Diploma Thesis

Κοζάνη // Ιούλιος 2024

Σχεδιασμός και υλοποίηση SoC σε αρχιτεκτονική ARM

Εργαστήριο Ρομποτικής, Ενσωματωμένων και Ολοκληρωμένων Συστημάτων

Νίκου Ραφαήλ

Επιβλέπων: Δασυγένης Μηνάς

http://arch.ece.uowm.gr/

Πίνακας Περιεχομένων

"Οι άνθρωποι δεν ενδιαφέρονται για το τι λες, ενδιαφέρονται για το τι φτιάχνεις".

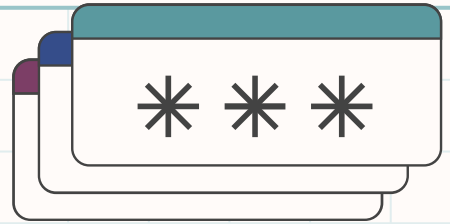
—Mark Zuckenberg,
Founder & CEO, Facebook 2024

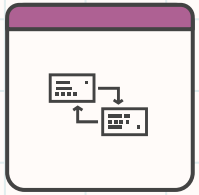
01

Εισαγωγή

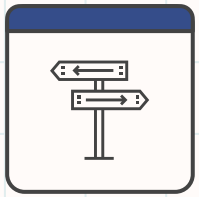
- ✓ **Σκοπός της Υλοποίησης**
- ✓ **Παρόμοια Συστήματα**

Σκοπός της Υλοποίησης





Δημιουργία ενός περιφερειακού συστήματος SHA-256 σε FPGA



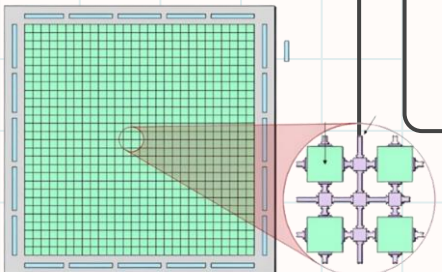
Ενσωμάτωση του με έναν επεξεργαστή αρχιτεκτονικής τύπου ARM

Χρήση του περιφερειακού με προγραμματισμό σε Python στον επεξεργαστή

Παρόμοια Συστήματα

Δημιουργία συστήματος κρυπτογράφησης

- Kria KR260 Robotics Starter Kit
- Zynq® UltraScale+ MPSoC-based
- Axi Interconnect
- Vitis HLS
- Βιβλιοθήκη Pynq Python



Αλγόριθμος SHA-256

Υλοποίηση σε HDL στην πλακέτα Spartan 6 SP605 Evaluation Board [3]

Υλικό/Λογισμικό Σύστημα

Σύστημα για εξοικονόμηση ενέργειας στην πλακέτα Xilinx Zynq 7000 [2]

Πλακέτες Virtex 4,5,6

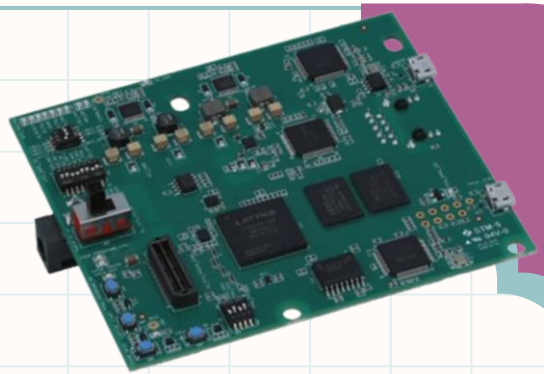
Ένας επεξεργαστής βασισμένος σε FPGA για τον αλγόριθμο ασφαλούς κατακερματισμού SHA-256 [4]

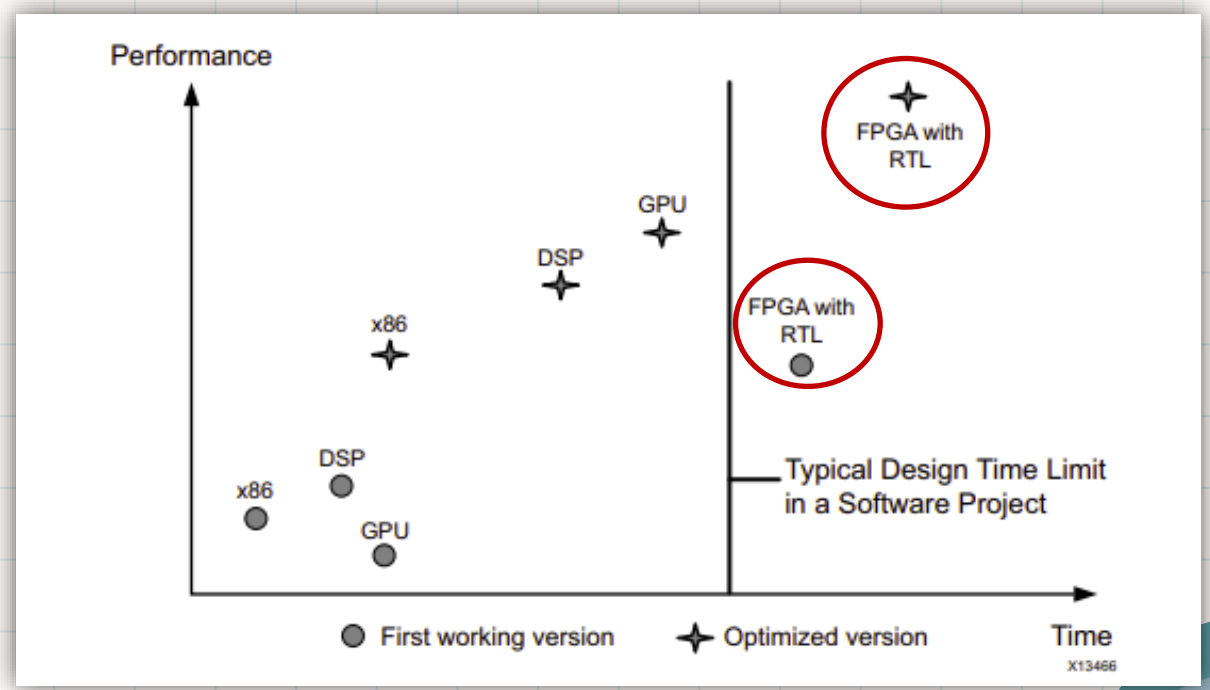
02

Θεωρητικό Υπόβαθρο

- ✓ Εξέλιξη και χρήση των FPGA
- ✓ Προγράμματα που αξιοποιήθηκαν

Θεωρητικό Υπόβαθρο 1/2





- Ταχύτητα
- Παραμετροποίηση
- Ευελιξία
- Ασφάλεια
- Παραλληλισμός
- Κατανάλωση Ενέργειας
- Κόστος

