



# FPGA

Créer du matériel en programmant

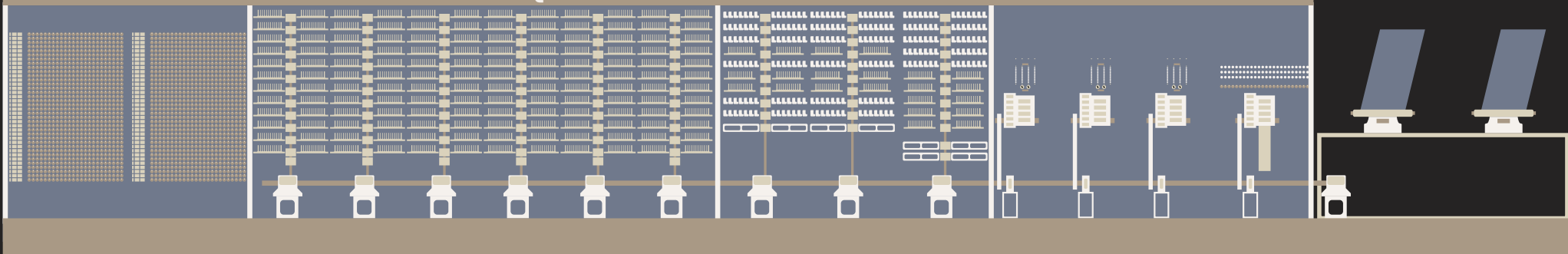
# SOMMAIRE

- Il était une fois : les CPU
- C'est quoi un FPGA ?
- Comment programmer un FPGA ?
- Un exemple : SimpleVGA
- Déboguer un circuit
- Et maintenant ?

**IL ÉTAIT UNE FOIS : LES CPU**

15,5 mètres

## IBM AUTOMATIC SEQUENCE CONTROLLED CALCULATOR



constantes

mémoire de données

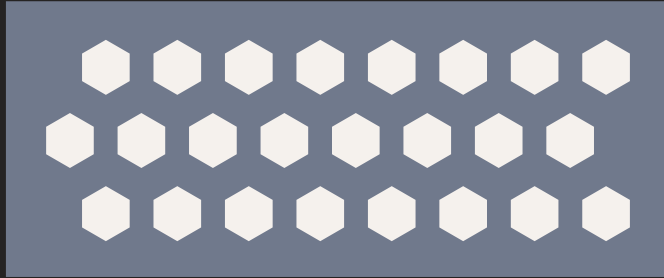
unité de traitement

unité de contrôle  
mémoire d'instructions

entrées/sorties

# HARVARD MARK I, 1944

MÉMOIRE



INSTRUCTIONS



UNITÉ DE CONTRÔLE



DONNÉES



UNITÉ DE TRAITEMENT

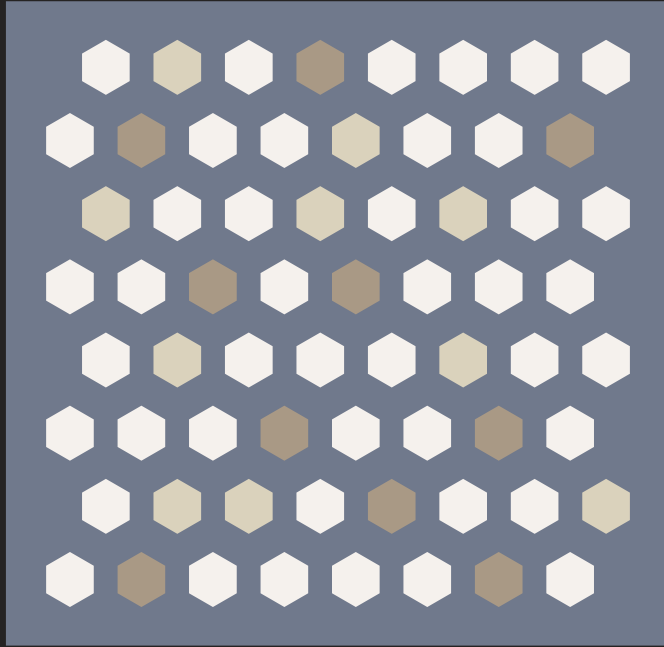


ENTRÉES-SORTIES

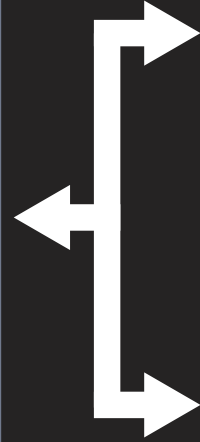


ARCHITECTURE HARVARD

MÉMOIRE



INSTRUCTIONS



DONNÉES

UNITÉ DE CONTRÔLE



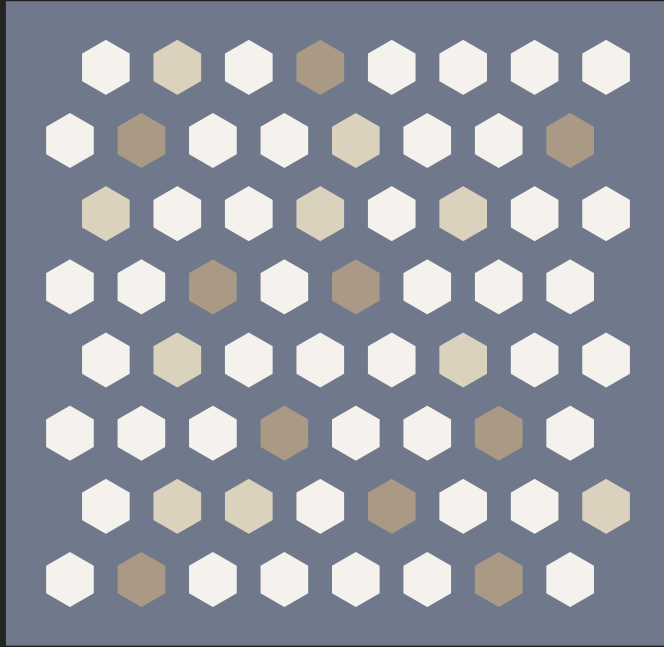
UNITÉ DE TRAITEMENT

ENTRÉES-SORTIES



ARCHITECTURE DE VON NEUMANN

MÉMOIRE



UNITÉ DE CONTRÔLE



UNITÉ DE TRAITEMENT



UNITÉ DE GESTION DES ENTRÉES-SORTIES

ENTRÉES-SORTIES



ÉVITONS LES PROCESSEURS PASSE-PLAT !

