

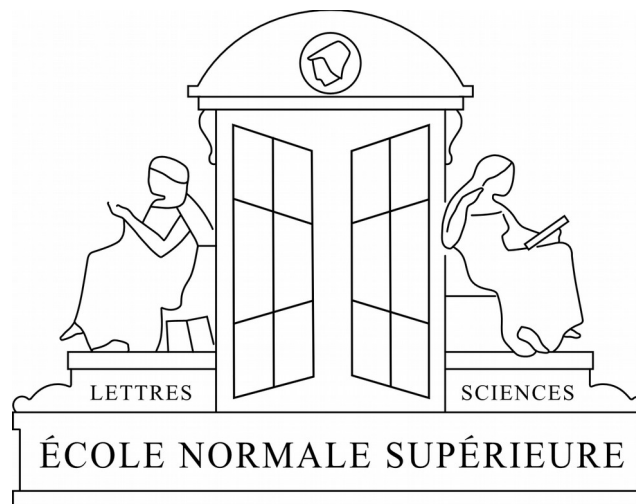
ENS L3 : "Systèmes numériques : de l'algorithme aux circuits"

Opérateurs spéciaux

Sylvain GUILLEY

< sylvain.guilley@telecom-paristech.fr >

< sylvain.guilley@secure-ic.com >



Agenda

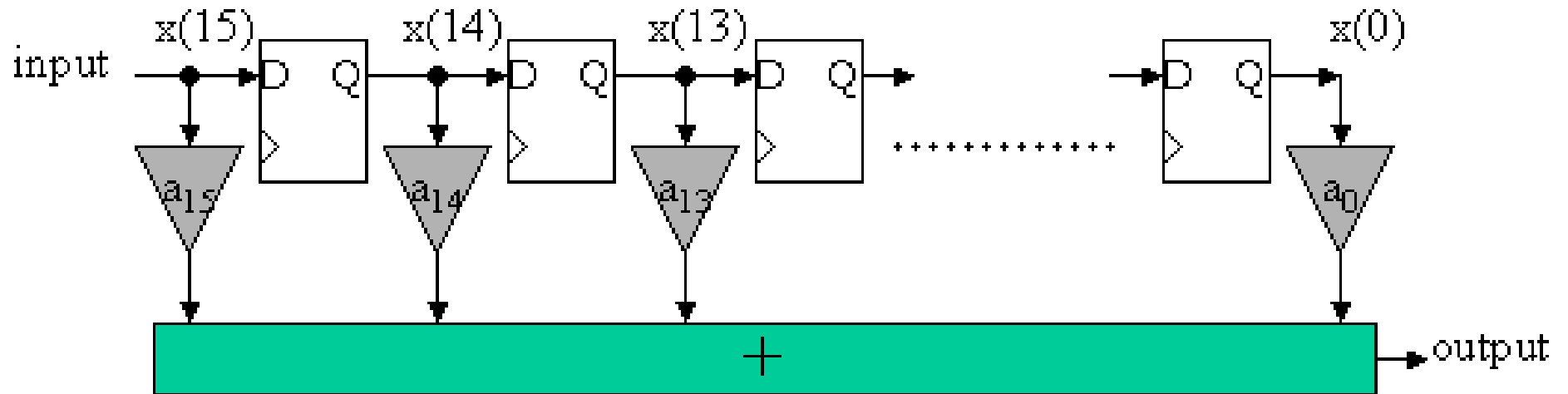
- FIR
- CRC
- CORDIC
- FFT
- Logique asynchrone
- Processeur et DMA
 - Cf. cours de Tim Bourke du 12 décembre

Finite Impulse Response Filter

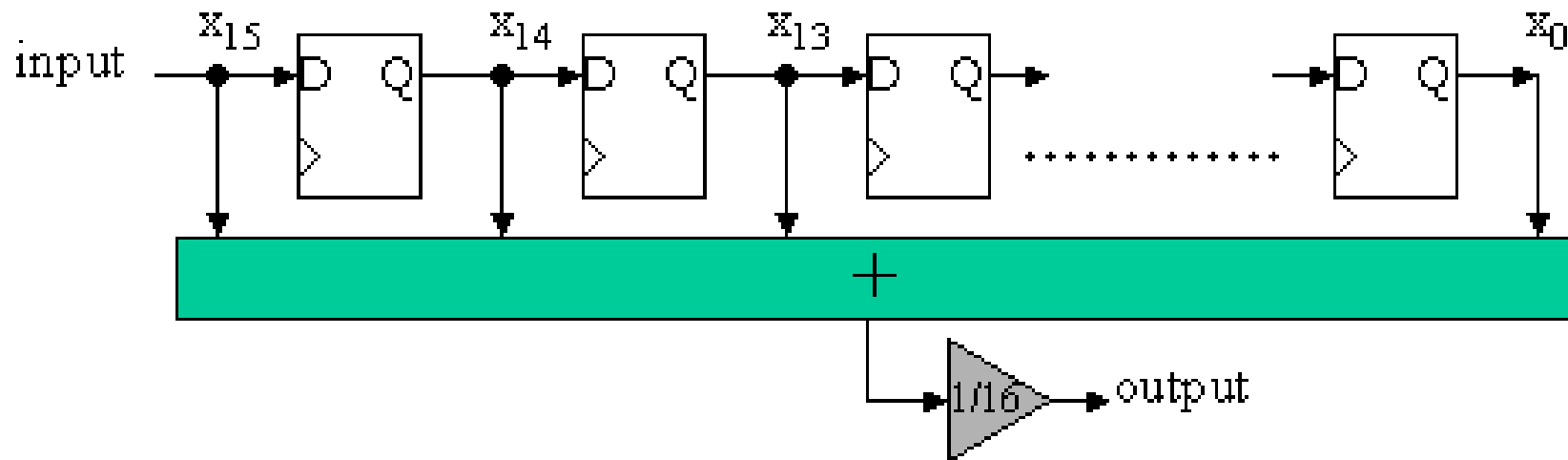
$$output(n) = \sum_{i=0}^{15} a_i \cdot x(n-15+i)$$

$$\begin{aligned} output(15) &= \sum_{i=0}^{15} a_i \cdot x(i) \\ &= a_0 \cdot x(0) + a_1 \cdot x(1) + \dots + a_{15} \cdot x(15) \end{aligned}$$

Finite Impulse Response Filter



(a) 16 Tap FIR Filter



(b) 16 Tap Averaging Filter

CRC : Cyclic Redundancy Checksum

$$(1x^3 + 0x^2 + 1x + 1) \rightarrow 1011$$

```
11010011101100 000 <--- input right padded by 3 bits
1011                <--- divisor
01100011101100 000 <--- result (note the first four bits are the XOR with the divisor
beneath, the rest of the bits are unchanged)
 1011                <--- divisor ...
00111011101100 000
 1011
00010111101100 000
 1011
00000001101100 000 <--- note that the divisor moves over to align with the next 1 in the
dividend (since quotient for that step was zero)
      1011          (in other words, it doesn't necessarily move one bit per
iteration)
00000000110100 000
      1011
00000000011000 000
      1011
00000000001110 000
      1011
00000000000101 000
      101 1
-----
00000000000000 100 <--- remainder (3 bits). Division algorithm stops here as dividend is
equal to zero.
```

Nota bene : 1-bit CRC = parity bit

CRC : Cyclic Redundancy Checksum

$(1x^3 + 0x^2 + 1x + 1)$ → polynomic division remainder

```
> R<X> := PolynomialRing(GF(2));  
> P := X^3 + X + 1;  
> IsIrreducible(P);  
true
```

← magma code

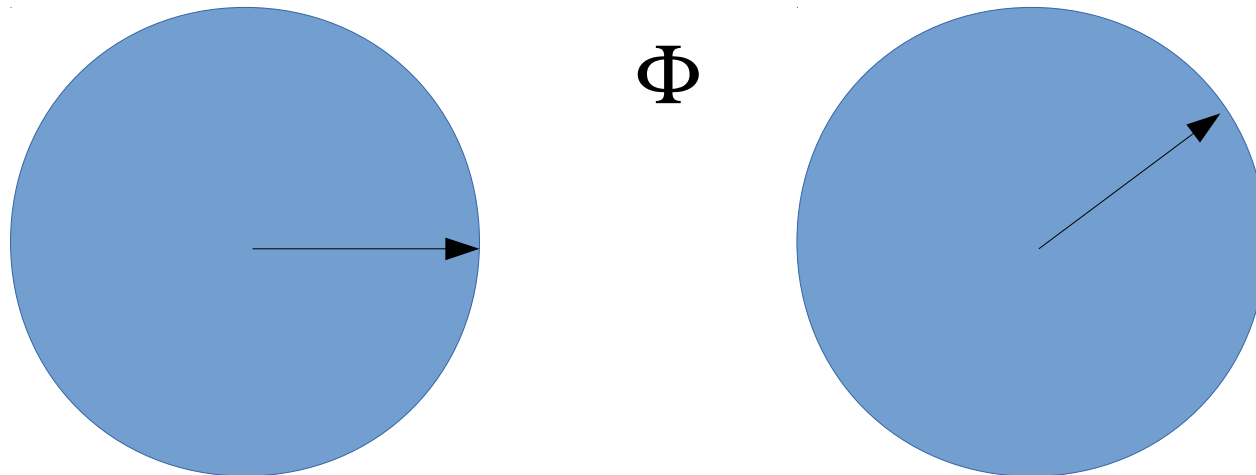
```
11010011101100 100 <--- input with check value  
1011             <--- divisor  
01100011101100 100 <--- result  
 1011           <--- divisor ...  
00111011101100 100  
  
.....  
  
00000000001110 100  
      1011  
00000000000101 100  
      101 1  
-----  
0 <--- remainder
```

Nota bene : 1-bit CRC = parity bit

CORDIC :

Coordinate Rotation Digital Computer

- Deprettere, E., Dewilde, P., and Udo, R.,
"Pipelined CORDIC Architecture for Fast VLSI
Filtering and Array Processing," Proc.
ICASSP'84, 1984, pp. 41.A.6.1-41.A.6.4
- In 2D plane $(x,y) \rightarrow (x',y')$
-



