# *Développez vos systèmes embarqués sur SoC FPGA*

***Comment embarquer Linux et développer en VHDL vos applicatifs dédiés au traitement d'image sans pénaliser le CPU.***

**Jean-Marie CODOL**
Développeur

Submarine Open Technologies
Montpellier

Avril 2022

# Préambule (1/2)

SUBTEQ.iO
SUBMARINE OPEN TECHNOLOGIES

## *Equipement portatif pour scaphandrier*

*Traitement du signal acoustique (FPGA)*
*Système embarqué*
- *Linux embarqué*
- *SIG*

**Jean-Marie CODOL**

- *Directeur R&D – co-fondateur*
- *Développeur (C / JAVA / Matlab) – I.A.*
- *Développeur Systèmes Embarqués (PIC / Arduino / FPGA)*
- *C.A.O. Electronique, modélisation 3D*
- *Scaphandrier*
- *Support à l'enseignement : Ecole des Mines d'Alès*

# Préambule (2/2)

***Plan de la formation :***

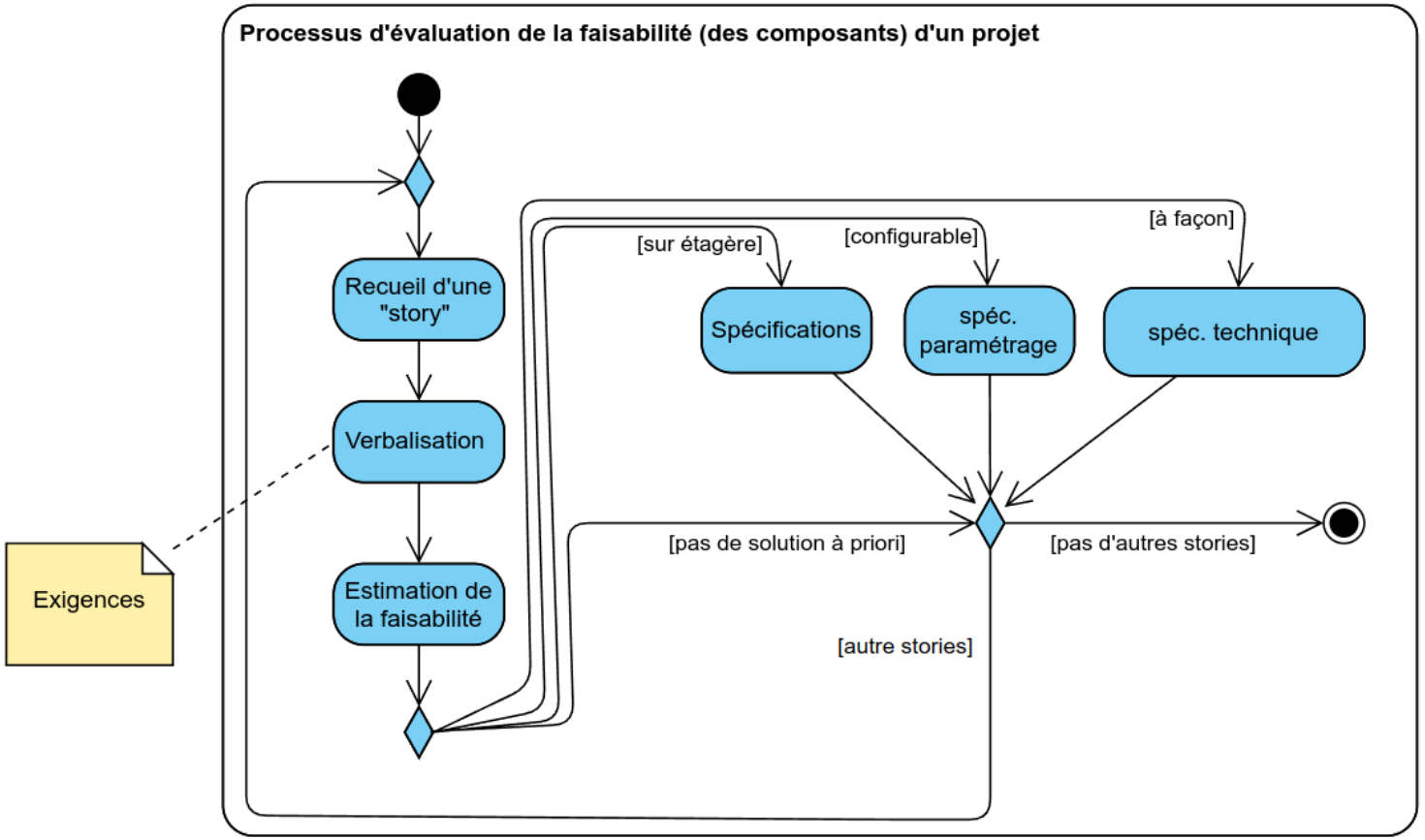
**JOUR 1 : Mise en œuvre du SoC FPGA de10-nano**
- Introduction
- Comment programmer un SoC FPGA chez Intel
- Mettre en œuvre la distribution Linux fournie
- *[Pour ceux qui sont en avance] Mettre en œuvre sa propre distribution Linux*

**JOUR 2 : Décharger le CPU d'une opération**
- Décharger le CPU d'une opération en un port classique
- Décharger le CPU en utilisant la RAM

