ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία

Έλεγχος Ασφάλειας Πτήσης σε UAV με FPGA

Συγγραφέας

ΚΑΡΝΑΒΑΣ ΑΠΟΣΤΟΛΟΣ

ΑΜ: 711161050

Επιβλέπων : ΙΩΑΝΝΗΣ ΒΟΓΙΑΤΖΗΣ

Αιγάλεω, Ιούλιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Έλεγχος Ασφάλειας Πτήσης σε UAV με FPGA

Μέλη Εξεταστικής Επιτροπής συμπεριλαμβανομένου και του Εισηγητή

Η πτυχιακή/διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

Α/α	ΟΝΟΜΑ ΕΠΩΝΥΜΟ	ΒΑΘΜΙΔΑ/ΙΔΙΟΤΗΤΑ	ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ
	Ioannis Voyiatzis		Digitally signed by Ioannis Voyiatzis Date: 2022.09.05 12:44:59 +03'00'
	Stavros Fatouros		Digitally signed by Stavros Fatouros Date: 2022.09.06 17:18:10 +03'00'
	Panagiotis Giannakopoulos		Digitally signed by Panagiotis Giannakopoulos Date: 2022.09.16 08:50:34 +03'00'

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΤΥΧΙΑΚΗΣ/ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

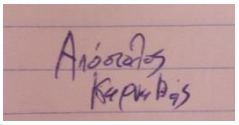
Ο κάτωθι υπογεγραμμένος **Απόστολος Καρναβάς** του **Ευσταθίου**, με αριθμό μητρώου **711161050** φοιτητής/τρια του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών , δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της πτυχιακής/διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών
Καρναβάς Απόστολος

*** Ονοματεπώνυμο /Ιδιότητα** **(Υπογραφή)**



Ψηφιακή Υπογραφή Επιβλέποντα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά των επιβλέποντα καθηγητή μου για την βοήθεια που μου έδωσε καθώς και τους κυρίους Δημήτρη Ψιλιά και Αθανάσιο Μηλιδώνη για την καθοδήγηση και βοήθεια που μου πρόσφεραν.

Περίληψη

Στην παρούσα διπλωματική εργασία ερευνάται η δημιουργία ενός συστήματος κρυπτογράφησης δεδομένων σε μία πλακέτα ανάπτυξης FPGA για εφαρμογές UAV. Στην αρχή γίνεται μια εισαγωγή στα μη επανδρωμένα ιπτάμενα οχήματα (UAV's) καθώς και αναφορά στην ιστορική τους πορεία. Στην συνέχεια γίνονται αναφορές στα FPGA για την αρχιτεκτονική τους και την ιστορία τους καθώς και παρουσιάζεται η πλακέτα ανάπτυξης FPGA που επιλέχθηκε για την παρούσα εργασία. Έπειτα αναφέρεται ο αλγόριθμος κρυπτογράφησης που επιλέχθηκε για την υλοποίηση του συστήματος, ο AES 128 σε CBC τρόπο λειτουργίας. Τέλος παρουσιάζεται η αρχιτεκτονική του συστήματος, η ανάλυση των εξαρτημάτων που το απαρτίζουν, το λογισμικό που επιλέχθηκε για την κατασκευή του καθώς και τα αποτελέσματα που λήφθηκαν με το πέρας της δημιουργίας του συστήματος.

Abstract

In this thesis, the creation of a data encryption system on an FPGA development board for UAV applications is researched. At the beginning there is an introduction to unmanned aerial vehicles (UAV's) as well as a reference to their historical course. References are then made to FPGAs for their architecture and history, and the FPGA development board chosen for this work is presented. Then the encryption algorithm chosen to implement the system, the AES 128 in CBC mode, is mentioned. Finally, the architecture of the system, the analysis of the components that make it up, the software chosen for its construction as well as the results obtained after the creation of the system are presented and discussed.

Περιεχόμενα

1. Εισαγωγή

Με την εξέλιξη της τεχνολογίας και ιδιαίτερα στον τομέα της αεροηλεκτρονικής τα drones έχουν αρχίσει να γίνονται ένα κομμάτι της καθημερινότητάς μας. Έχοντας χρησιμοποιηθεί αρχικά στον στρατιωτικό τομέα τώρα πια αξιοποιούνται στους τομείς της γεωπονίας, ασφάλειας, εμπορίου καθώς και ψυχαγωγίας για λήψη βίντεο και φωτογραφιών. Η αυξανόμενη χρήση τους συνεπώς δημιουργεί την ανάγκη για ασφάλισή τους καθώς τα UAV's είναι επιρρεπή σε μια πληθώρα επιθέσεων καθώς και μπορεί να χρησιμοποιηθούν για την εκτέλεση κακόβουλων ενεργειών όπως για παράδειγμα[14] :

- **GPS spoofing:** όπου οι επιτιθέμενοι τροφοδοτούν drones με ψεύτικες συντεταγμένες GPS και αναλαμβάνουν τον πλήρη έλεγχο της πλατφόρμας κάτι το οποίο θυμίζει τις επιθέσεις από botnets(κακόβουλοι υπολογιστές) μέσο ddos (distributed denial of service) στα δίκτυα του IoT(internet of things).
- **Downlink intercept:** Επιτρέπει στον επιτιθέμενο να έχει πρόσβαση σε όλα τα δεδομένα μεταξύ του drone και του ελεγκτή. Δεδομένου ότι η πλειονότητα των εμπορικών συστημάτων drones αλληλοεπιδρούν με τη βάση τους χρησιμοποιώντας μη κρυπτογραφημένα κανάλια επικοινωνίας, είναι ευάλωτα σε υποκλοπές.
- **Data exploitation:** Η χρήση drones μπορεί να ξεπεράσει τους φυσικούς περιορισμούς ασφάλειας και την ασφάλεια στον κυβερνοχώρο, καθώς ένα drone με έναν ενσωματωμένο μικροϋπολογιστή μπορεί να αξιοποιηθεί για να εκτελέσει κακόβουλες ενέργειες όπως υποκλοπή δεδομένων wi-fi , network hijacking κλπ.

Συνεπώς προκειμένου να αποφευχθούν τα παραπάνω θα πρέπει να ενσωματωθούν στα UAV's τα κατάλληλα συστήματα ασφαλείας, τόσο στο κομμάτι της μετάδοσης δεδομένων και βίντεο, όσο και στην φυσική πρόσβαση.

1.1 Αντικείμενο διπλωματικής εργασίας.

Σκοπός της παρούσας διπλωματικής εργασίας με τίτλο ''Έλεγχος ασφάλειας πτήσης σε UAV με FPGA" είναι να δημιουργηθεί ένα σύστημα κρυπτογράφησης δεδομένων ελέγχου, τηλεμετρίας και βίντεο προκειμένου να καλυφθεί η ανάγκη για ασφάλεια στο κομμάτι της μετάδοσης ευαίσθητων δεδομένων μεταξύ του σταθμού βάσης και του UAV. Ο αλγόριθμος κρυπτογράφησης και αποκρυπτογράφησης που επιλέχθηκε είναι ο AES (Advanced Encryption Standard) και η πλατφόρμα πάνω στην οποία υλοποιήθηκε το σύστημα αυτό είναι το Nexys Video που περιέχει το Xilinx Artix-7 FPGA (Field Programmable Gate Array). Εκτός από το σύστημα κρυπτογράφησης/αποκρυπτογράφησης έχουν υλοποιηθεί και άλλα εξαρτήματα προκειμένου να μπορεί το FPGA να επικοινωνήσει με τον ελεγκτή πτήσης του UAV καθώς και με τα εξαρτήματα επικοινωνίας. Στην παρούσα διπλωματική εργασία επιλέχθηκε να χρησιμοποιηθεί ο αλγόριθμος AES σε CBC μορφή προκειμένου να επιτευχθεί η αξιόπιστη κρυπτογράφηση των δεδομένων και κυρίως των δεδομένων βίντεο.

2. Σχετική δουλειά

2.1 UAV's

2.1.1 Ιστορική αναδρομή

Η παλαιότερη καταγεγραμμένη χρήση ενός μη επανδρωμένου εναέριου οχήματος ήταν για την χρήση του σε πολεμικές διενέξεις. Τον Ιούλιο του 1849 ο αυστριακός στρατός[23] ενώ πολιορκούσε την Βενετία επιχείρησε να χρησιμοποιήσει περίπου 200 εμπρηστικά μπαλόνια τα οποία και εκτόξευσε κυρίως από την στεριά καθώς και από το σκάφος SMS Vulcano . Παρόλο που τουλάχιστον ένα κατάφερε να πέσει στην πόλη λόγω της αλλαγής του ανέμου μετά την εκτόξευση η πλειοψηφία τους δεν κατάφεραν να πέσουν στον στόχο τους ενώ άλλα παρασύρθηκαν πίσω στις αυστριακές δυνάμεις και στο πλοίο SMS Vulcano. Αυτή αποτέλεσε μία από τις πρώτες απόπειρες χρήσης τους για πολεμικούς σκοπούς και θα περνούσαν πολλά χρόνια μέχρι να γίνουν αποτελεσματικά για στρατιωτική χρήση[21][22].

