Projet FPGA

E2I5 – S10

Projet FPGA

Liliana ANDRADE, Polytech Grenoble

Liliana.Andrade@univ-grenoble-alpes.fr

2022–2023

Projet intégrateur

Prérequis

- Maitrise du VHDL
- Architecture et jeu d'instructions RISC-V
- Programmation C
- Électronique numérique

Enseignants

- **Liliana Andrade** <Liliana.Andrade@univ-grenoble-alpes.fr>

Organisation et évaluation

Enseignements

- 1 séance de CM (2h) + 6 séances de TD/TP (4h)
  - **CM** : 5 février 2023
  - **TD/TP** : fév-juin 2023, CIME-Nanotech, Salle A446
- Attention à la composition des binômes !

Évaluation continue

- **Point d'avancement au début de chaque séance**
  - Présentation des réalisations
  - Questions sur les codes, les opérations et les applications proposées

Historique du projet

- Projet initialement réalisé à l'Ensimag, puis Phelma
  - Processeur MIPS
  - Stéphane Mancini, Olivier Muller (Grenoble-INP)

Projet IDEX Riscless-V (2018-2020)

- Adaptation des enseignements au marché des processeurs (RISC-V minimaliste)
  - Architecture des processeurs (S9)
  - Projet FPGA (S10)
- Partenaires : Polytech Grenoble (UGA) et Ensimag (Grenoble INP)
- 2 ingénieurs, 3 stagiaires, 6 enseignants
- **À Polytech Grenoble** : Liliana Andrade, Frédéric Rousseau

Consignes

Objectif du projet

Réalisation d'un système embarqué complet à base du processeur RISC-V capable d'exécuter une partie ou l'intégralité du jeu d'intructions

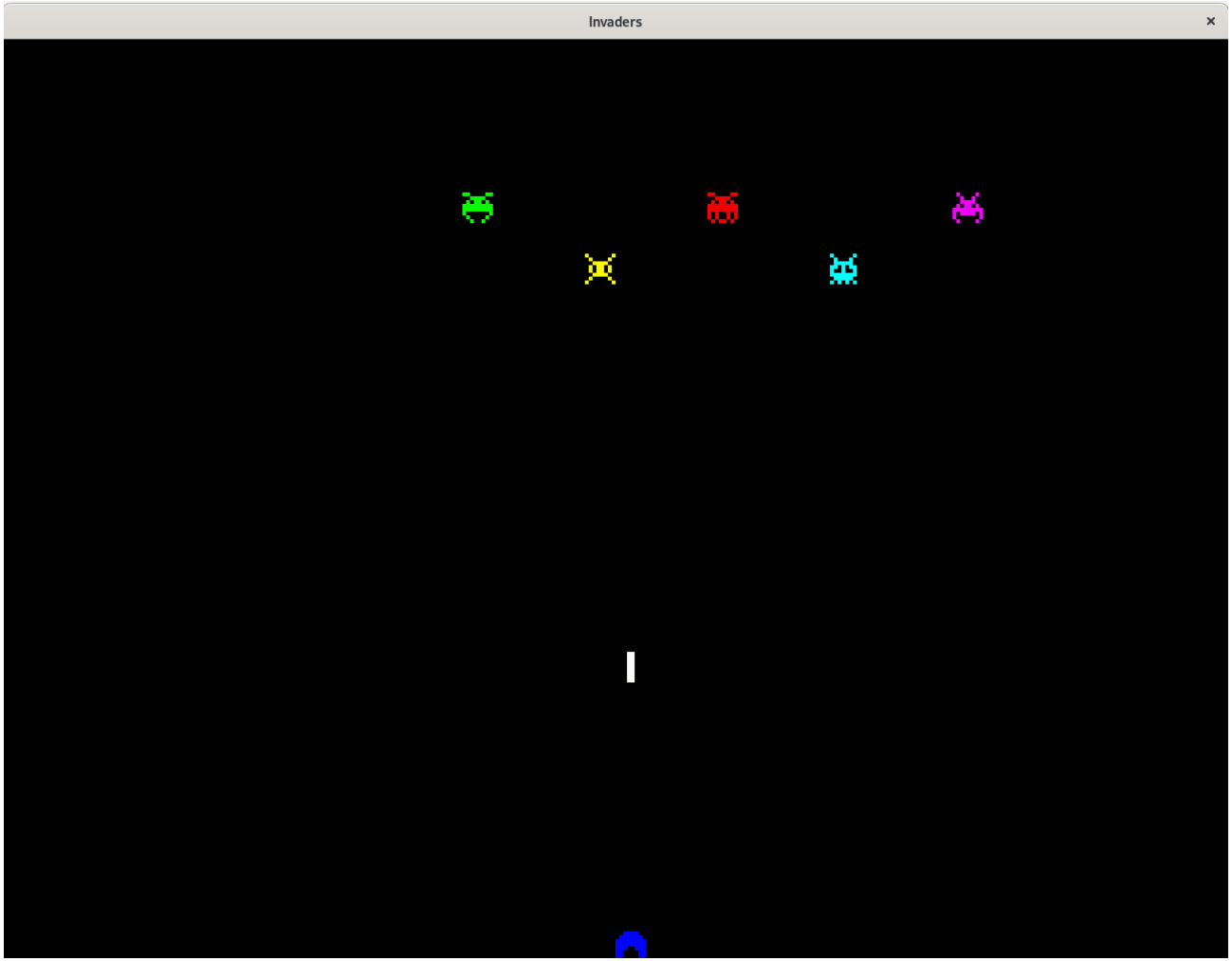
- Architecture matérielle du processeur RISC-V + periphériques
- Approche PC/PO
- Programme C ou binaire s'exécutant sur le processeur RISC-V
- Affichage sur écran (port VGA ou port HDMI)

Compétences requises (ou à acquérir)

- Mise en oeuvre des connaissances acquises dans les modules précédents
- Travail en équipe :
  - Organisaton, découpage des tâches, gestion de l'avancement, intégration des développements multiples
- Intégration logiciel/matériel