CORDIC

$$\begin{aligned}x' &= x \cos \phi - y \sin \phi \\y' &= y \cos \phi + x \sin \phi\end{aligned}$$



$$\begin{aligned}x' &= \cos \phi \cdot [x - y \tan \phi] \\y' &= \cos \phi \cdot [y + x \tan \phi]\end{aligned}$$



$$\tan(\phi) = \pm 2^{-i}$$

Arbitrary angles of rotation are obtainable by performing a series of successively smaller elementary rotations. If the decision at each iteration, i , is which direction to rotate rather than whether or not to rotate, then the $\cos(\delta_i)$ term becomes a constant (because $\cos(\delta_i) = \cos(-\delta_i)$). The iterative rotation can now be expressed as:

$$x_{i+1} = K_i [x_i - y_i \cdot d_i \cdot 2^{-i}]$$

$$y_{i+1} = K_i [y_i + x_i \cdot d_i \cdot 2^{-i}]$$

where:

$$K_i = \cos(\tan^{-1} 2^{-i}) = 1/\sqrt{1+2^{-2i}}$$

$$d_i = \pm 1$$

CORDIC

Lemma 1.

$$\cos(\arctan x) = \frac{1}{\sqrt{1+x^2}}.$$

CORDIC

Lemma 1.

$$\cos(\arctan x) = \frac{1}{\sqrt{1+x^2}}.$$

Proof.

$$\begin{aligned}\cos^2 y + \sin^2 y &= 1 \\ \implies 1 + \tan^2 y &= \frac{1}{\cos^2 y} \\ \implies \cos^2 y &= \frac{1}{1 + \tan^2 y}\end{aligned}$$

apply formula for $y = \arctan x$. Notice that when $x \geq 0$, $\arctan x \geq 0$, hence $\cos(\arctan x) \geq 0$. □

CORDIC

CORDIC equations are:

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}$$

$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i})$$

where

$d_i = -1$ if $z_i < 0$, $+1$ otherwise

which provides the following result:

$$x_n = A_n [x_0 \cos z_0 - y_0 \sin z_0]$$

$$y_n = A_n [y_0 \cos z_0 + x_0 \sin z_0]$$

$$z_n = 0$$

$$A_n = \prod_n \sqrt{1 + 2^{-2i}}$$

```
import math
A = 1
for i in range(0,10):
    A *= math.sqrt( 1+2**(-2*i) )
    print( "A_{i} = {A:.3f}".format( i=i, A=A ))
```

```
A_0 = 1.414
A_1 = 1.581
A_2 = 1.630
A_3 = 1.642
A_4 = 1.646
A_5 = 1.646
A_6 = 1.647
A_7 = 1.647
A_8 = 1.647
A_9 = 1.647
```

FFT

$$X_m = \sum_{n=0}^{N-1} x_n w^{nm},$$

where N is the size of the vectors, $w = e^{2i\pi/N}$ are the “roots-of-unity” (twiddle factors), and $0 \leq m < N$.

$$X_m = \sum_{n=0}^{N/2-1} x_n w^{nm} + w^{mN/2} \sum_{n=0}^{N/2-1} x_{n+N/2} w^{nm},$$

DIF

$$X_m = \sum_{n=0}^{N/2-1} x_{2n} w^{2nm} + w^m \sum_{n=0}^{N/2-1} x_{2n+1} w^{2nm}.$$

DIT

FFT example on 8 bits

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & w^8 & w^{10} & w^{12} & w^{14} \\ 1 & w^3 & w^6 & w^9 & w^{12} & w^{15} & w^{18} & w^{21} \\ 1 & w^4 & w^8 & w^{12} & w^{16} & w^{20} & w^{24} & w^{28} \\ 1 & w^5 & w^{10} & w^{15} & w^{20} & w^{25} & w^{30} & w^{35} \\ 1 & w^6 & w^{12} & w^{18} & w^{24} & w^{30} & w^{36} & w^{42} \\ 1 & w^7 & w^{14} & w^{21} & w^{28} & w^{35} & w^{42} & w^{49} \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}$$

FFT example on 8 bits

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & | & 1 & 1 & 1 & 1 \\ 1 & w^2 & w^4 & w^6 & | & w & w^3 & w^5 & w^7 \\ 1 & w^4 & w^8 & w^{12} & | & w^2 & w^6 & w^{10} & w^{14} \\ 1 & w^6 & w^{12} & w^{18} & | & w^3 & w^9 & w^{15} & w^{21} \\ \hline 1 & w^8 & w^{16} & w^{24} & | & w^4 & w^{12} & w^{20} & w^{28} \\ 1 & w^{10} & w^{20} & w^{30} & | & w^5 & w^{15} & w^{25} & w^{35} \\ 1 & w^{12} & w^{24} & w^{36} & | & w^6 & w^{18} & w^{30} & w^{42} \\ 1 & w^{14} & w^{28} & w^{42} & | & w^7 & w^{21} & w^{35} & w^{49} \end{bmatrix} \begin{pmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \\ \hline x_1 \\ x_3 \\ x_5 \\ x_7 \end{pmatrix}$$

FFT example on 8 bits

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w^4 & w^2 & w^6 & w & w^5 & w^3 & w^7 \\ \hline 1 & w^8 & w^4 & w^{12} & w^2 & w^{10} & w^6 & w^{14} \\ 1 & w^{12} & w^6 & w^{18} & w^3 & w^{15} & w^9 & w^{21} \\ \hline 1 & w^{16} & w^8 & w^{24} & w^4 & w^{20} & w^{12} & w^{28} \\ 1 & w^{20} & w^{10} & w^{30} & w^5 & w^{25} & w^{15} & w^{35} \\ \hline 1 & w^{24} & w^{12} & w^{36} & w^6 & w^{30} & w^{18} & w^{42} \\ 1 & w^{28} & w^{14} & w^{42} & w^7 & w^{35} & w^{21} & w^{49} \end{bmatrix} \begin{pmatrix} x_0 \\ x_4 \\ \hline x_2 \\ x_6 \\ \hline x_1 \\ x_5 \\ \hline x_3 \\ x_7 \end{pmatrix}$$

FFT example on 8 bits

$$w^n = w^{n+Nk}, N = 8, k = 0, 1, 2, \dots$$

$$w^n = -w^{n+N/2}$$

$$w^{Nk} = 1.$$

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{pmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & w^2 & -w^2 & w & -w & w^3 & -w^3 \\ \hline 1 & 1 & -1 & -1 & w^2 & w^2 & -w^2 & -w^2 \\ 1 & -1 & -w^2 & w^2 & w^3 & -w^3 & w & -w \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & w^2 & -w^2 & -w & w & -w^3 & w^3 \\ \hline 1 & 1 & -1 & -1 & -w^2 & -w^2 & w^2 & w^2 \\ 1 & -1 & -w^2 & w^2 & -w^3 & w^3 & -w & w \end{bmatrix} \begin{pmatrix} x_0 \\ x_4 \\ \hline x_2 \\ x_6 \\ \hline x_1 \\ x_5 \\ \hline x_3 \\ x_7 \end{pmatrix}$$

FFT example on 8 bits

ILLUSTRATION OF THE BIT-REVERSED INDICES.

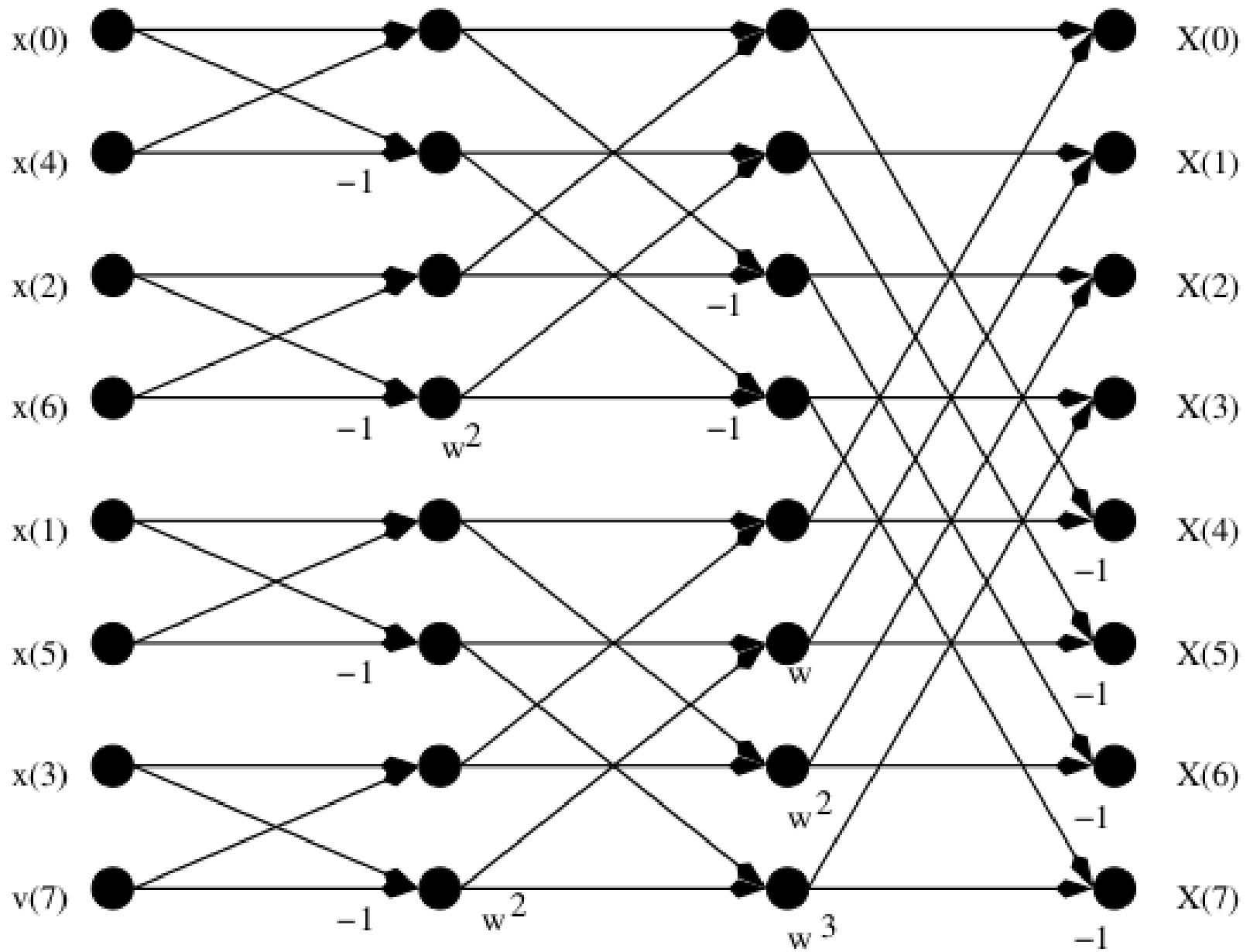
| Index | binary | Bit reversed index | binary |
|-------|--------|--------------------|--------|
| 0 | 000 | 0 | 000 |
| 1 | 001 | 4 | 100 |
| 2 | 010 | 2 | 010 |
| 3 | 011 | 6 | 110 |
| 4 | 100 | 1 | 001 |
| 5 | 101 | 5 | 101 |
| 6 | 110 | 3 | 011 |
| 7 | 111 | 7 | 111 |

