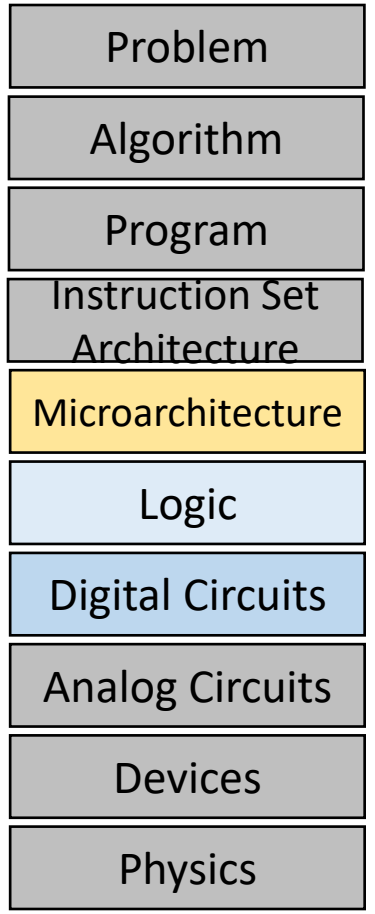
Universität Bremen

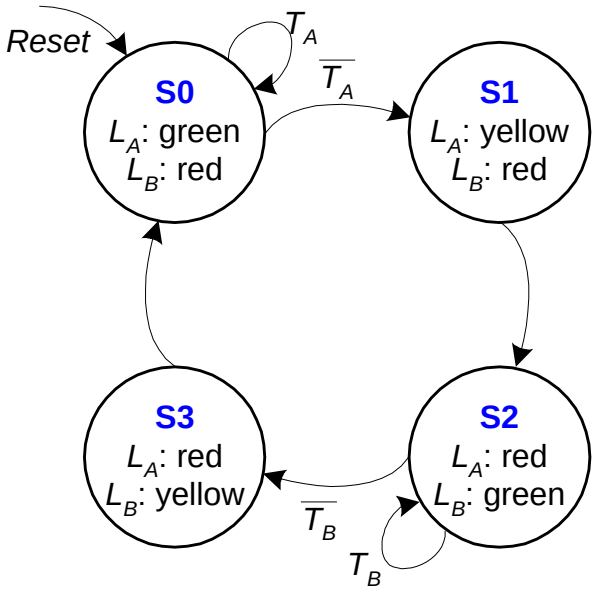
Arithmetic for Computer

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck

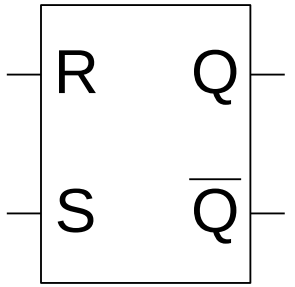


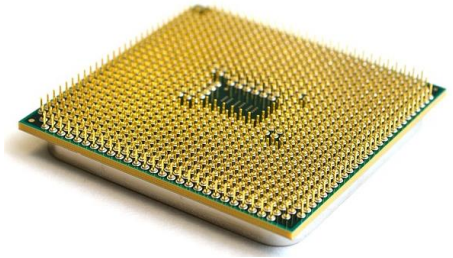
Revision

Synchronization

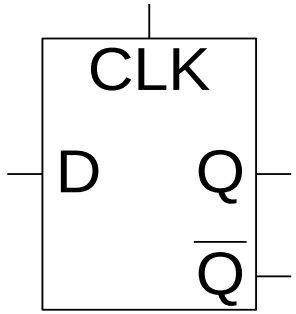


SR Latch Symbol





D Flip-flop



Universität Bremen

Agenda

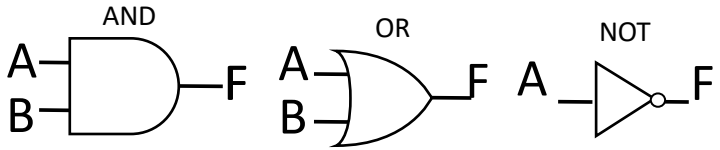
- Arithmetic for Computers
  - Operations on integers
  - Addition and subtraction
  - Dealing with overflow
  - Multiplication

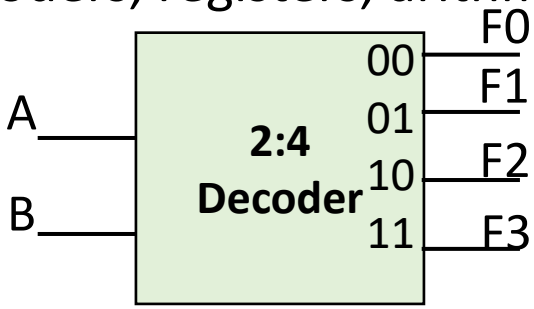
Universität Bremen

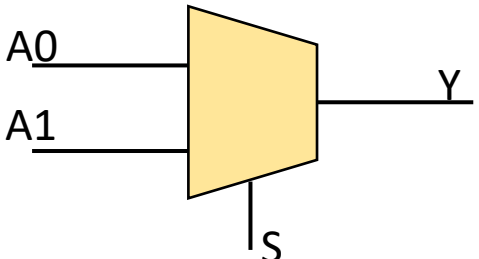
Efficient Processing

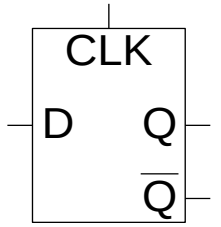
Introduction

- Digital building blocks:
  - Gates, multiplexers, decoders, registers, arithmetic circuits, counters, memory arrays, logic arrays





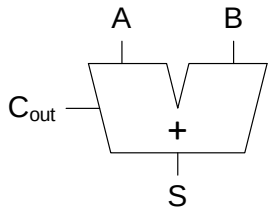


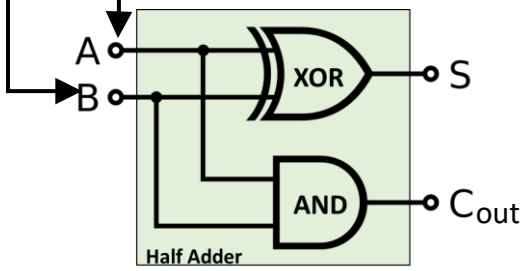


- Building blocks demonstrate hierarchy, modularity, and regularity:
  - Hierarchy of simpler components
  - Well-defined interfaces and functions
  - Regular structure easily extends to different sizes
- We'll use these building blocks in next lectures to build a microprocessor