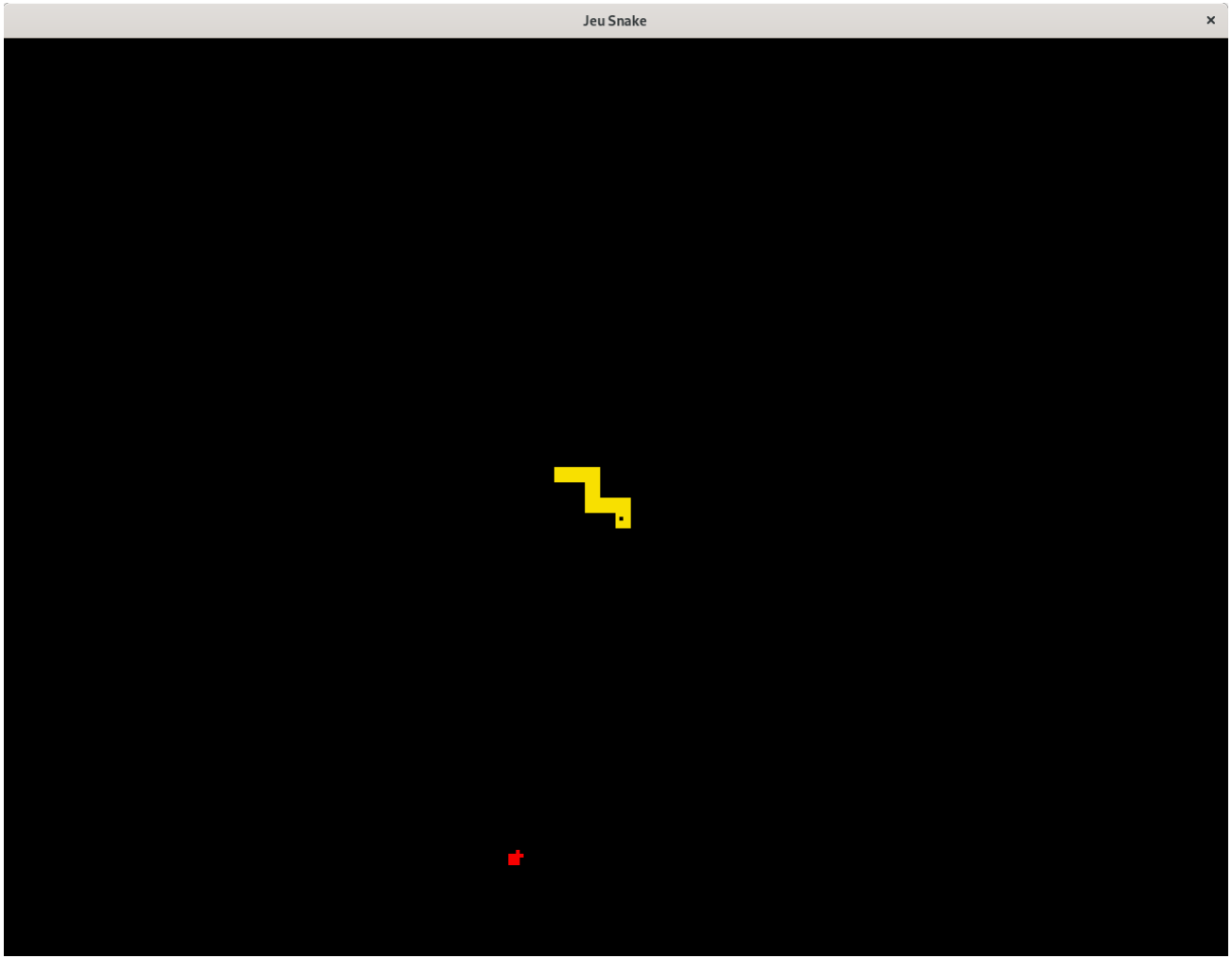
Applications visées

Avec l'intégralité du jeu d'instructions : Invaders



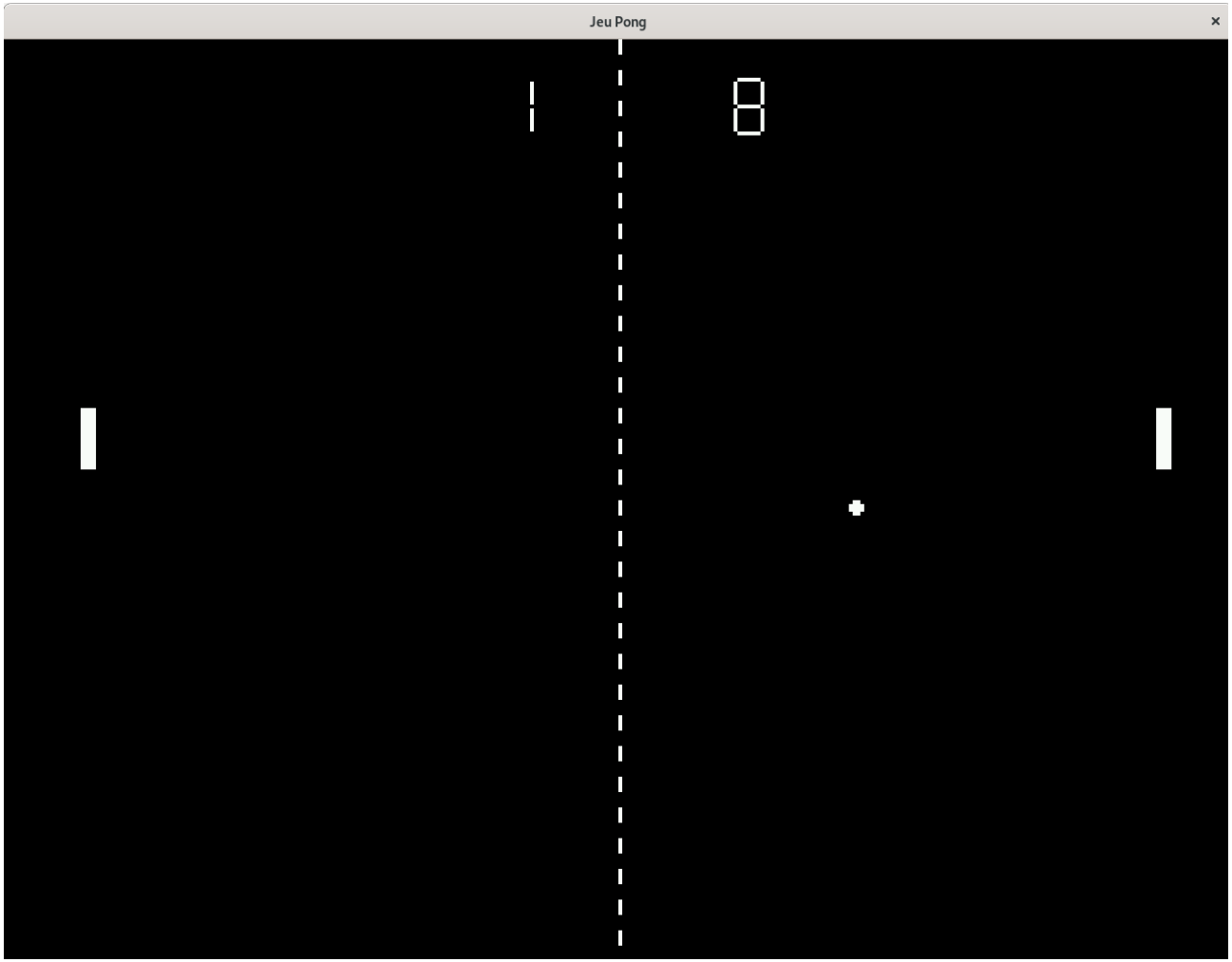
Applications visées

Avec l'intégralité du jeu d'instructions : Snake



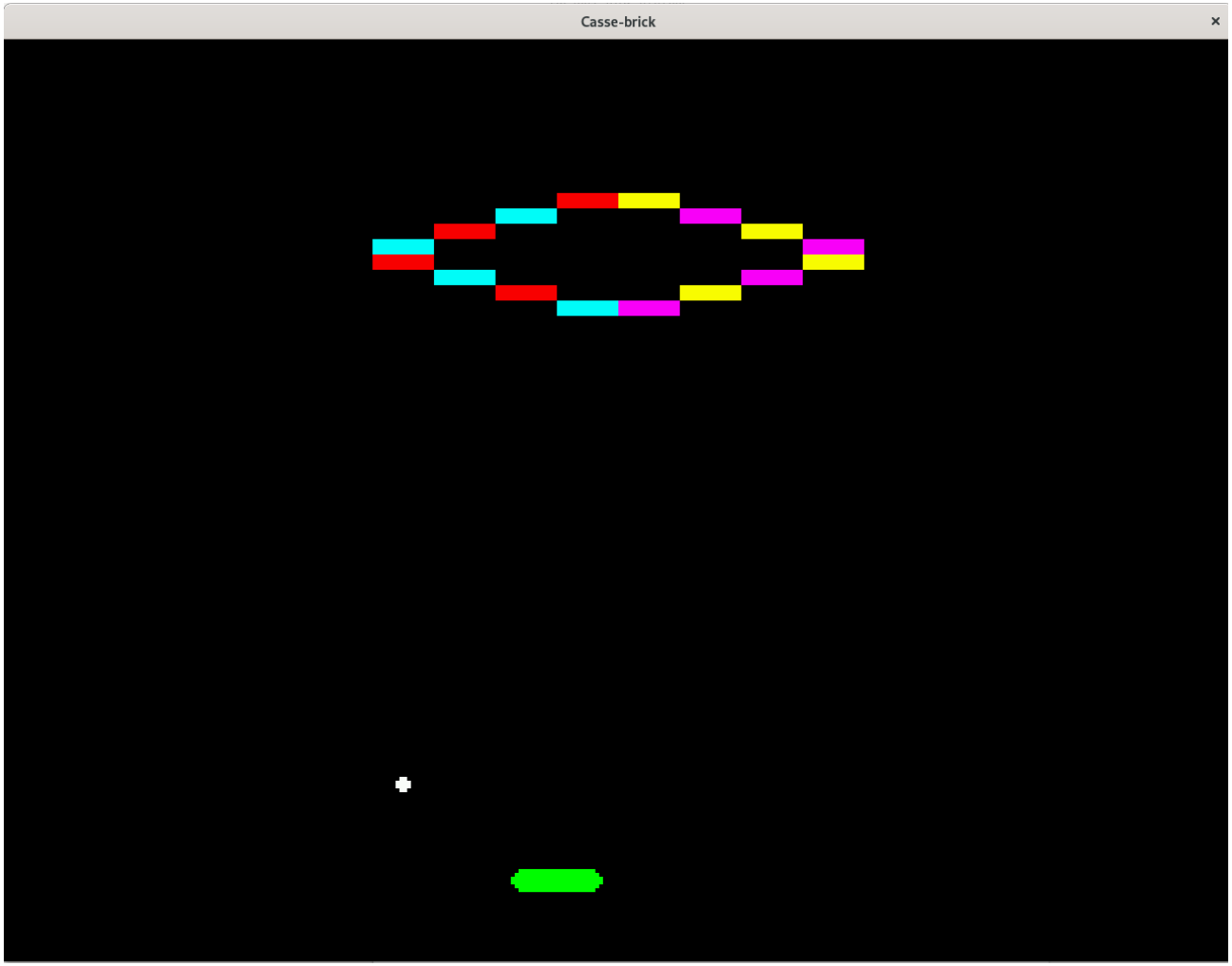
Applications visées

Avec l'intégralité du jeu d'instructions : Pong



Applications visées

Avec l'intégralité du jeu d'instructions : Brick



Architecture globale et simplifications

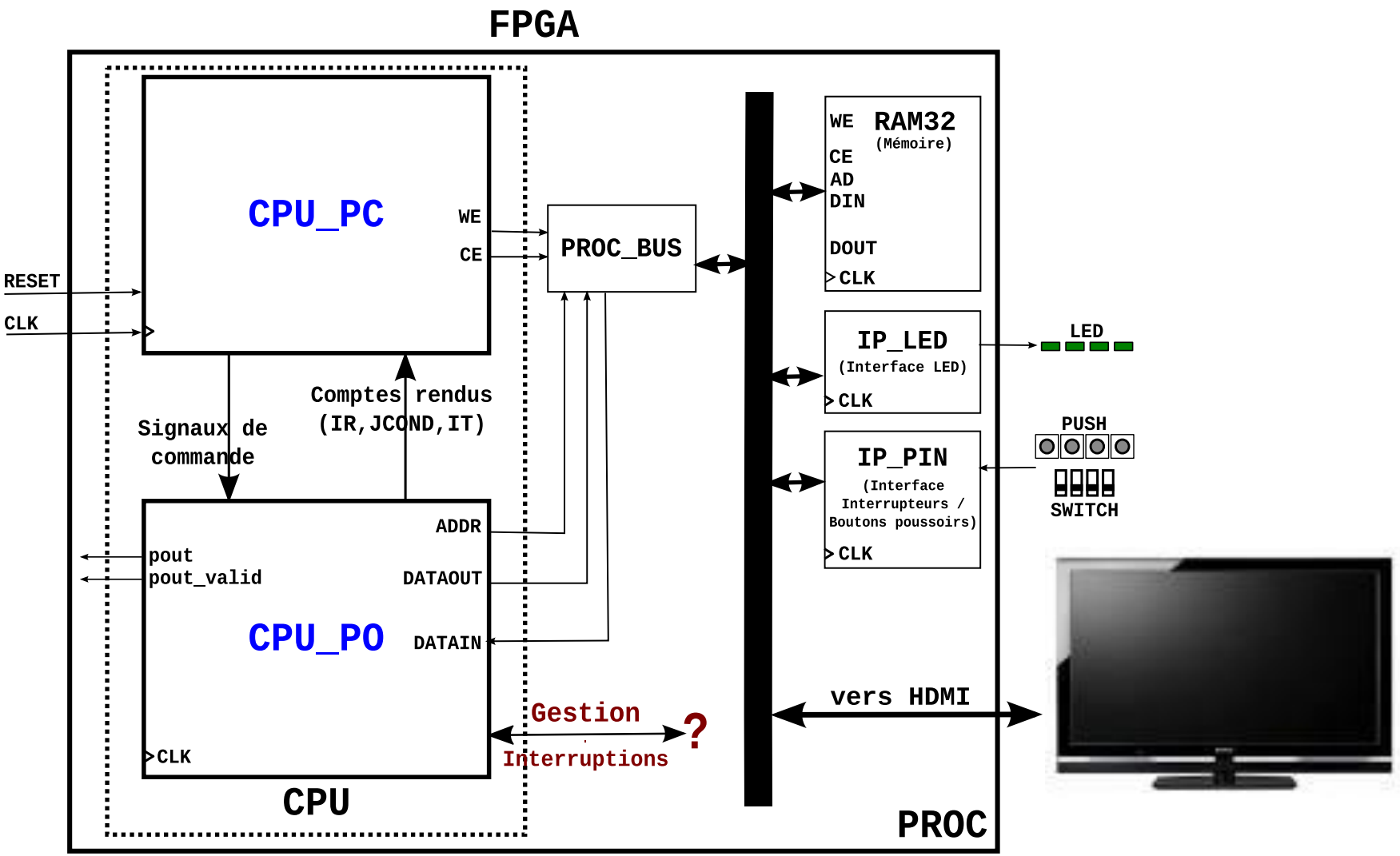
- **1 carte de prototypage FPGA Zybo Zynq-7000 ou Zybo-Z7 Zynq-7020**
  - Un processeur RISC-V (à compléter) + périphériques (fournis)
  - Un programme binaire (ou plusieurs ... selon votre implication !)
- 1 écran connecté à la carte de prototypage (VGA ou HDMI)
- 1 PC pour **développer**, **simuler** et **synthétiser** l'architecture du système embarqué
- Toutes les difficultés liées aux outils de prototypage seront cachées à l'aide de **scripts**
- Dans le temps disponible de l'enseignement, il est impossible de réaliser le projet depuis le début
  - Un certain nombre de modules vous seront donnés pour faciliter les premières mises en œuvre et atteindre les objectifs du projet
  - **Le temps de préparation avant chaque séance est indispensable !**

