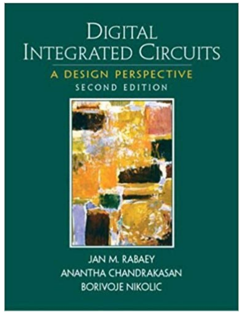
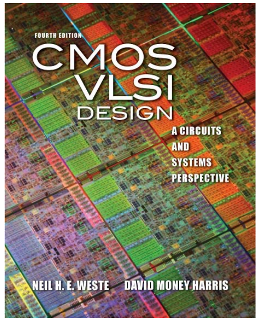
Microeletrônica

Aula #10 → Multiplicação e Divisão: conceitos básicos
Operações de rotação e deslocamento

- Professor: Fernando Gehm Moraes
- Livro texto:
  - Digital Integrated Circuits a Design Perspective - Rabaey
  - C MOS VLSI Design - Weste





Revisão das lâminas: 16/maio/2025

Conceitos básicos para multiplicação

```
multiplicand                 1101   (13)
multiplier                 * 1011   (11)
                           ------
                             1101
Partial products            1101
                           0000
                          1101
                         --------
                         10001111    (143)
```

- product of 2 n-bit numbers is an 2n-bit number
  - **sum of n n-bit partial products**
- unsigned

Multiplicação

				A3	A2	A1	A0
				B3	B2	B1	B0
				A3 B0	A2 B0	A1 B0	A0 B0
			A3 B1	A2 B1	A1 B1	A0 B1	
		A3 B2	A2 B2	A1 B2	A0 B2		
	A3 B3	A2 B3	A1 B3	A0 B3			
S7	S6	S5	S4	S3	S2	S1	S0

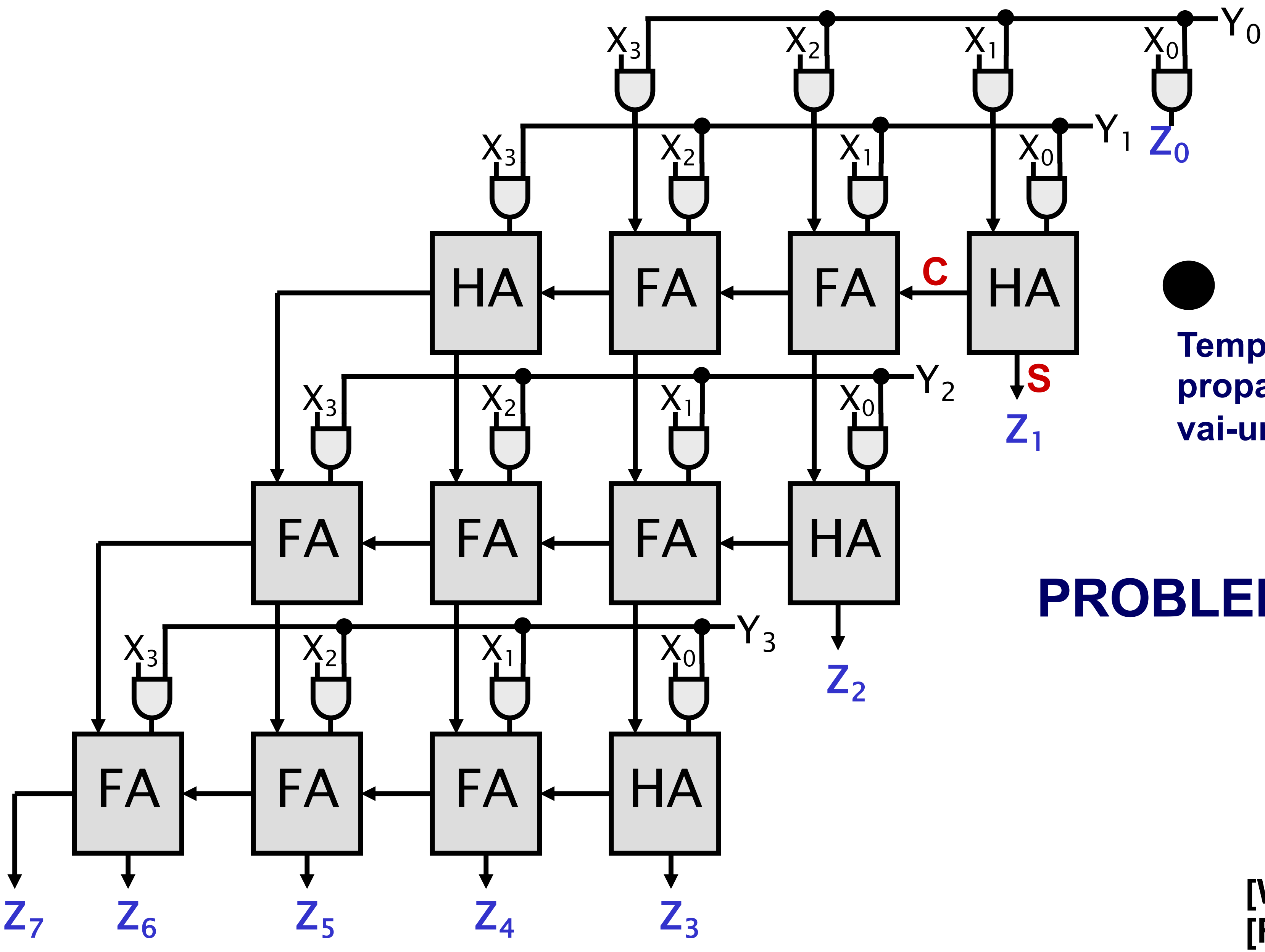
um carry propagado

dois carrys propagados

Quanto maior o número de produtos parciais a somar maior o número de bits de vai-um gerados

PROBLEMA

Multiplicador "array"



Tempo de propagação do vai-um

PROBLEMA?

[WE] p547
[Rab] p.409

Carry-save Addition

- Speeding up multiplication is a matter of speeding up the summing of the partial products.
- "Carry-save" addition can help.
- Carry-save addition passes (saves) the carries to the output, rather than propagating them.

- Example: sum three numbers,

  3_{10} = 0011, 2_{10} = 0010, 3_{10} = 0011

$$
\left.\begin{array}{rrl}
 & 3_{10} & 0011 \\
+ & 2_{10} & \underline{0010} \\
 & c & 0100 = 4_{10} \\
 & s & 0001 = 1_{10}
\end{array}\right\}\text{carry-save add}
$$

$$
\begin{array}{rl}
3_{10} & \underline{0011}
\end{array}
$$

$$
\text{carry-propagate add}\left\{\begin{array}{rl}
c & 0010 = 2_{10} \\
s & \underline{0110} = 6_{10} \\
 & 1000 = 8_{10}
\end{array}\right.
$$

- In general, carry-save addition takes in 3 numbers and produces 2.
- Whereas, carry-propagate takes 2 and produces 1.
- With this technique, we can avoid carry propagation until final addition

Carry-save Addition

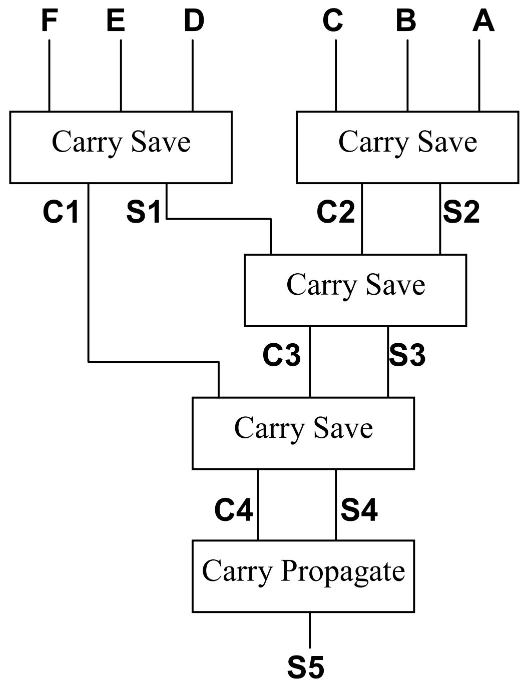
- Fazer a soma 7 + 3 + 4

	0 1 1 1	7	carry-save add
	0 0 1 1	3	
carry-save add	0	carry	
		soma	
	0 1 0 0	4	
	0	carry	
		soma	
		resultado	

O somador carry save, utilizado nos multiplicadores, serve para realizar a soma quando há várias parcelas a serem adicionadas.

Dados os seguintes valores:

A = 001101 (13_{10})
B = 000101 (5_{10})
C = 001011 (11_{10})
D = 001010 (10_{10})
E = 000111 (7_{10})
F = 001100 (12_{10})



Mostre a obtenção e os valores de (C1/S1), (C2/S2), (C3/S3), (C4/S4) e a soma final (S5).

D	**0**	**0**	**1**	**0**	**1**	**0**
E	**0**	**0**	**0**	**1**	**1**	**1**
F	**0**	**0**	**1**	**1**	**0**	**0**
S1						
C1						

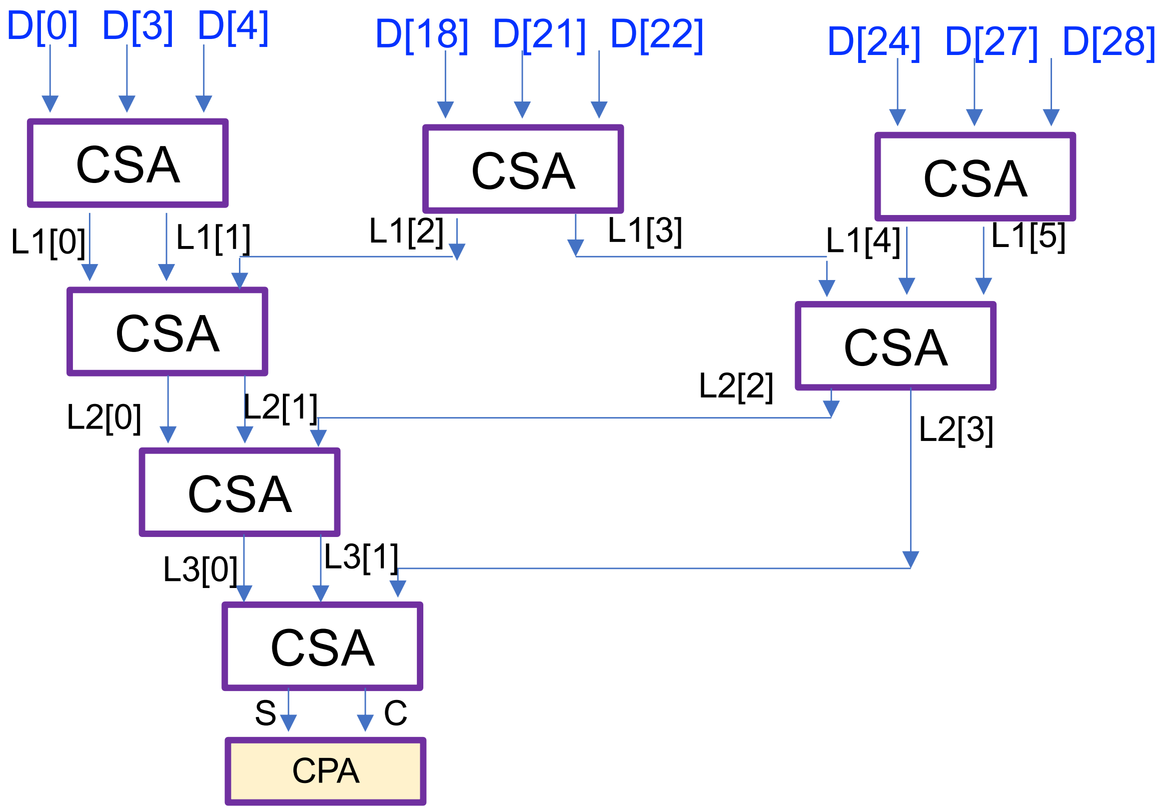
A	**0**	**0**	**1**	**1**	**0**	**1**
B	**0**	**0**	**0**	**1**	**0**	**1**
C	**0**	**0**	**1**	**0**	**1**	**1**
S2						
C2						
S1						
S3						
C3						
C1						
S4						
C4						
Soma						

Um projetista deve realizar a multiplicação de uma matriz de coeficientes [36x9] por um vetor de 36 posições. Analise as estruturas de dados, e aponte uma solução eficiente para **a multiplicação da primeira linha**.

$$
\begin{bmatrix}
1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&1&1&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1&0&0&0&0&0&0\\
0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&1&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0&0&1&0&0&1&1&0\\
0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1\\
0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&1&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&1&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&0&0&0&0&1&0&1&1&0&0&1&0&1&1
\end{bmatrix}
\begin{bmatrix}
D_0\\D_1\\D_2\\D_3\\D_4\\D_5\\D_6\\D_7\\D_8\\D_9\\D_{10}\\D_{11}\\D_{12}\\D_{13}\\D_{14}\\D_{15}\\D_{16}\\D_{17}\\D_{18}\\D_{19}\\D_{20}\\D_{21}\\D_{22}\\D_{23}\\D_{24}\\D_{25}\\D_{26}\\D_{27}\\D_{28}\\D_{29}\\D_{30}\\D_{31}\\D_{32}\\D_{33}\\D_{34}\\D_{35}
\end{bmatrix}
$$