MÉMOIRE

CACHE

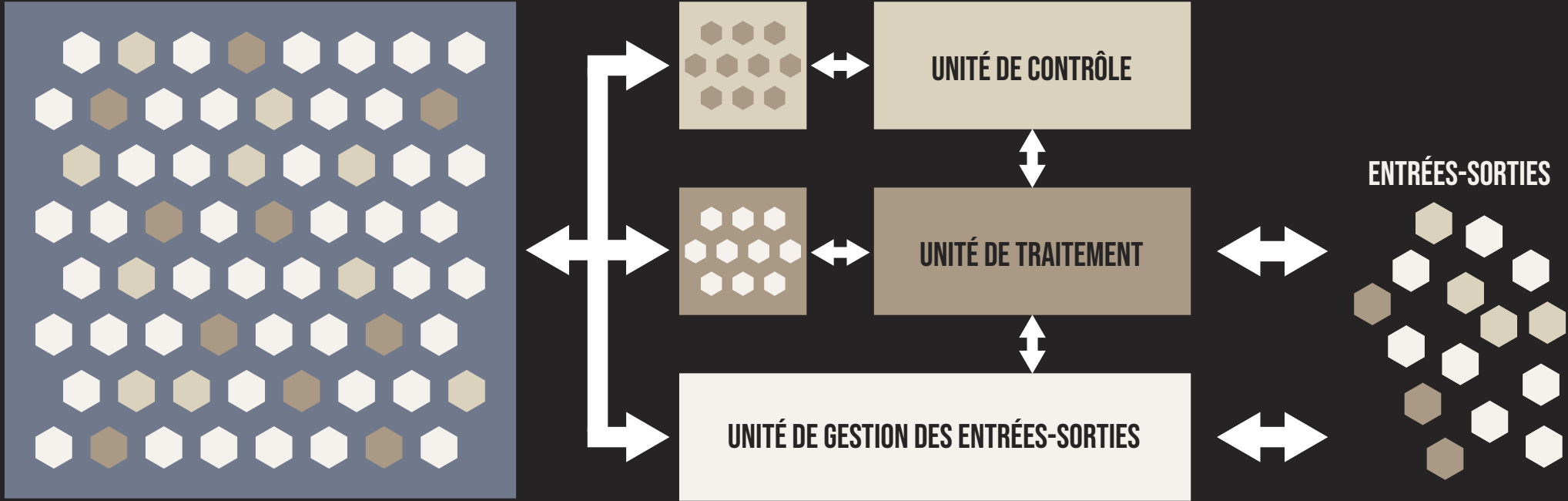
UNITÉ DE CONTRÔLE

UNITÉ DE TRAITEMENT

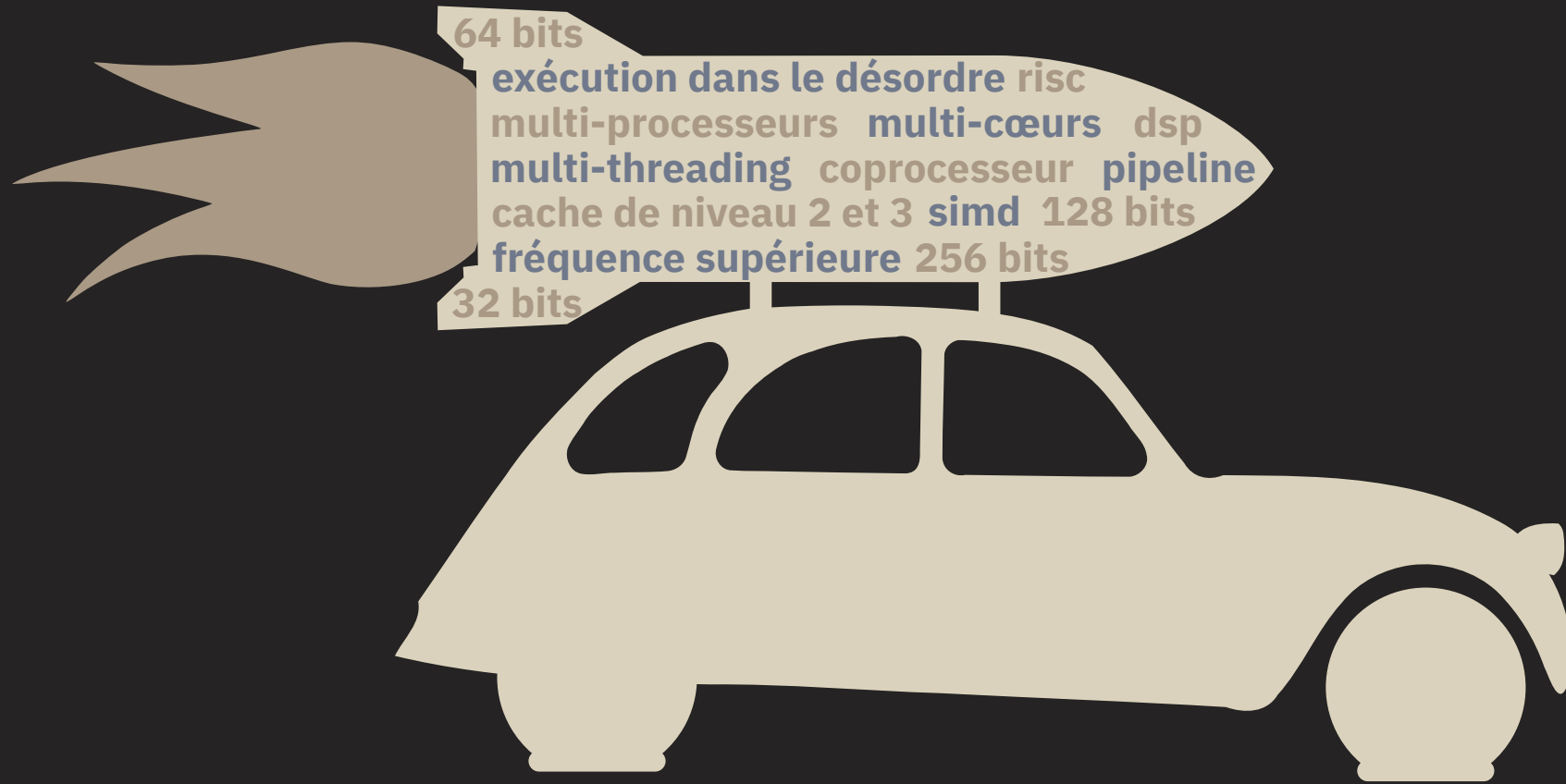
UNITÉ DE GESTION DES ENTRÉES-SORTIES

ENTRÉES-SORTIES

SOULAGEONS LE BUS DE DONNÉES

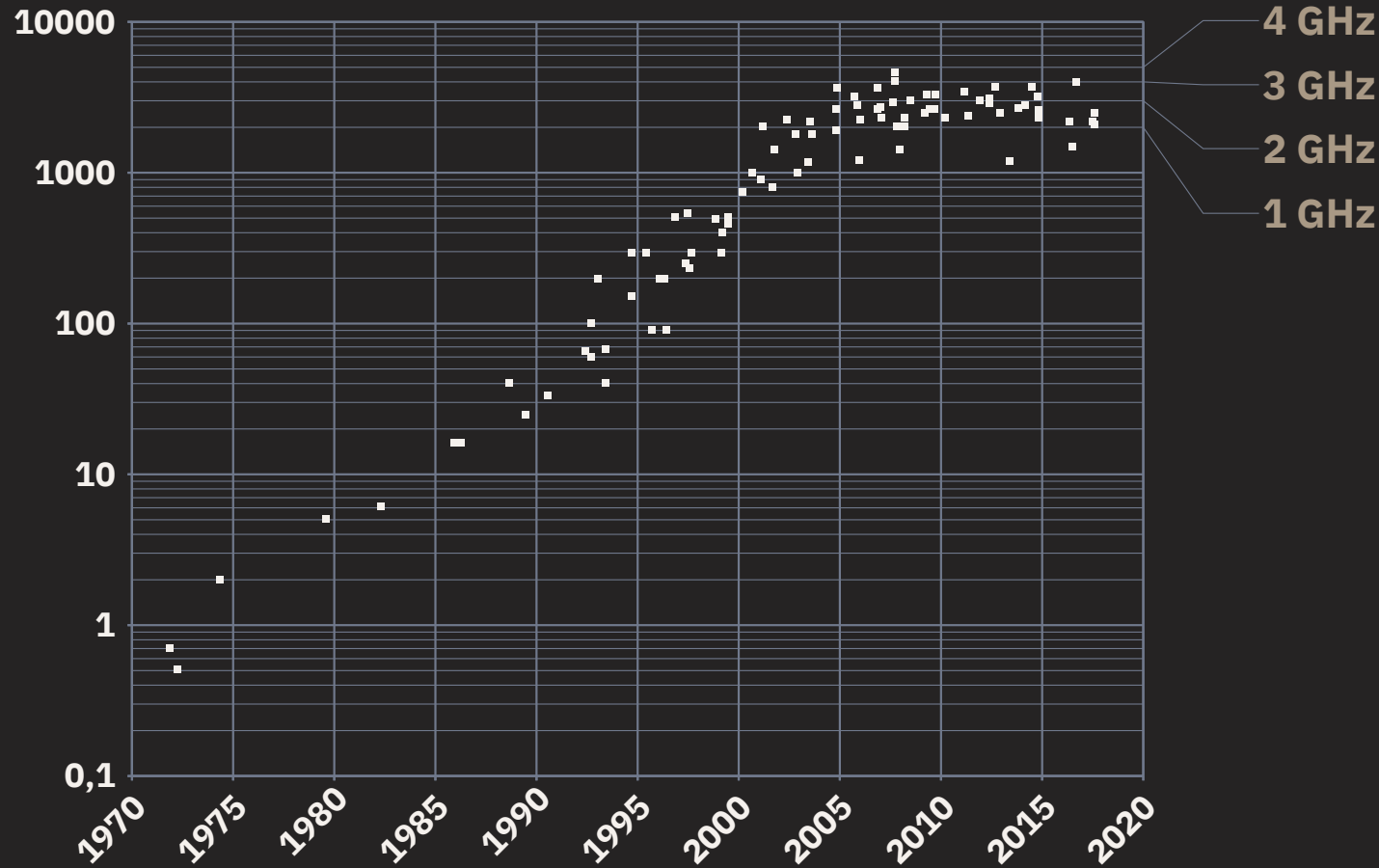




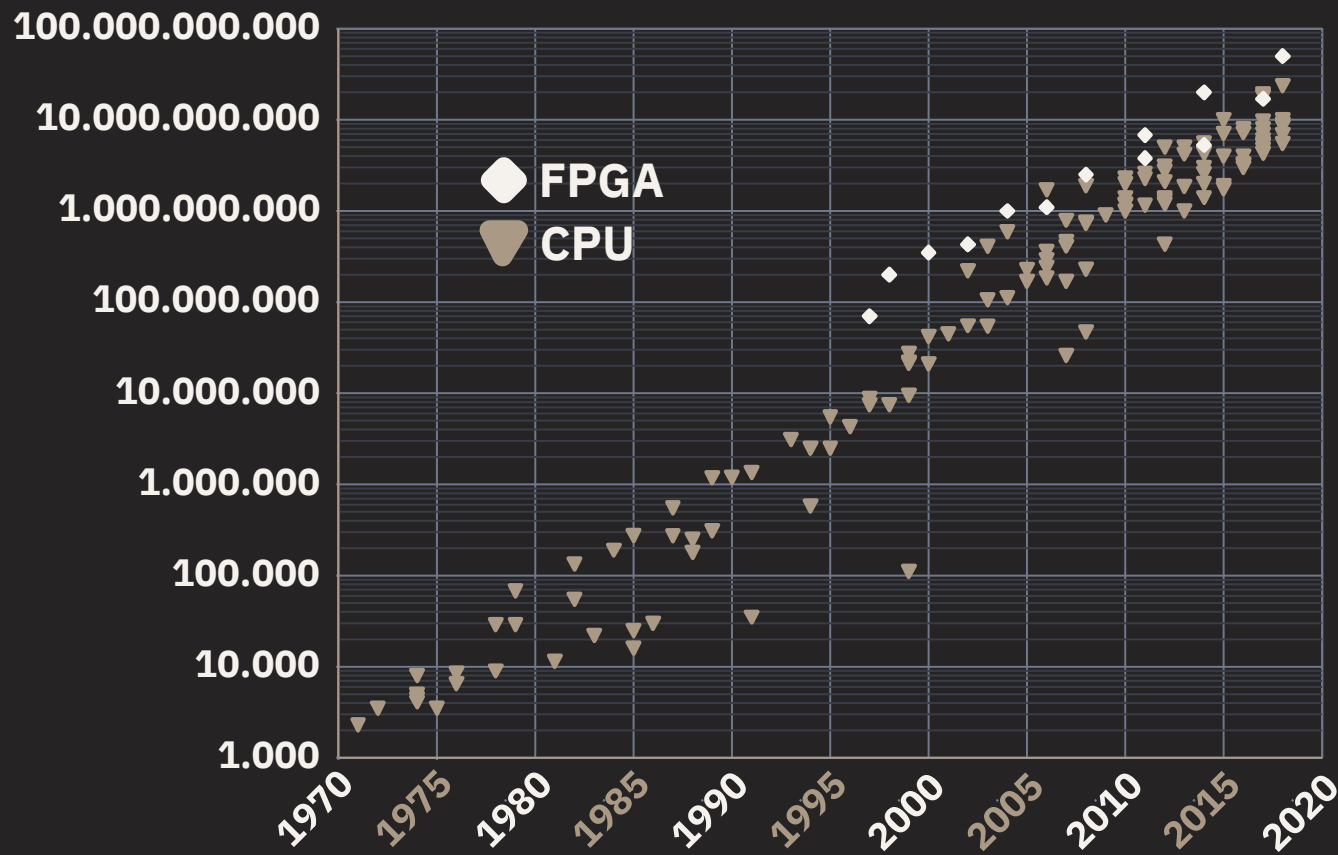


ACCÉLÉRER UNE 2CV N'EN FAIT PAS UNE FUSÉE

Fréquence des microprocesseurs en MHz

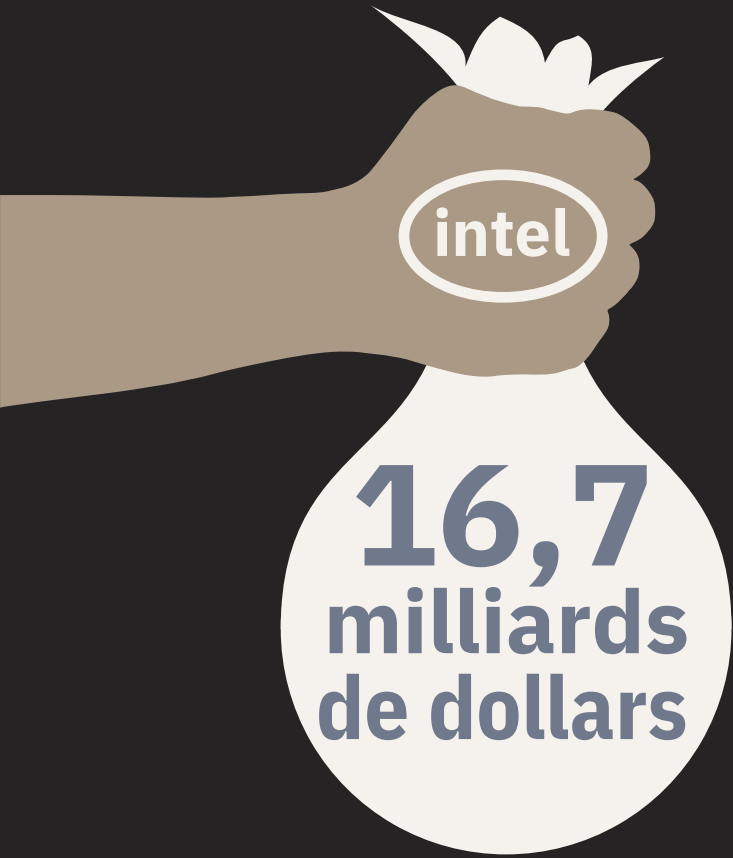


LES FRÉQUENCES STAGNENT DEPUIS 2005



Source: Transistor count, 30/05/2019  
[https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count)

LA LOI DE MOORE A PERDURÉ JUSQU'EN 2017



**ALTERA**



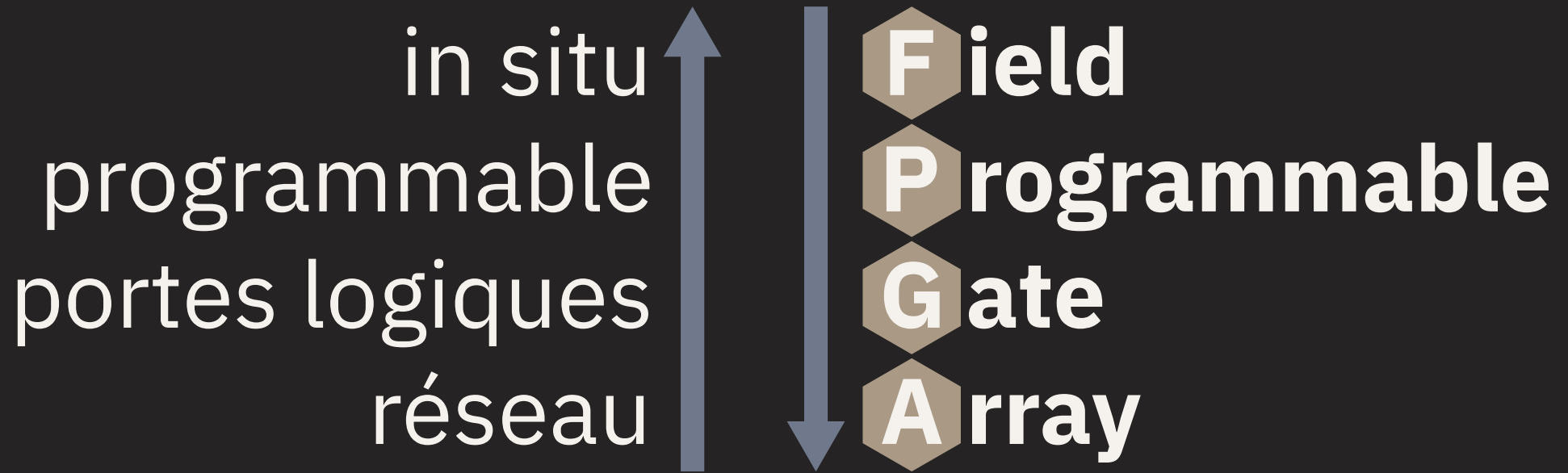
**EN 2015 INTEL ABSORBE ALTERA POUR 16,7 MILLIARDS \$**



**C'EST QUOI UN FPGA ?**

in situ  
programmable  
portes logiques  
réseau

**F**ield  
**P**rogrammable  
**G**ate  
**A**rray



The diagram illustrates the components of an FPGA. On the left, the text 'in situ programmable portes logiques réseau' is arranged vertically. To its right is a blue upward-pointing arrow. Further right is a blue downward-pointing arrow. To the right of the downward arrow is a vertical stack of four brown hexagons, each containing a letter: 'F', 'P', 'G', and 'A'. To the right of these hexagons are the words 'ield', 'rogrammable', 'ate', and 'rray' respectively, which together with the letters in the hexagons form the words 'Field Programmable Gate Array'.

FPGA



The word 'FPGA' is centered within a blue wavy horizontal band at the bottom of the image.



batterie



interrupteur



condensateur



100kΩ



470Ω



10kΩ



10kΩ



1kΩ



1kΩ

résistances

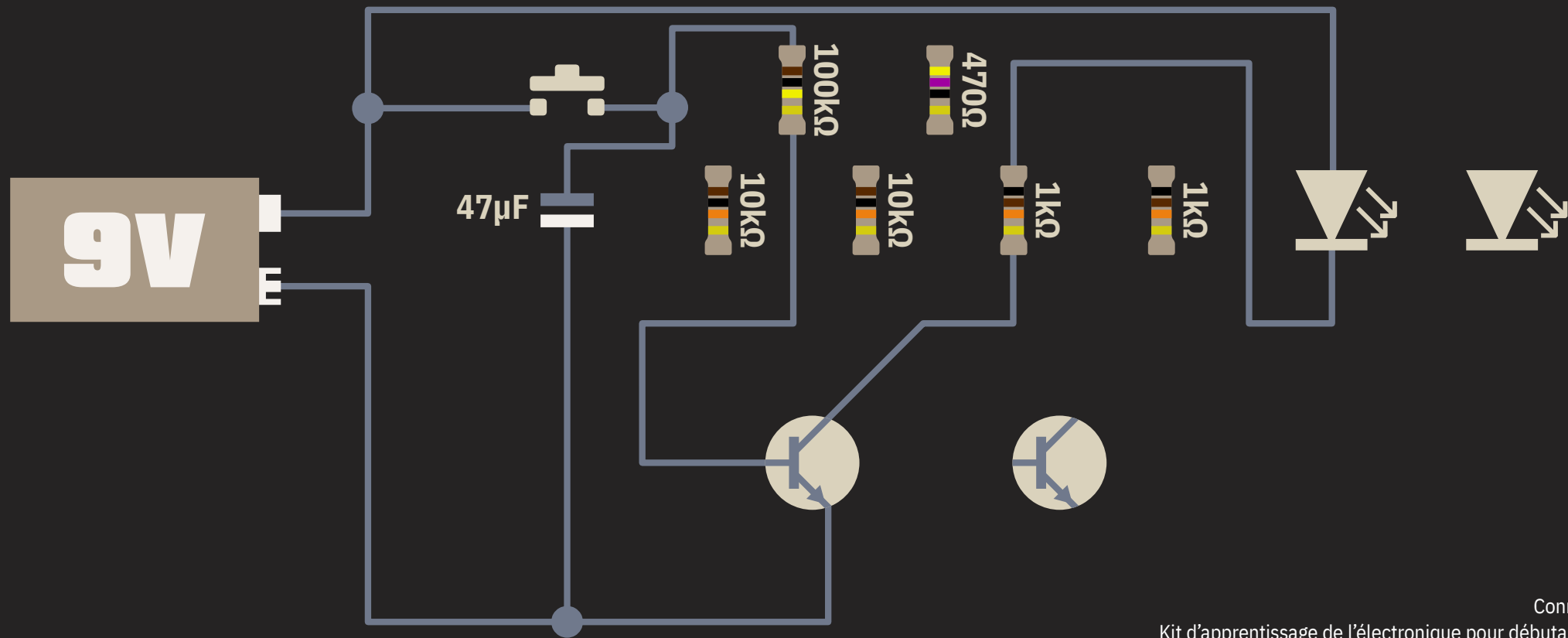


leds



transistors

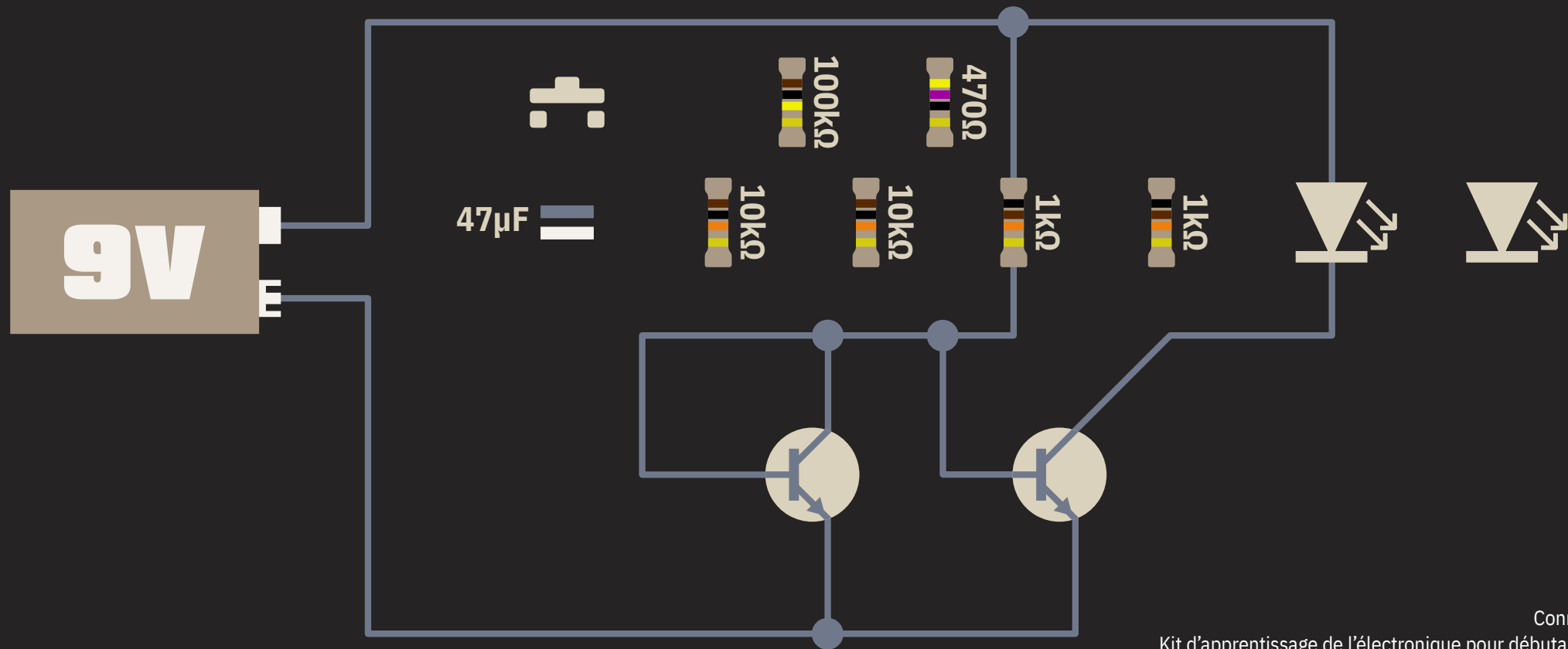
IMAGINEZ DES COMPOSANTS ÉLECTRONIQUES



Conrad  
Kit d'apprentissage de l'électronique pour débutants  
[https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERNPAKET\\_25\\_ELEKTRONIK\\_EXPERIMENTE.pdf](https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERNPAKET_25_ELEKTRONIK_EXPERIMENTE.pdf)

