Manuel d'utilisation

La description de l'utilisation de notre générateur d'IP qui suit a été faite en se basant sur la version **14.7 de la suite ISE de Xilinx** et **2.7 de Python**.

Sommaire

I. Présentation de l'IP

Avant de présenter le générateur et les modalités de passage sur FPGA, intéressons nous d'abord au fonctionnement des crypto-processeurs générés par notre générateur.

A. Versions disponibles

Nos crypto-processeurs ont pour mission de chiffrer ou de déchiffrer le message qu'on leur fournit en entrée (avec à une clé également fournie en entrée) selon l'algorithme de chiffrement Simon.

Vous pouvez choisir de générer l'une des versions suivantes, définies par le couple ***taille du message / taille de la clé*** :

Taille du message	Taille de la clé
32	64
48	72
	96
64	96
	128
96	96
	144
128	128
	192
	256

Nous vous offrons également la possibilité d'adapter le crypto-processeur à vos besoins selon les options suivantes :

- chiffrement seul
- déchiffrement seul
- chiffrement/déchiffrement

En ce qui concerne la politique d'utilisation de la clé, il vous est proposé d'avoir la possibilité de garder la même clé pour plusieurs chiffrements/déchiffrements successifs.

B. Interface et fonctionnement de l'IP

L'interface de nos crypto-processeur dépend des options que vous aurez choisi. Cependant, nos crypto-processeurs comportent au minimum cinq entrées et deux sorties comme on peut le voir sur le schéma ci-dessous :

Figure 1 : interface minimale du crypto-processeur

Les entrées "principales", ***msg_in*** et ***key***, permettent, pour la première de fournir le message que vous voulez chiffrer ou déchiffrer et la seconde de fournir la clé de chiffrement/déchiffrement. L'entrée ***start_stop*** permet de signaler à l'IP que vous souhaitez l'utiliser. Ce signal est à maintenir à l'état haut jusqu'à réception du message en sortie de l'IP. Les sorties, ***eoc*** et ***msg_out***, permettent respectivement de signaler la fin de l'opération de chiffrement et de récupérer votre message chiffré ou déchiffré. La taille des entrées ***msg_in*** et ***key*** et de la sortie ***msg_out*** dépendent de la version que vous avez générée. Le ***reset*** est actif à l'état haut.

Le tableau ci-dessous présente les entrées qui peuvent être ajoutées en fonction des options choisies :

Option	Chiffrement seul		Déchiffrement seul		Chiffrement/déchiffrement	
Politique gestion clé	redonner la clé	garder la clé	redonner la clé	garder la clé	redonner la clé	garder la clé
cipher_sig						
new_key						

L'entrée ***cipher_sig*** est présente lorsque vous avez générée une IP de chiffrement/déchiffrement, et permet de signaler si l'on souhaite chiffrer (état haut) ou déchiffrer (état bas). L'entrée ***new_key*** est présente lorsque vous avez demandé la possibilité de conserver la même clé pour plusieurs chiffrement ou déchiffrement.

Le chronogramme ci-dessous vous présente le fonctionnement global du crypto-processeur:

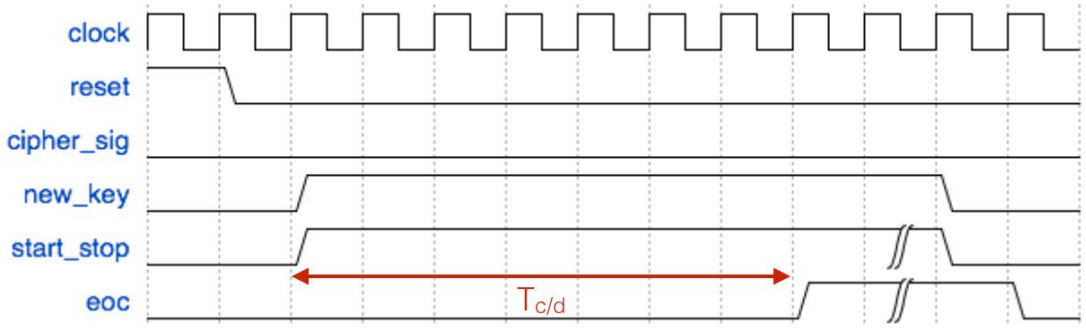

Figure 2 : chronogramme de l'IP (ici déchiffrement)