Travail incrémental

Vous partirez d'une version fonctionnelle d'un système :

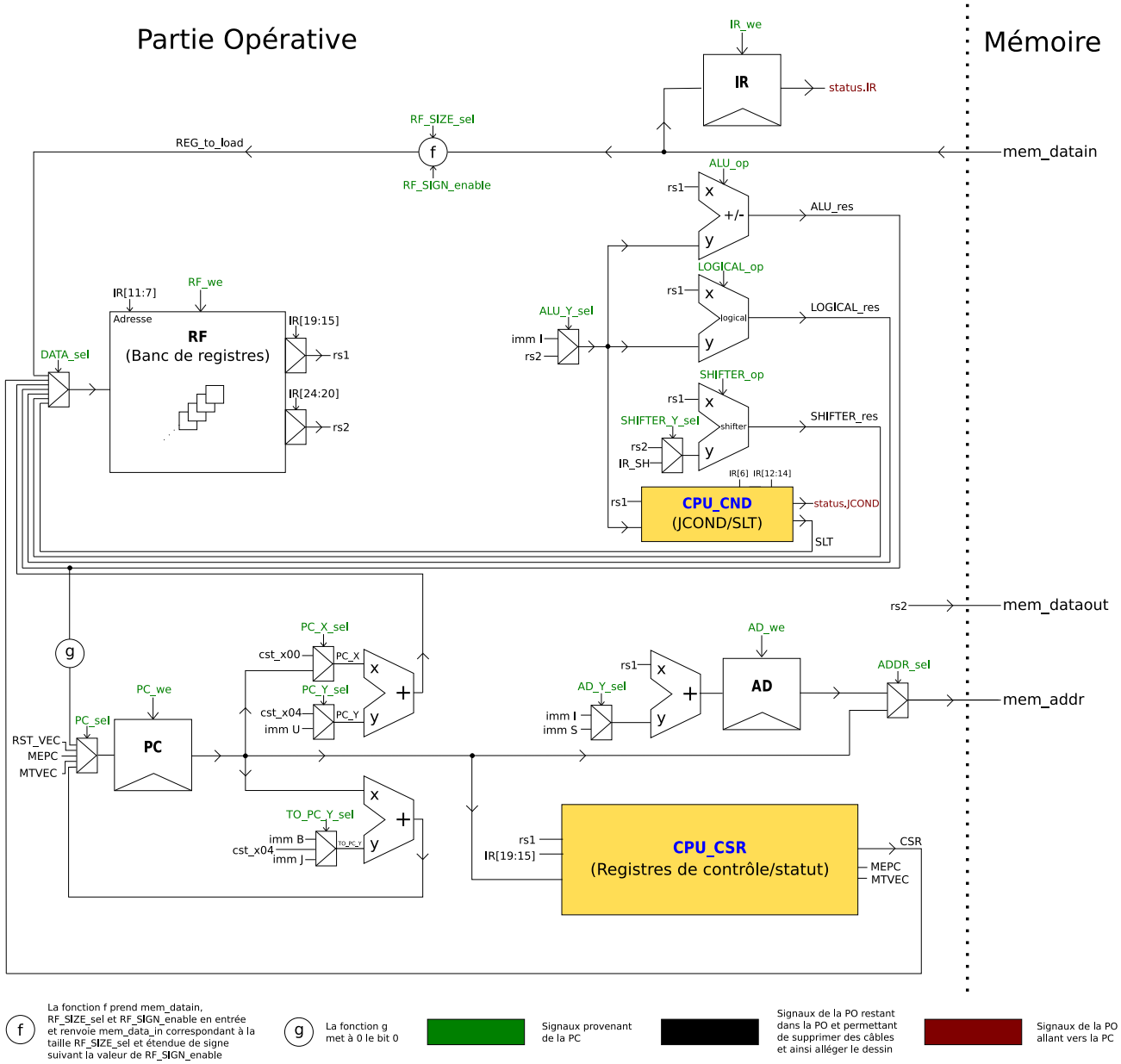
- La plupart des séances seront dédiées à **l'ajout des instructions**
  - La **partie opérative (PO)** du processeur RISC-V est donnée à 80%
  - La **partie contrôle (PC)** est donnée à 10% : Il s'agit de compléter la PC en ajoutant les états nécessaires à l'exécution de toutes les instructions
- Comme le compilateur gcc utilise a priori toutes les instructions disponibles, il faut **implémenter toutes les instructions** pour exécuter les applications données.
- L'ajout de **chaque instruction doit être testé** d'abord par simulation en écrivant un petit programme de test pour cette nouvelle instruction. Ces programmes de test seront écrits en **assembleur**, puis assemblés. Des outils fournis permettront de placer le code binaire dans la mémoire du processeur.
- Après validation par simulation, vous pourrez **valider** l'architecture et vos programme **sur les cartes de prototypage**.

L'architecture du processeur



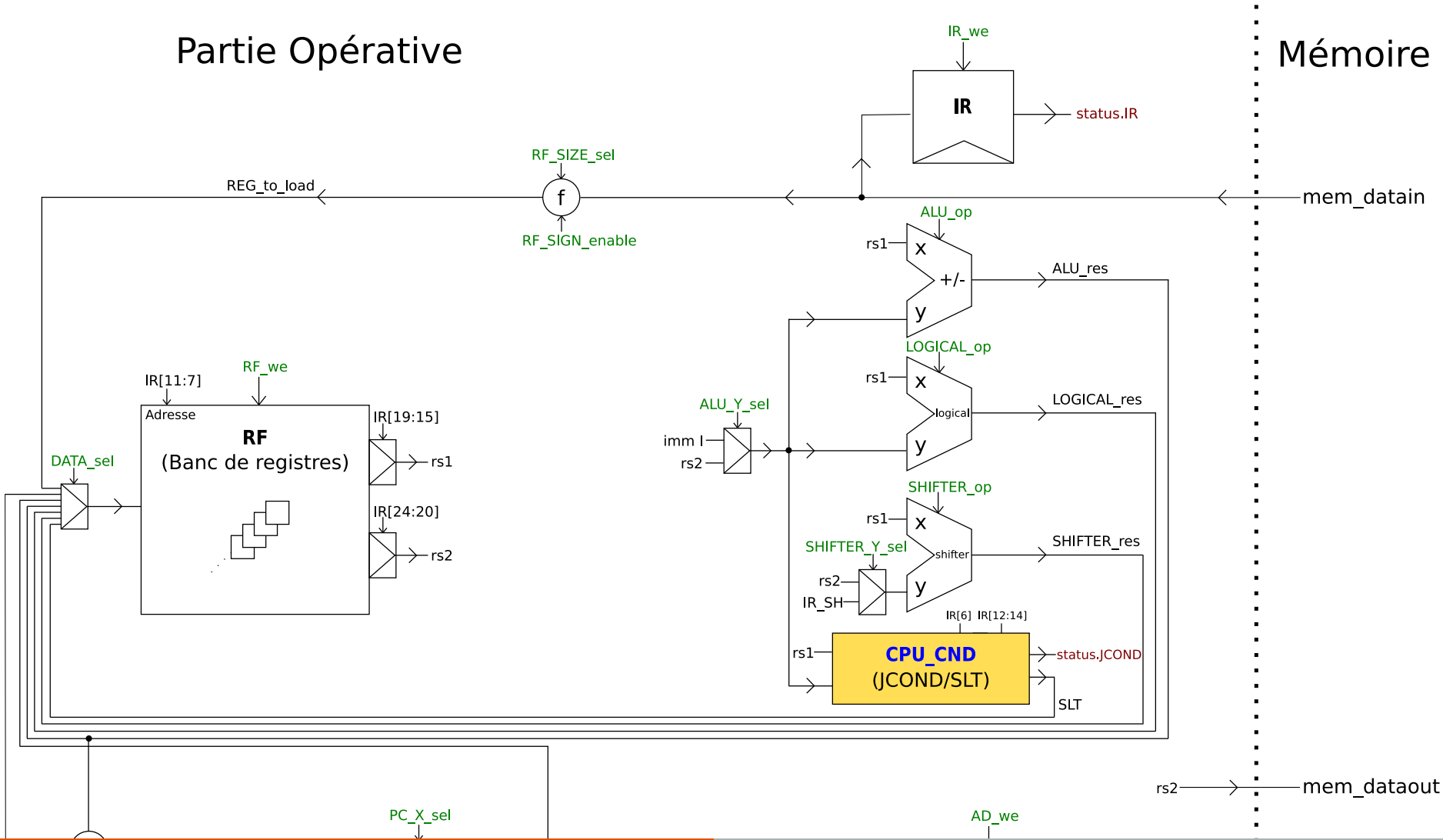
Système complet avec les périphériques sur le bus