FFT example on 8 bits

$$(X) = [A_2][A_1][A_0][P](x),$$

$$[A_0] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \quad [A_1] = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & w^2 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -w^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & w^2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -w^2 \end{bmatrix} \quad [A_2] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & w & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & w^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & w^3 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -w & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -w^2 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -w^3 \end{bmatrix}$$

Final architecture



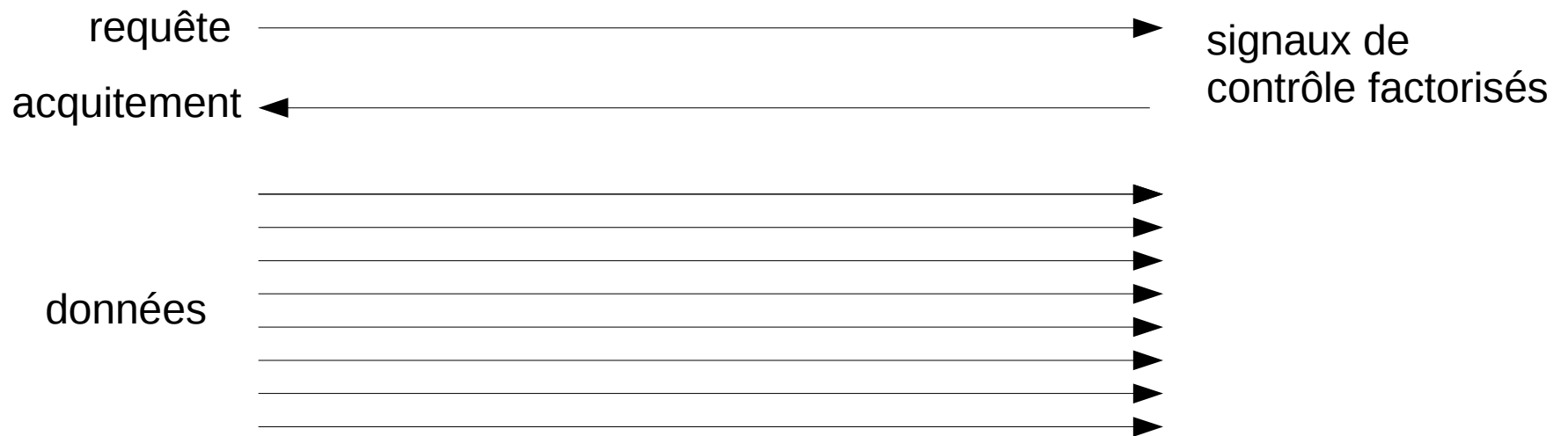
DCT : Discrete Cosine Transform

$$X_k = \sum_{n=0}^{N-1} x_n \cos \left[\frac{\pi}{N} \left(n + \frac{1}{2} \right) k \right] \quad k = 0, \dots, N - 1.$$

Used in JPEG image coding.

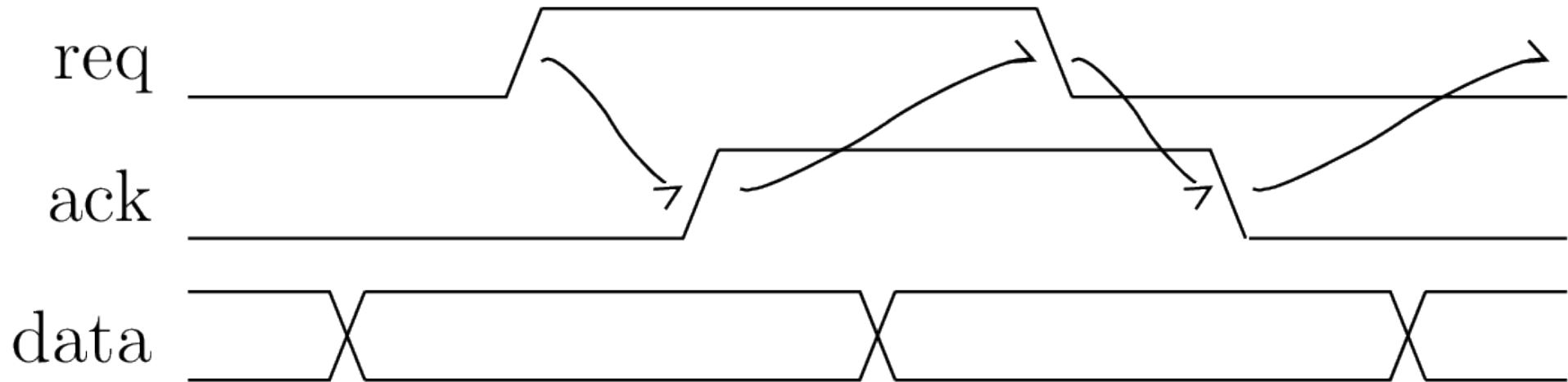
Logique asynchrone

- Illustration de logique séquentielle sans horloge
- Mise en place d'une signalisation



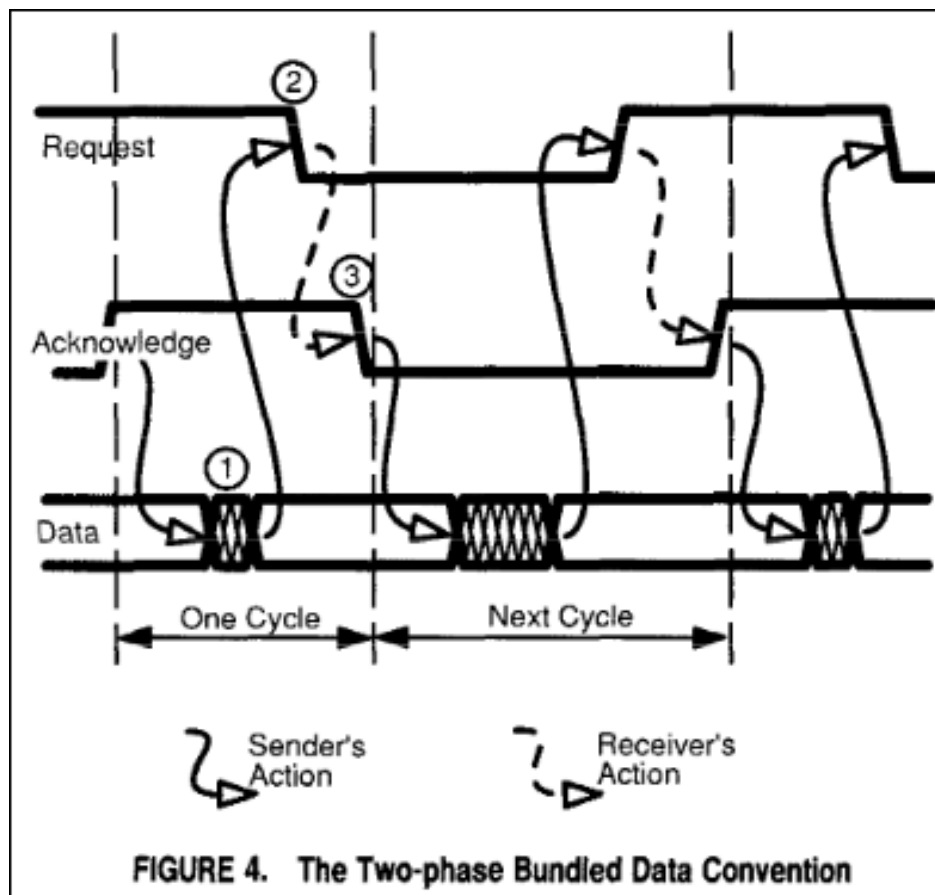
Logique asynchrone

- Illustration de logique séquentielle sans horloge
- Mise en place d'une signalisation



Logique asynchrone

- Illustration de logique séquentielle sans horloge
- Mise en place d'une signalisation



Ivan Sutherland,
Turing award 1988

Communications of
the ACM, June
1989, Volume 32,
Number 6.

Rendez-vous



IF inputs match in state
THEN copy it for output
ELSE hold previous state;



IF inputs match in state
THEN invert it for output
ELSE hold previous state;



IF inputs differ in state
THEN copy upper for output
ELSE hold previous state;

FIGURE 8. Muller C-Elements with Inverters

Muller C-elements contain storage to hold a previous state on some input conditions. When inverters are included in input or output wires, as indicated by the bubbles in this figure, the actions are as listed. Muller C-elements provide the AND function for events.

Micro-pipeline

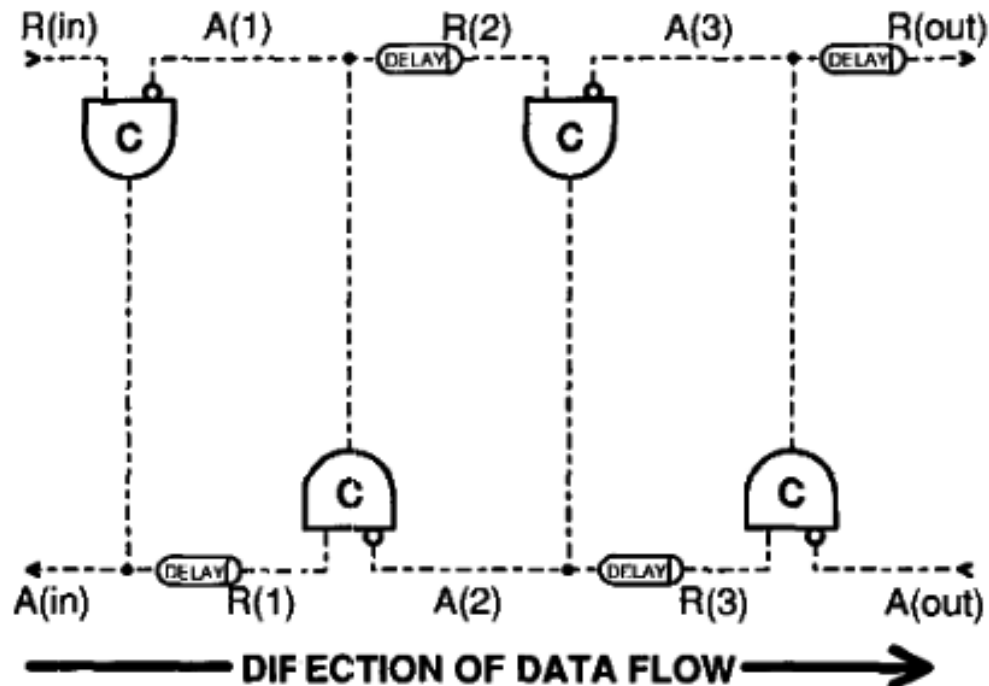


FIGURE 10. Control Circuit for a Micropipeline

With data paths omitted, the control circuit for a micropipeline is a string of Muller C-elements. In this figure one of four identical stages is shaded and alternate stages have been drawn upside down. At the input and output to each stage there are request, $R(n)$, and acknowledge, $A(n)$, signals. Inverters in the acknowledge paths are represented by "bubbles" at one input of each Muller C-element. The delays shown explicitly here may not be required for simple data paths. Notice that each loop in this circuit contains exactly one inversion, the bubble, and is therefore an oscillator. The Muller C-elements retard the oscillation in each loop to coordinate it with the actions of adjacent loops. In this and other figures, dotted wires carry event signals.

