TP de sécurité des systèmes embarqués

Hugues de Valon
Paul Luperini
Lucas Mahieu

23 janvier 2017

Table des matières

1 Introduction

De nos jours, les réseaux de neurones artificiels reprennent de plus en plus d'importance car la puissance de calculs disponible permet d'obtenir des résultats satisfaisant en temps raisonnable. Le traitement d'images, la reconnaissance vocale ou le traitements lexicaux sont des applications qui pourraient être intégrées dans des systèmes embarqués. Pour ce type d'application, il est possible d'implémenter sur des CPU ou GPU des algorithmes neuronales, mais la consommation et la vitesse de traitement deviendraient vite limitant.

Créer un composant électronique (ASIC ou une IP FPGA dans un premier temps) implémentant un réseau de neurones réduirait drastiquement sa consommation et améliorerait la vitesse du réseau par rapport à un CPU. De plus, étant donnée que l'IP serait spécialisé à cette application permettrait de rendre paramétrable dynamiquement le composant lui permettrait de s'adapter à de multiples applications.

Le projet "Réseau de neurones sur FPGA" s'inscrit dans ce cadre. Sous le tutorat de Frédéric Pétrot et Adrien Prost-Boucle, nous devons créer un tel composant, et tester ses performances en vue de le comparer à des systèmes existants tels que la puce Spinnaker ou TrueNorth d'IBM ou encore de systèmes en développement tel que les réseaux de neurones ternaires du laboratoire TIMA. Une fois implémenté et validé, nous utiliserons une carte FPGA Zedboard pour tester notre composant sur une application classique de reconnaissance de chiffres manuscrits, en utilisant la base de données MNIST.

2 TP 1

2.1 Quelle protections sont mises en place contre les attaques par fautes ?

2.2 Quelles sont les faiblesses de cette sécurité ?

2.3 Implémentation de l'attaque par faute

2.4 Embarcation du code

3 TP 2

3.1 Introduction

Dans cette partie nous allons effectuer une attaque par canaux auxiliaires sur un chiffrement AES. Cette attaque s'appelle Differential Power Analysis ou DPA, et consiste à étudier la consommation électrique du système visé sur de multiples exécutions et d'en déduire par des procédés statistiques l'information visée, ici la clé de chiffrement.

3.2 Théorie

La sécurité d'un système de chiffrement dépend de l'algorithme employé et de son implémentation sur circuit électronique. Dans le cas de la DPA, nous allons utiliser des faiblesses d'implémentations pour en déduire la clé de chiffrement employée.

La SBox (Substitution Box) est le composant non linéaire principal de l'AES. Il s'agit d'une substitution d'octets, cette opération est effectuée juste après l'ajout de la clé à la donnée que l'on veut chiffrer. Il est donc possible, à partir du résultat de la première itération de l'AES qui ne dépendent que des données d'entrées et de la clé, de retrouver la clé.

La DPA est une attaque par canaux cachés qui utilisent la consommation électrique du circuit pour valider ou invalider des hyptohèses sur la clé. La consommation d'un circuit logique dépend grandement des données traitées. Dans notre cas, il n'y a pas de contre-mesure

spécifique utilisée pour équilibrer la consommation. Ainsi, nous devrions être capables de réaliser une attaque DPA sur la SBox de l'AES.

3.3 Travail réalisé

Tout d'abord, nous devons transformer les fichiers VHDL qui nous ont été donnés en une description au niveau des transistors, pour pouvoir en extraire une consommation électrique simulée. Cette étape est réalisée à l'aide de l'environnement Cadence.

Une fois cette description générée, nous utilisons le simulateur de consommation électrique Synopsis Nanosim pour obtenir les profils de courants selon certains stimulis.

Le logiciel d'analyse DPA fourni nous permet de retrouver les clés utilisées. Il accepte deux paramètres : l'index du bit attaqué et le chemin vers le dossier contenant les fichiers de simulation. L'outil lit un fichier de configuration contenant une liste des vecteurs qui seront utilisés pour la DPA. Ensuite, les données simulées sont collectées à partir des fichiers de simulation. Pour chaque hypothèse de clé, il partitionne les traces de simulation de chaque message selon la valeur du bit attaqué. Enfin, il évalue les différences entre les moyennes de chaque partition et sélectionne la valeur la plus haute.

3.4 Résultats

Suite à nos expérimentations nous obtenons les résultats suivants :

Pour obtenir la clé qui a le plus de chances d'être la clé réelle, il suffit de prendre celle qui apparait le plus souvent sur les différents lancements du programme avec des bits différents.

Par exemple dans le cas de la clé 4A, celle qui est apparait le plus souvent est la 58 (0b111010).