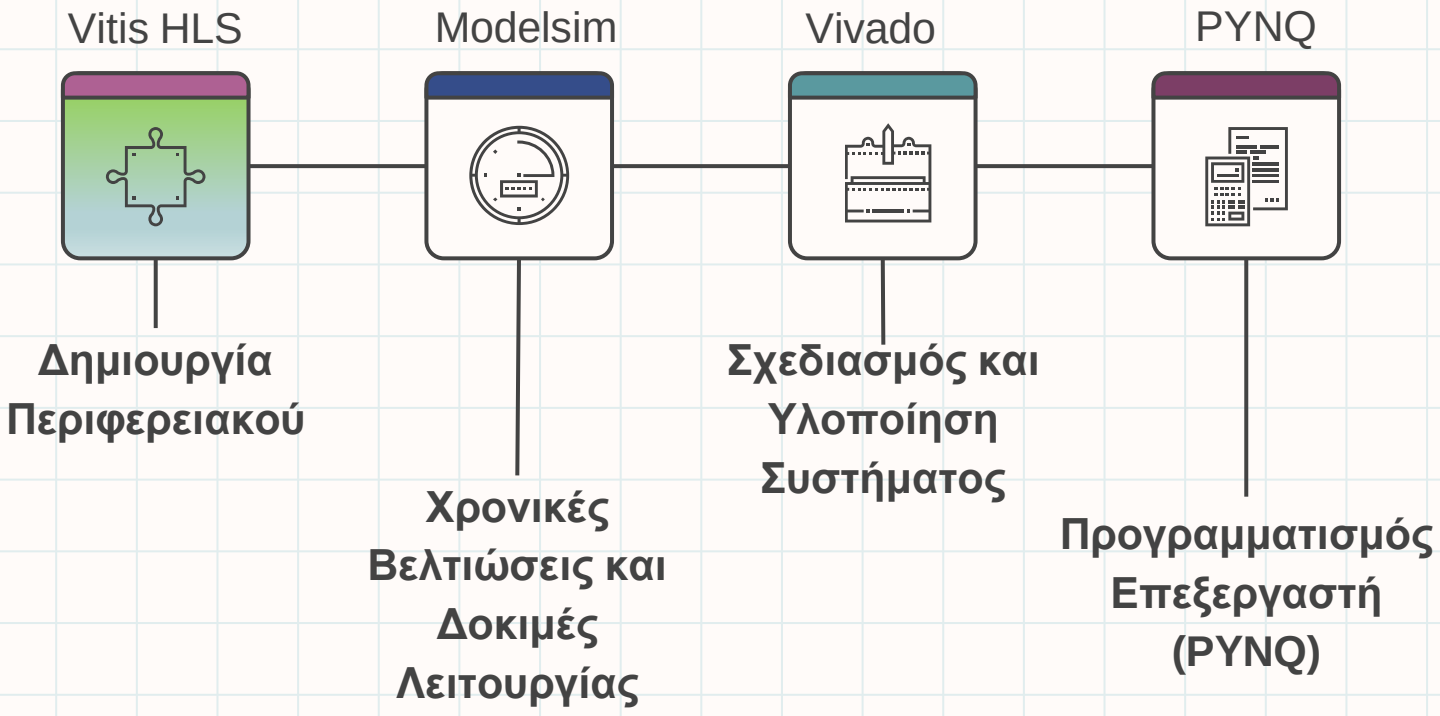
Θεωρητικό Υπόβαθρο 2/2

03

Ανάλυση Αλγορίθμου

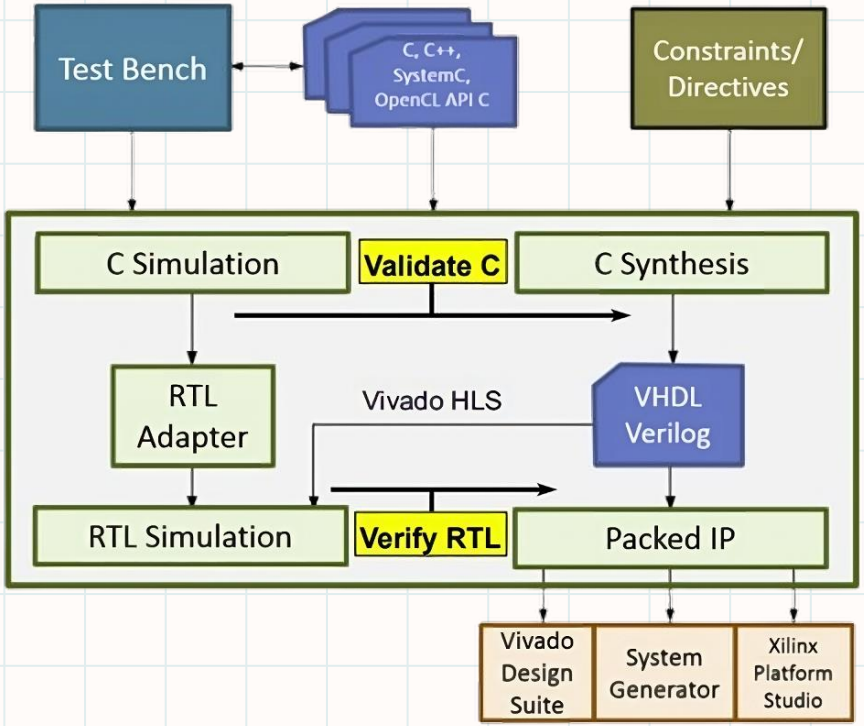
- ✓ Ανάλυση Προγράμματος Vitis HLS
- ✓ Παρουσίαση Αλγορίθμου Κρυπτογράφησης
- ✓ Δοκιμές Επαλήθευσης

Εξέλιξη Συστήματος Κρυπτογράφησης



Vitis HLS

- Το Vivado HLS (High Level Synthesis) της Xilinx - AMD® είναι ένα εργαλείο σχεδίασης που επιτρέπει τη δημιουργία και υλοποίηση application-specific IPs χρησιμοποιώντας γλώσσες προγραμματισμού υψηλού επιπέδου, όπως η C και η C++.



SHA-256

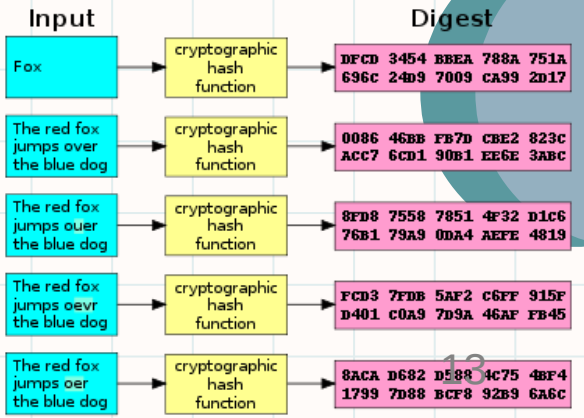
Ο SHA-256 (Secure Hash Algorithm 256-bit) είναι ένας από τους πιο ευρέως χρησιμοποιούμενους αλγορίθμους κατακερματισμού που αναπτύχθηκε από την Εθνική Υπηρεσία Ασφάλειας των ΗΠΑ (NSA) και δημοσιεύτηκε από το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST). Είναι μέρος της οικογένειας αλγορίθμων SHA-2.

Μέγεθος Μηνύματος

Μη αντιστρέψιμη διαδικασία (Fingerprints)

Μήκος Σύνοψης (Digest)

- Αλγόριθμος με μαθηματικές πράξεις χωρίς κλειδί
- Αντοχή σε Επίθεση
- Ασφάλεια



Εφαρμογές του αλγορίθμου SHA-256

SHA-256

- Αυθεντικοποίηση πακέτων λογισμικού Debian
- Unix & Linux για ασφαλή κατακερματισμό κωδικών πρόσβασης
- Κρυπτονομίσματα (Bitcoin) επαλήθευση συναλλαγών
- Πρωτόκολλα ασφαλείας

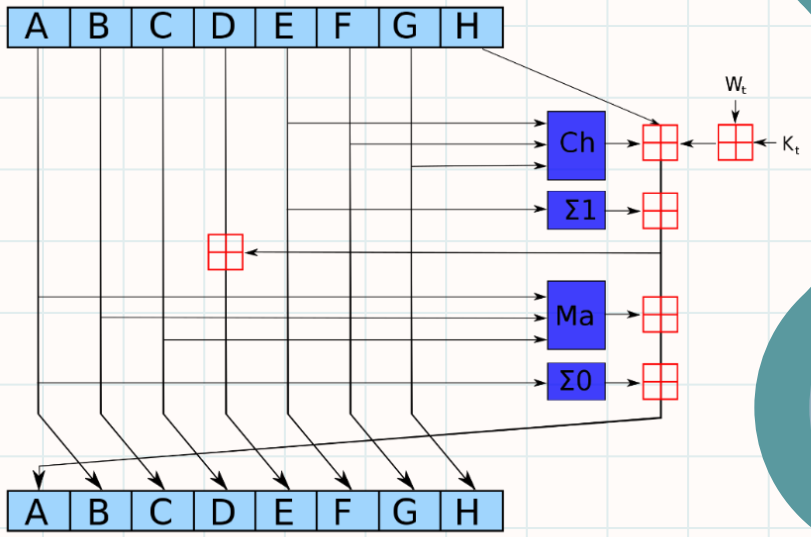
Βασικές Λειτουργίες του SHA-256 1/2

Οι βασικές λειτουργίες που εκτελούνται είναι οι ακόλουθες:

- Οι λογικές πράξεις **AND, XOR και OR** αναπαρίστανται με τα σύμβολα **∧, ⊕, |** .
- Οι πράξεις **συμπληρώματος κατά δύο** συμβολίζονται με το σύμβολο ¬.
- Η ακέραια πρόσθεση **modulo** 2^{32} που αναπαρίσταται με το σύμβολο **+**.
- Η **συνένωση δύο δυαδικών λέξεων** προσδιορίζεται από το σύμβολο #.
- **SHIFTRIGHT (x, c) = ((x) >> (c)),** χρησιμοποιείται για να αναπαραστήσει την πράξη δυαδικής μετατόπισης κατά δεξιά μιας δυαδικής λέξης x κατά ένα ακέραιο ποσό c bits, αποβάλλοντας τα δεξιότερα c bits και αντικαθιστώντας τα με μηδενικά στα αριστερά.
- **ROTATIONRIGHT (x, c) = (((x) >> (c)) | ((x) << (32 - (c)))),** είναι η πράξη περιστροφής προς τα δεξιά (κυκλική δεξιά μετατόπιση) ,όπου τα δεξιότερα c bit δεν απορρίπτονται. Αντίθετα συμπληρώνονται στην αριστερή πλευρά της δυαδικής λέξης.

