UE architecture numérique 2
Projet VHDL - CSSE
Réunion de lancement

Tanguy Philippe

Université Bretagne Sud

2023–2024

Plan

Sommaire

Objectifs pédagogiques

Le projet développé en séances de travaux pratiques vise à

- réaliser une implantation matérielle
- découvrir le flot de conception sur une cible matériel de type FPGA
- continuer l'apprentissage du langage VHDL
- étudier et implanter un algorithme/primitive cryptographique

Cette année nous allons étudier un algorithme de chiffrement de la catégorie des chiffrements par bloc appelé SERPENT [2]

Sommaire

Sommaire

Cahier des charges (1)

Contexte

Un client vient vous voir afin de vous sous-traiter un travail dont il a une expertise partielle. Il vous demande votre aide pour poursuivre un projet d'implémentation matériel d'un l'algorithme de chiffrement par bloc nommé Serpent.

Expression des besoins du client par ordre d'importance

1. code VHDL première version fonctionnelle de l'alg. Serpent avec une version utilisant une construction itérative
2. code (VHDL ou non) du Testbench associé
3. rapport de performance
4. test et validation IP seule sur cible
5. intégrer l'IP développée dans plateforme de test et validation du client

Cahier des charges (2)

Bonus

1. code VHDL première version fonctionnelle du chiffreur avec un *pipeline*
2. code VHDL Testbench
3. rapport de performance
4. intégrer l'IP développée dans plateforme de test et validation du client

Énoncé de travaux : liste des exigences

(E1) Support matériel cible

(E2) Projet VHDL Vivado v2019.1

(E3) Composant VHDL chiffreur alg. Serpent 128 bits

(E4) Composant VHDL chiffreur alg. Serpent générique

(E5) Testbench chiffreur alg. Serpent

(E6) Rapport de performance pour 128 bits

(E7) Test et validation sur cible

(E8) Intégration dans plateforme de test client

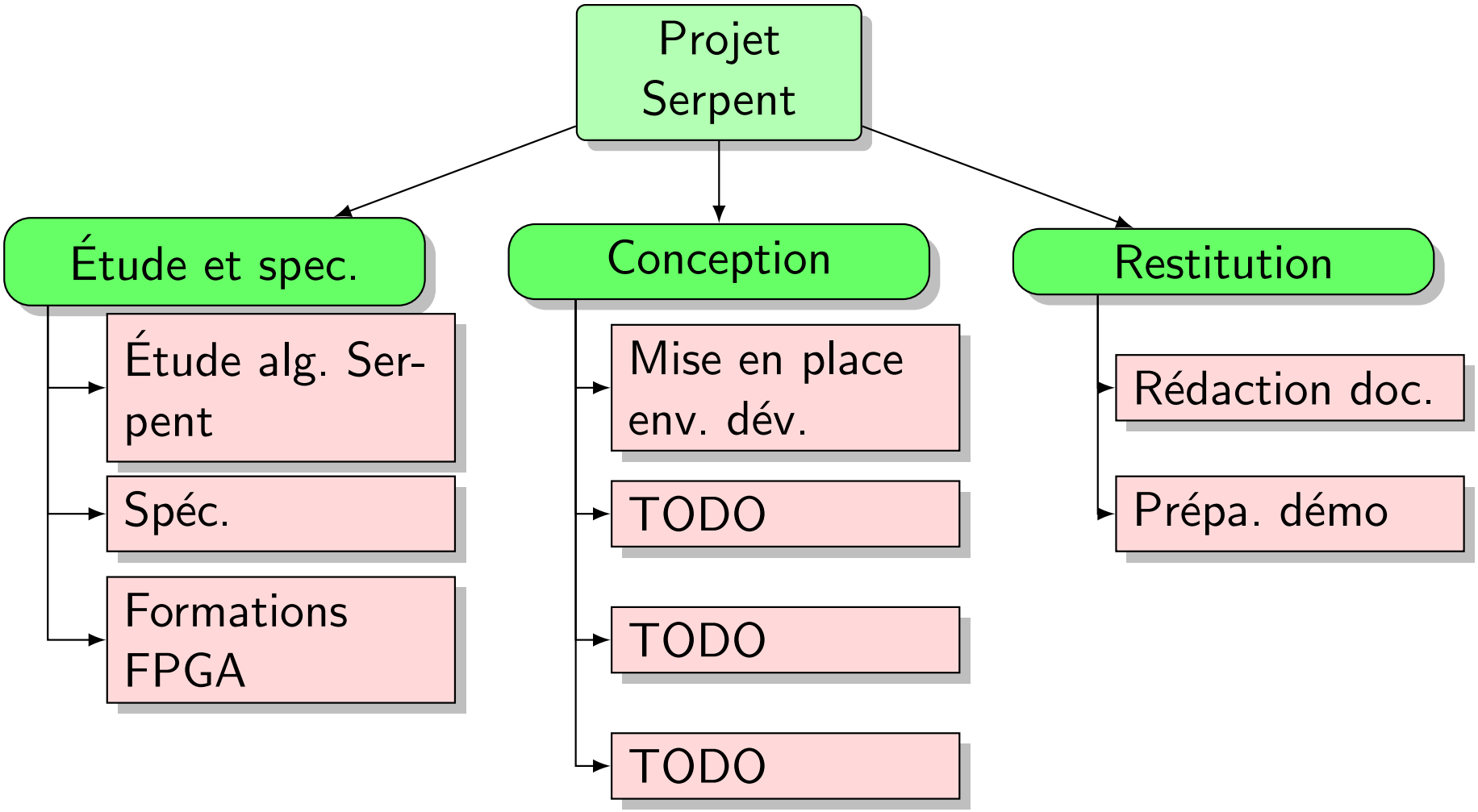
(E9) Documentation technique au format PDF

(E10) Démonstration fonctionnelle à l'équipe d'intégration produit du client

Les exigences ne sont pas classées ! cf. points associés dans la section *évaluation certificative* de la page du cours sur la plateforme pédagogique.

Sommaire

Plannification de projet : Work Breakdown Structure



Les parties notées TODO ne sont pas imposées, c'est à vous de vous organiser sur ces points !

Sommaire

Alg. Serpent en quelques mots ...

Fiche algorithme Serpent

- a été conçu par Ross Anderson, Eli Biham and Lars Knudsen [1, 2] (À lire !)
- est un chiffrement par bloc (cf. UE crypto.) qui est bien adapté pour une implantation matérielle [4].
- est caractérisé par
  - bloc de 128 bits
  - clé : 128, 192 ou 256 bits. Longueurs inférieures multiples de 8 possible.
  - structure : réseau de substitution-permutation (*Substitution-Permutation Network*, SPN)
  - 32 tours

Alg. Serpent en quelques mots ...

Fiche algorithme Serpent

- a été conçu par Ross Anderson, Eli Biham and Lars Knudsen [1, 2] (À lire !)
- est un chiffrement par bloc (cf. UE crypto.) qui est bien adapté pour une implantation matérielle [4].
- est caractérisé par
  - bloc de 128 bits
  - clé : 128, 192 ou 256 bits. Longueurs inférieures multiples de 8 possible.
  - structure : réseau de substitution-permutation (*Substitution-Permutation Network*, SPN)
  - 32 tours

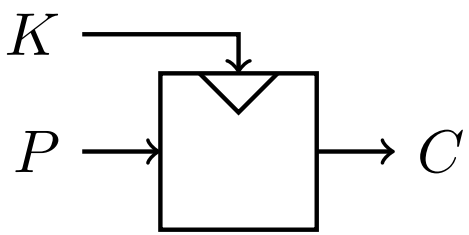
Explication de l'alg. dans la suite mais ne faites pas confiance à l'auteur de ces transparents, faites confiance à la spécification [2] !

Présentation algorithme SERPENT

Caractéristiques

- structure : réseau de substitution-permutation opérant sur 4 mots de 32 bits
- 32 tours
- 4-bit to 4-bit S-boxes
- little-endian

Aperçu boîte noire chiffreur alg. Serpent



avec P pour le *plain text*, C pour le *ciphered text* et K pour la clé de chiffrement.

Dans la suite, les termes anglais seront utilisés pour les parties du chiffreur !