1-bit Adders

Half Adder

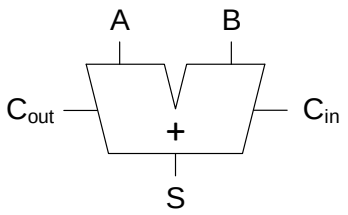




A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

S =
C_{out} =

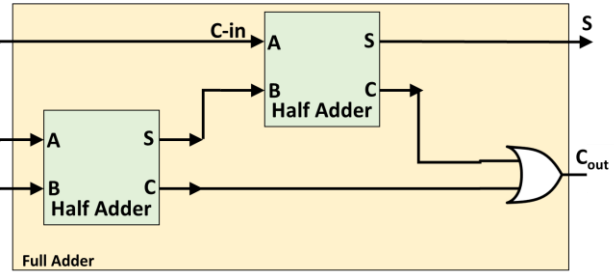
Full Adder



C_{in}	A	B	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

S =
C_{out} =

One way to do it



Full (1-bit) Adder – Method 2

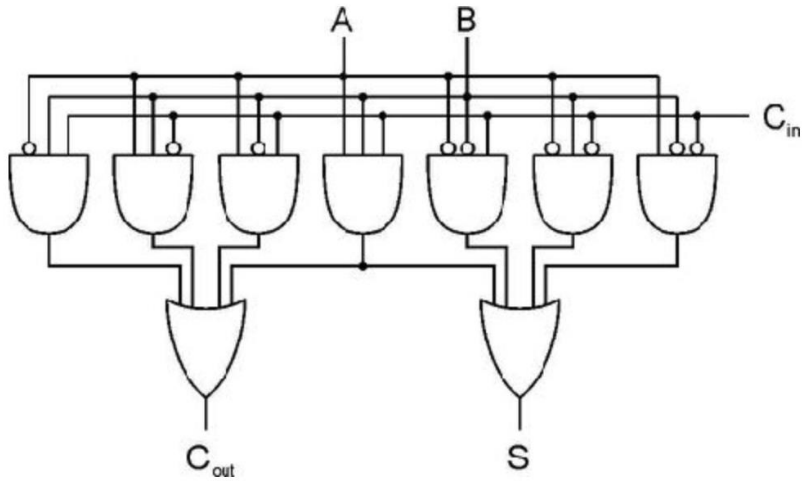
Add two bits and carry-in, produce one-bit sum and carry-out.

Truth Table

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

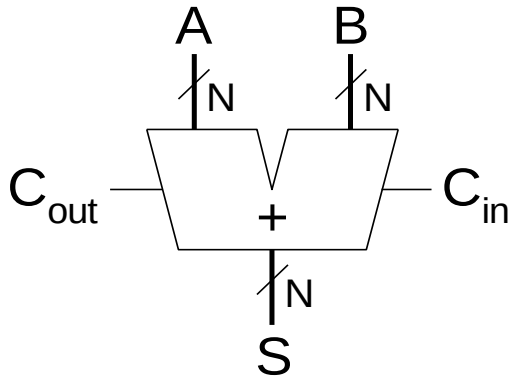
S = (~A & ~B & Cin)
| (~A & B & ~Cin)
| (A & ~B & ~Cin)
| (A & B & Cin)

C_{out} = (~A & B & Cin)
| (A & ~B & Cin)
| (A & B & ~Cin)
| (A & B & Cin)



Multibit Adders

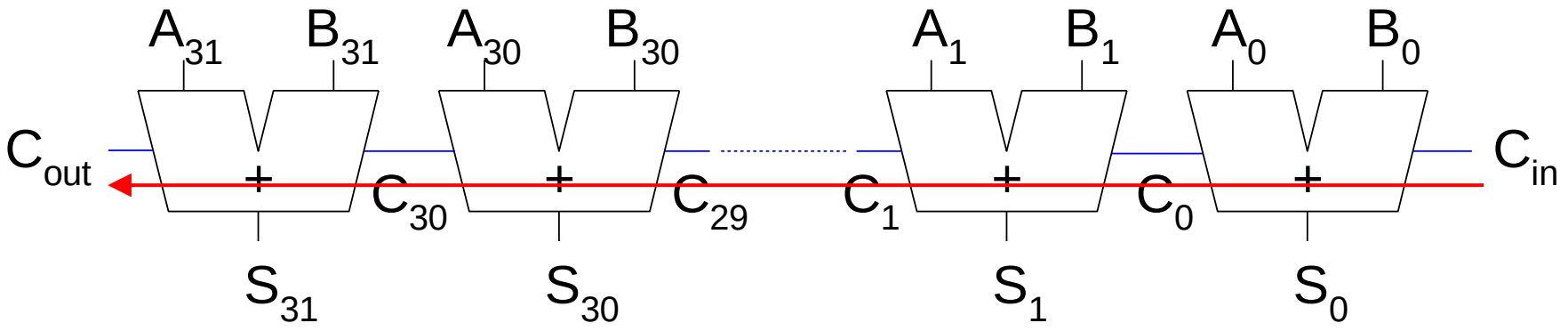
- Also called – Carry Propagate Adders (CPAs):
  - Ripple-carry (slow)
  - Carry-lookahead (CLA) (fast)
  - Prefix (faster)
- Carry-lookahead and prefix adders faster for large adders but require more hardware



Ripple Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: slow

Delay depends on the number of adders



Revisiting the Truth Table

1-bit Full Adder

A	B	C_{in}	C_{out}	Comments
0	0	0	0	No carry
0	0	1	0	No carry
0	1	0	0	No carry
0	1	1	1	Propagate
1	0	0	0	No carry
1	0	1	1	Propagate
1	1	0	1	Generate
1	1	1	1	Generate/ Propagate