Βασικές Λειτουργίες του SHA-256 2/2

- 64 δυαδικές λέξεις W(64) των 32 bit που πηγάζουν από το δεκαδικό μέρος των κυβικών ριζών των πρώτων 64 πρώτων αριθμών.
- Υπάρχουν 2 τριμερείς συναρτήσεις που δέχονται 3 εισόδους και 4 μοναδιαίες. (ROTATIONRIGHT (a, b) , CH (x, y, z) , MAJ (x, y, z) , EP0 (x) , EP1 (x) , SIG0(x) , SIG1(x))
- Για να διασφαλιστεί ότι το μήνυμα είναι ακέραιο πολλαπλάσιο του 512 bit, το μήνυμα συμπληρώνεται με μια μορφή padding αν χρειαστεί. Συγκεκριμένα, συμπληρώνεται με μηδενικά μέχρι να έχει μήκος 448 bits.
- Διαδικασία της Συμπίεσης = πράξεις + μετατοπισμός προς τα δεξιά των bit



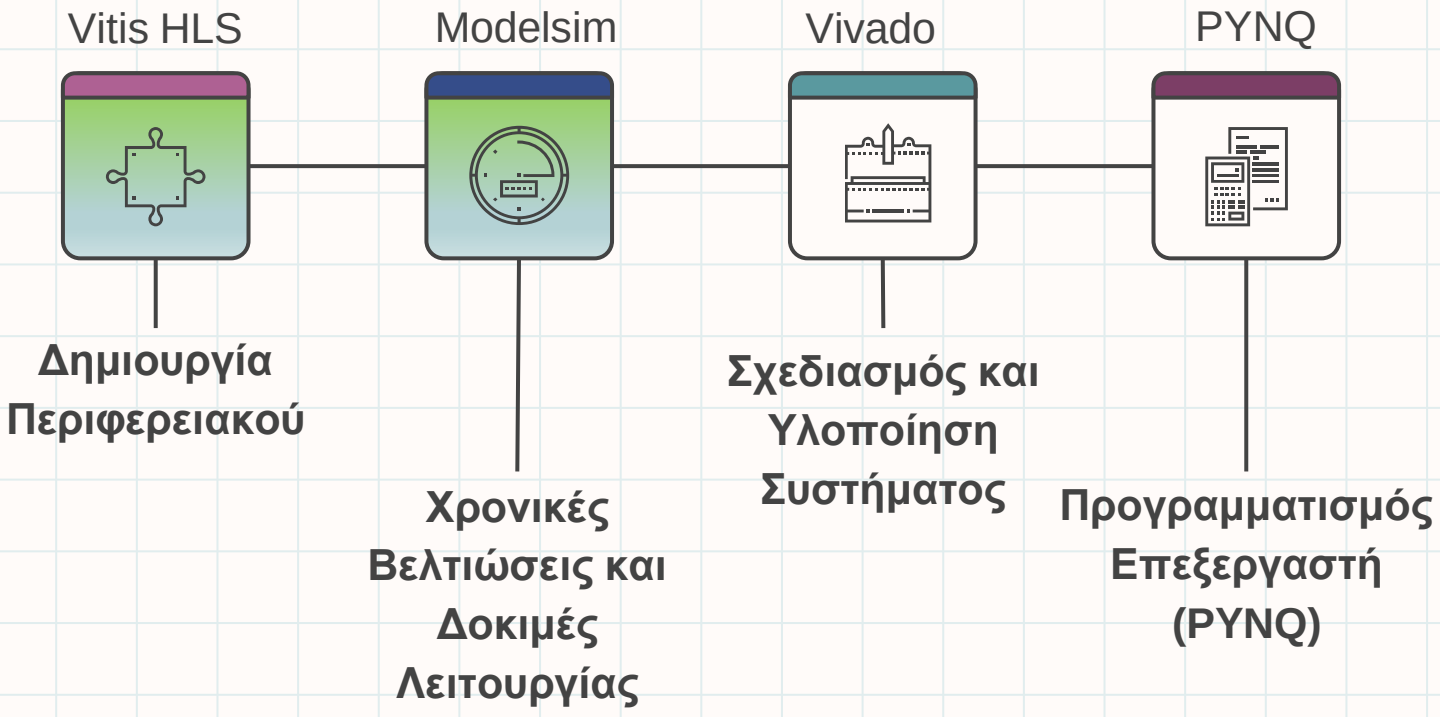
Το κόκκινο σχήμα ⊞ είναι η πρόσθεση modulo 2^{32} για SHA-256, ή 2^{64} για SHA-512

04

Βελτίωση & Επικύρωση

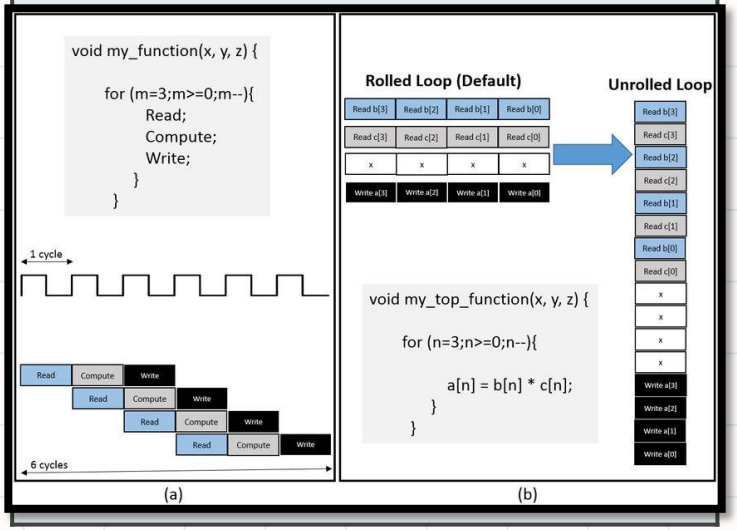
- ✓ Βελτιστοποίηση περιφερειακού HLS
- ✓ Δημιουργία δυναμικού Testbench
- ✓ Σύγκριση αποτελεσμάτων στο Modelsim