Μία από τις σημαντικές εξελίξεις στον τρόπο χειρισμού τους έγινε από τον Ισπανό μηχανικό Leonardo Torres y Quevedo όταν το 1903 στην Ακαδημία Επιστημών του Παρισιού εισήγαγε ένα σύστημα ελέγχου, με όνομα "Telekino", που χρησιμοποιούσε σαν βάση ραδιοκύματα προκειμένου να χειριστεί ένα αεροσκάφος δικού του σχεδιασμού χωρίς να διακινδυνεύσει ανθρώπινες ζωές.[21][22]

Στις αρχές του 20ου αιώνα τα drones ξεκίνησαν να έχουν σημαντική ανάπτυξη με πρώτο στόχο την παροχή πρακτικής εκπαίδευσης στρατιωτικού προσωπικού. Την πρώτη απόπειρα για μηχανοκίνητο UAV αποτέλεσε το "Aerial Target"

του A. M. Low το 1916. Ο Low επιβεβαίωσε ότι το μονοπλάνο του Geoffrey de Havilland ήταν αυτό που πέταξε υπό έλεγχο στις 21 Μαρτίου 1917 χρησιμοποιώντας το ραδιοφωνικό του σύστημα. Μετά από αυτή την επιτυχημένη επίδειξη την άνοιξη του 1917, ο Low μεταφέρθηκε για να αναπτύξει ελεγχόμενες με αεροσκάφη γρήγορες εκτοξεύσεις κινητήρα D.C.B. με το Βασιλικό Ναυτικό το 1918 με σκοπό να επιτεθεί σε ναυτιλιακές και λιμενικές εγκαταστάσεις. Ακολούθησαν και άλλες βρετανικές μη επανδρωμένες εξελίξεις, οι οποίες οδήγησαν στο στόλο με περισσότερους από 400 εναέριους στόχους "de Havilland 82 Queen Bee" που τέθηκαν σε λειτουργία το 1935. Ο Νίκολα Τέσλα, περιέγραψε το 1915 ένα στόλο από μη επανδρωμένα εναέρια οχήματα μάχης ενώ παράλληλα οι εξελίξεις που είδαν τα UAV's ενέπνευσαν την κατασκευή του "Kettering Bug" από τον Charles Kettering από το Dayton του Οχάιο και το "Hewitt-Sperry Automatic Airplane". Η ανάπτυξη συνεχίστηκε κατά τη διάρκεια του Πρώτου Παγκοσμίου Πολέμου, όταν η εταιρεία αεροπλάνων Dayton-Wright εφηύρε μια εναέρια τορπίλη χωρίς πιλότο που θα εκραγεί σε προκαθορισμένο χρόνο. Κατά την διάρκεια του Β' υπήρξε ιδιαίτερη ανάπτυξη στα UAV's όταν ο αστέρας του Χόλιγουντ και ενθουσιαστής τον αεροπλάνων Ρέτζιναλντ Ντένι κατέληξε στο συμπέρασμα ότι θα υπάρξει μεγάλη ζήτηση ραδιοελεγχόμενων αεροσκαφών χαμηλού κόστους για την εκπαίδευση οπλιτών αντιαεροπορικών όπλων με αποτέλεσμα το 1935 να παρουσιάσει στον στρατό των ΗΠΑ το πρωτότυπο ενός drone στόχου, εν ονόματι RP-1. Ύστερα ο Ντένι αγόρασε από τον Γουόλτερ Ράιτερ το 1938 ένα σχέδιο για UAV και άρχισε να το εμπορεύεται στους χομπίστες ως "Dennymite" το οποίο κιόλας και παρουσίασε στον στρατό των ΗΠΑ ως RP-2 και μετά από τροποποιήσεις ως RP-3 και RP-4 το 1939. Το 1940, ο Ντένι και οι συνεργάτες του κέρδισαν ένα συμβόλαιο στρατού για το ραδιοελεγχόμενο RP-4 τους, το οποίο έγινε το Radioplane OQ-2. Κατασκεύασαν σχεδόν δεκαπέντε χιλιάδες drones για τον στρατό κατά τη διάρκεια του Β' Παγκοσμίου Πολέμου[18][21][22].

Με το πέρας του Β' παγκοσμίου πολέμου και του ψυχρού πολέμου τα UAV's συνέχισαν να εξελίσσονται. Πέρα από τον στρατιωτικό τομέα είδαν και μία ραγδαία εξέλιξη όσον αφορά την χρήση τους από τον μέσο πολίτη καθώς το 2006[23] το FAA(Federal Aviation Administration) εξέδωσε την πρώτη εμπορική άδεια για χρήση drones τόσο για καταναλωτές όσο και για επαγγελματίες με αποτέλεσμα τα drones να γίνουν ευρέως διαθέσιμα.

Τώρα πια τα drones χρησιμοποιούνται ευρέως και σε άλλες εφαρμογές πέρα από την στρατιωτική χρήση[20][24][25] όπως:

- Διασωστικά έργα
- Επιτήρηση
- Αστυνόμευση
- Παρακολούθηση καιρού
- Πυρόσβεση
- Προσωπική χρήση
- Φωτογράφηση με drone
- Υπηρεσίες Υγείας
- Γεωργία
- Υπηρεσίες παράδοσης εμπορευμάτων

Η επικοινωνία του drone με τον σταθμό βάση ή με το χειριστήριο του χρήστη γίνεται ασύρματα μέσω ραδιοσυχνοτήτων, Wi-Fi η μέσω GPS συντεταγμένων. Οι πιο συνηθισμένες συχνότητες επικοινωνίας είναι στα 900MHz έως 5.8GHz για ραδιοσυχνότητες και 2.5GHz έως 5.8GHz για Wi-Fi. Για μεγάλες αποστάσεις χρησιμοποιούνται ραδιοσυχνότητες συνήθως καθώς λειτουργούν σε χαμηλότερη συχνότητα και μπορούν να πετύχουν επικοινωνία σε μεγαλύτερες αποστάσεις, το Wi-Fi χρησιμοποιείται συνήθως για μικρότερες αποστάσεις και για την μετάδοση βίντεο[1][2][4].

2.1.2 Τηλεμετρία

Σαν Τηλεμετρία[5] ορίζεται η συλλογή με επιτόπιο τρόπο μετρήσεων ή άλλων δεδομένων σε απομακρυσμένα σημεία/συστήματα και η αυτόματη μετάδοσή τους σε εξοπλισμό λήψης (στην περίπτωση των drones ο εξοπλισμός αυτός είναι συνήθως το χειριστήριο του χρήστη ή ο σταθμός βάσης) για παρακολούθηση. Στα drones τα δεδομένα που μεταδίδονται από ένα drone είναι συνήθως πληροφορίες όπως οι συντεταγμένες GPS του οχήματος, η ταχύτητά του, το υψόμετρο που βρίσκεται, η θερμοκρασία του, η απόστασή του από τον χειριστή, καθώς και άλλα δεδομένα όπως η

διαθέσιμη μπαταρία του drone και δεδομένα βίντεο. Συνήθως για την μετάδοση δεδομένων τηλεμετρίας χρησιμοποιείται διαφορετικό κανάλι από αυτό για την μετάδοση εντολών από τον χειριστή προκειμένου να υπάρχει μεγαλύτερη ασφάλεια[4].

2.1.3 Μετάδοση Βίντεο

Η μετάδοση του βίντεο μπορεί να γίνει με μια πληθώρα τρόπων. Οι ποιο συνήθεις είναι [17]:

- **WiFi** : Η μετάδοση δεδομένων βίντεο μέσο WiFi συνήθως χρησιμοποιείται για μικρές αποστάσεις καθώς η απόσταση επικοινωνίας του WiFi βρίσκετε μεταξύ των 300 και 2000 μέτρων. Για την μετάδοση χρησιμοποιείται συνήθως η συχνότητα των 5 GHz.

- **Ζώνη υπο 1 GHz** : Η μετάδοση σε αυτήν την ζώνη χρησιμοποιείται συνήθως για drones που μεταδίδουν βίντεο σε FPV(First Person View). Σε αυτήν χρησιμοποιούνται απλές αναλογικές κάμερες και η συχνότητα μετάδοσης των 900 MHz.

- **3G/4G** :Μπορεί κανείς να χρησιμοποιήσει τα dongles 3G/4G που είναι συνδεδεμένα στο drone, για ασύρματη μετάδοση υψηλού ρυθμού δεδομένων. Αυτή η λύση μπορεί να χρησιμοποιηθεί σύμφωνα με τη διαθεσιμότητα του δικτύου 3G / 4G σε αυτήν την περιοχή λειτουργίας.

- **Προσαρμοσμένη Λύση** : Πέραν των έτοιμων λύσεων , είναι εφικτό να επιλεχθεί μία “custom” λύση για την μετάδοση βίντεο. Για παράδειγμα μπορεί κάποιος να χρησιμοποιήσει ένα FPGA για την λήψη και την μετάδοση των δεδομένων βίντεο που προέρχονται είτε από το drone είτε από την κάμερα κατευθείαν. Μια τέτοια λύση γίνεται θεμιτή όταν οι παραπάνω επιλογές δεν αποτελούν τον βέλτιστο τρόπο μετάδοσης δεδομένων βίντεο.

2.1.4 Ασφάλεια δεδομένων βίντεο και τηλεμετρίας

Η ασφάλιση των δεδομένων εντολών και τηλεμετρίας είναι ένα σημαντικό κομμάτι της λειτουργίας ενός drone. Συνήθως δίνεται έμφαση στην ασφάλεια όσον άφορα drones που χρησιμοποιούνται για στρατιωτική χρήση, ενώ πολλά από τα drones που χρησιμοποιούνται για ψυχαγωγικούς σκοπούς καθώς και εκείνα που χρησιμοποιούνται για επιχειρησιακούς λόγους (παρακολούθηση καιρού, καλλιέργειες κλπ.) έχουν ελλιπή έως και καθόλου ασφάλεια με αποτέλεσμα να είναι επιρρεπή σε μία πληθώρα επιθέσεων. Ένας από τους λόγους που συμβαίνει αυτό είναι ότι είναι δύσκολο πολλές φορές να εφαρμοστεί αποτελεσματική ασφάλεια των δεδομένων τους, χωρίς να επηρεαστεί η λειτουργία τους, ειδικά στα drones που χρησιμοποιούνται για ψυχαγωγικούς σκοπούς, καθώς το κόστος τους πρέπει να είναι χαμηλό προκειμένου να είναι προσβάσιμα από το ευρύ κοινό. Για αυτόν τον λόγο στην παρούσα διπλωματική εργασία παρουσιάζεται η ανάπτυξη ενός συστήματος που καλύπτει τουλάχιστον την ανάγκη για την ασφάλεια των δεδομένων τηλεμετρίας και βίντεο χρησιμοποιώντας κρυπτογράφηση[3].

2.2 FPGA’s

2.2.1 Περίληψη

Τα FPGA[7] είναι ολοκληρωμένα κυκλώματα τα οποία είναι σχεδιασμένα για να έχουν την δυνατότητα να επαναπρογραμματίζονται μετά την κατασκευή τους. Για τον προγραμματισμό τους χρησιμοποιούνται γλώσσες περιγραφής υλικού(όπως η VHDL και η Verilog) παρόμοιες με αυτές που χρησιμοποιούνταν για τον σχεδιασμό ενός ολοκληρωμένου κυκλώματος για συγκεκριμένες εφαρμογές (ASIC).Παλαιότερα για τον προγραμματισμό τους χρησιμοποιούνταν διαγράμματα κυκλωμάτων αλλά με την εξέλιξη των εργαλείων αυτοματισμού ηλεκτρονικού σχεδίου η χρήση τους έχει αρχίσει να εκλείπει.
Στο εσωτερικό τους τα FPGA περιέχουν μια σειρά από προγραμματιζόμενα λογικά μπλοκ καθώς και μία ιεραρχία από επαναδιαμορφώσιμες διασυνδέσεις που επιτρέπουν στα μπλοκ να ενωθούν μεταξύ τους. Τα λογικά μπλοκ που απαρτίζουν το εσωτερικό του FPGA μπορούν να διαμορφωθούν κατάλληλα προκειμένου να εκτελέσουν σύνθετες συνδυαστικές λειτουργίες ή για να αναπαραστήσουν λογικές πύλες όπως είναι η AND ,η OR και η XOR. Επίσης σε πολλά FPGA τα λογικά μπλοκ περιλαμβάνουν και στοιχεία μνήμης , από απλά flip-flop έως και σύνθετες SRAM και DRAM μνήμες.