**JOUR 3 : Cas d'application en traitement d'image**
- Seuillage d'image sur FPGA
- Traitement d'image (format OpenCV)

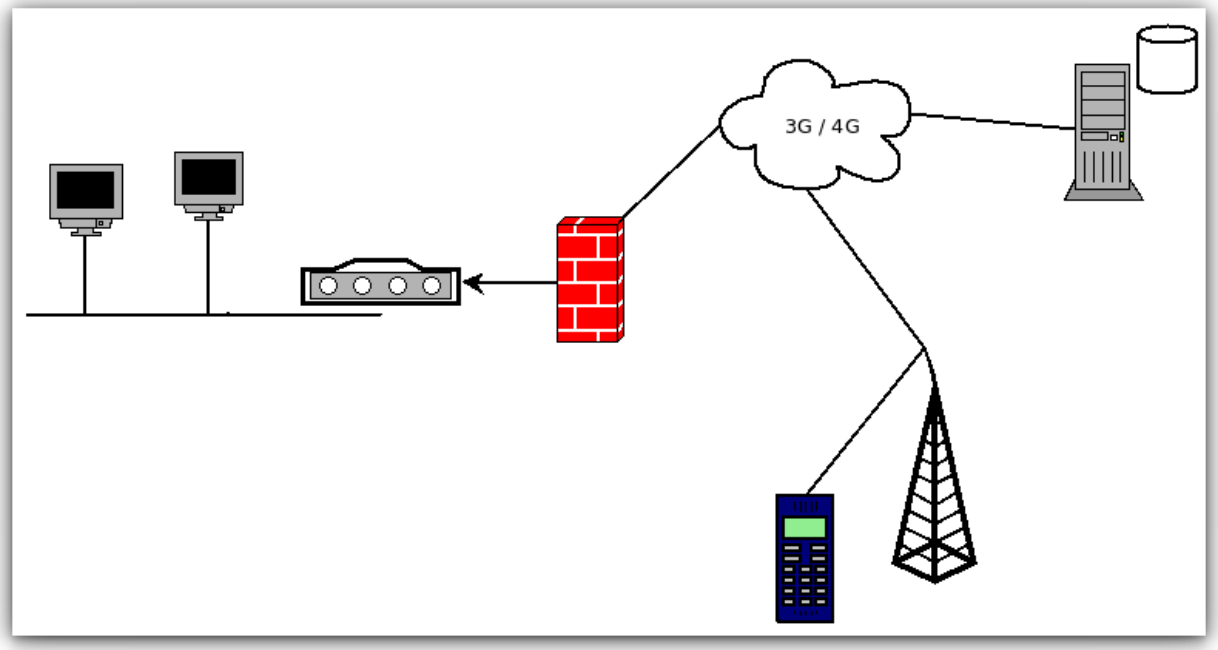
# Introduction

Du cahier des charges du projet à l'architecture réseau



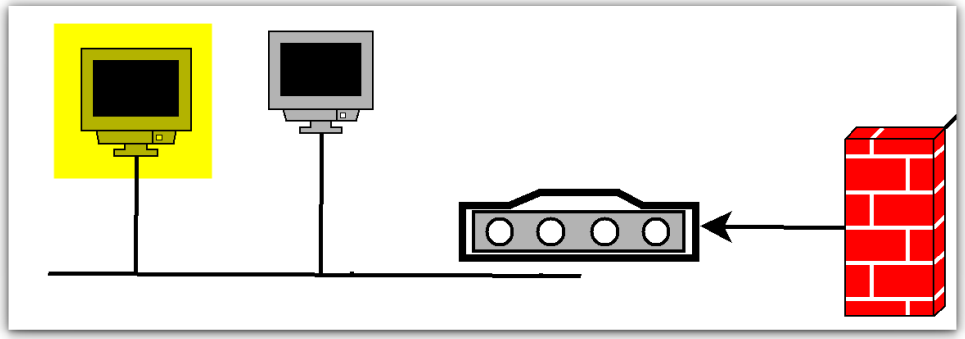
# Introduction

Du cahier des charges du projet à l'architecture réseau



# Introduction

Architecture réseau >> contraintes sur les unités de calcul embarquées

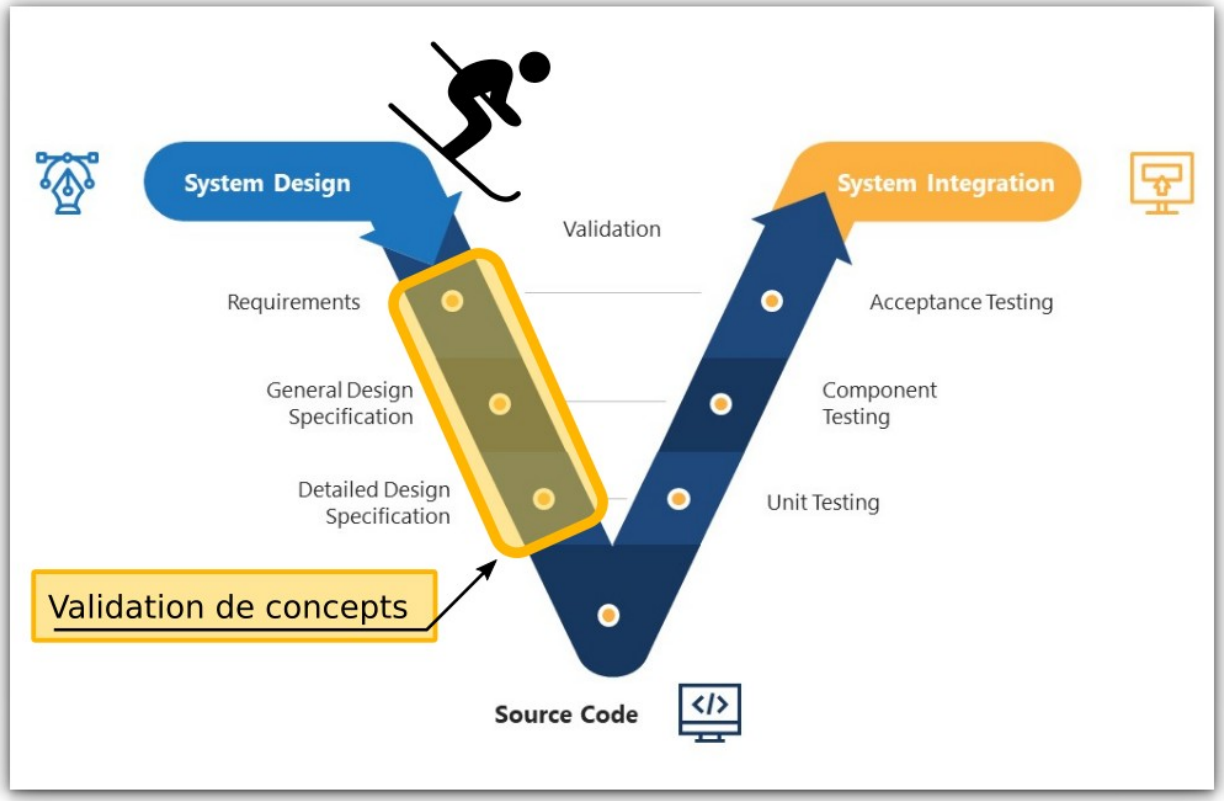


- ethernet
- écran
- GPS
- capteur acoustique
  * ech. 40kHz
  * stockage data : 24 heures
- camera
  * 1080p - 32 bit
  * 30 fps
  * détection de contour temps-réél

- WIFI
- capteur acoustique
  * ech. 400kHz
  * stockage data : 1 sec.
  * détection de pattern temps-réél
- autonomie énergétique
  * batterie 3.7 V
  * 24 heures

# Introduction

Architecture réseau >> contraintes sur les unités de calcul embarquées



# Introduction

## Calcul embarqué, contraintes et solutions sur le marché

Existance de périphériques (écran, réseau, divers capteurs)
Consommation énergétique,
Puissance de calcul embarqué,
Emcombrement,
Ports disponibles,
Temps-réel,
Sécurité
Coût,
Disponibilité des pièces,

PC de bureau
Mini-PC
Nano-ordinateur (typiquement arm)
Cartes µ-contrôleurs
FPGA
SoC FPGA
ASIC

| | Écran | Réseau | Capteurs | Faible énergie | Puissance de calcul | Faible encombrement | Ports disponibles | Temps-réel | sécurité | Faible coût | disponibilité des pièces |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PC de bureau | +++ | +++ | ++ | --- | +++ | --- | - | -- | +++ | --- | ++ |