L'architecture du processeur — **La partie opérative**



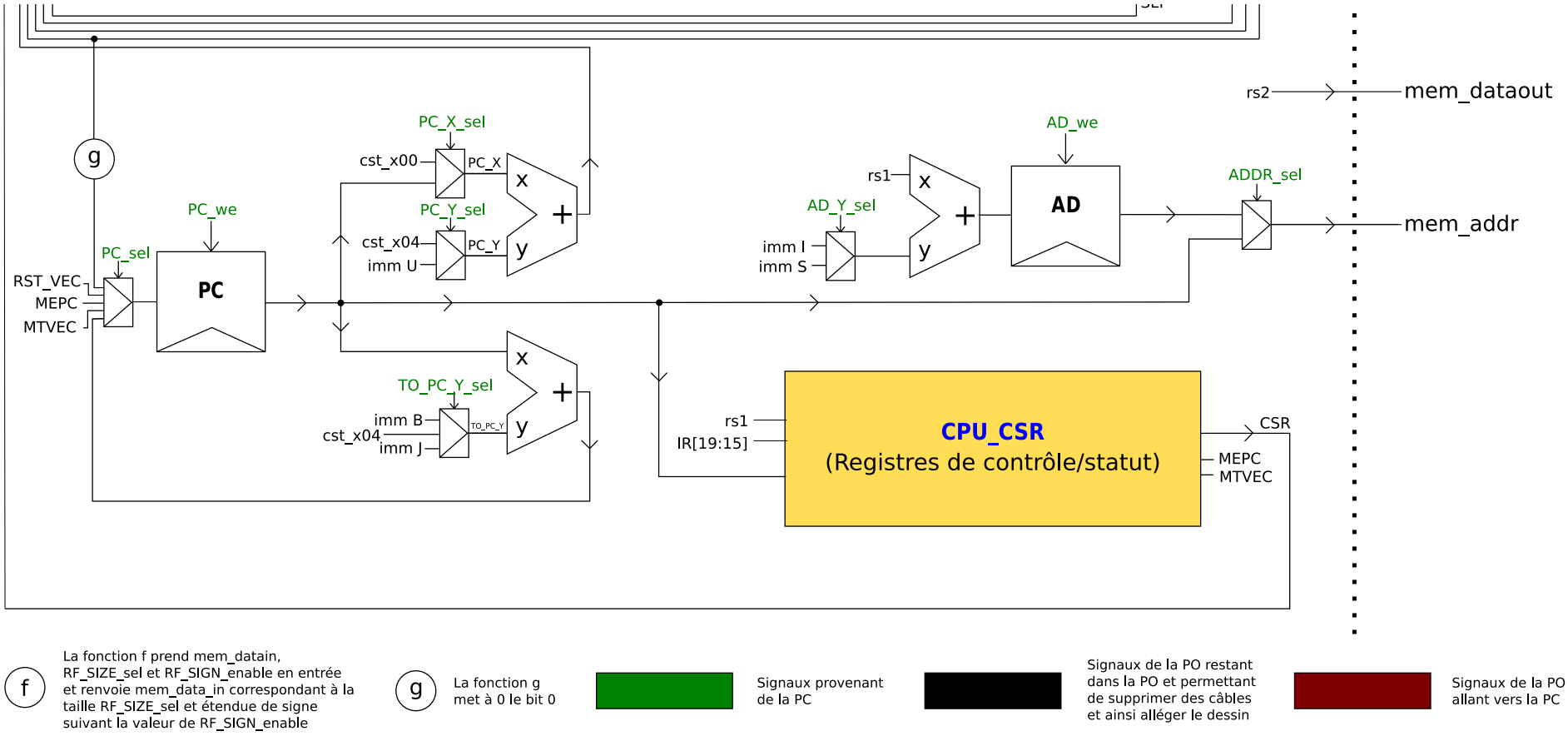
L'architecture du processeur — La partie opérative

Le banc de registres (RF), le registre d'instruction (IR), l'UAL



L'architecture du processeur — La partie opérative

Le compteur du programme (PC) et le registre d'adresse (AD)



f — La fonction f prend mem_datain, RF_SIZE_sel et RF_SIGN_enable en entrée et renvoie mem_data_in correspondant à la taille RF_SIZE_sel et étendue de signe suivant la valeur de RF_SIGN_enable

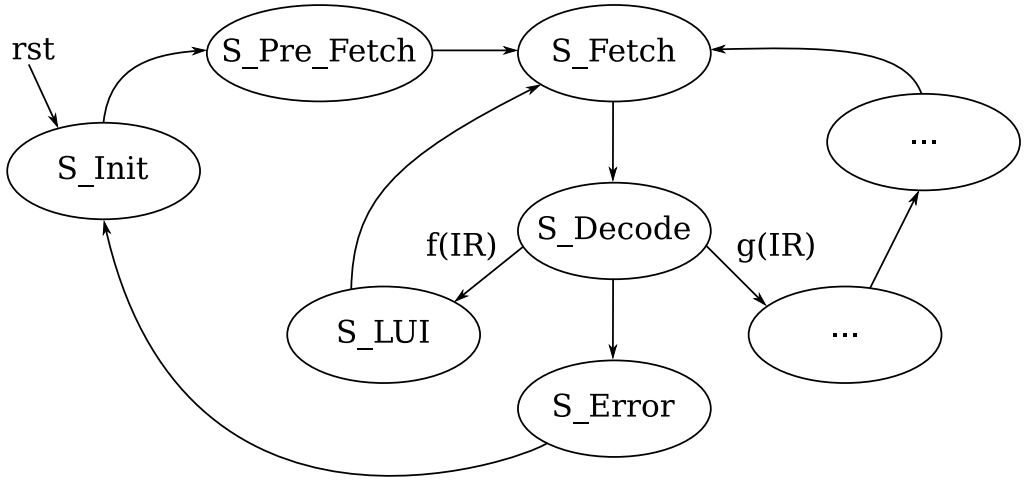
g — La fonction g met à 0 le bit 0

Signaux provenant de la PC

Signaux de la PO restant dans la PO et permettant de supprimer des câbles et ainsi alléger le dessin

Signaux de la PO allant vers la PC

L'architecture du processeur — **La partie contrôle**



États	Opérations entre registres
`S_Init`	`PC` ← `0x1000`
`S_Pre_Fetch`	`mem_addr` ← `mem[PC]` * (émission de `PC` sur le bus d'adresse)
`S_Fetch`	`IR` ← `mem_datain` (réception de `PC` sur le bus de données)
`S_Decode`	`PC` ← `PC+4` † (décodage de `IR`)
`S_LUI`	`RD` ← $IR_{31\ldots12} \| \| 0^{12}$; `mem_addr` ← `mem[PC]` (émission de `PC` sur le bus d'adresse)
...	...

*. Un accès mémoire nécessite un cycle. On demande un accès à la valeur `mem[PC]` qui sera fournie au cycle suivant sur le bus `mem_datain`.

†. Attention, les instructions de branchement et `auipc` n'incrémentent pas `PC` dans cet état.

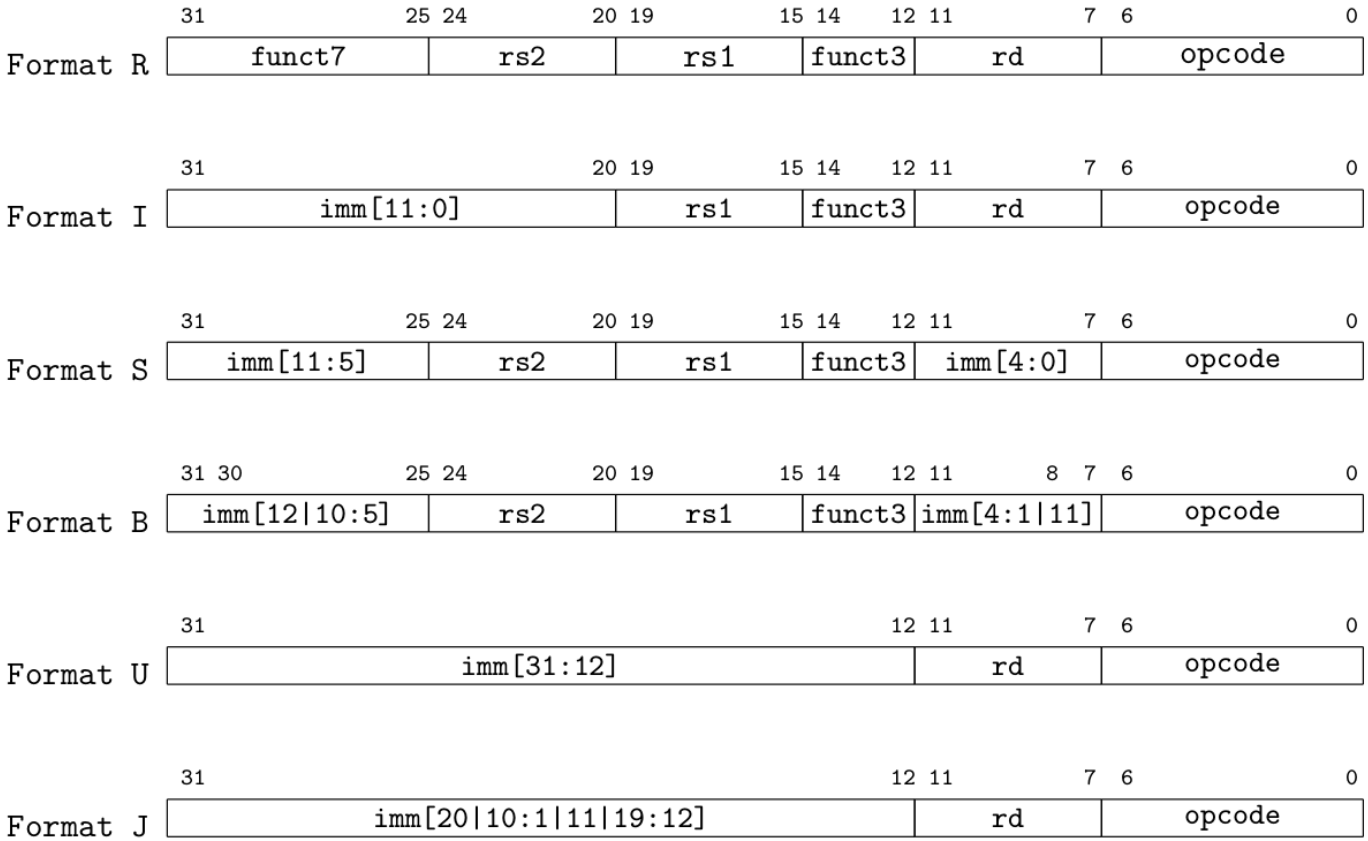
Les instructions RISC-V (RV32IM)

