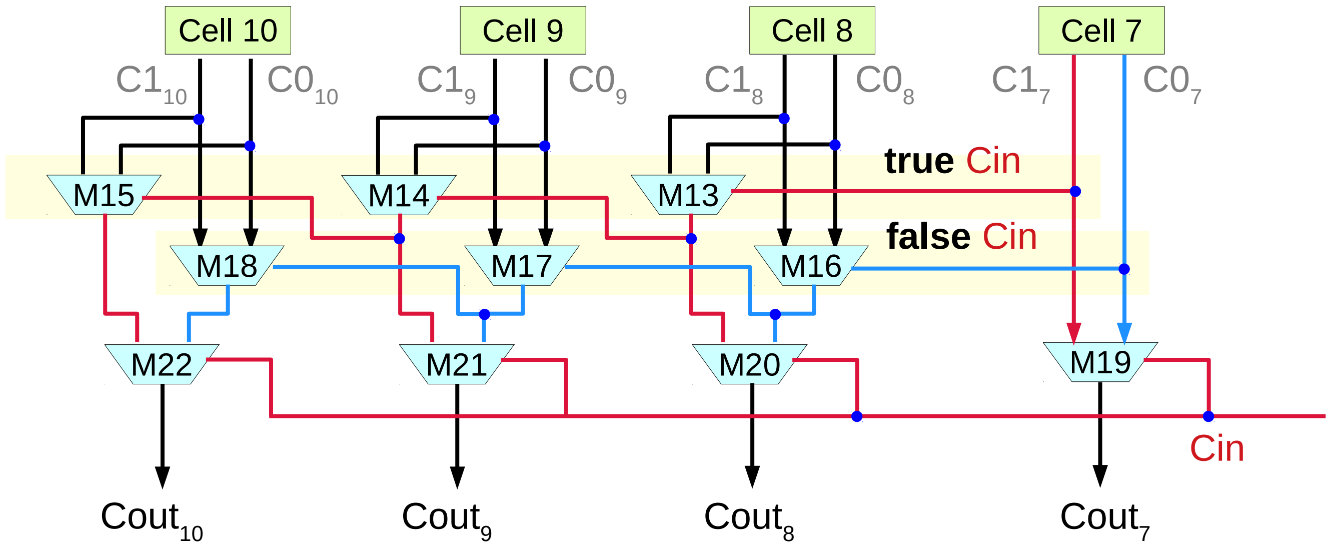
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

The 4th Carry Select Structure

Subsequent stages will continue to grow in length by one,

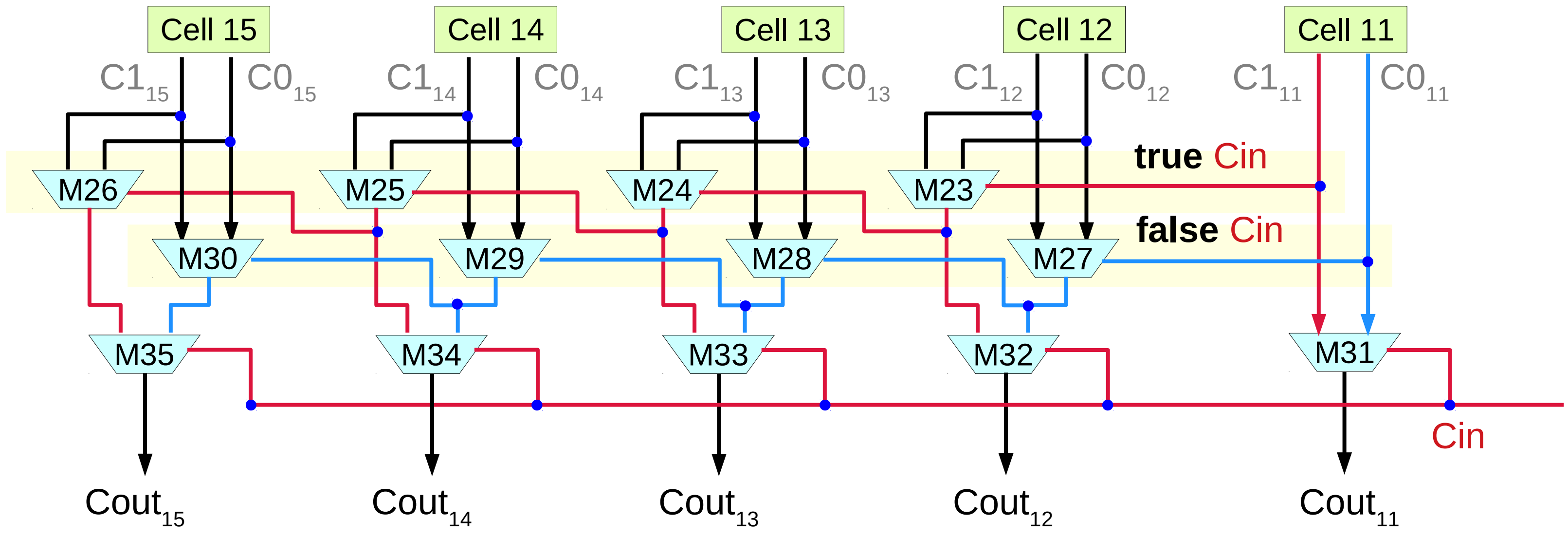
with cells7, cell8, cell9, cell10 in one block,

cell11, cell12, cell13, cell14, cell15 in another, and so on.



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

The 5th Carry Select Structure



Subsequent stages will continue
to grow in length by one,

cell11, cell12, cell13, cell14, cell15
in another,
and so on.