Data flow

```
if <cond> then <body1> else <body2>
```

```
while <cond> do <body>
```

(Fig. 1a) et

(Fig. 1b).

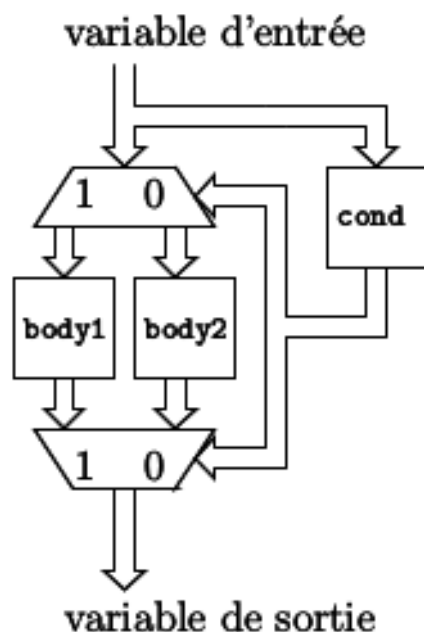


Fig. 1a : *Modèle d'architecture matérielle asynchrone pour le branchement if <cond> then <body1> else <body2>.*

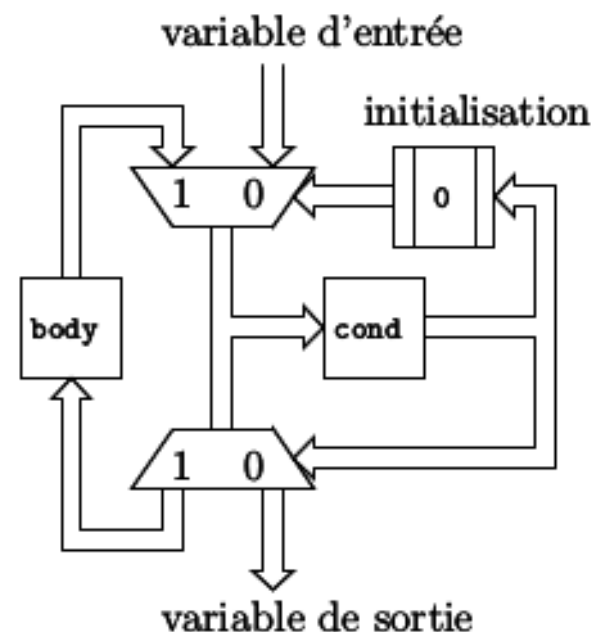


Fig. 1b : *Modèle d'architecture matérielle asynchrone pour le branchement while <cond> do <body>.*

Latch avec poignée de main

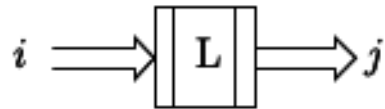


Fig. 2a : *Symbole du LATCH (partie « données »).*

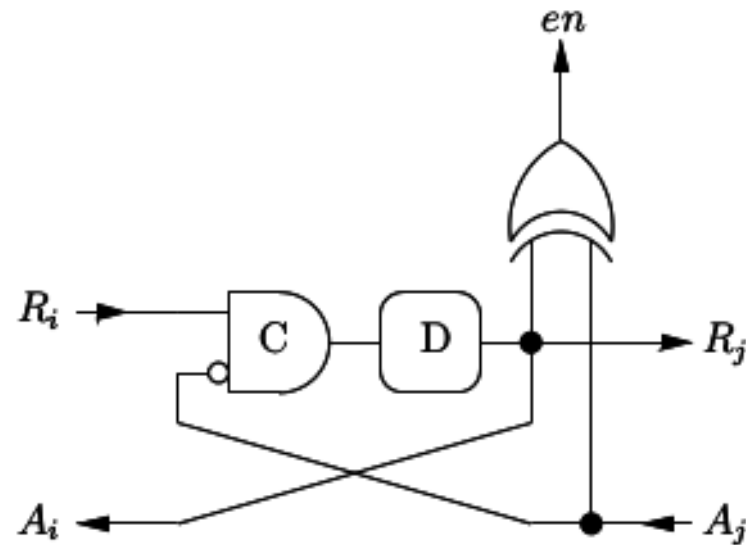


Fig. 2b : *Handshake pour le LATCH (partie « contrôle »). Le signal en sert à contrôler le latch.*