Carry Generate $= G_i$

Carry Propagate $= P_i$

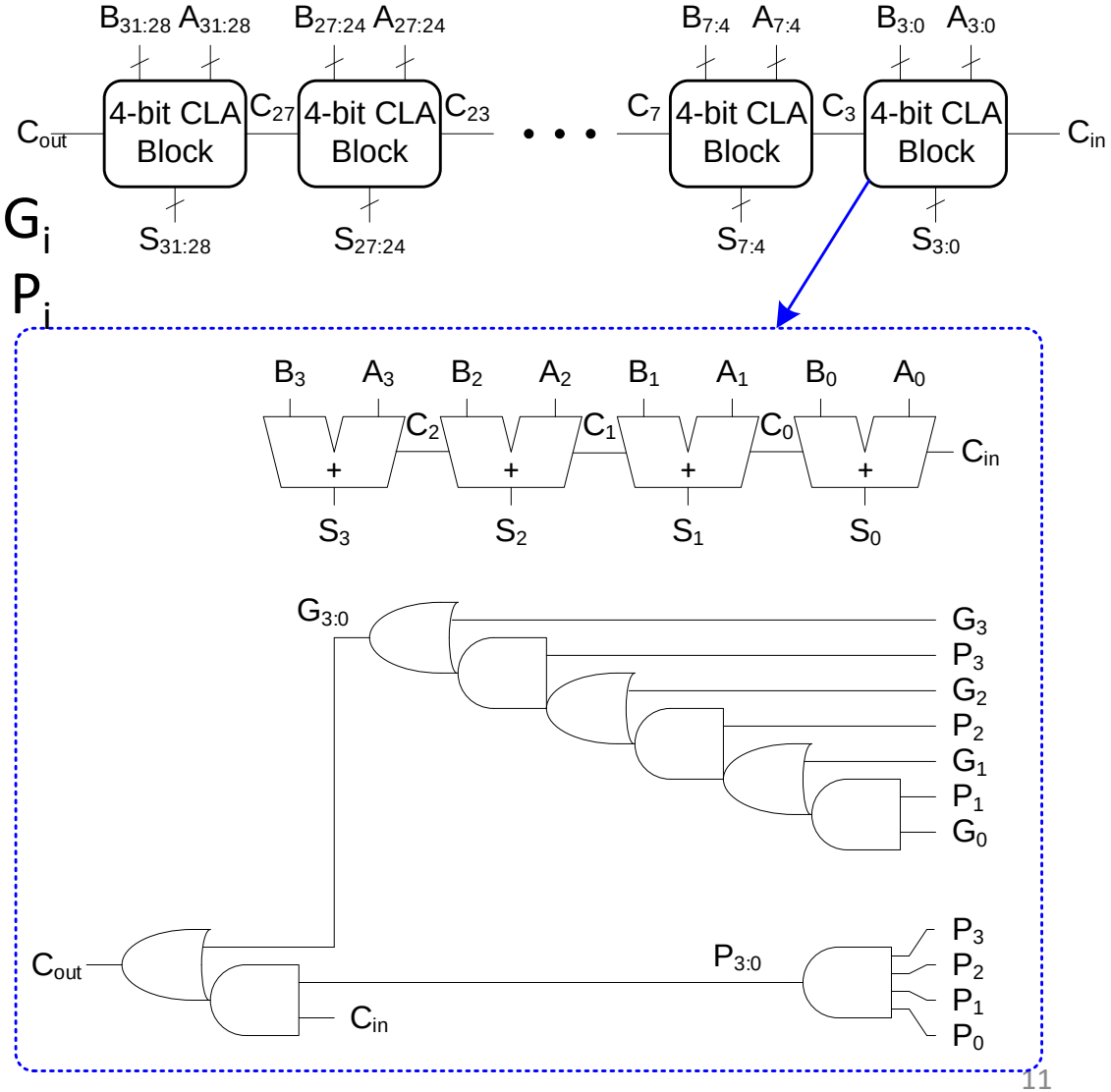
32-bit CLA with 4-bit Blocks

1-bit Full Adder

A	B	C_{in}	C_{out}	Comments
0	0	0	0	No carry
0	0	1	0	No carry
0	1	0	0	No carry
0	1	1	1	Propagate
1	0	0	0	No carry
1	0	1	1	Propagate
1	1	0	1	Generate
1	1	1	1	Generate/ Propagate

Carry Generate $= G_i$

Carry Propagate $= P_i$



Carry Look-ahead Adder

- Compute C_{out} for k-bit blocks using generate and propagate signals
  - **Some definitions:**
  - Column i produces a carry out by either ***generating*** a carry out or ***propagating*** a carry in to the carry out
  - Calculate generate (G_i) and propagate (P_i) signals for each column:
    - **Generate:** Column i will generate a carry out if A_i and B_i are both 1.

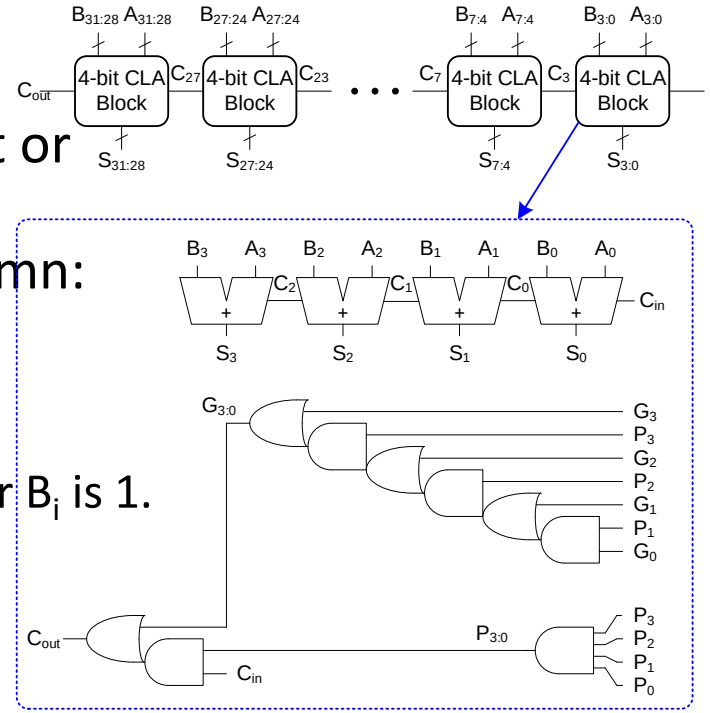
$$G_i = A_i B_i$$

    - **Propagate:** Column i will propagate a carry in to the carry out if A_i or B_i is 1.

$$P_i = A_i + B_i$$

    - **Carry out:** The carry out of column i (C_i) is:

$$C_i = A_i B_i + (A_i + B_i) C_{i-1} = G_i + P_i C_{i-1}$$



Example – Propagate and Generate

- Examples: Column propagate and generate signals:
  - Column propagate: $P_i = A_i + B_i$
  - Column generate: $G_i = A_i B_i$

A	B	C_{in}	C_{out}	Comments
0	0	0	0	No carry
0	0	1	0	No carry
0	1	0	0	No carry
0	1	1	1	Propagate
1	0	0	0	No carry
1	0	1	1	Propagate
1	1	0	1	Generate
1	1	1	1	Generate/ Propagate

1011	$A_{3:0}$
+ 0110	$B_{3:0}$

1011	$A_{3:0}$
+ 1001	$B_{3:0}$

$$C_i = G_i + P_i C_{i-1}$$

Universität Bremen

Block Propagate and Generate

- Now use column Propagate and Generate signals to compute ***Block Propagate*** and ***Block Generate*** signals for k-bit blocks, i.e.:
  - Compute if a ***k-bit group*** will ***propagate*** a carry in (of the block) to the carry out (of the block)
  - Compute if a ***k-bit group*** will ***generate*** a carry out (of the block)

Example

- Example: 4-bit blocks
  - **Block propagate signal**: $P_{3:0}$ (single-bit signal)
    - A carry-in would propagate through all 4 bits of the block:

$$P_{3:0} = P_3 P_2 P_1 P_0$$

- Examples:

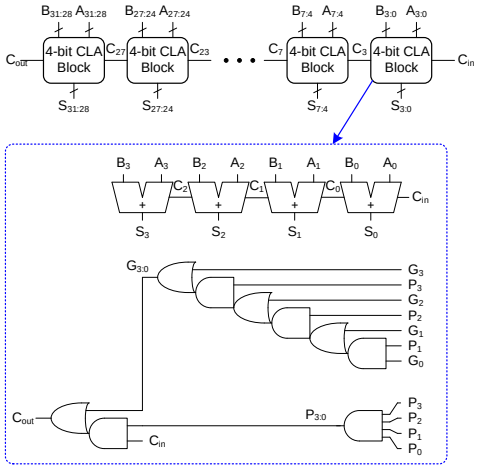
```
  1011
+ 0100
------
```

```
  1011
+ 0001
------
```