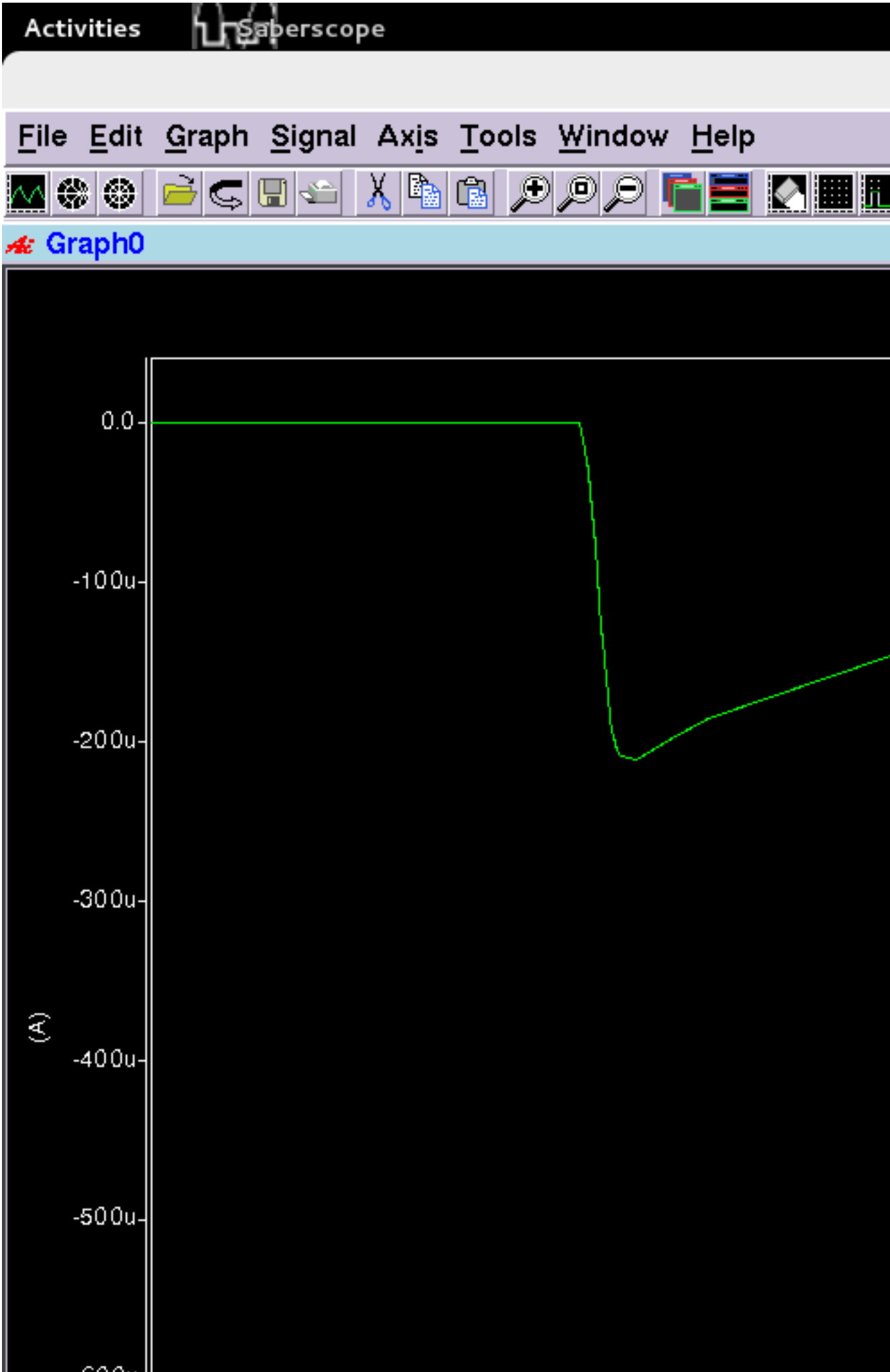
3.5 Conclusion

