Outline

- Digital CMOS Design
- Arithmetic Operators
  - Adders
  - Comparators
  - Shifters
  - Multipliers

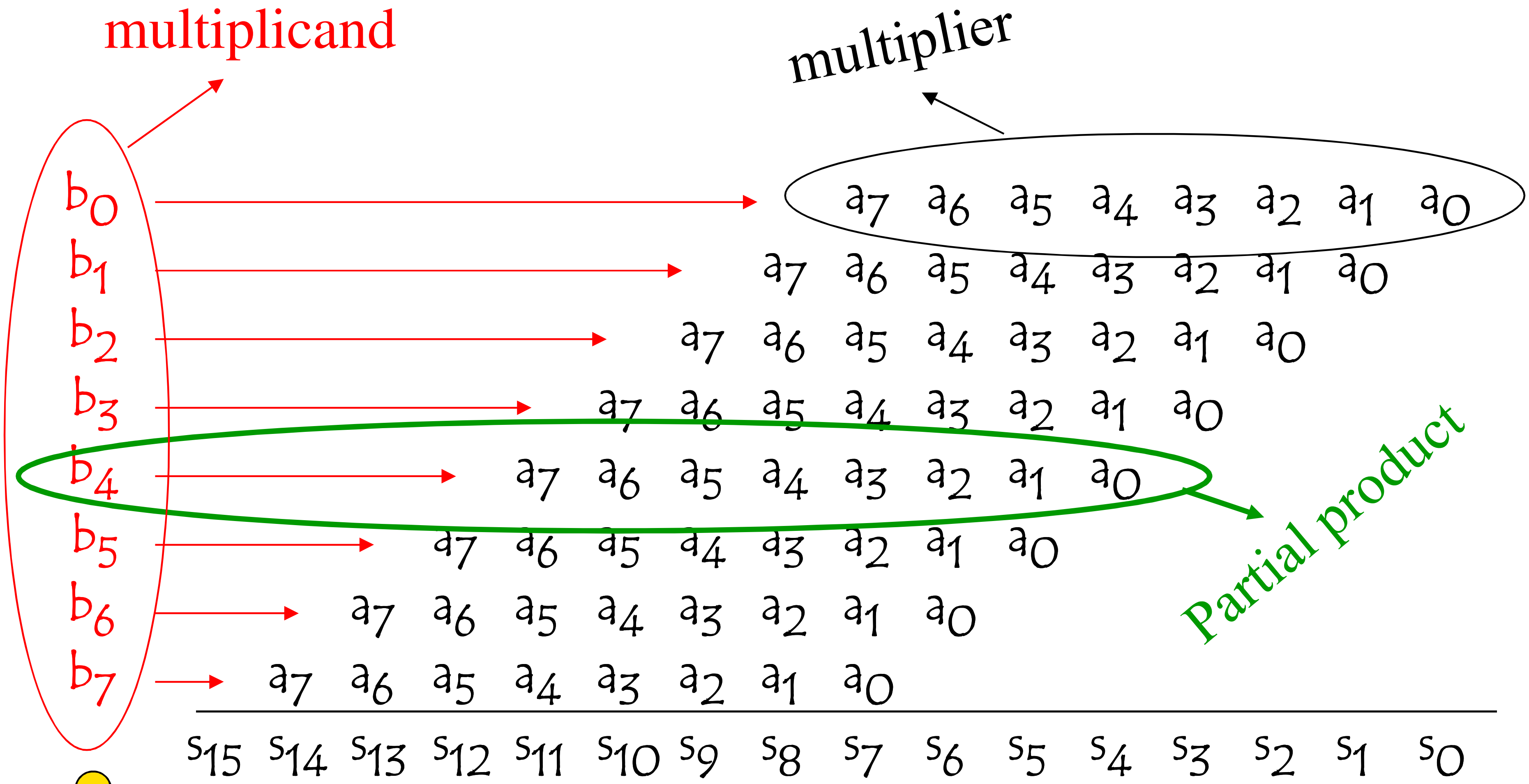
Multipliers

Two natural numbers a and b coded on n bits

the result of $a \times b$ is coded on 2n bits

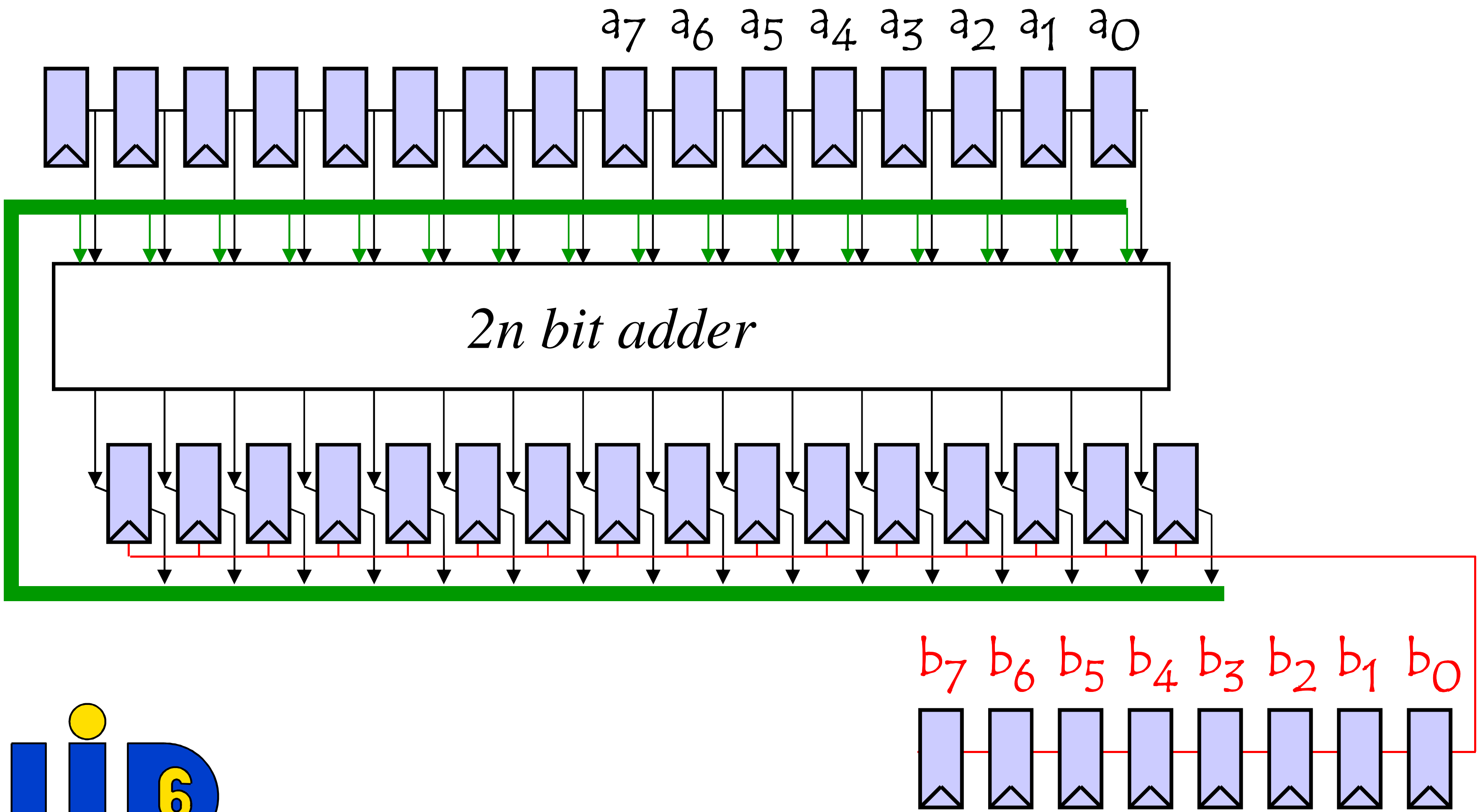
Classic method as learned in primary school