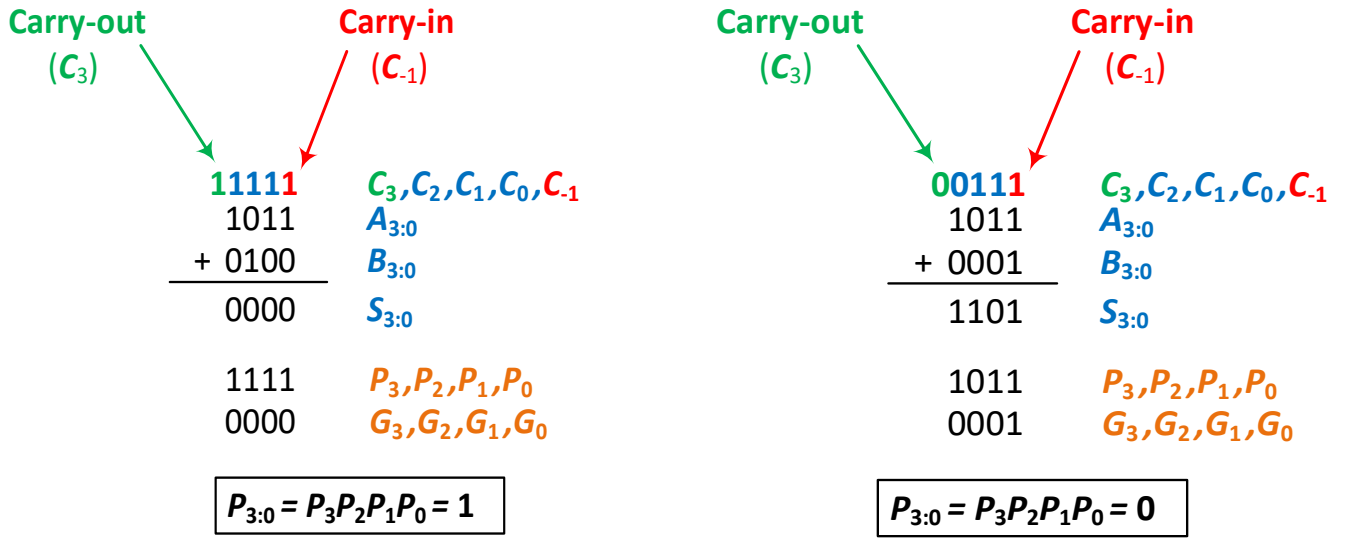
Example

- Example: 4-bit blocks
  - **Block propagate signal**: $P_{3:0}$ (single-bit signal)
    - A carry-in would propagate through all 4 bits of the block:

$$P_{3:0} = P_3P_2P_1P_0$$



- Examples:



Example cont …

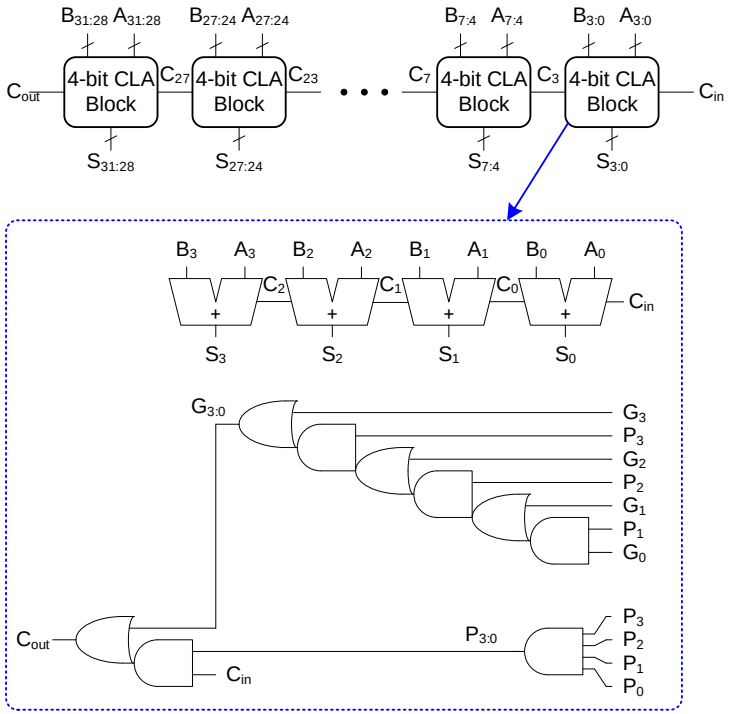
- Example: 4-bit blocks
  - ***Block propagate signal:*** $P_{3:0}$ (single-bit signal)
    - A carry-in would propagate through all 4 bits of the block

$$P_{3:0} = P_3P_2P_1P_0$$

  - ***Block generate signal:*** $G_{3:0}$ (single-bit signal)
    - A carry is generated:
      - in column 3, **or**
      - in column 2 and propagated through column 3, **or**
      - in column 1 and propagated through columns 2 and 3, **or**
      - in column 0 and propagated through columns 1-3

$$G_{3:0} = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$$

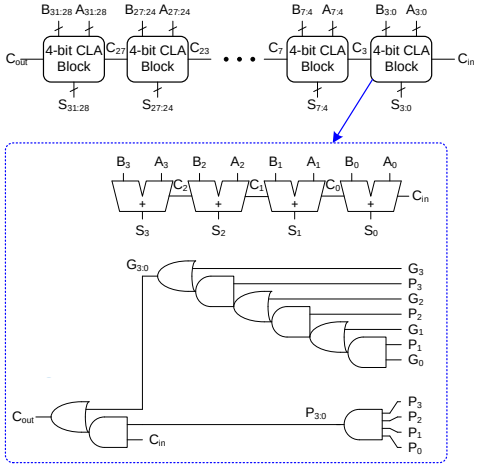
$$G_{3:0} = G_3 + P_3\,[G_2 + P_2\,(G_1 + P_1G_0)]$$



Example cont …

- Example: 4-bit blocks
  - ***Block generate signal: $G_{3:0}$*** (single-bit signal)
    - A carry is: generated in column 3, **or** generated in column 2 and propagated through column 3, **or** …

$$G_{3:0} = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$$

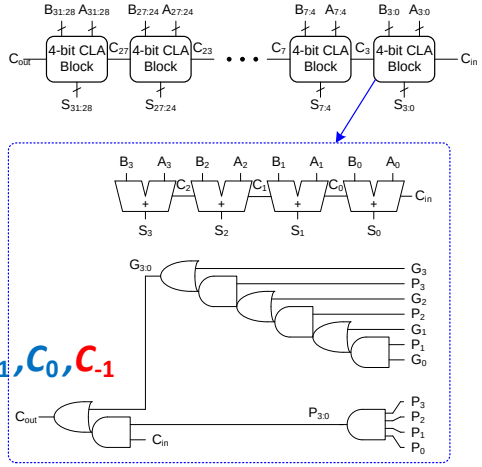


1001	1110	0110
+ 1100	+ 0100	+ 0010

Example cont …

- Example: 4-bit blocks
  - ***Block generate signal: $G_{3:0}$*** (single-bit signal)
    - A carry is: generated in column 3, **or** generated in column 2 and propagated through column 3, **or** …

$$G_{3:0} = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$$



Carry-out (C_3)		
10011	C_3,C_2,C_1,C_0,C_{-1}	
1001	$A_{3:0}$	
+ 1100	$B_{3:0}$	
0110	$S_{3:0}$	
1101	P_3,P_2,P_1,P_0	
1000	G_3,G_2,G_1,G_0	
$G_{3:0} = 1$		

Carry-out (C_3)		
11000	C_3,C_2,C_1,C_0,C_{-1}	
1110	$A_{3:0}$	
+ 0100	$B_{3:0}$	
0010	$S_{3:0}$	
1110	P_3,P_2,P_1,P_0	
0100	G_3,G_2,G_1,G_0	
$G_{3:0} = 1$		

Carry-out (C_3)		
01101	C_3,C_2,C_1,C_0,C_{-1}	
0110	$A_{3:0}$	
+ 0010	$B_{3:0}$	
1000	$S_{3:0}$	
0110	P_3,P_2,P_1,P_0	
0010	G_3,G_2,G_1,G_0	
$G_{3:0} = 0$		