6 formats d'instructions codées sur 32 bits

- **Format R** : opérations registre-registre (`add, xor, ...`)
- **Format I** : opérations immédiates courtes et "loads" (`addi, lw, ...`)
- **Format S** : opération "store" (`sw, sb, sh`)
- **Format B** : opérations de saut conditionnel (`beq, bne, ...`)
- **Format U** : opérations immédiates longues (`lui, auipc`)
- **Format J** : opérations de saut inconditionnel (`jal`)

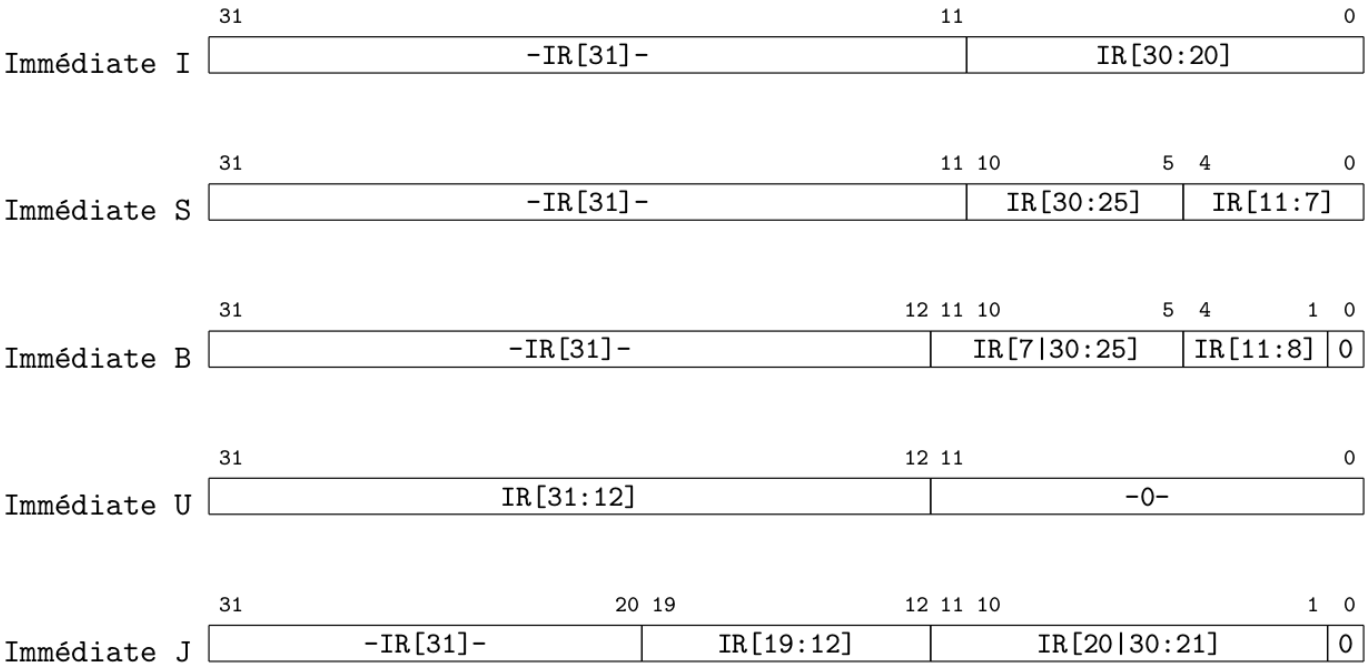
- Le processeur possède **32 registres de 32 bits** (notés `x0, ..., x31`)
- Aucune instruction n'utilise de registre implicite
- Le registre `x0` vaut toujours `'0'`

Les instructions RISC-V (RV32IM) — **Les 6 formats en détail**



- Le code de l'instruction (7 bits de poids faible)
- Les registres `rd`, `rs1`, `rs2` toujours sur les mêmes bits
- **L'immédiat** utilisée pour les décalages, branchements conditionnels, ... est construite de manière assez peu conventionnelle

Les instructions RISC-V (RV32IM) — **Les constantes immédiates**



- 5 types des constantes (toujours étendues en signe) produites à partir de l'instruction (`IR`)
- **Exemples** :
  - 12 bits (directs) pour les op. arithmétiques/logiques impliquant une constante (`immI`)
  - 5 bits (directs) pour les opérations de décalage impliquant une constante (`shamt`)
  - 12 bits (composés) pour les op. `'store'` (`immS`)
  - 12 bits (composés/mélangés) pour les branchements conditionnels (`immB`)
  - 20 bits (directs) pour les op. immédiates longues (`immU`)
  - 20 bits (composés/mélangés) pour les sauts inconditionnels (`immJ`)

Les instructions RISC-V (RV32IM) — **Encodage des instructions**

Sous-ensemble RV32I

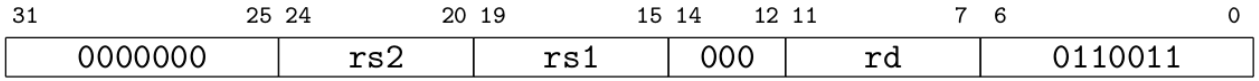
31 25	24 20	19 15	14 12	11 7	6 0	
imm[31:12]				rd	0110111	U lui
imm[31:12]				rd	0010111	U auipc
imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu
imm[11:0]		rs1	000	rd	0000011	I lb
imm[11:0]		rs1	001	rd	0000011	I lh
imm[11:0]		rs1	010	rd	0000011	I lw
imm[11:0]		rs1	100	rd	0000011	I lbu
imm[11:0]		rs1	101	rd	0000011	I lhu
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	S sb
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	S sh
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	S sw
imm[11:0]		rs1	000	rd	0010011	I addi
imm[11:0]		rs1	010	rd	0010011	I slti
imm[11:0]		rs1	011	rd	0010011	I sltiu
imm[11:0]		rs1	100	rd	0010011	I xori
imm[11:0]		rs1	110	rd	0010011	I ori
imm[11:0]		rs1	111	rd	0010011	I andi
0000000	shamt	rs1	001	rd	0010011	I slli
0000000	shamt	rs1	101	rd	0010011	I srli
0100000	shamt	rs1	101	rd	0010011	I srai
0000000	rs2	rs1	000	rd	0110011	R add
0100000	rs2	rs1	000	rd	0110011	R sub
0000000	rs2	rs1	001	rd	0110011	R sll
0000000	rs2	rs1	010	rd	0110011	R slt
0000000	rs2	rs1	011	rd	0110011	R sltu
0000000	rs2	rs1	100	rd	0110011	R xor
0000000	rs2	rs1	101	rd	0110011	R srl
0100000	rs2	rs1	101	rd	0110011	R sra
0000000	rs2	rs1	110	rd	0110011	R or
0000000	rs2	rs1	111	rd	0110011	R and

Les instructions RISC-V (RV32IM) — **Exemple**

> Extrait du cahier de charges ...

— add —

encodage



action

Addition registre registre signée.

syntaxe

`add rd, rs1, rs2`

description

Les contenus des registres `rs1` et `rs2` sont ajoutés pour former un résultat sur 32 bits qui est placé dans le registre `rd`.

opération

$\texttt{rd} \leftarrow \texttt{rs1} + \texttt{rs2}$

format R

Quel est le travail a fournir ?

Les instructions RISC-V (RV32IM) — **Méthode**

1. **Identifier l'instruction à implémenter** (CdC et doc. RISC-V)
   - **Format** et encodage
   - Besoin d'une **constante immédiate** et sa construction
   - **Opération** arithmétique/logique ou autres à effectuer

2. **Mettre en oeuvre l'instruction** (code VHDL)
   - **Décoder** l'instruction dans la `PC`
   - Identifier les signaux, registres et unités de calcul à interconnecter/commander dans la `PO`
   - Ajouter un **nouvel état** dans la `PC` (mise à jour des signaux de contrôle commandant la `PO`)

Les instructions RISC-V (RV32IM) — **Méthode**

3. **Tester l'instruction en simulation** (code assembleur)
   - Écrire un programme de **test en langage d'assemblage** (`.s`)
   - Lancer la simulation (à l'aide de scripts fournis)
   - Valider le comportement sur XSIM (Xilinx Vivado 2019.1)

4. **Tester l'instruction sur carte FPGA (Zybo ou Zybo Z7)**
   - Seulement si l'instruction à été validé en simulation
   - Si l'instruction le permet !

Le cahier de charges

> **Séance 1** (2h30) : lecture du CdC, tutoriel GIT en autonomie, gestion du dépôt GIT, début de l'analyse du code ...

Séance	Instruction typique	Famille	Programme à écrire et/ou tester
1 (1h30)	Analyse du code + décodage initial `lui`, `addi`		 `allumer_led.s`

2-3 (8h)	`add`
	
	`sll`
	`auipc`
		`and`, **`or`**, `ori`, `andi`,
		`xor`, `xori`, `sub`
		`srl`, `sra`, `srai`, `slli`, `srli`
			`compteur.s`
			
			`chenillard_minimaliste.s`

4-5 (8h)	PO : composant `CPU_CND`
	`beq`
	`slt`
		
		`bne`, `blt`, `bge`, `bltu`, `bgeu`
		`slti`, `sltu`, `sltiu`
			
			`chenillard_rotation.s`

6 (4h)	`lw`, `sw` `jal`	 `jalr`	 `invader, pong, snake, brick`
Optionnel	Nouvelle application		Jeu ?