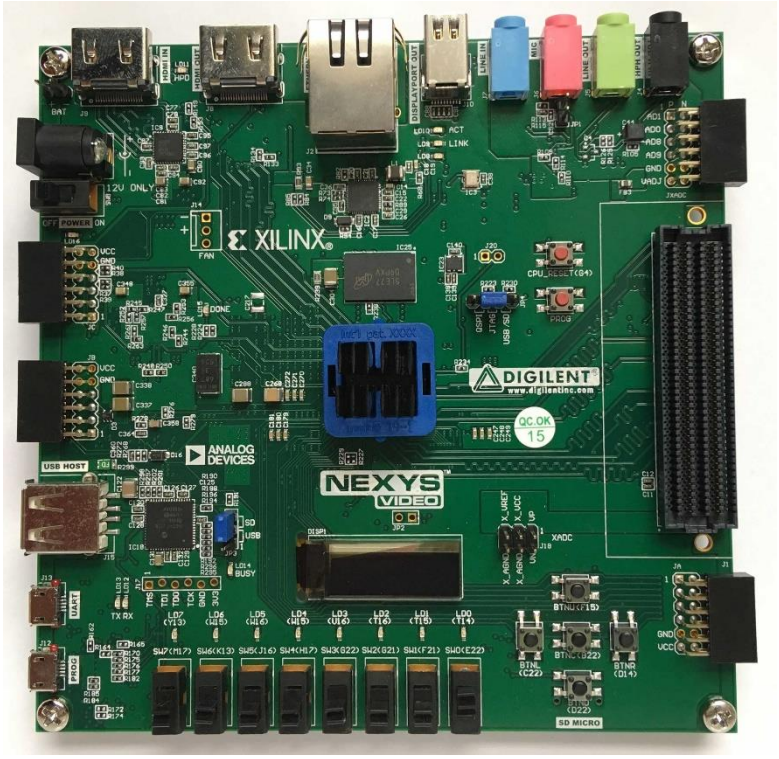
2.2.2 Ιστορική αναδρομή

Ο πρόδρομος της βιομηχανίας των FPGA ήταν οι μνήμες PROM(Programmable Read-Only Memory) και τα PLD(Programmable Logic Device). Οι συσκευές αυτές είχαν την δυνατότητα να προγραμματιστούν σε παρτίδες κατά

την παραγωγή τους στο εργοστάσιο ή κατά την εφαρμογή τους στο πεδίο χρήσης. Όμως το κομμάτι της προγραμματιζόμενης λογικής ήταν σκληρά συνδεδεμένο με τις λογικές πύλες που τα αποτελούσαν. Το 1984 η εταιρία Altera σχεδίασε την πρώτη επαναπρογραμματιζόμενη λογική συσκευή, την EP300. Κύριο χαρακτηριστικό της αποτέλεσε ένα μικρό "παράθυρο" χαλαζία το οποίο παρείχε την δυνατότητα στους χρήστες να του εφαρμόσουν υπεριώδες φως προκειμένου να "αδειάσουν" τις EPROM που περιείχαν τα δεδομένα διαμόρφωσης της συσκευής με αποτέλεσμα να μπορέσουν οι χρήστες να την προγραμματίσουν ξανά ανάλογα με τις απαιτήσεις τους.
Η επόμενη σημαντική εξέλιξη ήρθε το 1985 όταν οι συνιδρυτές της Xilinx, Ross Freeman και Bernard Vonderschmitt κατάφεραν να δημιουργήσουν την πρώτη εμπορικά βιώσιμη συστοιχία προγραμματιζόμενων στο πεδίο χρήσης πυλών με όνομα XC2064[7].
Το 1987, το Naval Surface Warfare Center χρηματοδότησε ένα πείραμα που πρότεινε ο Steve Casselman για την ανάπτυξη ενός υπολογιστή που θα εφαρμόσει 600.000 επαναπρογραμματιζόμενες πύλες. Ο Casselman ήταν επιτυχής και ένα δίπλωμα ευρεσιτεχνίας που σχετίζεται με το σύστημα εκδόθηκε το 1992.Έτσι στα μέσα της δεκαετίας του 1990 τα FPGA ξεκίνησαν να αποτελούν ένα επαναστατικό νέο τεχνολογικό εύρημα με μία πληθώρα εφαρμογών από τους τομείς των τηλεπικοινωνιών έως την αυτοκινητοβιομηχανία και τις συσκευές καθημερινής χρήσης[7][8].

Τα σύγχρονα FPGA διαθέτουν μεγάλους πόρους λογικών πυλών και μπλοκ RAM για την υλοποίηση πολύπλοκων ψηφιακών υπολογισμών. Καθώς τα σχέδια FPGA χρησιμοποιούν πολύ γρήγορους ρυθμούς εισόδου/εξόδου και αμφίδρομους διαύλους δεδομένων, γίνεται πρόκληση η επαλήθευση του σωστού χρονισμού των έγκυρων δεδομένων εντός του χρόνου εγκατάστασης και του χρόνου αναμονής.

Για την διπλωματική αυτή εργασία το FPGA που χρησιμοποιήθηκε είναι το Nexys Video[9][10].



Εικόνα 2.2.2.1 Nexys Video FPGA Board

Τα κύρια χαρακτηριστικά του είναι :

- Xilinx Artix-7 FPGA (XC7A200T-1SBG484C)
- 33.650 logic slices, το καθένα με τέσσερα LUT 6 εισόδων και 8 flip-flops
- 13 Mbit γρήγορης μπλοκ μνήμης RAM (3 φορές μεγαλύτερη από το Nexys 4 DDR)
- 10 πλακίδια διαχείρισης ρολογιού, το καθένα με βρόχο κλειδώματος φάσης (PLL)
- 740 τμήματα DSP
- Εσωτερικές ταχύτητες ρολογιού άνω των 450 MHz
- Μετατροπέας αναλογικού σε ψηφιακό σε τσιπ (XADC)
- Πομποδέκτες GTP έως 3,75 Gbps
- Προγραμματίζεται μέσω JTAG, Quad-SPI Flash, microSD και μονάδα flash USB
- 512MB 800MHz DDR3
- 32MB φλας Quad-SPI

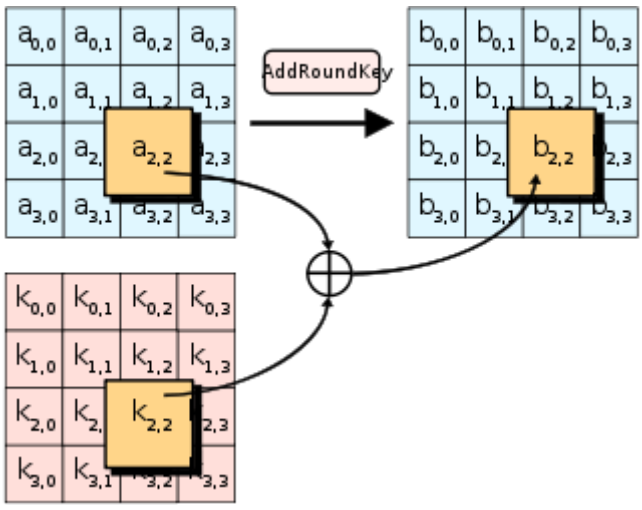
- Υποδοχή κάρτας microSD για μη πτητική αποθήκευση
- Αποκλειστική ενσωματωμένη θύρα USB για προγραμματισμό JTAG και μεταφορά δεδομένων
- Τροφοδοτείται από οποιαδήποτε πηγή 12V (περιλαμβάνεται τροφοδοσία τοίχου 36W)
- Διεπαφή HDMI και πηγή HDMI
- Πηγή Mini DisplayPort
- Κωδικοποιητής ήχου 240 bit με τέσσερις υποδοχές 3,5 mm
- 10/100/1000 Ethernet PHY με ενσωματωμένη μοναδική διεύθυνση MAC
- Για τον χρήστη EEPROM
- Γέφυρα USB-UART
- Επιμελής διεπαφή παράλληλης μεταφοράς (DTPI) για μεταφορά δεδομένων υψηλής ταχύτητας μέσω USB
- Επιμελής σειριακή περιφερειακή διεπαφή (DSPI) για μεταφορά δεδομένων μέσης ταχύτητας μέσω USB
- Ενσωματωμένα περιβάλλοντα χρήστη: μονόχρωμη οθόνη OLED 128x32, 5 κουμπιά χρήστη, 8 διακόπτες χρήστη, 8 LED χρήστη
- USB HID υποδοχής για ποντίκια και πληκτρολόγια
- Πλήρης υποδοχή LPC 160 ακίδων FMC
- Τέσσερις θύρες Pmod, Pmod για σήματα XADC

2.3 AES

2.3.1 Ιστορία και περιγραφή του αλγορίθμου

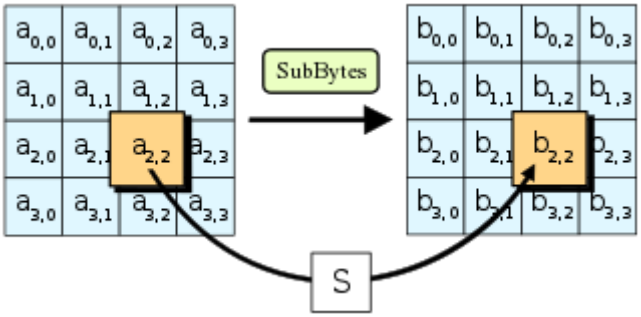
Ο αλγόριθμος AES (Advanced Encryption Standard) επίσης γνωστός ως Rinjael αποτελεί μια παραλλαγή του αλγορίθμου block cipher Rinjael, σχεδιάστηκε από τους Joan Daemen και Vincent Rijmen και αποτελεί την αντικατάσταση του DES(Data Encryption Standard) που υιοθετήθηκε από την κυβέρνηση των ΗΠΑ στις 26 Μαΐου το 2002 αφού ανακοινώθηκε από το NIST(National Institute of Standards and Technology) στης 26 Νοεμβρίου το 2001.Είναι ένας αλγόριθμος κρυπτογράφησης συμμετρικού κλειδιού , που σημαίνει ότι το κλειδί της κρυπτογράφησης είναι ίδιο με το κλειδί της αποκρυπτογράφησης. Υπάρχουν τρείς παραλλαγές του αλγορίθμου σχεδιασμένες να δέχονται κλειδιά διαφορετικού μεγέθους , συγκεκριμένα τα μεγέθη μπορεί να είναι 128, 192 και 256 bits , όμως και στις τρείς εκδοχές το μέγεθος των δεδομένων προς κρυπτογράφηση/αποκρυπτογράφηση πρέπει να είναι 128 bit. Ο AES είναι σχεδιασμένος βάση μίας αρχής γνωστής ως substitution-permutation network κάτι που τον κάνει αποδοτικό για υλοποίηση σε λογισμικό και σε υλικό. Ανάλογα με το μέγεθος του κλειδιού ο αλγόριθμος θα χρειαστεί : 10 κύκλους για κλειδί μήκους 128 bit ,12 για 192 bit και 14 για 256 bit.Η διαδικασία κρυπτογράφησης και αποκρυπτογράφησης αποτελείτε από πέντε διαδικασίες[6] :

- **KeyExpansion** : Η διαδικασία αυτή παράγει τα κλειδιά που θα χρησιμοποιηθούνε κατά την διάρκεια της κρυπτογράφησης/αποκρυπτογράφησης από τις άλλες διαδικασίες.
- **AddRoundKey** : Στην διαδικασία αυτή εκτελείται η πράξη XOR μεταξύ του κλειδιού που παράχθηκε από την **KeyExpansion** για αυτόν τον γύρο και των bits του γύρου.