Example cont …

- Example: 4-bit blocks
  - Block propagate signal: $P_{3:0}$ (single-bit signal)
    - A carry-in would propagate through all 4 bits of the block:

$$P_{3:0} = P_3 P_2 P_1 P_0$$

  - Block generate signal: G_{3-0} (single-bit signal)
    - A carry is generated:
      - in column 3, **or**
      - in column 2 and propagated through column 3, **or**
      - in column 1 and propagated through columns 2 and 3, **or**
      - in column 0 and propagated through columns 1-3

$$G_{3:0} = G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3$$

$$G_{3:0} = G_3 + P_3 [G_2 + P_2 (G_1 + P_1 G_0)]$$

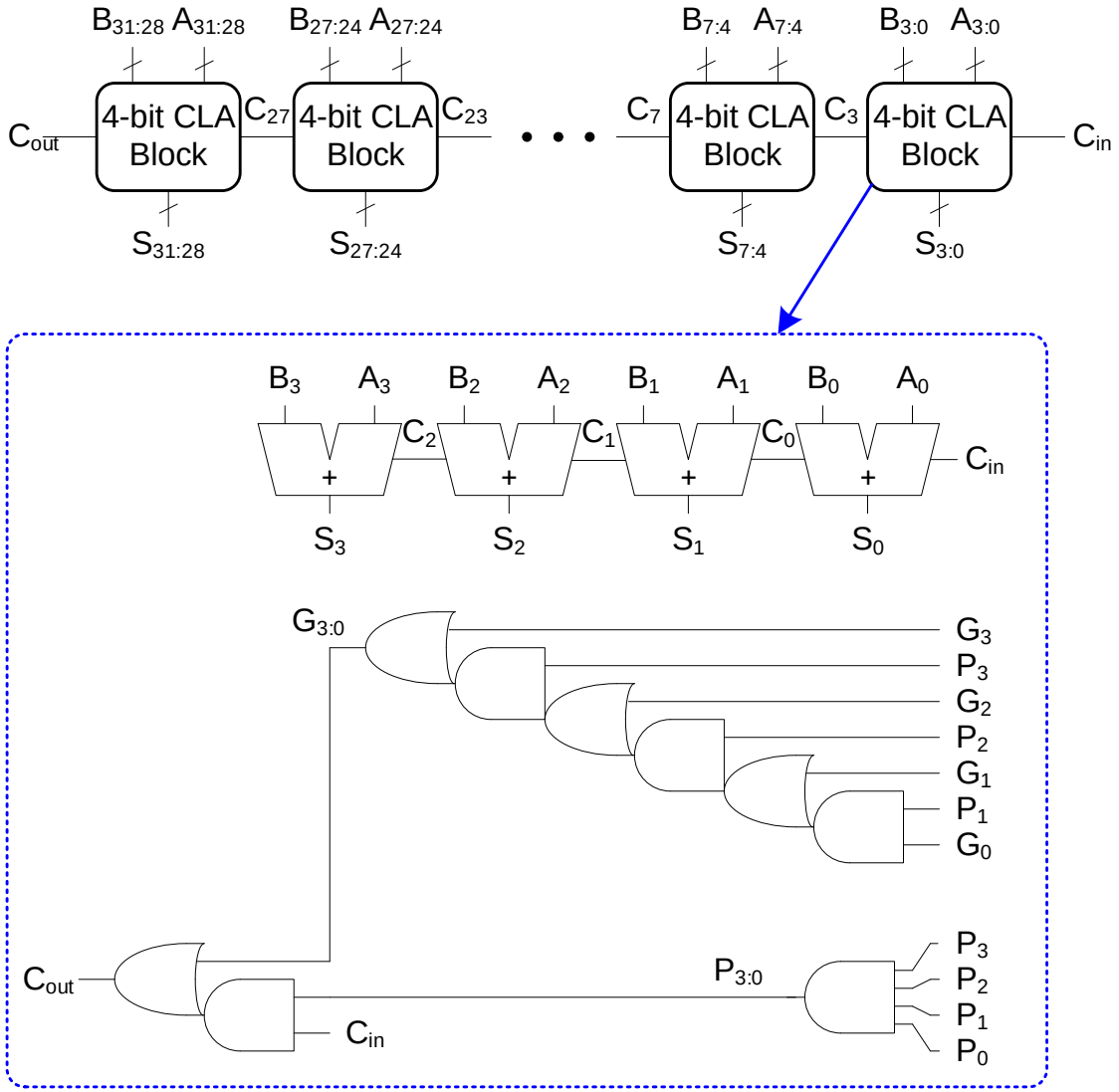
$$\boldsymbol{C_3 = G_{3:0} + P_{3:0} C_{-1}}$$

Example cont …

- Example: Block propagate and generate signals for 4-bit blocks ($P_{3:0}$ and $G_{3:0}$):

$$P_{3:0} = P_3 P_2 P_1 P_0$$
$$G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0))$$
$$C_3 = G_{3:0} + P_{3:0} C_{-1}$$

32-bit CLA with 4-bit Blocks



Universität Bremen

CLA Addition

- Step 1: Compute G_i and P_i for all columns
- Step 2: Compute G and P for k-bit blocks
- Step 3: C_{in} propagates through each k-bit propagate/generate logic (meanwhile computing sums)
- Step 4: Compute sum for most significant k-bit block

CLA Addition cont …

- Step 1: Compute G_i and P_i for all columns

$$G_i = A_i B_i$$

$$P_i = A_i + B_i$$

CLA Addition cont …

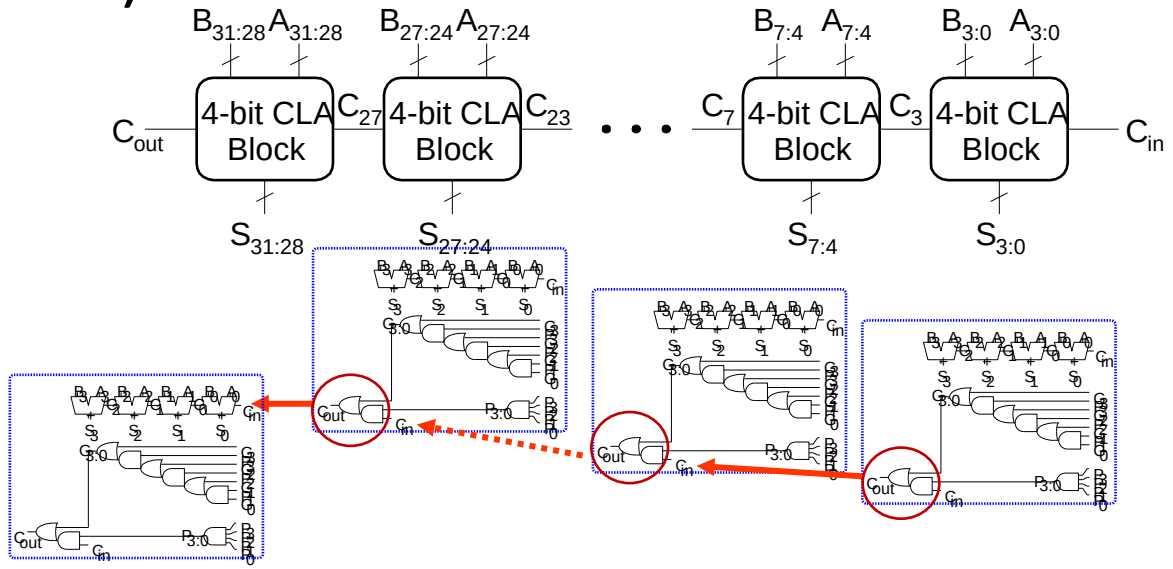
- Step 1: Compute G_i and P_i for all columns
- Step 2: Compute G and P for k-bit blocks