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

Carry Chain Resource

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

Must consider

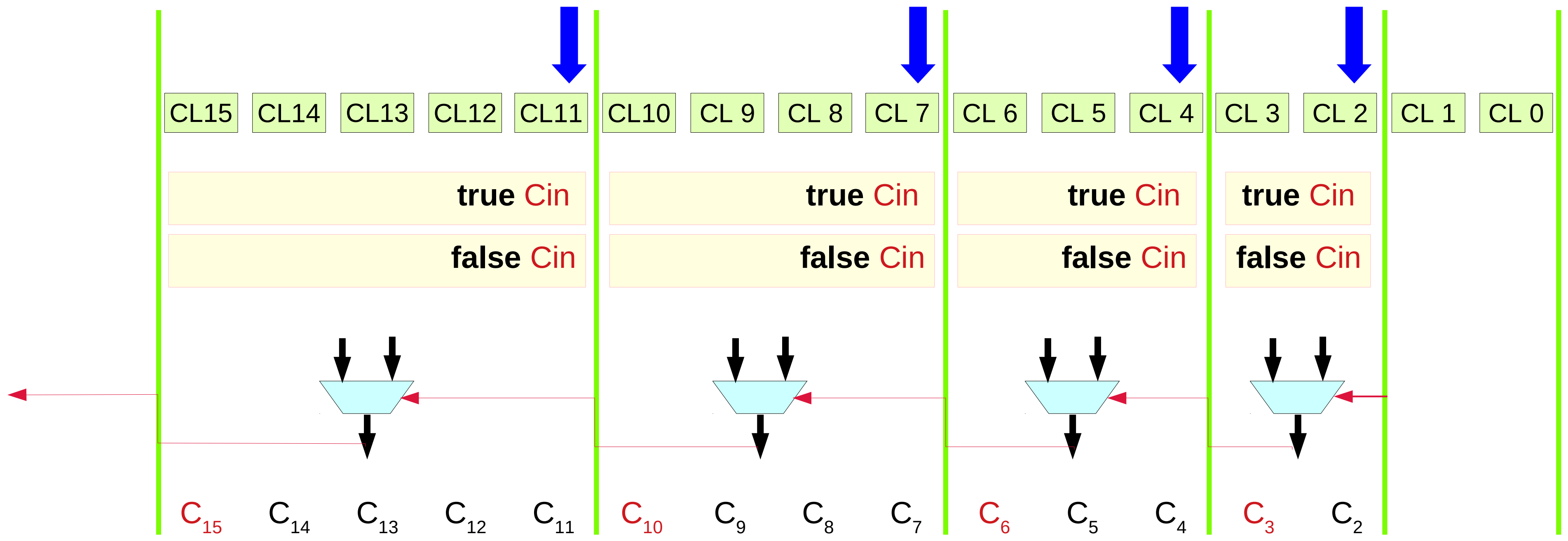
- the **carry delay** from the first
  to the last position in a carry chain,
- the delay for a **carry computation**
  beginning at any point within this column.

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

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

Broken carry chains

can start simultaneously for two possible case of Cin



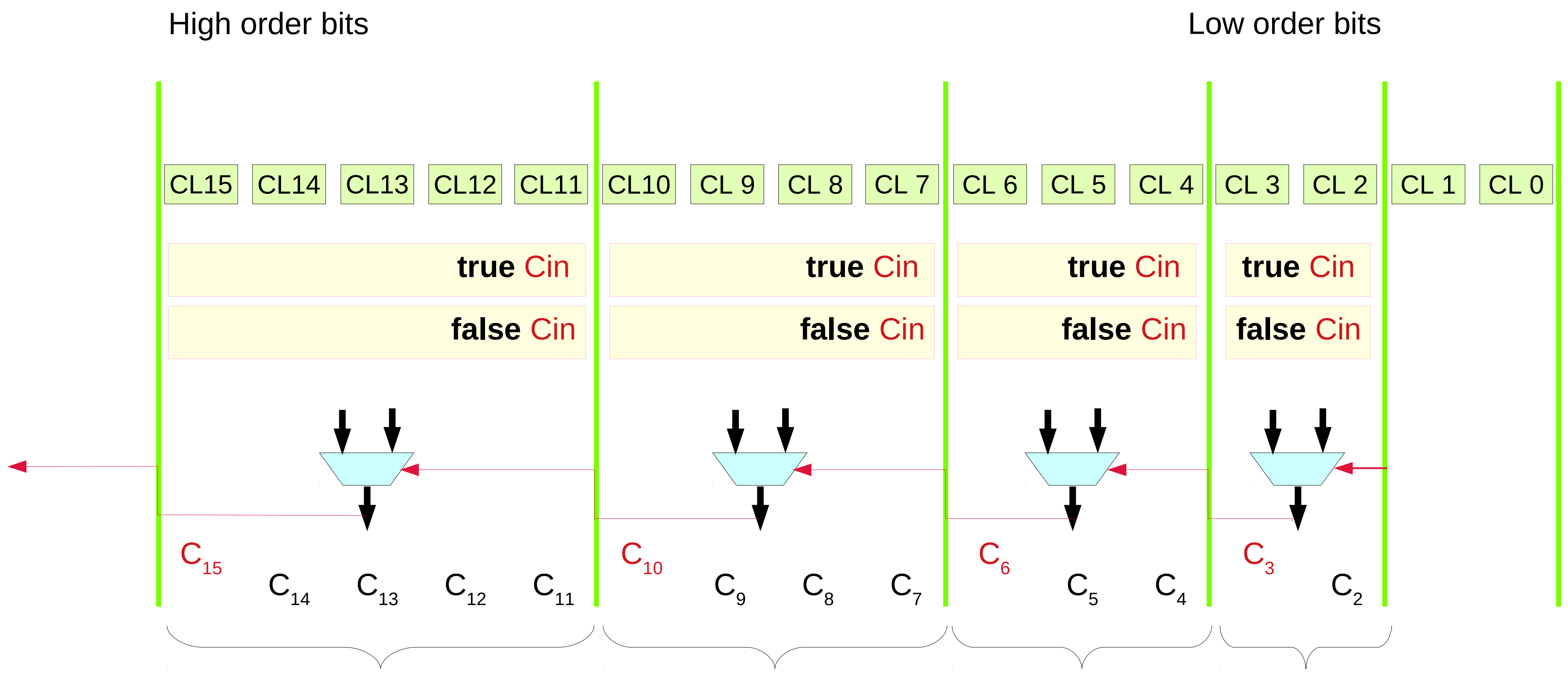
In a carry select adder
the **carry chain** is broken at a specific column,
and two separate adders are deployed

one assuming **true** Cin signal
the other assuming **false** Cin signal

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**

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

Broken carry chains



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

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

Two separate ripple carry adders

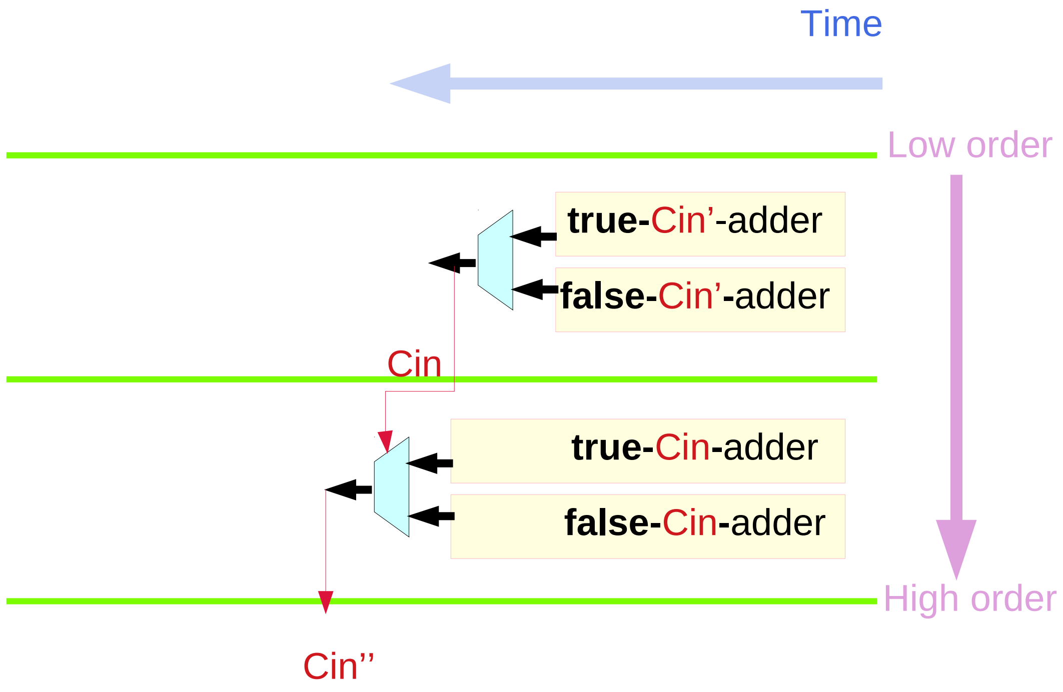
Cin signal is used to determine
which adder's outputs should be used