Un chiffrement par bloc avec réseau SPN ? (Rappels)

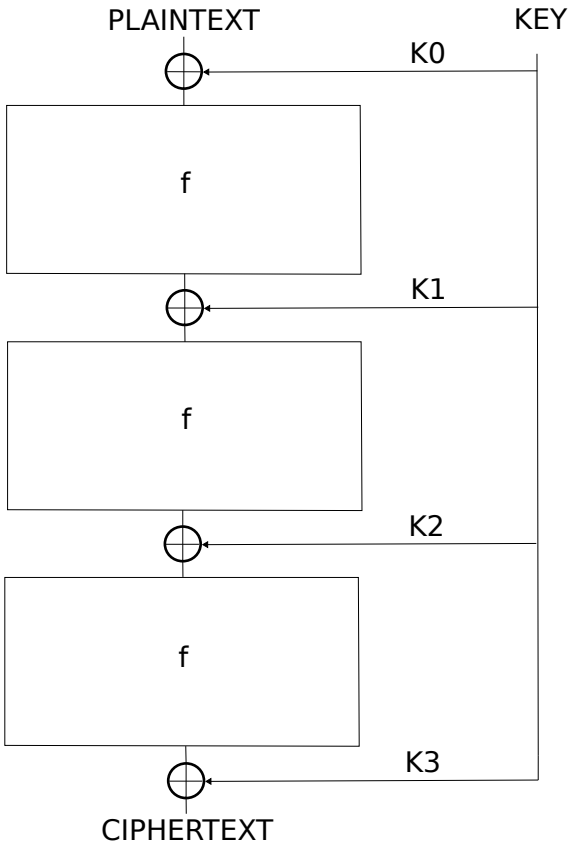


Figure 1 – Schéma général

1. Source Wikipedia

Un chiffrement par bloc avec réseau SPN ? (Rappels)

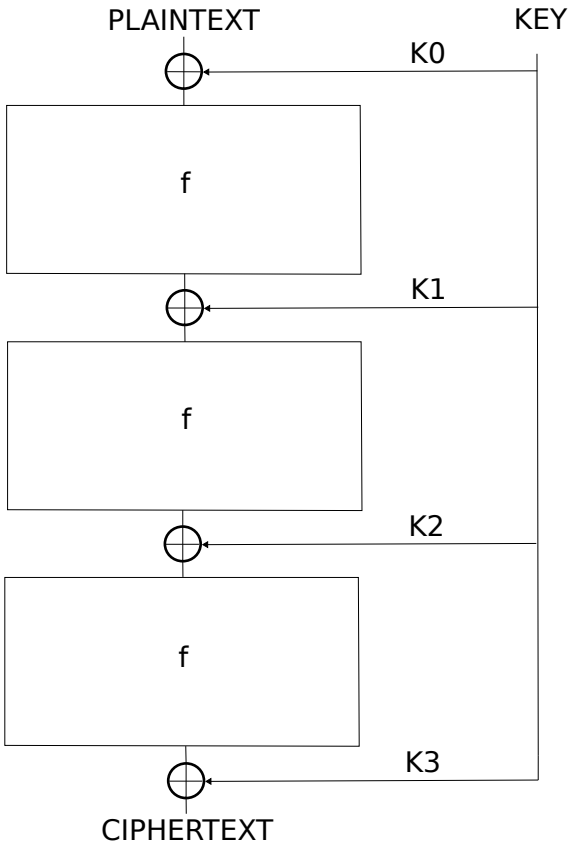


Figure 1 – Schéma général

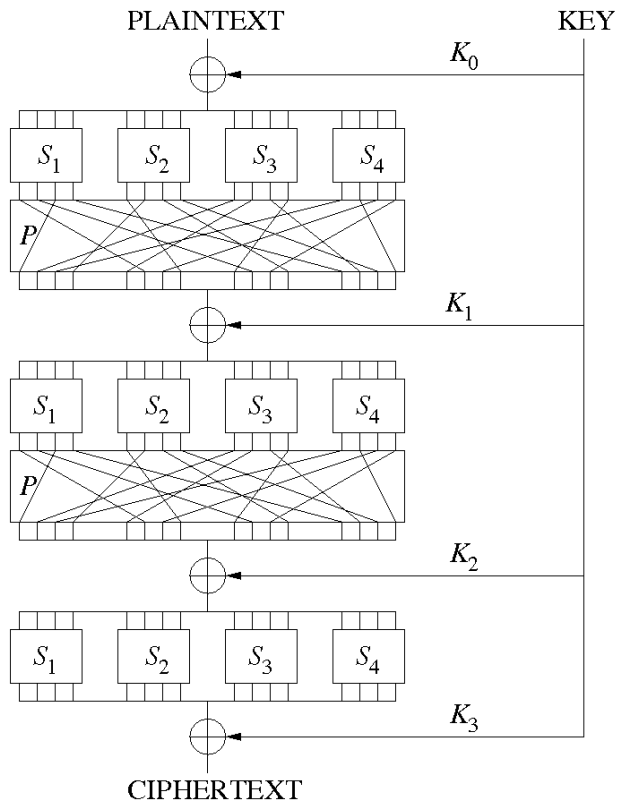


Figure 2 – Schéma [2] avec réseau SPN

1. Source Wikipedia

Chiffreur Serpent : équations

Équations décrites dans [2] :

$$\hat{B}_0 := IP(P)$$

$$\hat{B}_{i+1} := R_i(\hat{B}_i)$$

$$C := FP(\hat{B}_{32})$$

où

$$R_i(X) = L(\hat{S}_i(X \oplus \hat{K}_i)) \qquad i = 0, \ldots, 30$$

$$R_i(X) = \hat{S}_i(X \oplus \hat{K}_i) \oplus \hat{K}_{32} \qquad i = 31$$

avec

- IP pour permutation initiale, FP pour permutation finale
- $\hat{S}_i$ l'appliction de la S-box $S_{i \ mod \ 8}$ 32 fois
- L une transformation linéaire
- P pour le clair, C pour chiffré, K pour clé

Chiffreur Serpent : algorithme général

Algorithm 1 : Algorithme Serpent

$\hat{B}_0 \leftarrow IP(P)$; Initial Permutation

for $i \leftarrow 0$ **to** 31 **do**

  if $i < 31$ **then**

    $\hat{B}_{i+1} := L(\hat{S}_i(\hat{B}_i \oplus \hat{K}_i))$;

  else

    $\hat{B}_{i+1} := \hat{S}_i(\hat{B}_i \oplus \hat{K}_i) \oplus \hat{K}_{32}$;

$C \leftarrow FP(\hat{B}_{32})$; Final Permutation

Chiffreur Serpent : Initial permutation

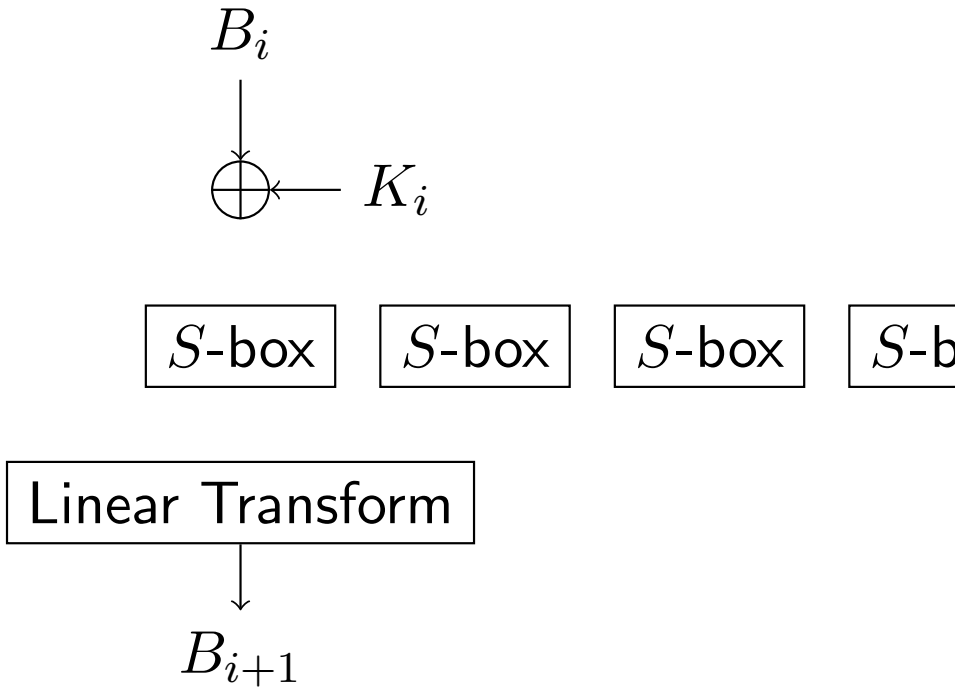
Annexe serpent spec. [2] : *Having value v (say, 32) at position p (say, 1) means that the output bit at position p(1) comes from the input bit at position v (32).*

```
0, 32, 64, 96, 1, 33, 65, 97, 2, 34, 66, 98, 3, 35, 67, 99,
4, 36, 68, 100, 5, 37, 69, 101, 6, 38, 70, 102, 7, 39, 71, 103,
8, 40, 72, 104, 9, 41, 73, 105, 10, 42, 74, 106, 11, 43, 75, 107,
12, 44, 76, 108, 13, 45, 77, 109, 14, 46, 78, 110, 15, 47, 79, 111,
16, 48, 80, 112, 17, 49, 81, 113, 18, 50, 82, 114, 19, 51, 83, 115,
20, 52, 84, 116, 21, 53, 85, 117, 22, 54, 86, 118, 23, 55, 87, 119,
24, 56, 88, 120, 25, 57, 89, 121, 26, 58, 90, 122, 27, 59, 91, 123,
28, 60, 92, 124, 29, 61, 93, 125, 30, 62, 94, 126, 31, 63, 95, 127,
```