Solução:

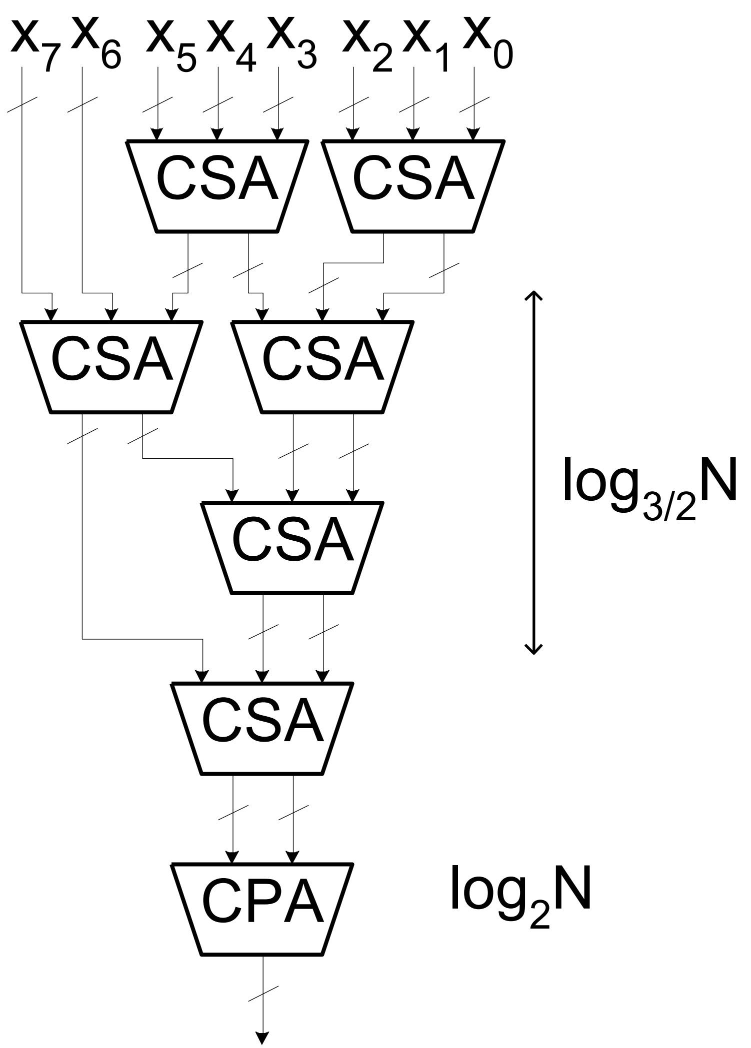
CSA_9 sp0 (D[D0], D[3], D[4], D[18], D[21], D[22], D[24], D[27], D[28], S[0]);



Carry-save Addition

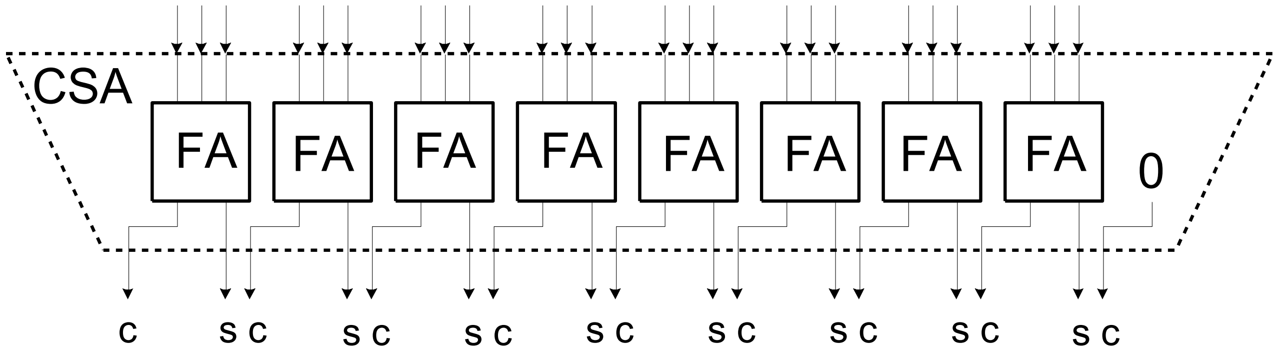
CSA is associative and communitive. For example:

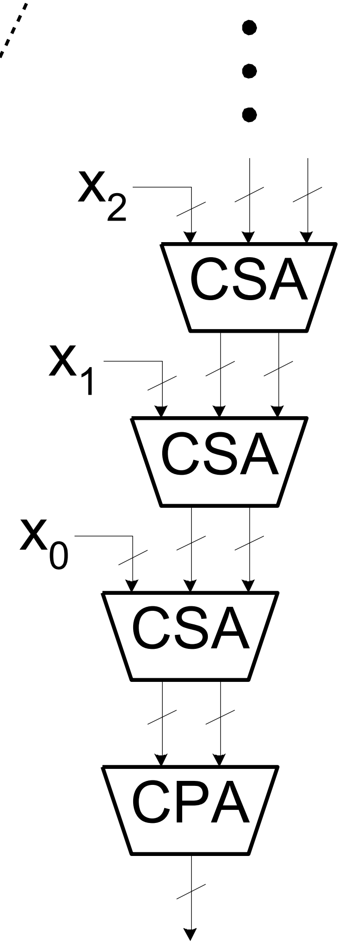
$$(((X_0 + X_1) + X_2) + X_3) = ((X_0 + X_1) + (X_2 + X_3))$$



- A balanced tree can be used to reduce the logic delay.
- This structure is the basis of the ***Wallace Tree Multiplier***.
- Partial products are summed with the CSA tree. Fast CPA is used for final sum.
- Multiplier delay α $\log_{3/2}N + \log_2 N$

Carry-save Circuits



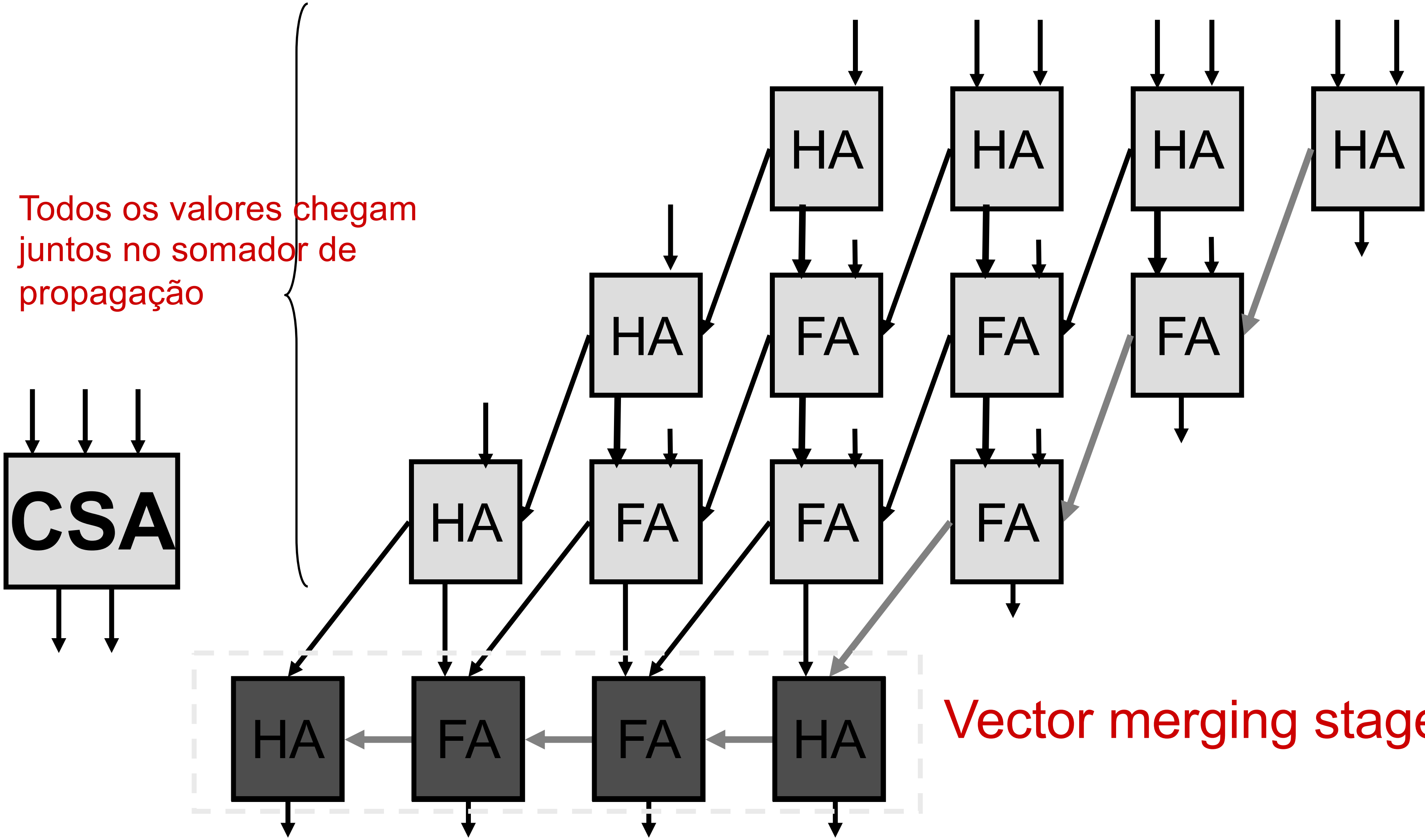


- When adding sets of numbers, carry-save can be used on all but the final sum.
- Standard adder (carry propagate) is used for final sum.

Carry-Save Multiplication

- Postpone the “carry propagation” operation to the last stage

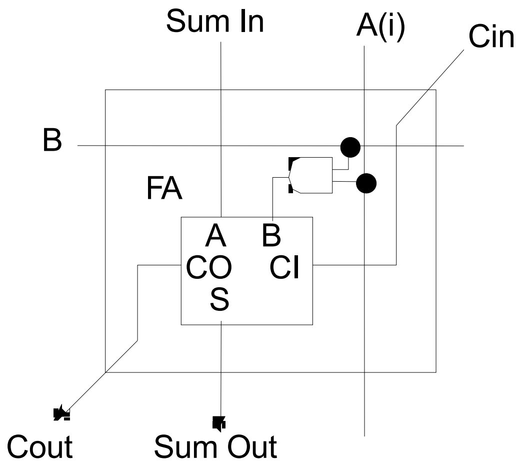
Todos os valores chegam juntos no somador de propagação



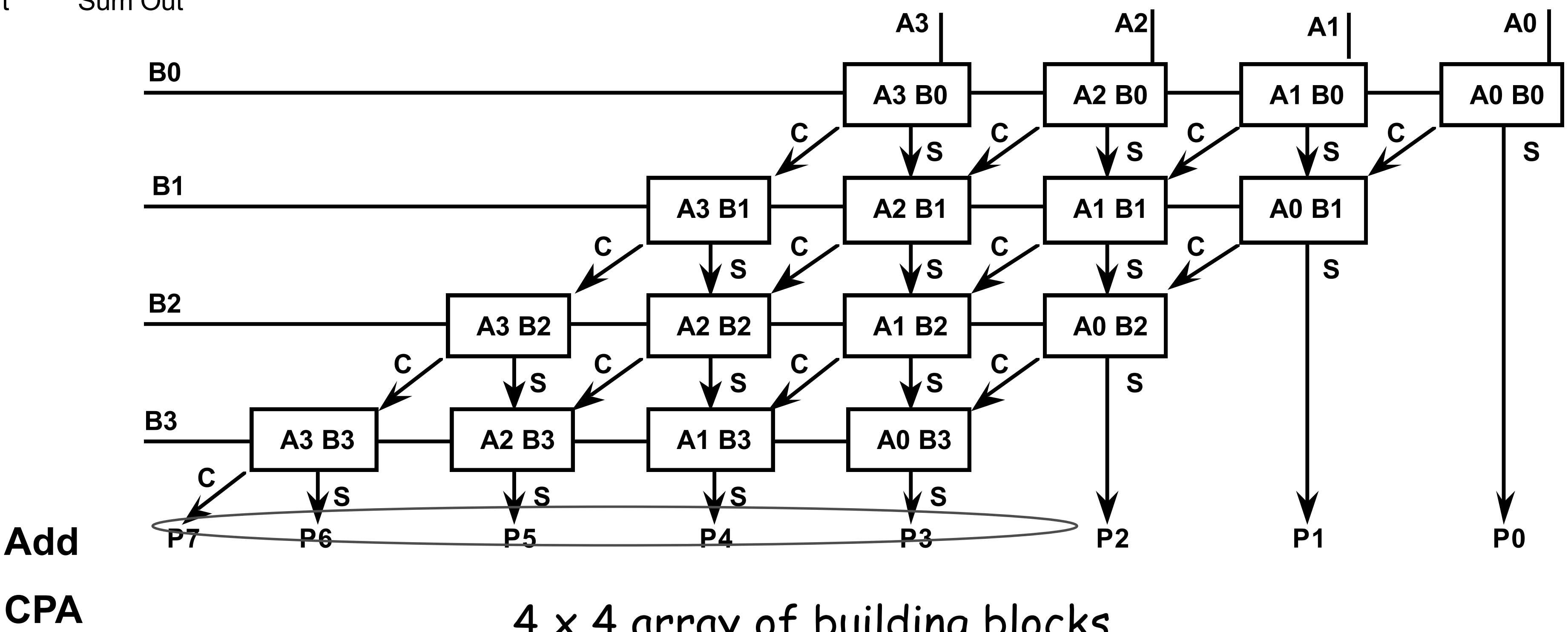
Vector merging stage

[Rabaey]