| Mini-PC | ++ | +++ | ++ | -- | ++ | -- | - | -- | +++ | -- | + |
| Nano-ordinateur (typiquement arm) | + | +++ | +++ | -- | + | -- | + | -- | ++ | - | +++ |
| Cartes µ-contrôleurs | --- | + | +++ | +++ | --- | +++ | ++ | +++ | -- | - | ++ |
| FPGA | -- | -- | + | ++ | + | + | +++ | +++ | -- | - | - |
| SoC FPGA | - | ++ | +++ | --- | +++ | ++ | +++ | +++ | - | -- | -- |
| ASIC | -- | -- | + | ++ | +++ | + | +++ | +++ | ++ | --- | +++ |

# Introduction

## Calcul embarqué, contraintes et solutions sur le marché

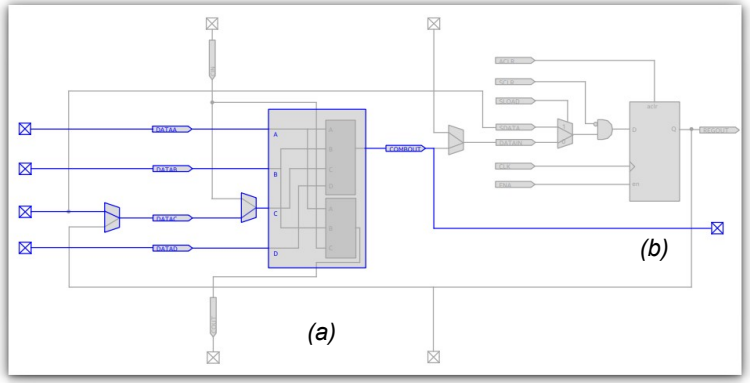
| | Écran | Réseau | Capteurs | Faible énergie | Puissance de calcul | Faible encombrement | Ports disponibles | Temps-réel | sécurité | Faible coût | disponibilité des pièces |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PC de bureau | +++ | +++ | ++ | --- | +++ | --- | - | -- | +++ | --- | ++ |
| Mini-PC | ++ | +++ | ++ | -- | ++ | -- | - | -- | +++ | -- | + |
| Nano-ordinateur (typiquement arm) | + | +++ | +++ | -- | + | -- | + | -- | ++ | - | +++ |
| Cartes µ-contrôleurs | --- | + | +++ | +++ | --- | +++ | ++ | +++ | -- | - | ++ |
| FPGA | -- | -- | + | ++ | + | + | +++ | +++ | -- | - | - |
| SoC FPGA | - | ++ | +++ | --- | +++ | ++ | +++ | +++ | - | -- | -- |
| ASIC | -- | -- | + | ++ | +++ | + | +++ | +++ | ++ | --- | +++ |