$$P_{3:0} = P_3 P_2 P_1 P_0$$

$$G_{3:0} = G_3 + P_3 (G_2 + P_2 (G_1 + P_1 G_0))$$

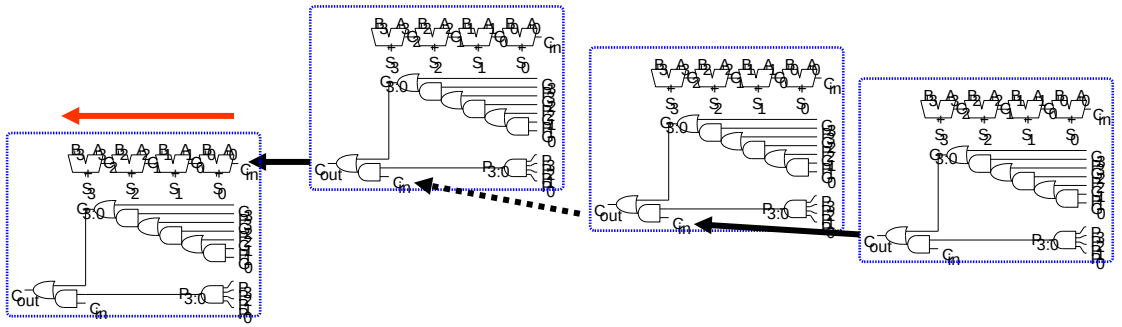
CLA Addition cont …

- Step 1: Compute G_i and P_i for all columns
- Step 2: Compute G and P for k-bit blocks
- Step 3: C_{in} propagates through each k-bit propagate/generate logic (meanwhile computing sums)



CLA Addition cont …

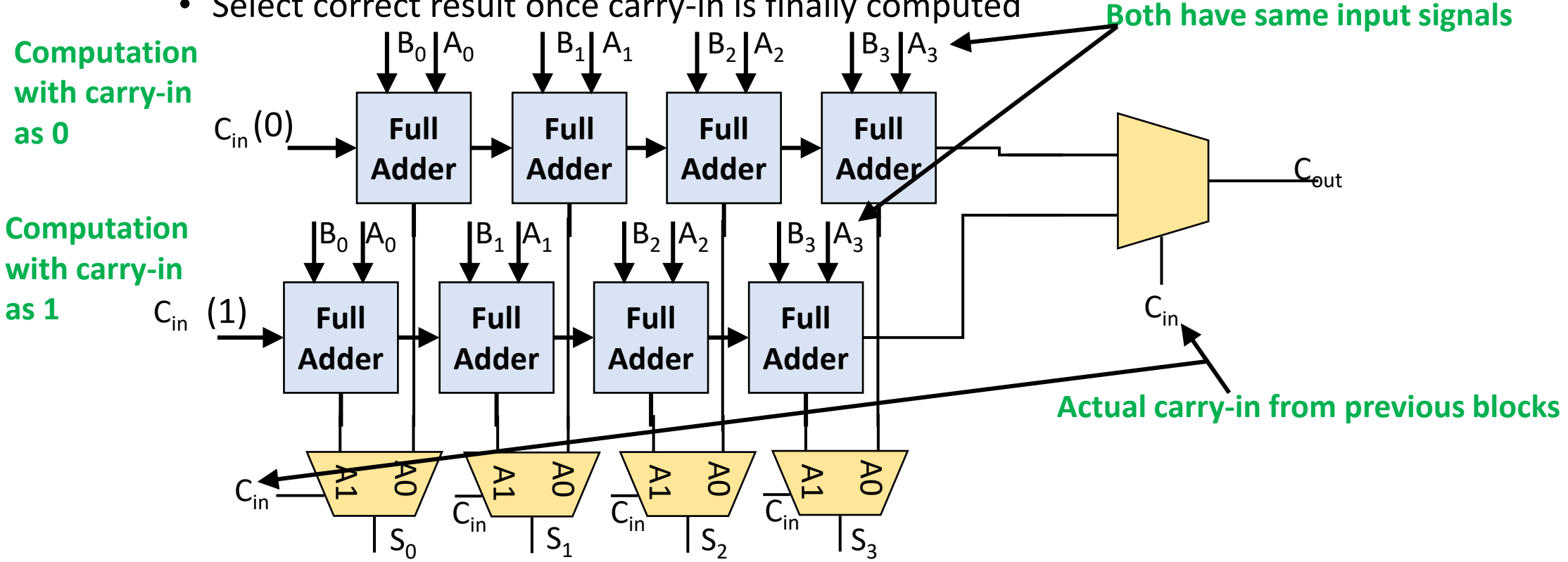
- Step 1: Compute G_i and P_i for all columns
- Step 2: Compute G and P for k-bit blocks
- Step 3: C_{in} propagates through each k-bit propagate/generate logic (meanwhile computing sums)
- Step 4: Compute sum for most significant k-bit block



Carry-select Adder

- Redundant hardware to make carry calculation go faster
  - Compute two high-order sums in parallel while waiting for carry-in
  - One assuming carry-in is 0 and another assuming carry-in is 1
  - Select correct result once carry-in is finally computed

Full adders can be replaced by Ripple carry adder or Carry Look-ahead adder



Integer Addition

- Example: 7 + 6

	(1)	(1)	(0)	←	(Carries)
+7:	0	1	1	1	
+6 :	0	1	1	0	
+13:	1	1	0	1	

- Overflow if result out of range
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

Integer Addition - Overflow

- Example: 15 + 15

(1)	(1)	(1)	(1)	←	(Carries)
+15:	1	1	1	1	
+15 :	1	1	1	1	
+30:	1	1	1	0	

- Overflow if result out of range
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

Shifters

- Logical shifter: shifts value to left or right and fills empty spaces with 0's
  - Ex: 11001 >> 2 = 00110
  - Ex: 11001 << 2 = 00100

Can be used as multipliers and dividers

- Rotator: rotates bits in a circle, such that bits shifted off one end are shifted into the other end
  - Ex: 11001 ROR 2 = 01110
  - Ex: 11001 ROL 2 = 00111

Multiplication

- Partial products formed by multiplying a single digit of the multiplier with multiplicand
- Shifted partial products summed to form result

Another cool way to do the multiplication
https://en.wikipedia.org/wiki/Lattice_multiplication