Element utile pour MUX / DEMUX

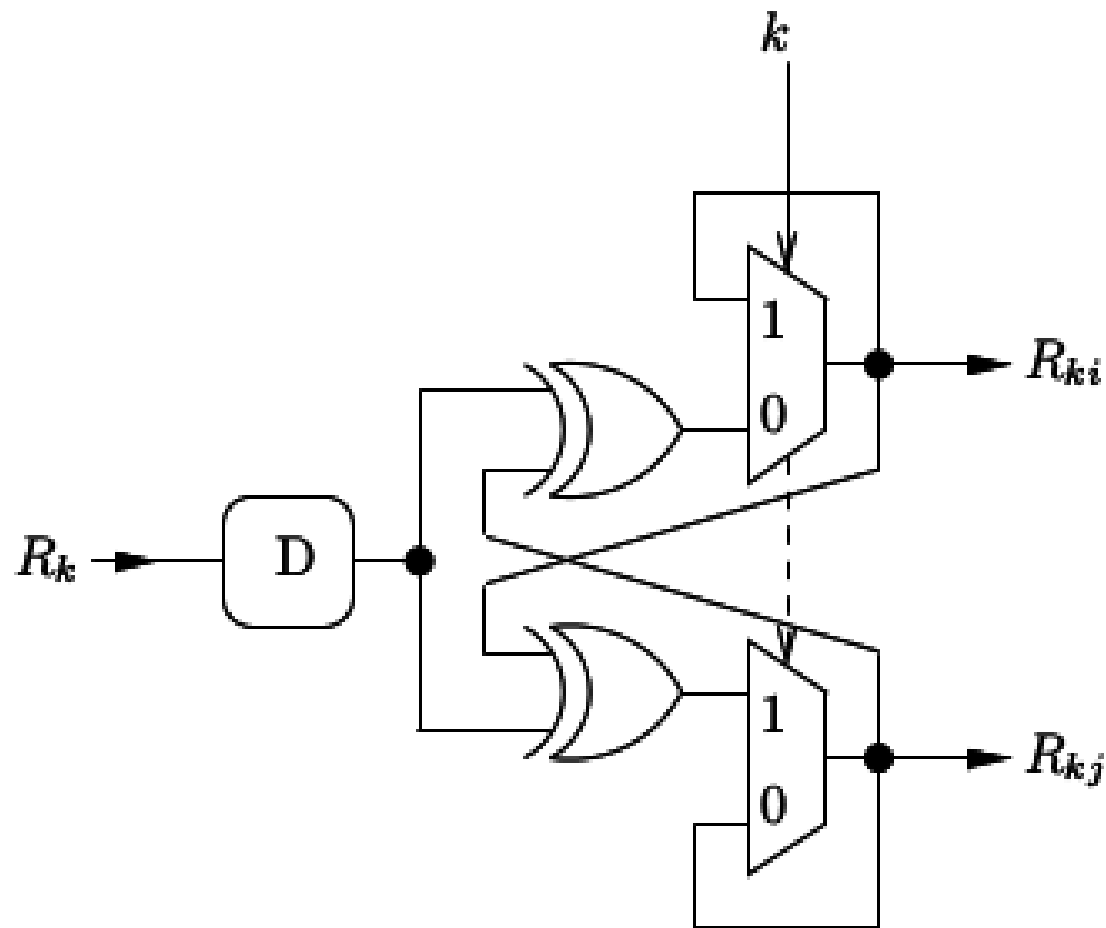


Fig. 3 : *Circuit de transformation de $\{k, R_k\}$ en $\{R_{ki}, R_{kj}\}$.*

MUX

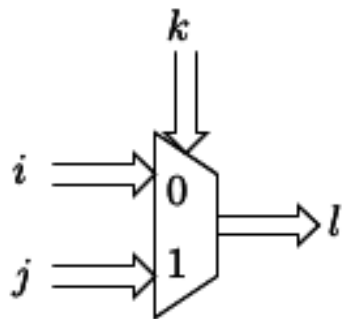


Fig. 4a : *Symbole du MUX22 (partie « données »).*

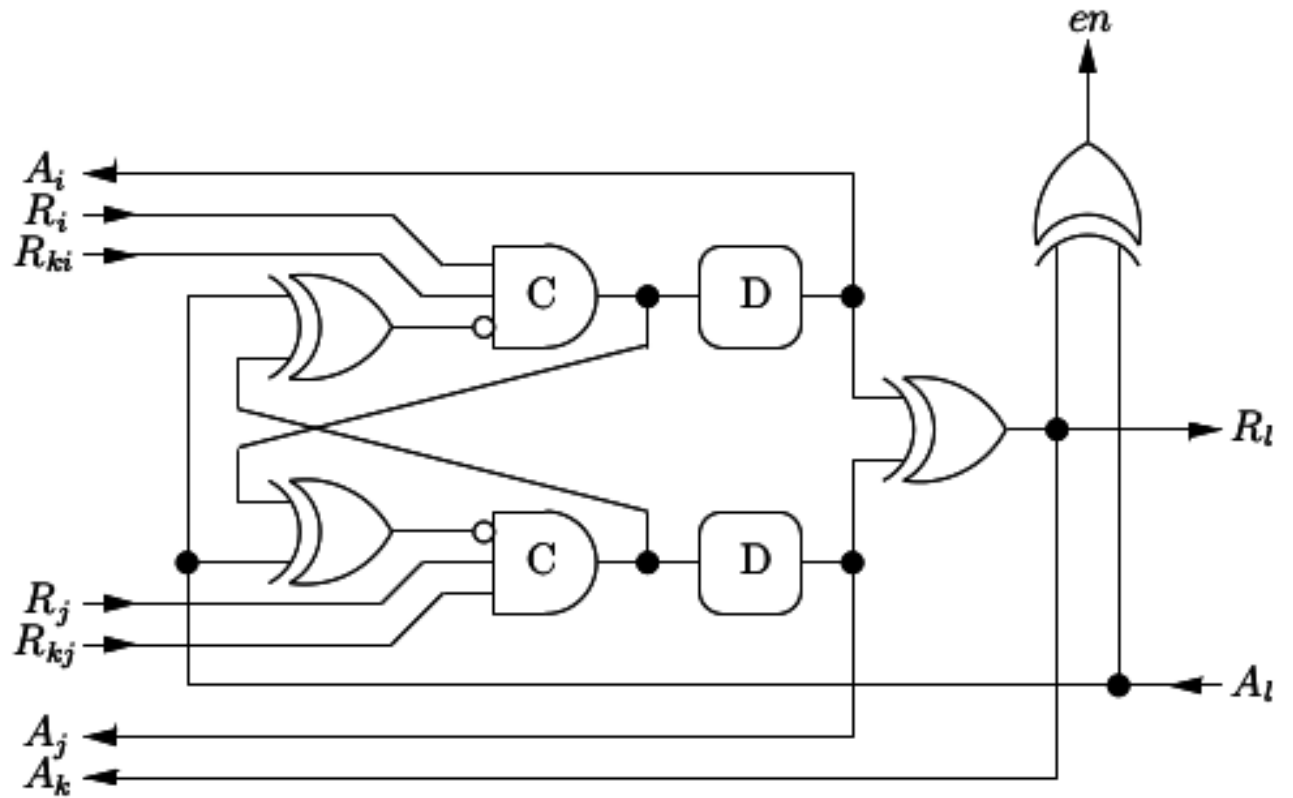


Fig. 4b : *Handshake pour le MUX22 (partie « contrôle »). Le signal en sert à contrôler le latch de sortie.*

DEMUX

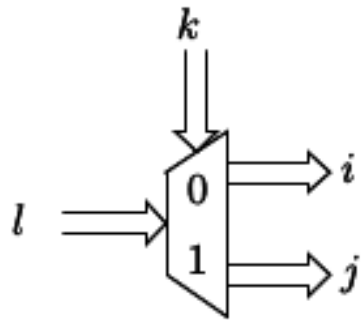


Fig. 5a : *Symbole du DEMUX22 (partie « données »).*

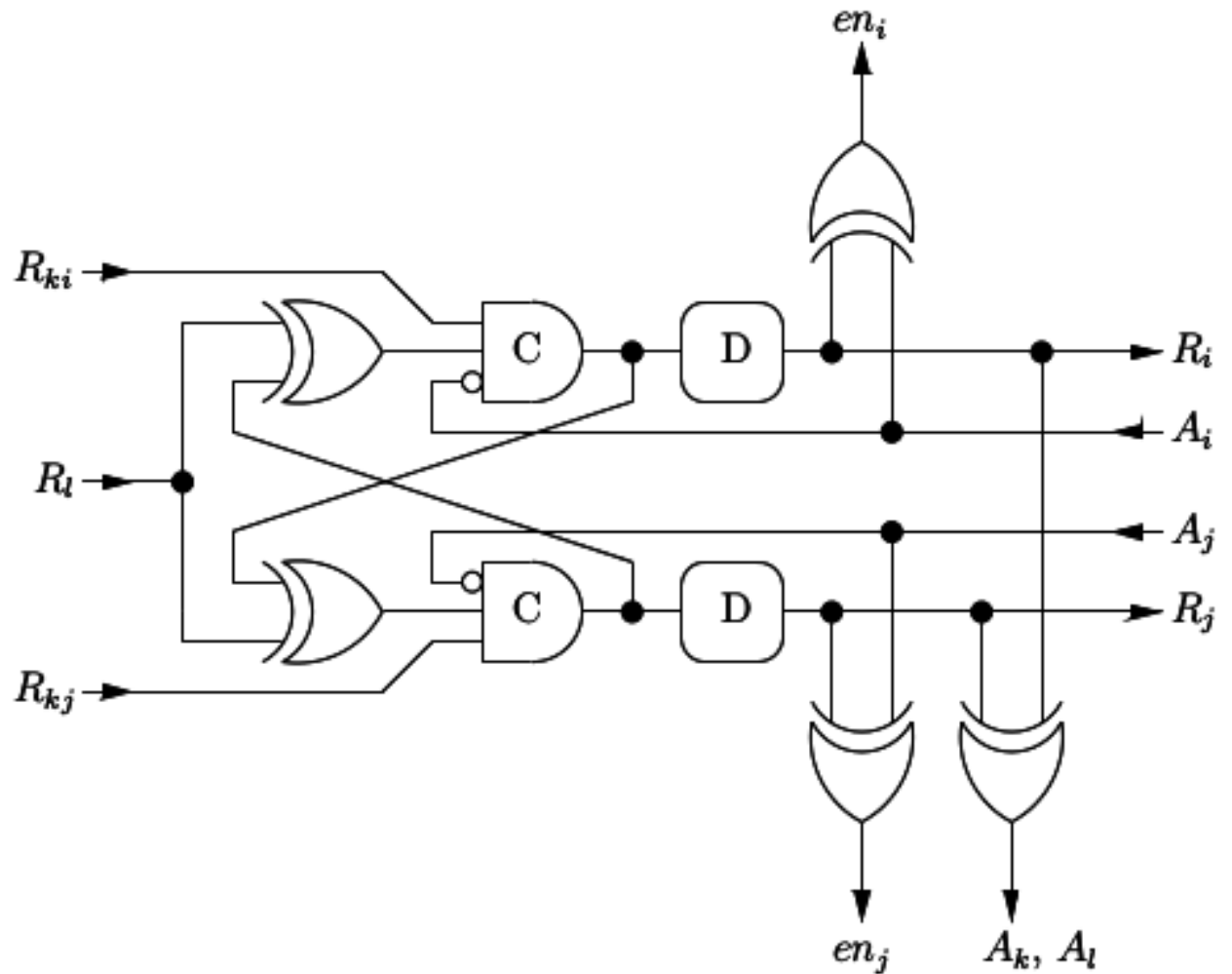


Fig. 5b : *Handshake pour le DEMUX22 (partie « contrôle »). Les signaux en_i et en_j (facultatifs) servent à contrôler les deux latch de sortie.*

Latch avec jeton initial

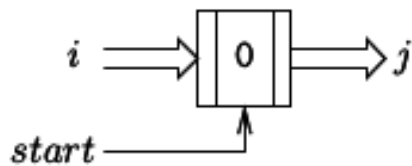


Fig. 6a : *Symbole du LATCH_INIT_0 qui émet initialement un 0 (partie « données »).*