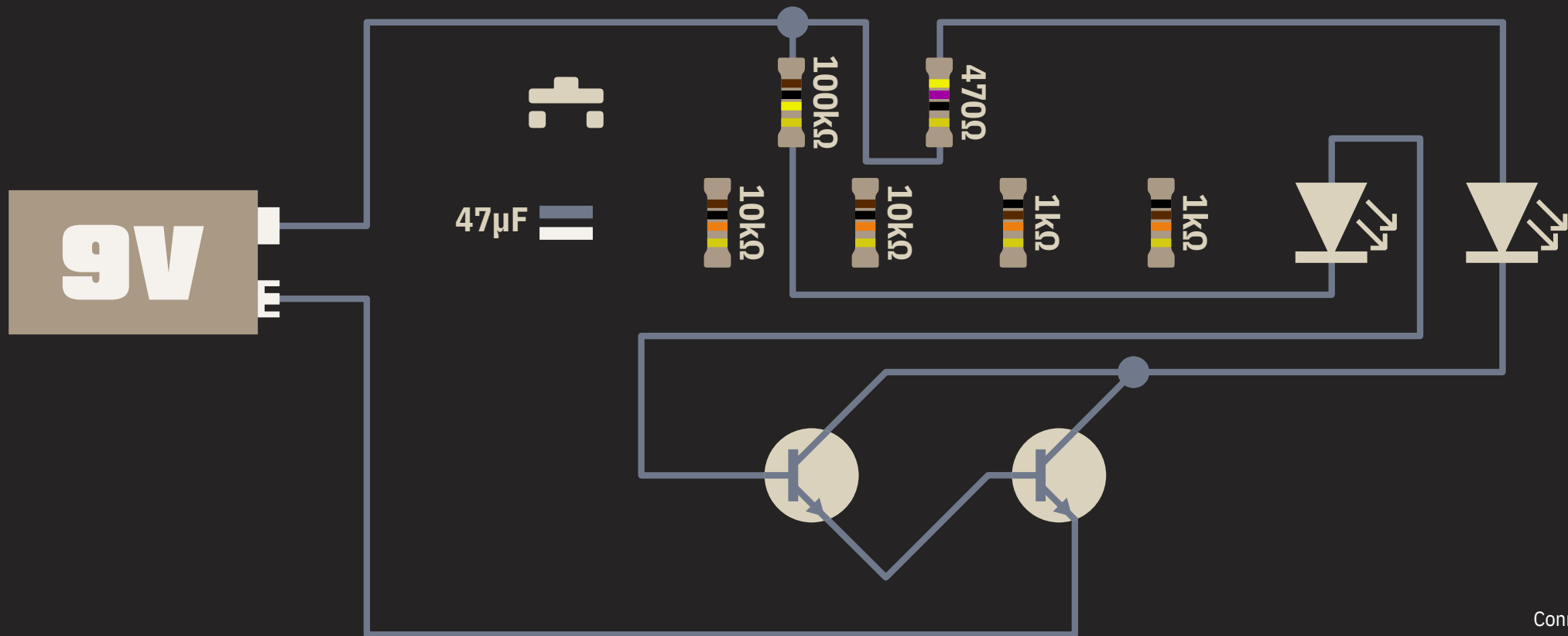
# ILS PEUVENT FORMER UN RETARDATEUR





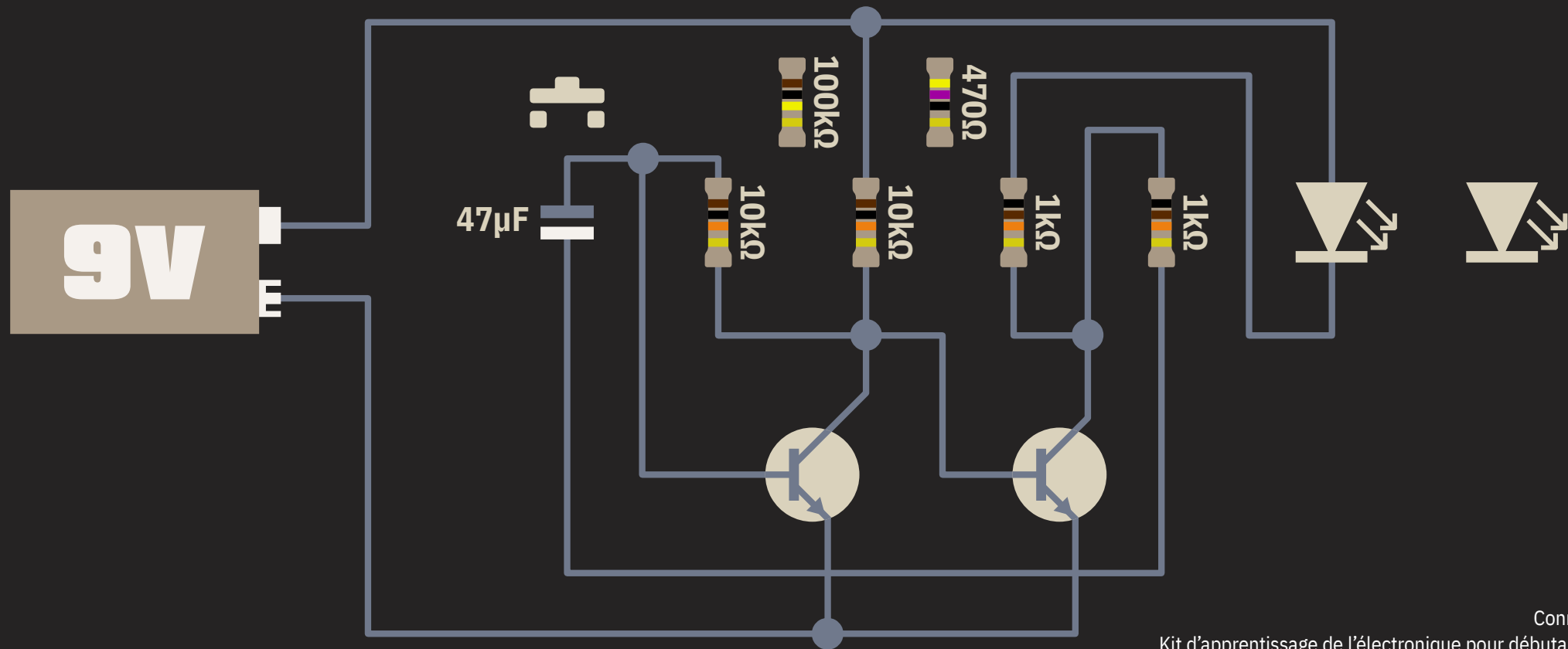
Conrad  
Kit d'apprentissage de l'électronique pour débutants  
[https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERNAKET\\_25\\_ELEKTRONIK\\_EXPERIMENTE.pdf](https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERNAKET_25_ELEKTRONIK_EXPERIMENTE.pdf)

# UN CAPTEUR DE TEMPÉRATURE



Conrad  
Kit d'apprentissage de l'électronique pour débutants  
[https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERNAKET\\_25\\_ELEKTRONIK\\_EXPERIMENTE.pdf](https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERNAKET_25_ELEKTRONIK_EXPERIMENTE.pdf)

# UN CAPTEUR DE LUMIÈRE



Conrad  
Kit d'apprentissage de l'électronique pour débutants  
[https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERPAKET\\_25\\_ELEKTRONIK\\_EXPERIMENTE.pdf](https://produktinfo.conrad.com/datenblaetter/175000-199999/192230-an-01-fr-LERPAKET_25_ELEKTRONIK_EXPERIMENTE.pdf)

# OU ENCORE UNE LED CLIGNOTANTE

# QUELQUES REMARQUES

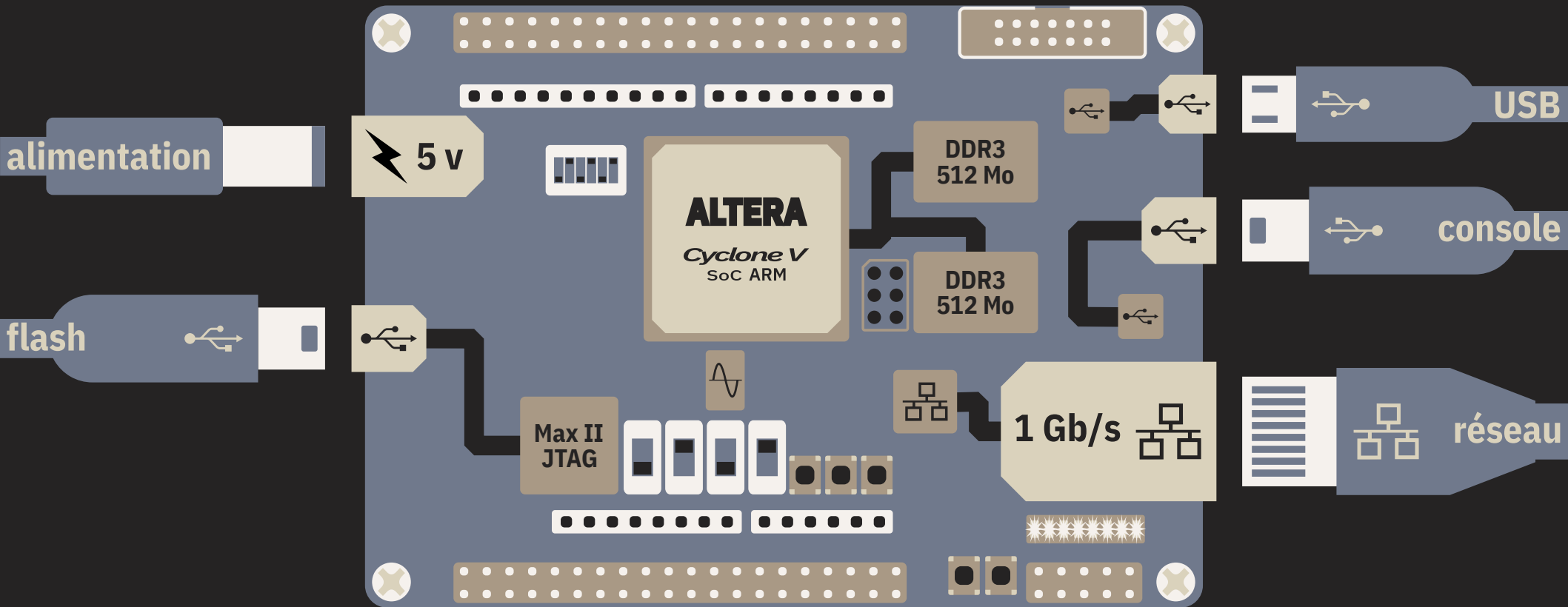
- Le câblage définit le fonctionnement du circuit
- Les composants
  - restent fixes entre les différents schémas
  - ne sont pas tous utilisés pour un schéma donné
- C'est le principe d'un FPGA !

# RÉSEAU DE PORTES LOGIQUES PROGRAMMABLE IN SITU

- Un FPGA a 3 éléments constitutifs
  - des élément logiques (ALM/CLB, mémoire, DSP...)
  - un réseau de pistes
  - une mémoire de configuration
- Le réseau de pistes est figé
  - un FPGA reste un circuit intégré
  - toutes les combinaisons ne sont pas possibles
  - il est reconfigurable à volonté grâce à la mémoire



**LE DEO-NANO-SOC**



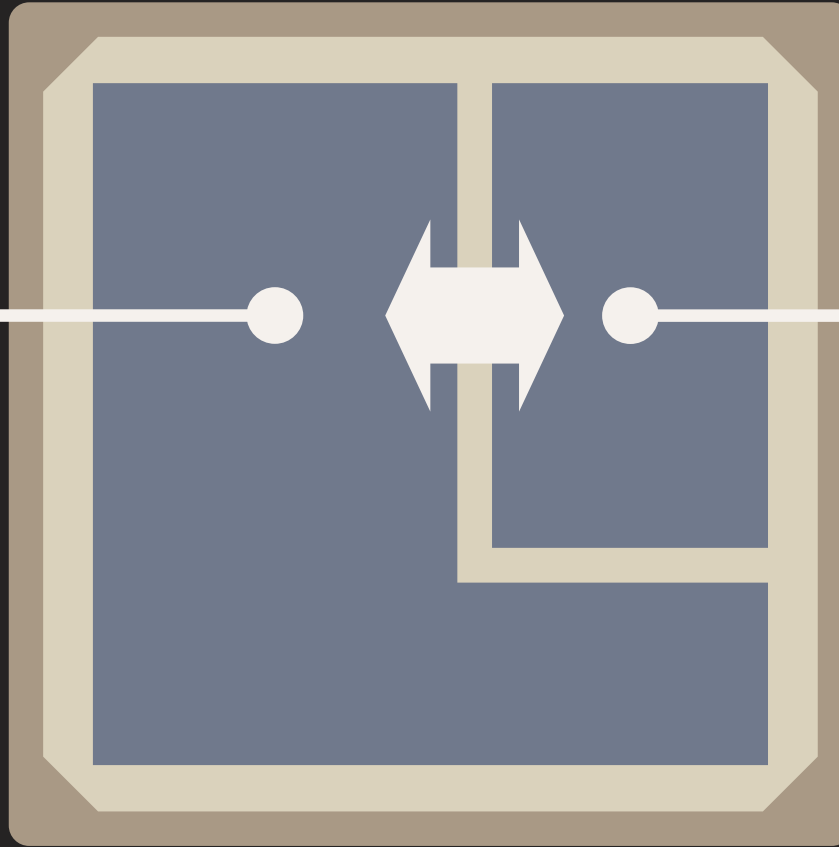
**LA PLATEFORME D'INITIATION DEO-NANO-SOC**