Multipliers



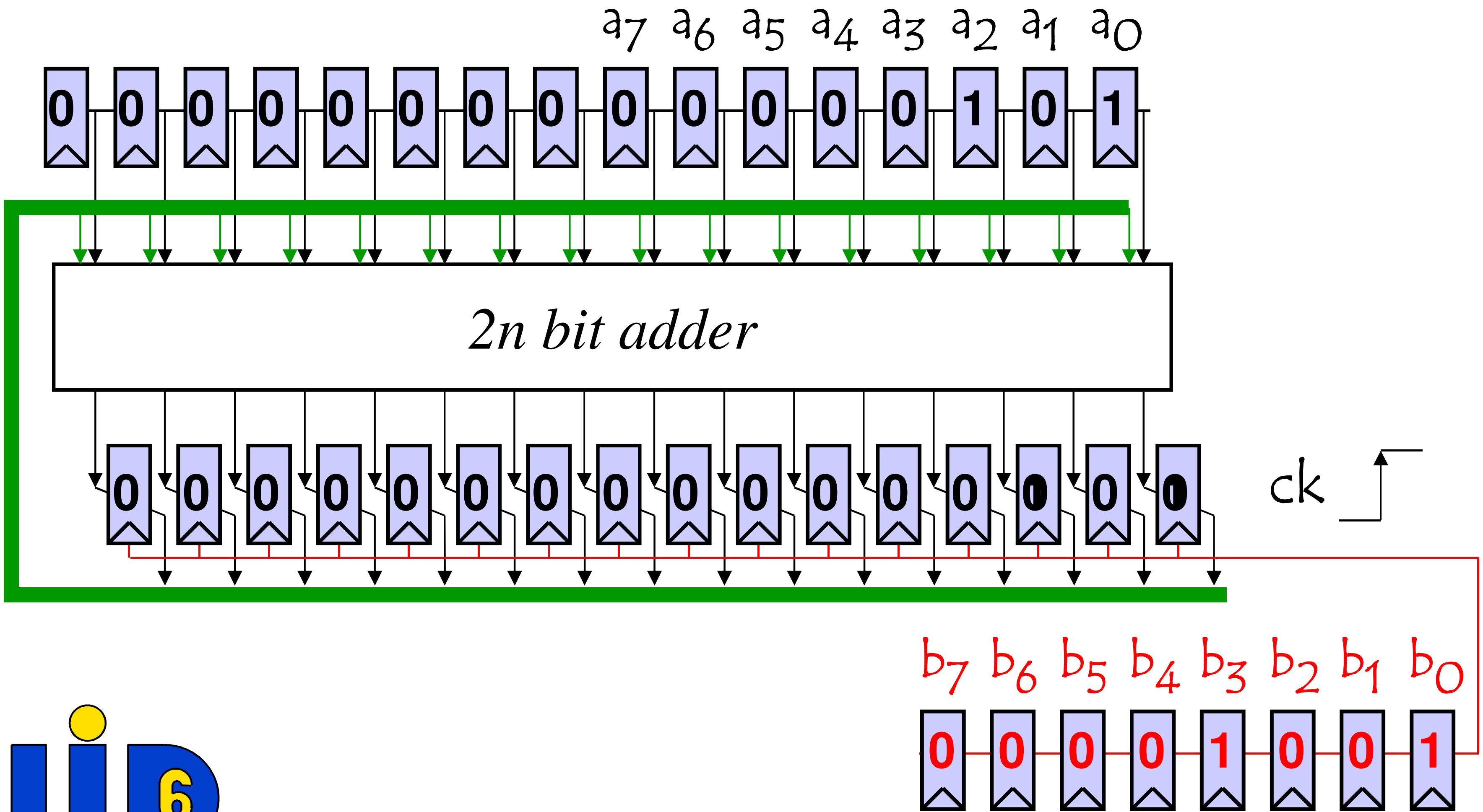
Multipliers

Implementation : sequential multiplier



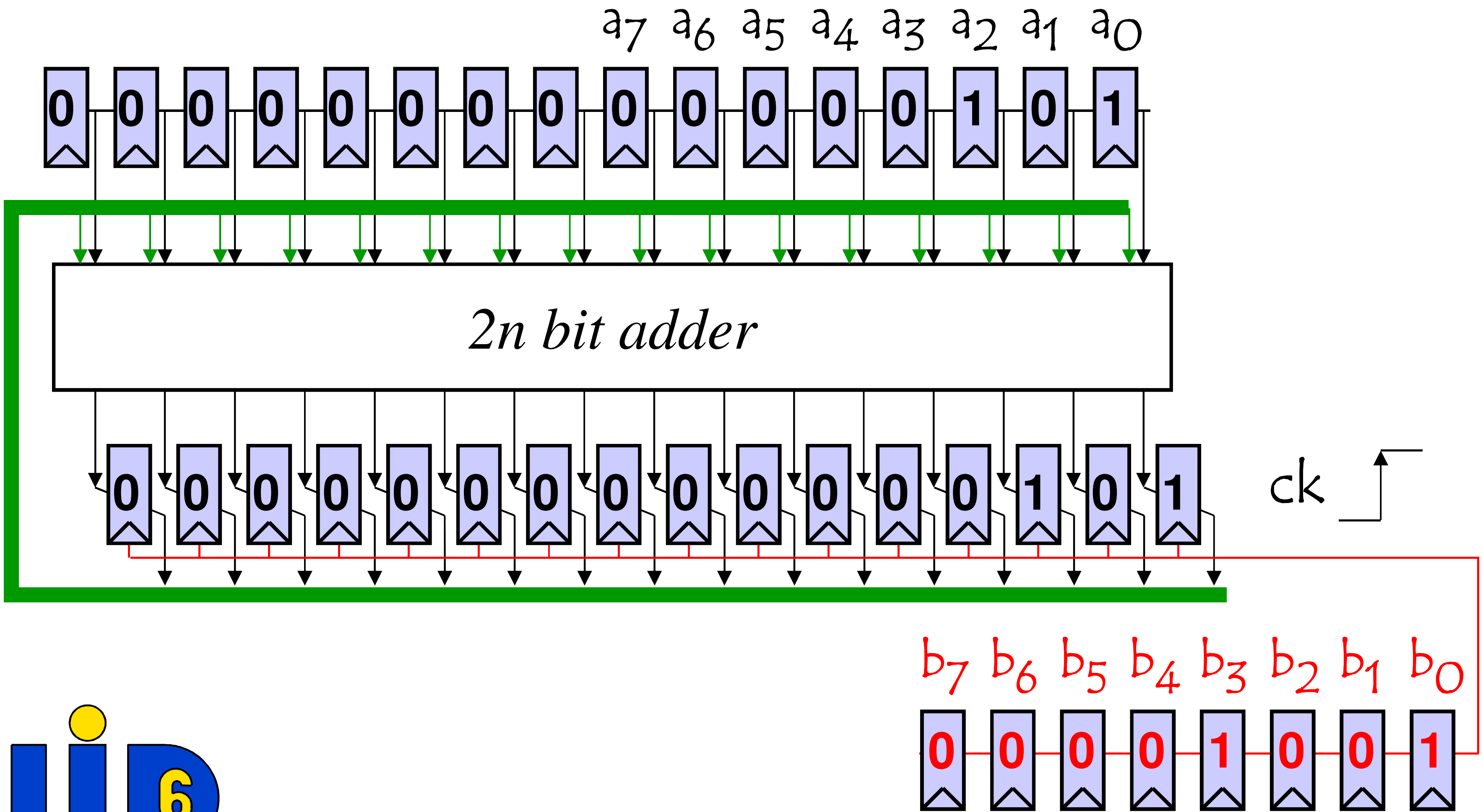
Multipliers

Implementation : sequential multiplier



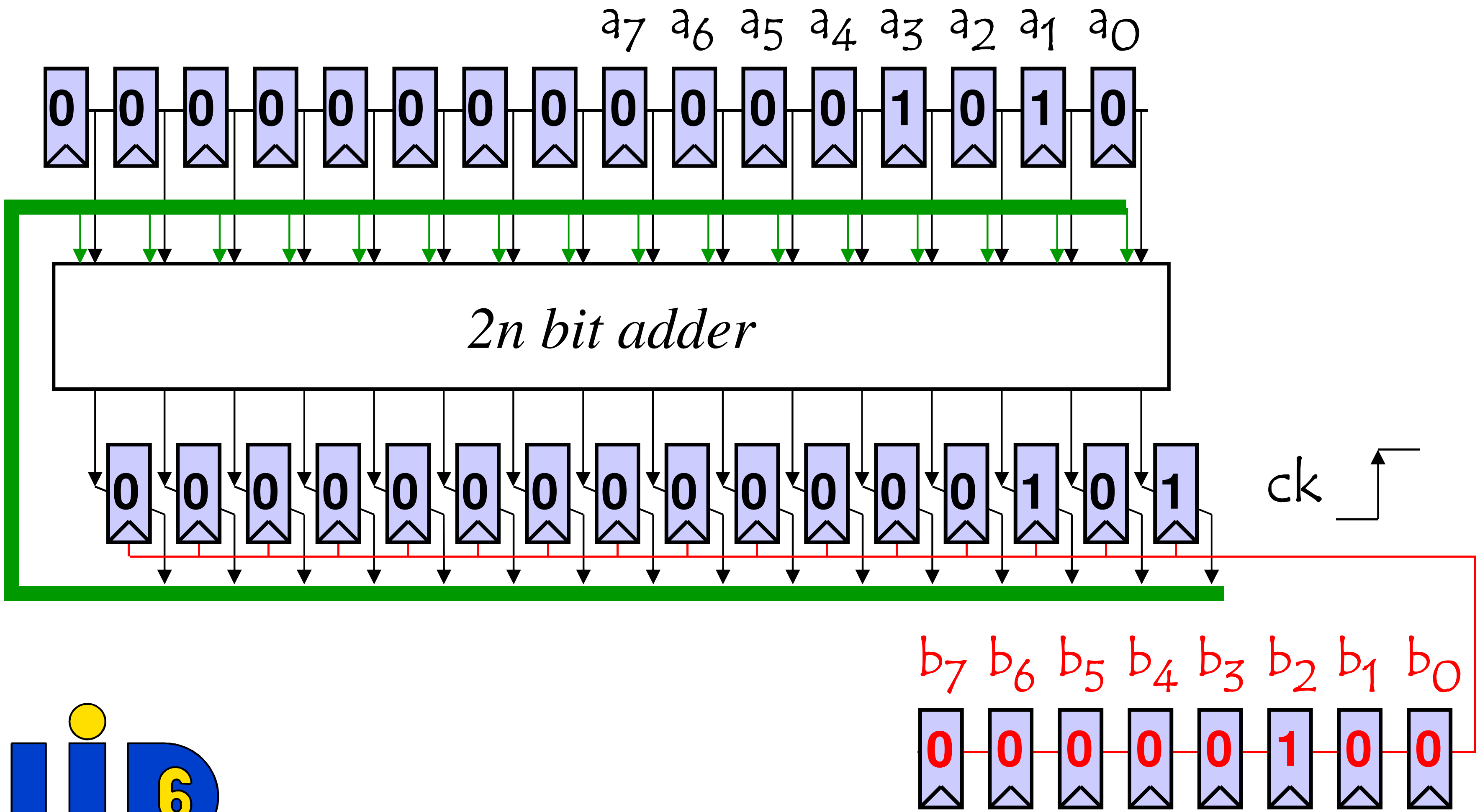
Multipliers

Implementation : sequential multiplier



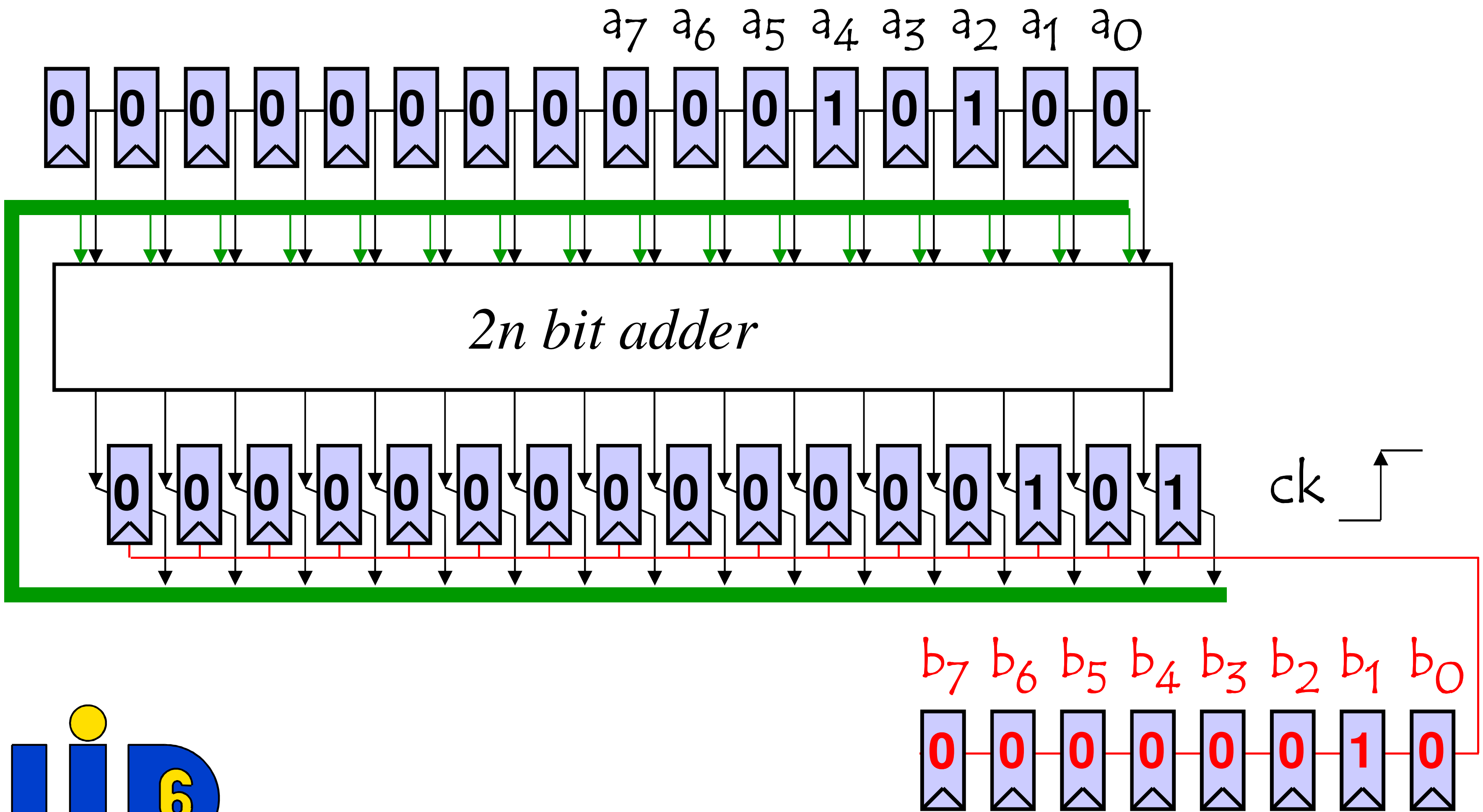
Multipliers

Implementation : sequential multiplier



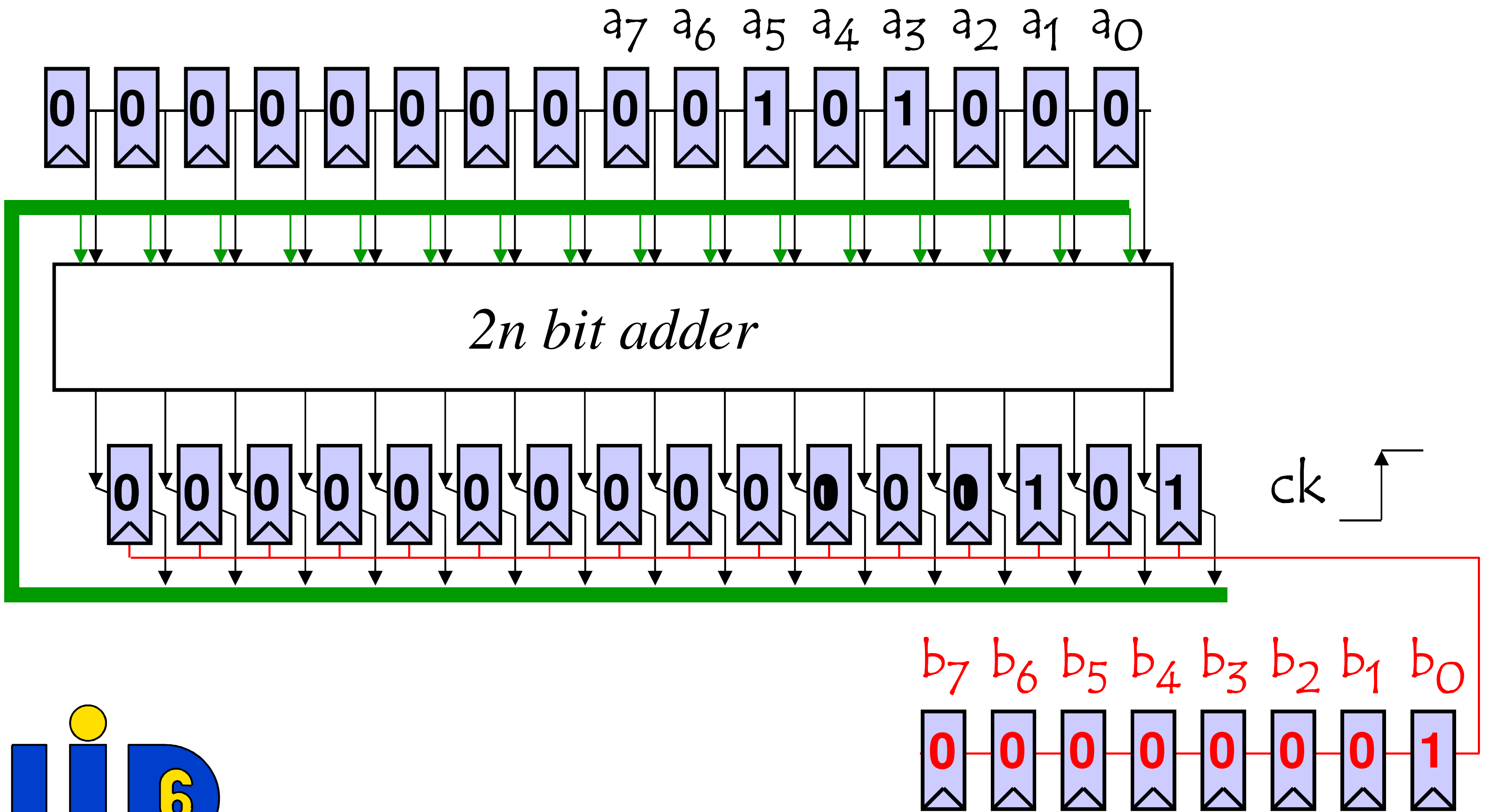
Multipliers

Implementation : sequential multiplier



Multipliers

Implementation : sequential multiplier



Multipliers

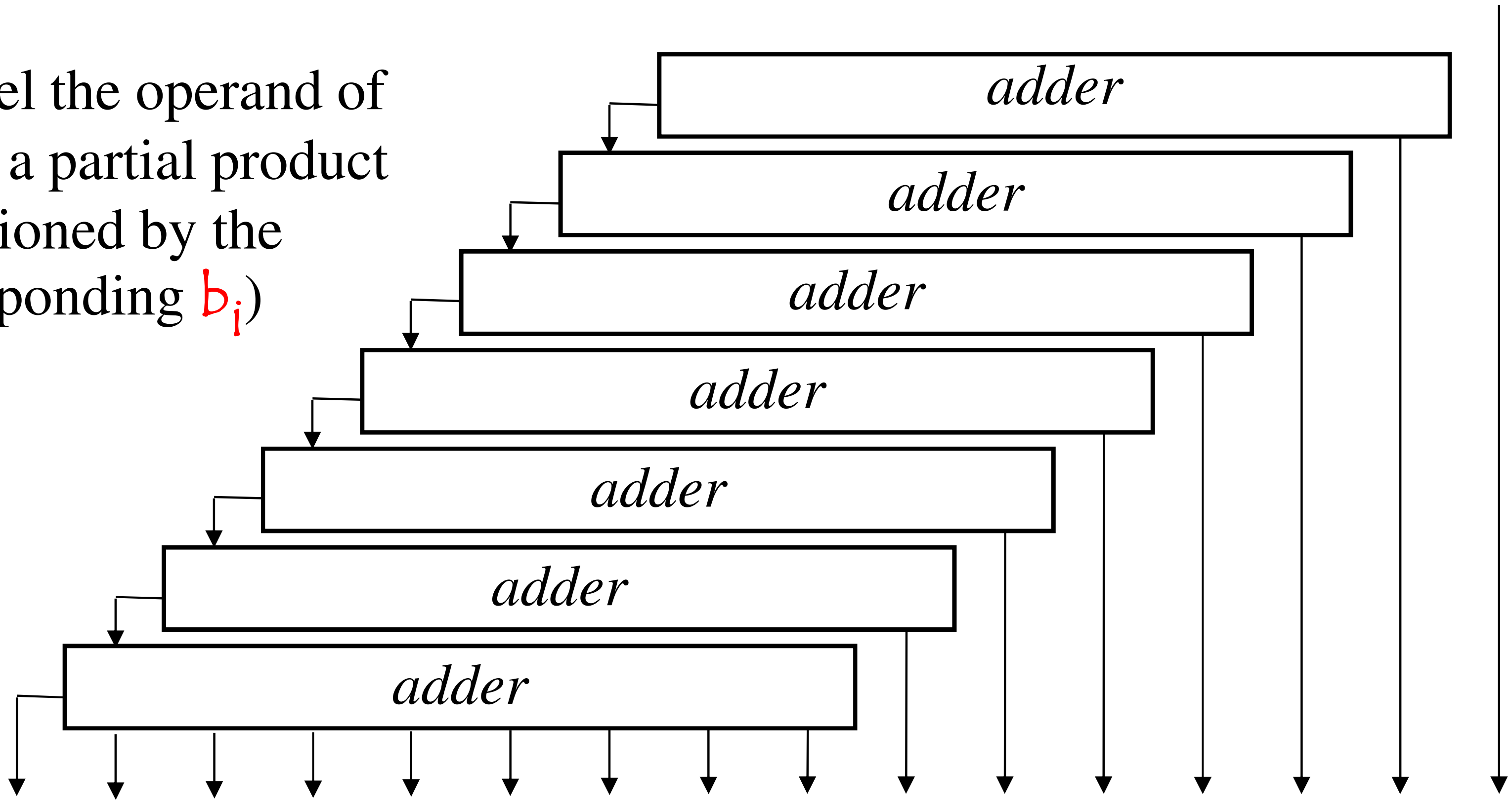
Implementation : parallel multiplier

b_0 →									a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
b_1 →								a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