if the Cin signal is **true**,
the output (carries) are selected from
the **true-Cin-adder**

if the Cin signal is **false**,
the output (carries) are selected from
the **false-Cin-adder**

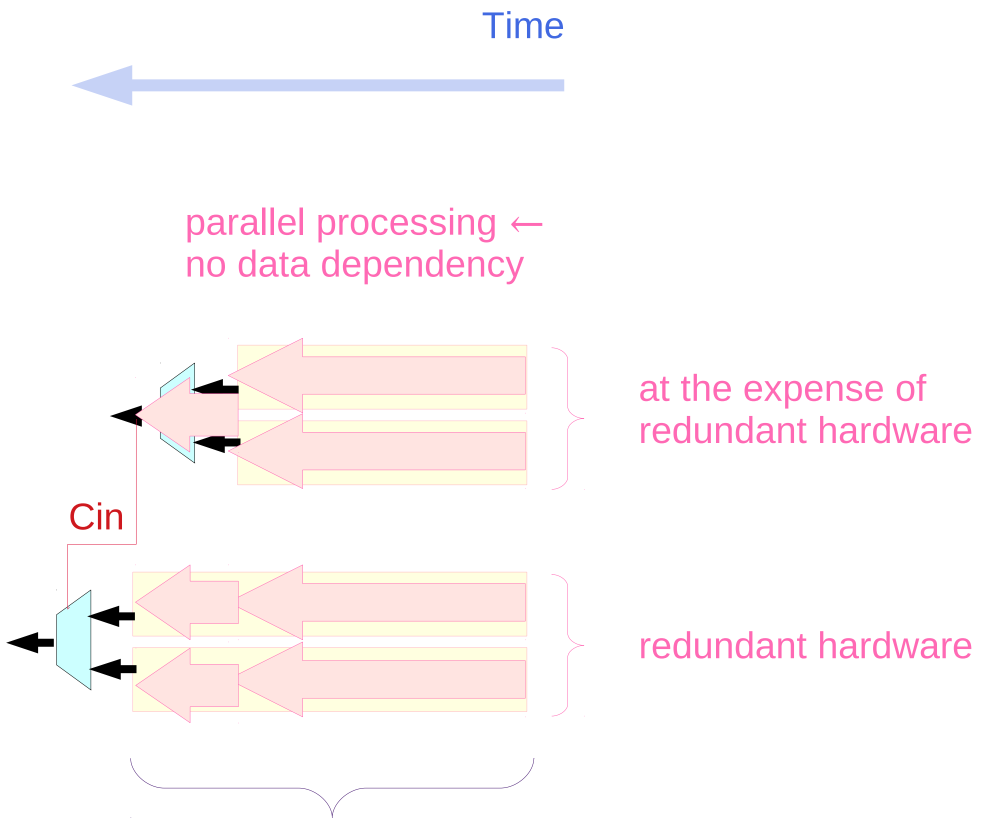
redundant hardware removes
Cin data dependency

first start redundant computation
later select the correct one

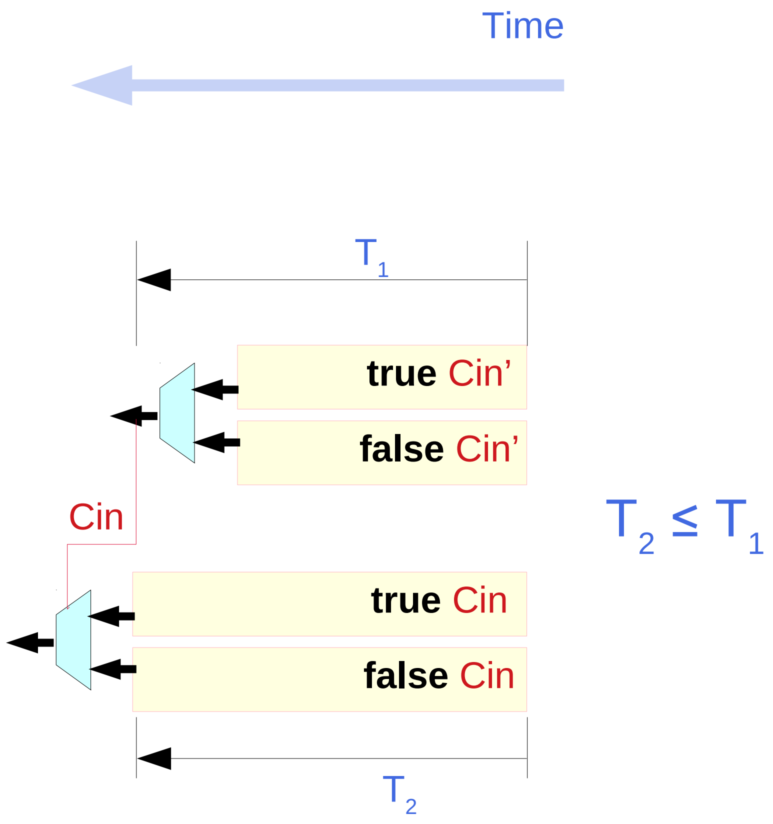


High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Timing in broken carry chains


Time
parallel processing ← no data dependency
at the expense of redundant hardware
Cin
redundant hardware