**Principaux fabriquants de FPGA**

Intel
Xilinx
Lattice

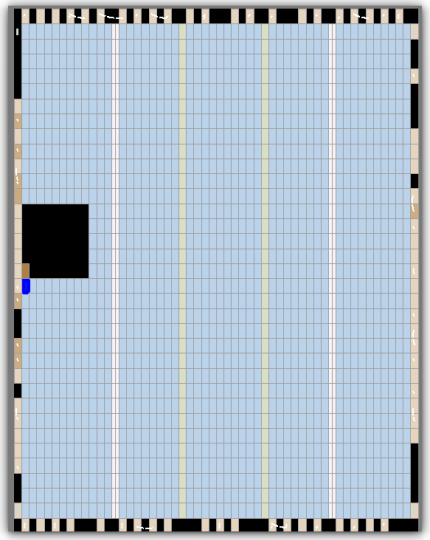
# Calculs sur FPGA

**Un FPGA est un réseau programmable de portes logiques et autres ressources**




*Logic element : (a) LUT ; (b) Registre*

*Bloc logique*



*FPGA (Cyclone IV E) : 22k L.E.*

*DE0-nano*
*$100*
*Cyclone IV E seul*
*$30*

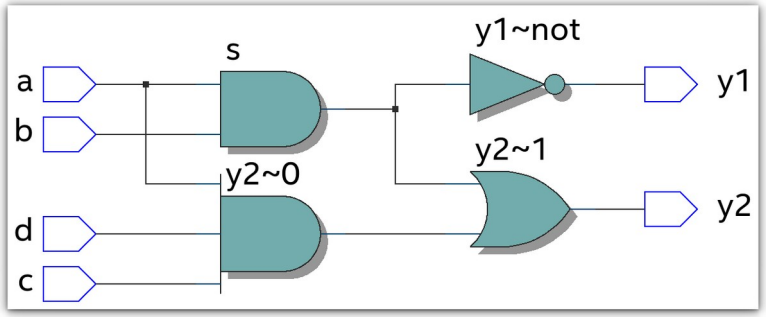
# Introduction

## Calculs sur FPGA

**Il existe des langage de programmation de haut niveau dédiés à la configuration de tels réseaux**

```
2   library ieee;
3   use ieee.std_logic_1164.all;
4
5   entity combi_arch is
6      port (
7         a, b, c, d    : in std_logic ;
8         y1, y2 : out std_logic
9      );
10  end entity;
11
12  architecture rtl of combi_arch is
13     signal s : std_logic ;
14  begin
15     s  <= a and b ;
16     y1 <= not s ;
17     y2 <= (a and c and d) or s ;
18  end architecture ;
19
```

*Description VHDL d'un circuit*




*Description RTL du même circuit*

# Introduction

## Calculs sur FPGA

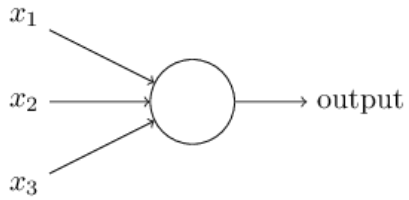
**Traitement du signal**

Software-defined radio (SDR)
Audio / Video temps-réel

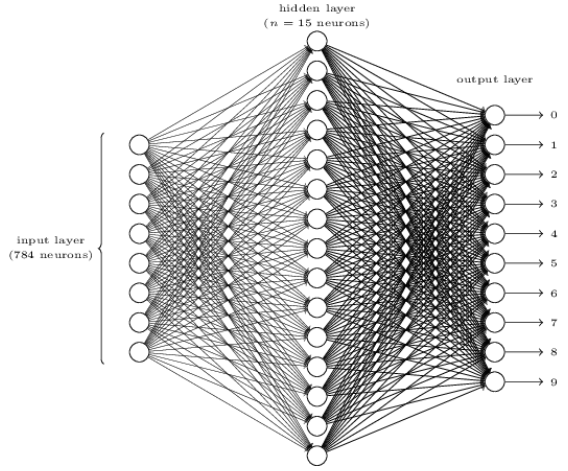
**Processus parallélisables**

Réseau de neurones artificiels (Deep Learning)
Minage de crypto-monnaies

(a)




(b)




(c)

*(a) MNIST jeu de donnée pour l'entrainement d'un algorithme de deep-learning ; (b) architecture d'un perceptron ; (c) exemple simple de classifieur. http://neuralnetworksanddeeplearning.com/chap1.html*

# Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal

**Introduction pour les non-développeurs FPGA**

Sinon rendez-vous à la page 24.