Outils et ressources à votre disposition

Simulation

- **Simulateur XSIM** de Xilinx Vivado Design Suite (2019.1)
- Scripts de simulation fournis

Synthèse

- **Xilinx Vivado Design Suite** (2019.1)
- Toute la synthèse sera gérée grâce à des scripts fournis
- Vous n'aurez pas à manipuler directement l'interface graphique (tout est fait dans le terminal)

Carte de développement à utiliser

- ZYBO Z7 : Xilinx Zynq XC7Z020-1CLG400C

`https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/start`

Les connaissances requises

- **VDHL : simulation et synthèse**
  - Lecture du code VHDL donné, modification de la machine d'état, modification des périphériques, mise en œuvre des outils de simulation et synthèse
- **Architecture des processeurs**
  - Architecture matérielle du RISC-V (en VHDL)
  - Jeu d'instructions, programmation en assembleur ou en C
- **Électronique numérique**
  - Modification ou ajout de fonctionnalités (décodeur 7 segments) (extensions)
  - Mise en œuvre des principes fondamentaux de l'électronique (gestion des bus, fonctionnement des périphériques, étude des chronogrammes pour valider le fonctionnement)
  - Principe et fonctionnement des FPGA
- **Pour se mettre à jour**
  - VHDL : Module de cours - TP
  - RISC-V : reprendre cours - TD - TP
  - Électronique numérique
    - Supposé acquis pour les principes de base
    - FPGA : Cours à suivre, de nombreux sites internet et tutoriels, `http://www.vicilogic.com` mis en place par NUI Galway

Documentation

- L'organisation en charge du maintien du standard et de la norme VHDL : `https://www.accellera.org/downloads/ieee`
- La norme du VHDL : IEEE Standard VHDL LRM
- Un site web très complet sur le VHDL : Hamburg VHDL archive
- La fameuse bible de la syntaxe du VHDL, le "VHDL-Cookbook" : VHDL-Cookbook
- Le site web officiel du RISC-V : `https://riscv.org/`
- Les livres autour du RISC-V : `https://riscv.org/risc-v-books/`
- Les wikipédia sur les sujets : VHDL, RISC-V (Attention : wikipedia n'est pas exemple d'erreurs !)

Travail à réaliser avant le début des TP

> Avant jeudi 23 février

- Activer votre compte gricad-gitlab :
  `https://gricad-gitlab.univ-grenoble-alpes.fr/`
- Envoyer à votre enseignant la composition des binômes et les logins associés à vos comptes

> Pendant la première séance de TD/TP (23 février 2023)

- Suivre un tutoriel de GIT :
  `https://git-scm.com/book/fr/v2/`
  Vous pouvez suivre les 3 premiers chapitres uniquement
- Selon vos besoins, suivre les tutoriaux Vicilogic :
  `https://www.vicilogic.com/`
  - RISC-V Processor Architecture and Applications
  - Digital Systems Design & FPGA Prototyping

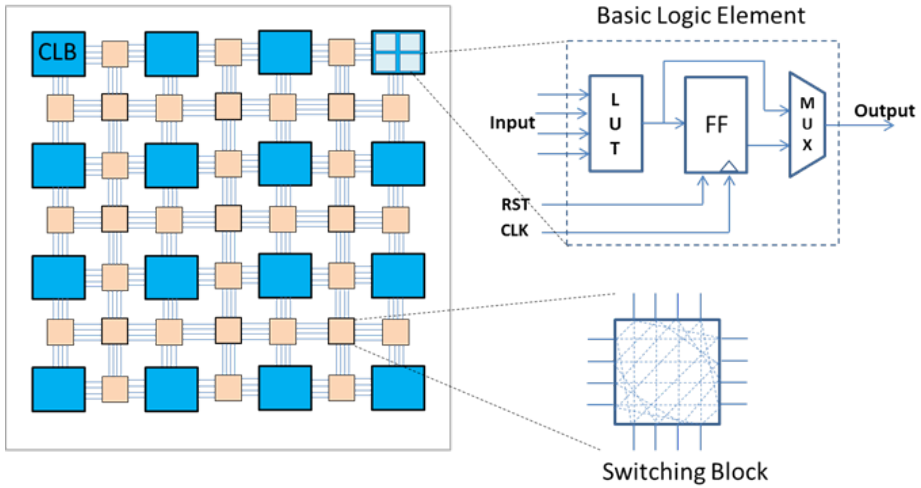
Travail à distance ?

Le travail à distance est tout à fait possible !

- Utilisation du dépôt GIT par chaque membre du binôme
- Utilisation de la VM pour la conception, simulation et test des instructions sur XSIM

Questions ?

Introduction aux FPGA



Field Programmable Gate Array = circuit programmable

- Matrice d'éléments configurables permettant l'implantation de logique combinatoire et séquentielle
- Matrice d'interconnexion
- Des blocs d'entrée-sortie pour les connexions externes
- Des macromodules matériels dédiés (mémoires, multiplieurs, blocs DSP, processeurs)
- Éléments configurables = assemblage de fonctions de générations (généralement des LUT : Lookup tables), de bascule D et de mux.

Généralités sur les FPGA

Différentes technologies de programmation selon les fournisseurs et les familles de produits

- fuse / anti-fuse → CPLD
- SRAM
- EPROM, EEPROM, Flash

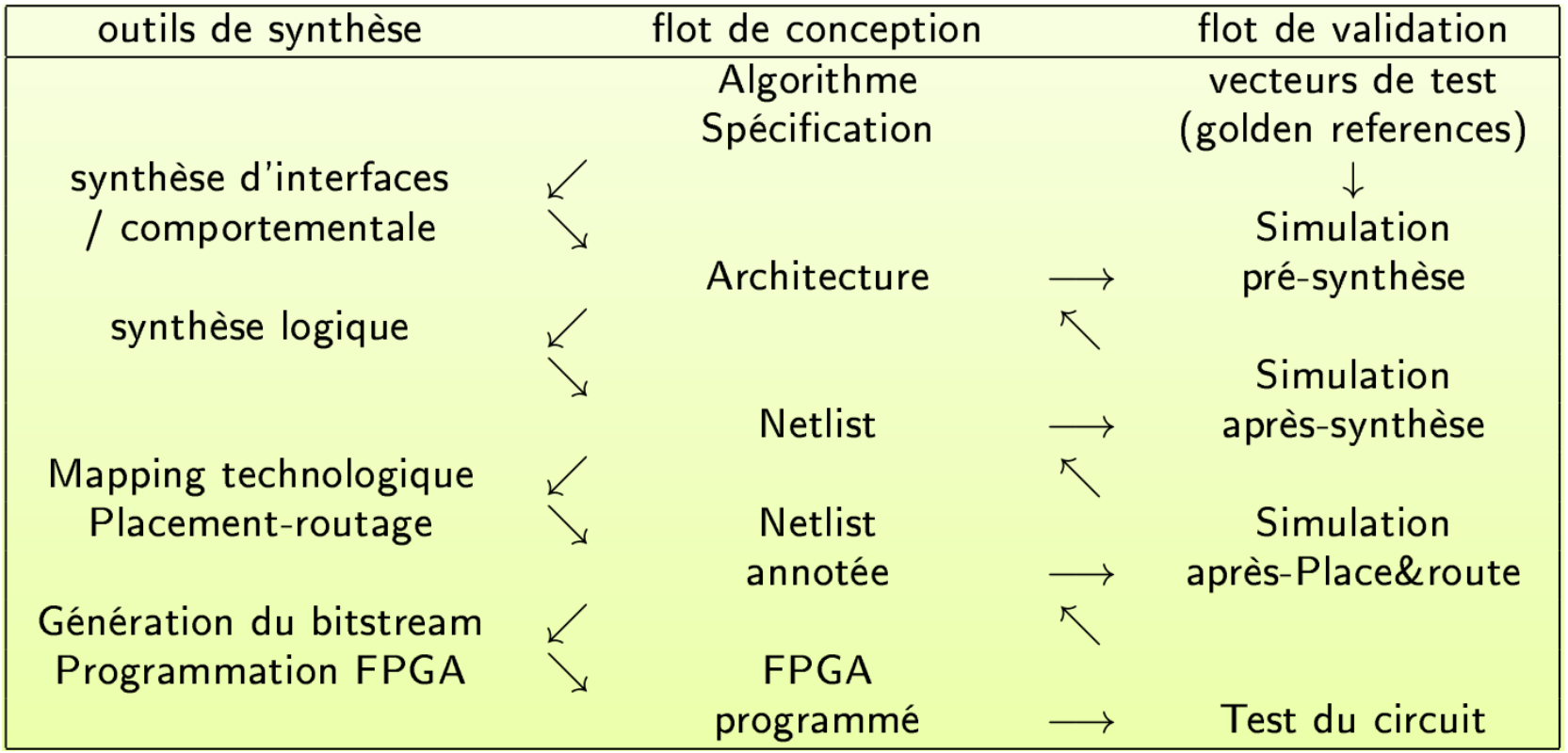
La conception sur FPGA en chiffres

- **Coût du circuit** : ingénieurs, outils de CAO 500000$ par an
  → < coût ASIC
- **TTM** bien plus court
- **Coût unitaire élevé** : > 1$ /Kportes
  en comparaison une puce ASIC : <0.1$ /Mportes
  → Intéressant pour les petites séries
- **FPGA vs. ASIC** : 3-5x plus lent, 20x plus gros, 10x plus gourmand