**FPGA**

**15880 CLB**  
**330 Ko RAM**  
**50 MHz**

**ARM Cortex-A9**

**2 cœurs**  
**925 MHz**



**AU CŒUR DU DEO-NANO-SOC : LE CYCLONE V**



50 MHz



25 MHz



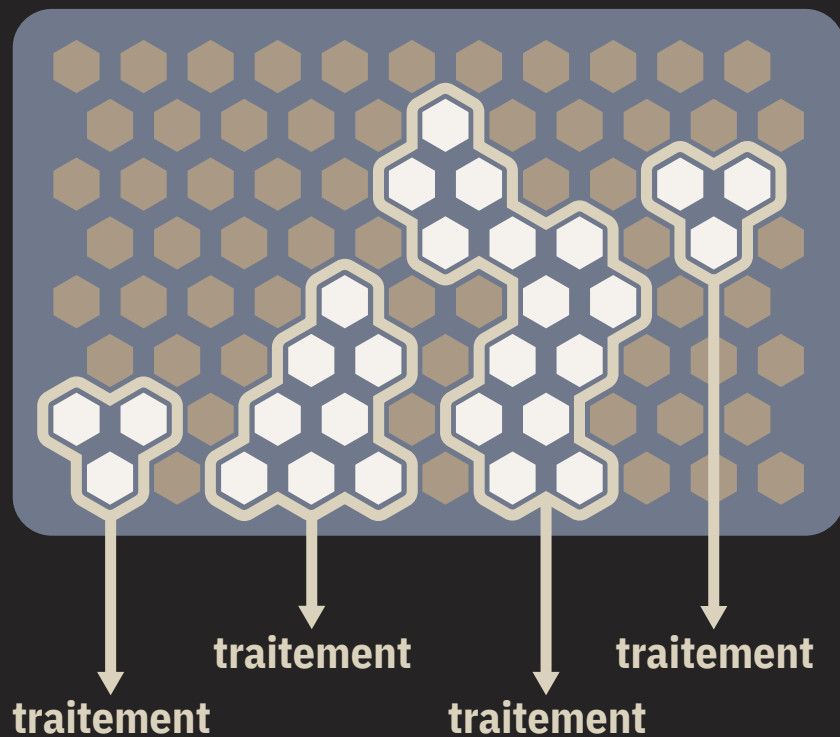
100 MHz



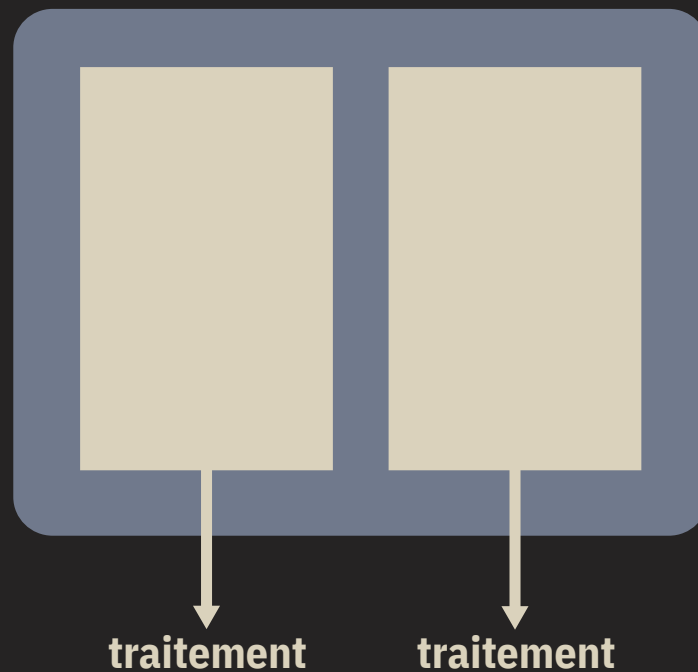
200 MHz

PLL, GÉNÉRER DE NOUVELLES FRÉQUENCES

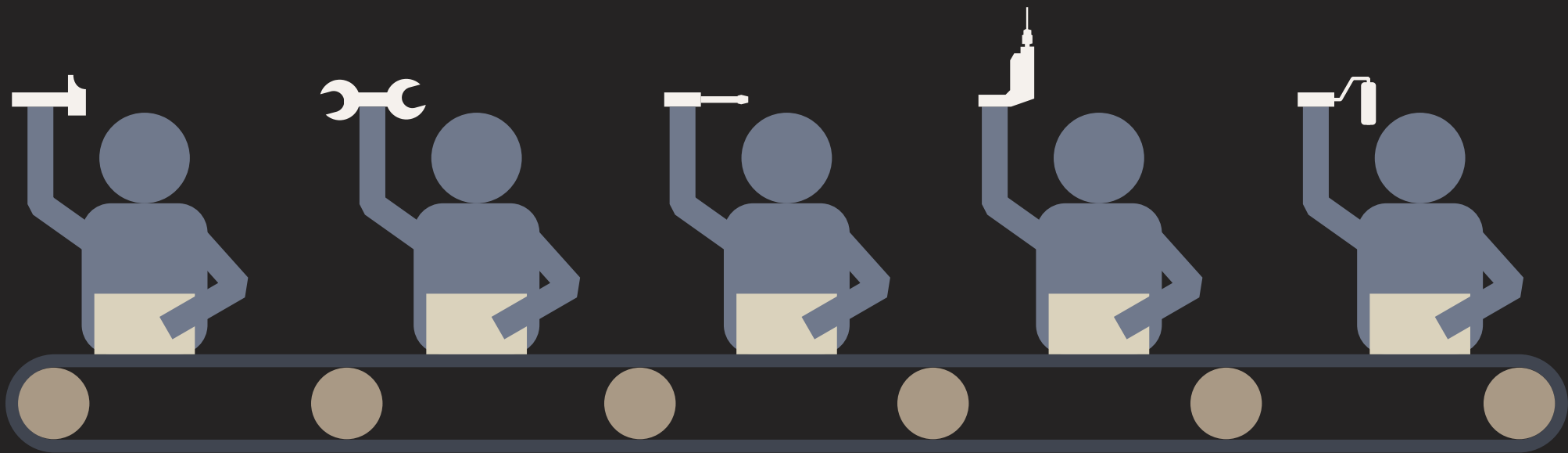
**15880 CLB**



**2 cœurs**



**MODULAIRE ET EXTRÊMEMENT PARALLÈLE**



**PIPELINE, TRAVAIL À LA CHAÎNE**

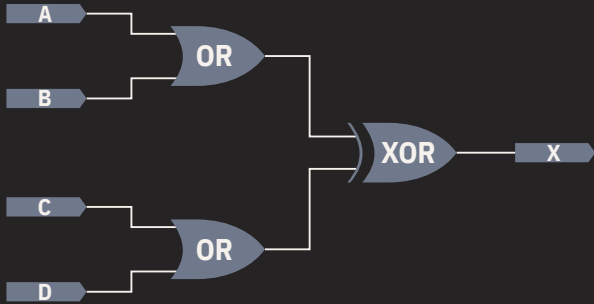


**COMMENT PROGRAMMER UN FPGA ?**

# LES OUTILS DISPONIBLES

- Chaque fabricant a ses propres outils
  - Xilinx → Vivado/ISE
  - Intel → Quartus Prime
  - Lattice → Diamond
  - etc.
- Ils sont **indispensables** pour générer l'image bitstream
  - ils sont gratuits pour les cartes d'initiation
  - les outils libres ne couvrent pas tout le flot de conception

dessiner un schéma



avec des portes logiques,  
des composants électroniques

utiliser un HDL ou langage  
de description de matériel

```
always @(posedge clk) begin
    if (draw) begin
        pixel <= 1'b1;
    end else begin
        pixel <= 1'b0;
    end
end
```

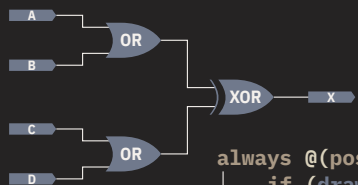
pour décrire le comportement  
du circuit souhaité

utiliser OpenCL

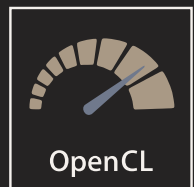


pour utiliser un langage  
de haut niveau

# 3 FAÇONS DE PROGRAMMER UN FPGA



```
always @(posedge clk) begin
  if (draw) begin
    pixel <= 1'b1;
  end else begin
    pixel <= 1'b0;
  end
end
```



simulation  
débogage

HDL

analyse  
synthèse

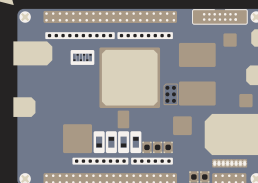
netlist

placement  
routage

assembleur

assemblage

bitstream



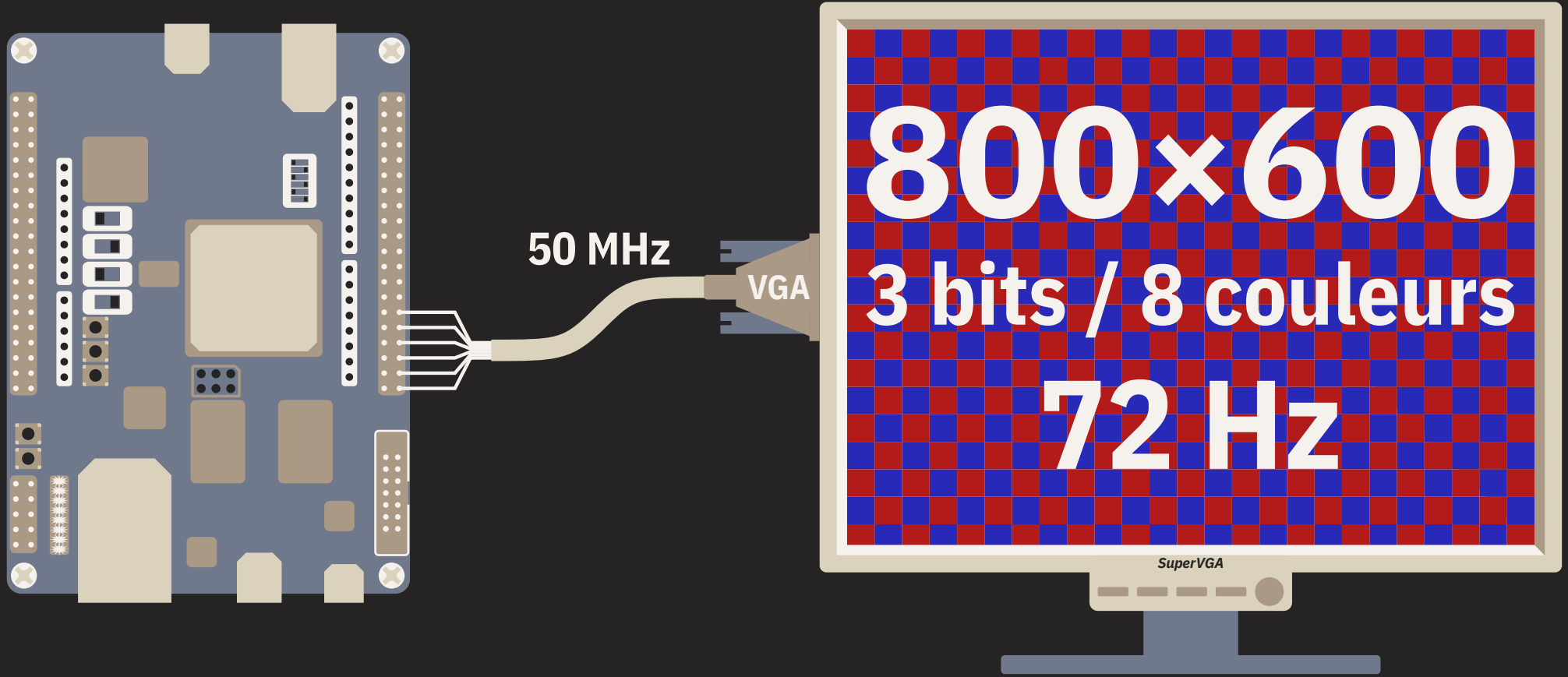
de quelques minutes à plusieurs heures !

# LE FLOT DE CONCEPTION

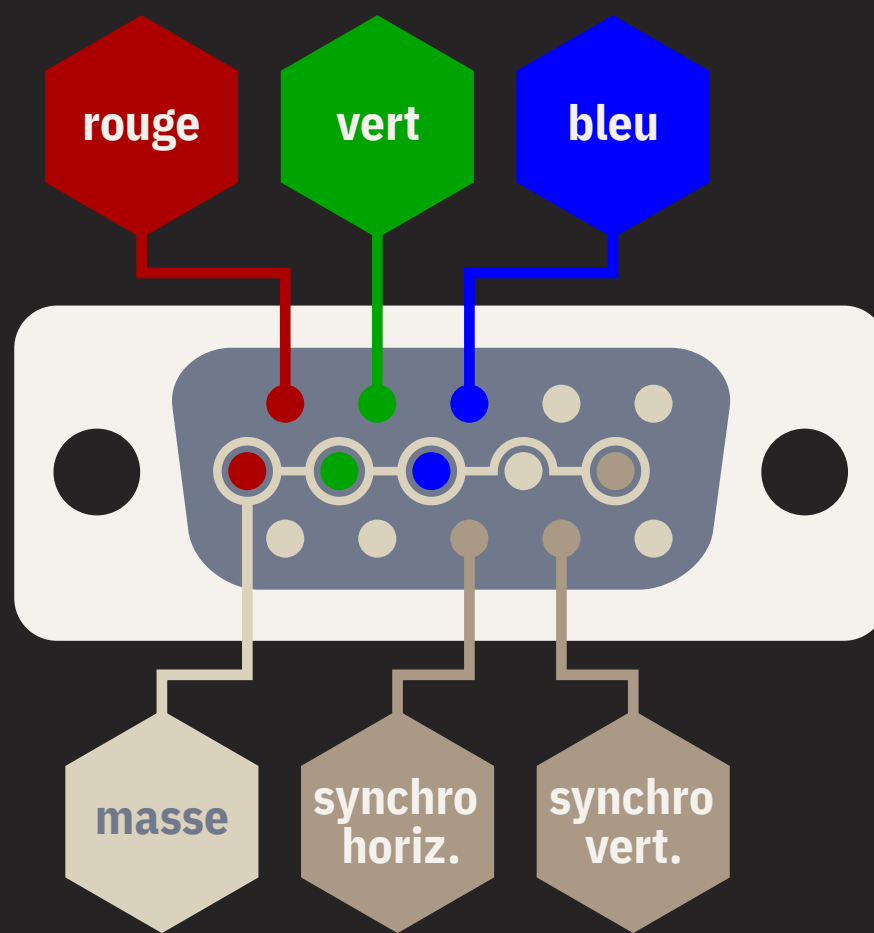


**UN EXEMPLE : SIMPLEVGA**

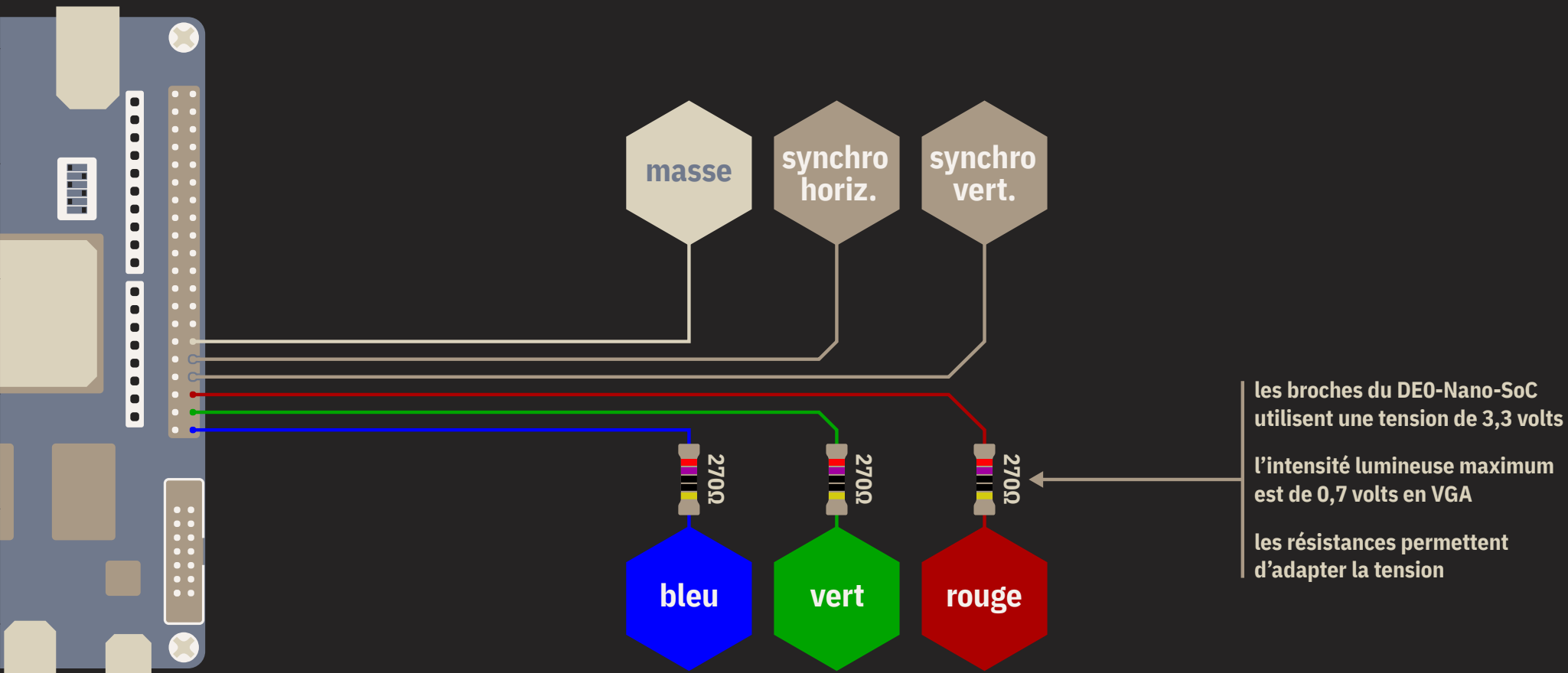




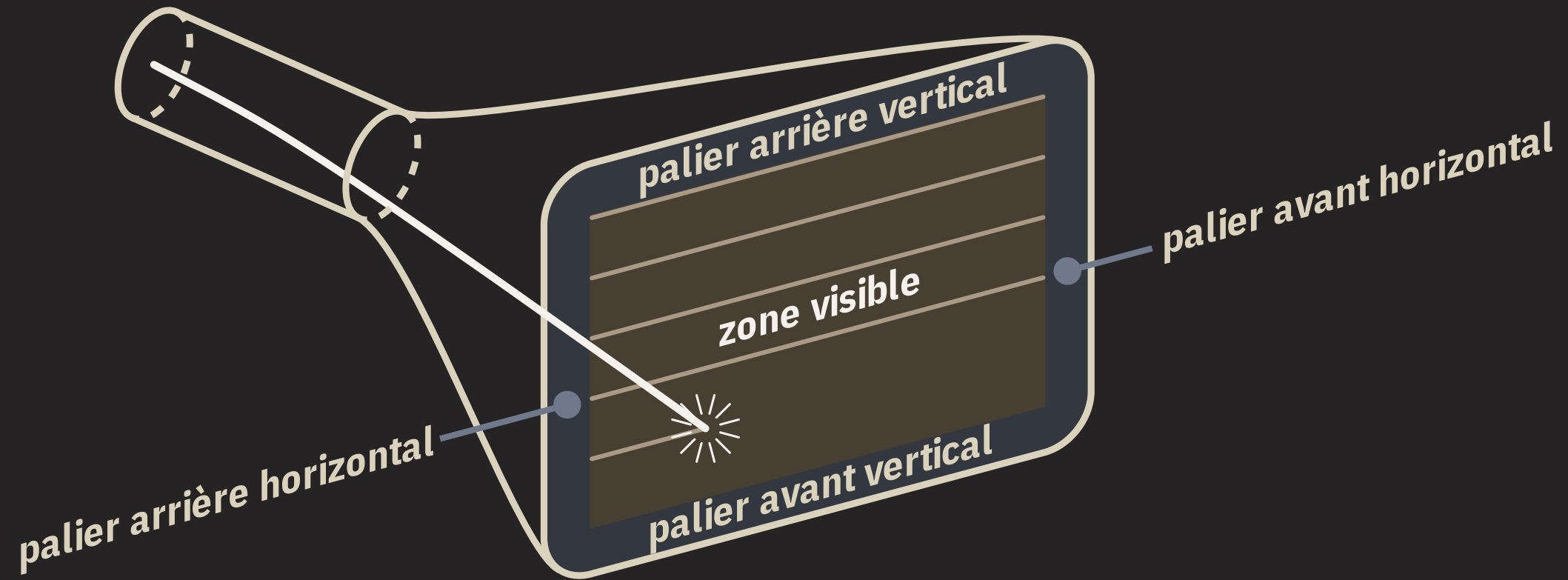
**GÉNÉRATION D'UN SIGNAL VIDÉO VGA**



**LA PRISE VGA**



# CONNECTER LA PRISE VGA AU DE0-NANO-SOC



**UN SIGNAL VIDÉO HÉRITÉ DES TUBES CATHODIQUES**



The diagram illustrates the structure of a VGA signal frame. On the left, a red and blue checkerboard pattern represents the 'ZONE VISIBLE'. To its right, a vertical strip contains three horizontal lines: a thin blue line at the top, a wider tan line in the middle, and another thin blue line at the bottom. Below the visible area, a horizontal strip contains three vertical lines: a thin blue line on the left, a wider tan line in the middle, and another thin blue line on the right. White lines with circular endpoints connect these lines to their respective labels on the right. The entire frame is set against a dark gray background.