Chiffreur Serpent : Round alg.

- key mixing :

$$\hat{B}_i \oplus \hat{K}_i$$



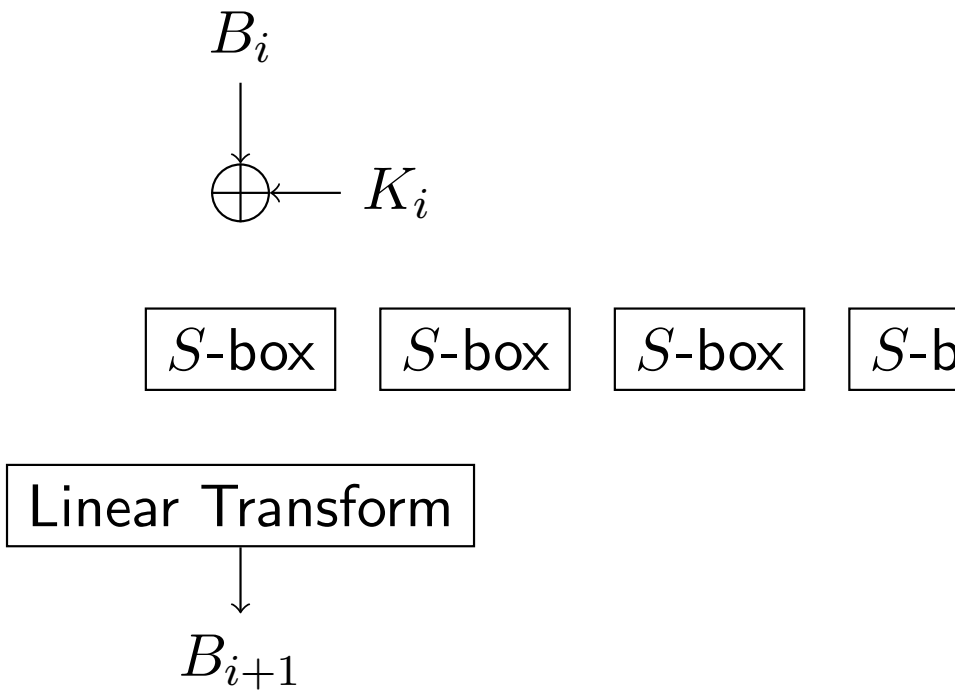
Chiffreur Serpent : Round alg.

- key mixing :

$$\hat{B}_i \oplus \hat{K}_i$$

- S-boxes :

$$S_i(\hat{B}_i \oplus \hat{K}_i))$$



Chiffreur Serpent : Round alg.

- key mixing :

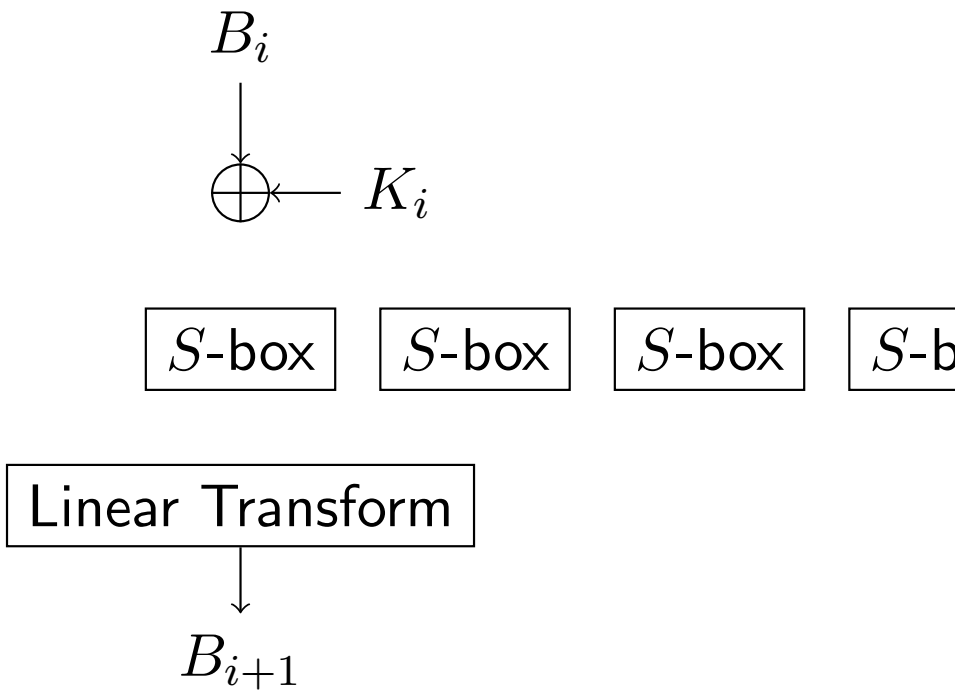
$$\hat{B}_i \oplus \hat{K}_i$$

- S-boxes :

$$S_i(\hat{B}_i \oplus \hat{K}_i))$$

- Linear transformation :

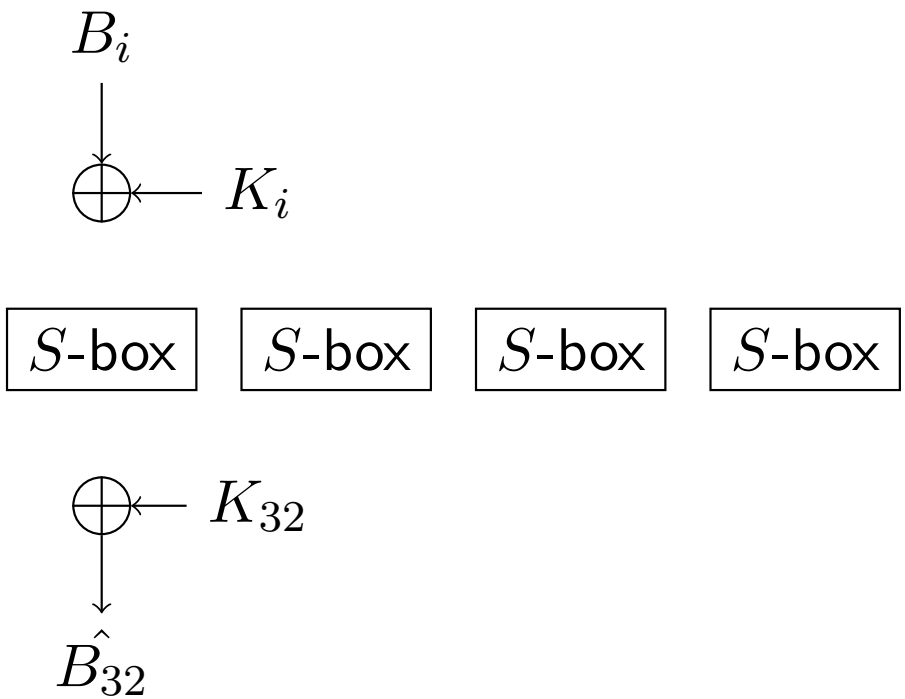
$$\hat{B}_{i+1} := L(\hat{S}_i(\hat{B}_i \oplus \hat{K}_i))$$