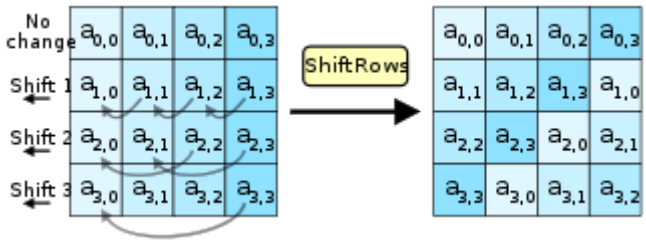
Εικόνα 2.3.1.1 Add Round Key

- **SubBytes** : Στην διαδικασία αυτή κάθε byte της κατάστασης στην οποία βρισκόμαστε αντικαθιστάται από ένα subbyte το οποίο βρίσκεται μέσα σε ένα 8 bit substitution box. Οι τιμές που βρίσκονται σε αυτό είναι προκαθορισμένες για τον AES και το box αυτό ονομάζεται Rijndael S-box , αξίζει να σημειωθεί πως πολλοί άλλοι block ciphers χρησιμοποιούν ξεχωριστές τιμές στο s-box ανάλογα την είσοδό τους. Επίσης οι τιμές που περιέχονται σε αυτό έχουν προέλθει από πολλαπλασιαστική αντιστροφή πάνω στο πεπερασμένο πεδίο GF(2^8)



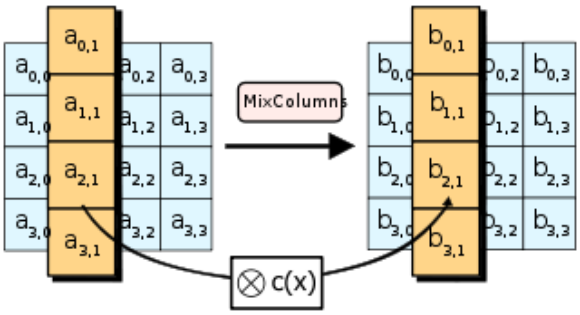
Εικόνα 2.3.1.2 Sub Bytes

- **ShiftRows** : Αυτή η διαδικασία αλλάζει κυκλικά τα bytes του εκάστοτε μπλοκ. Η πρώτη γραμμή παραμένει όπως είναι, τα bytes της ΄δεύτερης μετατοπίζονται κατά μία θέση αριστερά, της τρίτης κατά δύο και τις τέταρτης κατά τέσσερις θέσεις αντίστοιχα. Στόχος αυτής της διαδικασίας είναι να αποφευχθεί να κρυπτογραφηθούνε οι στήλες του block ξεχωριστά η μία από την άλλη καθώς αυτό θα έκανε τον AES να μετασχηματιστεί σε 4 ξεχωριστούς block ciphers.



Εικόνα 2.3.1.3 Shift Rows

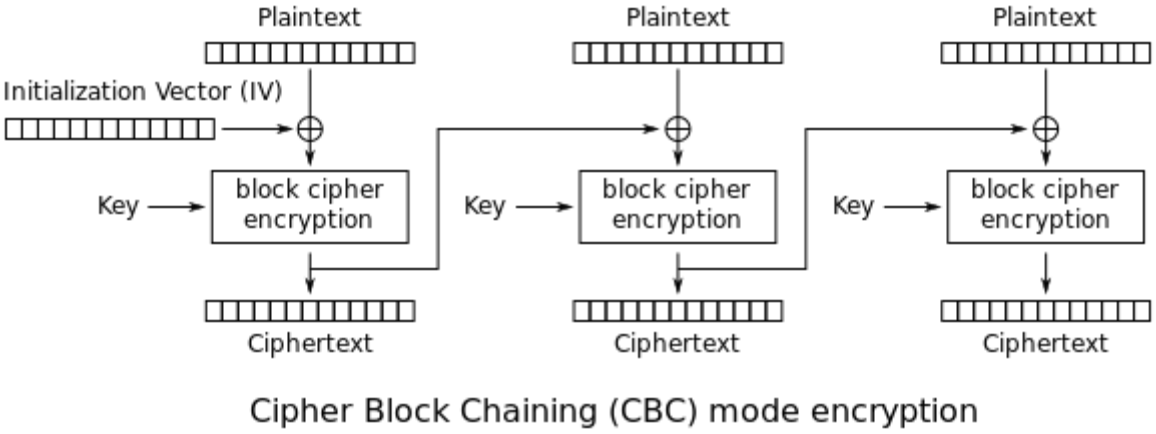
- **MixColumns** : Αυτή η διαδικασία ενώνει τα bytes των στηλών του μπλοκ χρησιμοποιώντας αντιστρέψιμο γραμμικό μετασχηματισμό. Το κάνει αυτό πραγματοποιώντας πολλαπλασιασμό πινάκων όπου ο ένας πίνακας είναι η στήλη και ο άλλος είναι ένας 4x4 πίνακας με προκαθορισμένες τιμές.



Εικόνα 2.3.1.4 Mix Columns

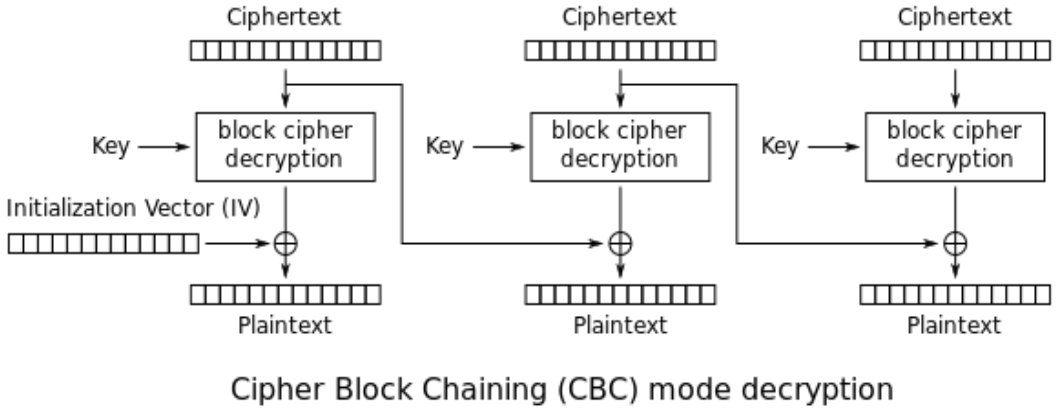
2.3.2 Ο AES στον CBC τρόπο λειτουργίας

Προκειμένου να κρυπτογραφηθεί αποτελεσματικά μία εικόνα είναι σημαντικό κάθε block 128 bit να είναι μοναδικό και να μην υπάρχει κανένα ίδιο με το άλλο. Προκειμένου να γίνει αυτό ο AES θα χρησιμοποιηθεί με τον CBC[16] τρόπο λειτουργίας. Σε αυτόν τον τρόπο λειτουργίας όταν φτάσει το πρώτο μπλοκ δεδομένων θα γίνει μια πράξη XOR μεταξύ αυτού και ενός διανύσματος αρχικοποίησης και στην συνέχεια θα κρυπτογραφηθεί. Έπειτα όταν φτάσει το επόμενο μπλοκ προς κρυπτογράφηση θα γίνει μια πράξη XOR μεταξύ αυτού και του προηγούμενου κρυπτογραφημένου μπλοκ. Η διαδικασία αυτή θα συνεχίσει να γίνεται για κάθε νέο μπλοκ προς κρυπτογράφηση εξασφαλίζοντας έτσι ότι κάθε κρυπτογραφημένο μπλοκ θα είναι μοναδικό.



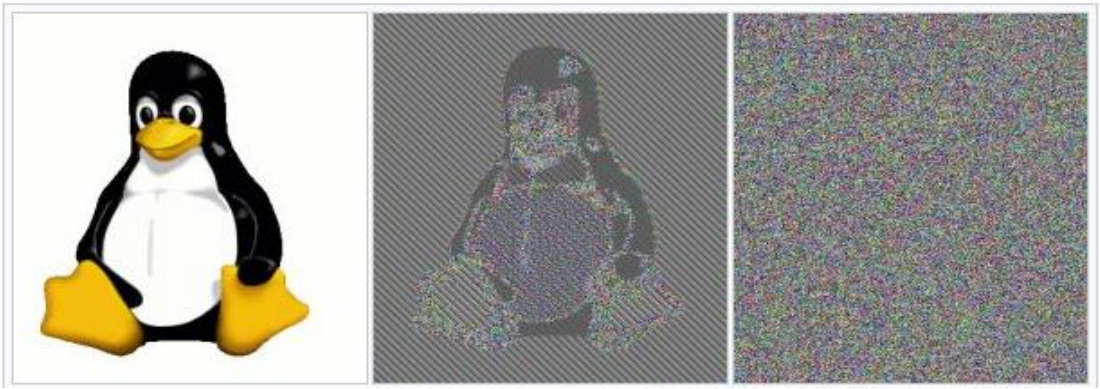
Εικόνα 2.3.2.1 CBC κρυπτογράφηση

Η διαδικασία αποκρυπτογράφησης στον CBC τρόπο λειτουργίας έχει ως εξής. Το πρώτο μπλοκ που θα ληφθεί θα αποκρυπτογραφηθεί κανονικά και το στο μπλοκ που θα παραχθεί θα γίνει η πράξη XOR χρησιμοποιώντας το ίδιο διάνυσμα αρχικοποίησης που χρησιμοποιήθηκε κατά την κρυπτογράφηση, το αποτέλεσμα αυτής της πράξης θα είναι το αποκρυπτογραφημένο αρχικό μπλοκ. Στην επόμενη αποκρυπτογράφηση η πράξη XOR θα γίνει χρησιμοποιώντας το κρυπτογραφημένο μπλοκ δεδομένων που λήφθηκε προηγουμένως. Η διαδικασία αυτή θα συνεχίσει να επαναλαμβάνεται χρησιμοποιώντας κάθε φορά για την πράξη XOR το προηγούμενο κρυπτογραφημένο μπλοκ.