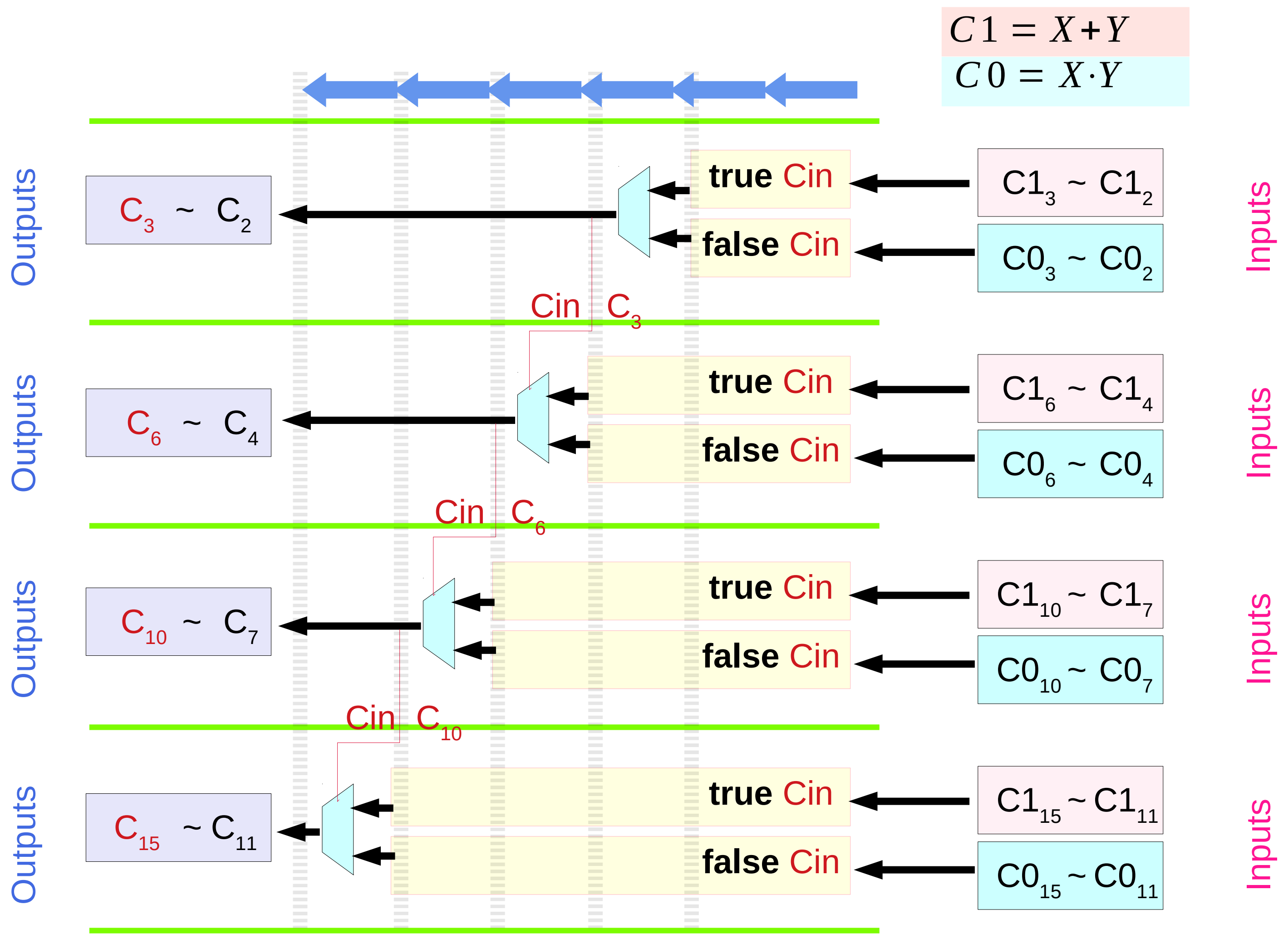
These computations can take place before the completion of the previous columns, since they do not depend on the actual value of the Cin signal



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

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

Carry Select Fast Carry Logic



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

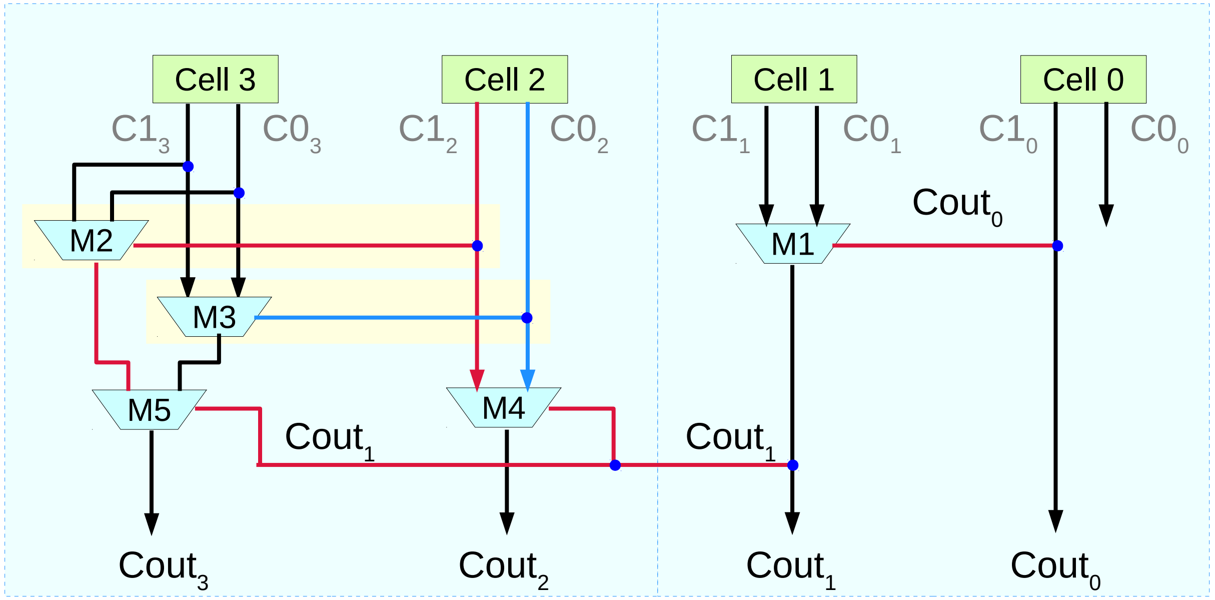
Analysis of the carry equation

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

of the 2nd carry select structure

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

Two ripple carry structure



$C1 = \overline{X}\,Y + X\,\overline{Y} + X\,Y$ $\quad C0 = X\,Y$

$\overline{C1} = \overline{X}\,\overline{Y}$ $\quad \overline{C0} = \overline{X}\,Y + X\,\overline{Y} + \overline{X}\,\overline{Y}$