Chiffreur Serpent : last round

- key mixing :

$$\hat{B}_i \oplus \hat{K}_i$$



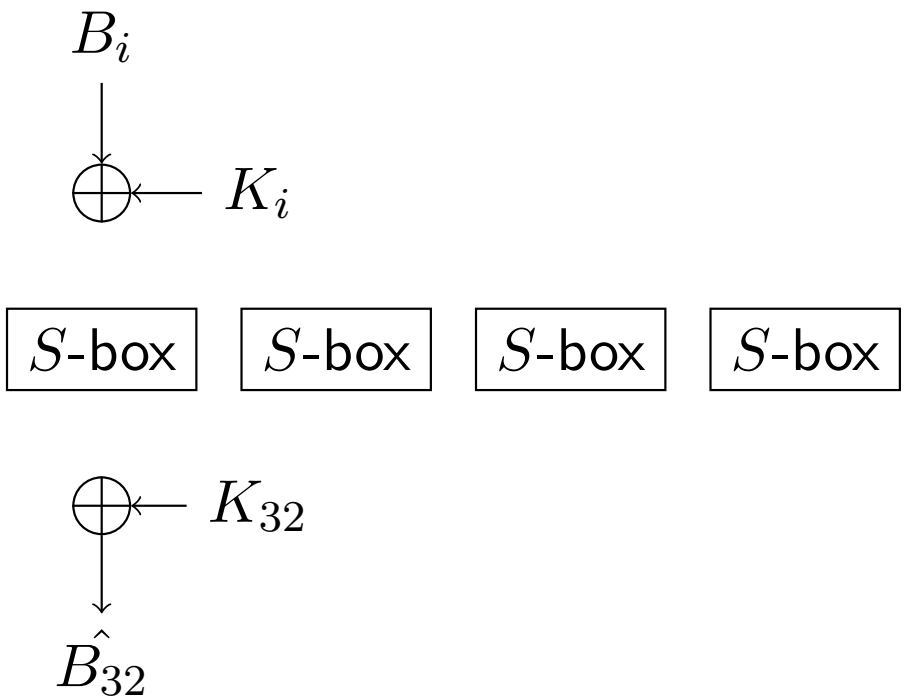
Chiffreur Serpent : last round

- key mixing :

$$\hat{B}_i \oplus \hat{K}_i$$

- S-boxes :

$$S_i(\hat{B}_i \oplus \hat{K}_i))$$



Chiffreur Serpent : last round

- key mixing :

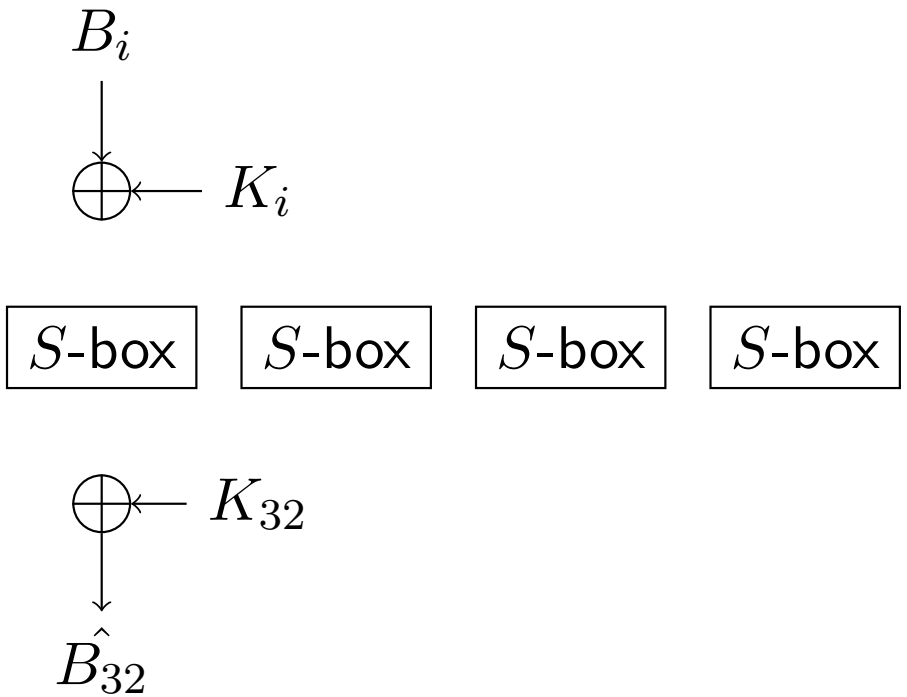
$$\hat{B}_i \oplus \hat{K}_i$$

- S-boxes :

$$S_i(\hat{B}_i \oplus \hat{K}_i))$$

- xor :

$$S_i(\hat{B}_i \oplus \hat{K}_{32}))$$



S-boxes

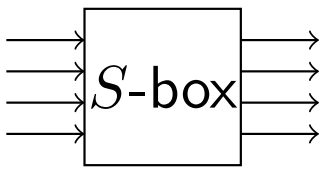
Annexe serpent spec. [2] :

- S_0 : {3,8,15,1,10,6,5,11,14,13,4,2,7,0,9,12}
- S_1 : {15,12, 2, 7, 9, 0, 5,10, 1,11,14, 8, 6,13, 3, 4 }
- S_2 : { 8, 6, 7, 9, 3,12,10,15,13, 1,14, 4, 0,11, 5, 2 }
- S_3 : { 0,15,11, 8,12, 9, 6, 3,13, 1, 2, 4,10, 7, 5,14 }
- S_4 : { 1,15, 8, 3,12, 0,11, 6, 2, 5, 4,10, 9,14, 7,13 }
- S_5 : { 15, 5, 2,11, 4,10, 9,12, 0, 3,14, 8,13, 6, 7, 1 }
- S_6 : { 7, 2,12, 5, 8, 4, 6,11,14, 9, 1,15,13, 3,10, 0 }
- S_7 : { 1,13,15, 0,14, 8, 2,11, 7, 4,12,10, 9, 3, 5, 6 }

Ce groupe de 8 S-boxes est utilisé 4 fois. Ce qui veut dire qu'après utilisation de S_7 au tour 7 c'est S_0 qui est de nouveau utilisé pour le tour 8, etc.

S-box

- Schéma S-box, 4 bits to 4 bits



- Contenu S-box S_0, opération de substitution

$$S_0 : \{3, 8, 15, 1, 10, 6, 5, 11, 14, 13, 4, 2, 7, 0, 9, 12\}$$

- Par exemple dans la liste de 0 à 15 entier de la boîte S_0, une entrée avec la valeur 0x3 donnera en sortie la valeur 0x1.

Chiffreur Serpent : linear transformation

Reformulation de [2]