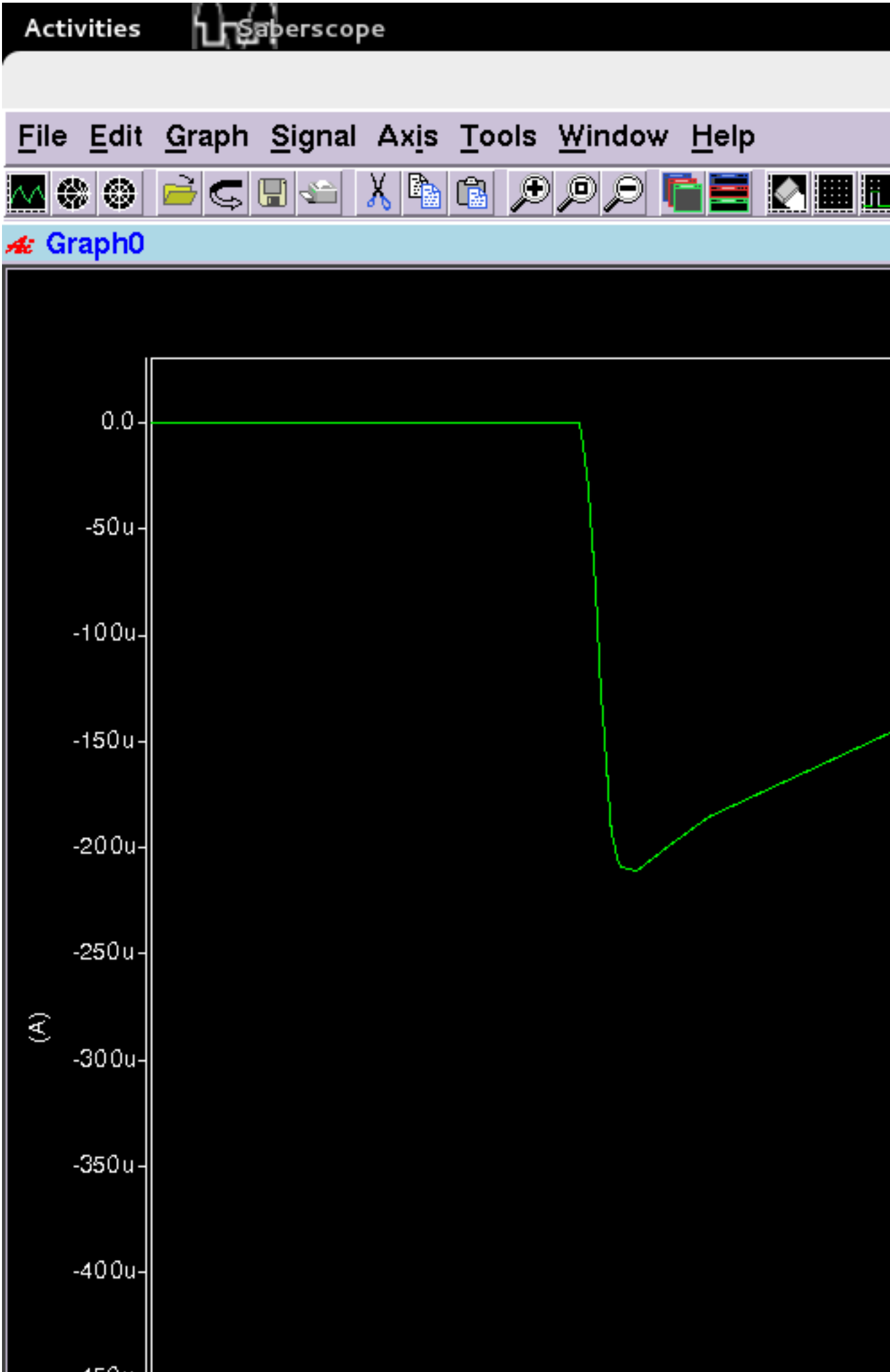
TP CLIC, TP QUI MARCHE, TP EASY.