**ZONE  
VISIBLE**

palier avant horizontal  
synchronisation horizontale  
palier arrière horizontal

palier avant vertical  
synchronisation verticale  
palier arrière vertical



A horizontal band with a repeating sawtooth or zigzag pattern, colored in a medium blue-gray, representing the horizontal sync signal.

**LE SIGNAL VGA**

# ÉCRITURE DU CODE VERILOG

1) Déclaration du module

2) Synchronisation horizontale

3) Synchronisation verticale

4) Damier rouge et bleu

1

```
module SimpleVGA (  
    input wire clk,  
  
    output wire hsync,  
    output wire vsync,  
    output reg red,  
    output reg green,  
    output reg blue  
);
```

2

```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else                xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;
```

3

```
reg [9:0] ypos = 0;  
always @(posedge clk)  
    if (xpos == 1039) begin  
        if (ypos == 665) ypos <= 0;  
        else                ypos <= ypos + 1;  
    end  
  
assign vsync = ypos < 637 || ypos >= 643;
```

4

```
always @(posedge clk)  
    if (xpos < 800 && ypos < 600) begin  
        red    <= ~xpos[5] ^ ypos[5];  
        green  <= 0;  
        blue   <= xpos[5] ^ ypos[5];  
    end else begin  
        red    <= 0;  
        green  <= 0;  
        blue   <= 0;  
    end  
  
endmodule
```



# DÉCLARATION DU MODULE

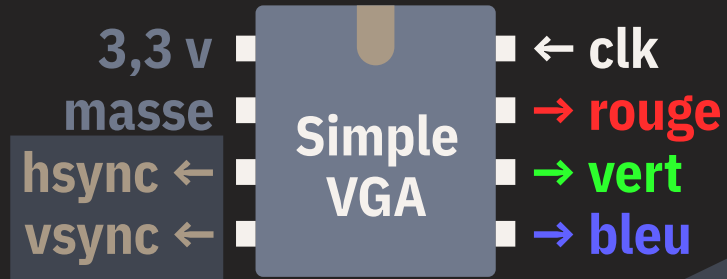




```
module SimpleVGA (  
    input wire clk,  
  
    output wire hsync,  
    output wire vsync,  
    output reg red,  
    output reg green,  
    output reg blue  
);
```

CRÉATION DU CIRCUIT INTÉGRÉ, HORLOGE





```
module SimpleVGA (  
    input wire clk,  
  
    output wire hsync,  
    output wire vsync,  
    output reg red,  
    output reg green,  
    output reg blue  
);
```

CRÉATION DU CIRCUIT INTÉGRÉ, SYNCHRONISATION



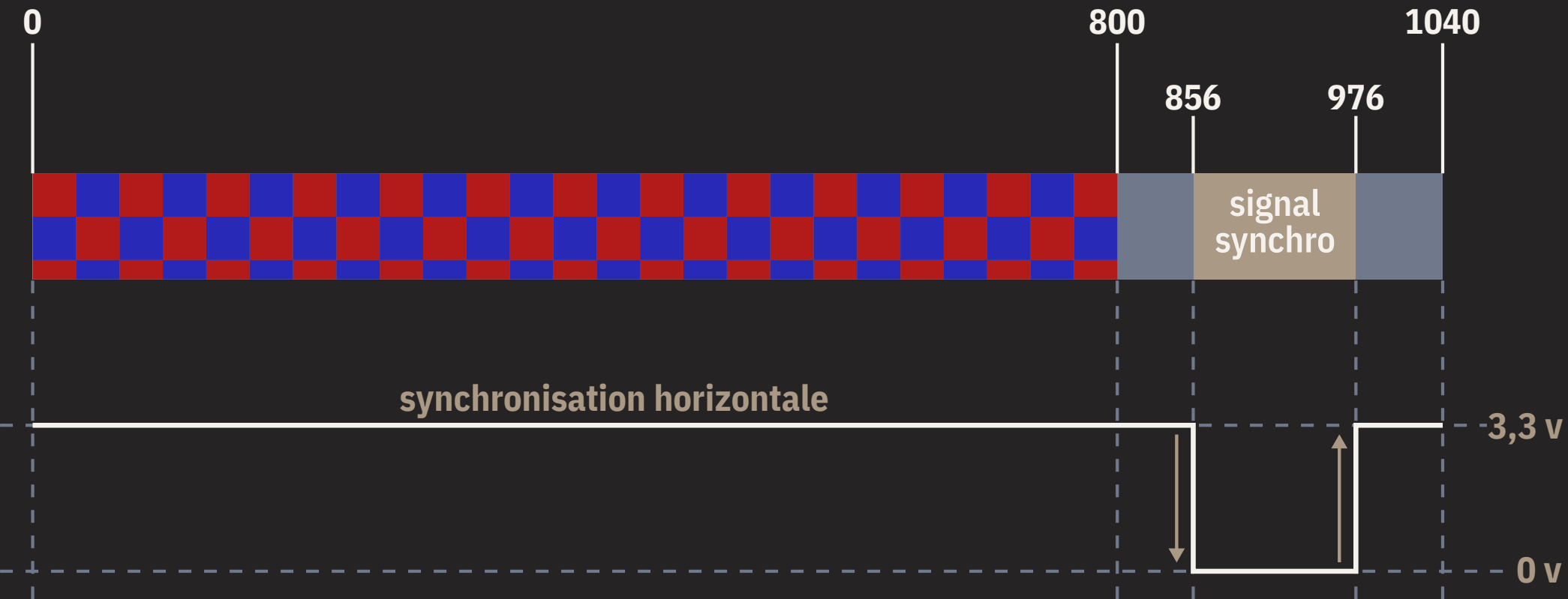
```
module SimpleVGA (  
    input wire clk,  
  
    output wire hsync,  
    output wire vsync,  
    output reg red,  
    output reg green,  
    output reg blue  
);
```

CRÉATION DU CIRCUIT INTÉGRÉ, COULEURS



# SYNCHRONISATION HORIZONTALE






```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else                xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;
```

CODE GÉNÉRANT LA SYNCHRONISATION HORIZONTALE

registre de 11 bits = valeurs de 0 à 2047



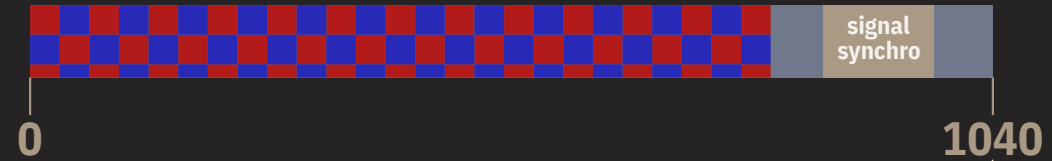
```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else                xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;
```

DÉCLARATION D'UN REGISTRE DE 11 BITS



```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else                xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;
```

EXÉCUTION À CHAQUE FRONT MONTANT



```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else                xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;
```

REMISE À ZÉRO APRÈS 1040 POINTS





```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else                xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;
```

COMPTAGE DE 0 À 1039

exécution continue

exécution rythmée par une horloge

```
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else  
        xpos <= xpos + 1;
```

```
assign hsync = xpos < 856 || xpos >= 976;
```

856 976

signal  
synchro

3,3 v  
masse  
← 50 MHz  
← rouge  
→ vert  
← bleu  
synchro horiz. ←  
synchro vert. ←

# GÉNÉRATION DE LA SYNCHRONISATION HORIZONTALE



# SYNCHRONISATION VERTICALE



```
reg [9:0] ypos = 0;
always @(posedge clk)
    if (xpos == 1039) begin
        if (ypos == 665) ypos <= 0;
        else                ypos <= ypos + 1;
    end

assign vsync = ypos < 637 || ypos >= 643;
```

CODE GÉNÉRANT LA SYNCHRONISATION VERTICALE

registre de 10 bits = valeurs de 0 à 1023

```
reg [9:0] ypos = 0;  
always @(posedge clk)  
    if (xpos == 1039) begin  
        if (ypos == 665) ypos <= 0;  
        else  
            ypos <= ypos + 1;  
    end  
  
assign vsync = ypos < 637 || ypos >= 643;
```

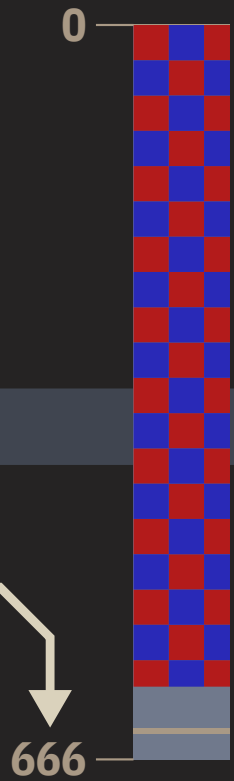
DÉCLARATION D'UN REGISTRE DE 10 BITS



```
reg [9:0] ypos = 0;  
always @(posedge clk)  
    if (xpos == 1039) begin  
        if (ypos == 665) ypos <= 0;  
        else  
            ypos <= ypos + 1;  
    end  
  
assign vsync = ypos < 637 || ypos >= 643;
```

CALCUL À CHAQUE FIN DE LIGNE

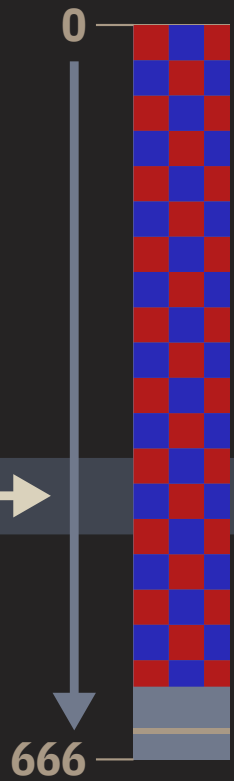
```
reg [9:0] ypos = 0;
always @(posedge clk)
    if (xpos == 1039) begin
        if (ypos == 665) ypos <= 0;
        else
            ypos <= ypos + 1;
    end
assign vsync = ypos < 637 || ypos >= 643;
```



REMISE À ZÉRO APRÈS 666 LIGNES

```
reg [9:0] ypos = 0;
always @(posedge clk)
    if (xpos == 1039) begin
        if (ypos == 665) ypos <= 0;
        else                ypos <= ypos + 1;
    end

assign vsync = ypos < 637 || ypos >= 643;
```

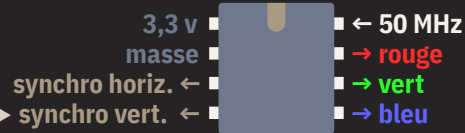


COMPTAGE DE 0 À 665



```
reg [9:0] ypos = 0;  
always @(posedge clk)  
    if (xpos == 1039) begin  
        if (ypos == 665) ypos <= 0;  
        else  
            ypos <= ypos + 1;  
    end
```

```
assign vsync = ypos < 637 || ypos >= 643;
```



637  
643

# GÉNÉRATION DE LA SYNCHRONISATION VERTICALE



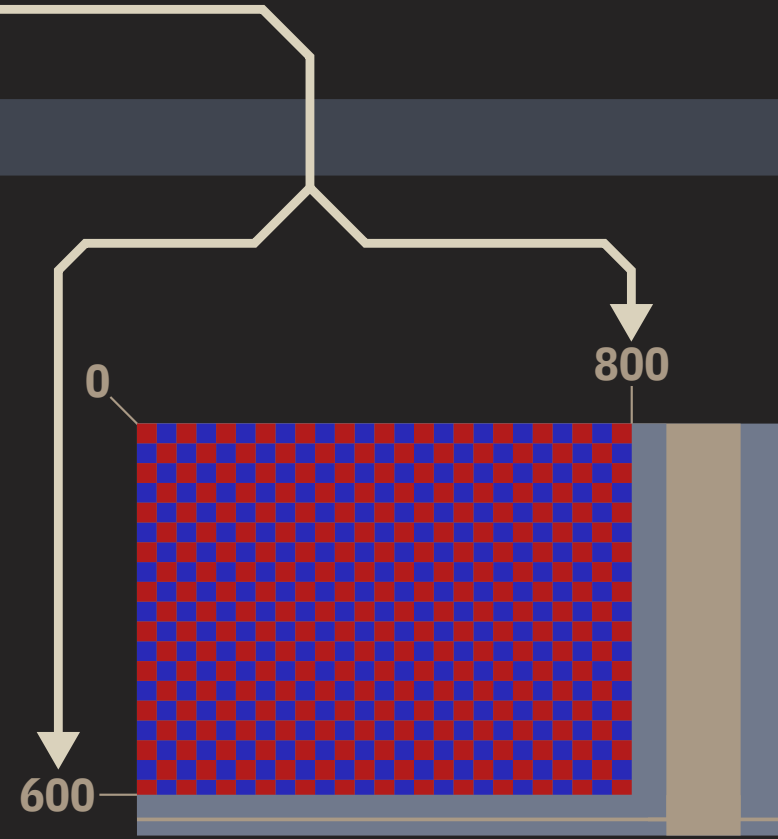
**DAMIER ROUGE ET BLEU**



```
always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin
        red    <= ~xpos[5] ^ ypos[5];
        green  <= 0;
        blue   <= xpos[5] ^ ypos[5];
    end else begin
        red    <= 0;
        green  <= 0;
        blue   <= 0;
    end
end
```

CODE GÉNÉRANT LE DAMIER ROUGE ET BLEU

```
always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin
        red    <= ~xpos[5] ^ ypos[5];
        green  <= 0;
        blue   <= xpos[5] ^ ypos[5];
    end else begin
        red    <= 0;
        green  <= 0;
        blue   <= 0;
    end
end
```



TEST DE LA ZONE VISIBLE

```
always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin
        red    <= ~xpos[5] ^ ypos[5];
        green  <= 0;
        blue   <= xpos[5] ^ ypos[5];
    end else begin
        red    <= 0;
        green  <= 0;
        blue   <= 0;
    end
end
```

les 3 calculs et affectations  
sont exécutés en parallèle

UN FPGA TRAVAILLE EN PARALLÈLE

```

always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin

```

```

        red  <= ~xpos[5] ^ ypos[5];

```

```

        green <= 0;

```

```

        blue <= xpos[5] ^ ypos[5];

```

```

    end else begin

```

```

        red  <= 0;

```

```

        green <= 0;

```

```

        blue <= 0;

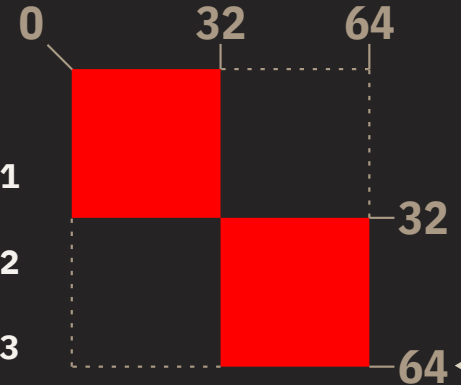
```

```

    end

```

10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	= 0
0	0	0	0	0	0	1	1	1	1	1	= 31
0	0	0	0	0	1	0	0	0	0	0	= 32
0	0	0	0	0	1	1	1	1	1	1	= 63

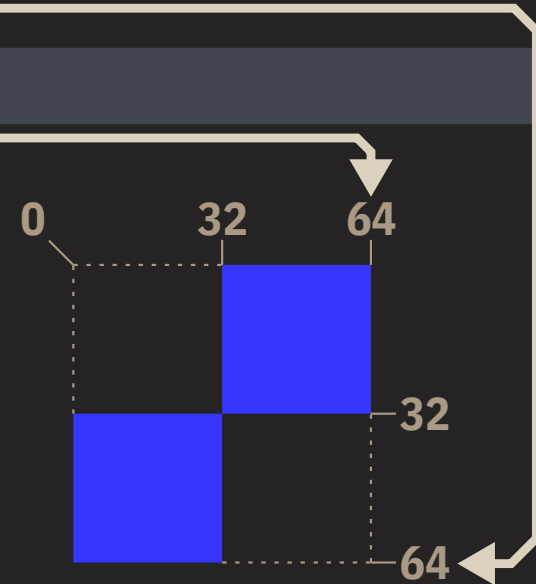


# ALTERNER LES CARRÉS ROUGES

```
always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin
        red    <= ~xpos[5] ^ ypos[5];
        green  <= 0;
        blue   <= xpos[5] ^ ypos[5];
    end else begin
        red    <= 0;
        green  <= 0;
        blue   <= 0;
    end
end
```