Les signaux ***cipher_sig***, ***new_key*** et ***start_stop*** doivent être maintenus tout au long du processus de chiffrement. Le message et la clé doivent être maintenus seulement deux périodes d'horloge.
Selon la version générée, le temps de réponse n'est pas le même. Le résultat est obtenu au bout de $T \times T_{clock} + 3 \times T_{clock}$ pour toutes les versions sauf dans le cas d'un déchiffrement avec un nouvelle clé où il est obtenu au bout de $2T \times T_{clock} + 3 \times T_{clock}$, T étant le nombre de rondes (que vous trouverez à la fin de ce manuel).
Une fois que le signal start_stop passe à l'état bas, accusant de la réception du message en sortie de l'IP, une période d'horloge est nécessaire avant de pouvoir utiliser l'IP.

II. Générateur

Le générateur prend en entrée la taille du message, la taille de la clé, l'option de chiffrage et la politique d'utilisation de la clé, pour générer la description VHDL de l'IP voulue par l'utilisateur :

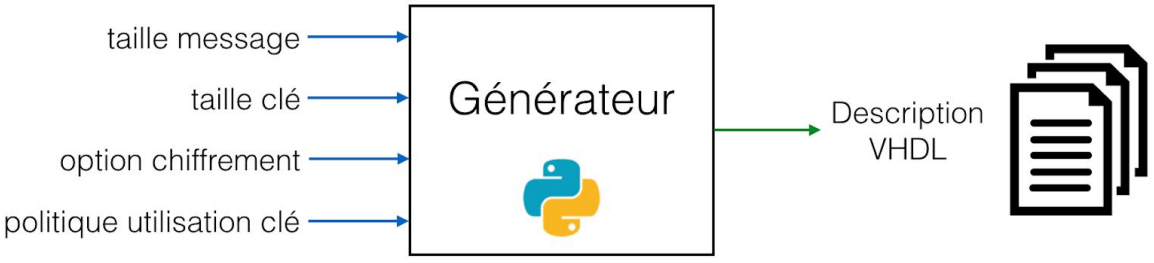

Figure 3 : fonctionnement du générateur

La ligne de commande qui permet d'exécuter le générateur est la suivante :

```
python generator.py -k "taille clé" -b "taille message" -p "option de chiffrement" -opt "politique d'utilisation de la clé"
```

Les options de chiffrement sont les suivantes :

- **cd** : chiffrement/déchiffrement
- **c** : chiffrement seul

- **d** : déchiffrement seul

Les politiques d'utilisation de la clé sont les suivantes :
- **c** : possibilité de garder la clé
- **r** : la clé doit être rechargé à chaque chiffrement/déchiffrement

Le tableau ci-dessous reprend ces options :

Options de chiffrement	Commande
chiffrement seul	-p c
déchiffrement seul	-p d
chiffrement/déchiffrement	-p cd
conservation de la clé	-opt c
recharger la clé à chaque chiffrement	-opt r

Par exemple, pour lancer la génération un crypto-processeur qui prend un message de 32 bits, une clé de 64 bits, capable de chiffrer et déchiffrer (cd) ainsi que de conserver la même clé pour plusieurs chiffrement ou déchiffrement (c), il vous suffit de taper une ligne de commande (dans le dossier du générateur) comme la suivante:

```
$ python generator.py -b 32 -k 64 -p cd -opt c
```

La description VHDL générée se trouve dans un dossier nommé **cryptoProc** à la racine du générateur.