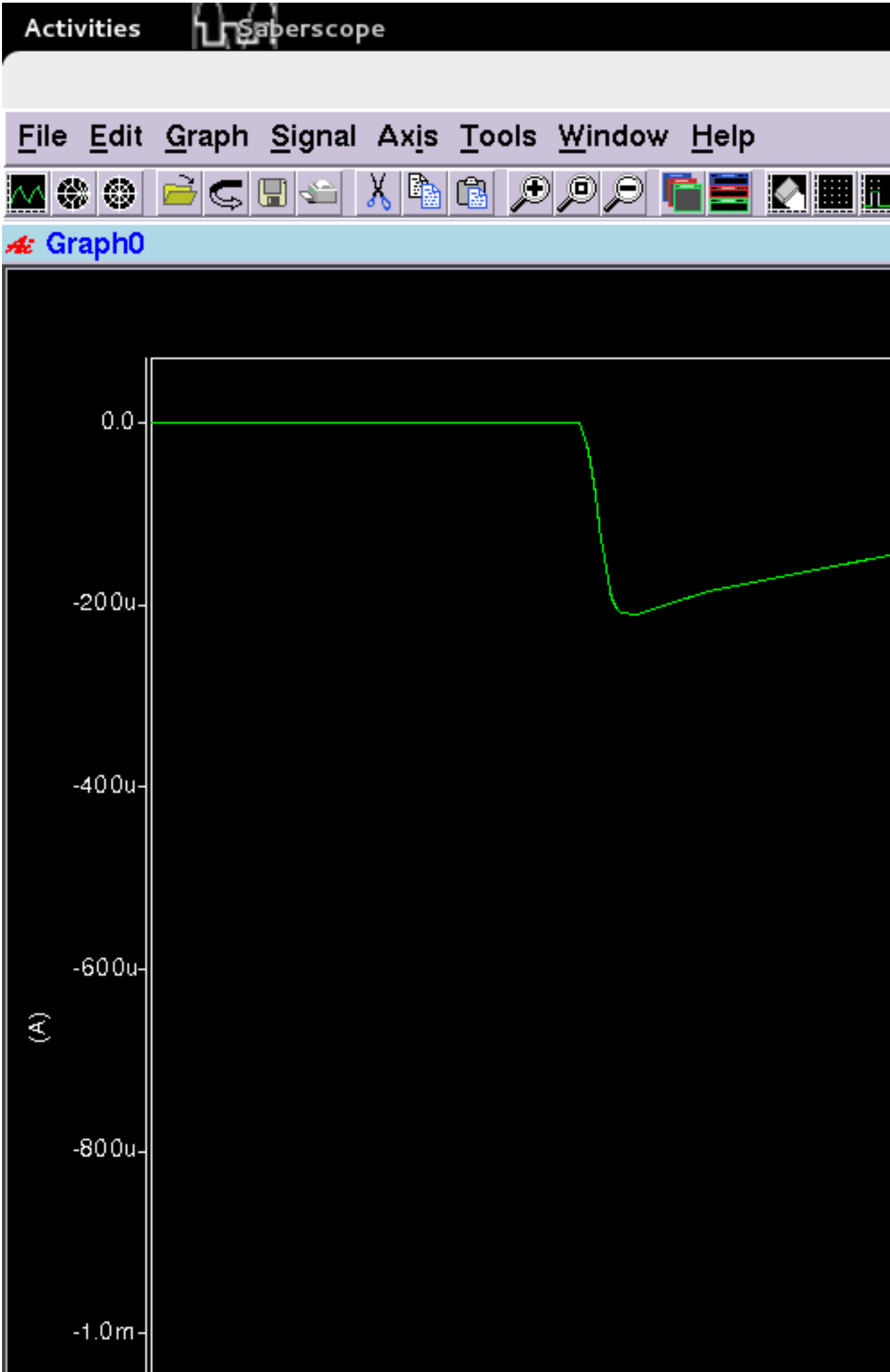
4 Conclusion

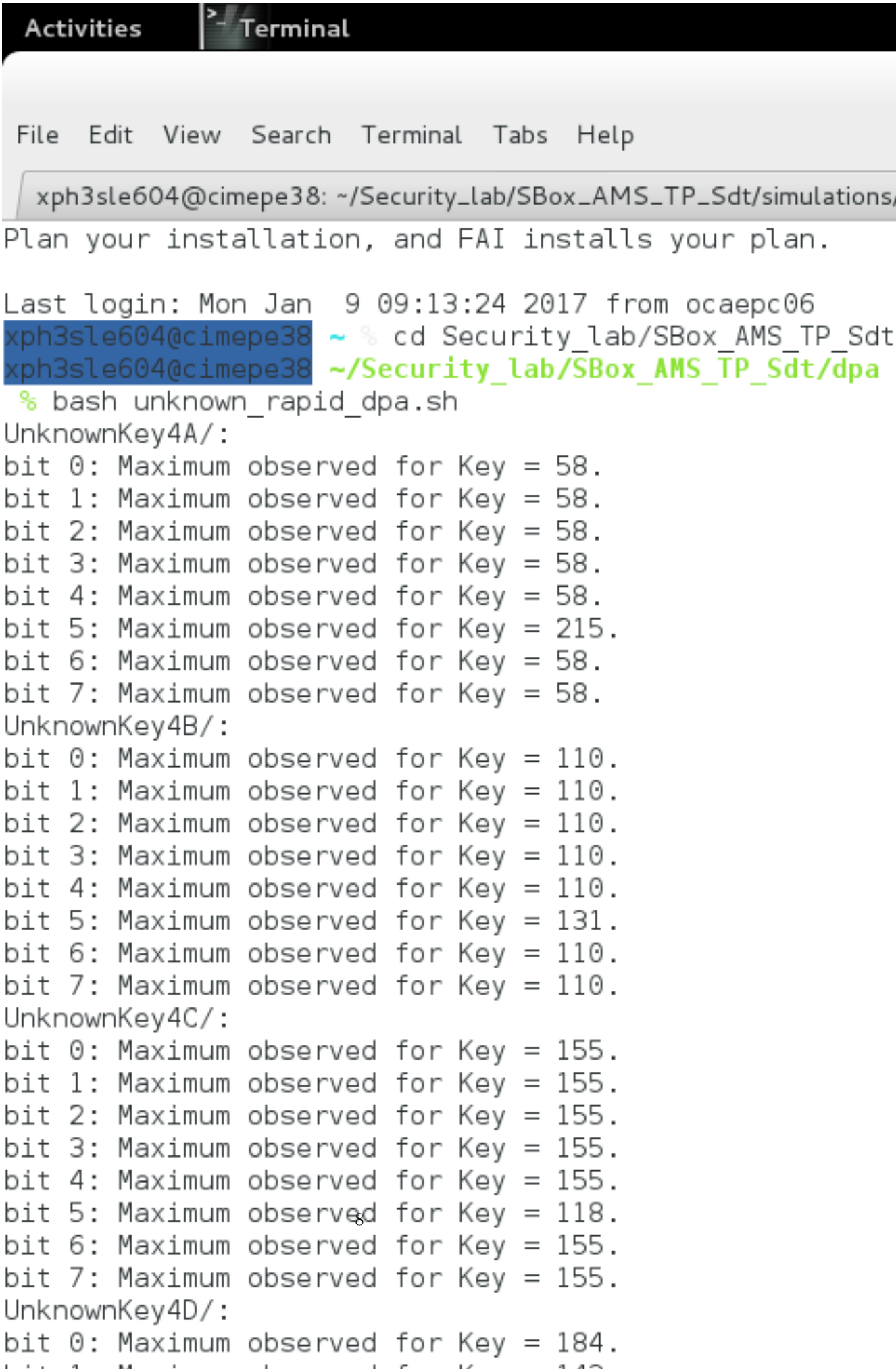
Merci à nos chers professeurs de nous avoir fait découvrir cette délicieuse matière. Merci beaucoup, du fond du coeur.

Guillaume Anoufa


Activities
Saberscope
File Edit Graph Signal Axis Tools Window Help
Graph0
0.0
-100u
-200u
-300u
-400u
-500u
(A)


Activities
Saberscope
File Edit Graph Signal Axis Tools Window Help
Graph0
0.0
-50u
-100u
-150u
-200u
-250u
-300u
-350u
-400u
(A)


Activities
Saberscope
File Edit Graph Signal Axis Tools Window Help
Graph0
0.0
-200u
-400u
-600u
-800u
-1.0m
(A)


Activities
Terminal
File Edit View Search Terminal Tabs Help
xph3sle604@cimepe38: ~/Security_lab/SBox_AMS_TP_Sdt/simulations/
Plan your installation, and FAI installs your plan.
Last login: Mon Jan 9 09:13:24 2017 from ocaepc06
xph3sle604@cimepe38 ~ % cd Security_lab/SBox_AMS_TP_Sdt
xph3sle604@cimepe38 ~/Security_lab/SBox_AMS_TP_Sdt/dpa
% bash unknown_rapid_dpa.sh
UnknownKey4A/:
bit 0: Maximum observed for Key = 58.
bit 1: Maximum observed for Key = 58.
bit 2: Maximum observed for Key = 58.
bit 3: Maximum observed for Key = 58.
bit 4: Maximum observed for Key = 58.
bit 5: Maximum observed for Key = 215.
bit 6: Maximum observed for Key = 58.
bit 7: Maximum observed for Key = 58.
UnknownKey4B/:
bit 0: Maximum observed for Key = 110.
bit 1: Maximum observed for Key = 110.
bit 2: Maximum observed for Key = 110.
bit 3: Maximum observed for Key = 110.
bit 4: Maximum observed for Key = 110.
bit 5: Maximum observed for Key = 131.
bit 6: Maximum observed for Key = 110.
bit 7: Maximum observed for Key = 110.
UnknownKey4C/:
bit 0: Maximum observed for Key = 155.
bit 1: Maximum observed for Key = 155.
bit 2: Maximum observed for Key = 155.
bit 3: Maximum observed for Key = 155.
bit 4: Maximum observed for Key = 155.
bit 5: Maximum observed for Key = 118.
bit 6: Maximum observed for Key = 155.
bit 7: Maximum observed for Key = 155.
UnknownKey4D/:
bit 0: Maximum observed for Key = 184.
bit 1: Maximum observed for Key = 143.