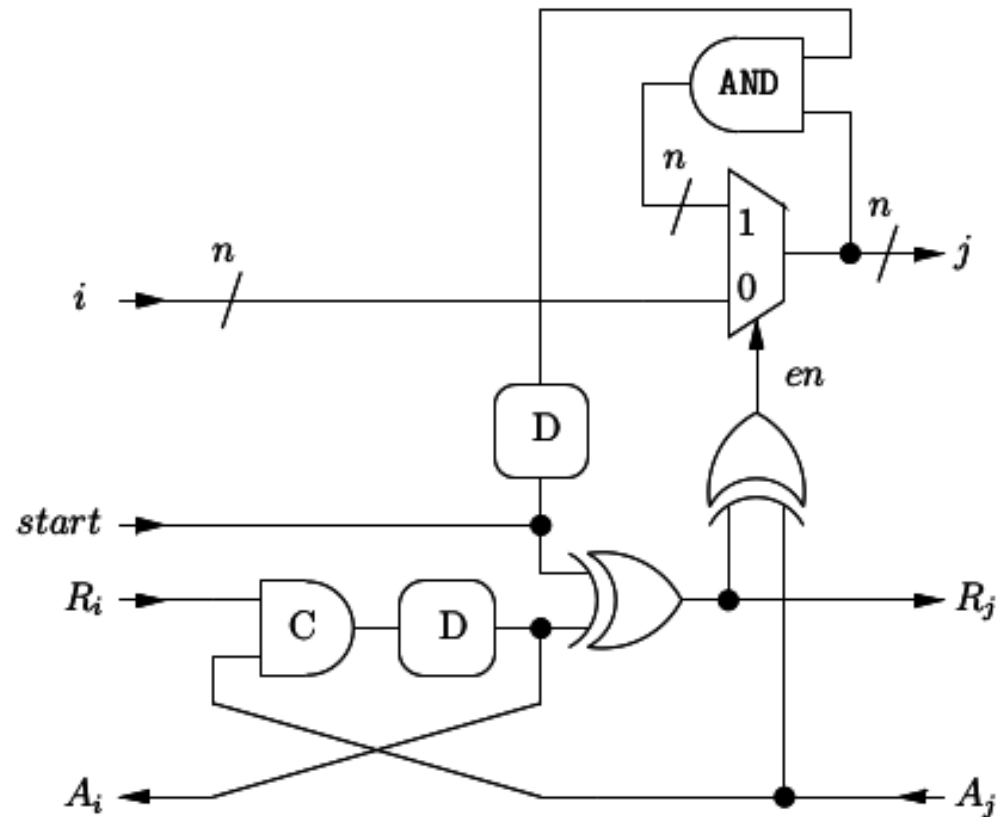
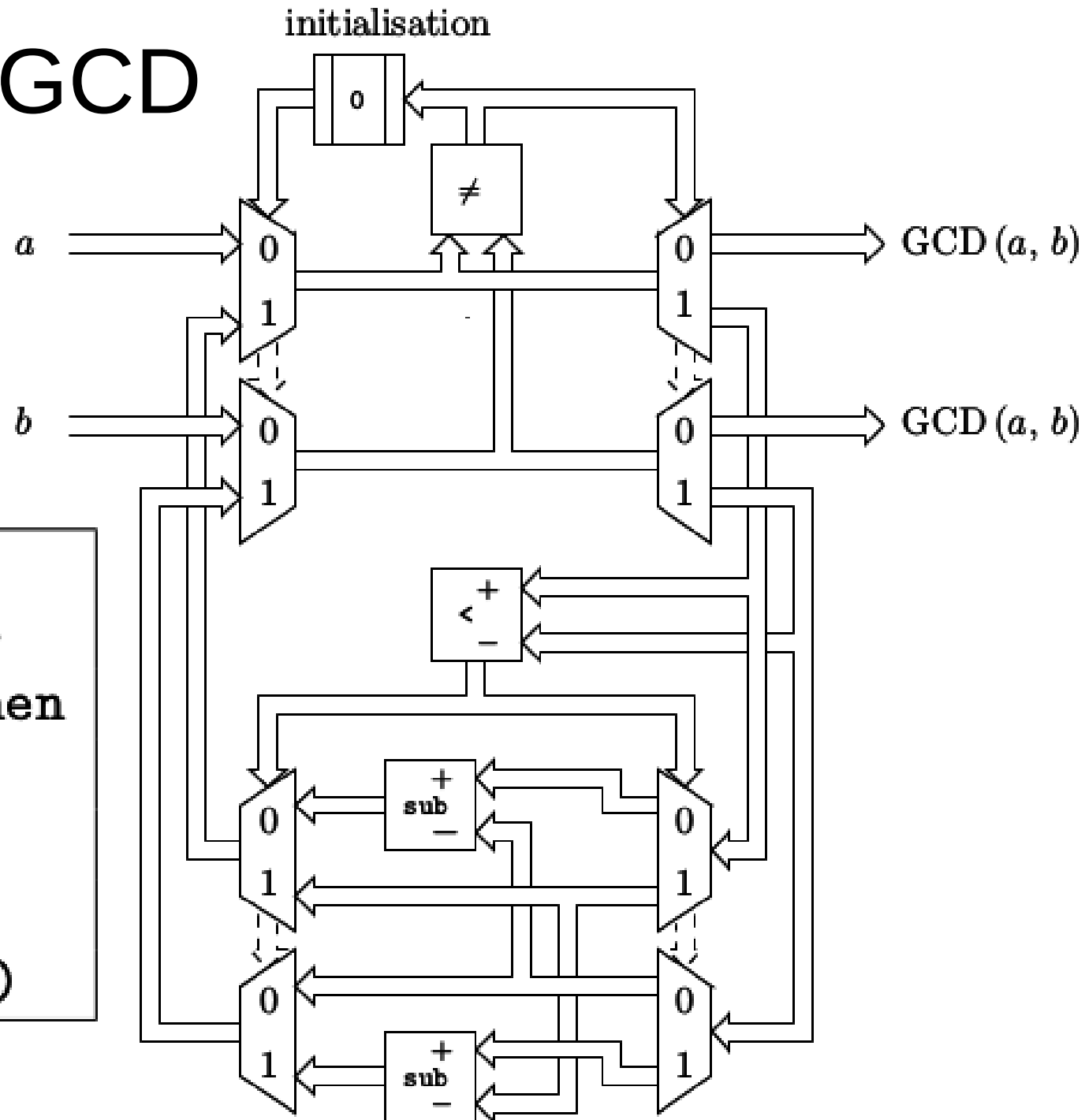


Fig. 6b : *Handshake pour le LATCH_INIT_0 (parties « données » et « contrôle »). Le signal start est actif sur transition montante.*

Algorithme GCD

```

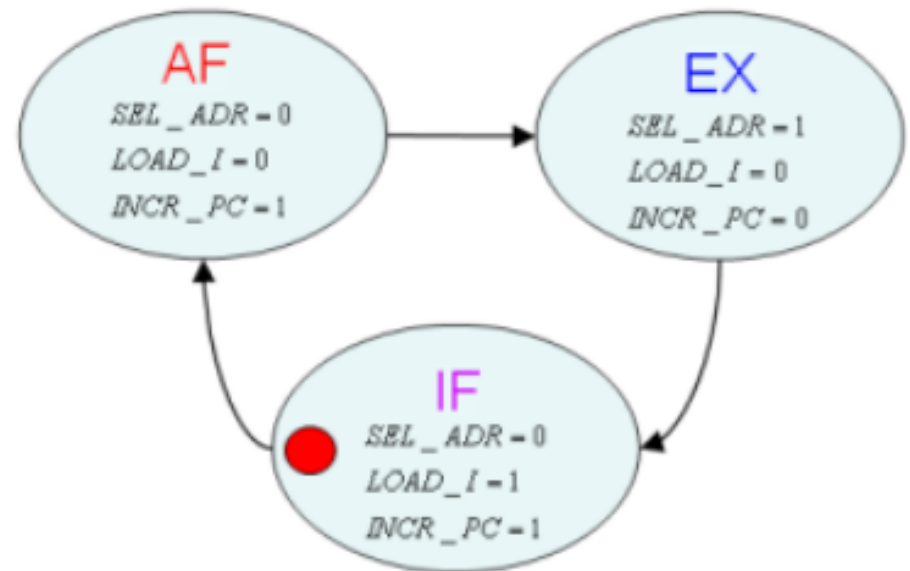
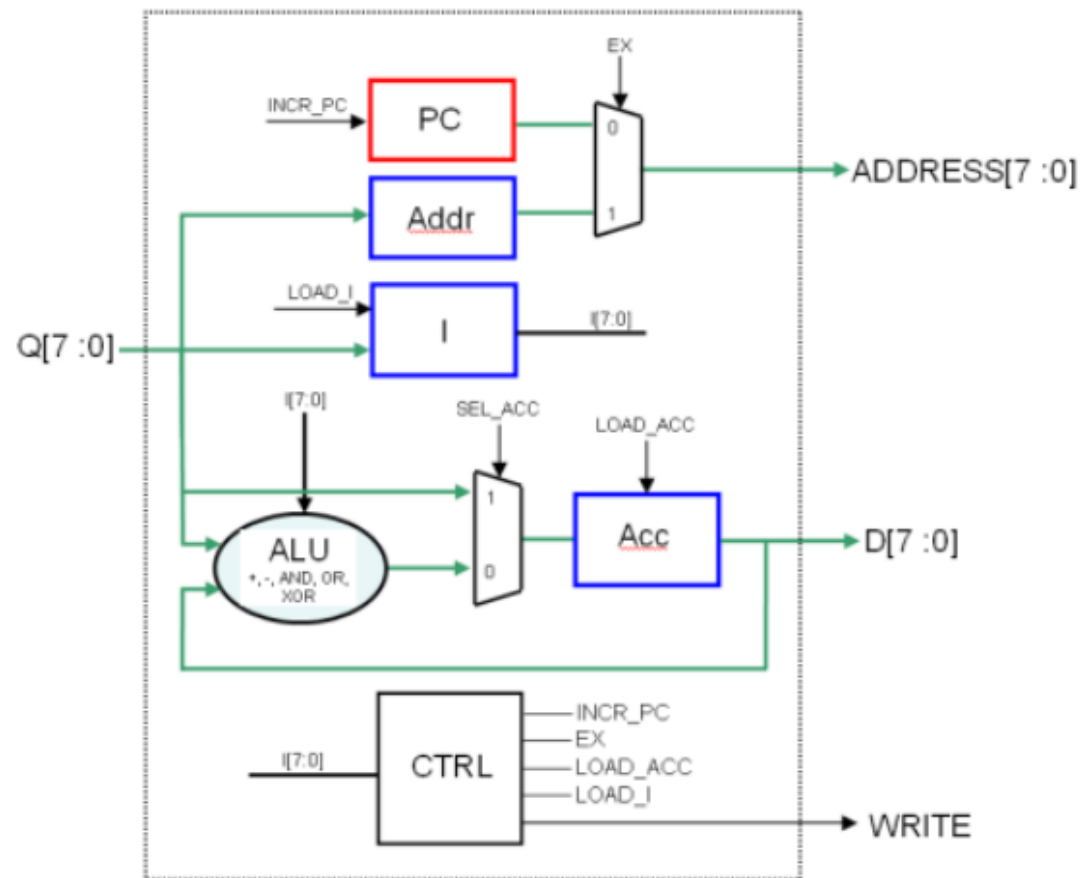
Input :  $a, b > 0$ 
while (  $a \neq b$  ) do
  if (  $a < b$  ) then
     $b \leftarrow b - a$ 
  else
     $a \leftarrow a - b$ 
Output : GCD (  $a, b$  )
  