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

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

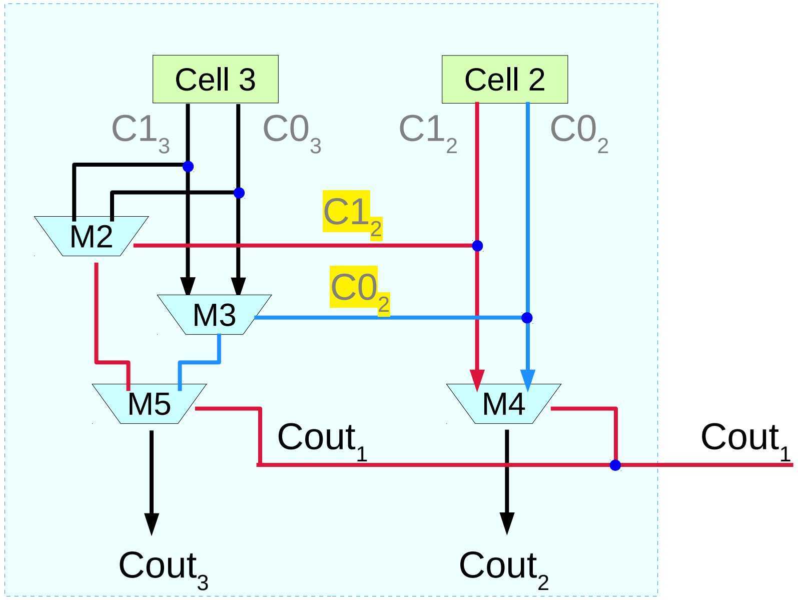
$= (Cout_0 \cdot C1_1) + (\overline{Cout_0} \cdot C0_1)$

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

$$= (Cout_1 \cdot (C1_3 \cdot (\overline{X}_2 Y_2 + X_2 \overline{Y}_2 + X_2 Y_2) + C0_3 \cdot \overline{X}_2 \overline{Y}_2)) + (\overline{Cout_1} \cdot (C1_3 \cdot X_2 Y_2 + C0_3 \cdot (\overline{X}_2 Y_2 + X_2 \overline{Y}_2 + \overline{X}_2 \overline{Y}_2)))$$

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

$Cout_3$ in terms of $Cout_1$



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

$C1 = \overline{X}\,Y + X\,\overline{Y} + X\,Y$ $\quad C0 = X\,Y$

$\overline{C1} = \overline{X}\,\overline{Y}$ $\quad \overline{C0} = \overline{X}\,Y + X\,\overline{Y} + \overline{X}\,\overline{Y}$

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

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

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

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

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

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

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

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

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

Two mutually exclusive cases Cout$_1$ and $\overline{\text{Cout}_1}$

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

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

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

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

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

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

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

$$((Cout_1 \cdot C1_2) + (\overline{Cout_1} \cdot C0_2)) \quad is\ false$$

$$((Cout_1 \cdot C1_2)) = F \;\wedge\; ((\overline{Cout_1} \cdot C0_2)) = F$$

Two mutually exclusive cases

when $Cout_1$ *is true*

$C1_2$ *must be false*

because $(Cout_1 \cdot C1_2)$ *must be false*

$\Rightarrow$ *therefore* $(\overline{C1_2} Cout_1)$

when $\overline{Cout_1}$ *is true*

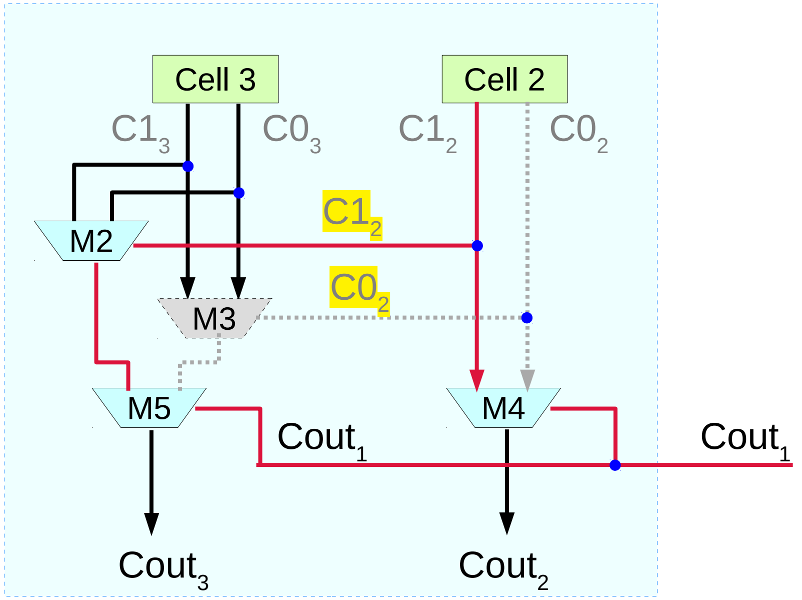
$C0_2$ *must be false*

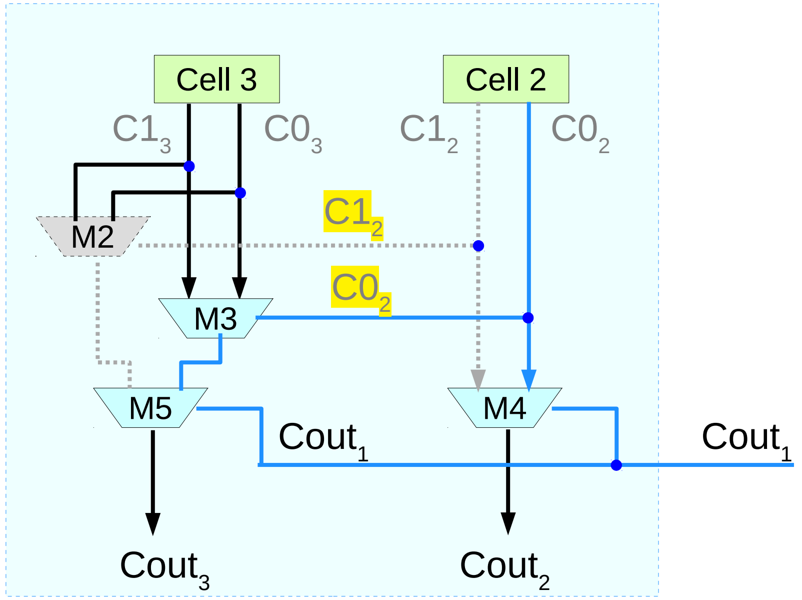
because $(\overline{Cout_1} \cdot C0_2)$ *must be false*