Decimal		**Binary**
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

230 x 42 = 9660

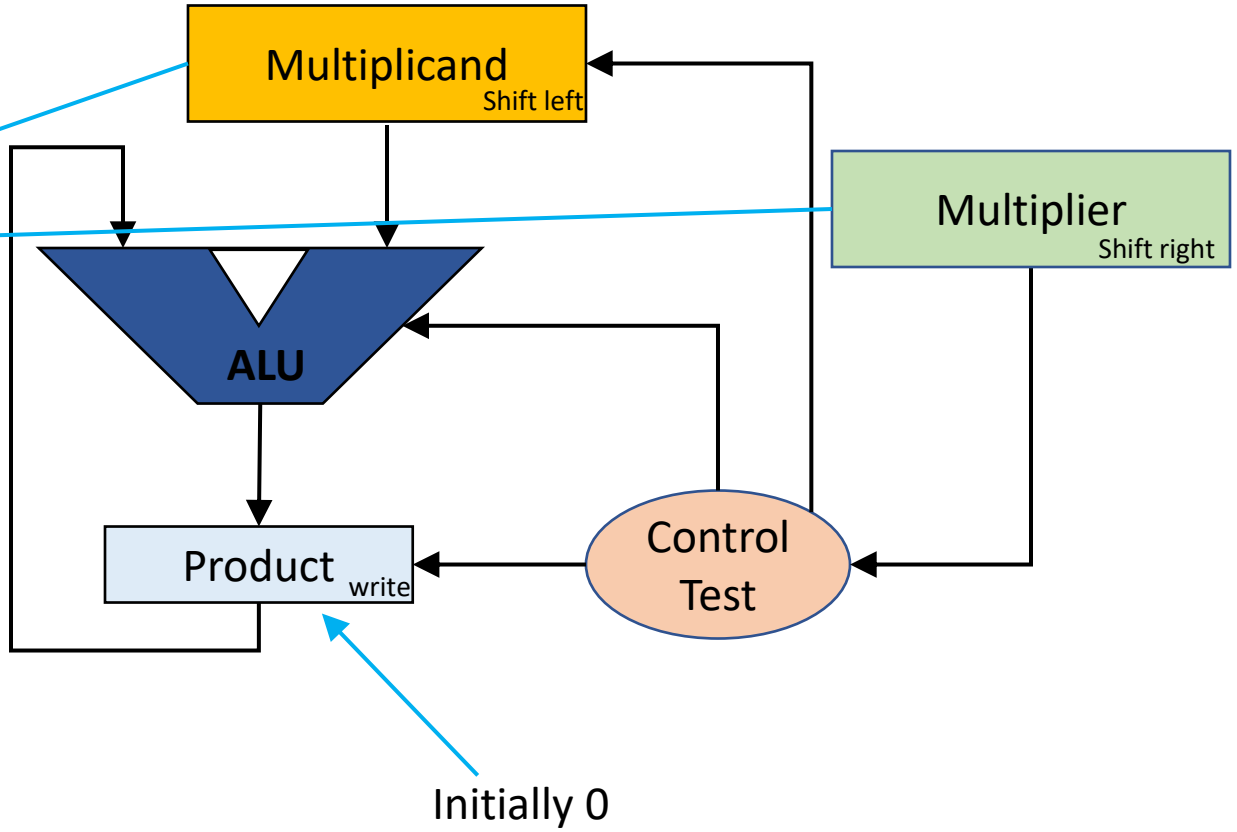
5 x 7 = 35

Slow 4-bit Multiplier

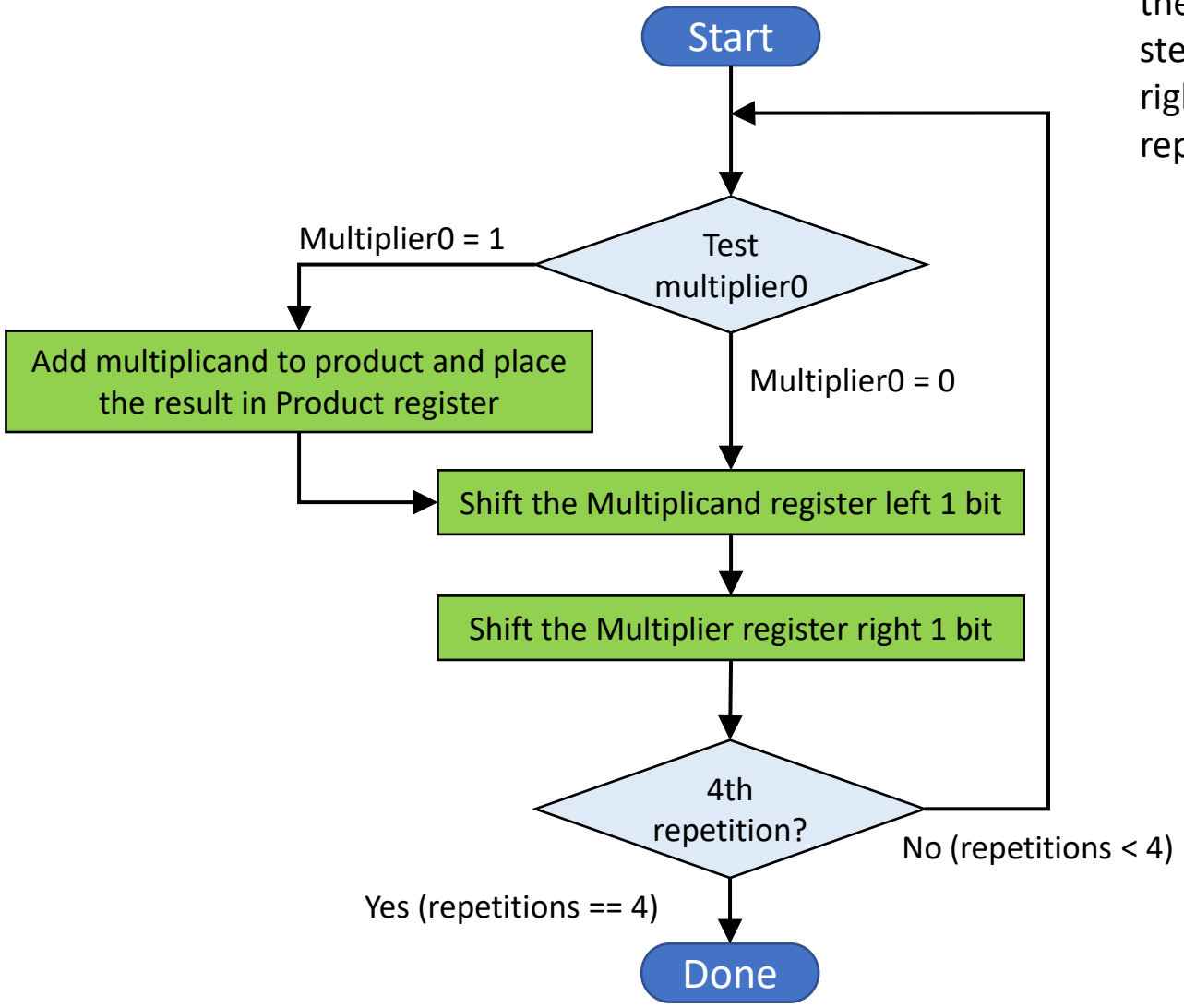
Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

230 x 42 = 9660

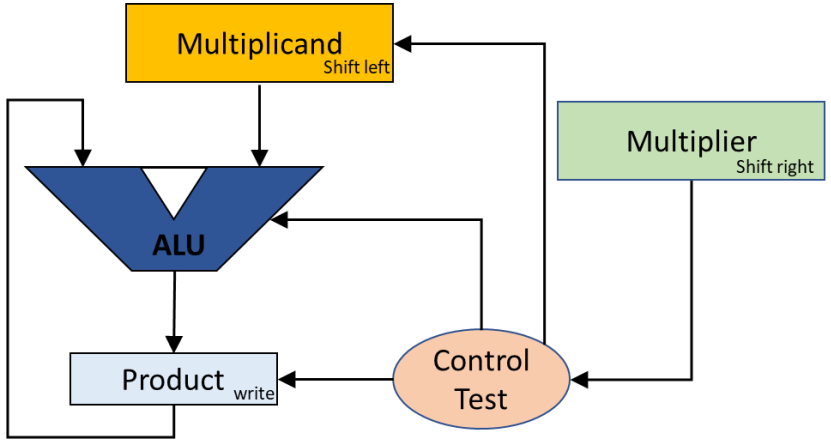
5 x 7 = 35



4-bit Multiplier



If the least significant bit of the multiplier is 1, add the multiplicand to the product. If not, go to the next step. Shift the multiplicand left and the multiplier right in the next two steps. These three steps are repeated 4 times.