Εξέλιξη Συστήματος Κρυπτογράφησης

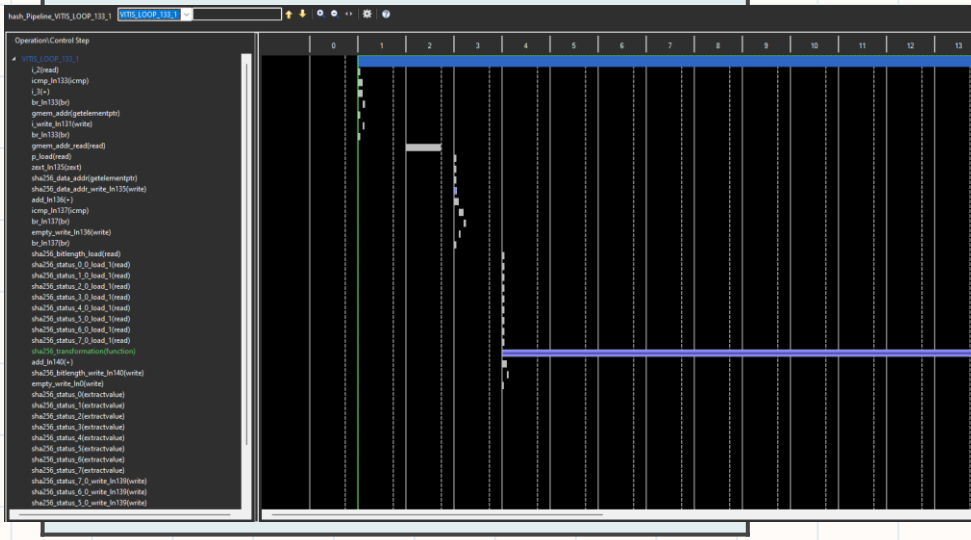


Βελτιστοποίηση περιφερειακού HLS

Directives



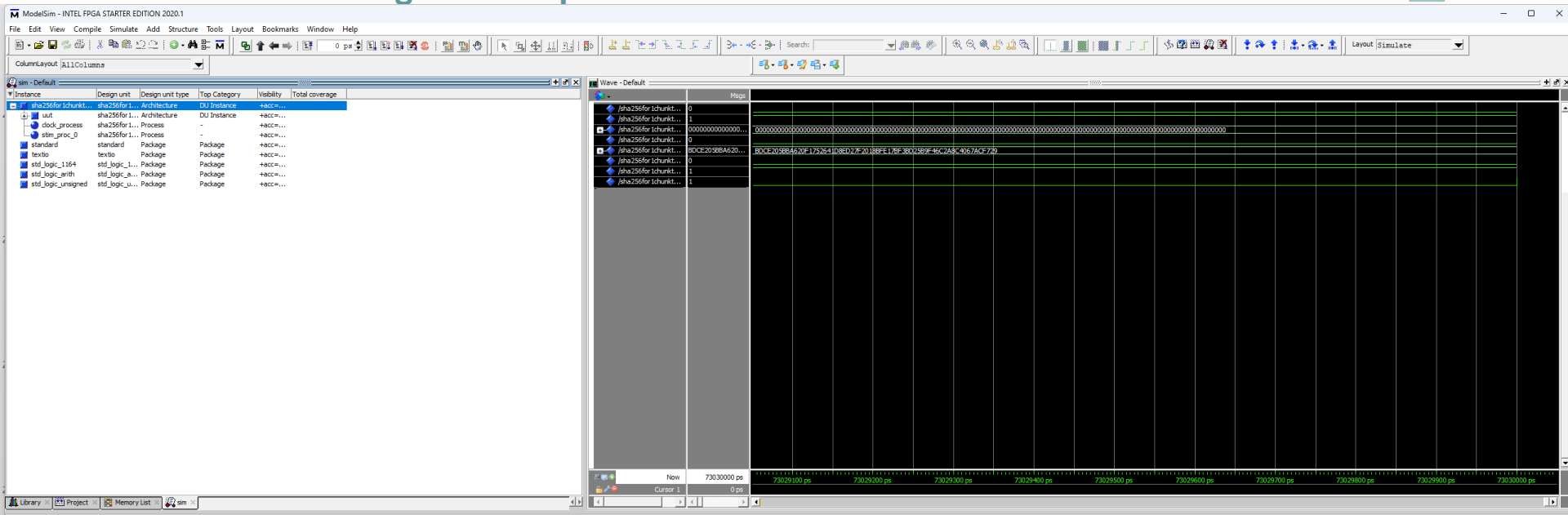
Εκτέλεση Assembly



100 τυχαία αποτελέσματα για 3ψήφια τυχαία strings

```
INH: 6d746778756d73746a66, OUThash: 65d3dc7d4e1c68be8a11bb728e439b2249508aad94aafbd8ad2c29b75215e1fe
INH: 7567657277746d6d6e68, OUThash: 986518a1b4df3b813fe3c39a912e24e99c2154669075c4743b1cb59fc29b7d84
INH: 6a69626a716366686f78, OUThash: 0139e7c1a88677952f7418d89038b8f1cf7058f04d6b87c0233753e96c8560da
INH: 6e636c67726161786365, OUThash: 0b0747389174abf1be96820533ba9663ed88a18f3aa0b9a9d34241a62a7b6750
INH: 6a64686163686378716b, OUThash: 62f28b140df7e6eb504cd22f59268171d0439614b501b950b864c7084e029a74
INH: 6f6767776f716e726f69, OUThash: 4307fcd6f6844cc8c75f54ecd86f88f412ac5445f28a7e558417f983a32dcb00
INH: 706a6e696f70636e7672, OUThash: be6d91629b02edc5c5422eb83223dfbd5e1f1d26c78b99aa6206abb76920bf73
INH: 6e617a647870656a7870, OUThash: 5c6bec774a5f19a769d802100328257ec2cc9066b1fc5242e25c3917bf965f9c
INH: 71656173766472646362, OUThash: cba5728552ca066a8e3ff900db719216cb80defa3045b6973e93a3d31f9eddd0
INH: 687763626f7972716470, OUThash: f74c8da37b86cacceabe693f431c11222710cdb533656472bddf7d4e0c494550
INH: 7a796a69716c767a6867, OUThash: c6dd9df9841869b0d43b19cf336bb279aef153d935b86fc598029ccd4940debe
INH: 796a6c78767366726777, OUThash: 678cc7735f710ce9786c35213876de7fdad7daa87b02986fbfa923d58cea55e9
INH: 756f77627465787a7369, OUThash: d4166a00246810bf0551e3f463c83ba484b2896285899a20849fef25695e91f8
INH: 686f626d62746f6f7774, OUThash: 696a99fcab2ec1180c493e61a8e16a3424fea933a15fe3db8f989938110687c7
INH: 6e6d6572666977647369, OUThash: 95308b2b16afa570ac07be3e7828a5646ab009666b8937eedc08b84bf122b976
INH: 76756766727261676378, OUThash: e18f5b3f4f40000bb06fbd57f3c9a331233953a1a065230579d4fa585dd115ae
INH: 79747a74757a65647167, OUThash: ae74ae3fc621494f2a574d242b8ffe040d6c8ea4006a9da288733f863587cb57
INH: 6473756b6769676a796c, OUThash: 7623fdc0fbcec9a073a343e8145d43ab8164e9eed31604ff5c8b268a51216917
INH: 666f656a656f686c616b, OUThash: 9c0dbcda3e23174791611eb82d6d3b361ae4e75aaef176d98b6f0dcb1921f691
INH: 786279766977756e616e, OUThash: 7e1af60372b246d2d976ed6504fc1b9178d40211c0b1144827f78747b5742385
INH: 62726a6770736c68776b, OUThash: 7c98685dc4e7d740d77a5871502175bb31715bfd5248000e3b55d83891caa73a
INH: 6677616c67776a70736e, OUThash: b846c2703779a338ab97fb9f5b690452ed68a06808bbb3a667f1c445a9b9417f
INH: 6d6163637a676e616464, OUThash: bf7b1110c435b444a9f99a26af643e52456db6519c94bd6468dd94aee02d2f37
INH: 726e6b6e626e646b6977, OUThash: ceb11f75d198d350070d075710603ba3dc42c984456479df4fe497c6f96a5e9c
INH: 66687a7475756b796d6d, OUThash: 0456eab91ff68540a5dcbd98bab9d9a463287433f5b328f014bc0e4a3d53b0b4
INH: 617366676f7669776e73, OUThash: a36574a66bd9b7f5695a0d537193a93ed5e72f87e93963d75c618597450b869d
INH: 77696773666577637875, OUThash: 80eaee9e870e595153a079d747621b10cfd1595290d0084b99f8c6a2f57460c1
```

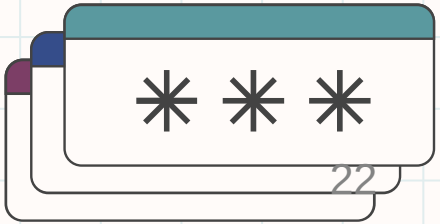
```vhdl
begin
reset <= '1';
wait for clock_period;
reset <= '0';
wait for clock_period;
load <= '1';
plain <= x"6c62788000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018";
wait for clock_period;
load <= '0';
plain <= (others => '0');
wait for clock_period*70;
assert (digest = x"d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b")
 report "Test failed for input 0 : Expected: d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b" severity error;
assert (digest /= x"d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b")
 report "Test succeeded for input 0 : Expected: d3bf8202ad139083e2c24a18771b65a55919215d123eab85963f83c699b4749b" severity note;
reset <= '1';
wait for clock_period;
reset <= '0';
wait for clock_period;
load <= '1';
plain <= x"7371768000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000018";
wait for clock_period;
load <= '0';
plain <= (others => '0');
wait for clock_period*70;
assert (digest = x"223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a")
 report "Test failed for input 1 : Expected: 223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a" severity error;
assert (digest /= x"223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a")
 report "Test succeeded for input 1 : Expected: 223403da4e6fa1411f331d5d845c94949be1cc882cffb5d3c37b5d4666e36c5a" severity note;
```


ModelSim - INTEL FPGA STARTER EDITION 2020.1
File Edit View Compile Simulate Add Structure Tools Layout Bookmarks Window Help
sim - Default
Instance
Design unit
Design unit type
Top Category
Visibility
Total coverage
sha256for1chunkt...
uut
clock_process
stim_proc_0
standard
textio
std_logic_1164
std_logic_arith
std_logic_unsigned
Wave - Default
Msgs
Now
Cursor 1
Transcript
** Note: Test succeeded for input 84 : Expected: a1a38b5bae5866d7d998a9834229ec2f9db7a4fc8fb6f58b1115a96a446875ff
** Note: Test succeeded for input 85 : Expected: 3615a514034eaa73fdff83531a1eef567e826b9098dad1ef23d0eb5dc7a40bc
** Note: Test succeeded for input 86 : Expected: 849a6b08e5be7df60430a765966707649e7e6c51b56992ff4fc4fdecb1ab4924
** Note: Test succeeded for input 87 : Expected: bf92a04e60b4b2362d45490e5142e47b687c2ebb5898da6bc71b697602016e6e
** Note: Test succeeded for input 88 : Expected: fce1d0ddab76d8cc3e28e0de2a9daf92583af1e8f4b1d39314cacffabc5831e7
** Note: Test succeeded for input 89 : Expected: fee6e802336142b0e5183025f9fc8ee4747d1e37593865490ae060c9e7766887
** Note: Test succeeded for input 90 : Expected: 0a33cbfb6158af6710715fe38d2bed12fb4fac3c553f70b589df6de95bdc21f1
** Note: Test succeeded for input 91 : Expected: 0daebf18b62d45891e7e35ef0705e85c01c7de33b73dfc5dfe5155b41b9c29db
** Note: Test succeeded for input 92 : Expected: a047663119665f31a3457fc5f0414a3e5091888eb1b2bda6ee066fc9d0af78c8
** Note: Test succeeded for input 93 : Expected: 77de3f554afe363a72d260e848f600c2b1dd8583a4ff2b1db98a1c8fb2f1597b
** Note: Test succeeded for input 94 : Expected: 4623648d30197a49fc0986e7499a2a288e18fe18beb2ea9d6fd926f566966e5f
** Note: Test succeeded for input 95 : Expected: f63443e43b193e5f437b29c8e54cfd6ab0693c035b843f8e35ae737ad220ecbb
** Note: Test succeeded for input 96 : Expected: 8a9e7bdc551bb2a0ef8bce1a8b2d4ff8e11a6f93d10ec8de51e417b7a541b282
** Note: Test succeeded for input 97 : Expected: 8e7c0686e2853efdc540902a9e22bf1d95a3c830db620e0b0b05c4a91db7b680
** Note: Test succeeded for input 98 : Expected: a047663119665f31a3457fc5f0414a3e5091888eb1b2bda6ee066fc9d0af78c8
** Note: Test succeeded for input 99 : Expected: bdce205bba620f1752641d8ed27f2018bfe17bf3bd25b9f46c2a8c4067acf729