Implementação do multiplicador



Building block: full adder + and

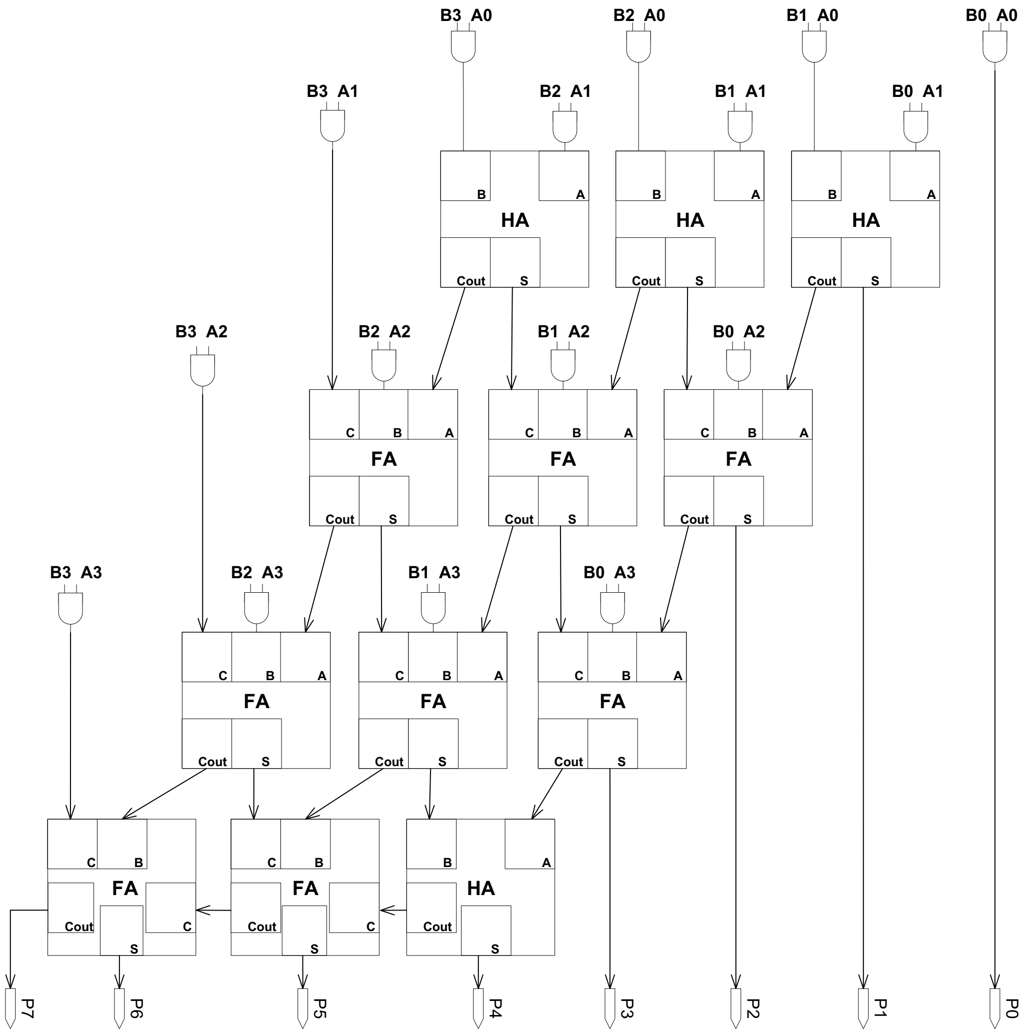


4 x 4 array of building blocks

Multiplicação. Apresenta-se abaixo o diagrama lógico de um multiplicador de 4 bits.

a) Explique como o carry é propagado no interior do circuito. Qual a vantagem deste esquema de propagação de carry?

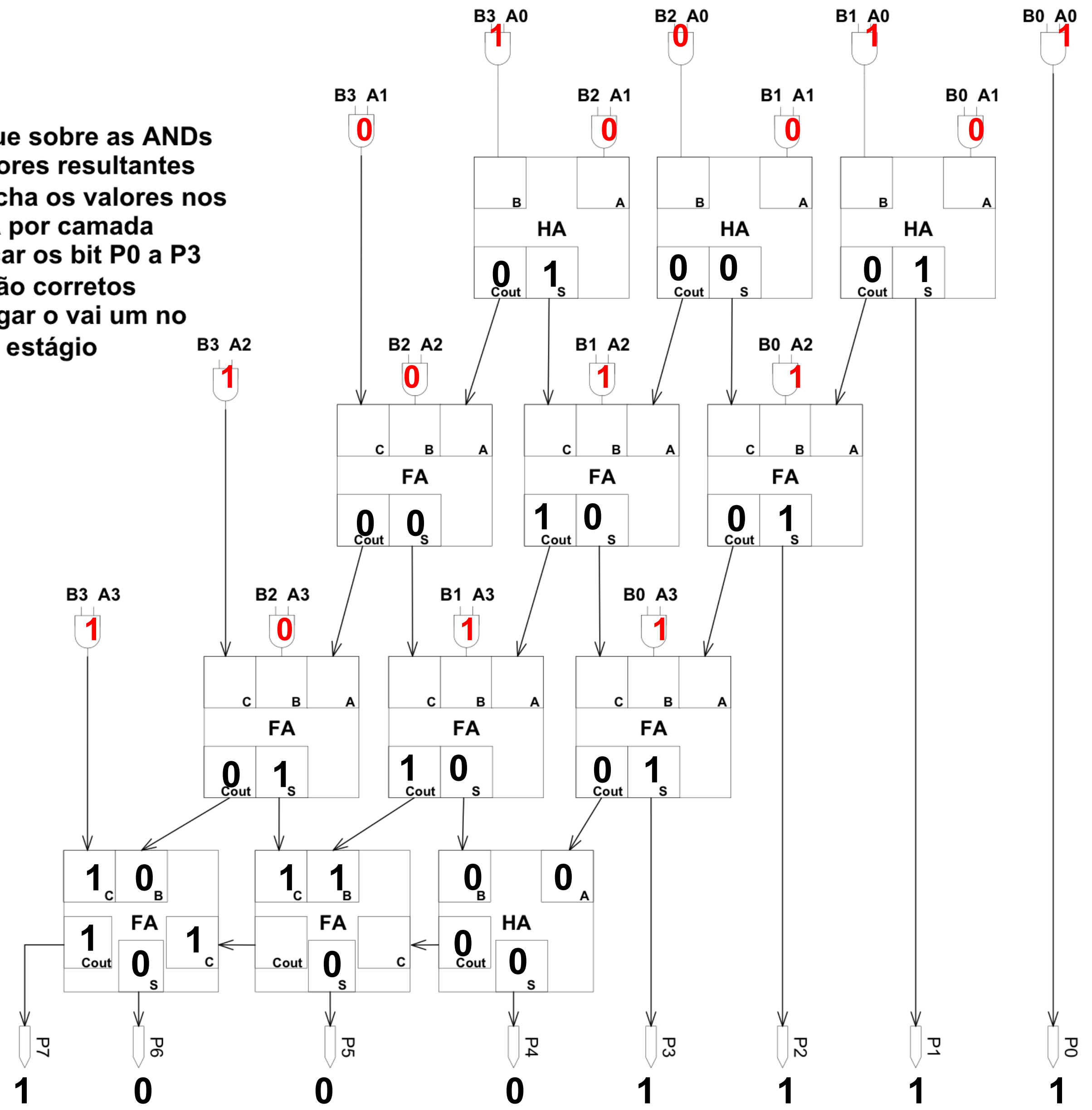
b) Considere: A=1011 e B=1101. Desenhe sobre o circuito os valores booleanos correspondentes, verificando se o valor obtido pela multiplicação é o correto.



Multiplicador com carry-save

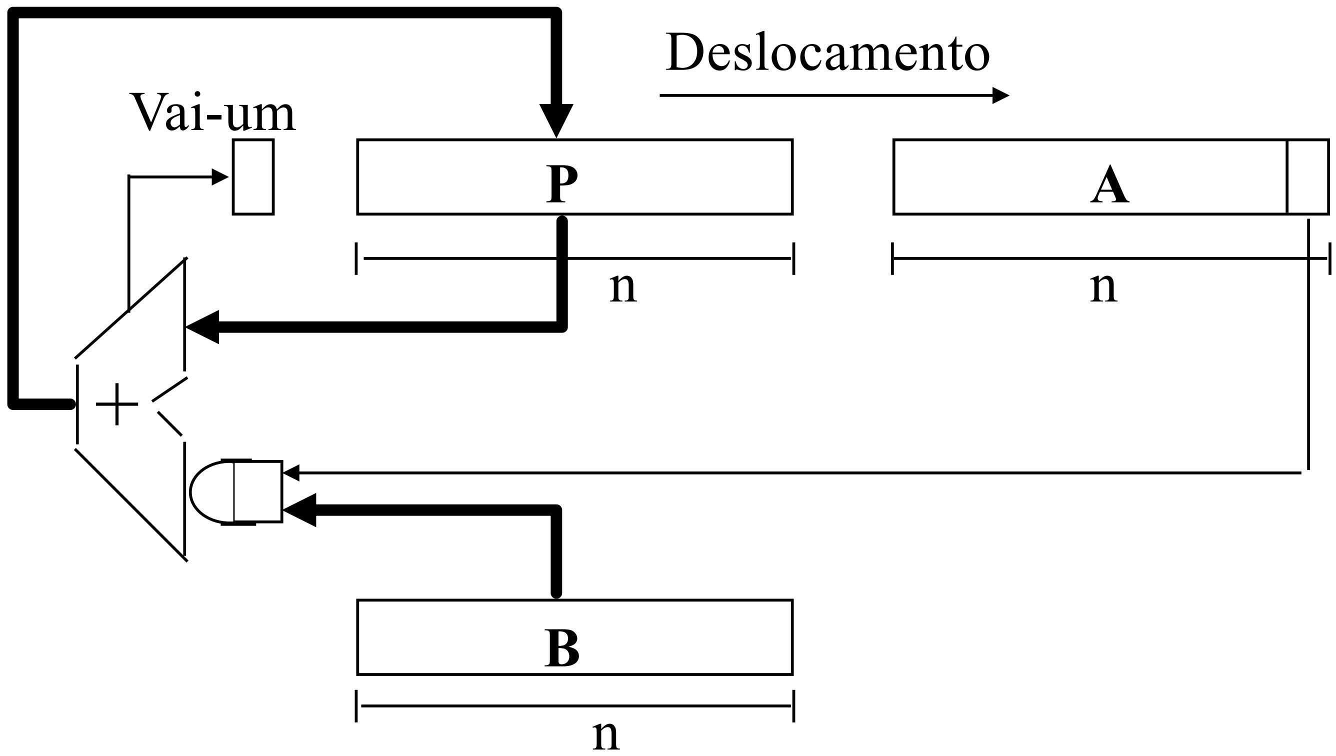
A=1101
B=1011

DICA:
1- coloque sobre as ANDs os valores resultantes
2- Preencha os valores nos HA/FA por camada
3- Verificar os bit P0 a P3 se estão corretos
4- Propagar o vai um no último estágio



Multiplicação serial

- Solução natural para a*b: somas sucessivas *n* passos



- Inicialmente, P=0, A=a, B=b. Cada passo, duas partes:
  - **soma carregada em P;**
  - **P & A deslocado um bit para a direita.**