$$X_0, X_1, X_2, X_3 := S_i(\hat{B}_i \oplus \hat{K}_i)$$

$$X_0 := X_0 <<< 13$$

$$X_2 := X_2 <<< 3$$

$$X_1 := X_1 \oplus X_0 \oplus X_2$$

$$X_3 := X_3 \oplus X_2 \oplus (X_0 << 3)$$

$$X_1 := X_1 <<< 1$$

$$X_3 := X_3 <<< 7$$

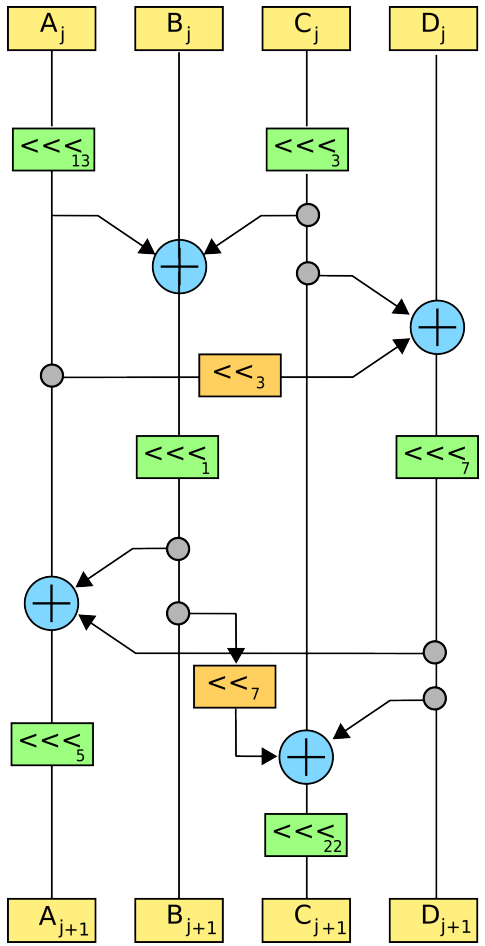
$$X_0 := X_0 \oplus X_1 \oplus X_3$$

$$X_2 := X_2 \oplus X_3 \oplus (X_1 << 7)$$

$$X_0 := X_0 <<< 5$$

$$X_2 := X_2 <<< 22$$

$$\hat{B}_{i+1} := X_0, X_1, X_2, X_3$$



Source img : Wikipedia

Chiffreur Serpent : Final permutation

Annexe serpent spec. [2] :

```
0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60,
64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112, 116, 120, 124,
1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61,
65, 69, 73, 77, 81, 85, 89, 93, 97, 101, 105, 109, 113, 117, 121, 125,
2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62,
66, 70, 74, 78, 82, 86, 90, 94, 98, 102, 106, 110, 114, 118, 122, 126,
3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63,
67, 71, 75, 79, 83, 87, 91, 95, 99, 103, 107, 111, 115, 119, 123, 127,
```

Chiffreur Serpent partie key-schedule

Serpent key-schedule

C'est une opération de préparation des sous-clés utilisées lors des tours du chiffrement par bloc. La *user key* est tranformée en une clé K de 256 bits si nécessaire. Cette clé est ensuite transformé par l'alg. 2 en $\hat{K}_i$ *round keys* avec $i = 0, \ldots, 32$.

- standard mode
- bitslice mode

key-schedule alg.

Algorithm 2 : Algorithme Serpent key schedule

$\phi \leftarrow 0x9e3779b9$;
$K \leftarrow mapping(user\ key)$; mapping user key to 256 bits if necessary ;
Split K to $w_{-8}, \ldots, w_{-1}$ avec W a 32-bit words ;
for $i \leftarrow 0$ **to** 131 **do**
  $w_i := (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) <<< 11$; generate prekey
$\{k_0, k_1, k_2, k_3\} := S_3(w_0, w_1, w_2, w_3)$;
$\{k_4, k_5, k_6, k_7\} := S_2(w_4, w_5, w_6, w_7)$;
$\{k_8, k_9, k_{10}, k_{11}\} := S_1(w_8, w_9, w_{10}, w_{11})$;
$\{k_{12}, k_{13}, k_{14}, k_{15}\} := S_0(w_{12}, w_{13}, w_{14}, w_{15})$;
$\{k_{16}, k_{17}, k_{18}, k_{19}\} := S_7(w_{16}, w_{17}, w_{18}, w_{19})$;
$\ldots$;
$\{k_{124}, k_{125}, k_{126}, k_{127}\} := S_4(w_{124}, w_{125}, w_{126}, w_{127})$;
$\{k_{128}, k_{129}, k_{130}, k_{131}\} := S_3(w_{128}, w_{129}, w_{130}, w_{131})$;
for $i \leftarrow 0$ **to** 32 **do**
  $K_i := \{k_{4i}, k_{4i+1}, k_{4i+2}, k_{4i+3}\}$;
for $i \leftarrow 0$ **to** 32 **do**
  $\hat{K}_i \leftarrow IP(K_i)$;

key-schedule alg., compléments

- opération de *mapping* consiste à ajouter '1' sur le MSB puis autant de zéros (*padding*) pour compléter

Implémentation logicielle

- Implémentation de référence en C fourni par les auteurs de Serpent
- Implémentation de référence en Python [3] ("Nothing better than a Python to write a Serpent", Frank Stajano) fait par un développeur différent qui permis de déboguer l'implémentation de référence ☺
- Utile pour déboguer votre implémentation matérielle !

3. `https://www.cl.cam.ac.uk/~fms27/serpent/`

Implémentation matérielle

Votre contribution !

TODO liste

- Spécification interne du chiffreur
- Développement VHDL
- Tesbench
- Tests
- Intégration au sein de l'API proposée
- Validation

Suite de test ?

- Important car ce sera votre référence !
- Utilisez les implémentations de référence pour le debug ?
- Confronter votre implémentation ?

Aspects sécurité

Vis-à-vis des protections de la cryptographie

Résistance à cryptanalyse classique [2] . . . *vous avez eu une introduction dans l'UE crypto*

Pas le propos du projet mais devrait être inclus dans le flot de conception !

Vis-à-vis de l'implémentation

- Résistance aux attaques physiques par observations . . . *vous avez eu une introduction dans l'UE crypto premier semestre !*
- Résistance aux attaques physiques par injections de fautes . . . *vous avez eu une introduction dans l'UE crypto premier semestre !*
- Phase de caractérisation sur cible vis-à-vis des attaques par observation et/ou injection de fautes
- Mise en œuvre de protections vis-à-vis des attaques par observation et/ou injection de fautes

Sommaire

Spécifications chiffreur Serpent : entité

SERPENT_128			
Nom	Direction	Nombre de bits	Description
XXX	ENTREE	XXX	XXX
XXX	SORTIE	XXX	XXX

À affiner selon vos besoins !

Spécifications chiffreur Serpent : séquencement

1. État *load* : chargement *key* et *plain text*
2. État *busy* : opération de chiffrement
3. État *load* : *cipher text* disponible en sortie

À affiner selon vos besoins !

Spécification banc de test et validation client (1)

Aperçu banc de test

- Banc de test du client fourni
- La partie PS de la Zedboard sera utilisée

Sommaire

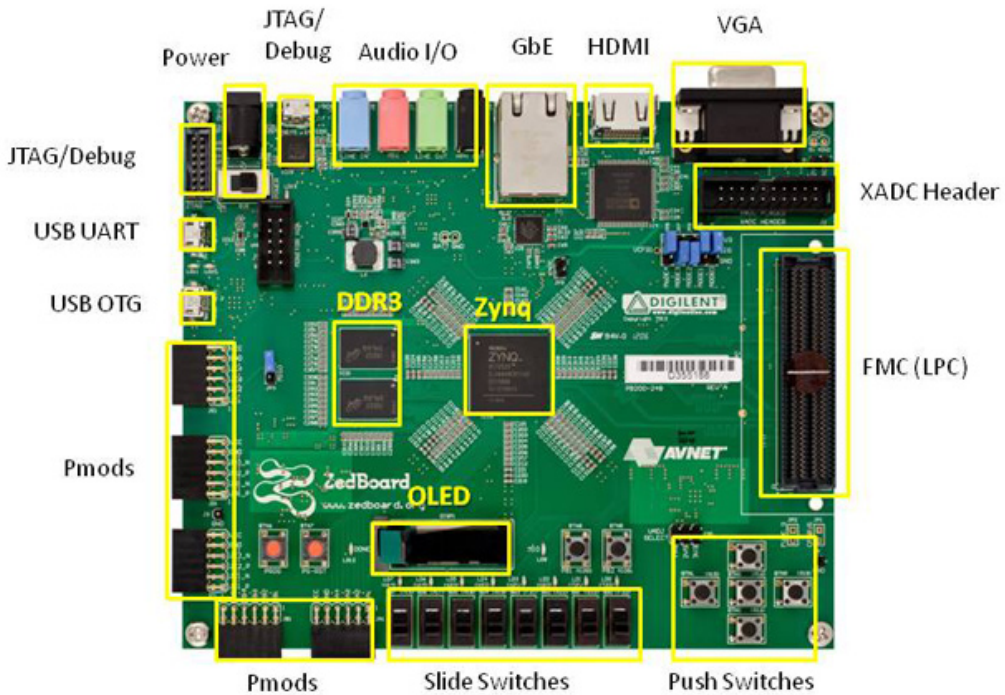
Introduction

Objectif

- Présentation carte Zedboard
- Présentation du SoC Zynq-7000
- Introduction à la technologie FPGA en prenant l'exemple du Zynq