b_2 →							a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0		
b_3 →						a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0			
b_4 →					a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0				
b_5 →				a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0					
b_6 →			a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0						
b_7 →		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0							
	s_{15}	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

Multipliers

Implementation : parallel multiplier

At each level the operand of the adder is a partial product (conditioned by the corresponding b_i)



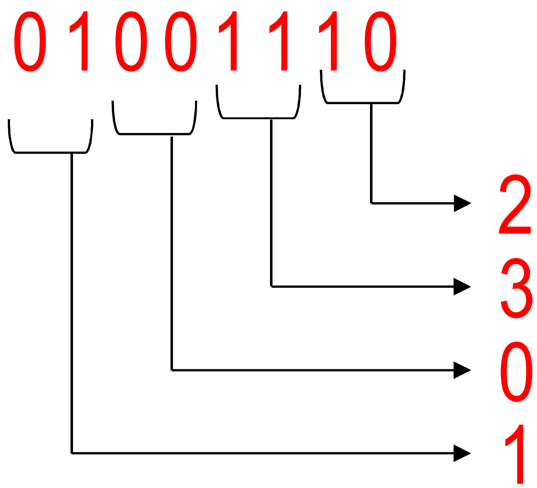
Multipliers

Implementation : parallel multiplier

Improvement : Reduce the number of partial products

$$x = \sum_{i=0}^{n-1} x_i \times 2^i \qquad x_i \in \{0, 1\}$$