Multiplicação A*B

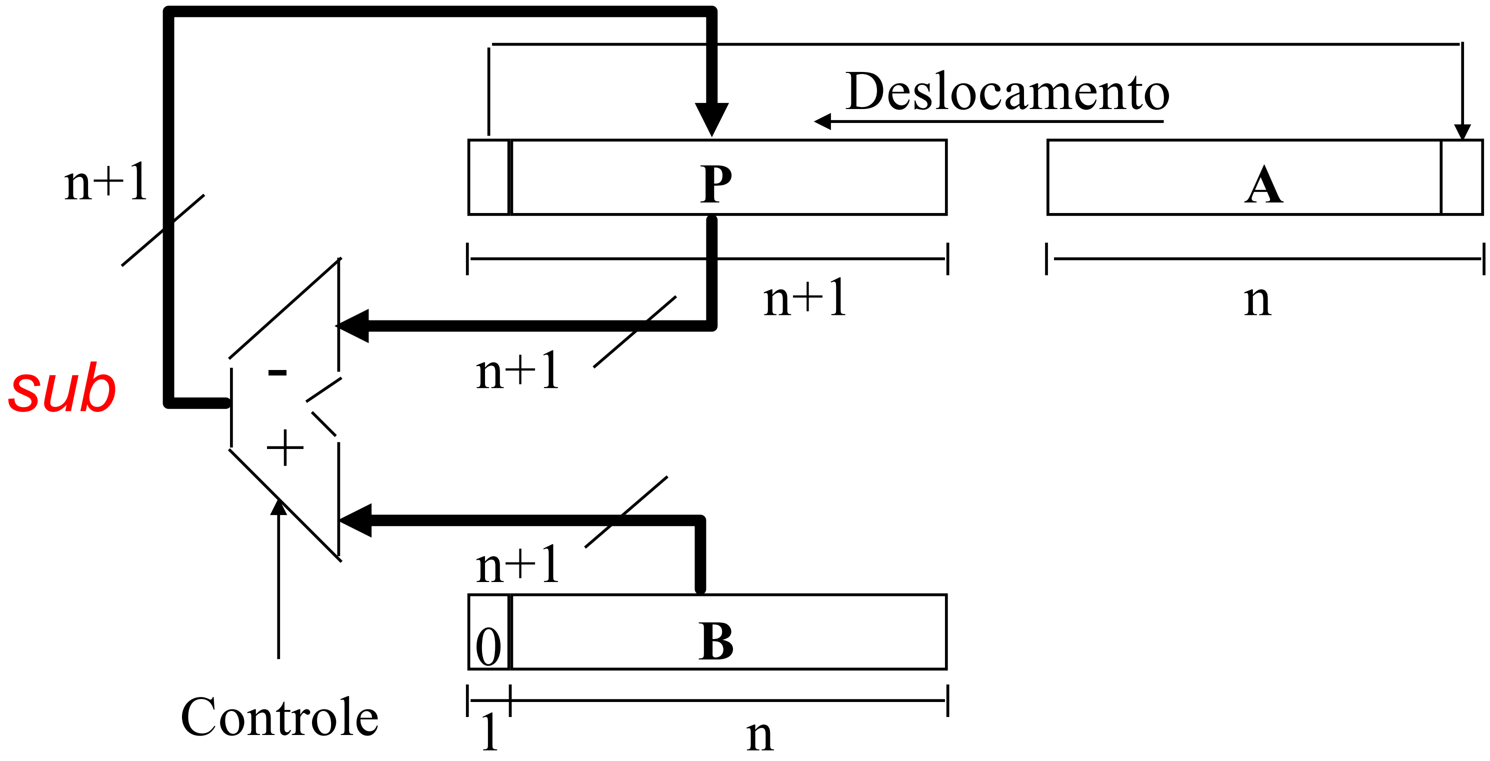
A = 11011 (27)
B = 00101 (5)

135 → 100 00111

		P					A				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	1	0	1	1	1	0	1	1
	0	0	0	0	1	0	1	1	1	0	1
2	0	0	0	1	1	1	1	1	1	0	1
	0	0	0	0	1	1	1	1	1	1	0
3	0	0	0	0	1	1	1	1	1	1	0
	0	0	0	0	0	1	1	1	1	1	1
4	0	0	0	1	1	0	1	1	1	1	1
	0	0	0	0	1	1	0	1	1	1	1
5	0	0	1	0	0	0	0	1	1	1	1
	0	0	0	1	0	0	0	0	1	1	1

Divisão serial

- Solução para a/b: subtrações sucessivas, *n* passos



- Algoritmo:
  1) desloca P&A p/ esq 1 bit; sub ← P-B;
  2) if (sub<0), A_0=0 else { A_0 =1; P ←sub}

Divisão A/B

A = 11011 (27)
B = 00101 (5)

1) desloca P&A p/ esq 1 bit; sub ← P-B;
2) if (sub<0), A0=0 else { A0 =1; P ←sub}

	P (conterá o resto)						A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2											
3											
4											
5											

Divisão A/B

A = 11011 (27)
B = 00101 (5)

1) desloca P&A p/ esq 1 bit; sub ← P-B;
2) if (sub<0), A0=0 else { A0 =1; P ←sub}

	P (conterá o resto)						A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3											
4											
5											

Divisão A/B

A = 11011 (27)
B = 00101 (5)

1) desloca P&A p/ esq 1 bit; sub ← P-B;
2) if (sub<0), A0=0 else { A0 =1; P ←sub}

	P (conterá o resto)						A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3	0	0	0	1	1	0	1	1	0	0	0
	0	0	0	0	0	1	1	1	0	0	1
4											
5											

00110 - 00101 = 001

Divisão A/B

A = 11011 (27)
B = 00101 (5)

1) desloca P&A p/ esq 1 bit; sub ← P-B;
2) if (sub<0), A0=0 else { A0 =1; P ←sub}

	P (conterá o resto)						A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3	0	0	0	1	1	0	1	1	0	0	0
	0	0	0	0	0	1	1	1	0	0	1
4	0	0	0	0	1	1	1	0	0	1	0
	0	0	0	0	1	1	1	0	0	1	0
5											

Divisão A/B

A = 11011 (27)
B = 00101 (5)

1) desloca P&A p/ esq 1 bit; sub ← P-B;
2) if (sub<0), A0=0 else { A0 =1; P ←sub}

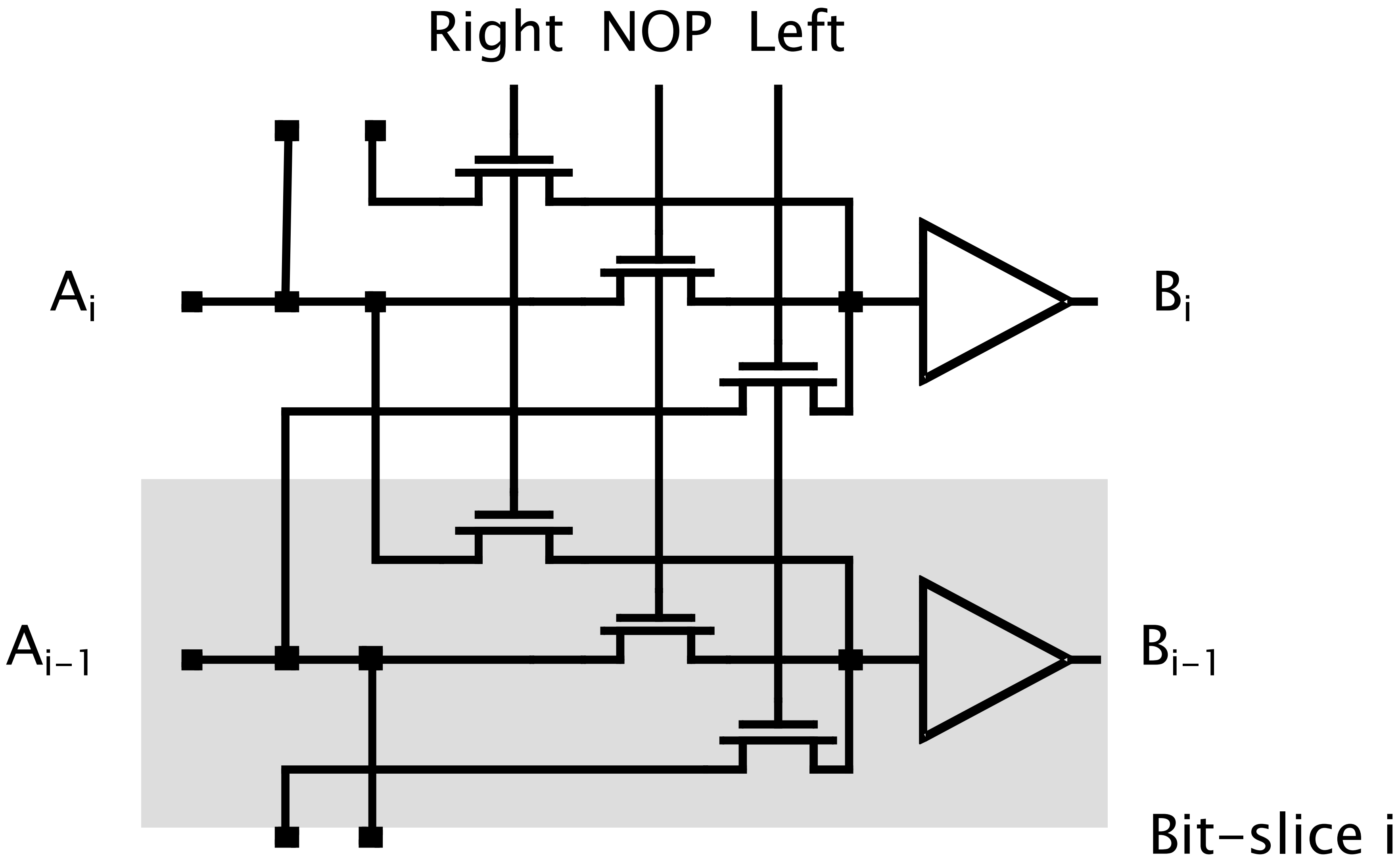
		P (conterá o resto)					A (conterá a divisão)				
passo	0	0	0	0	0	0	1	1	0	1	1
1	0	0	0	0	0	1	1	0	1	1	0
	0	0	0	0	0	1	1	0	1	1	0
2	0	0	0	0	1	1	0	1	1	0	0
	0	0	0	0	1	1	0	1	1	0	0
3	0	0	0	1	1	0	1	1	0	0	0
	0	0	0	0	0	1	1	1	0	0	1
4	0	0	0	0	1	1	1	0	0	1	0
	0	0	0	0	1	1	1	0	0	1	0
5	0	0	0	1	1	1	0	0	1	0	0
	0	0	0	0	1	0	0	0	1	0	1

Resto = 2 **resultado=5**

Shift and Rotate Operations

- Used in:
  - Microprocessors
  - Encryption algorithms
- If fixed shift, simply wire the inputs to the correct output positions
- Variable shift
  - One-bit shifter
  - Barrel shifter
  - Logarithmic shifter

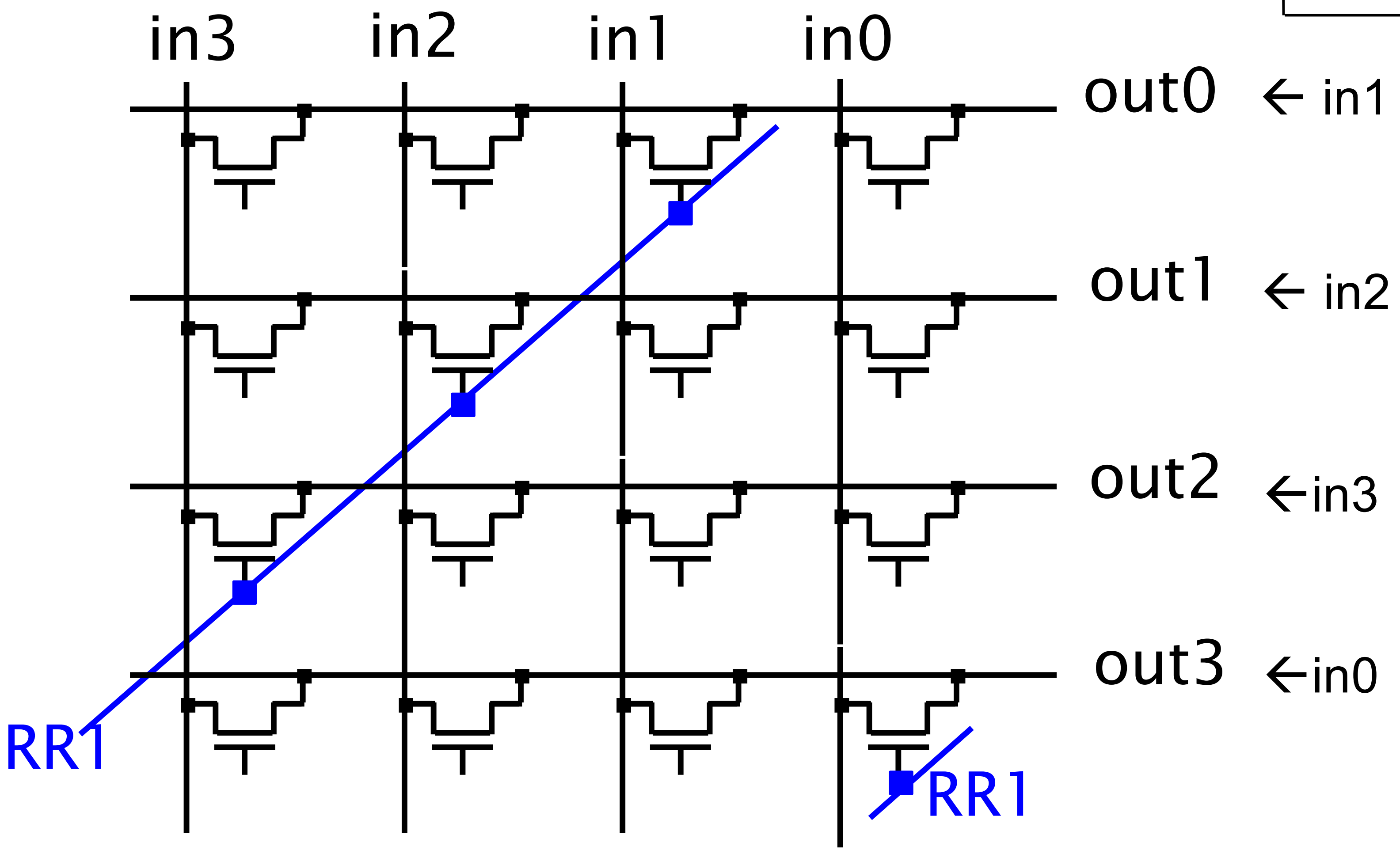
One-bit Shifter



n-bit Shifter (barrel shifter)