$\Rightarrow$ *therefore* $(\overline{C0_2}\,\overline{Cout_1})$

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

When $Cout_1$ and $\overline{Cout_1}$





when $Cout_1$ *is true*

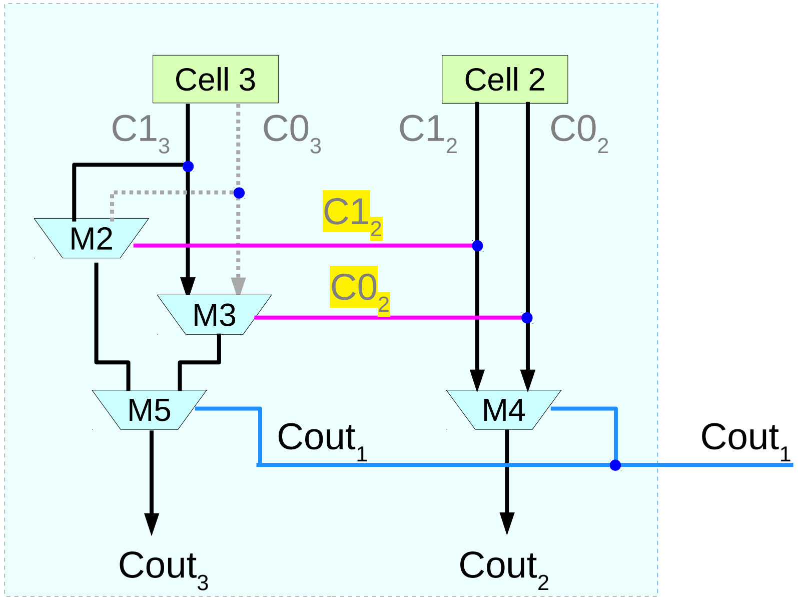
$(C1_3 C1_2 + C0_3 \overline{C1_2}) Cout_1$

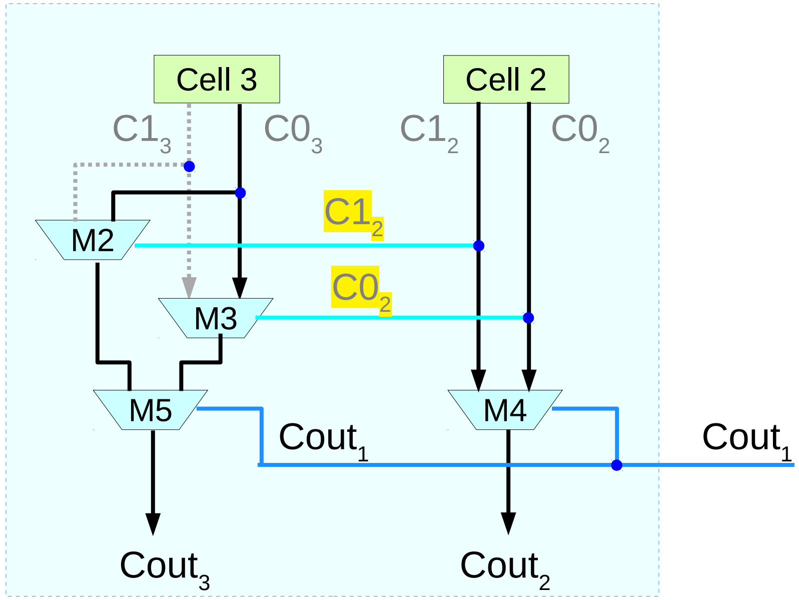
when $\overline{Cout_1}$ *is true*

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

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

When $C1_2 \cdot C0_2$ and $\overline{C1_2} \cdot \overline{C0_2}$





when $C1_2 \cdot C0_2$ $\boxed{Generate}$

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

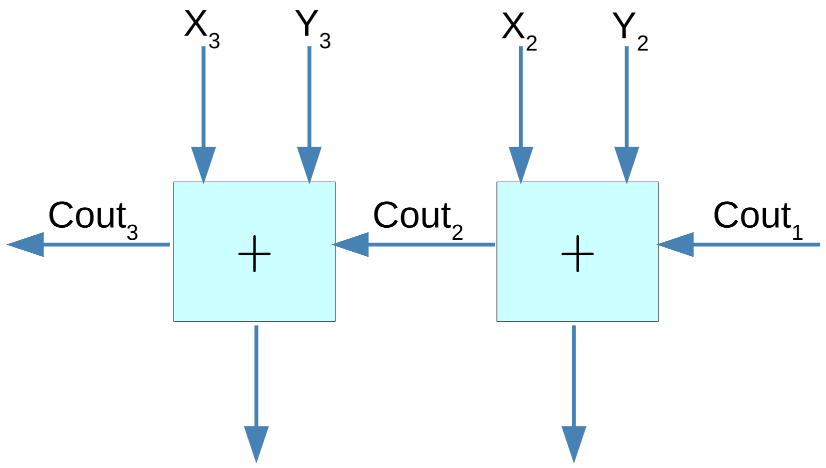
when $\overline{C1_2} \cdot \overline{C0_2}$ $\boxed{Kill}$

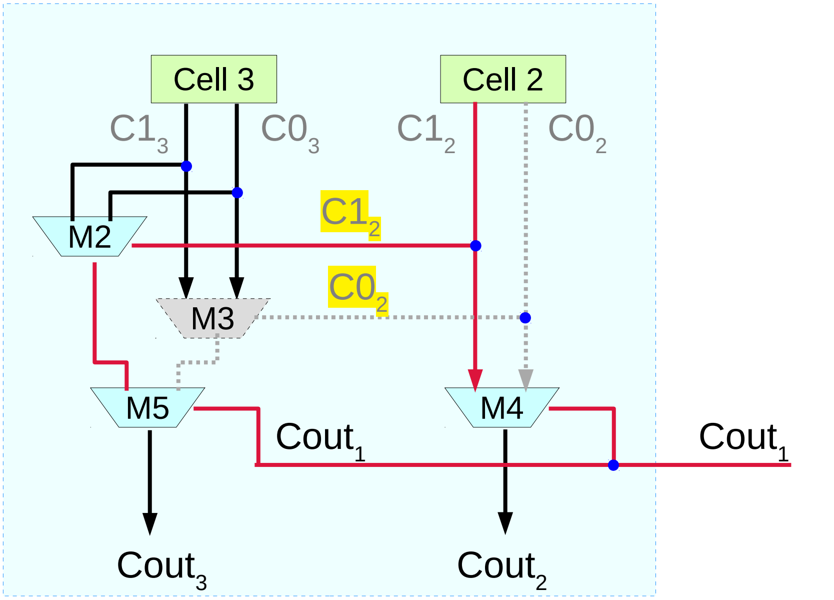
$(C0_3\, \overline{C1_2})Cout_1 + (C0_3\, \overline{C0_2})\overline{Cout_1}$

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

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

When Cout1 = 1





$$C1_3 \cdot C1_2 \cdot Cout_1$$

P	P
P	G
G	P
G	G

$$C0_3 \cdot \overline{C1_2} \cdot Cout_1$$

G	$\overline{PG}$

P = X ⊕ Y

G = X · Y

C1 = P + G

C0 = G

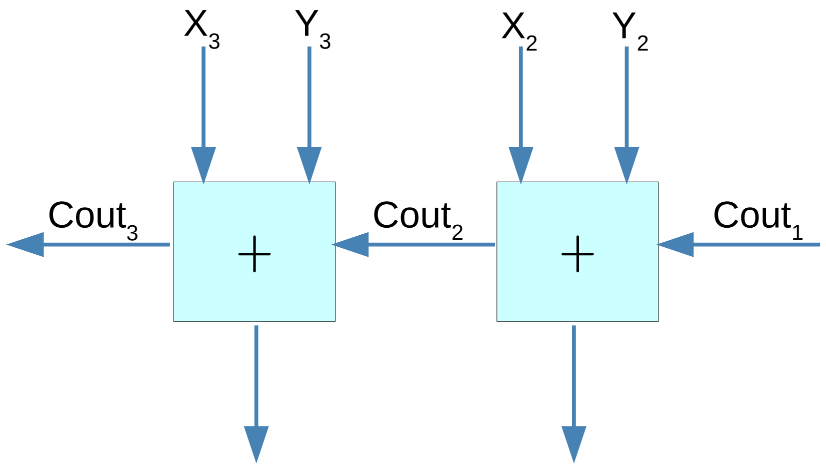
when *$Cout_1$* *is true* $(C1_3 C1_2 + C0_3 \overline{C1_2}) Cout_1$