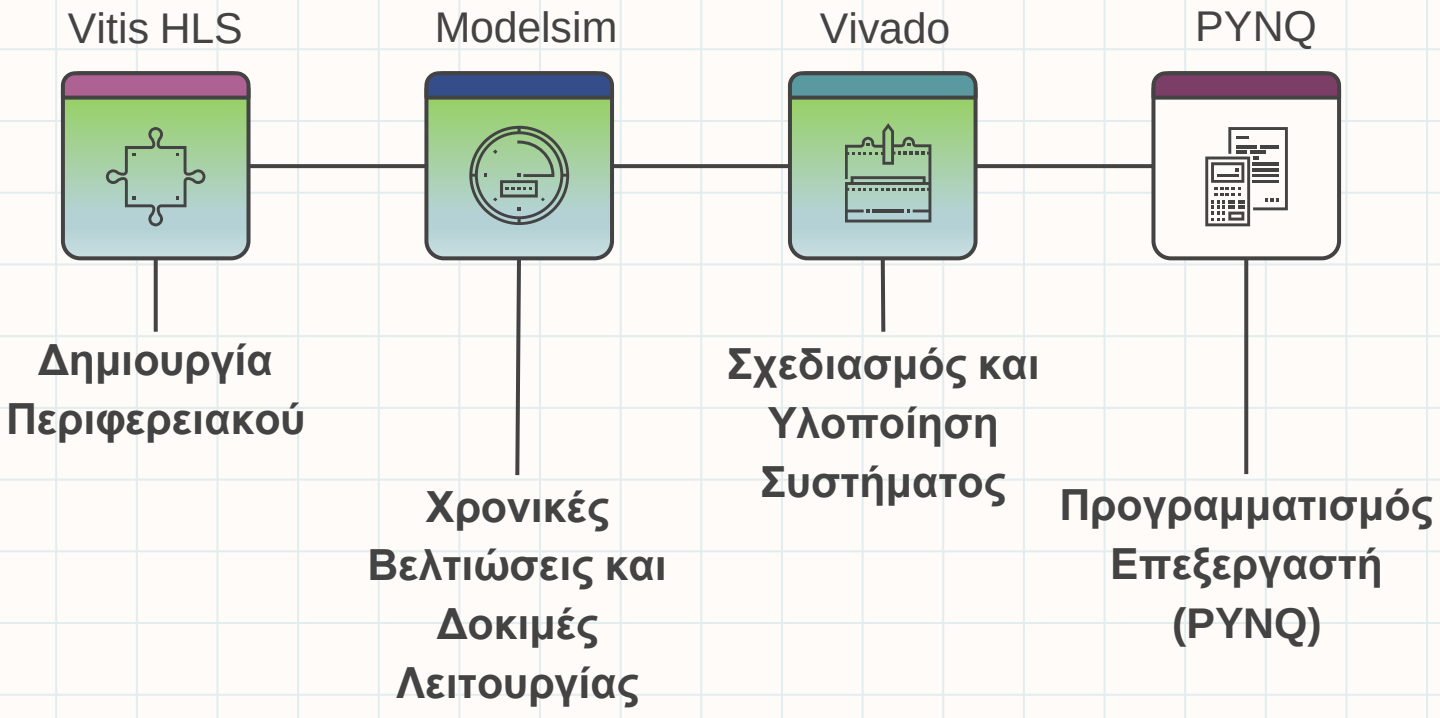
05

Σχεδίαση & Υλοποίηση

- ✓ Μεθοδολογία Σχεδιασμού
- ✓ Δημιουργία Συστήματος
- ✓ Αποτελέσματα Υλοποίησης



Εξέλιξη Συστήματος Κρυπτογράφησης



Θεωρητικός Σχεδιασμός Συστήματος

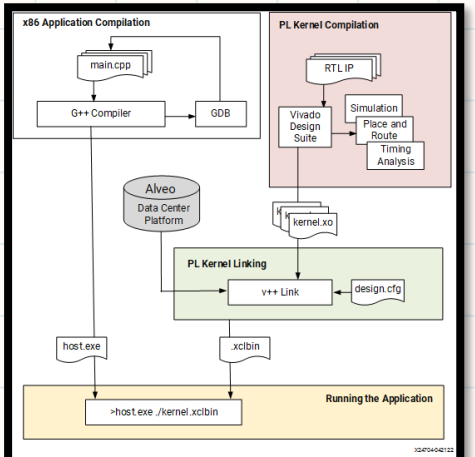
Σύστημα Κρυπτογράφησης

Περιφερειακό

- RTL (Hash)
- VHDL & Verilog

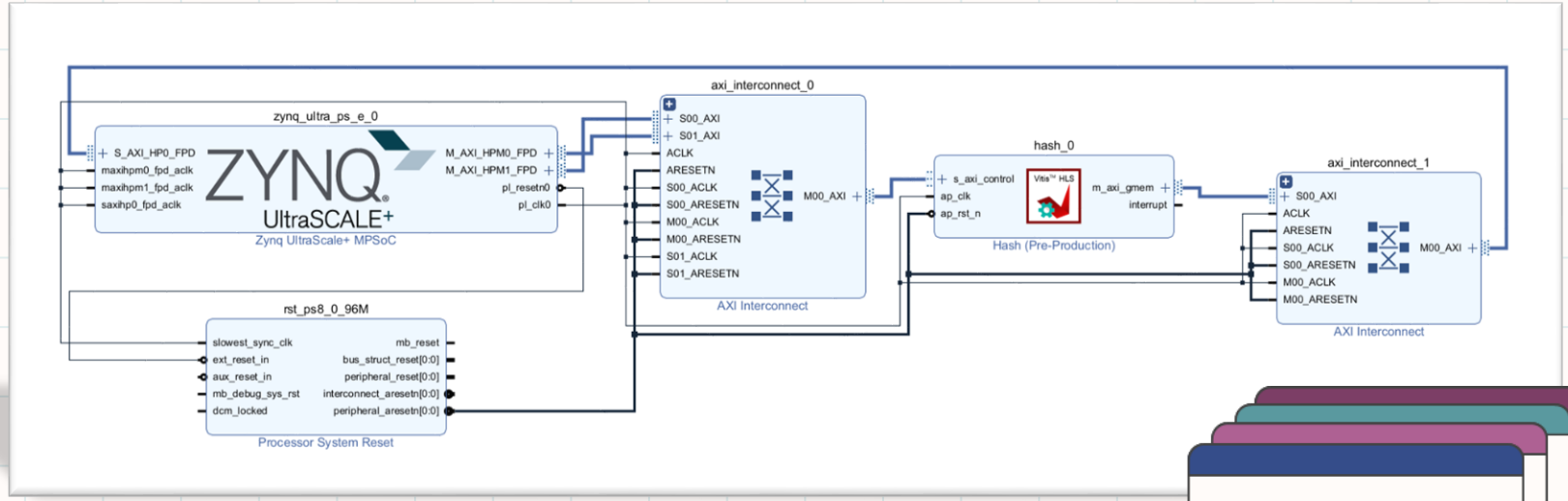
Περιβάλλον Vivado

- Zynq UltraScale+ MPSoC
- AXI Interconnect



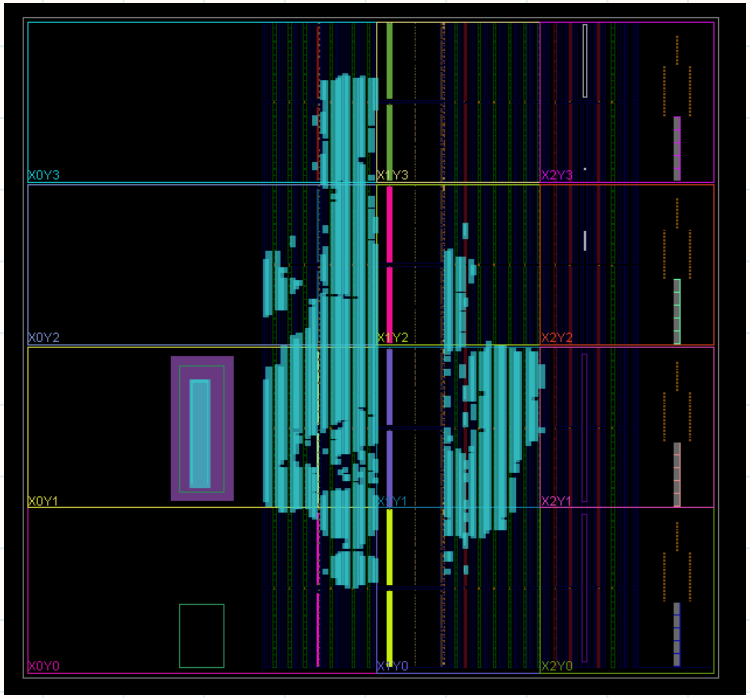
Bitstream (.bit) & IP File (.hwh)