- Quadratic number of transistors
- One switch per path

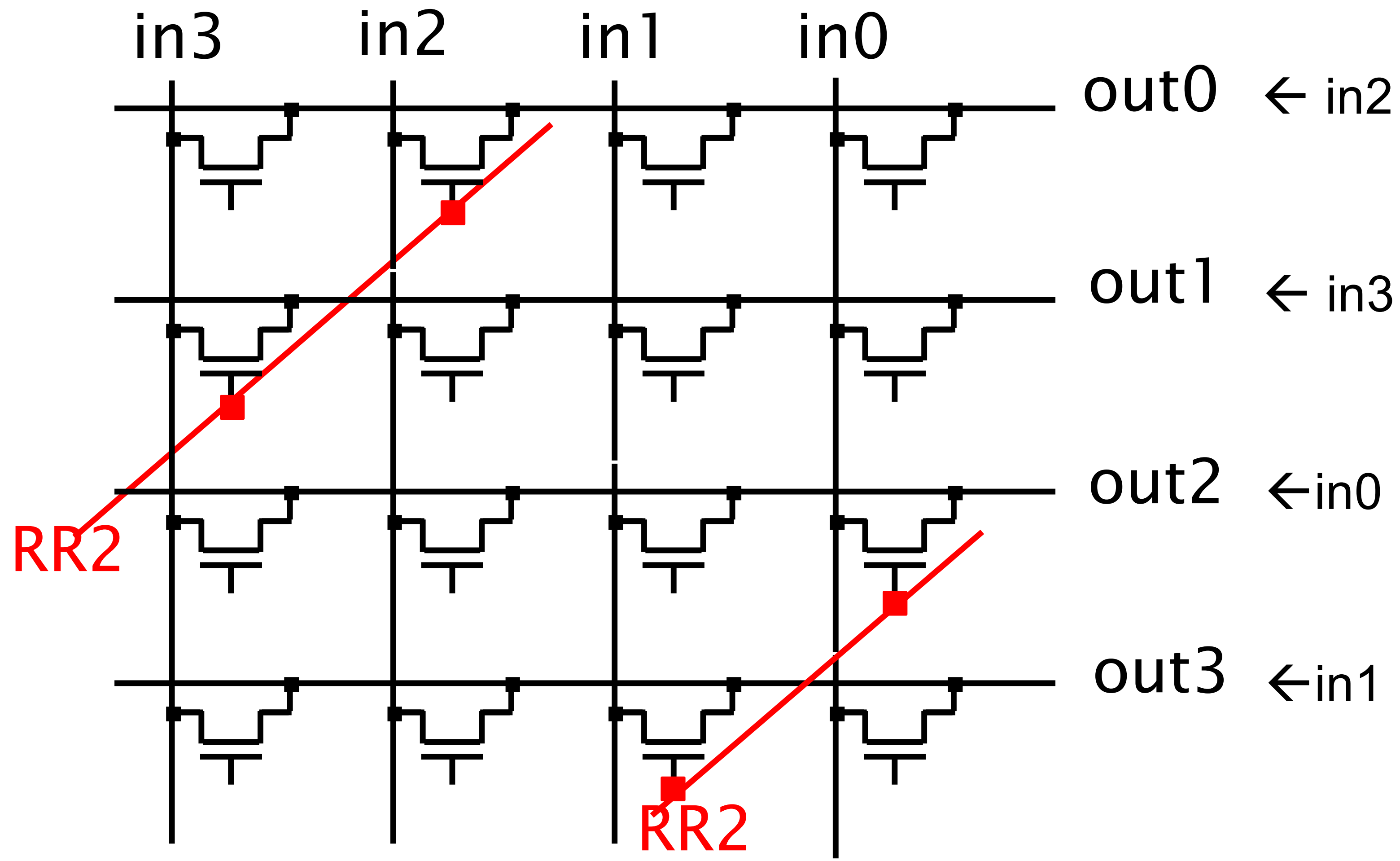
Rotacioando 1 bit para direita:
rotate 1 right(1100) → 0110



n-bit Shifter (barrel shifter)

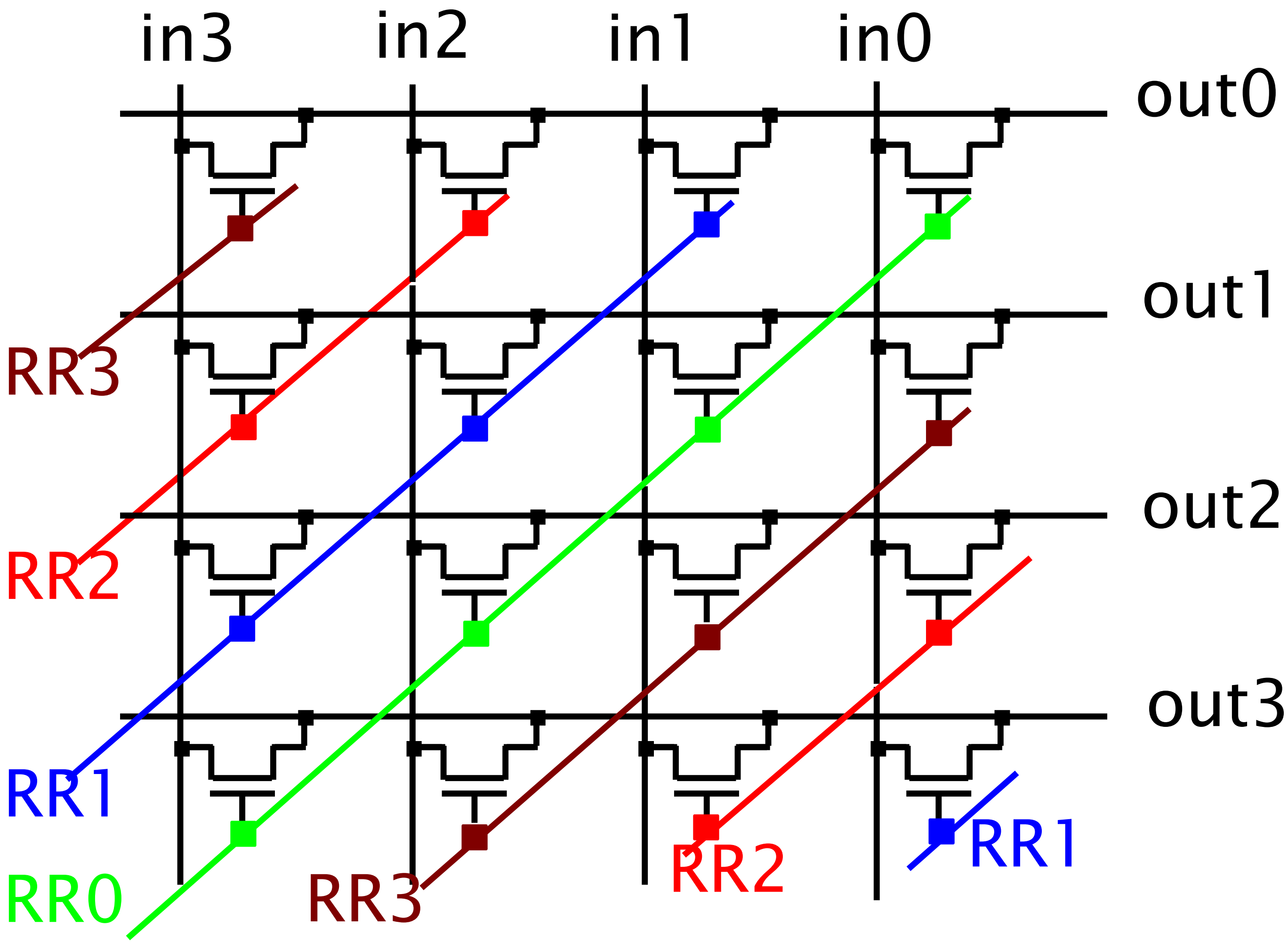
Rotacioando 2 bits para direita:

rotate 1 right(1100) → 0 0 11



n-bit Shifter (barrel shifter)

Circuito completo:



Deslocamento Aritmético

- Deslocamento lógico

  10001 → 1 bit para a direita 01000

  1 bit para a esquerda 00010

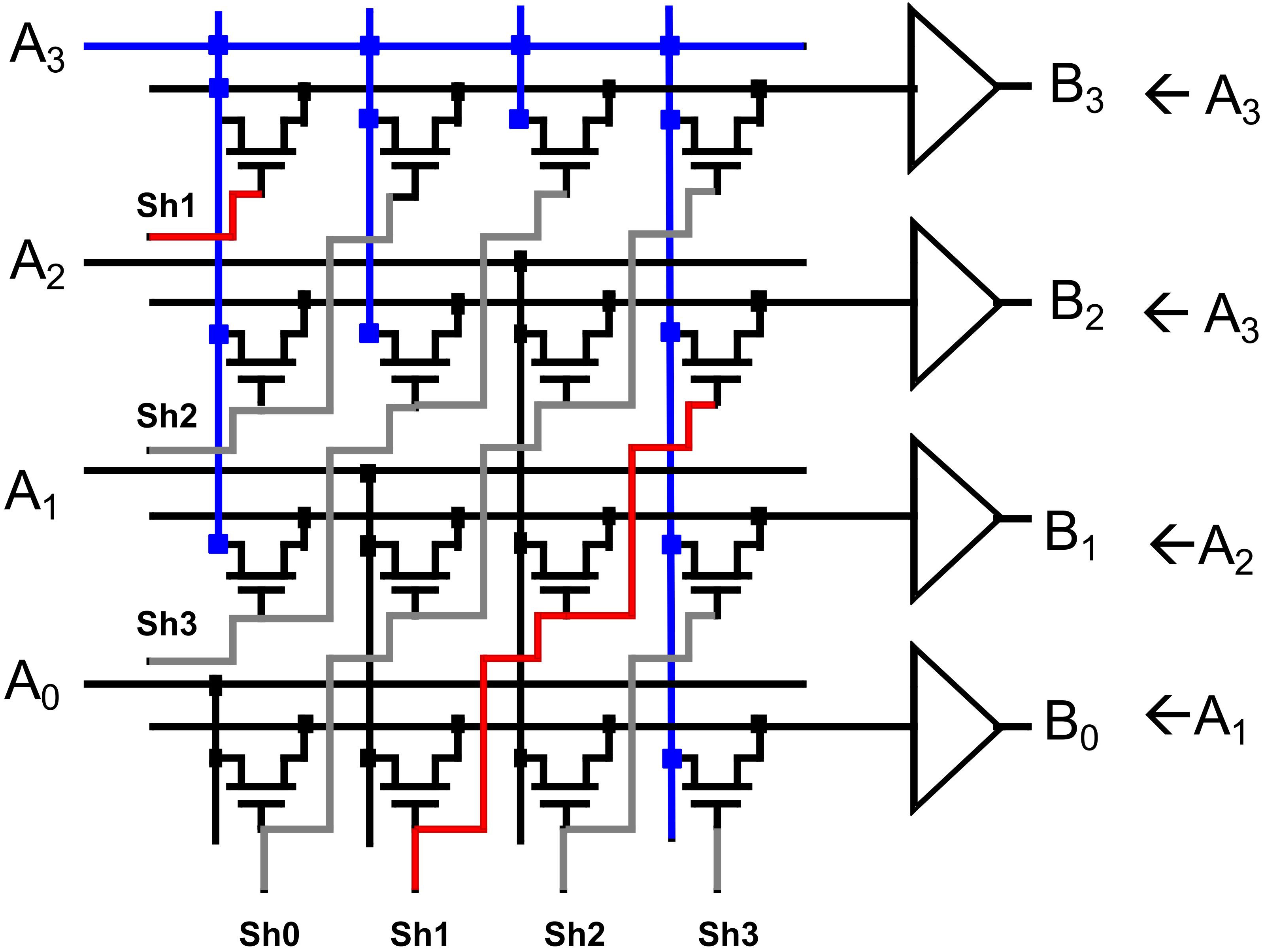
- Deslocamento aritmético

  00100 → 1 bit para a direita 00010 (de 4 para 2)

  10100 → 1 bit para a direita 11010 (de -12 para -6)