```



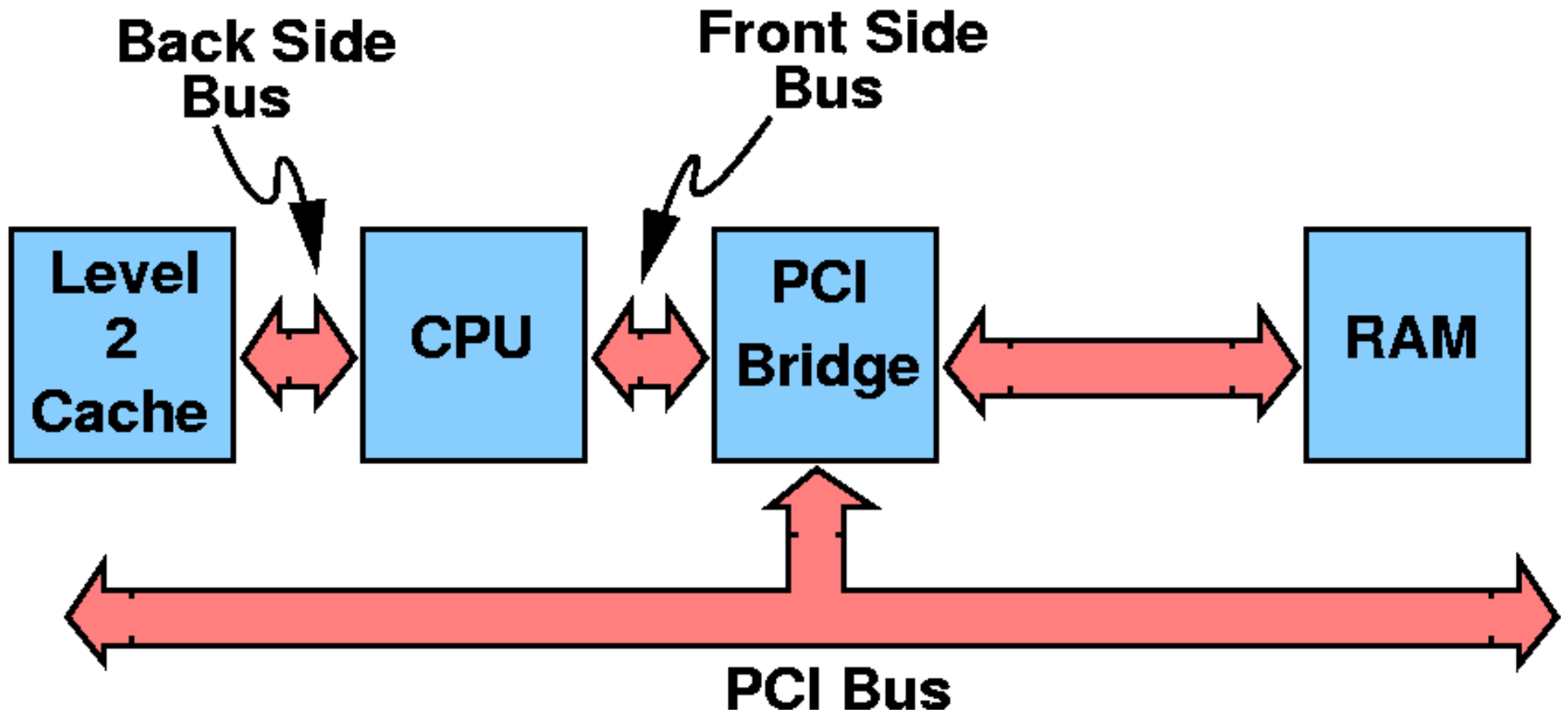
Validation par simulation



Processeur



Processor + bus



Direct Memory Map

- Copy from buffer to buffer
 - Why is it inefficient ?

```
void my_strcpy(char *dest, const char *src)
{
    while(*dest++ = *src++);
}
```

- DMA :
 - Do not use CPU for those deterministic copies

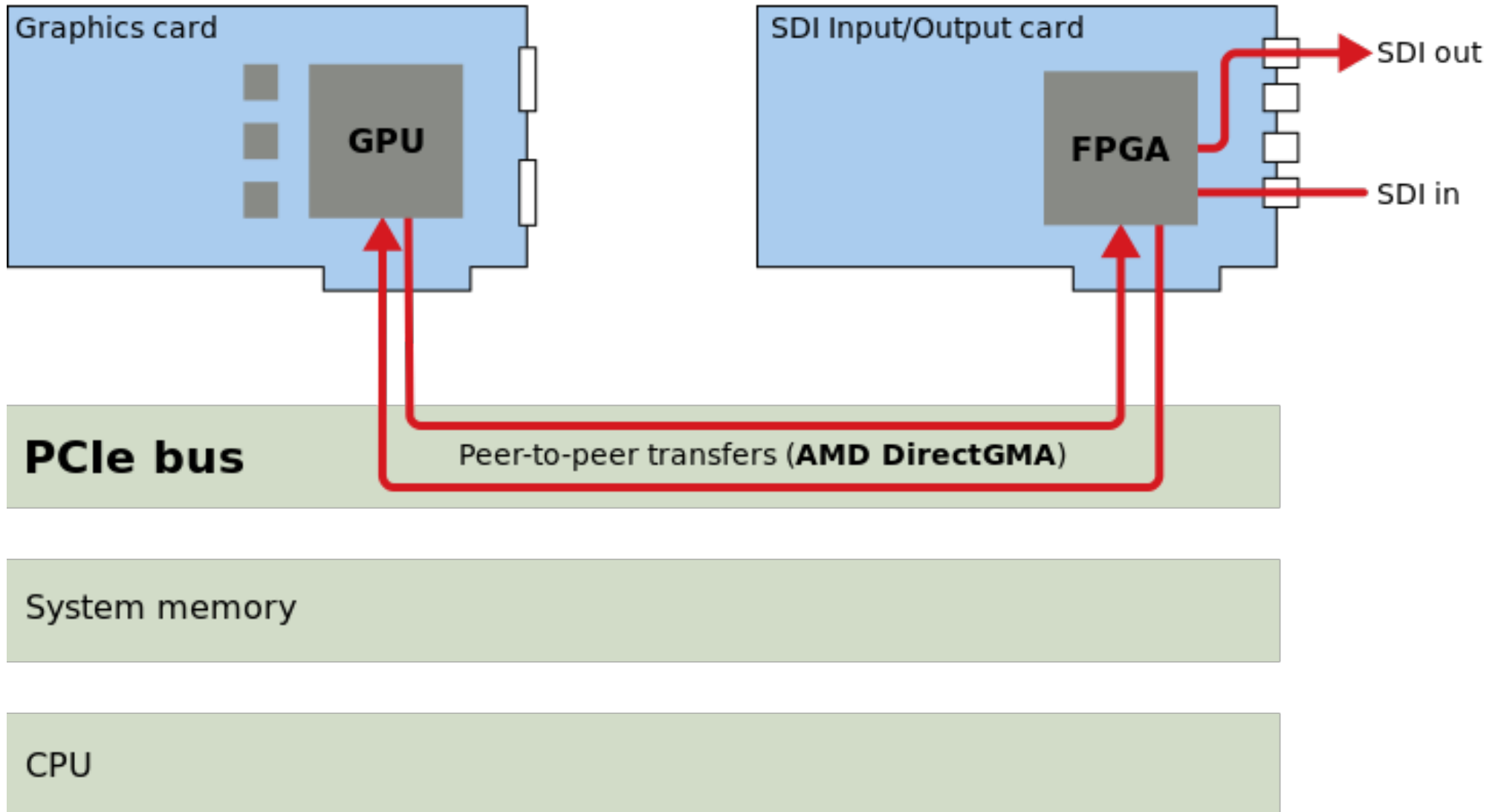
A simple implementation of `strncpy()` might be:

```
char *
strncpy(char *dest, const char *src, size_t n)
{
    size_t i;

    for (i = 0; i < n && src[i] != '\0'; i++)
        dest[i] = src[i];
    for ( ; i < n; i++)
        dest[i] = '\0';

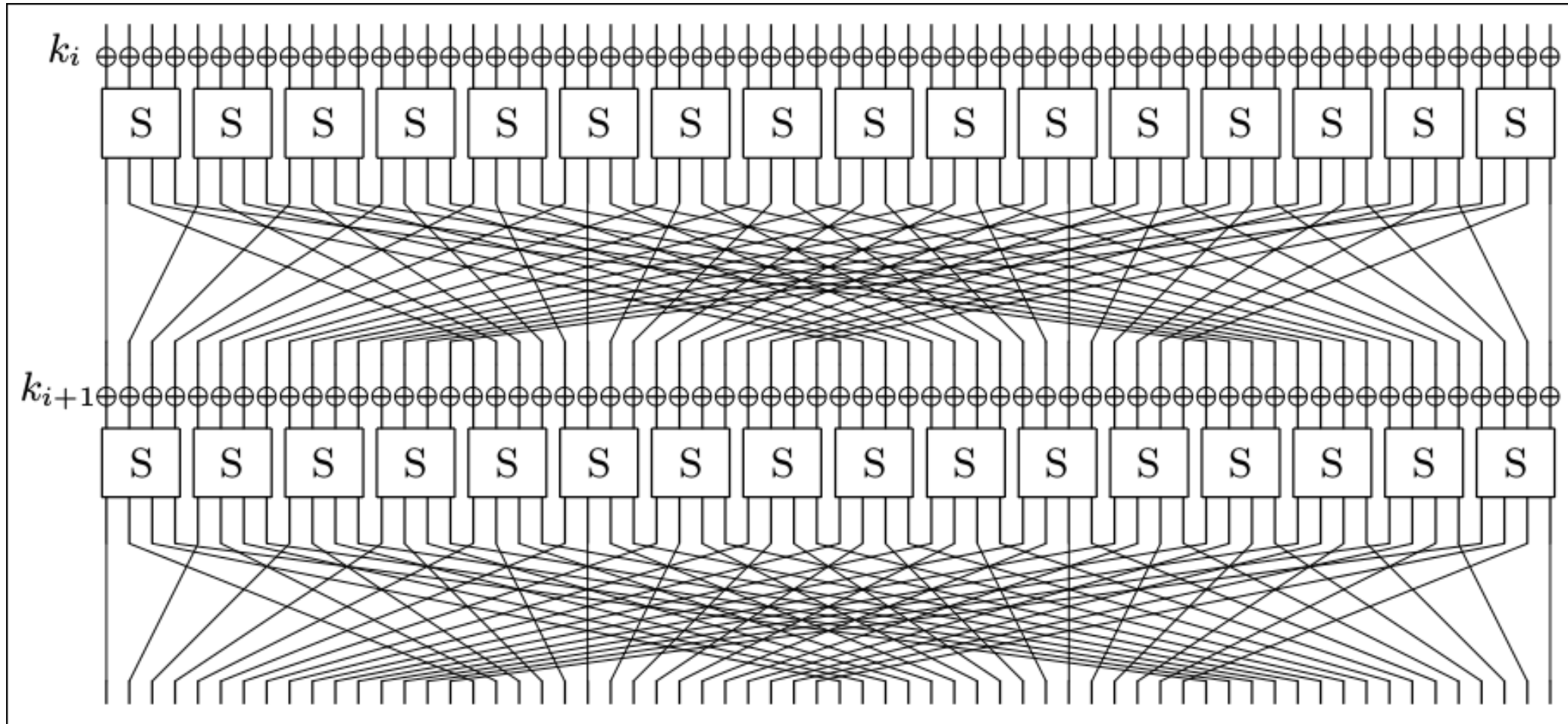
    return dest;
}
```

DMA : exemple



Bitslice implementation

optimiser du logiciel en s'inspirant du matériel



{ 0xc, 0x5, 0x6, 0xb, 0x9, 0x0, 0xa, 0xd, 0x3, 0xe, 0xf, 0x8, 0x4, 0x7, 0x1, 0x2 }