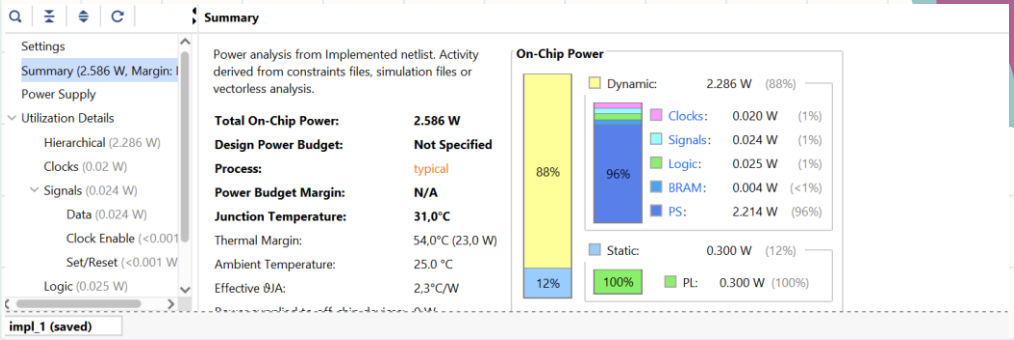
Υλοποίηση Συστήματος Κρυπτογράφησης

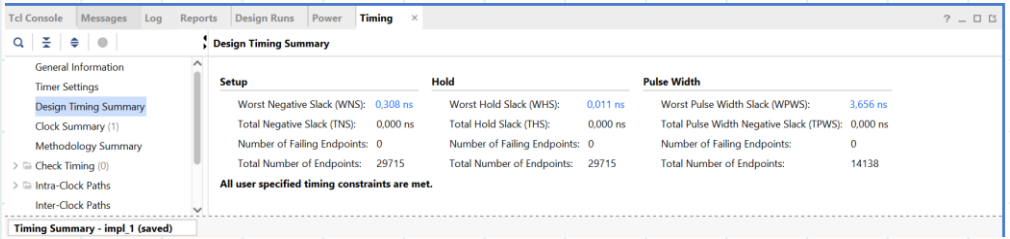


Μια εικόνα είναι χίλιες λέξεις

Στοιχεία του Συστήματος Κρυπτογράφησης

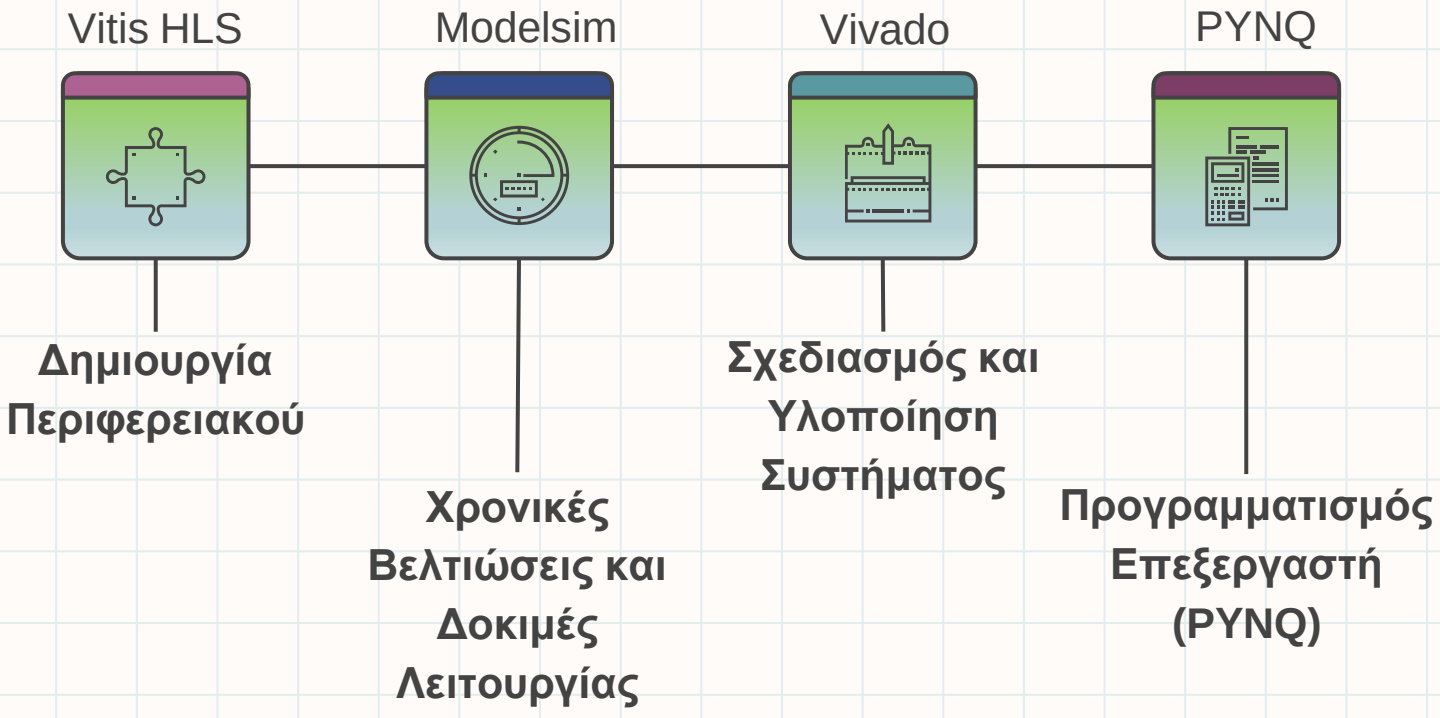






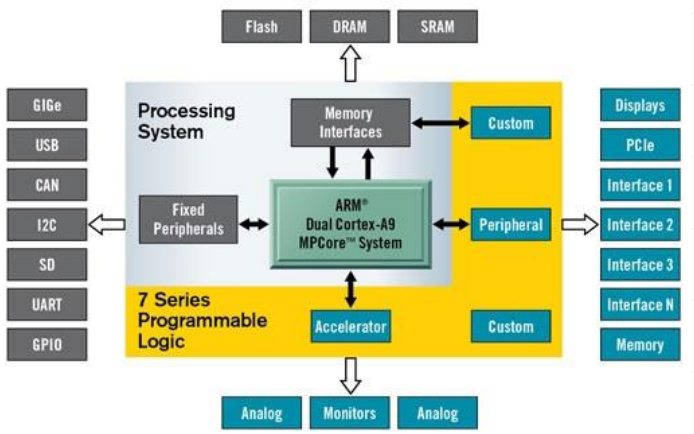
LUT	FF	BRAM	URAM	DSP	Start	Elapsed	Run Strategy
0	**0**	**0**	**0**	**0**	**6/16/24, 2:13 PM**	**00:00:29**	**Vivado Synthesis Defaults (Vivado Synthesis 2023)**
20274	13858	7	0	0	6/16/24, 2:13 PM	00:10:31	Vivado Implementation Defaults (Vivado Implementation 2023)

Εξέλιξη Συστήματος Κρυπτογράφησης



Βιβλιοθήκη PYNQ 1/2

Οι επικαλύψεις (Overlays) ή βιβλιοθήκες υλικού είναι προγραμματιζόμενα/διαμορφώσιμα σχέδια FPGA που επεκτείνουν την εφαρμογή του χρήστη από το σύστημα επεξεργασίας του Zynq στην προγραμματιζόμενη λογική.



- Το PYNQ παρέχει μια διεπαφή Python, η οποία επιτρέπει τον έλεγχο των επικαλύψεων στο PL από την Python που εκτελείται στο PS.