LE VERT N'EST JAMAIS UTILISÉ

```
always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin
        red    <= ~xpos[5] ^ ypos[5];
        green  <= 0;
        blue   <= xpos[5] ^ ypos[5];
    end else begin
        red    <= 0;
        green  <= 0;
        blue   <= 0;
    end
end
```



ALTERNER LES CARRÉS BLEUS



```
always @(posedge clk)
    if (xpos < 800 && ypos < 600) begin
        red    <= ~xpos[5] ^ ypos[5];
        green  <= 0;
        blue   <= xpos[5] ^ ypos[5];
    end else begin
        red    <= 0;
        green  <= 0;
        blue   <= 0;
    end
end
```

AUCUNE COULEUR EN DEHORS DE LA ZONE VISIBLE

# REMARQUES ET PIÈGES

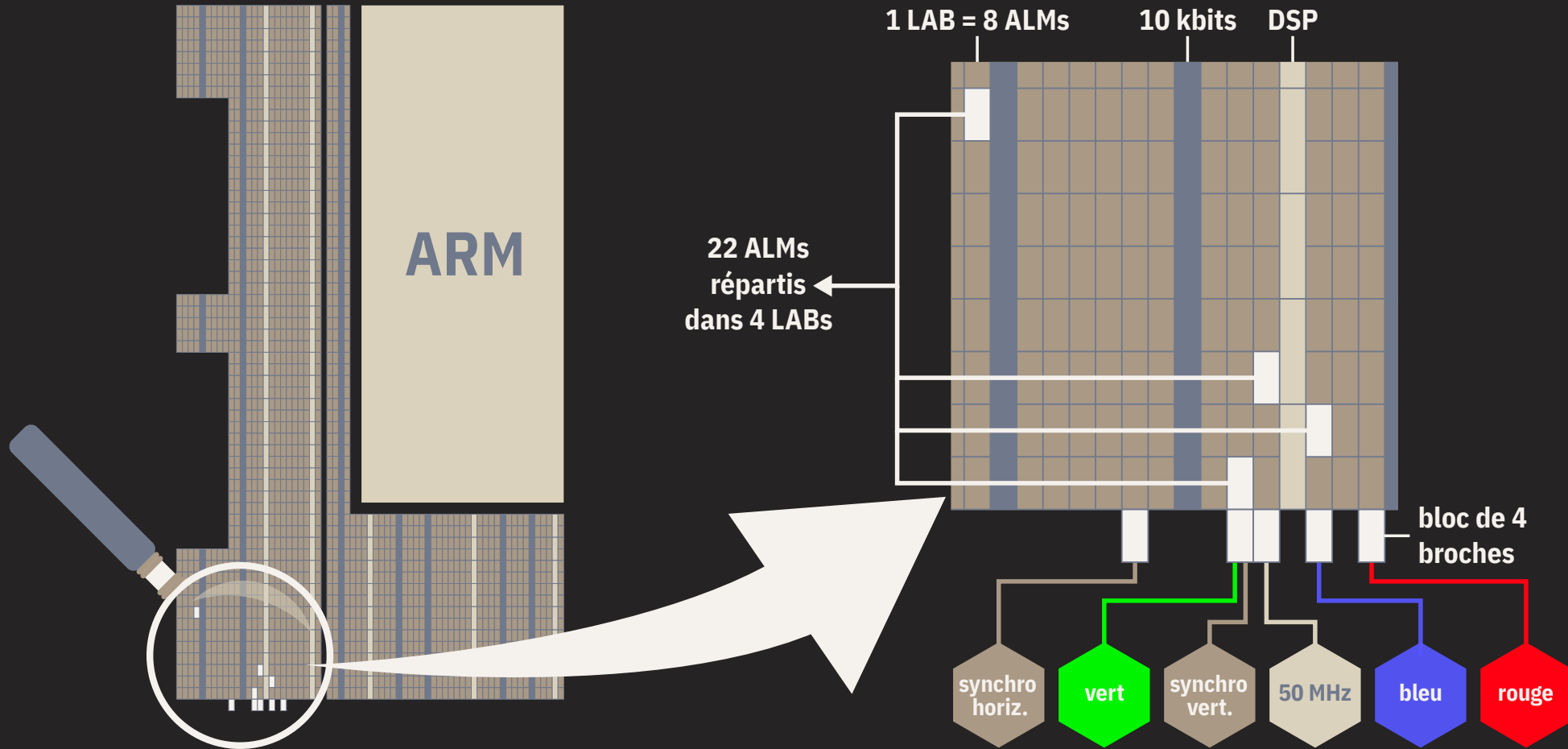
- Verilog n'est pas un langage procédural
- Un bit peut avoir 4 états
  - 0, 1, X et Z → faux, vrai, inconnu, impédance haute
- Vous êtes responsables des délais de propagation
- Générer une image « bitstream » est long...
- Tous les codes HDL ne sont pas synthétisables



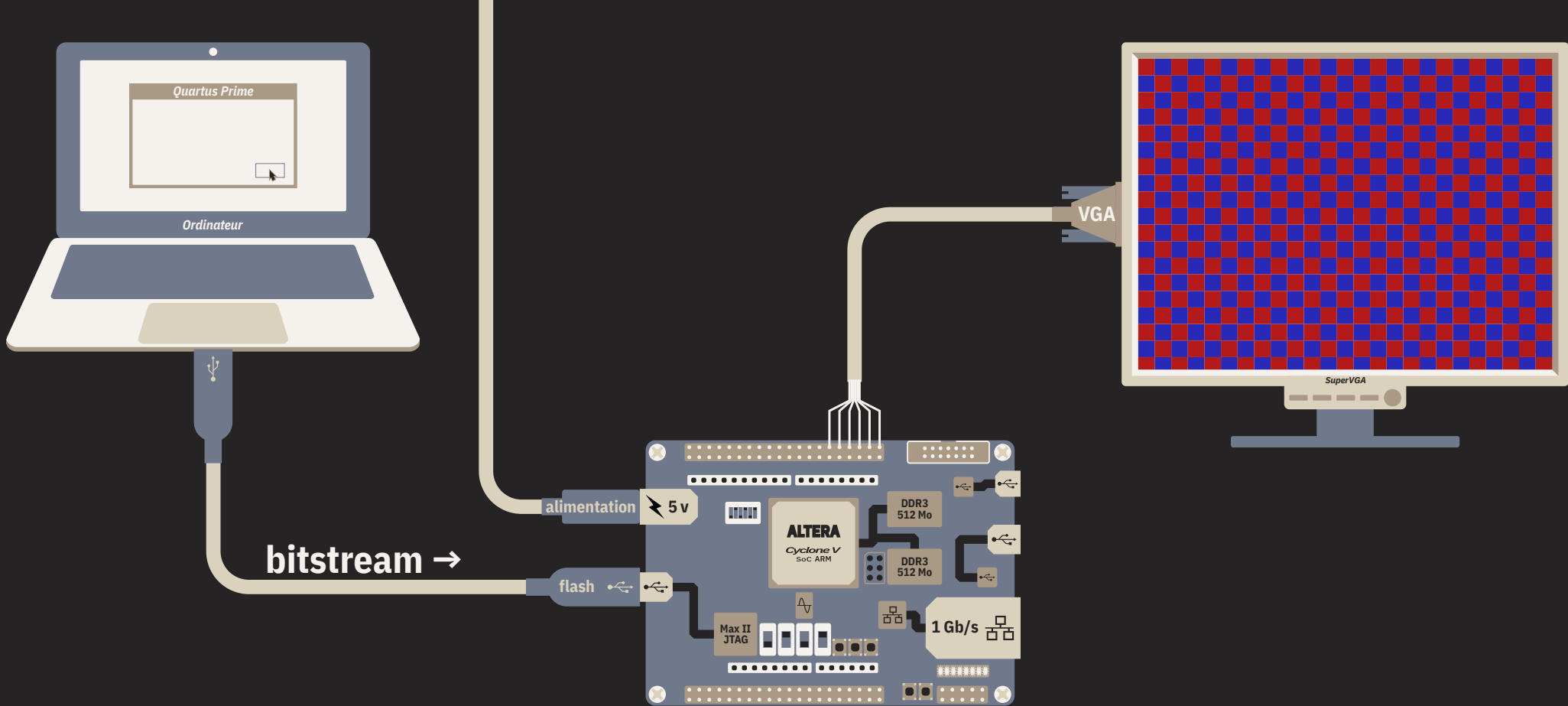
# GÉNÉRATION DE L'IMAGE « BITSTREAM »





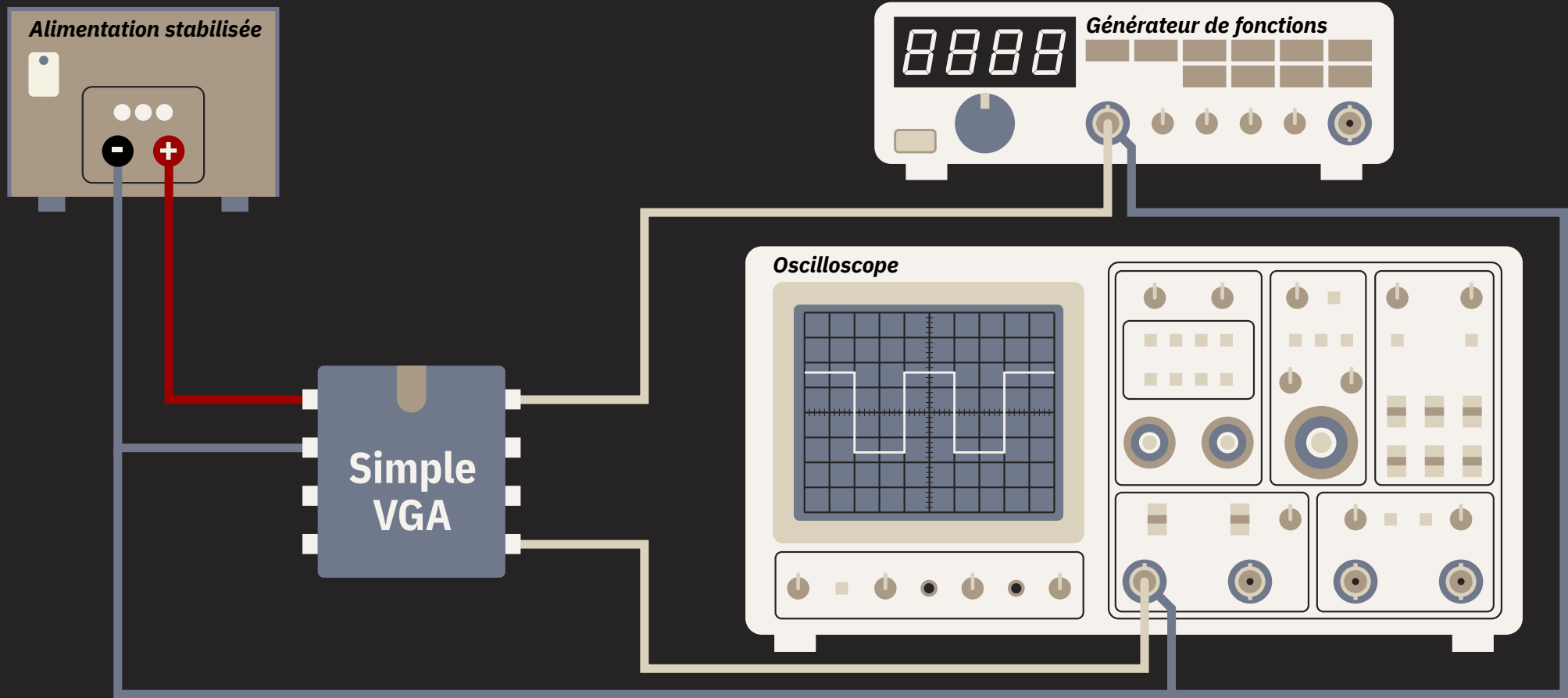


## À L'INTÉRIEUR DU FPGA : PLACEMENT ET ROUTAGE



ENVOI DE L'IMAGE « BITSTREAM »

# DÉBOGUEUR UN CIRCUIT



**DÉBOGUEUR PHYSIQUEMENT**



## Icarus Verilog

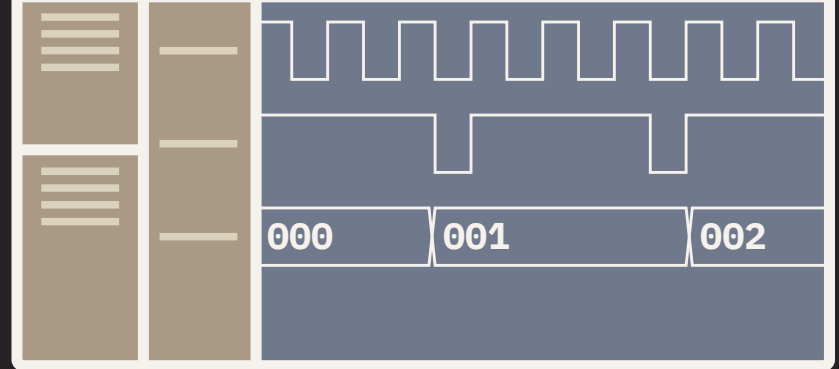
```
module SimpleVGA (  
    input wire clk,  
  
    output wire hsync,  
    output wire vsync,  
    output reg red,  
    output reg green,  
    output reg blue  
);  
  
reg [10:0] xpos = 0;  
always @(posedge clk)  
    if (xpos == 1039) xpos <= 0;  
    else xpos <= xpos + 1;  
  
assign hsync = xpos < 856 || xpos >= 976;  
  
reg [10:0] ypos = 0;  
always @(posedge clk)  
    if (ypos == 665) ypos <= 0;  
    else ypos <= ypos + 1;  
  
end  
  
assign vsync = ypos < 637 || ypos >= 643;  
  
always @(posedge clk)  
    if (xpos < 800 && ypos < 600) begin  
        red <= xpos[5] ^ ypos[5];  
        green <= 0;  
        blue <= ~xpos[5] ^ ypos[5];  
    end else begin  
        red <= 0;  
        green <= 0;  
        blue <= 0;  
    end  
  
endmodule
```

**SimpleVGA + Banc d'essai**

```
`timescale 10ns / 10ns  
module SimpleVGA_tb;  
    reg clk = 0;  
    always #1 clk = !clk;  
  
    wire hsync;  
  
    SimpleVGA testbench(  
        .clk (clk),  
        .hsync (hsync)  
    );  
  
    in $dumpfile("SimpleVGA_tb.vcd");  
    $dumpvars(0, testbench);  
  
    #1385280 $finish;  
  
end  
  
reg [24:0] counter = 0;  
always @(negedge hsync) begin  
    counter = counter + 1;  
    $display("HSYNC = %0d @ %0t ns", counter, $time);  
end  
endmodule
```



## GTKWave



# SIMULER POUR DÉBOGUEUR

# ORGANISATION DU BANC D'ESSAI

- 1) Génération d'une horloge
- 2) Instanciation de SimpleVGA
- 3) Lancement de la simulation
- 4) Création d'un compteur

1

```
`timescale 10ns / 10ns
module SimpleVGA_tb;
    reg clk = 0;
    always #1 clk = !clk;

    wire hsync;
```

2

```
SimpleVGA testbench(
    .clk (clk),
    .hsync (hsync)
);
```

3

```
initial begin
    $dumpfile("SimpleVGA_tb.vcd");
    $dumpvars(0, testbench);

    #1385280 $finish;
end
```

4

```
reg [24:0] counter = 0;
always @(negedge hsync) begin
    counter = counter + 1;
    $display("HSYNC = %0d @ %0t ns", counter, $time);
end
endmodule
```