X	Y	C1	C0	
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	$X\,Y$

$C1 = X + Y$ $C0 = X \cdot Y$

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

When Cout1 = 0



$$C1_3 \cdot C0_2 \cdot \overline{Cout}_1$$

P G
G G

$$C0_3 \cdot \overline{C0_2} \cdot \overline{Cout}_1$$

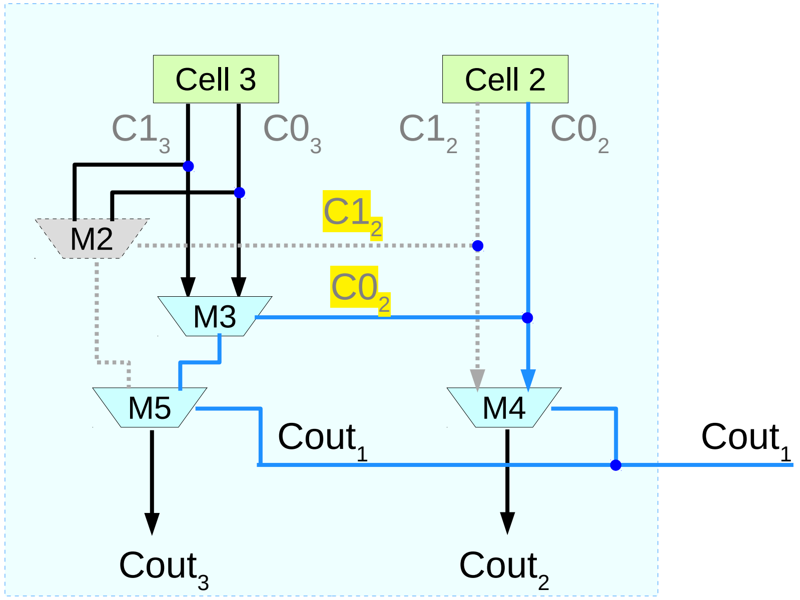
P $\overline{P}$G
P P

P = X ⊕ Y

G = X · Y

C1 = P + G

C0 = G



when $\overline{Cout_1}$ *is true*

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

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

Generate and Kill conditions

$$C1_3 \cdot C1_2 \cdot Cout_1$$

P	P	*Propagate*
G	P	
P	G	*Generate*
G	G	

$$C1_3 \cdot C0_2 \cdot \overline{Cout}_1$$

P	G	*Generate*
G	G	

$$C0_3 \cdot \overline{C1_2} \cdot Cout_1$$

G	$\overline{P}\,\overline{G}$	*Kill*

$$C0_3 \cdot \overline{C0_2} \cdot \overline{Cout}_1$$

P	$\overline{P}\,\overline{G}$	*Kill*
P	P	*Propagate*

P = X ⊕ Y

G = X · Y

C1 = P + G

C0 = G

P = $C1\overline{C0}$

G = C0

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

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

Propagate, Generate, and Kill conditions

		C1	C0	Name	
$\overline{P}$	$\overline{G}$	0	0	0	Kill
0	1	0	1	$\overline{Cin}$	Inverse Propagate
P	0	1	0	Cin	Propagate
0	G	1	1	1	Generate

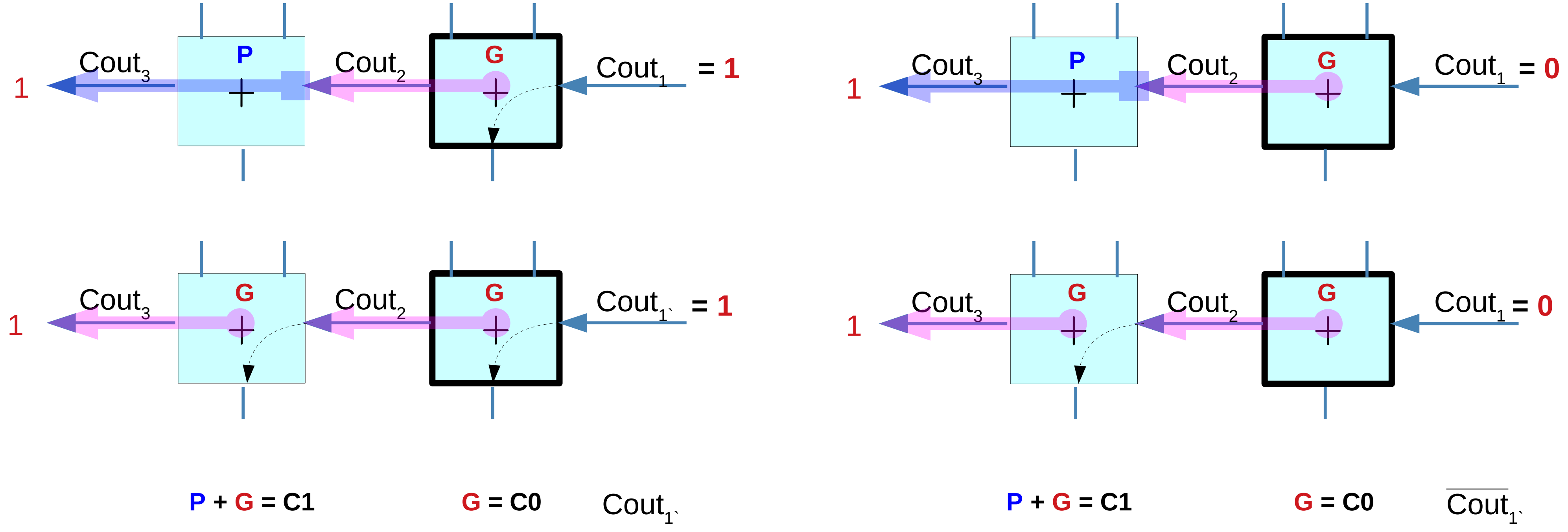
$\overline{C1}=\overline{X}\,\overline{Y}$	$\overline{C0}=\overline{X}\,Y+X\,\overline{Y}+\overline{X}\,\overline{Y}$	$\overline{C1}\,\overline{C0}$	$\overline{X}\,\overline{Y}$	$\boxed{\mathit{Kill}}$
$\overline{C1}=\overline{X}\,\overline{Y}$	$C0=X\,Y$	$\overline{C1}\,C0$		
$C1=\overline{X}\,Y+X\,\overline{Y}+X\,Y$	$\overline{C0}=\overline{X}\,Y+X\,\overline{Y}+\overline{X}\,\overline{Y}$	$C1\,\overline{C0}$	$\overline{X}\,Y+X\,\overline{Y}$	$\boxed{\mathit{Propagate}}$
$C1=\overline{X}\,Y+X\,\overline{Y}+X\,Y$	$C0=X\,Y$	$C1\,C0$	$X\,Y$	$\boxed{\mathit{Generate}}$