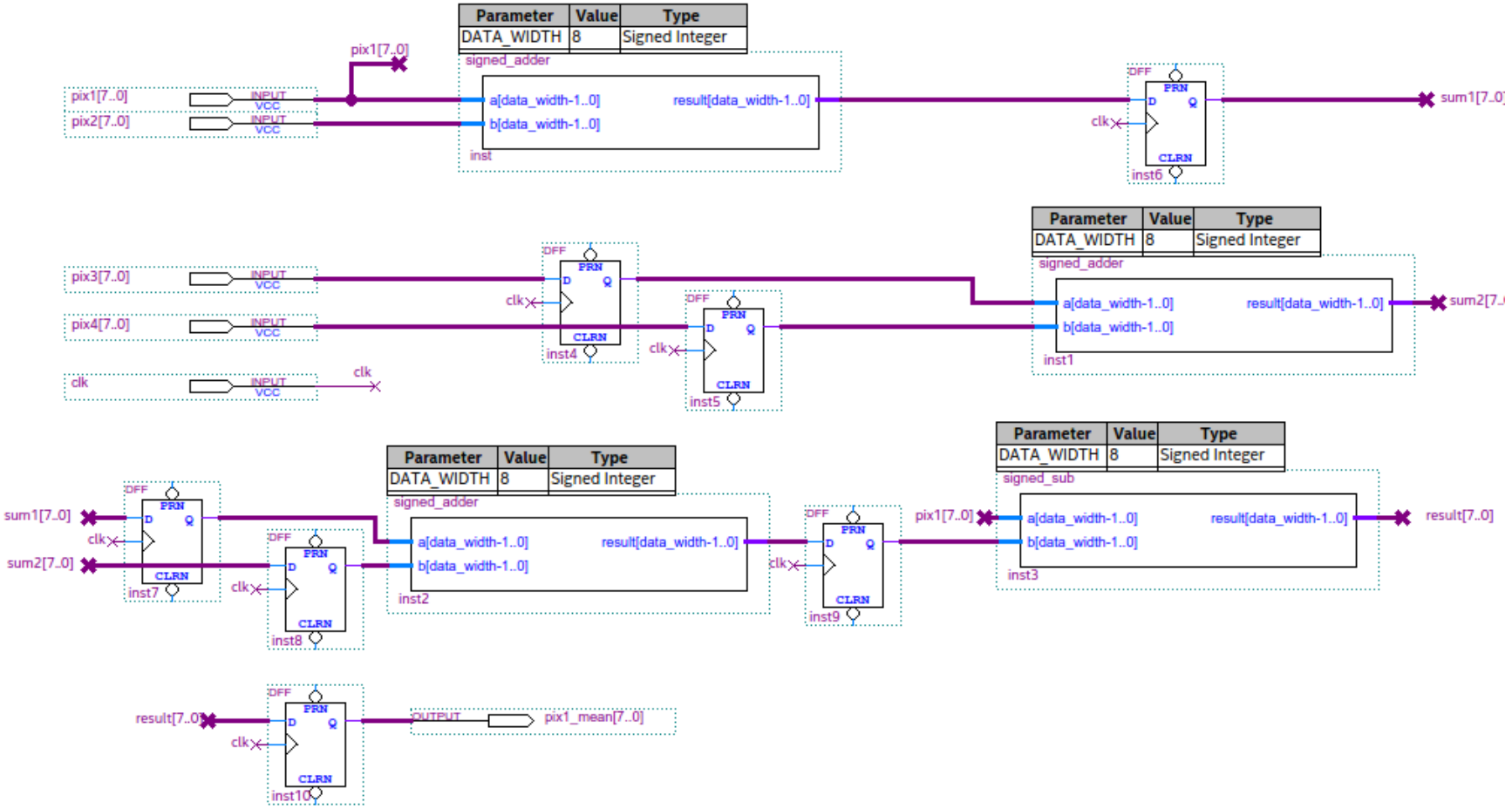
# Comment programmer un SoC FPGA chez Intel

Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal

```c
/*
 * main.c
 *
 *  Created on: 22 avr. 2022
 *      Author: jmecodol
 */

#include <stdio.h>
#include <stdlib.h>

signed char pix1 = 20 ;
signed char pix2 = -5 ;
signed char pix3 = 30 ;
signed char pix4 = -9 ;
signed char sum  =  0 ;
signed char result = 0 ;

int main(int argc, char **argv) {
    while (1) {
        sum    = pix1 + pix2 + pix3 + pix4 ;
        result = pix1  - sum   ;
    }
    return 0 ;
}
```

# Comment programmer un SoC FPGA chez Intel

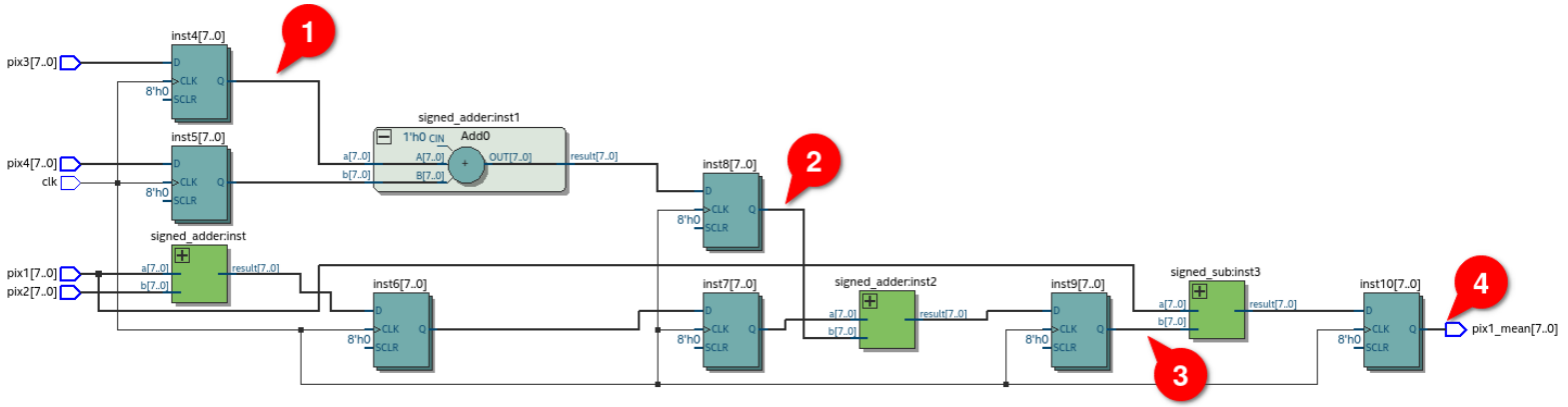
## Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal




*Projet Quartus :*
*<captronic_formation_fpga_img_proc> /quartus_proj/simple_1/*

# Comment programmer un SoC FPGA chez Intel

## Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal




Slow 1100mV 100C Model Fmax Summary

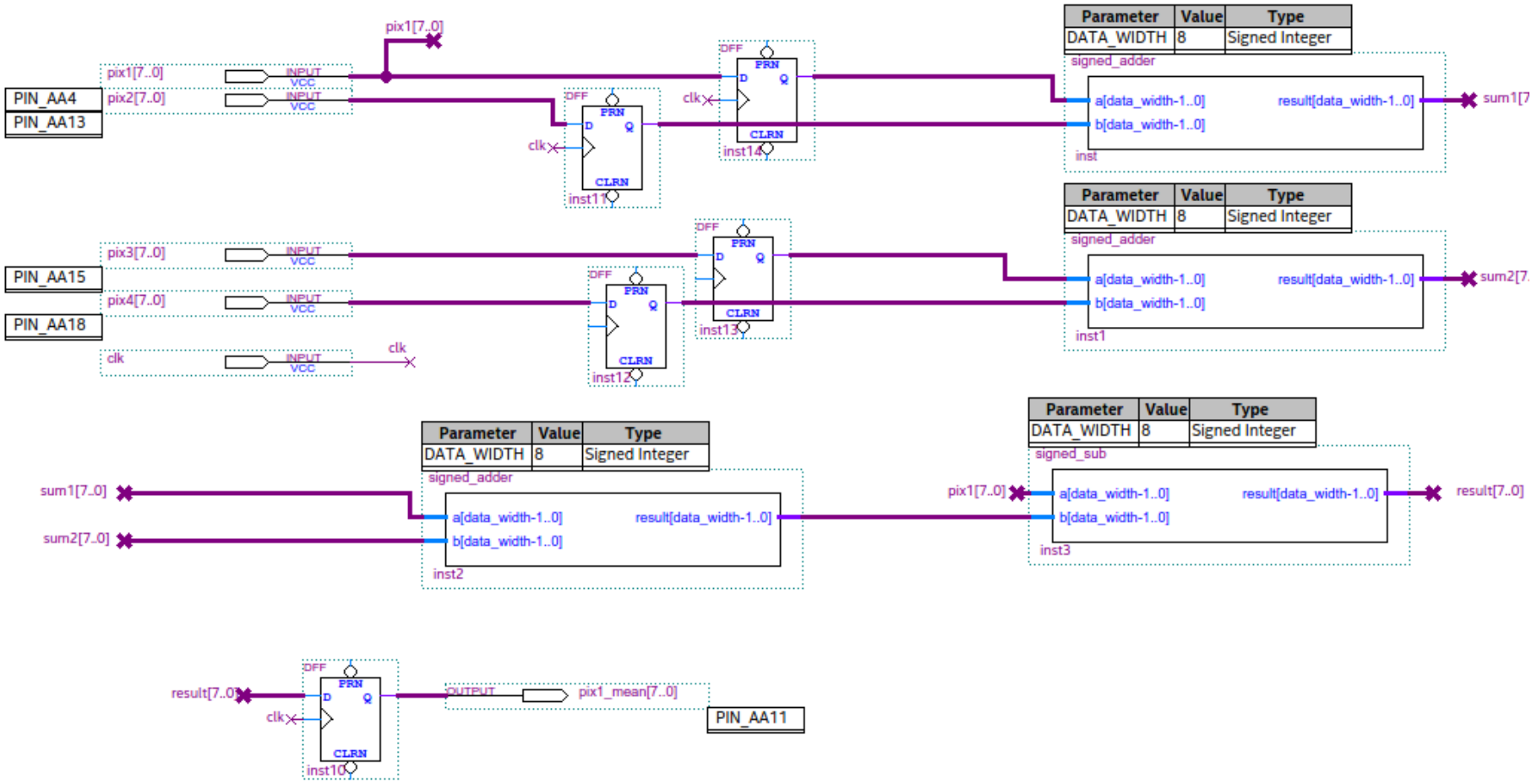
<<Filter>>

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 424.81 MHz | 424.81 MHz | clk | |

Traitement : non-pipeline : 420/4 ~ **100 M.** cycles / secondes
Pipeline : 420 M. cycles / secondes

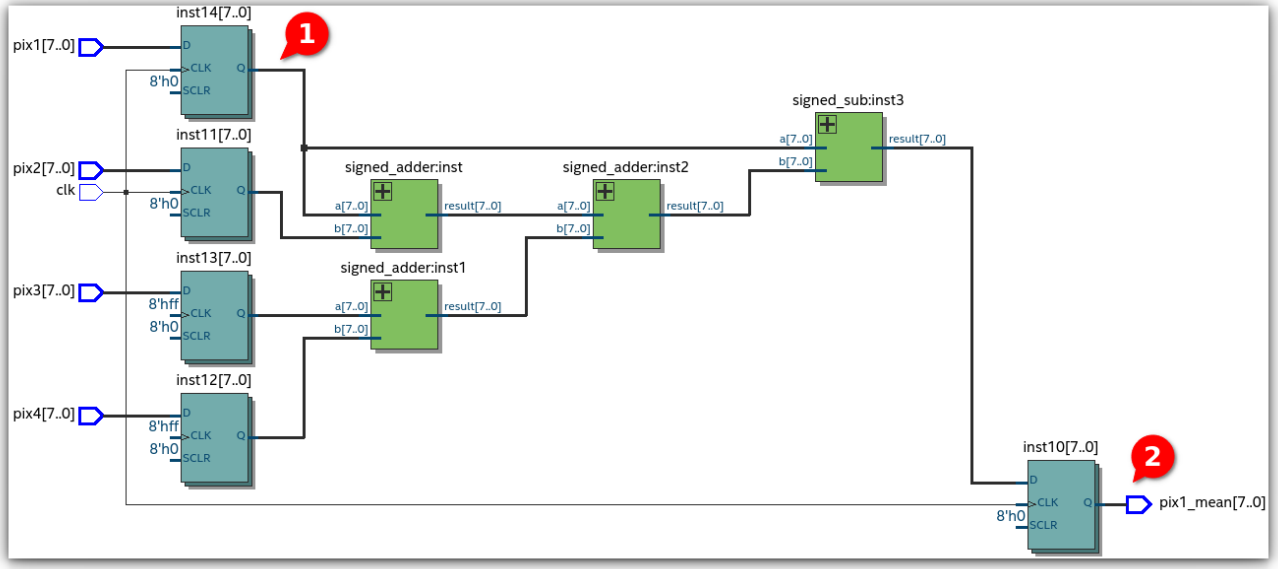
# Comment programmer un SoC FPGA chez Intel

Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal



# Comment programmer un SoC FPGA chez Intel

## Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal




Slow 1100mV 100C Model Fmax Summary

<<Filter>>

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 268.74 MHz | 268.74 MHz | clk | |

```
Traitement : non-pipeline : 420/4.=105 cycles / secondes
             Pipeline     : 420 M.     cycles / secondes
             Regroupé     : 268 M.     cycles / secondes
```

# Comment programmer un SoC FPGA chez Intel

Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal

**Conclusions #1 :**
Un FPGA peut **supprimer des séquences**
Un FPGA peut **créer des pipelines**

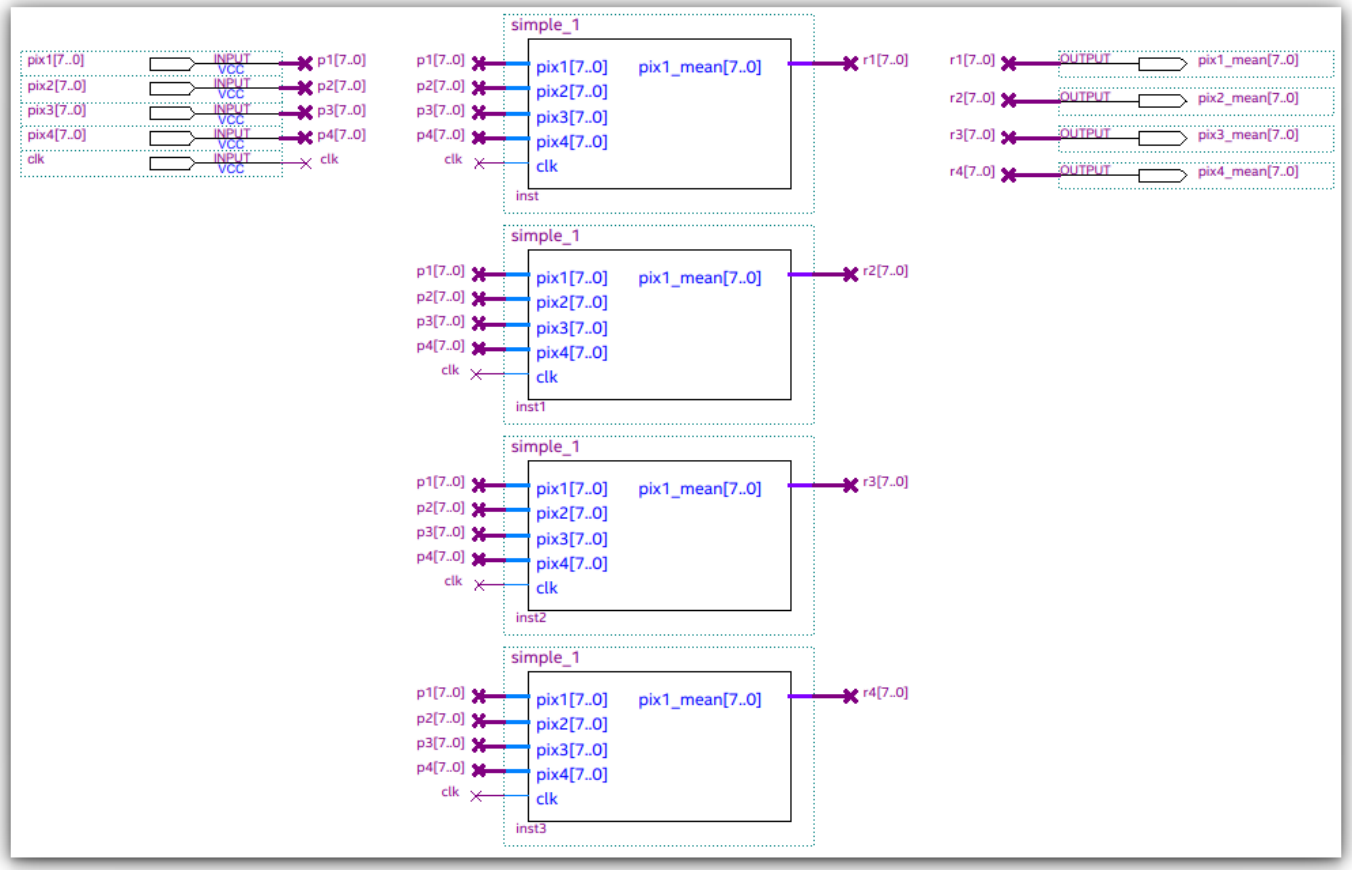
# Comment programmer un SoC FPGA chez Intel

Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal

```
11 signed char pix1_1 = 20 ;
12 signed char pix2_1 = -5 ;
13 signed char pix3_1 = 30 ;
14 signed char pix4_1 = -9 ;
15 signed char pix1_2 = 20 ;
16 signed char pix2_2 = -5 ;
17 signed char pix3_2 = 30 ;
18 signed char pix4_2 = -9 ;
19 signed char pix1_3 = 20 ;
20 signed char pix2_3 = -5 ;
21 signed char pix3_3 = 30 ;
22 signed char pix4_3 = -9 ;
23 signed char pix1_4 = 20 ;
24 signed char pix2_4 = -5 ;
25 signed char pix3_4 = 30 ;
26 signed char pix4_4 = -9 ;
27 signed char sum_1  =  0 ;
28 signed char sum_2  =  0 ;
29 signed char sum_3  =  0 ;
30 signed char sum_4  =  0 ;
31 signed char result_1 = 0 ;
32 signed char result_2 = 0 ;
33 signed char result_3 = 0 ;
34 signed char result_4 = 0 ;
35
36 int main(int argc, char **argv) {
37     while (1) {
38         sum_1    = pix1_1 + pix2_1 + pix3_1 + pix4_1 ;
39         result_1 = pix1_1  - sum_1   ;
40         sum_2    = pix1_2 + pix2_2 + pix3_2 + pix4_2 ;
41         result_2 = pix1_2  - sum_2   ;
42         sum_3    = pix1_3 + pix2_3 + pix3_3 + pix4_3 ;
43         result_3 = pix1_3  - sum_3   ;
44         sum_4    = pix1_4 + pix2_4 + pix3_4 + pix4_4 ;
45         result_4 = pix1_4  - sum_4   ;
46     }
47     return 0 ;
48 }
```