Βιβλιοθήκη PYNQ 2/2

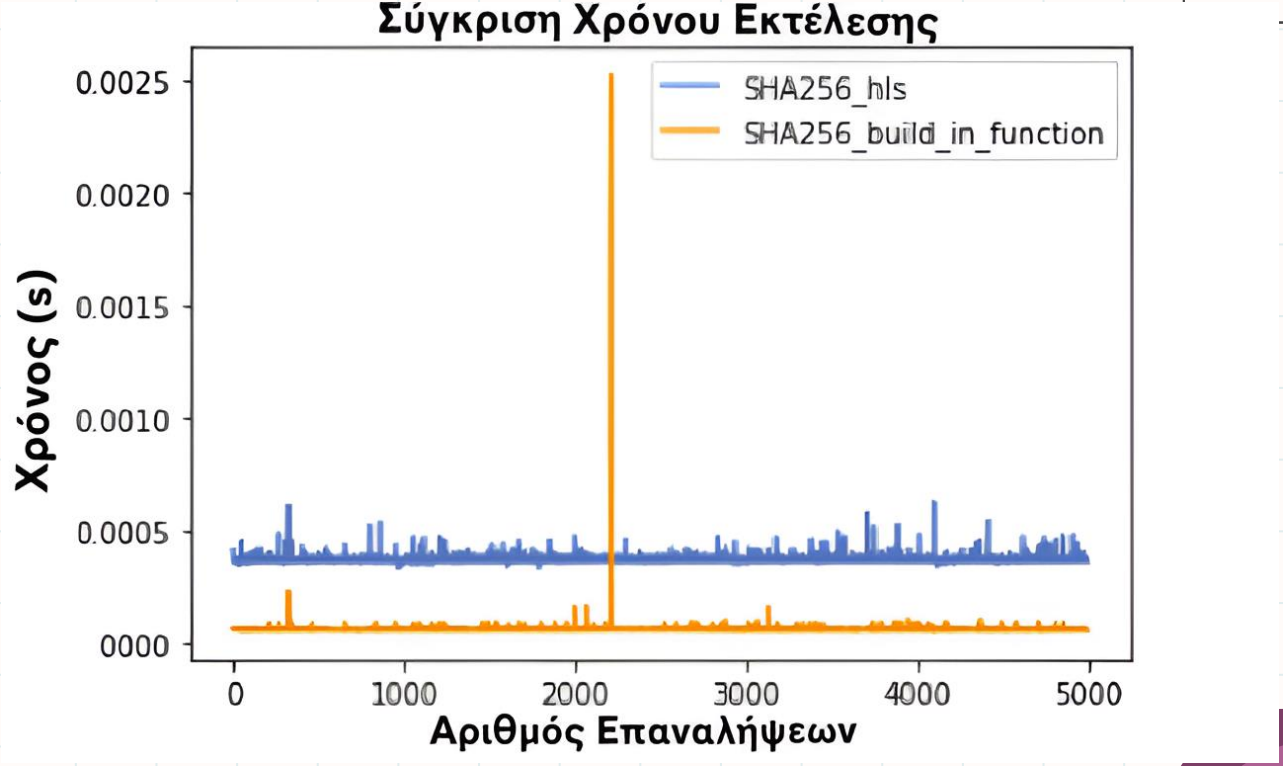
First we must configure the hardaware. We programm the bitstream overlay with the instructions in its page

```
In [38]: from pynq import Overlay, allocate

         # Loading the overlay as described in the doctumentation
         overlay = Overlay('news.bit')
```

Ακολουθεί ζωντανή επίδειξη

Μετρικά Αποτελέσματα



06

Μελλοντικές Βελτιώσεις

Μελλοντικές Βελτιώσεις

Σύστημα

- **Βελτιστοποίηση Πόρων**
- **Αλλαγή Μεταβλητών στο Περιφερειακό της C++**
- **Παράλληλη Επεξεργασία**
- **Επανασχεδιασμός στο Επίπεδο του Vivado**

Βιβλιογραφική Αναφορά

1. A. Sideris, T. Sanida, M. V. Sanida, M. Dossis, and M. Dasygenis, “Accelerate Processing of Image with the Keccak-512 Algorithm on Cryptoprocessor,” in 2023 8th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDACECNSM), Aug. 2023, pp. 1–4. doi: 10.1109/SEEDA-CECNSM61561.2023.10470528.
2. M. Kammoun, M. Elleuchi, M. Abid, and M. S. BenSaleh, “FPGA-based implementation of the SHA-256 hash algorithm,” in 2020 IEEE International Conference on Design & Test of Integrated Micro & Nano-Systems (DTS), Jun. 2020, pp. 1–6. doi: 10.1109/DTS48731.2020.9196134.
3. Ν. Σ. Μεταξάκης, “Σχεδιασμός αλγορίθμου SHA-256 στην γλώσσα VHDL και υλοποίησή του σε FPGA,” Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2022. Accessed: May 28, 2024. [Online]. Available: https://ikee.lib.auth.gr/record/345355
4. R. García, I. Algredo-Badillo, M. Morales-Sandoval, C. Feregrino, and R. Cumplido, “A compact FPGA-based processor for the Secure Hash Algorithm SHA-256,” Comput. Electr. Eng., vol. 40, Jan. 2013, doi: 10.1016/j.compeleceng.2013.11.014.
5. Κ. Μπαρκάτσας, “Υλοποίηση ML-AI σε high-end FPGA,” Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης, 2022. [Online]. Available: https://ikee.lib.auth.gr/record/342735/?ln=el

Νίκου Ραφαήλ

Ευχαριστώ για τον χρόνο σας!

Έχετε ερωτήσεις;

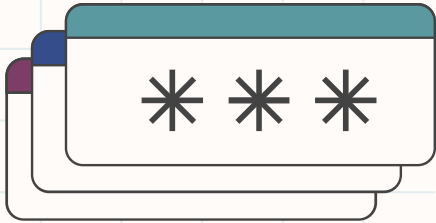
nikourafail@gmail.com | +30 697 556 5773 | nikourafail.com

Diploma Thesis

07

Βοηθητικό Υλικό

Διάγραμμα Χρονοπρογραμματισμού



Εργασία 1
Αναζήτηση και κατασκευή Περιφερειακού

Εργασία 2
Σχεδιασμός και Υλοποίηση Συστήματος

Εργασία 3
Συγγραφή Διπλωματικής Εργασίας

Οκτ2023	Δεκ	Ιαν	Φεβρ	Απρ	Ιουν2024

Εργασία	Περιγραφή	Ημερομηνία	Κατάσταση
Εργασία 1	Αναζήτηση και κατασκευή Περιφερειακού	Οκτωβρίου 1 - Δεκεμβρίου 15	Ολοκληρωμένη
Εργασία 2	Σχεδιασμός και Υλοποίηση Συστήματος	Δεκεμβρίου 15 - Απριλίου 10	Ολοκληρωμένη
Εργασία 3	Συγγραφή Διπλωματικής Εργασίας	Απριλίου 15 - Ιουνίου 25	Ολοκληρωμένη

Σταθερές του αλγορίθμου SHA-256

64 δυαδικές λέξεις W(64) των 32 bit που πηγάζουν από το δεκαδικό μέρος των κυβικών ριζών των πρώτων 64 πρώτων αριθμών.

```
static const uint32_t W[64] = {
        0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,
        0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,
        0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240calcc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,
        0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,
        0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,
        0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,0x106aa070,
        0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,
        0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2
};
```

Συναρτήσεις του αλγορίθμου SHA-256

- ✓ Οι λέξεις αναπαρίστανται σαν x, y και z.
- ✓ Υπάρχουν 2 τριμερείς συναρτήσεις που δέχονται 3 εισόδους και 4 μοναδιαίες.

```
//defining the rotations and the algorithms praxis
#define ROTATIONRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))

#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTATIONRIGHT(x,2) ^ ROTATIONRIGHT(x,13) ^ ROTATIONRIGHT(x,22))
#define EP1(x) (ROTATIONRIGHT(x,6) ^ ROTATIONRIGHT(x,11) ^ ROTATIONRIGHT(x,25))
#define SIG0(x) (ROTATIONRIGHT(x,7) ^ ROTATIONRIGHT(x,18) ^ ((x) >> 3))
#define SIG1(x) (ROTATIONRIGHT(x,17) ^ ROTATIONRIGHT(x,19) ^ ((x) >> 10))
```

ROTATIONRIGHT (a, b): Ορίζει την κυκλική μετατόπιση δεξιά. Μετατοπίζει τα bits του a κατά b θέσεις προς τα δεξιά και τα bits που "περισσεύουν" τα μεταφέρει στην αρχή (αριστερά).

CH (x, y, z): Ορίζει τη λογική πράξη "Choice" (επιλογή). Επιλέγει bits από το y ή το z ανάλογα με τα bits του x.

MAJ (x, y, z): Ορίζει τη λογική πράξη "Majority" (πλειοψηφία). Για κάθε bit, επιστρέφει το bit που εμφανίζεται περισσότερο στα x, y, και z.

EP0 (x): Ορίζει τη συνάρτηση "Sum 0". Κάνει τρεις κυκλικές μετατοπίσεις δεξιά και παίρνει το XOR των αποτελεσμάτων.

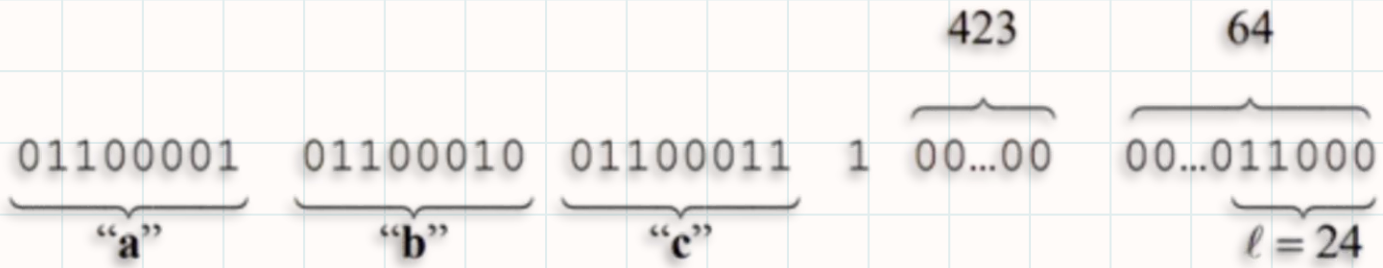
EP1 (x): Ορίζει τη συνάρτηση "Sum 1". Κάνει τρεις κυκλικές μετατοπίσεις δεξιά και παίρνει το XOR των αποτελεσμάτων.