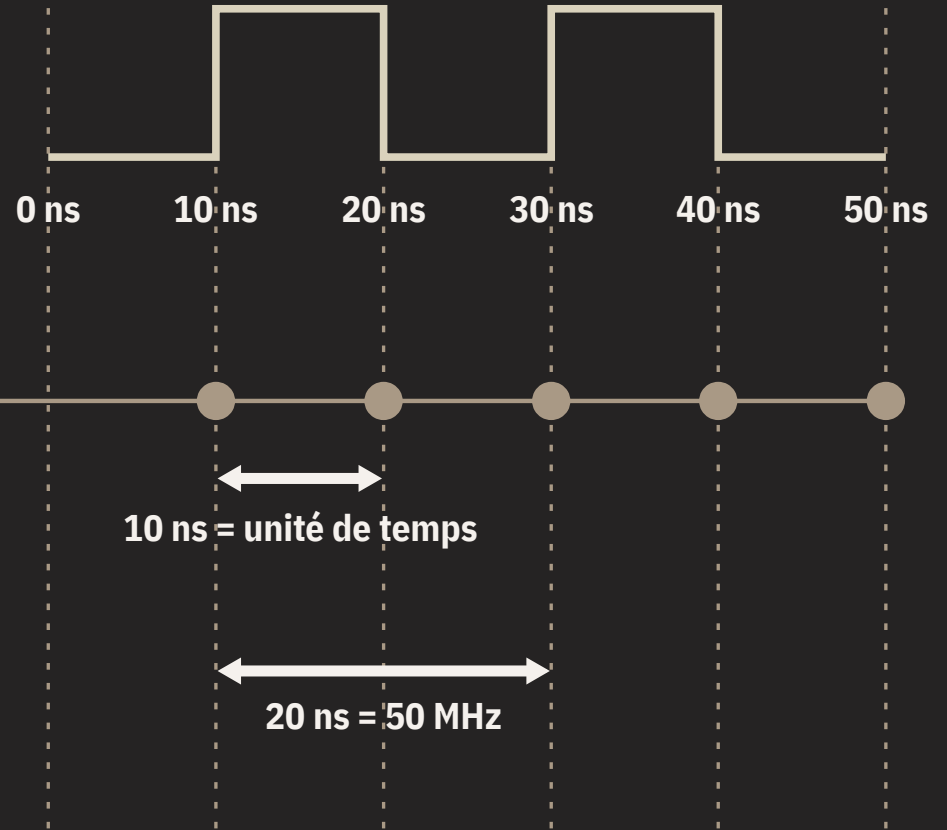
```
`timescale 10ns / 10ns
```

```
module SimpleVGA_tb;
```

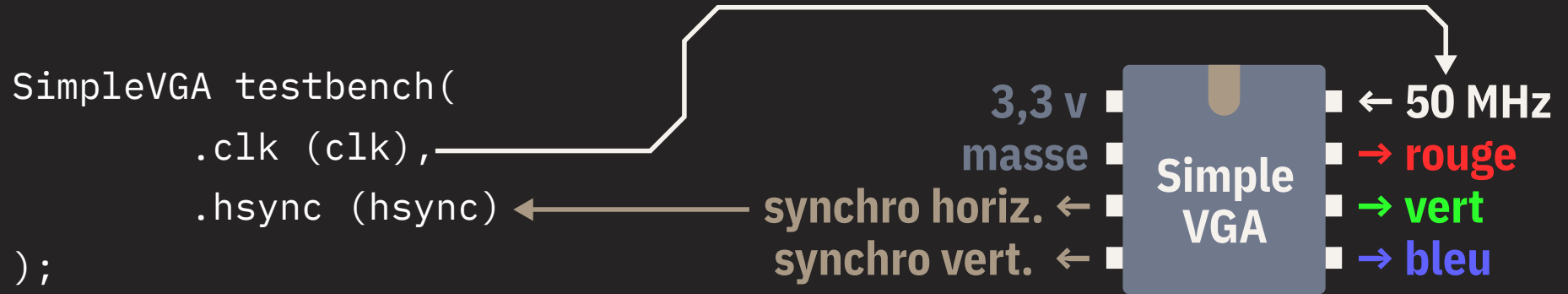
```
  reg clk = 0;
```

```
  always #1 clk = !clk;
```

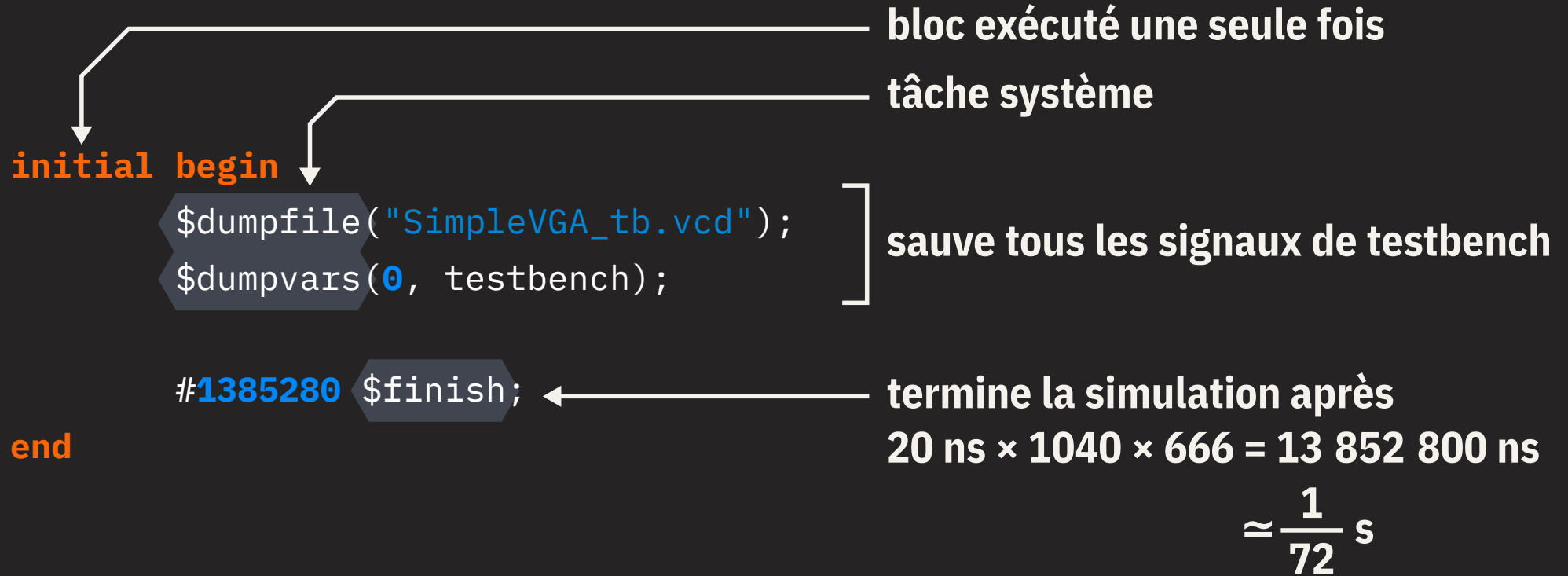
```
  wire hsync;
```



# GÉNÉRATION D'UNE HORLOGE



INSTANCIATION DE SIMPLEVGA



**LANCEMENT DE LA SIMULATION**

```
reg [24:0] counter = 0;  
always @(negedge hsync) begin  
    counter = counter + 1;  
    $display("HSYNC = %0d @ %0t ns", counter, $time);  
end
```

affichage du compteur



## CRÉATION D'UN COMPTEUR

*Ligne de commande*

```
$ iverilog -o SimpleVGA_tb SimpleVGA_tb.v SimpleVGA.v
```

**COMPILATION DU BANC D'ESSAI**

*Ligne de commande*

```
$ iverilog -o SimpleVGA_tb SimpleVGA_tb.v SimpleVGA.v  
$ ./SimpleVGA_tb  
VCD info: dumpfile SimpleVGA_tb.vcd opened for output.  
HSYNC = 1 @ 1711 ns  
HSYNC = 2 @ 3791 ns  
HSYNC = 3 @ 5871 ns  
# ...  
HSYNC = 664 @ 1380751 ns  
HSYNC = 665 @ 1382831 ns  
HSYNC = 666 @ 1384911 ns
```

# EXÉCUTION DE LA SIMULATION

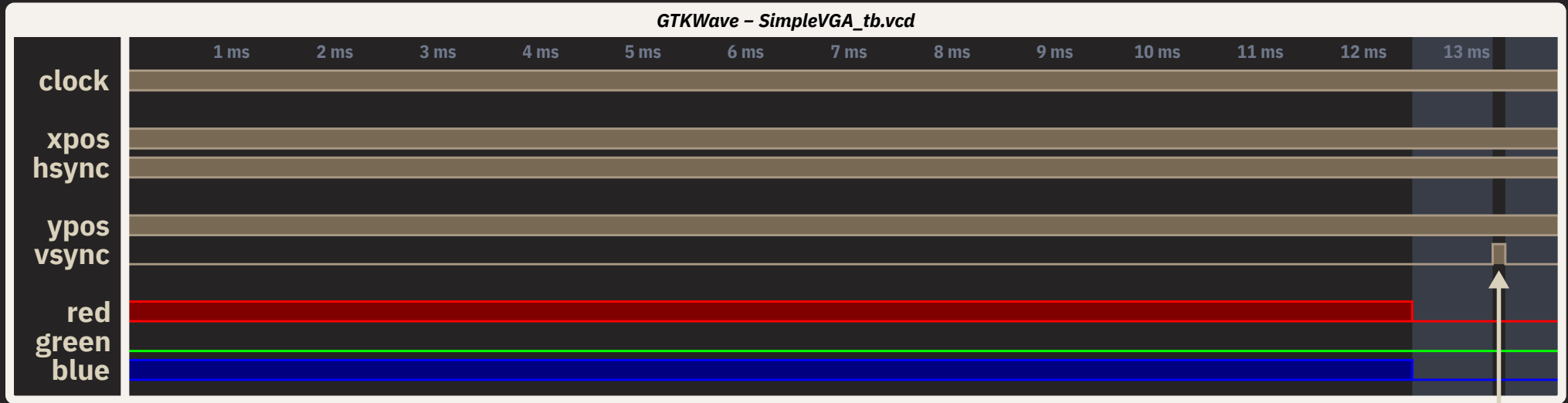


*Ligne de commande*

```
$ iverilog -o SimpleVGA_tb SimpleVGA_tb.v SimpleVGA.v  
$ ./SimpleVGA_tb  
VCD info: dumpfile SimpleVGA_tb.vcd opened for output.  
HSYNC = 1 @ 1711 ns  
HSYNC = 2 @ 3791 ns  
HSYNC = 3 @ 5871 ns  
# ...  
HSYNC = 664 @ 1380751 ns  
HSYNC = 665 @ 1382831 ns  
HSYNC = 666 @ 1384911 ns  
$ gtkwave SimpleVGA_tb.vcd
```

**LANCEMENT DE GTKWAVE**

1 frame



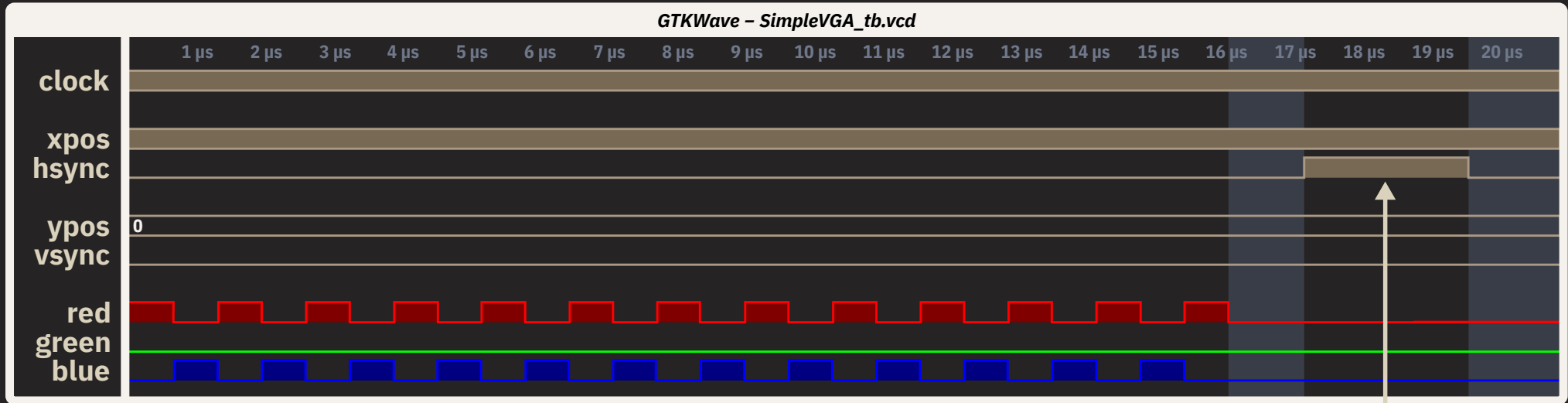
palier avant vertical

synchronisation verticale

palier arrière vertical

ÉVOLUTION DES SIGNAUX (FRAME)

1 ligne



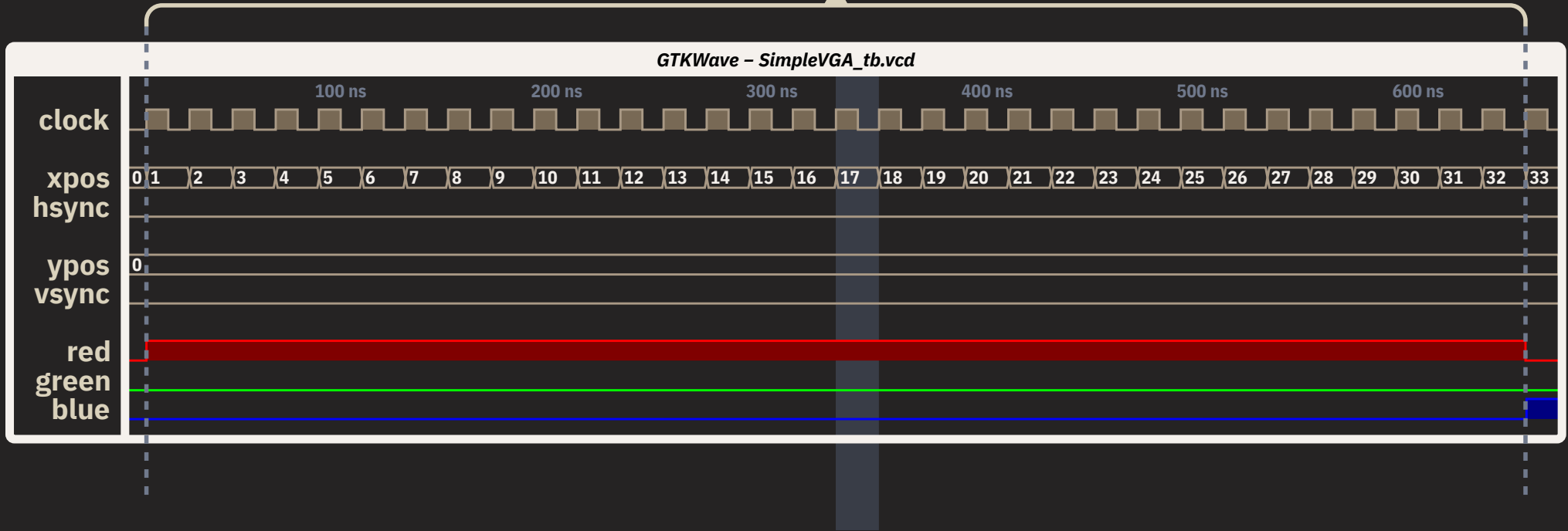
palier avant horizontal

synchronisation horizontale

palier arrière horizontal

ÉVOLUTION DES SIGNAUX (LIGNE)

32 cycles



1 cycle = 20 ns / 50 MHz = 1 pixel

ÉVOLUTION DES SIGNAUX (HORLOGE)