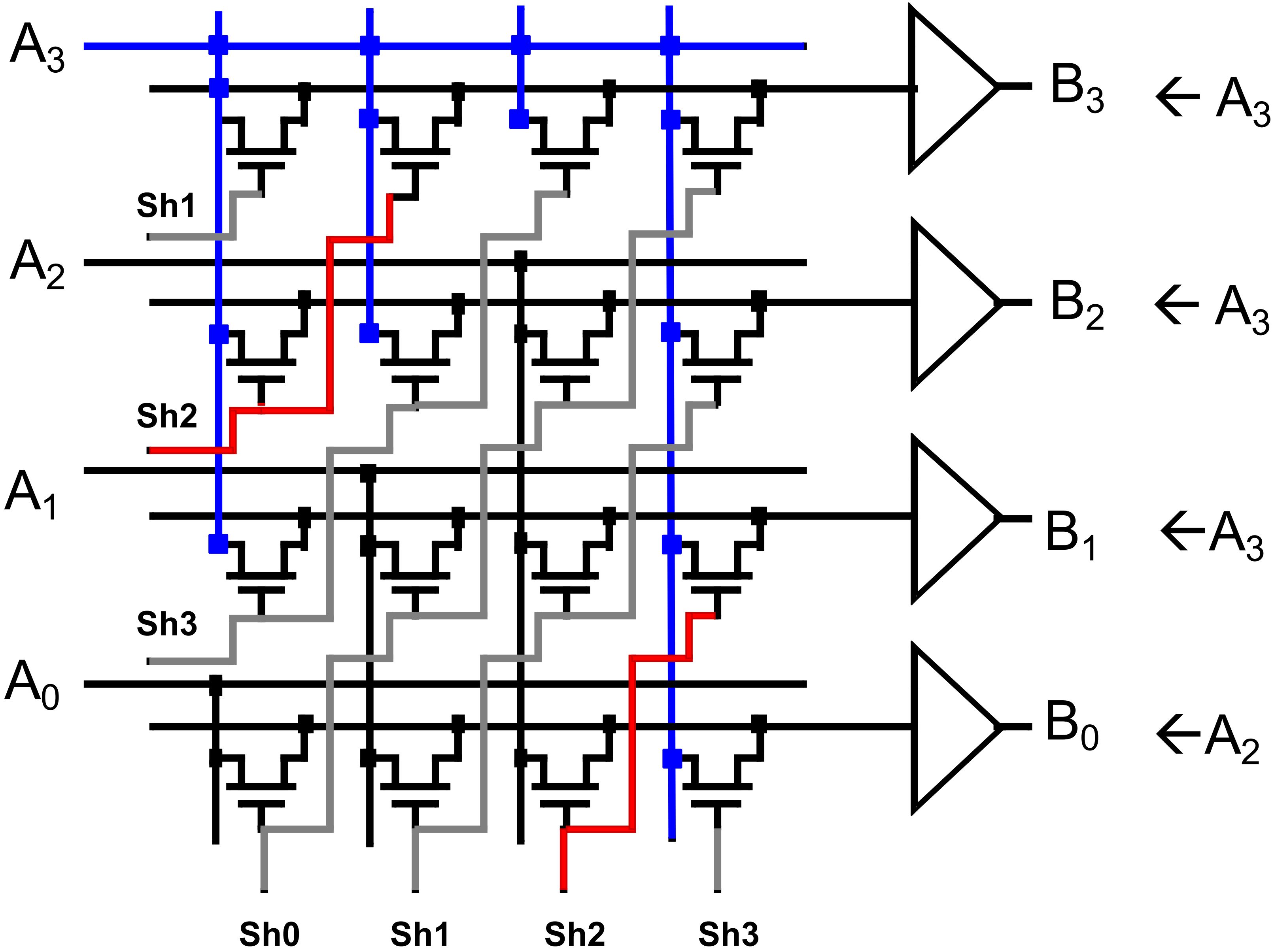
SH1 ← 1

Bit 3 wrapped around



Area dominated by wiring

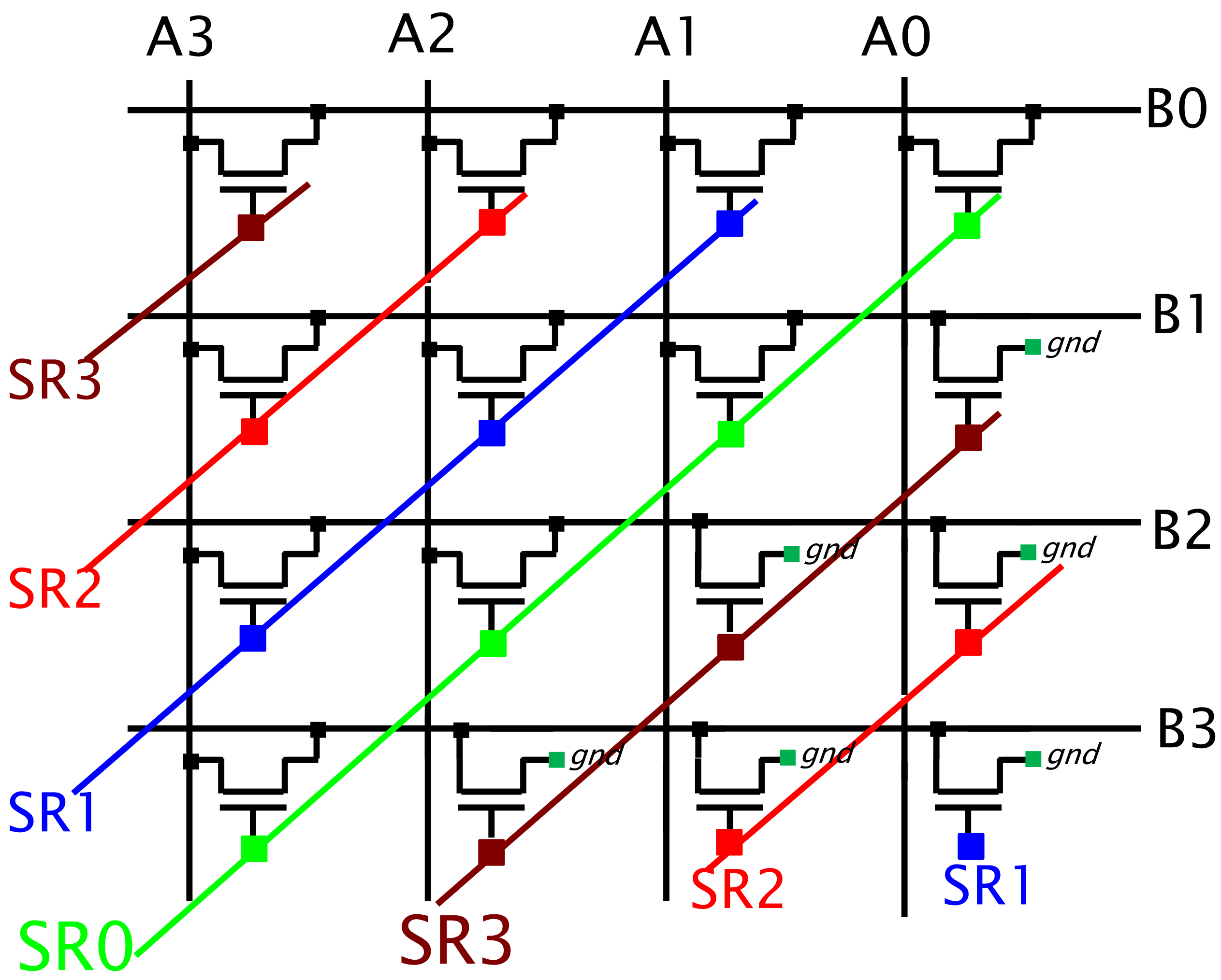
SH2 ← 1



A_3
A_2
A_1
A_0
Sh1
Sh2
Sh3
B_3
B_2
B_1
B_0
$\leftarrow A_3$
$\leftarrow A_3$
$\leftarrow A_3$
$\leftarrow A_2$
Sh0
Sh1
Sh2
Sh3

4. (1,5) **Circuitos de rotação e deslocamento.** Apresentar uma matriz de transistores capaz de realizar deslocamentos para a direita para palavras de 4 bits, onde o bit mais significativo recebe '0' (*gnd*). A tabela abaixo ilustra os valores que os bits de saída (B3 a B0) devem receber em função dos bits de entrada (A3 a A0).

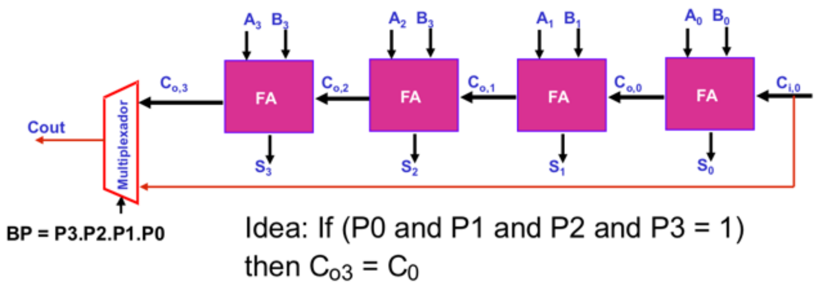
Ação	Comando	B3	B2	B1	B0
Não desloca	SR0	A3	A2	A1	A0
Deslocamento de 1 bit a direita	SR1	0	A3	A2	A1
Deslocamento de 2 bits a direita	SR2	0	0	A3	A2
Deslocamento de 3 bits a direita	SR3	0	0	0	A3



Considere o somador *carry bypass* com duas configurações distintas: estágios (M) de 4 bits e de 8 bits, com t_{FA}= 90ps e t_{MUX}= 20ps.

a) Determine o valor do número de bits em que o atraso dos somadores com M=4 e M=8 se igualam → N_{igual}

b) De 8 bits até N_{igual} qual configuração do somador é mais rápida?



Estágio de 4 bits do *carry bypass* (**CBP4**)

$$T = 2 \cdot M \cdot t_{FA} + \left(\frac{N}{M}\right) \cdot t_{mux}$$

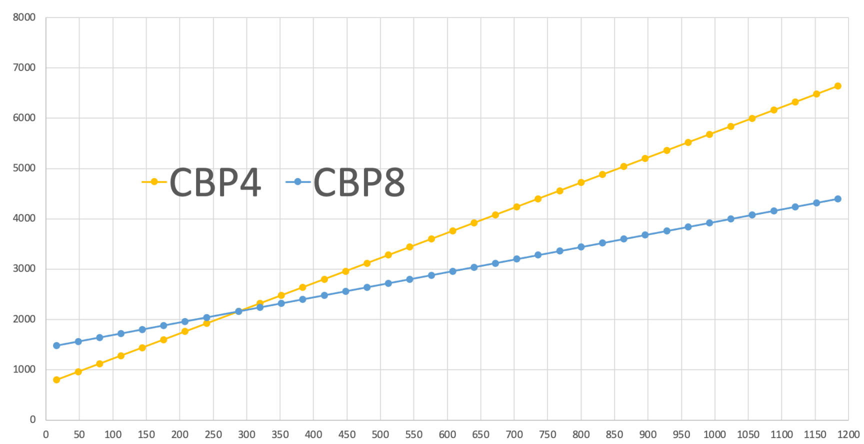
$$2 * \mathbf{8} * 90 + \frac{N}{\mathbf{8}} * 20 < 2 * \mathbf{4} * 90 + \frac{N}{\mathbf{4}} * 20$$

$$1440 + 2{,}5N < 720 + 5N$$

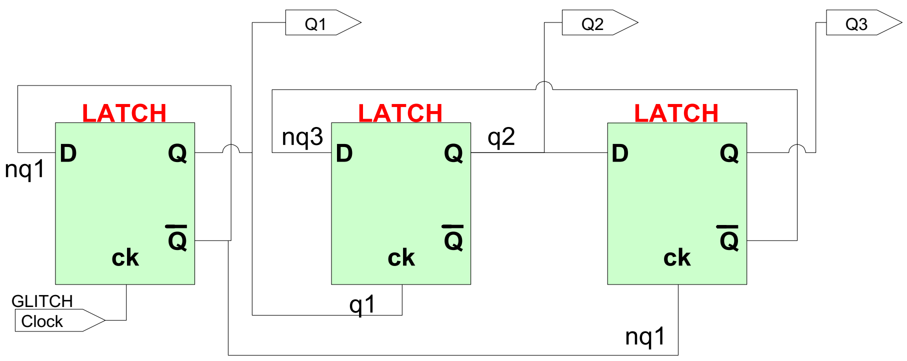
$$2{,}5N > 720$$

$$N > 288$$

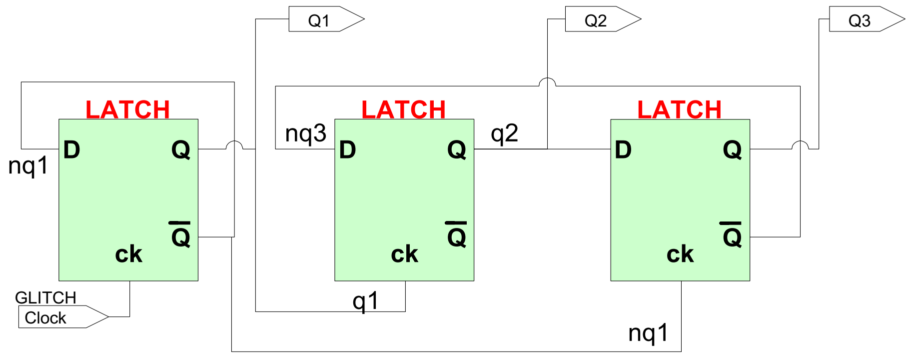
Até 288 bits o CP4 é mais rápido que o CP8. Após 288 bits o CP8 é mais rápido, pois há menos muxes no caminho