Decimal		Binary
230	multiplicand	0101
x 42	multiplier	x 0111
460	partial products	0101
+ 920		0101
9660		0101
		+ 0000
	result	0100011

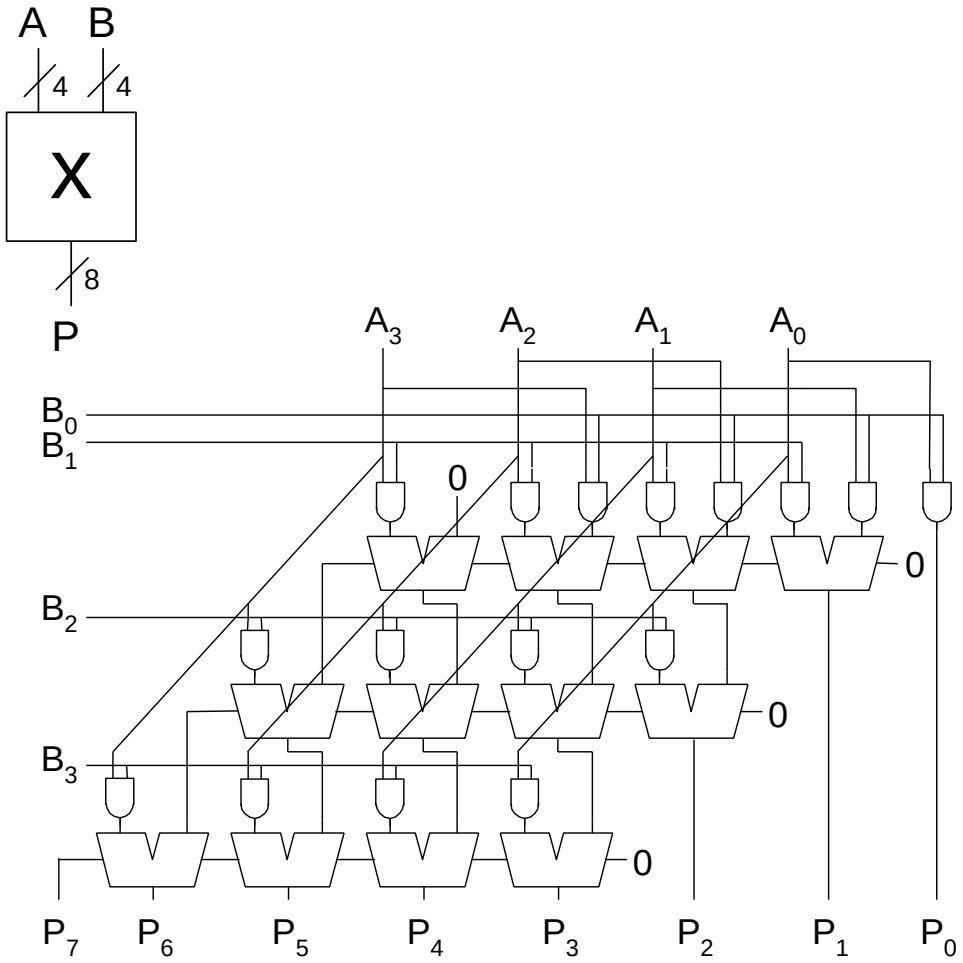
230 x 42 = 9660

5 x 7 = 35

Fast 4-bit Multiplier

- Uses multiple adders
  - Cost/performance tradeoff
- Can be pipelined
  - Parallel computation

$$
\begin{array}{rcccccccc}
 & & & A_3 & A_2 & A_1 & A_0 \\
 & \times & & B_3 & B_2 & B_1 & B_0 \\
\hline
 & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & \\
A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & \\
+ \quad A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$



Universität Bremen

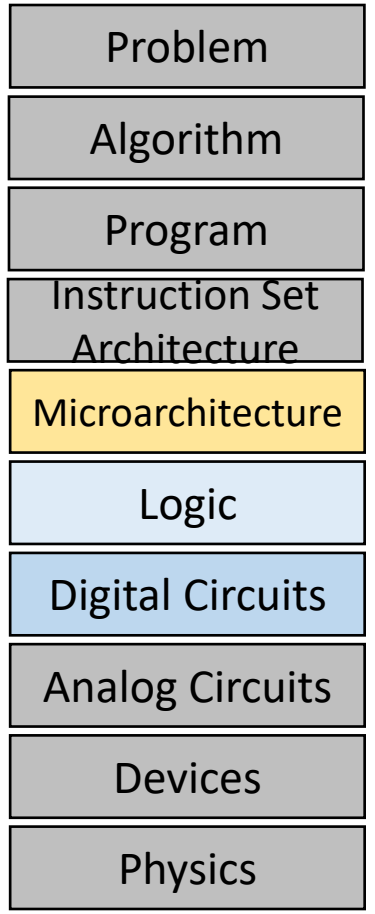
Fast Multipliers

- Array Multipliers
- Booth Multipliers
- Wallace Tree Multipliers

Universität Bremen

Arithmetic for Computer

Prof. Dr. Rolf Drechsler
Dr. Muhammad Hassan
M.Sc. Jan Zielasko
M.Sc. Milan Funck



Universität Bremen

Literature

- D. Patterson, J. Hennessy: Computer Organization and Design RISC-V Edition – The Hardware Software Interface, Elsevier, 2020.
- S. L. Harris, D. Harris: Digital Design and Computer Architecture, RISC-V Edition, Elsevier, 2021.
- ARM University program: Introduction-to-Computer-Architecture-Education.
- J. Teich, C. Haubelt: Digitale Hardware/Software-Systeme – Synthese und Optimierung, Springer Verlag, 2. Auflage, 2007.
- G. De Micheli: Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.
- G. D. Hachtel, F. Somenzi: Logic Synthesis and Verification Algorithms, Kluwer, 1996.

Universität Bremen

Literature

- H. Bähring, J. Dunkel, G. Rademacher: Mikrorechner-Systeme: Mikroprozessoren, Speicher, Peripherie, Springer-Verlag, 2. Auflage, 1994.
- J. P. Hayes: Computer Architecture and Organization, McGraw-Hill, 1998.
- M. G. Arnold: Verilog Digital Computer Design: Algorithms to Hardware, Prentice Hall, 1998.
- A. Tanenbaum: Structured Computer Organization, Prentice Hall, 5th Edition, 2006.
- D. A. Patterson, J. L. Hennessy: Computer Organization and Design - The Hardware/Software-Interface, Morgan Kaufmann, 3. Auflage, 2007.
- J. L. Hennessy, D. A. Patterson: Computer Architecture - A Quantitative Approach, Morgan Kaufmann, 4. Auflage, 2007.
- H. Kaeslin: Digital Integrated Circuit Design, From VLSI to CMOS Fabrication, Cambridge University Press, 2008.
- A. Biere, D. Kroening, G. Weissenbacher, C. M. Wintersteiger: Digitaltechnik – Eine praxisnahe Einführung, Springer-Verlag, 2008.