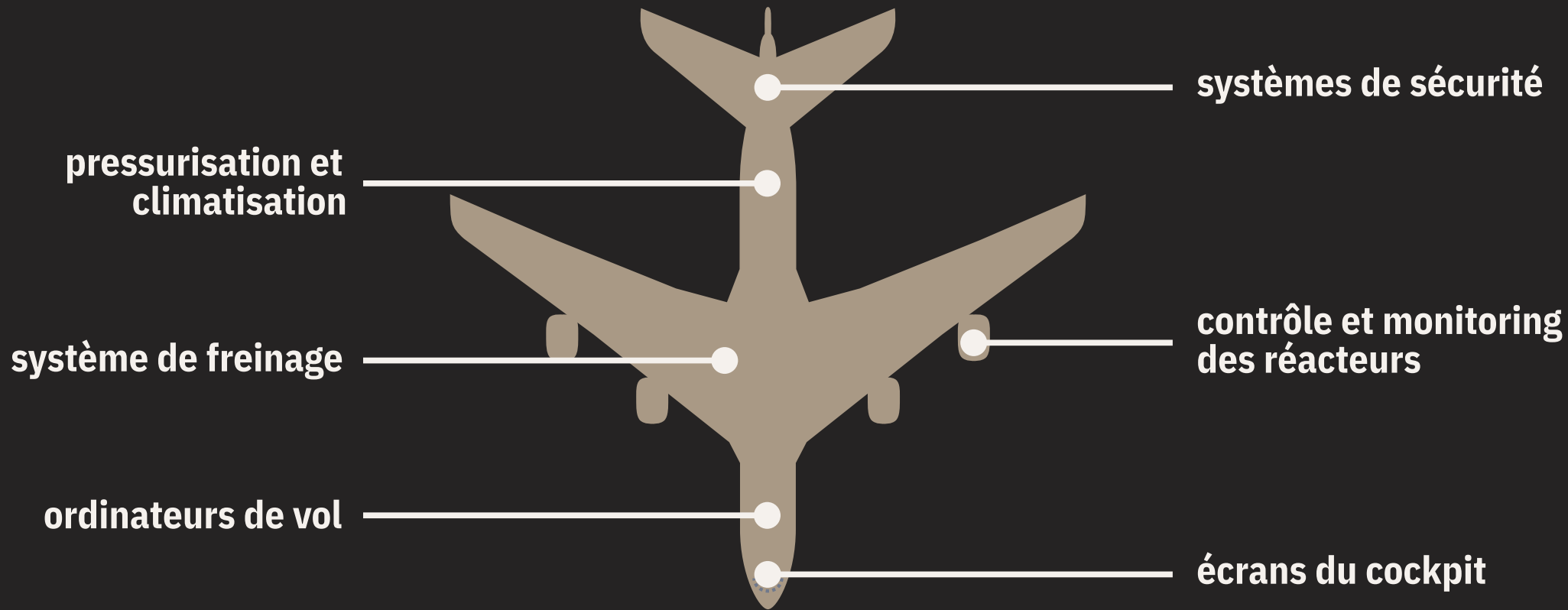


**ET MAINTENANT ?**



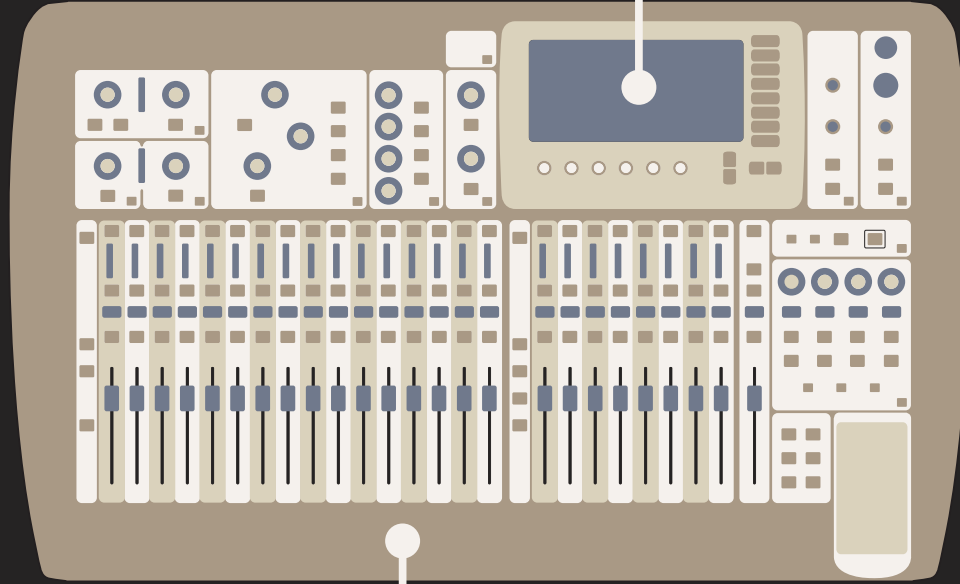
# QUELQUES EXEMPLES D'UTILISATION





**AIRBUS A380**

effets en temps réel



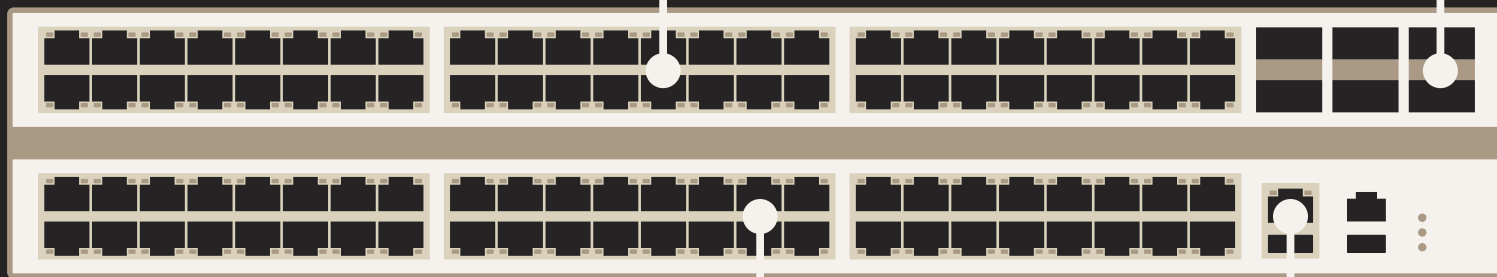
mixage de nombreuses entrées

## TABLE DE MIXAGE BEHRINGER X32



**très faible latence**

**très haut débit**



**gestion de nombreuses entrées**

**mises à jour**

**SWITCH CISCO**

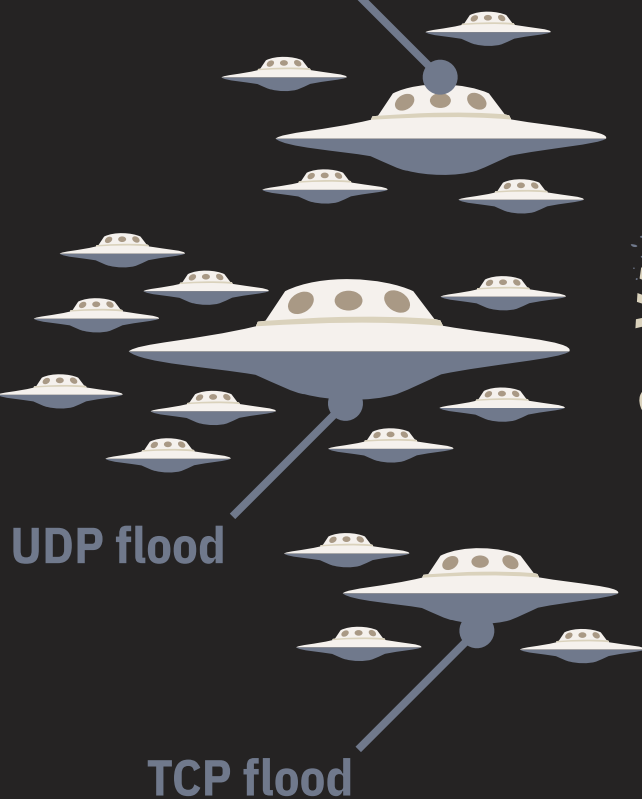
traitement d'image  
après acquisition



traitement des  
données brutes

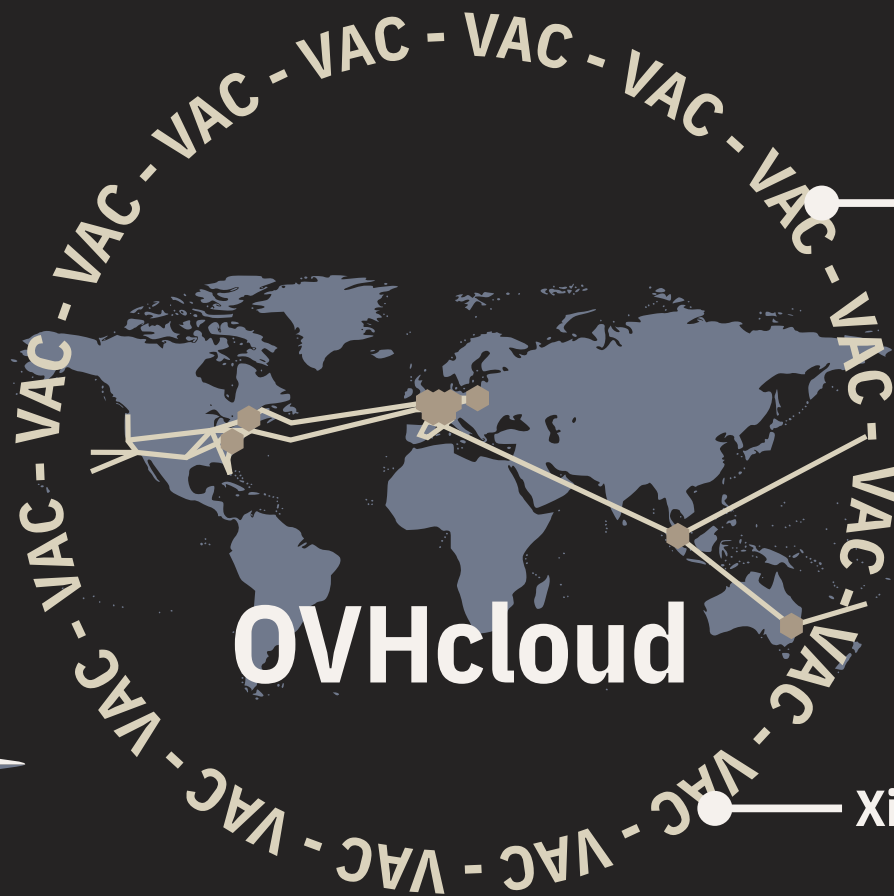
IMAGERIE MÉDICALE : IRM

SYN flood



UDP flood

TCP flood



Intel Stratix V

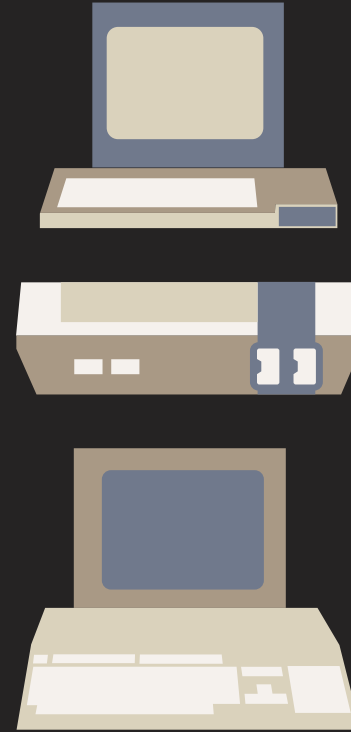
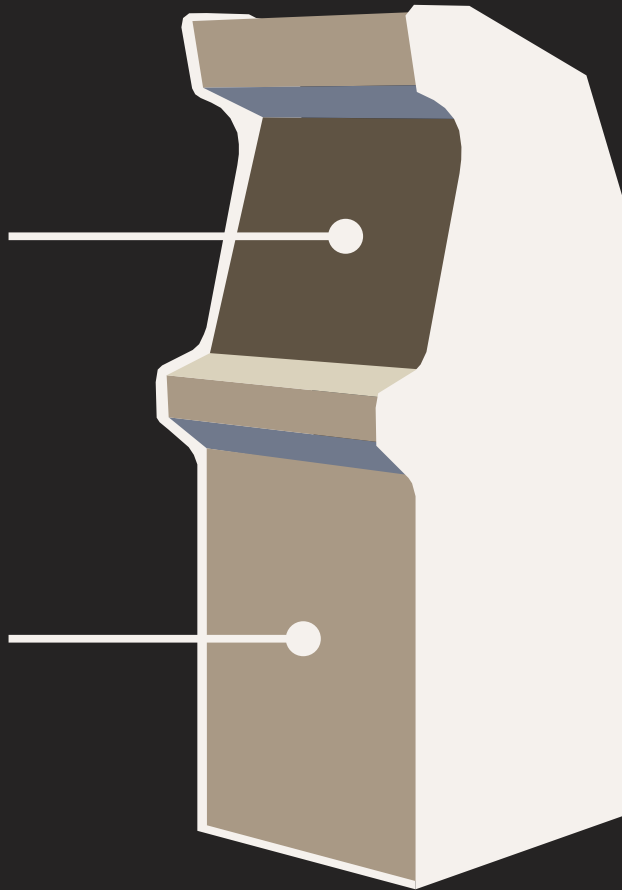
OVHcloud

Xilinx Virtex UltraScale+

SYSTÈME ANTI-DDOS D'OVHCLOUD

projet MiSTer

Terasic DE10-nano

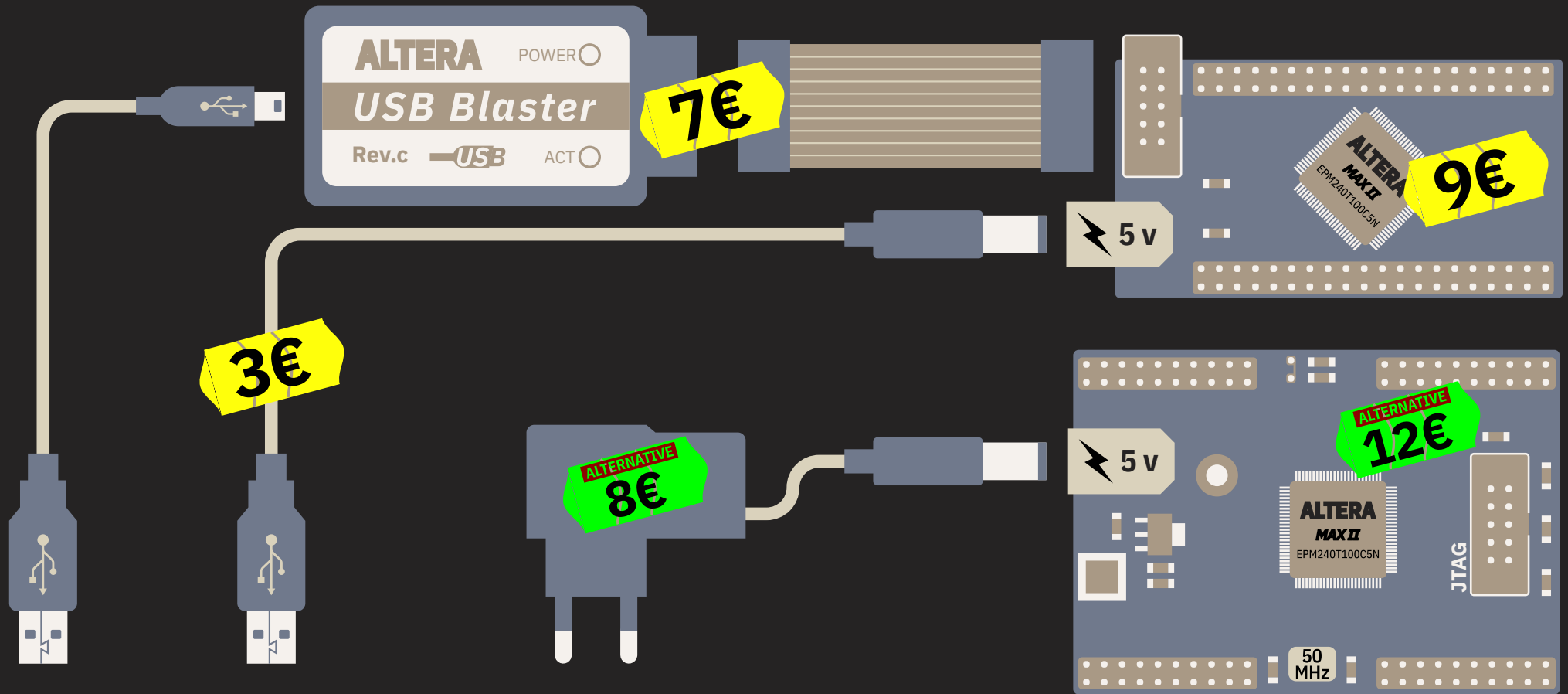


ÉMULATION FINE D'ANCIENNES PLATEFORMES



**PAR OÙ COMMENCER ?**





AVEC UN BUDGET DE 20€



# **AVANTAGES/INCONVÉNIENTS DES FPGA**



# INCONVÉNIENTS DES FPGA

- Autre façon de programmer
  - paradigme
  - temps de compilation
  - débogage
- Positionnement
  - ASIC
  - GPU
  - microcontrôleur
- Ticket d'entrée
  - coût des logiciels
  - coût des circuits
- Écosystème limité
  - open source peu présent
  - disponibilité des compétences
  - OpenCL



# AVANTAGES DES FPGA

- Plus proche de l'électron
  - flexibilité des entrées/sorties
  - moins de composants
  - consommation électrique
- Puissance de calcul
- Précision du signal
  - comportement déterministe
  - faible latence
- Reconfigurable
- Sécurité
  - pas de dépassement
  - offuscation ?



# **Merci de votre attention !**

Merci à

Virginie Férey Rochefeuille, Rémi Passerieu, David Glaude, Thoma Hauc,  
Tristan Groléat (OVHcloud), Francis Trautmann, Sébastien Dupire,  
Loïc “Iooner”, Pascale Lambert-Charreteur,  
Échelle Inconnue et l'équipe du Devfest Nantes

# LICENCES









- Présentation sous Licence CC-BY 4.0
  - textes, images, gabarits... sauf mention contraire
  - <https://creativecommons.org/licenses/by/4.0/legalcode.fr>
- Les polices suivantes ont été utilisées
  - IBM Plex™, licence OFL
  - Bebas Neue, licence OFL

# LOGICIELS

- Logiciels

- Intel® Quartus® Prime Lite
- ModelSim ASE
- Icarus Verilog 
- GTKWave 

- Présentation

- Inkscape 
- LibreOffice Impress 
- LibreOffice Writer 
- LibreOffice Calc 
- JabRef 
- hilite.me 
- Coolors.co 
- TinyVGA 

# **BIBLIOGRAPHIE**