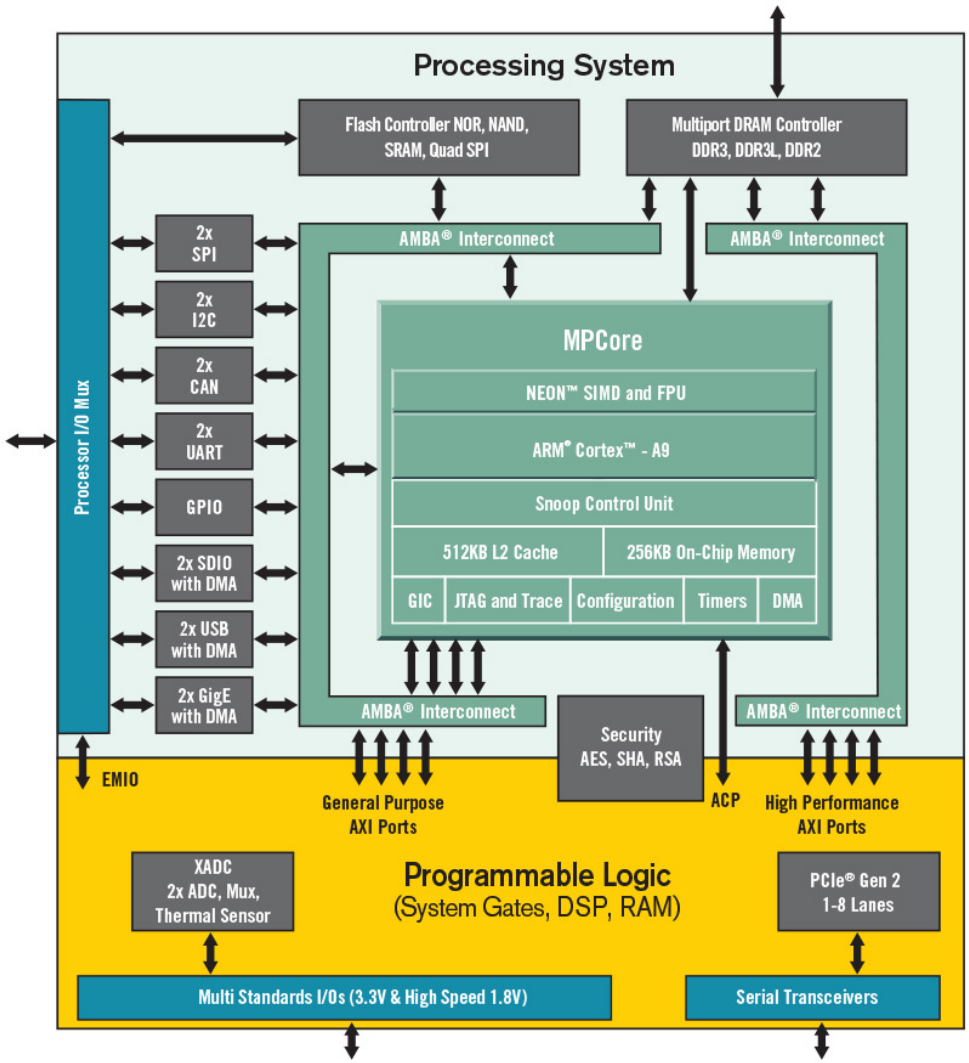
Plate-forme cible

Carte de développement Xilinx Zedboard (SoC-FPGA)



* SD card cage and QSPI Flash reside on backside of board

SoC Zynq


Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA® Interconnect
MPCore
NEON™ SIMD and FPU
ARM® Cortex™ - A9
Snoop Control Unit
512KB L2 Cache
256KB On-Chip Memory
GIC
JTAG and Trace
Configuration
Timers
DMA
AMBA® Interconnect
AMBA® Interconnect
Security AES, SHA, RSA
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe® Gen 2 1-8 Lanes
Multi Standards I/Os (3.3V & High Speed 1.8V)
Serial Transceivers

SoC Zynq caractéristiques

Zynq®-7000 All Programmable SoC Family

		Cost-Optimized Devices						Mid-Range Devices			
	Device Name	Z-7007S	Z-7012S	Z-7014S	Z-7010	Z-7015	Z-7020	Z-7030	Z-7035	Z-7045	Z-7100
	Part Number	XC7Z007S	XC7Z012S	XC7Z014S	XC7Z010	XC7Z015	XC7Z020	XC7Z030	XC7Z035	XC7Z045	XC7Z100
Processing System (PS)	Processor Core	Single-Core ARM® Cortex™-A9 MPCore™ Up to 766MHz			Dual-Core ARM Cortex-A9 MPCore Up to 866MHz			Dual-Core ARM Cortex-A9 MPCore Up to 1GHz[1]			
	Processor Extensions	NEON™ SIMD Engine and Single/Double Precision Floating Point Unit per processor									
	L1 Cache	32KB Instruction, 32KB Data per processor									
	L2 Cache	512KB									
	On-Chip Memory	256KB									
	External Memory Support[2]	DDR3, DDR3L, DDR2, LPDDR2									
	External Static Memory Support[2]	2x Quad-SPI, NAND, NOR									
	DMA Channels	8 (4 dedicated to PL)									
	Peripherals	2x UART, 2x CAN 2.0B, 2x I2C, 2x SPI, 4x 32b GPIO									
	Peripherals w/ built-in DMA[2]	2x USB 2.0 (OTG), 2x Tri-mode Gigabit Ethernet, 2x SD/SDIO									
	Security[3]	RSA Authentication of First Stage Boot Loader, AES and SHA 256b Decryption and Authentication for Secure Boot									
	Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts									
Programmable Logic (PL)	7 Series PL Equivalent	Artix®-7	Artix-7	Artix-7	Artix-7	Artix-7	Artix-7	Kintex®-7	Kintex-7	Kintex-7	Kintex-7
	Logic Cells	23K	55K	65K	28K	74K	85K	125K	275K	350K	444K
	Look-Up Tables (LUTs)	14,400	34,400	40,600	17,600	46,200	53,200	78,600	171,900	218,600	277,400
	Flip-Flops	28,800	68,800	81,200	35,200	92,400	106,400	157,200	343,800	437,200	554,800
	Total Block RAM (# 36Kb Blocks)	1.8Mb (50)	2.5Mb (72)	3.8Mb (107)	2.1Mb (60)	3.3Mb (95)	4.9Mb (140)	9.3Mb (265)	17.6Mb (500)	19.2Mb (545)	26.5Mb (755)
	DSP Slices	66	120	170	80	160	220	400	900	900	2,020
	PCI Express®	—	Gen2 x4	—	—	Gen2 x4	—	Gen2 x4	Gen2 x8	Gen2 x8	Gen2 x8
	Analog Mixed Signal (AMS) / XADC[2]	2x 12 bit, MSPS ADCs with up to 17 Differential Inputs									
	Security[3]	AES & SHA 256b Decryption & Authentication for Secure Programmable Logic Config									
	Speed Grades Commercial	-1			-1			-1			-1
	Speed Grades Extended	-2			-2,-3			-2,-3			-2
	Speed Grades Industrial	-1, -2			-1, -2, -1L			-1, -2, -2L			-1, -2, -2L

Notes:
1. 1 GHz processor frequency is available only for -3 speed grades in Z-7030, Z-7035, and Z-7045 devices. See DS190, *Zynq-7000 All Programmable SoC Overview* for details.
2. Z-7007S and Z-7010 in CLG225 have restrictions on PS peripherals, memory interfaces, and I/Os. Please refer to UG585, *Zynq-7000 All Programmable SoC Technical Reference Manual* for more details.
3. Security block is shared by the Processing System and the Programmable Logic.