$$x = \sum_{i=0}^{n/2-1} (x_{2i} + 2x_{2i+1}) \times 2^{2i}$$



= 78

half less terms ⇒
half less partial products

$$x = 2\times 2^0 + 3\times 2^2 + 0\times 2^4 + 1\times 2^6 = 78$$

Multipliers

Implementation : parallel multiplier
Improvement : Reduce the number of partial products

$$x = \sum_{i=0}^{n-1} x_i \times 2^i$$

$$\begin{aligned} x = {} & x_0 \times 2^0 + x_1 \times 2^1 + \\ & x_2 \times 2^2 + x_3 \times 2^3 + \\ & x_4 \times 2^4 + x_5 \times 2^5 + \\ & x_6 \times 2^6 + x_7 \times 2^7 \end{aligned}$$

$$x = \sum_{i=0}^{n/2-1} (x_{2i} + 2x_{2i+1}) \times 2^{2i}$$

$$\begin{aligned} x = {} & (x_0 + x_1 \times 2) \times 2^0 + \\ & (x_2 + x_3 \times 2) \times 2^2 + \\ & (x_4 + x_5 \times 2) \times 2^4 + \\ & (x_6 + x_7 \times 2) \times 2^6 \end{aligned}$$

Multipliers

Implementation : parallel multiplier

Improvement : Reduce the number of partial products

$$x = \sum_{i=0}^{n-1} x_i \times 2^i \qquad x_i \in \{0, 1\}$$

$$x = \sum_{i=0}^{n/2-1} (x_{2i} + 2x_{2i+1}) \times 2^{2i} \qquad \in \{0, 1, 2, 3\}$$

$$2 = 4 - 2$$

$$x = \sum_{i=0}^{n/2-1} (x_{2i} + 4x_{2i+1} - 2x_{2i+1}) \times 2^{2i}$$

Multipliers

Implementation : parallel multiplier

Improvement : Reduce the number of partial products

$$x = \sum_{i=0}^{n/2-1} (x_{2i} + 4x_{2i+1} - 2x_{2i+1}) \times 2^{2i}$$

$$\begin{aligned} x = {} & (x_0 + 4x_1 - 2x_1) + \\ & (x_2 + 4x_3 - 2x_3) \times 4 + \\ & (x_4 + 4x_5 - 2x_5) \times 16 + \cdots \end{aligned}$$

$$\begin{aligned} x = {} & (x_0 + 0 - 2x_1) + \\ & (x_2 + x_1 - 2x_3) \times 4 + \\ & (x_4 + x_3 - 2x_5) \times 16 + \cdots \end{aligned}$$