III. Scripts

Dans l'archive du projet, nous vous fournissons deux scripts:
- ***script_synth32_64.sh*** : permettant de faire la génération de l'IP 32/64, une compilation, une première simulation avant synthèse, la synthèse avec génération de la netlist, une nouvelle compilation et enfin une simulation après synthèse. Pour modifier les options de générations, il vous suffit de modifier la ligne de commande du générateur dans le script, comme expliqué dans la partie précédente.

Note : si vous souhaitez lancer l'interface de simulation pour voir le chronogramme, il faut remplacer les lignes de simulation dans le script par :

```
./projetNum.exe -gui -wdb projetNum.wdb
```

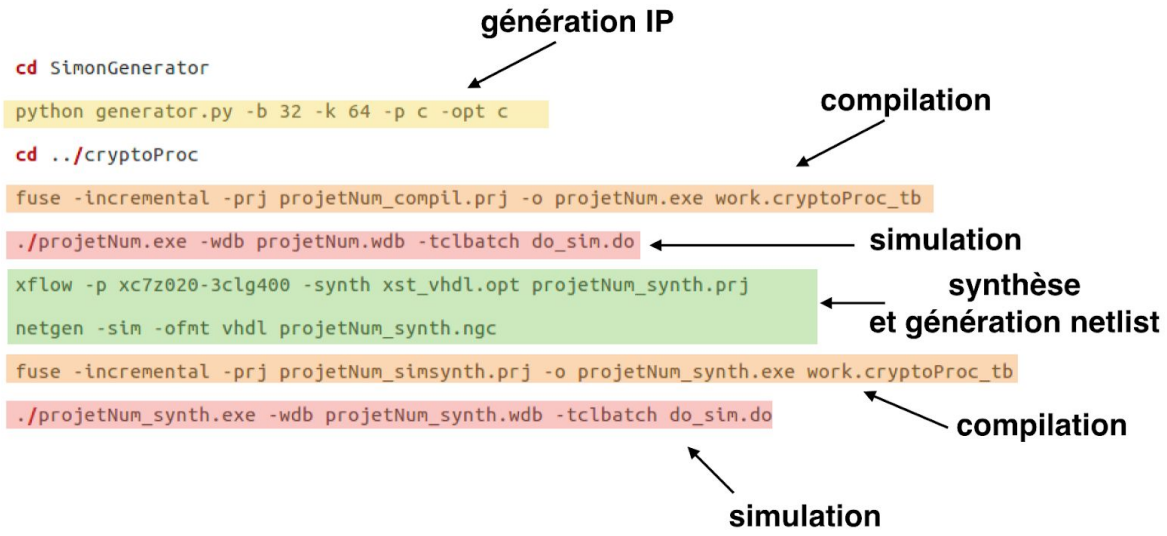

Figure 4 : script script_synth32_64.sh

Attention : il faut impérativement adapter le source au début du script à votre machine !

- ***clean.sh*** : permettant de supprimer tous les fichiers et dossier générés. Il n'est **pas** nécessaire de faire appel à ce script entre deux générations, car le générateur ré-écrase automatiquement les fichiers et dossier générés lors d'une précédente utilisation.

Pour lancer un script, il vous faut taper sur le terminal, à la racine du générateur :

```
$ bash #nom du script#.sh
```

IV. Implémentation sur FPGA

A. Chargement de l'IP sur la carte FPGA

Pour implémenter votre IP sur une carte FPGA, nous vous proposons d'utiliser le logiciel IMPACT de Xilinx. Nous avons utilisé la carte **ZedBoard Zynq-7000**, le brochage de la carte (inclu dans le fichier .bit) correspond donc à cette carte. Si vous souhaitez utiliser une autre carte, il vous faudra modifier le fichier .ucf contenu dans le dossier généré.