SIG0(x): Ορίζει τη συνάρτηση "Sigma 0". Κάνει δύο κυκλικές μετατοπίσεις δεξιά και μια κανονική μετατόπιση δεξιά, παίρνοντας το XOR των αποτελεσμάτων.

SIG1(x): Ορίζει τη συνάρτηση "Sigma 1". Κάνει δύο κυκλικές μετατοπίσεις δεξιά και μια κανονική μετατόπιση δεξιά, παίρνοντας το XOR των αποτελεσμάτων

Διαδικασία Padding του αλγορίθμου SHA-256

- ✓ Για να διασφαλιστεί ότι το μήνυμα είναι ακέραιο πολλαπλάσιο του 512 bit, το μήνυμα συμπληρώνεται με μια μορφή padding αν χρειαστεί. Συγκεκριμένα, συμπληρώνεται με μηδενικά μέχρι να έχει μήκος 448 bits.



Δρομολόγηση μηνύματος του SHA-256

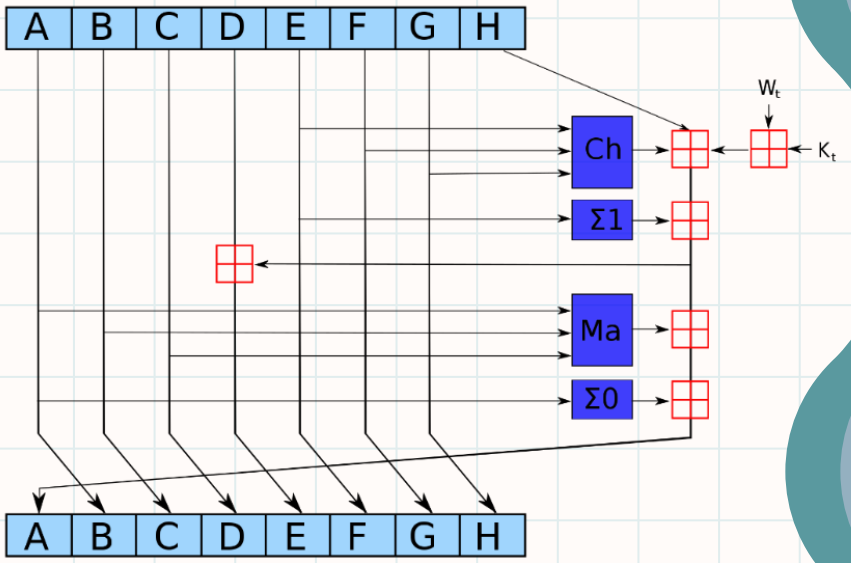
- ✓ Οι πρώτες 16 λέξεις παράγονται διαιρώντας το αρχικό μήνυμα των 512 bit και την καταχώριση αυτών των τιμών σε καταχωρητές 32 bit.
- ✓ Για να παράγουμε τις υπόλοιπες 48 λέξεις, χρησιμοποιούμε μερικές από τις λειτουργίες που παρουσιάσαμε νωρίτερα.

$$W[i] := W[i-16] + s_0 + W[i-7] + s_1$$

Διαδικασία της Συμπίεσης του SHA-256

$for(i\ από\ 0\ εώς\ 63)$
{
$S_1 = (e\ \textbf{rightrotate}\ 6)\ xor\ (e\ \textbf{rightrotate}\ 11)\ xor\ (e\ \textbf{rightrotate}\ 25)$
$ch = (e\ \textbf{and}\ f)\ \textbf{xor}\ ((\textbf{not}\ e)\ \textbf{and}\ g)$
$temp_1 = h + s_1 + ch + m[i] + w[i]$
$s_0 = (a\ \textbf{rightrotate}\ 2)\ \textbf{xor}\ (a\ \textbf{rightrotate}\ 13)xor\ (a\ \textbf{rightrotate}\ 22)$
$maj = (a\ \textbf{and}\ b)\ \textbf{xor}\ (a\ \textbf{and}\ c)\ \textbf{xor}\ (b\ \textbf{and}\ c)$
$temp_2 = s_0 + maj$

$h = g$
$g = f$
$h = g$
$f = e$
$e = d + temp_1$
$d = c$
$c = b$
$b = a$
$a = temp_1 + temp_2$
}



Το κόκκινο σχήμα ⊞ είναι η πρόσθεση modulo 2^{32} για SHA-256, ή 2^{64} για SHA-512

Αρχικοποίηση τιμών Hashing:

(αποτελούν τα πρώτα 32 bit των κλασματικών μερών των τετραγωνικών ριζών των 8 πρώτων αριθμών 2..19):

h0 := 0x6a09e667
h1 := 0xbb67ae85
h2 := 0x3c6ef372
h3 := 0xa54ff53a
h4 := 0x510e527f
h5 := 0x9b05688c
h6 := 0x1f83d9ab
h7 := 0x5be0cd1

Αρχικοποίηση του πίνακα σταθερών:

(αποτελείται από τα πρώτα 32 bits των κλασματικών μερών των κυβικών ριζών των πρώτων 64 πρώτων αριθμών 2..311):

W[0..63] := 0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5, 0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174, 0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da, 0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967, 0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85, 0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070, 0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

Επεξεργασία του μηνύματος σε διαδοχικά τμήματα των 512 bit:

Απαιτείται ο τεμαχισμός του μηνύματος σε κομμάτια των 512 bit
για κάθε τεμάχιο:
{
Δημιουργία ενός πίνακα μηνύματος 64 καταχωρήσεων m[0..63] από λέξεις 32-bit
(Οι αρχικές τιμές στο w[0..63] δεν έχουν σημασία)
Αντιγραφή των τμημάτων στις πρώτες 16 λέξεις m[0..15] του πίνακα μηνυμάτων

Επέκταση των πρώτων 16 λέξεων στις υπόλοιπες 48 λέξεις m[16..63] του πίνακα μηνυμάτων:
for i από 16 έως 63
{
(Εκτέλεση των παρακάτω πράξεων)
s0 := (m[i-15] rightrotate 7) xor (m [i-15] rightrotate 18) xor (m [i-15] rightshift 3)
s1 := (m[i-2] rightrotate 17) xor (m [i-2] rightrotate 19) xor (m [i-2] rightshift 10)
m[i] := m[i-16] + s0 + m[i-7] + s1
}
Αρχικοποίηση των μεταβλητών εργασίας a έως h στην παρούσα τιμή hashing:
a := h0
b := h1
c := h2
d := h3
e := h4
f := h5
g := h6
h := h7

Προεπεξεργασία (Padding):

i. *Ξεκίνα με το αρχικό μήνυμα μήκους L bits.*
ii. *Προσθήκη ενός μόνο bit «1».*
iii. *Προσθήκη K bit «0», όπου K είναι ο ελάχιστος αριθμός >= 0 τέτοιο ώστε (L + 1 + K + 64) να είναι πολλαπλάσιο του 512.*
iv. *Προσάρτηση του L ως ακέραιου big-endian αριθμού 64 bit, καθιστώντας το συνολικό μέγεθος του επεξεργασμένου μηνύματος να έχει μήκος πολλαπλάσιο των 512 bit.*
v. ***Right Rotate (κυκλική δεξιά μετατόπιση)**: Μετακινεί τα bits μιας λέξης προς τα δεξιά, με τα bits που "φεύγουν" από τη μία άκρη να επανεισάγονται στην άλλη άκρη.*
vi. ***Right Shift (δεξιά μετατόπιση)**: Μετακινεί τα bits μιας λέξης προς τα δεξιά, εισάγοντας μηδενικά (ή το πρόσημο για signed αριθμούς) στα πιο αριστερά bits που κενώνονται.*

Κύριος βρόχος λειτουργίας συμπίεσης (βασική πράξη):
for i από 0 έως 63
{
(Εκτέλεση των παρακάτω πράξεων και μετατοπισμός των τιμών προς τα κάτω)
S1 := (e rightrotate 6) xor (e rightrotate 11) xor (e rightrotate 25)
ch := (e and f) xor ((not e) και g)
temp1 := h + S1 + ch + m[i] + W[i]
S0 := (a rightrotate 2) xor (a rightrotate 13) xor (a rightrotate 22)
maj := (a and b) xor (a και c) xor (b and c)
temp2 := S0 + maj

h := g
g := f
f := e
e := d + temp1
d := c
c := b
b := a
a := temp1 + temp2
}
Προσθήκη του συμπιεσμένου κομματιού στην παρούσα τιμή hashing:
h0 := h0 + a
h1 := h1 + b
h2 := h2 + c
h3 := h3 + d
h4 := h4 + e
h5 := h5 + f
h6 := h6 + g
h7 := h7 + h
}
Παραγωγή της τελικής τιμής κατακερματισμού (big-endian):
digest := hash := h0 append h1 append h2 append h3 append h4 append h5 append h6 append h7

Kria KR260 Robotics Starter Kit