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

Generate conditions

when $C1_2 \cdot C0_2$ $\boxed{Generate}$

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

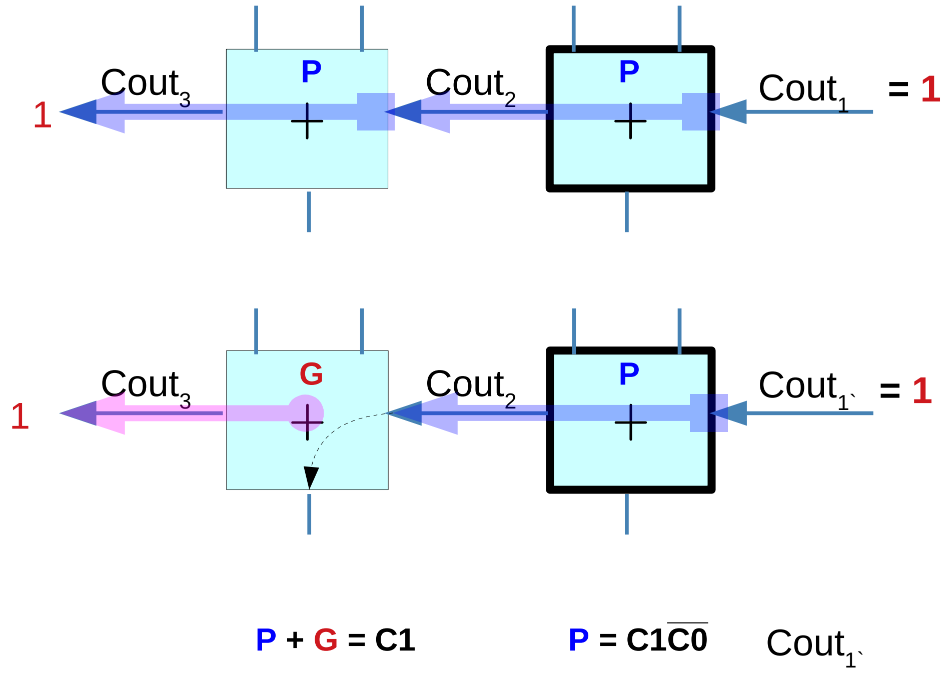


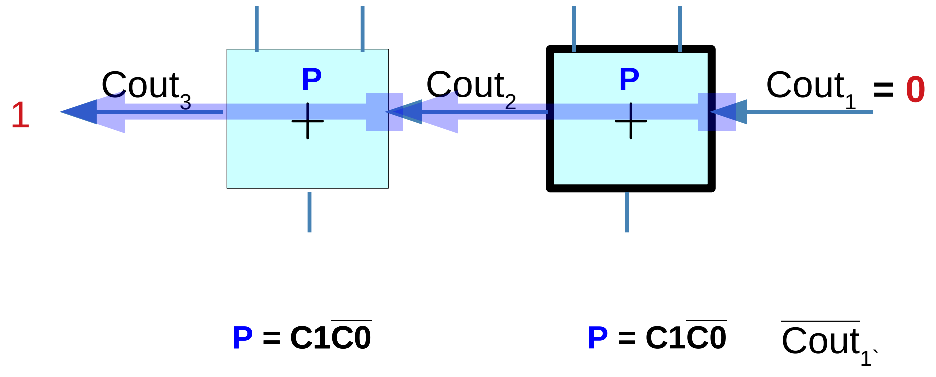
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Propagate conditions

when $C1_2 \cdot \overline{C0_2}$ $\boxed{Propagate}$

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



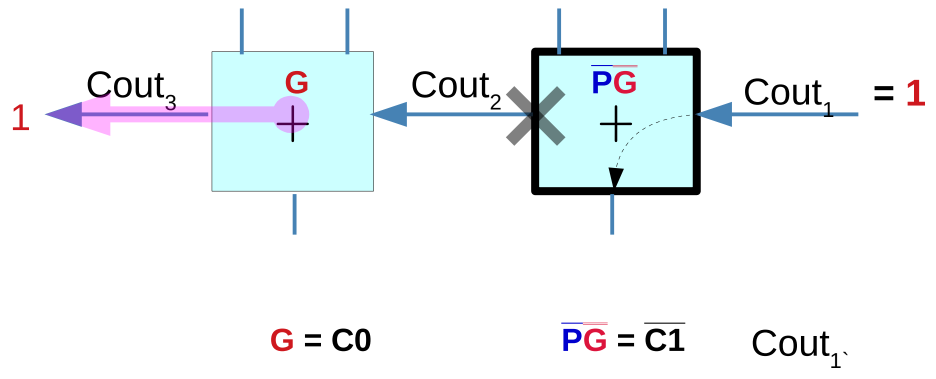


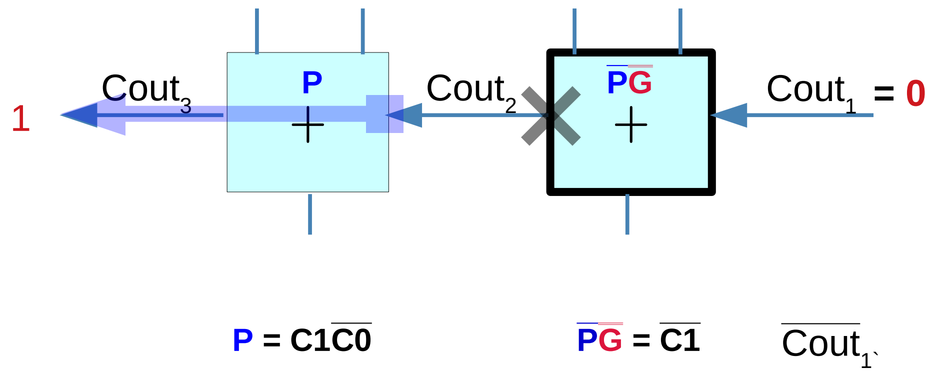
High Performance Carry Chains for FPGAs, S. Hauck, M. M. Hosler, T. W. Fry

Kill conditions

when $\overline{C1_2} \cdot \overline{C0_2}$ $\boxed{Kill}$

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





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