Pour commencer, tapez la ligne de commande suivante:

```
$ source /softslin/configCAO/Xilinx/start_impact147.sh
```

L'interface du logiciel IMPACT apparaît alors sur l'écran.
Créez un nouveau projet, puis sélectionnez la *ZedBoard "xc7z020"* avant de continuer. Double-cliquez sur le composant *"xc7z020"* et charger le fichier .bit. Ensuite, faite un clic droit sur le même composant puis sélectionnez "programmer", pour programmer la carte FPGA.

Le brochage tel que défini dans le fichier .ucf correspond aux entrées mentionnées sur cette photo :

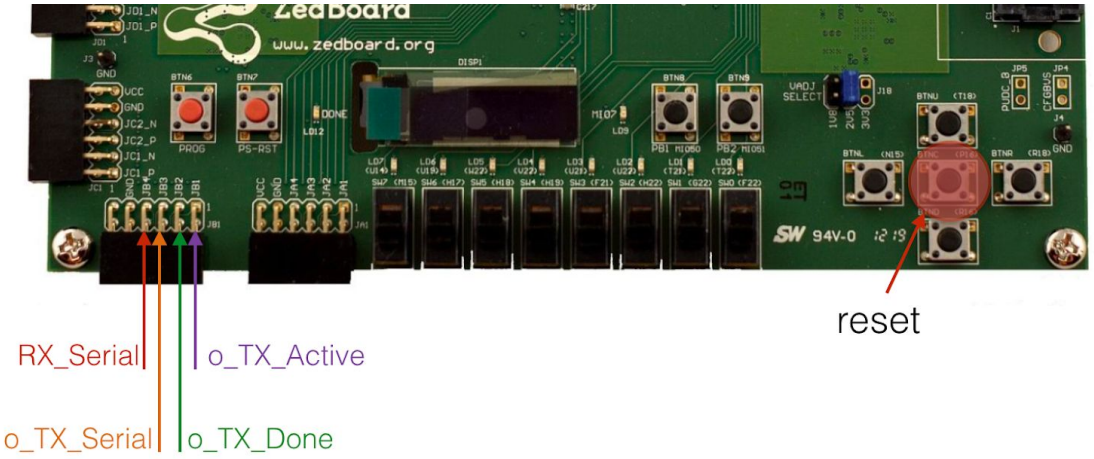

Figure 5 : connections sur la carte FPGA

Normalement, les fils du câble RS232 sur la ZedBoard correspondent à :

- fil noir → GND
- fil orange (entrée) → JB4
- fil jaune (sortie) → JB3

L'IP est à présent chargée sur la ZedBoard. Il ne vous reste plus qu'à envoyer votre message.

B. Envoi et réception des messages depuis l'ordinateur

Pour cela, nous allons utiliser un programme Python (python 2.7). Vous trouverez dans le dossier "**dialogue UART (.py)**" quelques exemples de fichiers Python :

```
chif_new_cle.py       --chiffrage nouvelle clé
zero.py          --accusé de réception
chif_m_cle.py         --chiffrage même clé
dechif_m_cle.py --déchiffrage même clé
dechif_new_cle.py     --déchiffrage nouvelle clé
```

Voici le contenu typique de ces fichiers :

```
#importation des librairies utiles
import time
import serial

# configuration de la connection série
ser = serial.Serial(
    port='/dev/ttyUSB0', # à modifier selon les ports USB disponible
sur votre machine
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
```

```
    timeout=1
)

#ouverture de la liaison série
ser.isOpen()

idn=(0x02)
length= (0x04)
mode= (0x03) # write
data= (0x19)

##envoi de la trame
# envoi de la clé
ser.write(chr(0x00))
ser.write(chr(0x01))
ser.write(chr(0x02))
ser.write(chr(0x03))
ser.write(chr(0x08))
ser.write(chr(0x09))
ser.write(chr(0x0a))
ser.write(chr(0x0b))
ser.write(chr(0x10))
ser.write(chr(0x11))
ser.write(chr(0x12))
ser.write(chr(0x13))
ser.write(chr(0x18))
ser.write(chr(0x19))
ser.write(chr(0x1a))
ser.write(chr(0x03))
# envoi du message
ser.write(chr(0x13))
ser.write(chr(0xd6))
ser.write(chr(0x31))
ser.write(chr(0x50))
ser.write(chr(0xfa))
ser.write(chr(0x1a))
ser.write(chr(0x0b))
ser.write(chr(0x21))
# envoi des signaux de contrôle
ser.write(chr(0x05))

# réception du résultat
serial_readline = ser.readline()
time.sleep(0.01)
print(serial_readline.encode("hex")) # affichage du résultat sur le
terminal
#fermeture de la liaison série
ser.close()

print("fin de la routine")
exit
```