Εικόνα 2.3.2.2 CBC αποκρυπτογράφηση

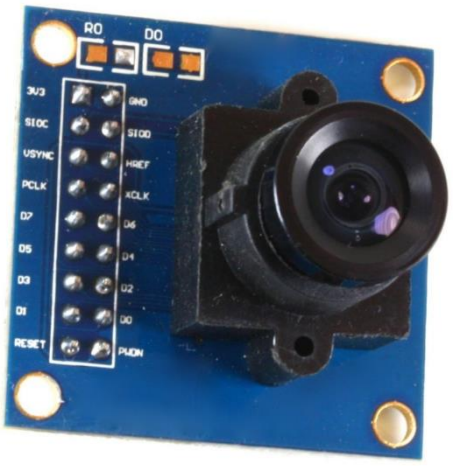
Στην παρούσα διπλωματική επιλέχθηκε μία έτοιμη υλοποίηση[15] του αλγόριθμου AES-128 η οποία δεν καταναλώνει πολλούς από τους πόρους του συστήματος η οποία τροποποιήθηκε κατάλληλα έτσι ώστε να λειτουργεί με τον CBC τρόπο λειτουργίας. Στην εικόνα 2.3.2.3 παρουσιάζονται τρεις φωτογραφίες, η πρώτη είναι μια απλή φωτογραφία, η δεύτερη είναι η φωτογραφία κρυπτογραφημένη με απλό AES και η τρίτη είναι η φωτογραφία κρυπτογραφημένη με AES CBC. Από αυτές παρατηρούμε ότι αν η εικόνα κρυπτογραφηθεί με τον απλό AES υπάρχει περίπτωση να διαρρεύσουν δεδομένα ενώ άμα κρυπτογραφηθεί με CBC AES όχι. Αυτός είναι ο λόγος που επιλέχθηκε αυτή η υλοποίηση του αλγορίθμου.



Εικόνα 2.3.2.3 Κρυπτογράφηση εικόνας με απλό AES και με AES CBC[28]

2.4 Η κάμερα OV7670

Για την υλοποίηση του συστήματος επιλέχθηκε να χρησιμοποιηθεί σαν βάση για το βίντεο η κάμερα OV7670.Η OV7670[11] είναι μία κάμερα-αισθητήρας εικόνας CMOS χαμηλής τάσης που παρέχει την πλήρη λειτουργικότητα μιας κάμερας VGA ενός τσιπ και επεξεργαστή εικόνας. Η OV7670 παρέχει 8-bit πλήρους καρέ, υποδειγματοληψίας ή παραθύρου εικόνας σε ένα ευρύ φάσμα μορφών, που ελέγχονται μέσω του Serial Camera Control Bus (SCCB). Διαθέτει μια συστοιχία εικόνων ικανή να λειτουργεί με ταχύτητα έως και 30 καρέ ανά δευτερόλεπτο (fps)σε VGA με πλήρη έλεγχο από τον χρήστη στην ποιότητα της εικόνας, τη μορφοποίηση και τη μεταφορά δεδομένων εξόδου. Όλες οι απαιτούμενες λειτουργίες επεξεργασίας εικόνας, όπως έλεγχος έκθεσης, γάμμα, ισορροπία λευκού, χρώμα ,ο κορεσμός, ο έλεγχος απόχρωσης και άλλα, μπορούν επίσης να προγραμματιστούν μέσω της διεπαφής SCCB.
Η παραμετροποίηση της κάμερα θα γίνει από την πλευρά του ελεγκτή πτήσης καθώς στην περίπτωσή μας θα ασχοληθούμε με την ασφάλιση των δεδομένων βίντεο.

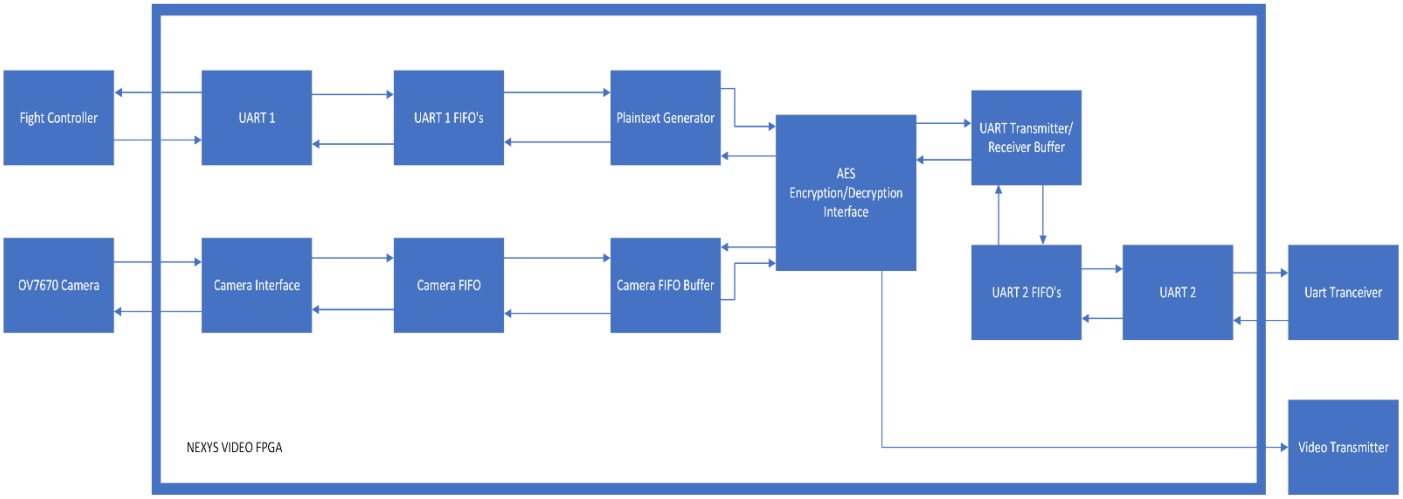


Εικόνα 2.4.1 Κάμερα OV7670[11]

3. Αρχιτεκτονική συστήματος

3.1 Διασύνδεση εξαρτημάτων συστήματος

Στην εικόνα 3.1.1 παρουσιάζεται ένα διάγραμμα της διασύνδεσης των εξαρτημάτων του συστήματος.

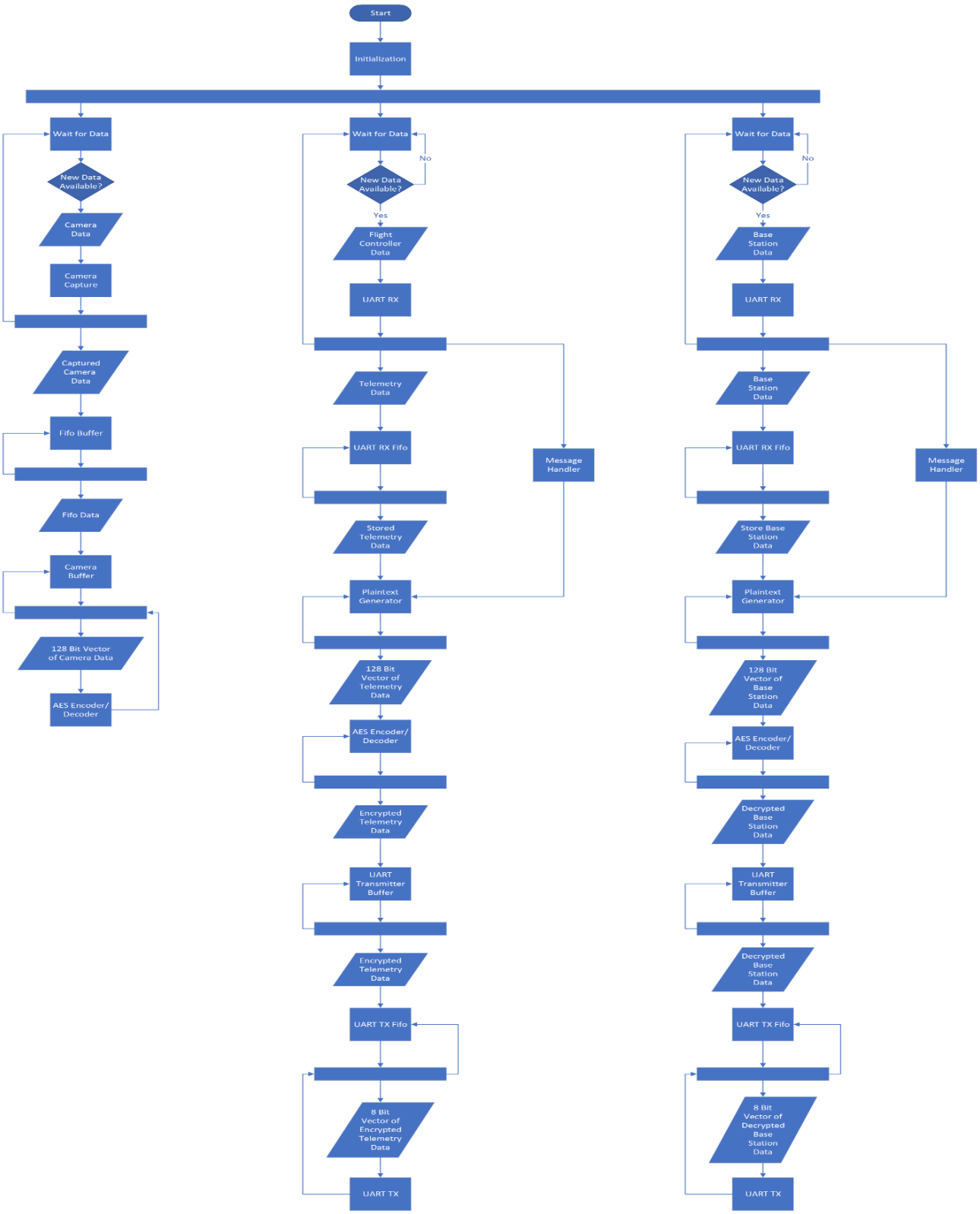


Εικόνα 3.1.1 Μπλοκ διάγραμμα εξαρτημάτων συστήματος

Θεωρούμε ότι ο ελεγκτής πτήσης επικοινωνεί με το FPGA χρησιμοποιώντας το πρωτόκολλο σειριακής επικοινωνίας UART. Αυτό είναι συνηθισμένο καθώς πολλοί ελεγκτές πτήσεις επικοινωνούν έτσι με το εξάρτημα ασύρματης μετάδοσης δεδομένων. Στην αρχιτεκτονική μας το FPGA τοποθετείται μεταξύ του ελεγκτή πτήσης και των