Multipliers

Implementation : parallel multiplier
Improvement : Reduce the number of partial products

$$x = \sum_{i=0}^{n-1} x_i \times 2^i \qquad x_i \in \{0, 1\}$$

$$x = \sum_{i=0}^{n/2-1} (x_{2i} + 4x_{2i+1} - 2x_{2i+1}) \times 2^{2i}$$

next weight

previous weight

$$x = \sum_{i=0}^{n/2} (x_{2i-1} + x_{2i} - 2x_{2i+1}) \times 2^{2i}$$

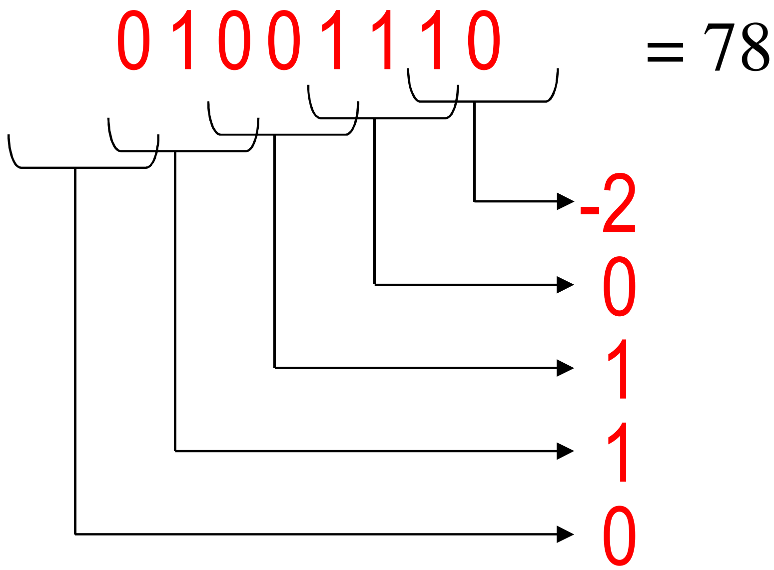
$$x = \sum_{i=0}^{n/2} x'_{2i} \times 2^{2i} \qquad x'_i \in \{-2, -1, 0, 1, 2\}$$

Multipliers

Implementation : parallel multiplier
Improvement : Reduce the number of partial products

$$x = \sum_{i=0}^{n/2} (x_{2i-1} + x_{2i} - 2x_{2i+1}) \times 2^{2i}$$

Booth encoding



$$x = -2\times 2^0 + 0\times 2^2 + 1\times 2^4 + 1\times 2^6 + 0\times 2^8 = 78$$

Multipliers

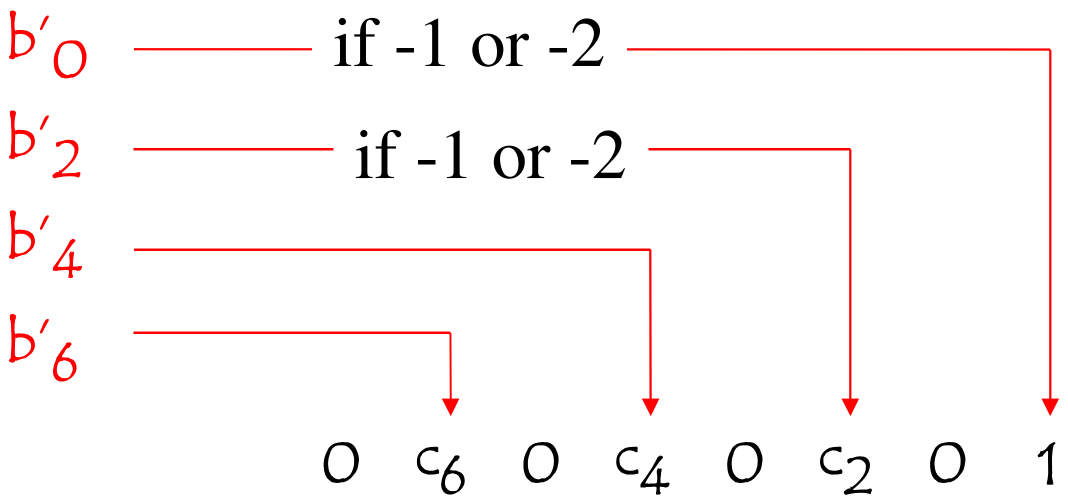
Implementation : parallel multiplier
Improvement : Reduce the number of partial products

recoded multiplicand bit (b'_i)										
		a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	multiplier
2	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0	partial product
1	0	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	partial product
0	0	0	0	0	0	0	0	0	0	partial product
-1	1	$\overline{a_7}$	$\overline{a_6}$	$\overline{a_5}$	$\overline{a_4}$	$\overline{a_3}$	$\overline{a_2}$	$\overline{a_1}$	$\overline{a_0}$	partial product
-2	$\overline{a_7}$	$\overline{a_6}$	$\overline{a_5}$	$\overline{a_4}$	$\overline{a_3}$	$\overline{a_2}$	$\overline{a_1}$	$\overline{a_0}$	1	partial product
	a'_8	a'_7	a'_6	a'_5	a'_4	a'_3	a'_2	a'_1	a'_0	

Multipliers

Implementation : parallel multiplier
Improvement : Reduce the number of partial products

An additional partial product is generated to take into account the input carry in case of subtraction