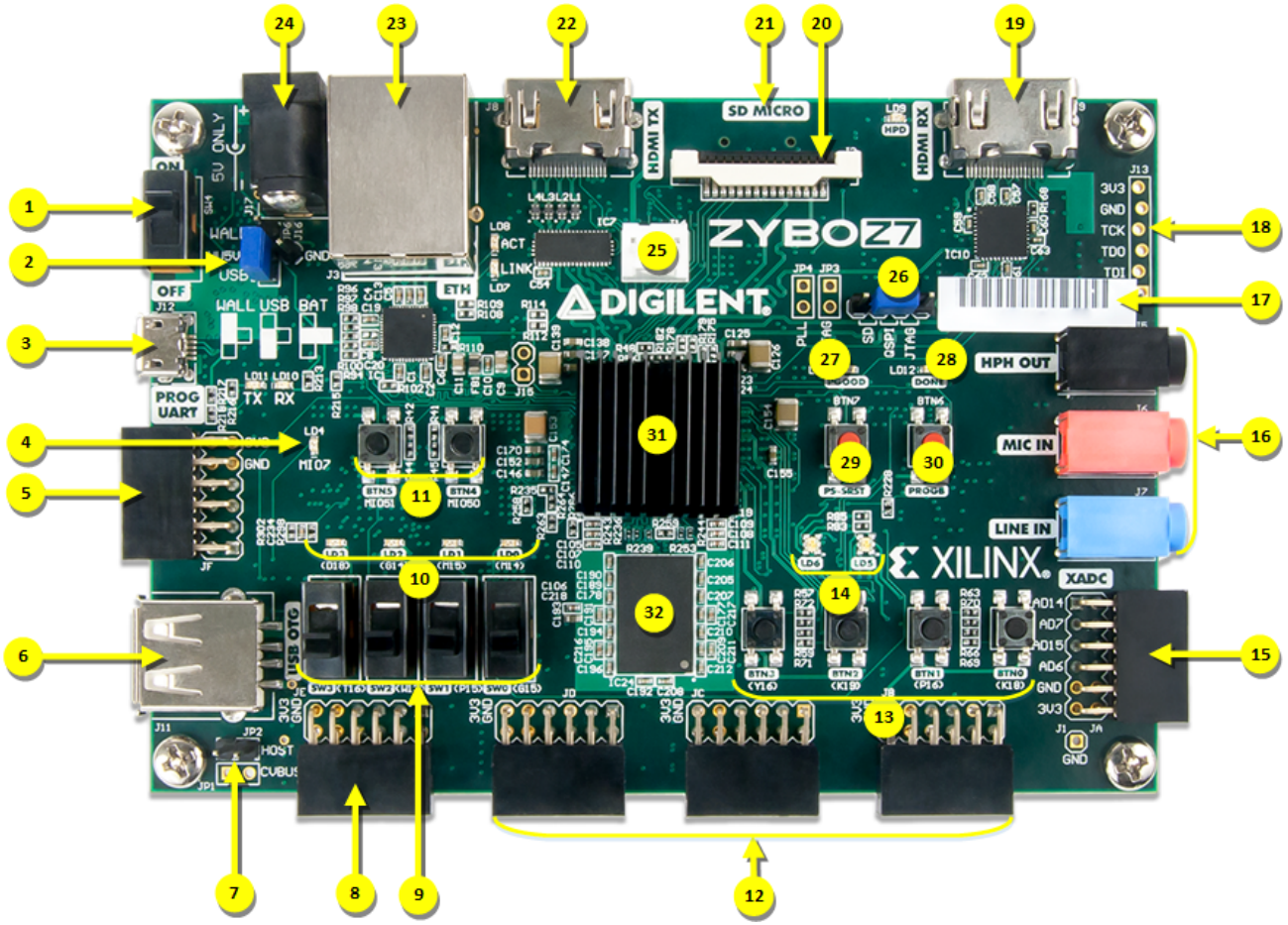
Des FPGA : pourquoi ?

- **Prototypage**
  - Emulation de circuit : bien plus rapide que la simulation
  - Facilite la phase de debug
  - Réduit les problèmes au tape out
  - Bugs détectés après le tape out → lourd coût additionnel (NRE)
- **Matériel reconfigurable** → virtualisation du matériel
- **Accélérateurs** pour des applications spécifiques
- **Sureté de fonctionnement**
  ex : spatial (durcissement pour résistance aux rayonnement cosmique + reconfiguration)

Flot de conception FPGA



La carte de développement



(1) Power switch, (2) Power select, (3) JTAG USB port, (8) Standard Pmod ports, (9) User switches, (10) User LEDs, (13) Logic Push buttons, (27) Power LED, (28) Configuration LED, (31) Zynq-7000

PMODs

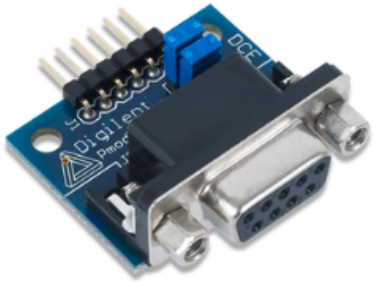
Pmod SSD: Seven-segment Display

Pmod OLEDrgb: 96 x 64 RGB OLED Display with 16-bit Color Resolution

Pmod KYPD: 16-button Keypad

Pmod ENC: Rotary Encoder

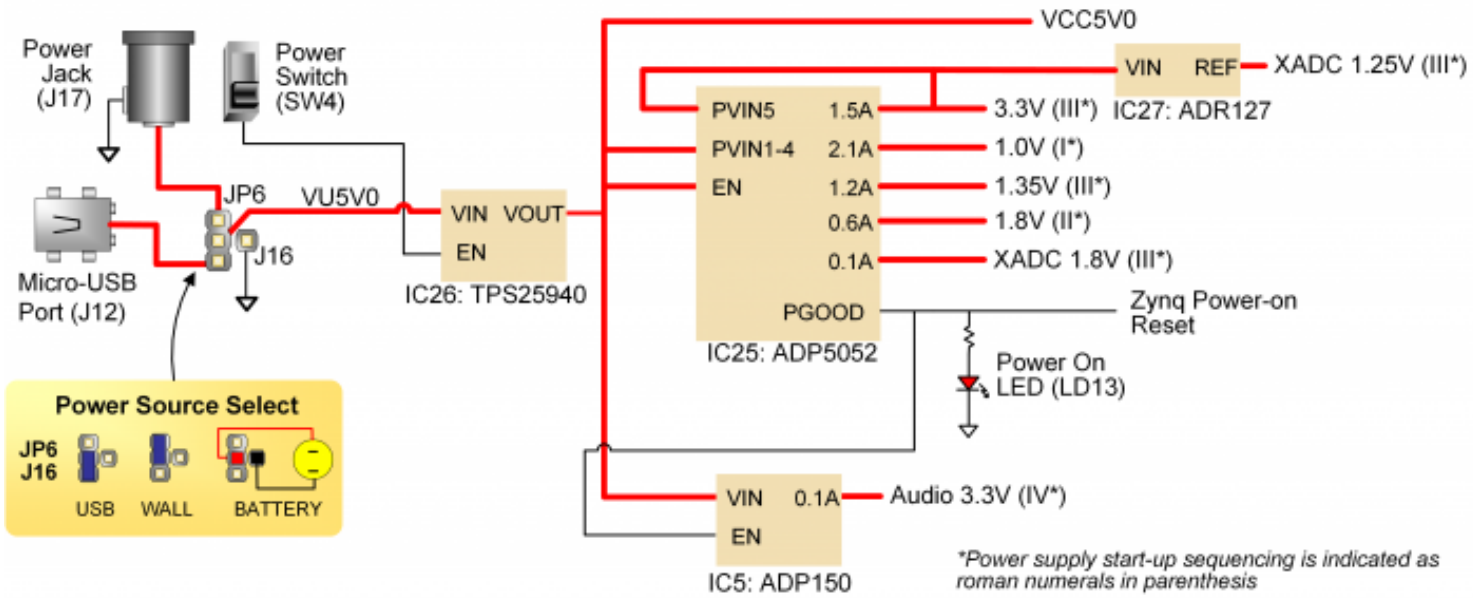
Pmod SWT: 4 User Slide Switches



Pmod RS232: Serial Converter and Interface Standard

https://store.digilentinc.com/boards-**and**-components/expansion-modules/pmods/

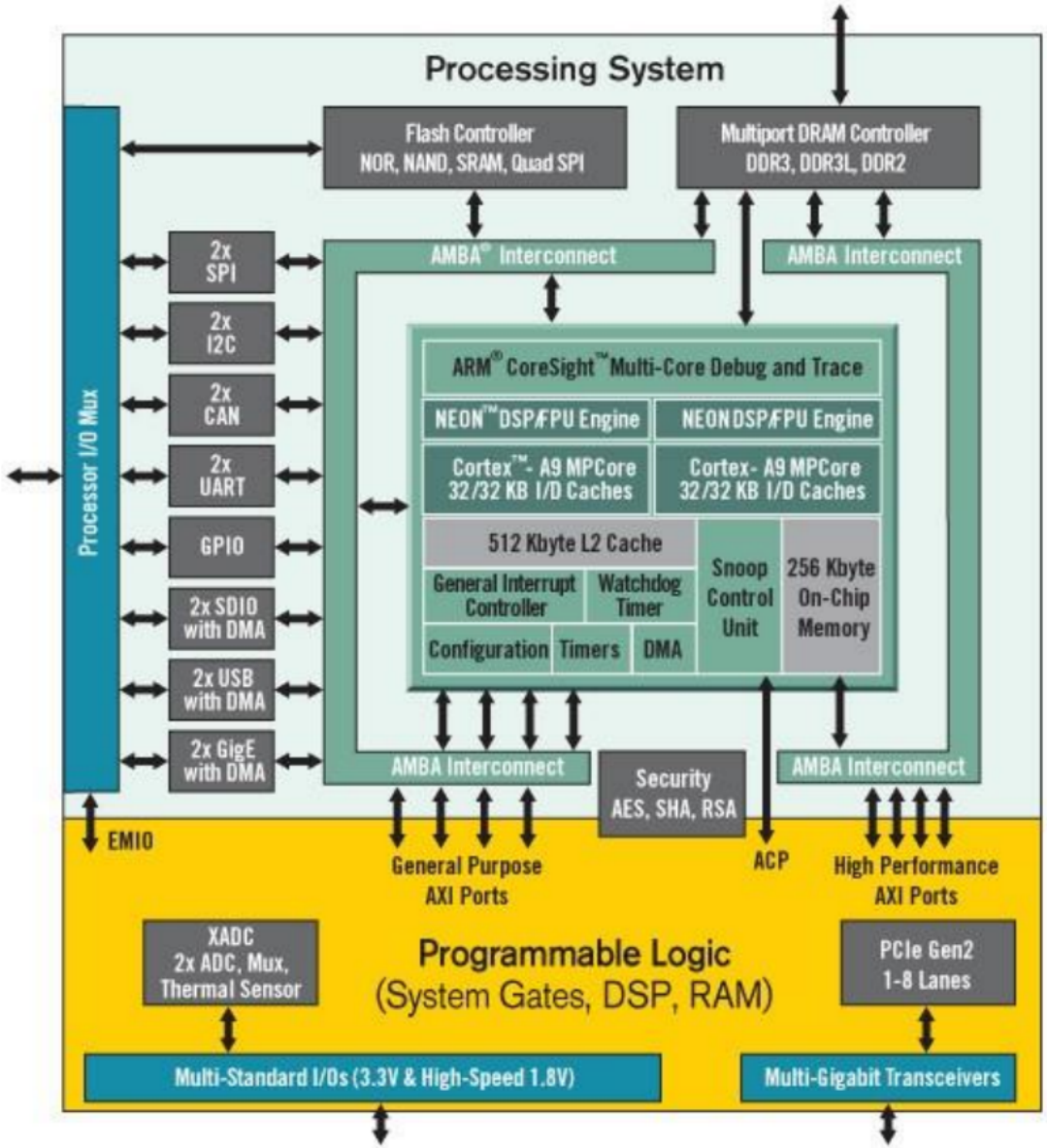
Sources d'alimentation



Connector Type	JP6 Configuration	Connector Label	Schematic net name	Min/Rec/Max Voltage (V)	Current Limit (A)
Barrel jack	WALL	J17	VJACK	4.5/5/5.5	4.0
Battery/Other	BAT	JP6, J16	VU5V0	4.5/5/5.5	4.0
USB	USB	J12	VBUS	See USB specification	0.75