εξαρτημάτων ασύρματης επικοινωνίας. Επίσης για το συγκεκριμένο σύστημα θεωρούμε ότι όταν ο ελεγκτής πτήσης στείλει επιτυχώς ένα μήνυμα θα περιμένει για ένα χρόνο ίσο με 1 millisecond ή και περισσότερο. Αυτό είναι σημαντικό καθώς για να μην υπάρξει καθυστέρηση, από την στιγμή που το μήνυμα φτάνει μέσω UART, έως την στιγμή που βγαίνει μέσω UART, θα πρέπει να μην έχει ξεκινήσει να στέλνεται το επόμενο μήνυμα. Ο χρόνος που θα πρέπει να περιμένει ο ελεγκτής πτήσης είναι ανάλογος με την ταχύτητα επικοινωνίας του UART καθώς και με το πόσο γρήγορα το σύστημα μπορεί να κρυπτογραφήσει τα δεδομένα που λαμβάνει.

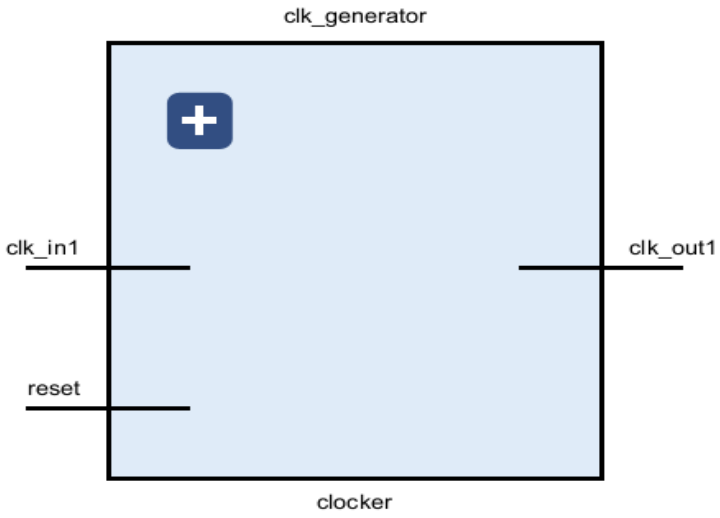
Στην εικόνα 3.1.2 παρουσιάζετε η ροή πληροφοριών μεταξύ των εξαρτημάτων του συστήματος τα οποία αναλύονται στην επόμενη ενότητα.



Εικόνα 3.1.1 Διάγραμμα ροής εξαρτημάτων συστήματος

3.2 Εξαρτήματα συστήματος

3.2.1 Clock Generator



Εικόνα 3.2.1.1 Εξάρτημα clock_generator

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
clk_in	Είσοδος	1	Ρολόι συχνότητας 100MHz
reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει επανεκκίνηση.

Πίνακας 3.2.1 Είσοδοι clock_generator

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
clk_out1	Έξοδος	1	Ρολόι συχνότητας 250MHz.

Πίνακας 3.2.2 Έξοδοι clock_generator

Το εξάρτημα Clock Generator είναι υπεύθυνο για την δημιουργία γρήγορου ρολογιού χρησιμοποιώντας σαν βάση το ρολόι που βρίσκεται ενσωματωμένο στην πλακέτα του FPGA. Το εξάρτημα αυτό και ο κώδικας που το συνοδεύει είναι έτοιμα από την Xilinx[27]. Σαν είσοδο δέχεται το ρολόι της πλακέτας του FPGA και στην έξοδό του έχει το νέο ρολόι. Η συχνότητα του καινούργιου ρολογιού επιλέχθηκε να είναι ίση με 250MHz.

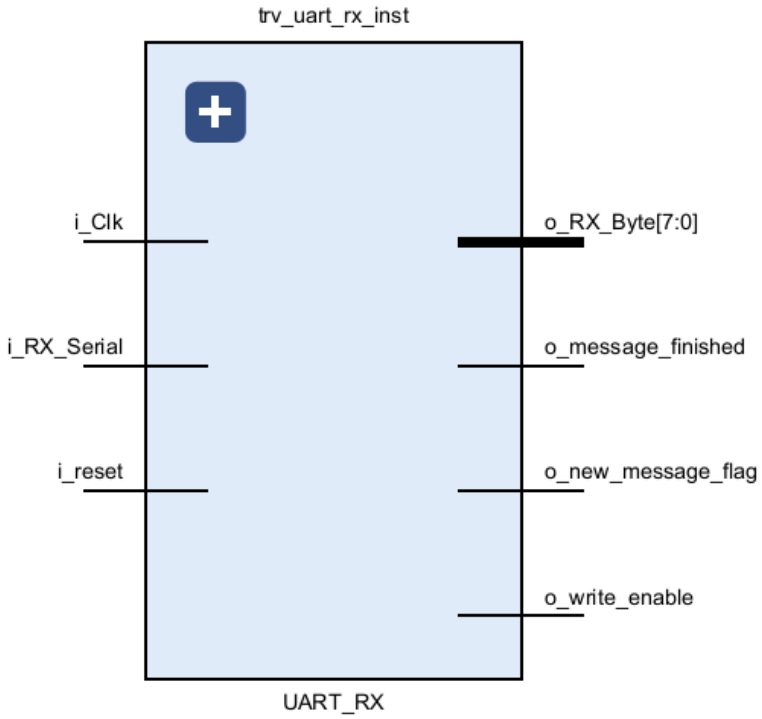
3.2.2 UART

Το UART[13] (Universal asynchronous receiver-transmitter) είναι ένα κύκλωμα το οποίο υλοποίει σειριακή ασύγχρονη επικοινωνία μεταξύ δύο συστημάτων/εξαρτημάτων. Η ταχύτητα επικοινωνίας (ρυθμός μετάδοσης) μπορεί να παραμετροποιηθεί και μετριέται σε baudrate (bits/sec).Το κύκλωμα του UART αποτελείται από δύο εξαρτήματα, το UART Receiver(UART RX) και το UART Transmitter(UART TX), το πρώτο είναι υπεύθυνο για την λήψη δεδομένων ενώ το δεύτερο για την αποστολή δεδομένων.

Η μεταφορά δεδομένων από το UART RX στο UART TX γίνεται με τον εξής τρόπο : Όσο δεν υπάρχουν διαθέσιμα δεδομένα το UART RX κρατάει την γραμμή δεδομένων στην λογική τιμή 1, με το που υπάρξουν δεδομένα η τιμή της θα γίνει ίση με 0 για ένα κύκλο ρολογιού του UART(η περίοδος και η συχνότητα του οποίου εξαρτώνται από το baudrate). Ύστερα στέλνονται τα 8 bit δεδομένα με το κάθε bit να στέλνεται ανά κύκλο ρολογιού. Με το πέρας αυτής της διαδικασίας θα σταλθεί το parity bit και τα stop bits (1 ή 2). Τα εξαρτήματα UART που χρησιμοποιήσαμε ήταν έτοιμα[13] και γίναν οι κατάλληλες αλλαγές στον κώδικά τους προκειμένου να μπορούν να χρησιμοποιηθούν στο σύστημά μας.

Στο σύστημα κρυπτογράφησης/αποκρυπτογράφησης υπάρχουν δύο UART RX εξαρτήματα και δύο UART TX εξαρτήματα. Το κάθε ζευγάρι (UART RX-TX) αναλαμβάνει την λήψη/μετάδοση δεδομένων για το κομμάτι του ξεχωριστά , δηλαδή υπάρχει ένα ζευγάρι που αναλαμβάνει την λήψη δεδομένων τηλεμετρίας από τον ελεγκτή πτήσης και την αποστολή τους στον UART Tranceiver και ένα ζευγάρι για την λήψη δεδομένων εντολών από τον UART Tranceiver και την αποστολή τους στον ελεγκτή πτήσης. Τα δύο UART λειτουργούν με το ίδιο baudarate το οποίο είναι ίσο με 115200 bps , χωρίς parity bit και δέχονται ένα byte ανά κύκλο start-end (8N1).

3.2.2.1 UART RX



Εικόνα 3.2.2.1 Εξάρτημα UART RX

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_Clk	Είσοδος	1	Ρολόι εξαρτήματος συχνότητας 250MHz
i_reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα διακόπτει την λειτουργία του και επιστρέφει στην αρχική του κατάσταση.
i_RX_Serial	Είσοδος	1	Δεδομένα εισόδου από τον ελεγκτή πτήσης.

Πίνακας 3.2.3 Είσοδοι UART RX

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_message_finished	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε έχουν σταλθεί 8 bit.
o_new_message_flag	Έξοδος	1	Όταν η τιμή του είναι 1 τότε έχουν ξεκινήσει να στέλνονται 8 bit από τον ελεγκτή πτήσης.
o_write_enable	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε είναι έτοιμα να αποθηκευτούν στην FIFO 8 bit.
o_RX_Byte	Έξοδος	8	Διάνυσμα δεδομένων που λήφθηκαν από τον ελεγκτή πτήσης.

Πίνακας 3.2.4 Έξοδοι UART RX

Η λειτουργικότητα του εξαρτήματος περιγράφεται από δύο διεργασίες :

- **P_SAMPLE** : Η διεργασία αυτή ενεργοποιείτε σε κάθε θετικό παλμό του ρολογιού του εξαρτήματος και σκοπός της είναι να μεταφέρει την τιμή της εισόδου i_RX_Serial στο clock domain του ρολογιού εισόδου. Είναι σημαντικό να γίνει αυτό καθώς το UART λειτουργεί με ένα baudrate του οποίου η συχνότητα είναι διαφορετική από εκείνη του ρολογιού εισόδου με αποτέλεσμα αν προσπαθήσουμε να διαβάσουμε την τιμή εισόδου i_RX_Serial χρησιμοποιώντας με βάση το ρολόι i_Clk να έχουμε λάθος τιμή. Για να διαβάσουμε την σωστή τιμή χρησιμοποιούμε αυτή τη διαδικασία και περνάμε την είσοδο μέσα από δύο flip-flop προκειμένου να συγχρονιστεί με το ρολόι i_Clk.