La première étape consiste à modifier ces fichiers pour pouvoir envoyer vos trames. Les trames sont définies comme suit :

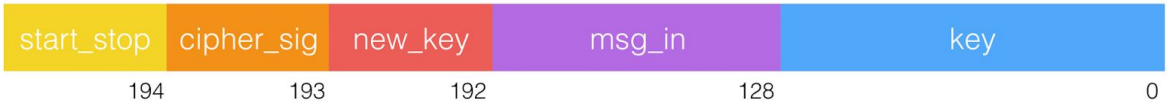

Figure 6 : trame d'envoi

Les signaux de contrôle peuvent varier selon la version choisie. Si votre version prévoit moins de signaux de contrôle, il vous suffit de supprimer le signal de contrôle de la trame sans changer l'ordre des informations (message, clé, et autres signaux de contrôles).

Note : sur les fichiers Python, il faut entrer la trame en partant des bits de poids faible.La première ligne d'envoie correspond au bit de poids faible la dernière, au bit de poids fort.

Les trois dernier bits, start_stop, cipher_sig et new_key sont regroupé dans cette version d'IP, en un octet. On a donc dans le fichier python, la dernière ligne d'envoie qui correspond à ces trois bits. Voici un tableau récapitulatif de la valeur à mettre dans cette dernière ligne dans le cas d'une IP complète:

Option	Code hexadécimal
déchiffrement même clé	04
déchiffrement nouvelle clé	05
chiffrement même clé	06
chiffrement nouvelle clé	07

Il ne vous reste plus qu'à envoyer la trame à la carte FPGA.
Il vous suffit de taper sur le terminal :

$ python2.7 #nom_du_fichier_python#.py

Vous verrez alors le résultat du chiffrement/déchiffrement s'afficher sur le terminal (ici, le nom du fichier python est dechif_new_key):

$ python2.7 dechif_new_cle.py
Cyphertext : 756e64206c696b65 **← résultat de l'opération**
fin de la routine

ANNEXE 1 : tableau des 10 versions de base de la famille Simon

Note : cette annexe est utile pour le calcul des temps de réponse. Utilisez la dernière colonne (nombre de rondes T).

Taille du bloc **2n**	Taille de la clé **mn**	Nombre de rondes **T**
32	64	32
48	72	36
	96	36
64	96	42
	128	44
96	96	52
	144	54
128	128	68
	192	69
	256	72

ANNEXE 2 : fréquences maximales d'utilisation

Suite aux simulations après synthèse voici les fréquences maximales d'utilisation théorique pour quelques version de crypto-processeur :

Version	Option	Politique clé	Fréquence max (MHz)
32/64	CD	C	432.751
48/72	CD	C	426.021
48/96	CD	C	426.021
64/96	CD	C	397.646
64/128	CD	C	397.646
96/96	CD	C	571.2
96/144	CD	C	558.815
128/128	CD	C	553.434
128/192	CD	C	508.156
128/256	CD	C	498.306

Version	Option	Politique Clé	Fréquence max (MHz)
64/128	C	R	588.859
		C	459.074
	D	R	458.19
		C	456.1
	CD	R	397.646
		C	397.646