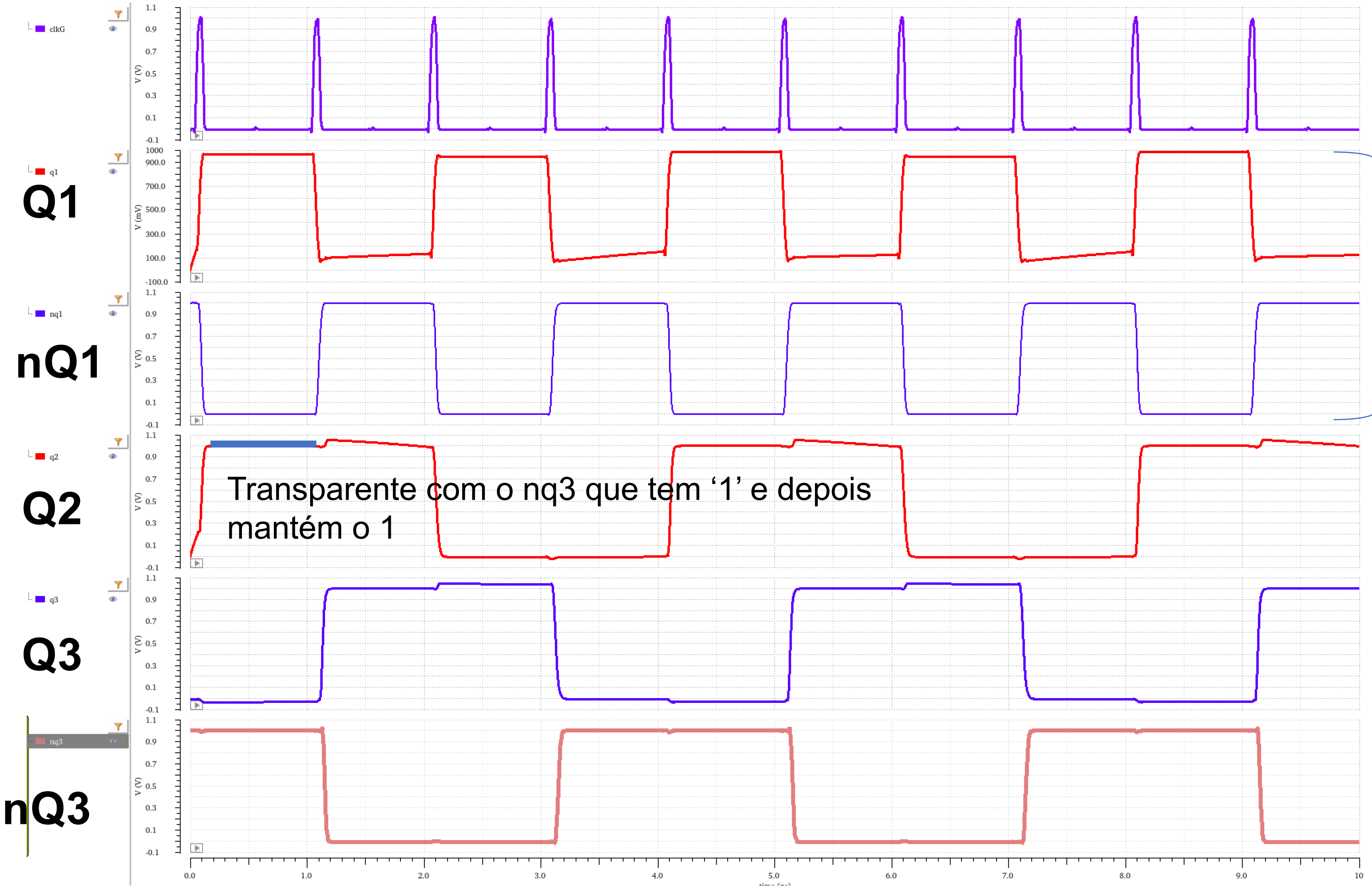


Um projetista desenvolveu o circuito abaixo para implementar um contador, usando apenas flip-flops do tipo *latch*, transparentes ao nível '1'do sinal 'ck', e um *clock* na forma de pulso (*glitch clock* - GCK) conectado ao pino 'ck' da primeira *latch*.



a) Apresente, para os primeiro **8 pulsos** de **GCK** (assumindo q1, q2, e q3 incialmente em zero) as formas de onda para os sinais: GCK, q1, nq1, q2, q3.
b) O sinal 'q1' opera como um divisor de *clock*? Explique.
c) Qual o comportamento do par 'q2-q3' em relação ao sinal 'q1'?





Divisor de clock

Divisor de clock (de Q1)

Exemplo de Circuito Complexo

IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 43, NO. 1, JANUARY 2008 29

An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS

Sriram R. Vangal, *Member, IEEE*, Jason Howard, Gregory Ruhl, *Member, IEEE*, Saurabh Dighe, *Member, IEEE*, Howard Wilson, James Tschanz, *Member, IEEE*, David Finan, Arvind Singh, *Member, IEEE*, Tiju Jacob, Shailendra Jain, Vasantha Erraguntla, *Member, IEEE*, Clark Roberts, Yatin Hoskote, *Member, IEEE*, Nitin Borkar, and Shekhar Borkar, *Member, IEEE*

***Abstract*—This paper describes an integrated network-on-chip architecture containing 80 tiles arranged as an 8×10 2-D array of floating-point cores and packet-switched routers, both designed to operate at 4 GHz. Each tile has two pipelined single-precision floating-point multiply accumulators (FPMAC) which feature a single-cycle accumulation loop for high throughput. The on-chip 2-D mesh network provides a bisection bandwidth of 2 Terabits/s. The 15-FO4 design employs mesochronous clocking, fine-grained clock gating, dynamic sleep transistors, and body-bias techniques. In a 65-nm eight-metal CMOS process, the 275 mm^2 custom design contains 100 M transistors. The fully functional first silicon achieves over 1.0 TFLOPS of performance on a range of benchmarks while dissipating 97 W at 4.27 GHz and 1.07 V supply.**

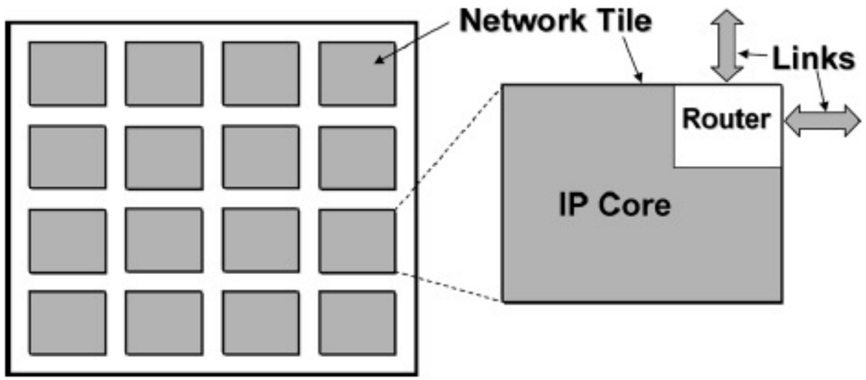
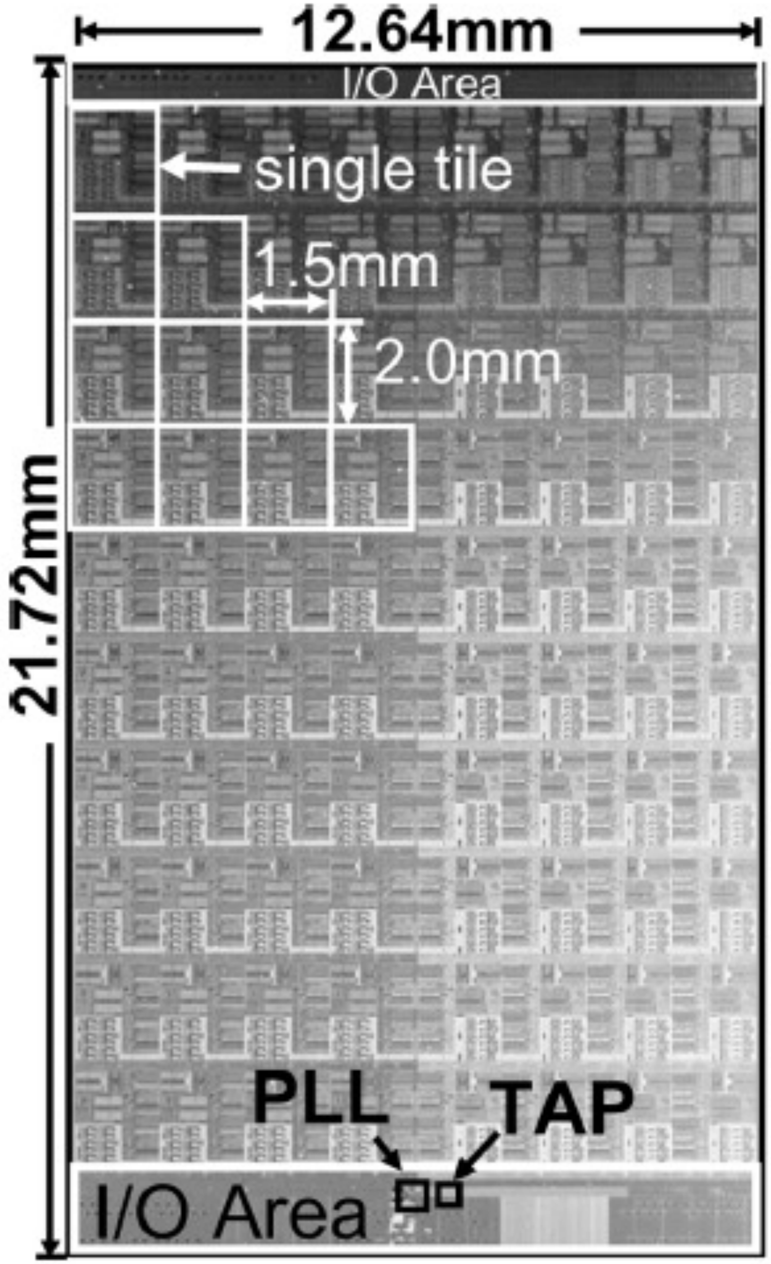
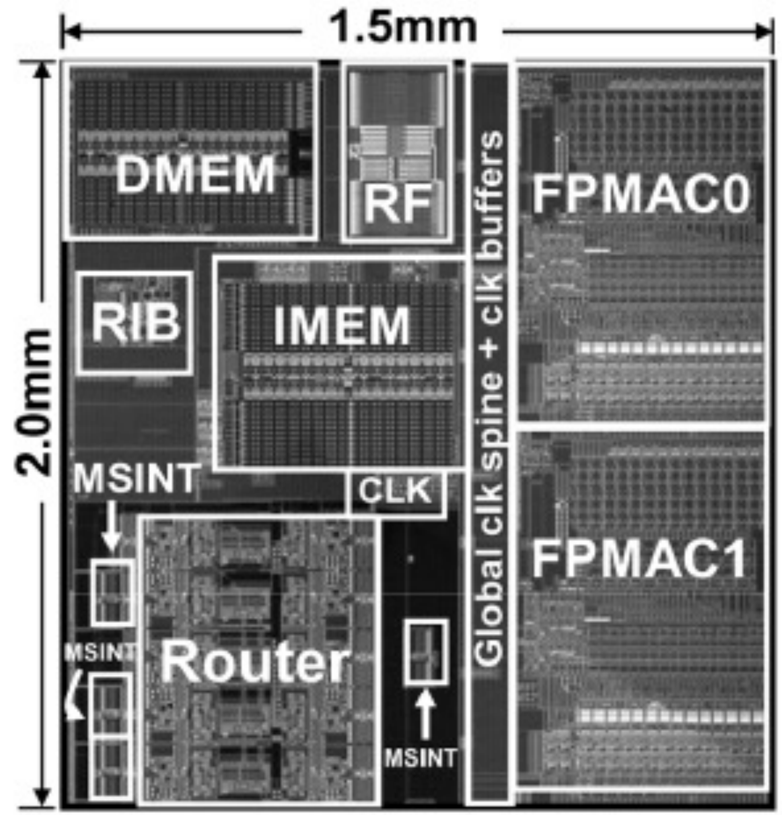
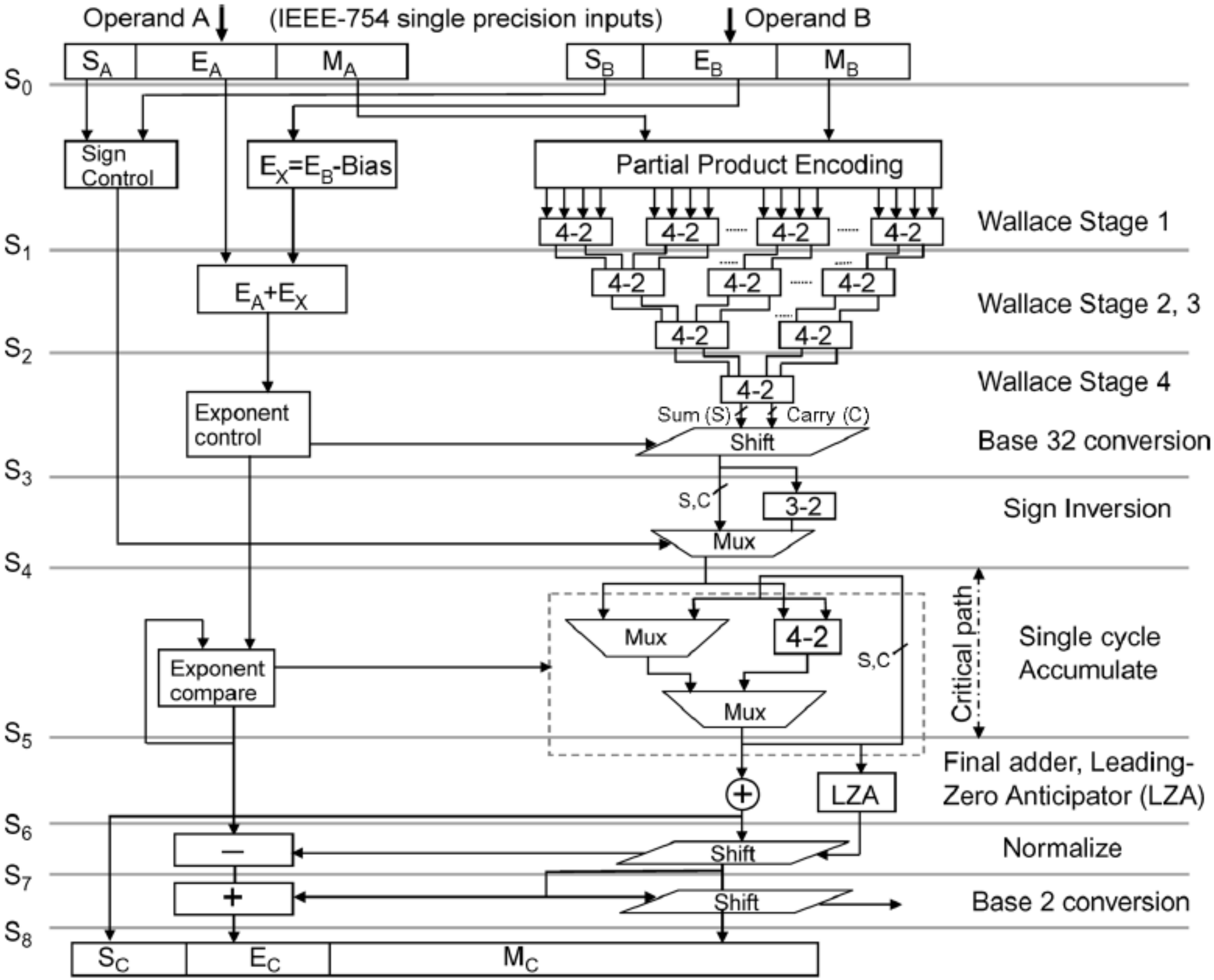