Page 2

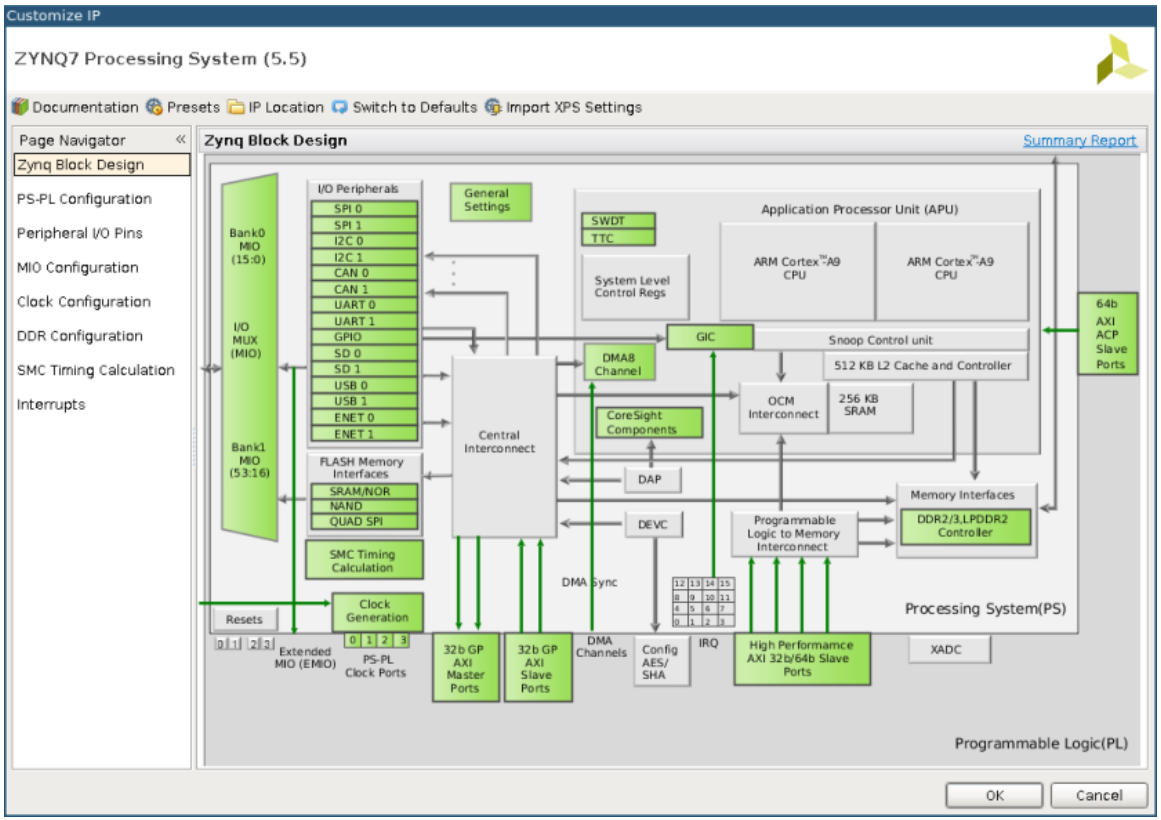
XILINX ALL PROGRAMMABLE.

Source : `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.htmlx`

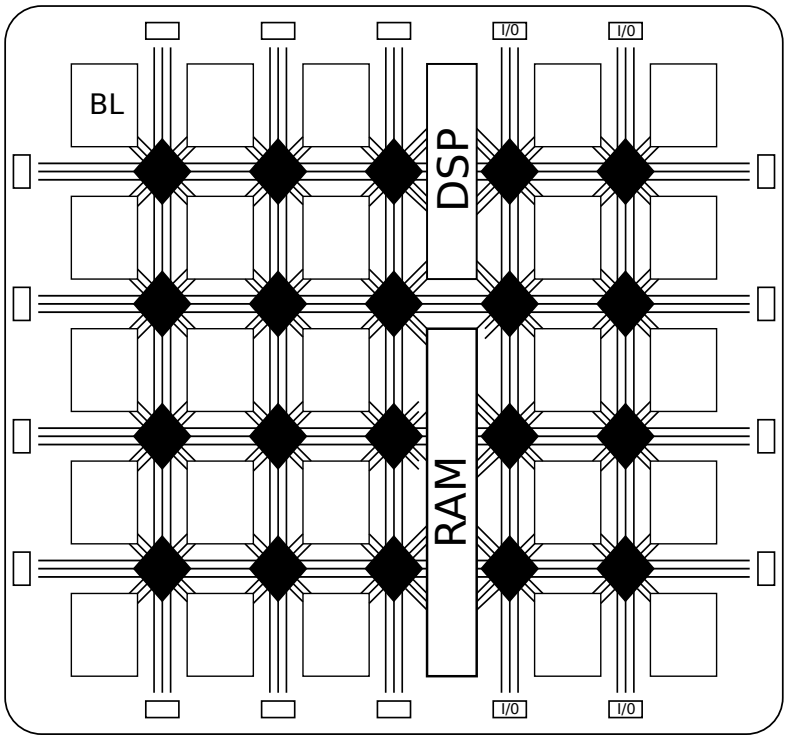
SoC Zynq : partie PS

Application Processor Unit (APU)

- Deux coeurs ARM cortex A9
- Accès PL par port Accelerator Coherence Port via le Snoop Control Unit (SCU)

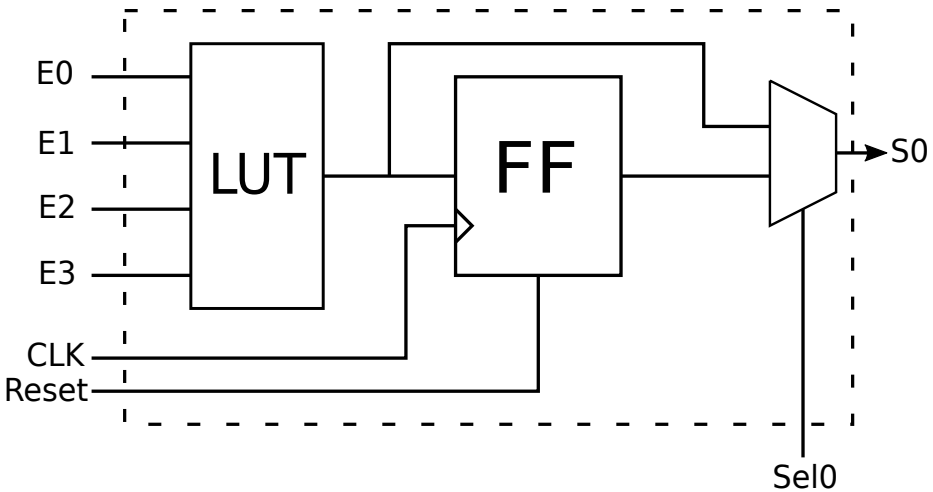


FPGA : architecture générale simplifiée



- BL : bloc logique
- I/O : bloc entrée/sortie
- Interconnection
- RAM : bloc mémoire RAM
- DSP : bloc arithmétique traitement du signal