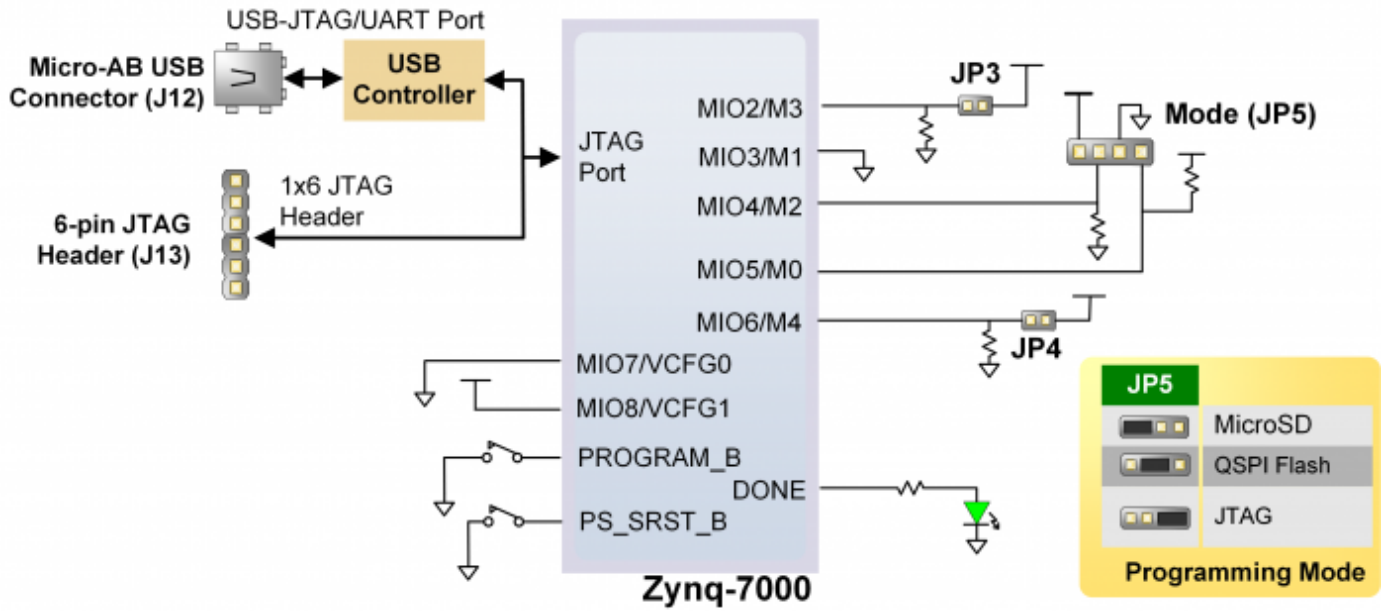
Table 1.1.1. Zybo Z7 Power Input Specifications.

Zynq SoC Architecture



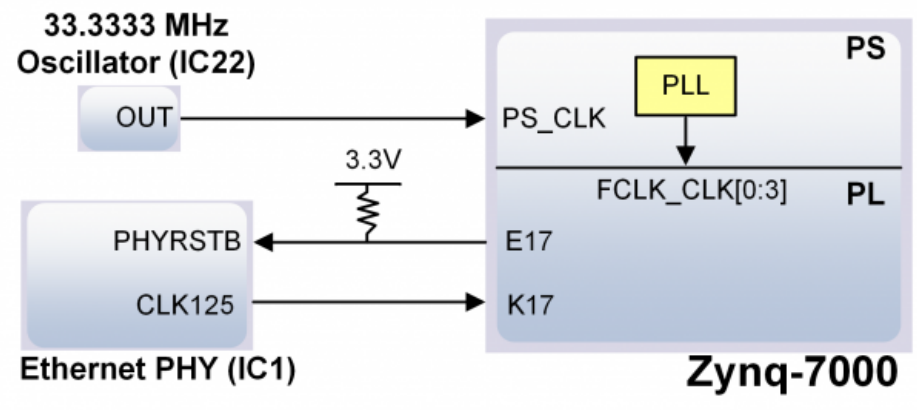
Le SoC Zynq = **Le système de traitement** (PS) + **la logique programmable** (PL)

Les broches de configuration



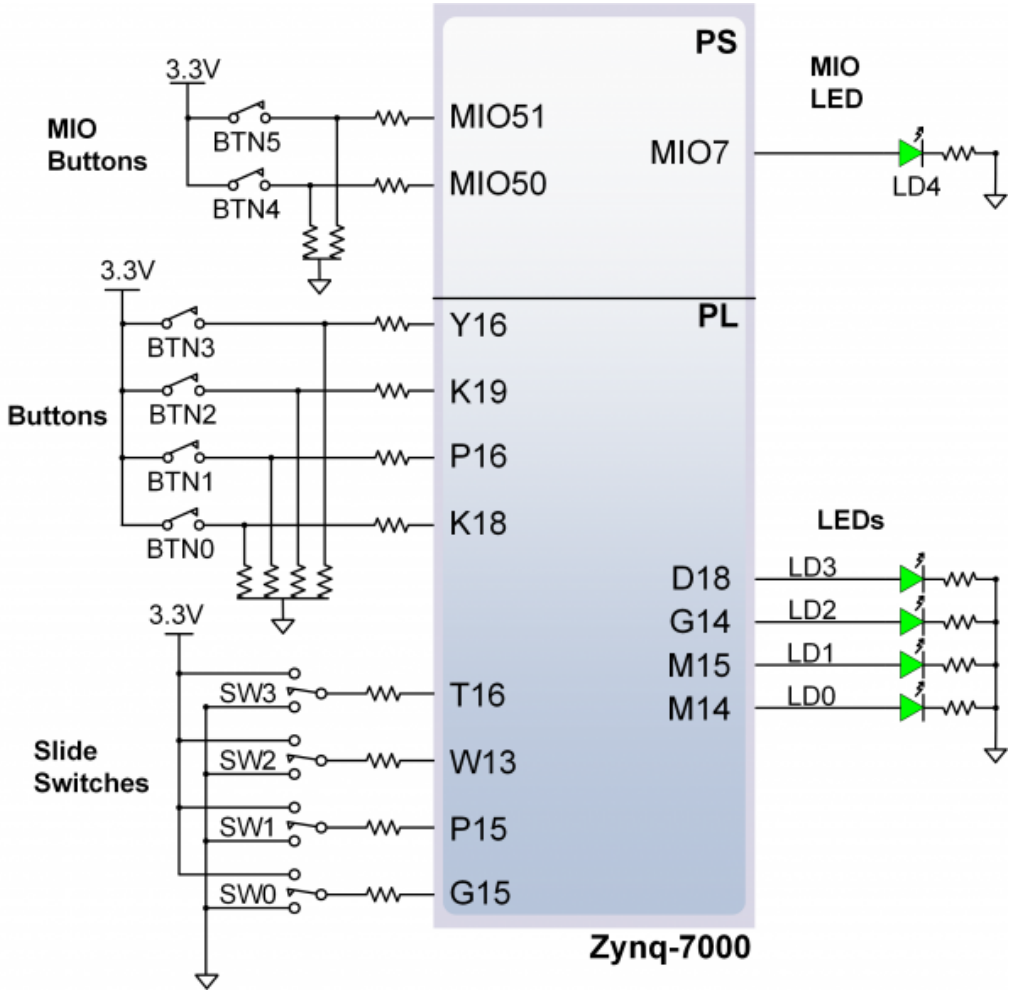
- 3 modes possibles : Micro SD, QSPI Flash, JTAG

Les sources d'horloge



- La ZYBO fournit une horloge de référence externe de **125 MHz**
- Cet horloge est directement sur la broche **K17** du PL

Les GPIO



La carte ZYBO Z7 comprend 4 **interrupteurs**, 4 **boutons poussoirs** et 4 **LED individuelles** connectées au Zynq PL.

Les contraintes du design (.xdc)

```
##Clock signal
set_property -dict {PACKAGE_PIN K17 IOSTANDARD LVCMOS33} [get_ports {CLK}];
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports{CLK}];

##Switches
set_property -dict {PACKAGE_PIN G15 IOSTANDARD LVCMOS33} [get_ports { sw[0] }];
set_property -dict {PACKAGE_PIN P15 IOSTANDARD LVCMOS33} [get_ports { sw[1] }];
set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports { sw[2] }];
set_property -dict {PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports { sw[3] }];

##Buttons
set_property -dict {PACKAGE_PIN K18 IOSTANDARD LVCMOS33} [get_ports { btn[0] }];
set_property -dict {PACKAGE_PIN P16 IOSTANDARD LVCMOS33} [get_ports { btn[1] }];
set_property -dict {PACKAGE_PIN K19 IOSTANDARD LVCMOS33} [get_ports { btn[2] }];
set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports { btn[3] }];

##LEDs
set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports { led[0] }];
set_property -dict {PACKAGE_PIN M15 IOSTANDARD LVCMOS33} [get_ports { led[1] }];
set_property -dict {PACKAGE_PIN G14 IOSTANDARD LVCMOS33} [get_ports { led[2] }];
set_property -dict {PACKAGE_PIN D18 IOSTANDARD LVCMOS33} [get_ports { led[3] }];
```