Source : http://lightweightcrypto.org/present/present_ches2007.pdf

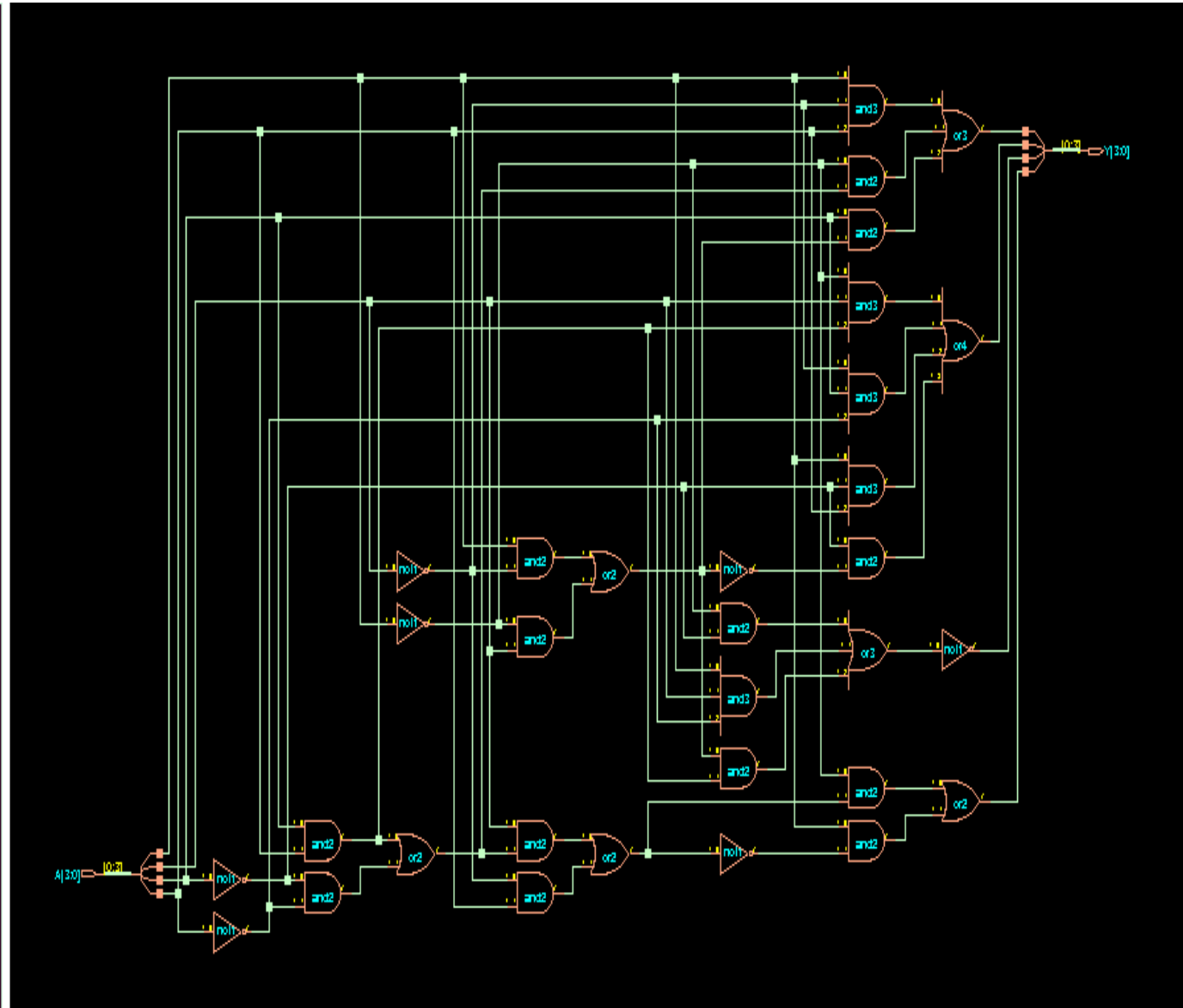
=> 49 gates

With library:

- IV (inverter): 1 A.U. area
- AN (and), IO (inclusive or), EO (exclusive or): 2 A.U. area

```
module present_sbox( Y, A );
  input  [3:0] A;
  output [3:0] Y;
  reg    [3:0] Y;

  always @(A)
  case (A)
    4'h0 : Y <= 4'hc;
    4'h1 : Y <= 4'h5;
    4'h2 : Y <= 4'h6;
    4'h3 : Y <= 4'hb;
    4'h4 : Y <= 4'h9;
    4'h5 : Y <= 4'h0;
    4'h6 : Y <= 4'ha;
    4'h7 : Y <= 4'hd;
    4'h8 : Y <= 4'h3;
    4'h9 : Y <= 4'he;
    4'ha : Y <= 4'hf;
    4'hb : Y <= 4'h8;
    4'hc : Y <= 4'h4;
    4'hd : Y <= 4'h7;
    4'he : Y <= 4'h1;
    4'hf : Y <= 4'h2;
    default : $display("Bad input");
  endcase
endmodule
```



```
module present_sbox( Y, A );
```

```
input  [3:0] A;  
output [3:0] Y;  
wire   [9:0] T;
```

```
not U0 ( T[0], A[0] );  
xor U1 ( T[1], A[1], A[2] );  
and U2 ( T[2], A[2], T[1] );  
xor U3 ( T[3], A[3], T[2] );  
xor U4 ( Y[0], A[0], T[3] );  
and U5 ( T[4], T[1], T[3] );  
xor U6 ( T[5], Y[0], T[1] );  
xor U7 ( T[6], A[2], T[4] );  
or  U8 ( T[7], A[0], T[6] );  
xor U9 ( Y[1], T[5], T[7] );  
xor UA ( T[8], T[0], T[6] );  
xor UB ( Y[3], Y[1], T[8] );  
or  UC ( T[9], T[5], T[8] );  
xor UD ( Y[2], T[3], T[9] );
```

```
endmodule
```

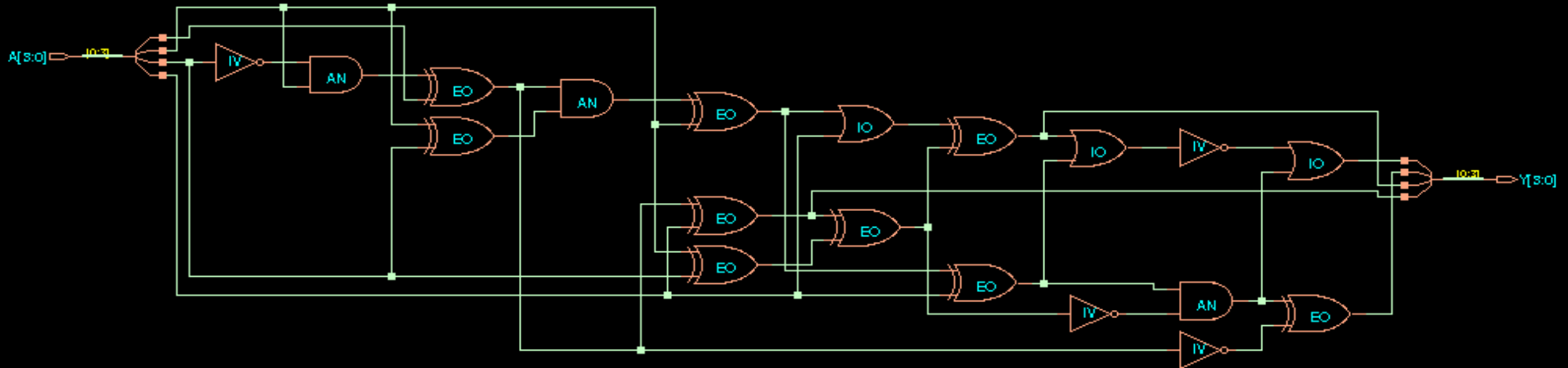
=> 14 gates

With library:

- IV (inverter): 1 A.U. area

- AN (and), IO (inclusive or), EO (exclusive or): 2 A.U. area

=> 19 gates




```
A = [
  0b1010101010101010,
  0b1100110011001100,
  0b1111000011110000,
  0b1111111100000000 ]
```

```
T = [0b0000000000000000]*10
Y = [0b0000000000000000]*4
```

<https://eprint.iacr.org/2011/475.pdf>

```
T[0]=A[0]^0xffff
T[1]=A[1]^A[2]
T[2]=A[2]&T[1]
T[3]=A[3]^T[2]
Y[0]=A[0]^T[3]
T[4]=T[1]&T[3]
T[5]=Y[0]^T[1]
T[6]=A[2]^T[4]
T[7]=A[0]|T[6]
Y[1]=T[5]^T[7]
T[8]=T[0]^T[6]
Y[3]=Y[1]^T[8]
T[9]=T[5]|T[8]
Y[2]=T[3]^T[9]
```

Parallel execution of the 16 sboxes

```
void pLayer(word *X, word *Y) {
  X[ 0] = Y[ 0], X[ 1] = Y[ 4], X[ 2] = Y[ 8], X[ 3] = Y[12];
  X[ 4] = Y[16], X[ 5] = Y[20], X[ 6] = Y[24], X[ 7] = Y[28];
  X[ 8] = Y[32], X[ 9] = Y[36], X[10] = Y[40], X[11] = Y[44];
  X[12] = Y[48], X[13] = Y[52], X[14] = Y[56], X[15] = Y[60];
  X[16] = Y[ 1], X[17] = Y[ 5], X[18] = Y[ 9], X[19] = Y[13];
  X[20] = Y[17], X[21] = Y[21], X[22] = Y[25], X[23] = Y[29];
  X[24] = Y[33], X[25] = Y[37], X[26] = Y[41], X[27] = Y[45];
  X[28] = Y[49], X[29] = Y[53], X[30] = Y[57], X[31] = Y[61];
  X[32] = Y[ 2], X[33] = Y[ 6], X[34] = Y[10], X[35] = Y[14];
  X[36] = Y[18], X[37] = Y[22], X[38] = Y[26], X[39] = Y[30];
  X[40] = Y[34], X[41] = Y[38], X[42] = Y[42], X[43] = Y[46];
  X[44] = Y[50], X[45] = Y[54], X[46] = Y[58], X[47] = Y[62];
  X[48] = Y[ 3], X[49] = Y[ 7], X[50] = Y[11], X[51] = Y[15];
  X[52] = Y[19], X[53] = Y[23], X[54] = Y[27], X[55] = Y[31];
  X[56] = Y[35], X[57] = Y[39], X[58] = Y[43], X[59] = Y[47];
  X[60] = Y[51], X[61] = Y[55], X[62] = Y[59], X[63] = Y[63];
}
```