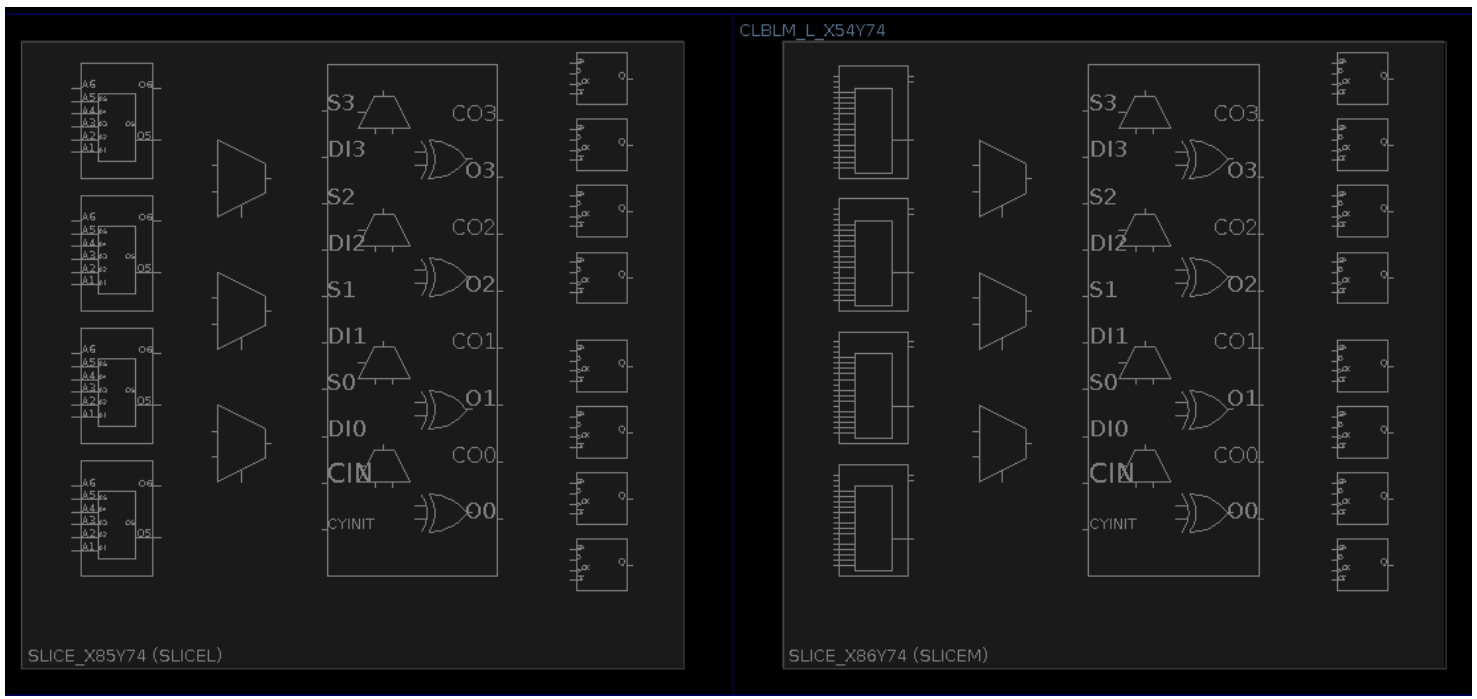
Bloc logique : architecture simplifiée



- LUT : look up table
- FF : Flip Flop
- Multiplexeur

SoC Zynq, partie PL : Configurable Logic Block

- Xilinx doc UG474
- Configurable Logic Block (CLB)
- Contient 2 slices : sliceL et sliceM
- 1 slice contient 4 LUT et 8 bascules



SoC Zynq, Zedboard : ressources d'horloge

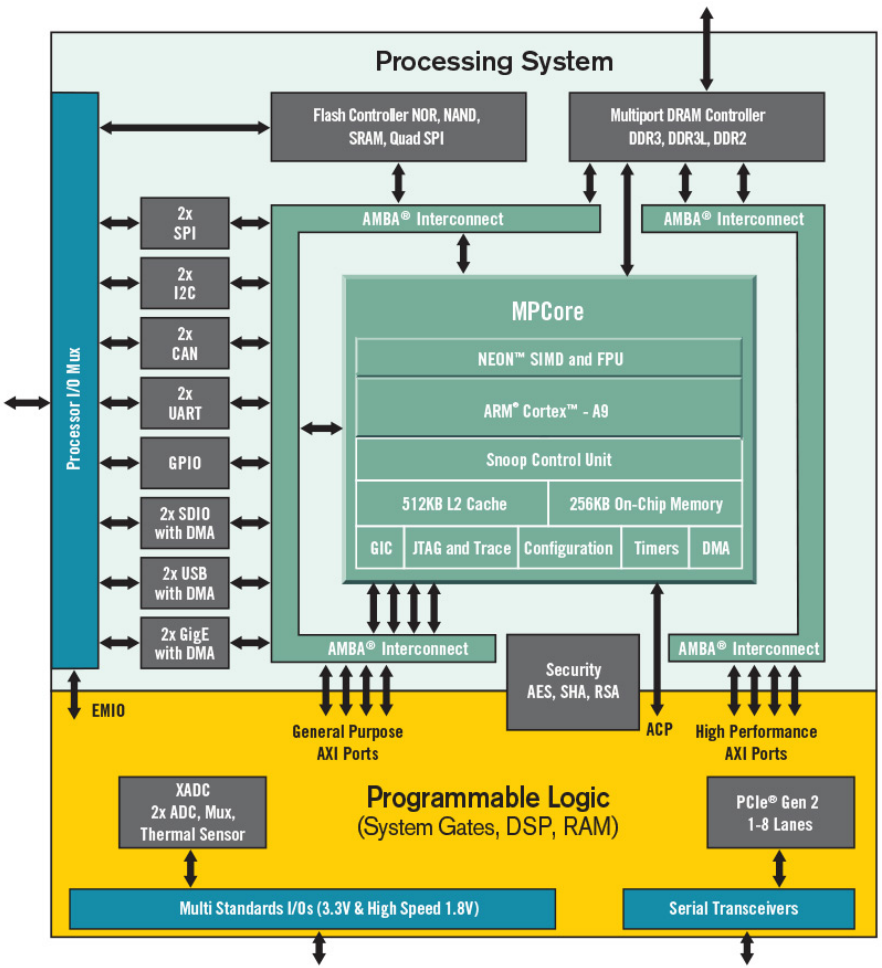
Zynq

- Mixed-Mode Clock Managers (MMCM)
- Phase Locked Loop (PLL)

Zedboard

- Documentation Zedboard Hardware User Guide
- Sources horloge (extrait doc)
  - Zynq-7000 AP SoCs PS subsystem uses a dedicated 33.3333 MHz clock source, IC18, Fox 767-33.333333-12, with series termination. The PS infrastructure can generate up to four PLL-based clocks for the PL system.
  - An on-board 100 MHz oscillator, IC17, Fox 767-100-136, supplies the PL subsystem clock input on bank13, pin Y9.

SoC Zynq : interconnections PS et PL



- 32-bit Master AXI ports (PS master)
- 2 32-bit Slave AXI ports (PL Master)
- 4 32/64-bit Slave High Performance Ports (PL Master)
- 1 64-bit Slave Accelerator Coherency Port (ACP) (PLMaster)
- 4 horloges PS vers PL
- PS vers PL interruption
- PL vers PS interruption
- DMA

Pour aller plus loin . . .

Sélection de ressources à lire !

- Zynq-7000 SoC Technical Reference Manual, UG585
- Article (FR) générale sur FPGA [5]
- Livre en ligne sur le Zynq, Zynq Book [3]

Sommaire

Qu'est-ce qui reste à faire ?

Après la réunion de lancement il faudra

- revoir T1.1 et finaliser sa compréhension de l'algorithme
- revoir T1.2 et comprendre les spécifications haut niveau
- revoir T2.1 et maîtriser l'environnement de développement ainsi que la carte FPGA
- définir le reste de votre travail en fonction du cahier des charges et du flot de conception pour faire les tàches T2.XX

Sommaire

Recommandations

- Autonomie
- Respect planification de projet proposée
- Respect des exigences
- Gestion du code source avec Git et la plate-forme Gitlab de l'UBS
  1. Création dépôt sur `https://forgens.univ-ubs.fr/gitlab/` en mode privé
  2. Nommmez votre dépôt *nom-prenom-projet-vhdl*
  3. Ajoutez l'intervenant de TP en tant que membre avec les droits *Developer*

Pour vous guider et savoir ce qu'on attends de vous lisez la grille critériée ainsi que le syllabus de cours disponible sur la plateforme pédagogique (Moodle) !

Conseils

- Abordez l'architecture demandée en faisant un schéma papier des différents blocs
- Rappelez vous de vos erreurs et points faibles en VHDL
- Prenez en compte le flot de conception dans votre gestion du temps
- Prenez en compte le temps de rédaction des livrables
- Ne cherchez pas à optimiser dès le début car ce sera source d'erreurs !
- Profitez-en pour mettre en place des bonnes pratiques de gestion de projet informatique (Git, etc.), de style de code, de rédaction de documentation, etc.

Références (1)

[1] R. Anderson, E. Biham, and L. Knudsen.
Serpent website.

[2] R. J. Anderson, E. Biham, and L. R. Knudsen.
Serpent : A proposal for the advanced encryption standard, 1998.

[3] L. Crockett, R. Elliot, M. Enderwitz, B. Stewart, and D. Northcote.
Zynqbook website, 2019.

[4] J. R. Nechvatal, E. B. Barker, L. E. Bassham, W. E. Burr, M. J. Dworkin, J. Foti, and E. Roback.
Report on the Development of the Advanced Encryption Standard (AES).
106 No. 3, June 2001.

Références (2)

[5] O. Sentieys and A. Tisserand.
Architectures reconfigurables FPGA.
page 25, 2012.

Crédits

- Tikz for Cryptographers `https://www.iacr.org/authors/tikz/`

Fin réunion de lancement !