- **P_UART_RX** : Η διεργασία αυτή ενεργοποιείται σε κάθε θετικό παλμό του ρολογιού του εξαρτήματος. Αποτελείται από μία μηχανή πεπερασμένων καταστάσεων της οποίας οι καταστάσεις ονομάζονται s_Idle, s_RX_Start_Bit, s_RX_Data_Bits, s_RX_Stop_Bit και s_Cleanup. Επίσης υπάρχει και μια συνθήκη if..else η οποία ενεργοποιείται αν το σήμα εισόδου i_reset είναι ίσο με 1 και μεταφέρει την μηχανή πεπερασμένων καταστάσεων στην αρχική συνθήκη s_idle προκειμένου να ξεκινήσει πάλι από την αρχή σε περίπτωση επανεκκίνησης του συστήματος.

    1. **s_Idle** : Η πρώτη κατάσταση. Όταν βρίσκεται σε αυτήν όλες οι έξοδοι είναι ίσες με 0 και περιμένει η τιμή του σήματος εισόδου i_RX_Serial να γίνει μηδέν, όταν γίνει αυτό τότε έχουμε την συνθήκη εκκίνησης του UART ,θα μεταβούμε στην κατάσταση **s_RX_Start_Bit** και θα κάνουμε την τιμή της εξόδου o_new_message_flag ίση με 1 .

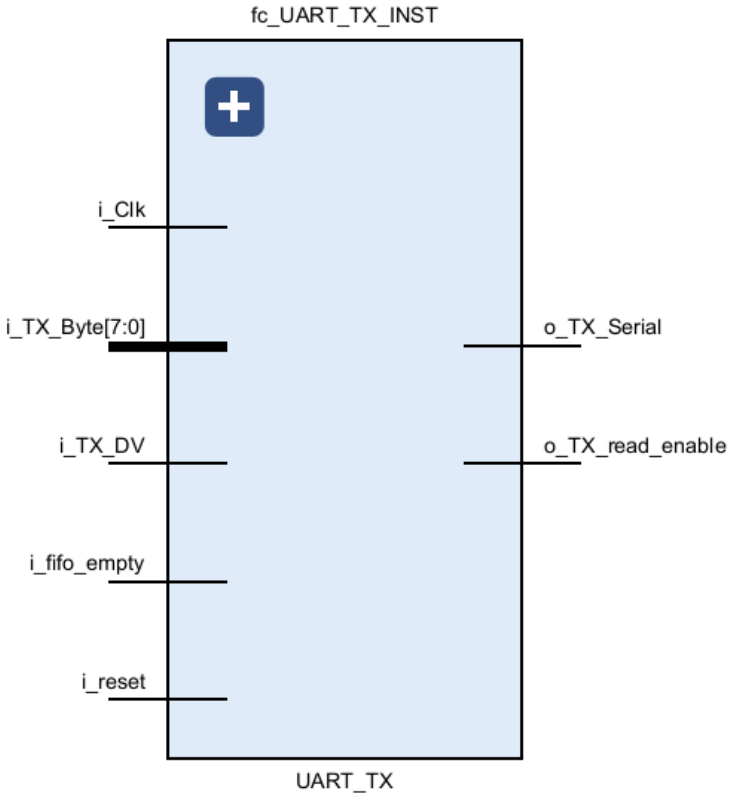
    2. **s_RX_Start_Bit**: Σε αυτήν την κατάσταση ανιχνεύεται η "μέση" του bit εκκίνησης που διαβάστηκε από την πρώτη κατάσταση. Χρησιμοποιώντας έναν μετρητή με όνομα r_Clk_Count μετράει το εξάρτημα παλμού ρολογιού έως ότου φτάσει στην τιμή (g_CLKS_PER_BIT – 1)/2, με το πέρας αυτής της διαδικασίας μεταφέρεται στην κατάσταση **s_RX_Data_Bits** και ο μετρητής μηδενίζεται. Είναι σημαντική αυτή η κατάσταση καθώς αντιμετωπίζει την περίπτωση όπου υπήρχε σφάλμα κατά την εκκίνηση του UART, στην περίπτωση που η τιμή στην γραμμή εισόδου αλλάξει από 0 σε ένα και δεν έχει ολοκληρωθεί η συνθήκη του μετρητή το εξάρτημα θα μεταβεί ξανά στην πρώτη κατάσταση όπου και θα περιμένει για το bit εκκίνησης.

    3. **s_RX_Data_Bits**: Σε αυτή την κατάσταση το εξάρτημα διαβάζει την τιμή της γραμμής εισόδου κάθε g_CLKS_PER_BIT–1 χρησιμοποιώντας τον μετρητή r_Clk_Count και όταν ολοκληρωθεί η συνθήκη αυτή αποθηκεύει την τιμή που διάβασε σε ένα διάνυσμα με όνομα r_RX_Byte έως ότου αποθηκεύσει συνολικά 8 bit , με το που γίνει αυτό μεταβαίνει το εξάρτημα στην κατάσταση **s_RX_Stop_Bit** .

    4. **s_RX_Stop_Bit** : Σε αυτήν την κατάσταση το εξάρτημα θα περιμένει για άλλη μια φορά τον μετρητή r_Clk_Count να φτάσει στην τιμή g_CLKS_PER_BIT–1**.** Με το πέρας της διαδικασίας αυτής η τιμή της εξόδου o_message_finished θα γίνει ίση με 1 και της εξόδου o_new_message_flag ίση με 0. Έπειτα στην έξοδο o_RX_Byte θα σταλθεί το διάνυσμα r_RX_Byte και η τιμή της εξόδου o_write_enable θα γίνει ίση με 1 και θα μεταβεί το εξάρτημα στην κατάσταση **s_Cleanup.**

5. **s_Cleanup :** Στην τελική αυτή κατάσταση του εξαρτήματος οι τιμές των σημάτων εξόδου o_message_finished και o_write_enable θα γίνουν 0 και το εξάρτημα θα μεταβεί στην αρχική κατάσταση **S_Idle** προκειμένου να ξεκινήσει πάλι η διαδικασία λήψης δεδομένων του UART RX.

3.2.2.2 UART TX



Εικόνα 3.2.2.2 Εξάρτημα UART TX

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_Clk	Είσοδος	1	Ρολόι εξαρτήματος συχνότητας 250MHz.
i_reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα διακόπτει την λειτουργία του και επιστρέφει στην αρχική του κατάσταση.
i_TX_DV	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε μπορούν να διαβαστούν δεδομένα από την FIFO.
i_TX_Byte	Είσοδος	8	Διάνυσμα δεδομένων εισόδου.
i_fifo_empty	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO είναι άδεια.

Πίνακας 3.2.5 Είσοδοι UART TX

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_TX_Serial	Έξοδος	1	Γραμμή δεδομένων εξόδου.
o_TX_read_enable	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα διαβάζει δεδομένα από την FIFO.

Πίνακας 3.2.6 Έξοδοι UART TX

Η λειτουργικότητα του εξαρτήματος περιγράφεται από μία διεργασία :

- **P_UART_TX** : Η διεργασία αυτή ενεργοποιείται σε κάθε θετικό παλμό του ρολογιού του εξαρτήματος.

Αποτελείται από μία μηχανή πεπερασμένων καταστάσεων της οποίας οι καταστάσεις ονομάζονται s_Idle, s_TX_Start_Bit, s_TX_Data_Bits, s_TX_Stop_Bit, s_Cleanup και s_latch_byte. Επίσης υπάρχει και μια συνθήκη if..else η οποία ενεργοποιείται αν το σήμα εισόδου i_reset είναι ίσο με 1 και μεταφέρει την μηχανή πεπερασμένων καταστάσεων στην αρχική συνθήκη **s_idle** προκειμένου να ξεκινήσει πάλι από την αρχή σε περίπτωση επανεκκίνησης του συστήματος.

1. **s_Idle** : Η πρώτη κατάσταση του εξαρτήματος. Όταν βρίσκεται σε αυτήν όλες οι έξοδοι είναι ίσες με 0 και περιμένει η τιμή του σήματος εισόδου i_fifo_empty να γίνει μηδέν, όταν η FIFO που είναι συνδεδεμένη με το εξάρτημα περιέχει δεδομένα προς μετάδοση και το εξάρτημα θα μεταβεί στην κατάσταση **s_latch_byte** και θα κάνει την τιμή του σήματος o_TX_read_enable ίση με 1 .

2. **s_latch_byte** : Σε αυτήν την κατάσταση το εξάρτημα κάνει την τιμή του σήματος εξόδου o_TX_read_enable ίση με 0, στην συνέχεια ελέγχεται η τιμή του σήματος εισόδου i_TX_DV, αν η τιμή του είναι ίση με 1 τότε αποθηκεύεται η τιμή του διανύσματος εισόδου i_TX_Byte στο τοπικό διάνυσμα r_TX_Data, η τιμή του σήματος εξόδου o_TX_read_enable παραμένει ίση με 0 και το εξάρτημα μεταβαίνει στην κατάσταση **s_TX_Start_Bit**.