Multipliers

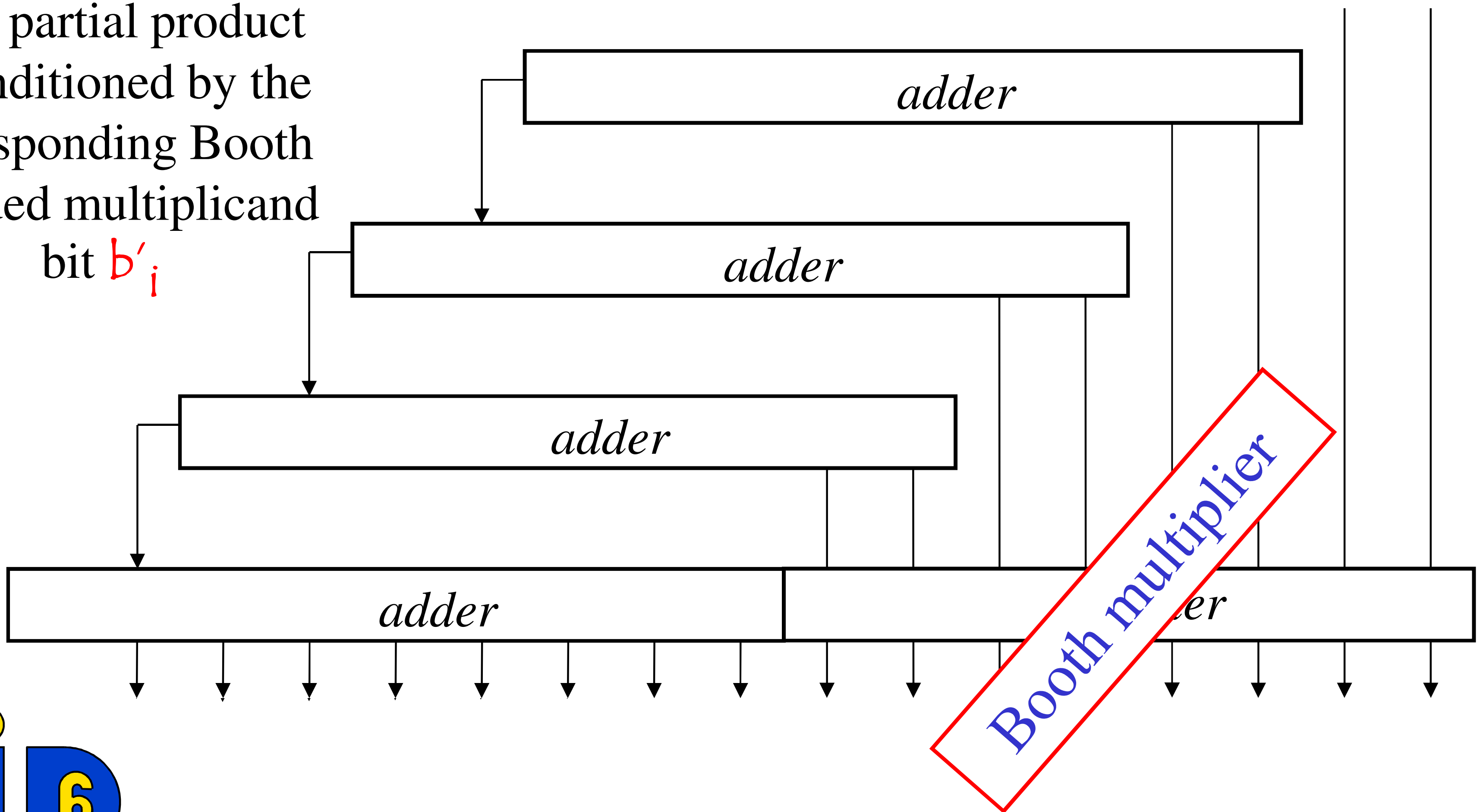
Implementation : parallel multiplier

b'_0 →									a'_8	a'_7	a'_6	a'_5	a'_4	a'_3	a'_2	a'_1	a'_0
b'_2 →							a'_8	a'_7	a'_6	a'_5	a'_4	a'_3	a'_2	a'_1	a'_0		
b'_4 →					a'_8	a'_7	a'_6	a'_5	a'_4	a'_3	a'_2	a'_1	a'_0				
b'_6 →			a'_8	a'_7	a'_6	a'_5	a'_4	a'_3	a'_2	a'_1	a'_0						
b'_8	a'_8	a'_7	a'_6	a'_5	a'_4	a'_3	a'_2	a'_1	a'_0								
										c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0
		s_{15}	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0

Multipliers

Implementation : parallel multiplier

Each partial product is conditioned by the corresponding Booth encoded multiplicand bit b'_i



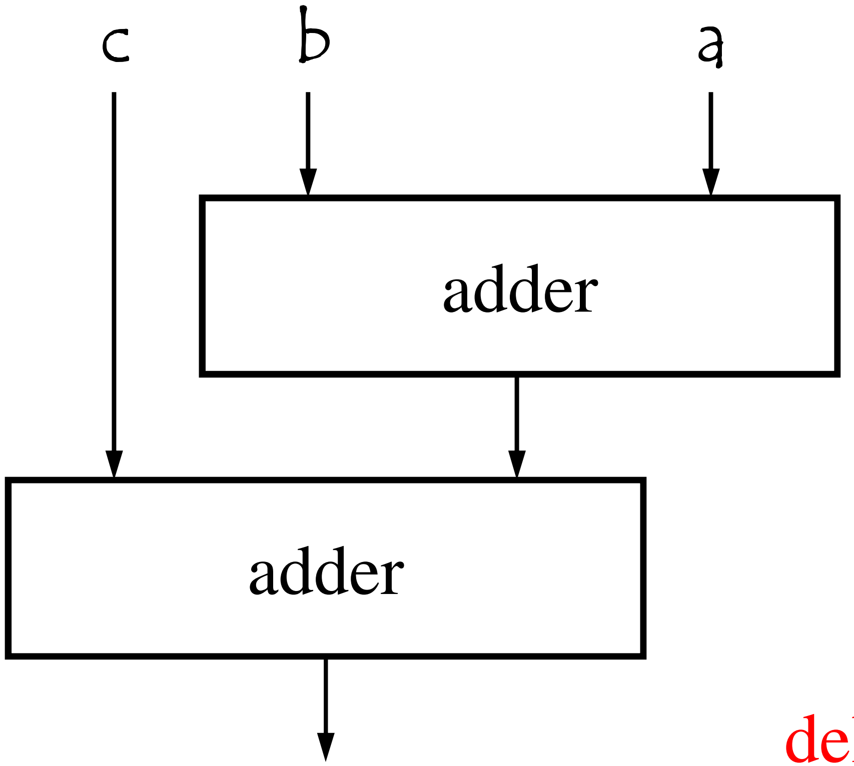
Multipliers

Implementation : fast parallel multiplier

Basically for a n×n multiplication,
we have to add n partial products
(2n-bit numbers)

Multipliers

Adding three natural numbers



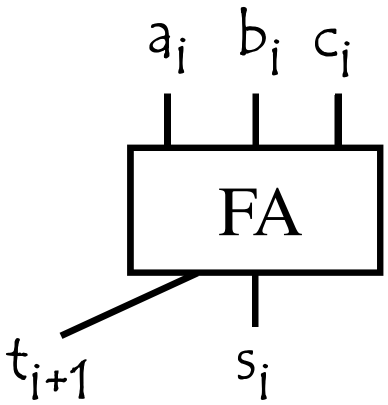
delay

Multipliers

Adding three natural numbers

$s_i = a_i \oplus b_i \oplus c_i$ $\qquad$ $c_{i+1} = a_i.b_i + a_i.c_i + b_i.c_i$