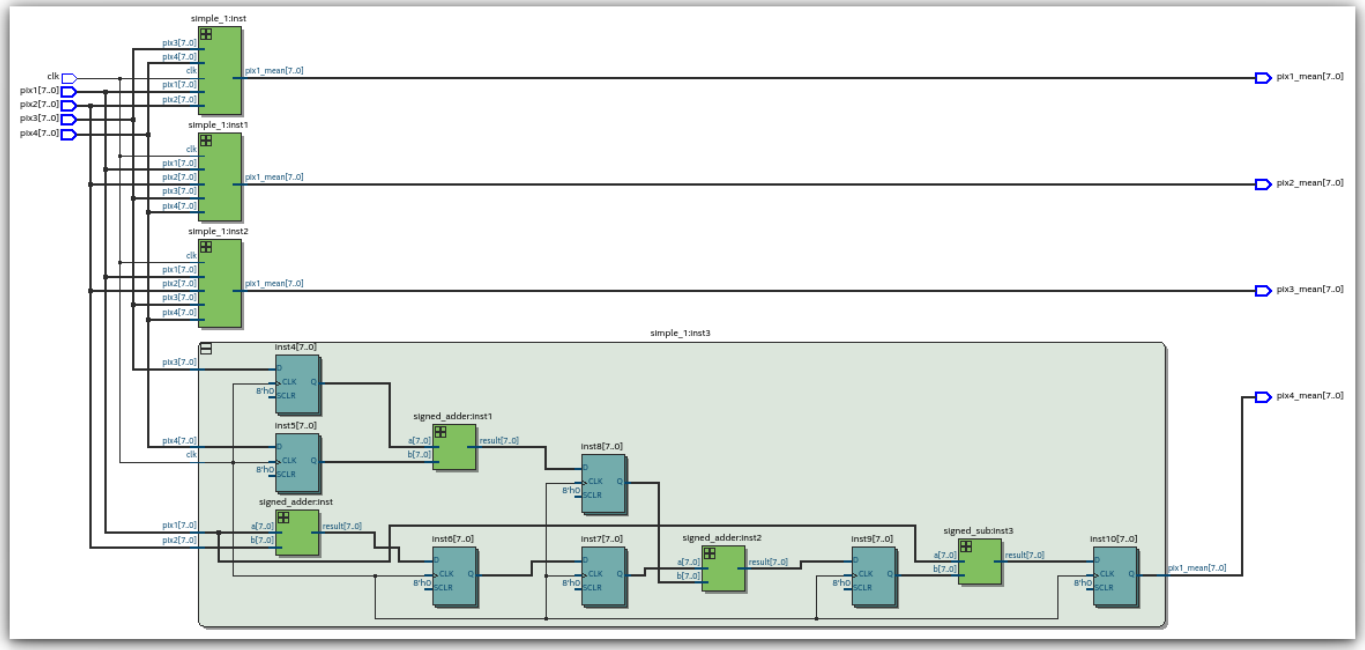
# Comment programmer un SoC FPGA chez Intel

Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal



# Comment programmer un SoC FPGA chez Intel

Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal




**Slow 1100mV 100C Model Fmax Summary**

<<Filter>>

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 341.65 MHz | 341.65 MHz | clk | |

```
Traitement : sequences              : 340/4/4*4 = 85   Mcycles / sec.
             Parallisme + pipeline  : 340*4     = 1360 Mcycles / sec.
```

# Comment programmer un SoC FPGA chez Intel

## Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal

```
Conclusions #1 :
 Un FPGA peut supprimer des séquences
 Un FPGA peut créer des pipelines

Conclusion #2 :
 Un FPGA parallise : exemple d'un gain de 16

Sur l'exemple précédent :
Sur Cyclone v de la carte DE10-nano :
 > Freq CPU  : 1GHz
 > Freq FPGA : 50MHz => PLL => 340 Mhz

Le FPGA apporterait une puissance de calcul équivalente à
 > Un second CPU à 100% des capacités
 > Tournant à 340MHz * 16 = 5,4 GHz
```

# Comment programmer un SoC FPGA chez Intel

## Comment un FPGA réalise l'implémentation d'un algorithme de traitement du signal

Fin d'introduction pour les non-développeurs FPGA

Plan

JOUR 1

§001 Faire communiquer Linux avec un FPGA sur le SoC intel DE10-nano

- distribution fournie par Altera
- créer une image SD (en partie sur un serveur distant)
- compilation croisée (eclipse CDT embedded)
- activer les bridges HPS <-> FPGA par un device tree
- utiliser le bridge HPS2FPGA

J 2

§002 Partager une zone de RAM entre Linux et un FPGA sur le SoC intel DE10-nano

- projet de lecture/écriture en RAM sur FPGA
- projet de lecture/écriture en RAM sur CPU

J 3

§003 Cas pratique avec seuillage d'image

- projet openCV
- échange sur mémoire RAM