Fig. 1. NoC architecture.





Technology	65nm, 1 poly, 8 metal (Cu)
Transistors	100 Million (full-chip) 1.2 Million (tile)
Die Area	$275mm^2$ (full-chip) $3mm^2$ (tile)
C4 bumps #	8390



FPMAC nine-stage pipeline with single-cycle accumulate loop.

Flip-flops utilizados no circuito

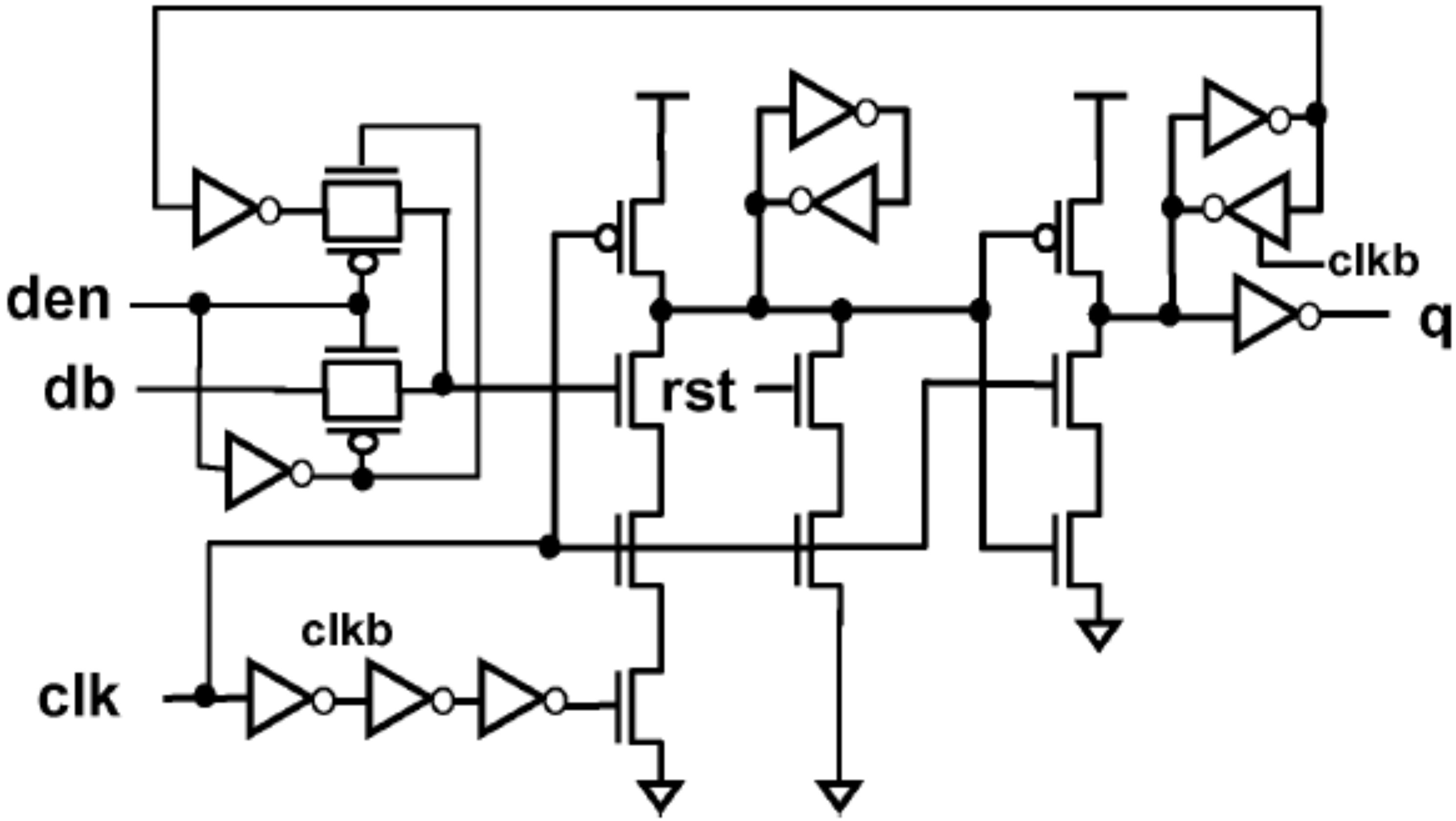
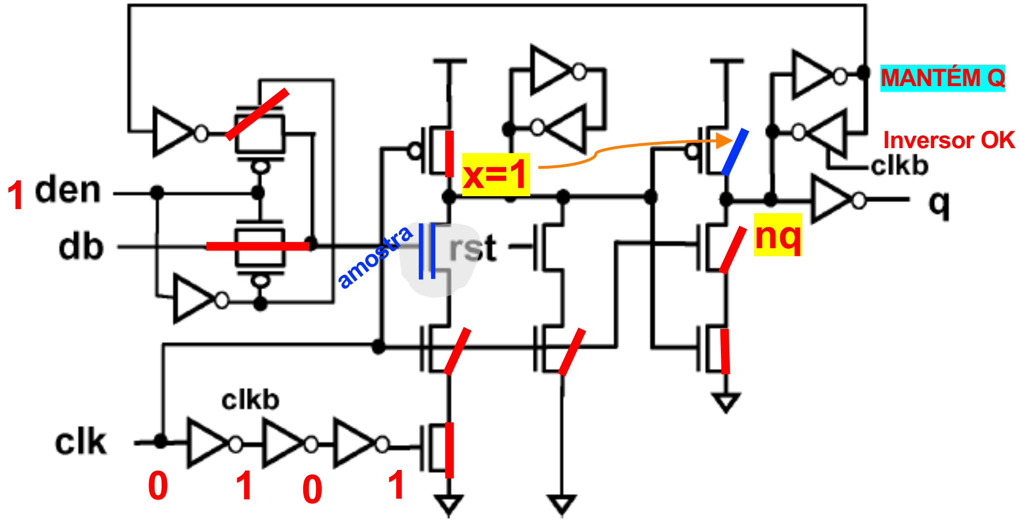
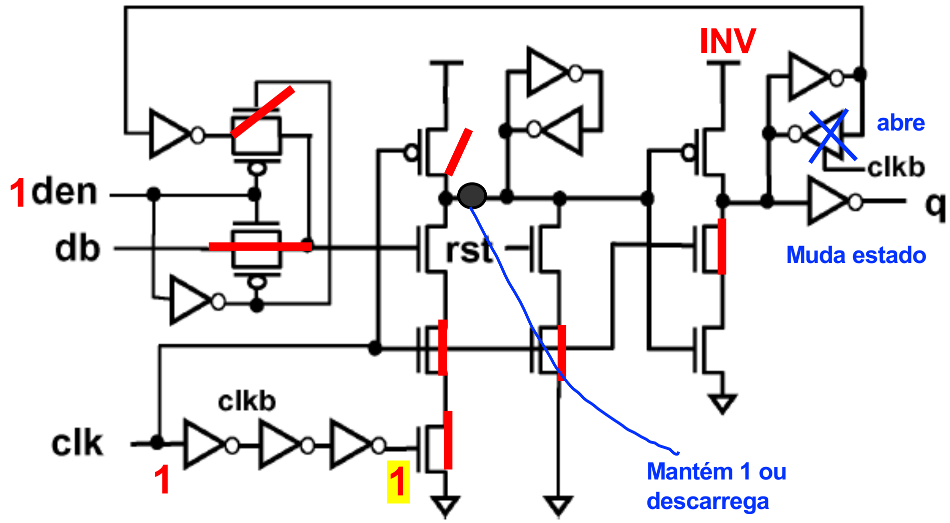


Fig. 8. Semi-dynamic flip-flop (SDFF) schematic.

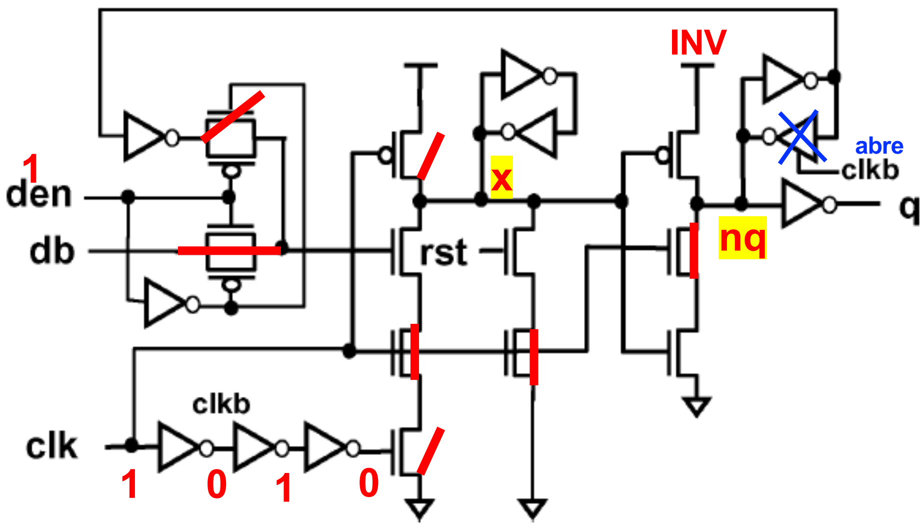


x : em pré-carga
com rst vai a zero - última informação amostrada

ck=0



ck= sobe



ck=1

x : último dado, isolado por 'Z' – não muda

ck= desce

- mesma condição de ck=0
- transfere Q do laço interno para o laço externo

den=0 : mantém a informação anterior, independente do clock