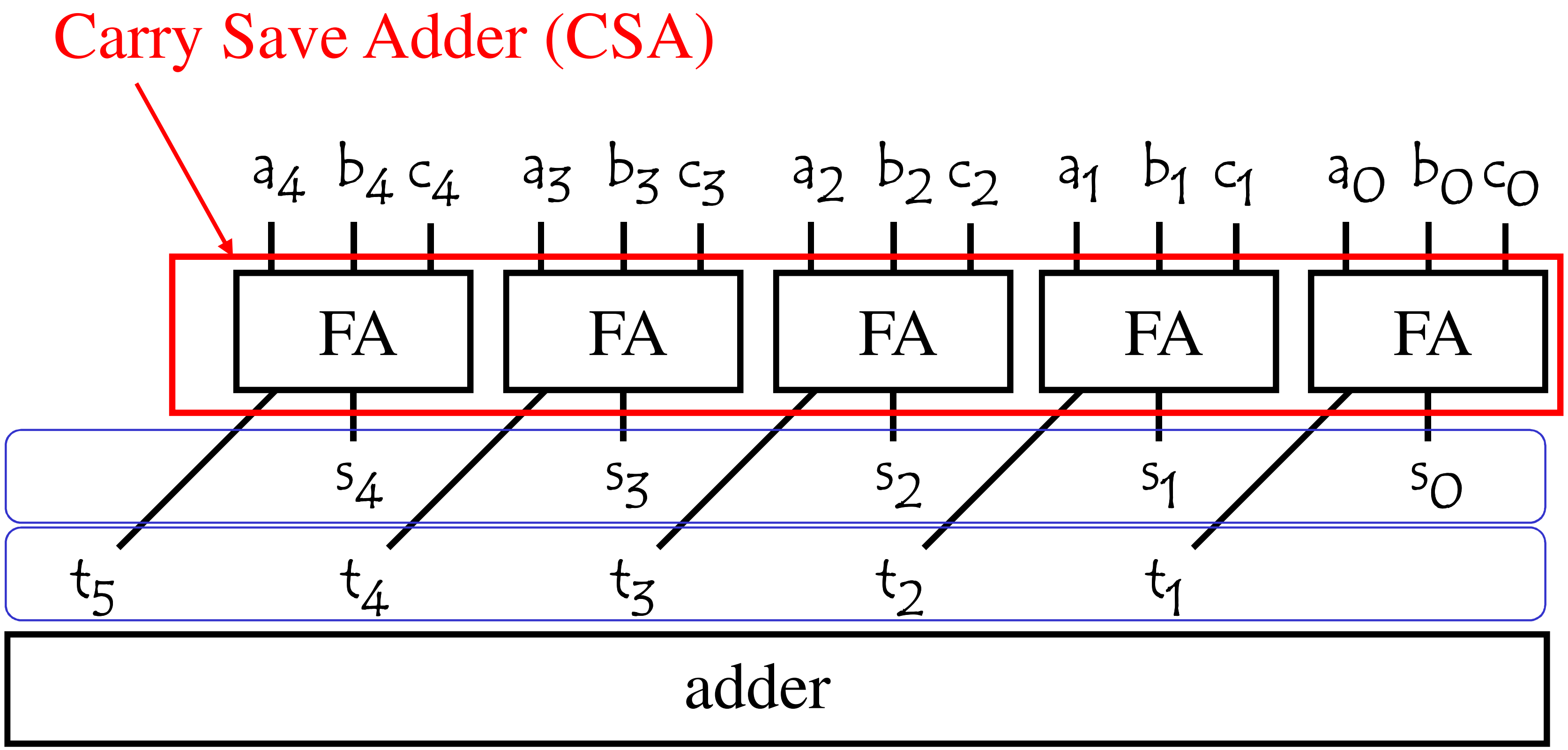
the expressions are symmetrical in regard of a, b and c



A full adder creates 2 numbers from 3

Multipliers

Adding three natural numbers



Multipliers

Implementation : fast parallel multiplier

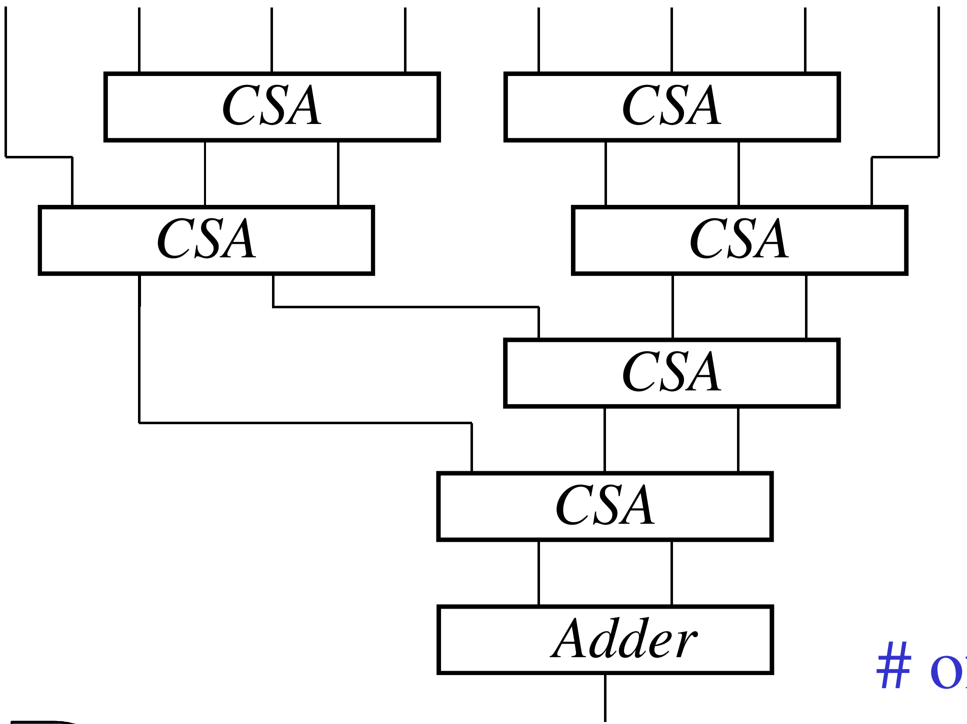
Basically for a $n \times n$ multiplication,
we have to add n partial products
(2n-bit numbers)

➡ Use CSA (Carry Save Adder)
to reduce 3 partial products
into 2

Multipliers

Implementation : fast parallel multiplier

8 partial products



Wallace multiplier

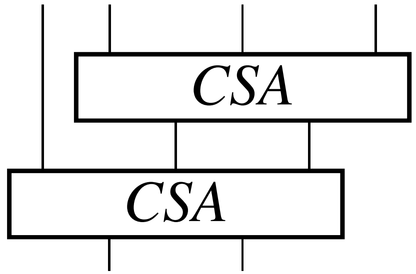
# of CSA layers ≈ $\log_{3/2}(n/2)$

Multipliers

Implementation : fast parallel multiplier

32×32 bits multiplier $32 \rightarrow 22 \rightarrow 15 \rightarrow 10 \rightarrow 7 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2$

Using $4 \rightarrow 2$ reduction leads to a more regular hardware implementation



32×32 bits multiplier $32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2$

# of CSA layers ≈ 2 log (n/2)

Multipliers

Implementation : fast parallel multiplier

32×32 bits multiplier $32 \rightarrow 22 \rightarrow 15 \rightarrow 10 \rightarrow 7 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2$

$42 \rightarrow 28 \rightarrow 19 \rightarrow 13 \rightarrow 9 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 2$

The margin on the number of ‘partial products’ can be used to extended the functionality of the multiplier

Multipliers

Two relative numbers a and b coded on n bits

$$p = a \times b$$

How it works if b < 0 ?

$$p = a \times b = (-a) \times (-b) = (\overline{a} + 1) \times (\overline{b} + 1)$$

$$= \overline{a} \times \overline{b} + \overline{a} + \overline{b} + 1$$

Multipliers

Multiply and add

$$p = a \times b + c$$

if b < 0

$$p = a \times b + c = (-a) \times (-b) + c \quad = (\overline{a} + 1) \times (\overline{b} + 1) + c$$

$$= \overline{a} \times \overline{b} + \overline{a} + \overline{b} + 1 + c$$

Multipliers

Multiply and subtract

$$p = c - a \times b$$

if b > 0

$$p = c - a \times b = c + a \times (-b) \quad = c + a \times (\overline{b} + 1)$$

$$= c + a \times \overline{b} + a$$