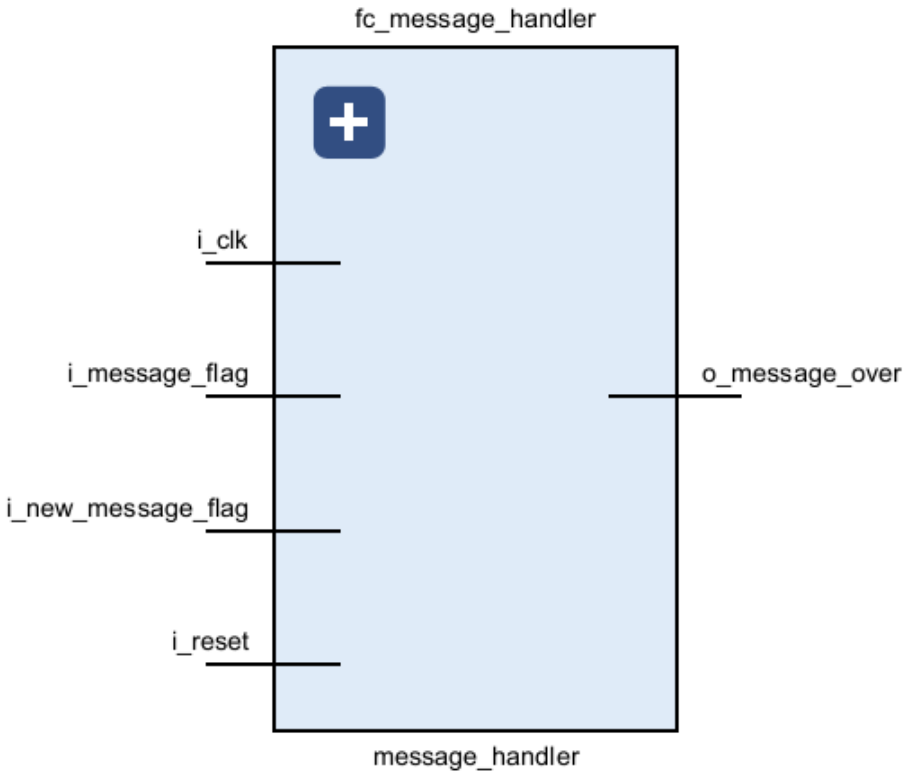
3. **s_TX_Start_Bit** : Σε αυτή την κατάσταση το εξάρτημα αρχικά κάνει την τιμή του σήματος εξόδου o_TX_Serial ίση με 0 προκειμένου να ξεκινήσει η διαδικασία αποστολής δεδομένων στο UART. Στην συνέχεια το εξάρτημα θα περιμένει για χρόνο ίσο με g_CLKS_PER_BIT-1 προκειμένου να μπορεί να ανιχνευθεί το start bit από τον transmitter. Προκειμένου να επιτευχθεί αυτό ,το εξάρτημα χρησιμοποιεί έναν μετρητή με όνομα r_Clk_Count τον οποίο αυξάνει κατά 1 σε κάθε θετικό παλμό του ρολογιού εισόδου έως ότου φτάσει στην τιμή g_CLKS_PER_BIT-1.Όταν γίνει αυτό , το εξάρτημα θα μηδενίσει τον μετρητή και θα μεταβεί στην κατάσταση **s_TX_Data_Bits**.

4. **s_TX_Data_Bits** : Σε αυτήν την κατάσταση το εξάρτημα θα γράψει στο σήμα εξόδου o_TX_Serial τα δεδομένα του τοπικού διανύσματος r_TX_Data. Για να γίνει αυτό το εξάρτημα χρησιμοποιεί μία μεταβλητή με όνομα r_Bit_Index της οποίας η τιμή αντιστοιχεί στην θέση ενός bit μέσα στο r_TX_Data καθώς και τον μετρητή r_Clk_Count. Για να βγει ένα δεδομένο από το εξάρτημα θα πρέπει να ισχύει η συνθήκη r_Clk_Count = g_CLKS_PER_BIT-1, κάθε φορά που συμβαίνει αυτό ο μετρητής r_Clk_Count θα γίνεται 0 και θα ελέγχεται η τιμή της μεταβλητής r_Bit_Index. Αν η τιμή της είναι μικρότερη από το επτά τότε θα αυξάνεται κατά 1 και θα επαναλαμβάνεται η παραπάνω διαδικασία. Όταν η τιμή της γίνει ίση με επτά τότε το εξάρτημα θα την μηδενίσει και θα μεταβεί στην κατάσταση **s_TX_Stop_Bit**.

5. **s_TX_Stop_Bit** : Σε αυτήν την κατάσταση το εξάρτημα κάνει την τιμή του σήματος εξόδου o_TX_Serial ίση με 1 προκειμένου να ξεκινήσει η διαδικασία τερματισμού της επικοινωνίας του UART. Θα παραμείνει σε αυτήν την κατάσταση έως ότου r_Clk_Count = g_CLKS_PER_BIT-1, τότε θα μεταβεί στην κατάσταση **s_Cleanup**.

6. **s_Cleanup** : Το εξάρτημα θα περιμένει σε αυτήν την κατάσταση για ένα κύκλο ρολογιού και στην συνέχεια θα μεταβεί στην αρχική κατάσταση **s_Idle**.

3.2.3 Message Handler



Εικόνα 3.2.3.1 Εξάρτημα Message Handler

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_clk	Είσοδος	1	Ρολόι εξαρτήματος συχνότητας 250MHz.
i_reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα διακόπτει την λειτουργία του και επιστρέφει στην αρχική του κατάσταση.
i_message_flag	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε έχει ληφθεί από το UART RX ένα byte.
i_new_message_flag	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το UART RX έχει ξεκινήσει να λαμβάνει ένα byte.

Πίνακας 3.2.7 Είσοδοι Message Handler

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_message_over	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε έχει ληφθεί μήνυμα με μήκος μη διαιρέσιμο με το 128.

Πίνακας 3.2.8 Έξοδοι Message Handler

Το Message Handler εξάρτημα είναι υπεύθυνο για να μπορέσει το σύστημα να κρυπτογραφήσει/αποκρυπτογραφήσει δεδομένα όταν το μέγεθός τους είναι μικρότερο από 128 bit (το μέγεθος δεδομένων που πάνε στον AES) ,το οποίο συμβαίνει όταν ένα μήνυμα από τον ελεγκτή πτήσης ή τον σταθμό βάσης έχει μέγεθος το οποίο δεν διαιρείται τέλεια με το 128 .Καταφέρνει να το κάνει αυτό με τον εξής τρόπο : Κάθε φορά που τελειώνει το UART RX το message handler

αρχίζει να αυξάνει έναν μετρητή , όταν ο μετρητής φτάσει σε μία συγκεκριμένη τιμή τότε στέλνει ένα σήμα στο εξάρτημα με όνομα Plaintext generator το οποίο θα γεμίσει τις άδειες θέσεις ενός διανύσματος μεγέθους 128 bit που αποθηκεύει τα δεδομένα που θα κρυπτογραφηθούνε/αποκρυπτογραφηθούνε με τα δεδομένα του προηγούμενου μηνύματος προκειμένου ο AES να μπορέσει να τα κρυπτογραφήσει/αποκρυπτογραφήσει. Η τιμή του μετρητή είναι ανάλογη με τον χρόνο που περιμένει ο ελεγκτής πτήσης για να στείλει το επόμενο μήνυμα .

Η βασική λειτουργία αυτού του εξαρτήματος είναι να αυξάνει έναν μετρητή με όνομα message_counter κάθε φορά που η τιμή του σήματος εισόδου i_message_flag γίνεται ίση με 1.Αυτό συμβαίνει όταν το εξάρτημα UART RX που συνδέεται με το message_handler λαμβάνει επιτυχώς ένα byte και το αποστέλλει στην FIFO που συνδέεται με αυτό.

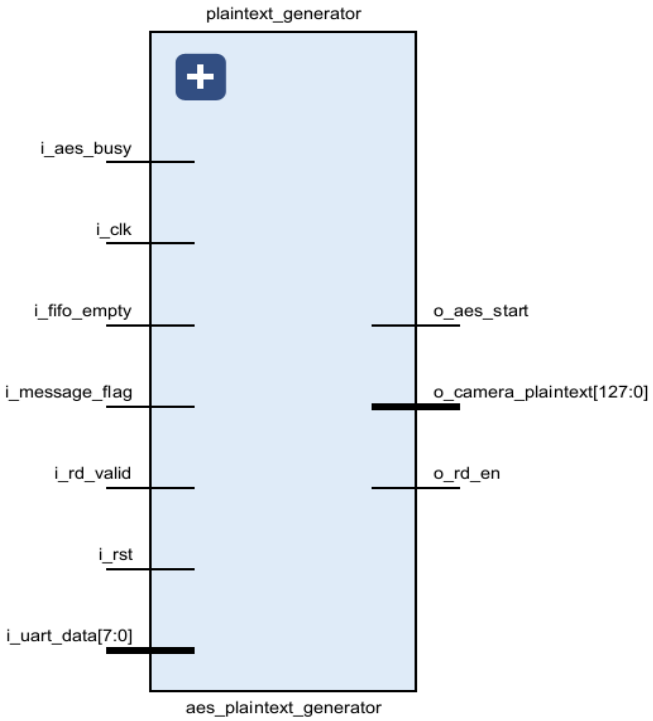
Η λειτουργία του εξαρτήματος περιλαμβάνεται μέσα σε μία μηχανή πεπερασμένων καταστάσεων με ονόματα reset, start, wait_for_message . Πριν μπει στην μηχανή αυτή το εξάρτημα ,πρώτα θα ελέγξει την τιμή του σήματος i_reset.

1. **reset** : Το εξάρτημα πηγαίνει σε αυτήν την κατάσταση όταν το σήμα εισόδου **reset** γίνει ίσο με 1, τότε όλα τα εσωτερικά σήματα γίνονται καθώς και τα σήματα εξόδου γίνονται ίσα με το 0 και το εξάρτημα μεταβαίνει στην αρχική κατάσταση **wait_for_message**.

2. **wait_for_message** : Αυτή είναι η αρχική κατάσταση του εξαρτήματος κατά την οποία περιμένει την τιμή του σήματος εισόδου i_message_flag να γίνει ίση με 1 ενώ παράλληλα βάζει στο σήμα εξόδου o_message_over την τιμή 0. Για να γίνει αυτό θα πρέπει το εξάρτημα UART RX που συνδέεται με το message_handler να λάβει επιτυχώς ένα byte και να το στείλει στην FIFO που συνδέεται μαζί του. Όταν γίνει αυτό ο μετρητής message_counter θα αυξηθεί κατά 1 και το εξάρτημα θα μεταβεί στην κατάσταση **start**.

3. **start** : Σε αυτήν την κατάσταση του εξαρτήματος αρχικά ελέγχεται η τιμή του μετρητή message_counter. Αν είναι ίση με 16 τότε σημαίνει ότι έχουν αποσταλεί από το UART RX συνολικά 128 bit (16 byte) τα οποία μπορούν να κρυπτογραφηθούν από τον AES, ο μετρητής message_counter γίνεται ίσως με το 0 και το εξάρτημα μεταβαίνει στην αρχική κατάσταση **wait_for_message**. Αν ο μετρητής message_counter είναι μικρότερος από το 16 τότε αρχικά ελέγχεται η τιμή του σήματος εισόδου i_new_message flag, αν είναι ίση με το 0 τότε αρχίζει να αυξάνεται κατά 1 η τιμή ενός άλλου μετρητή με όνομα counter έως ότου γίνει ίση με 6510. Αν η τιμή του counter γίνει 6510 και η τιμή του i_new_message_flag είναι 0 τότε σημαίνει ότι έχει ληφθεί όλο το μήνυμα από τον ελεγκτή πτήσης ή από τον σταθμό βάσης και το μέγεθος του συνολικά δεν διαιρείται με το 128 οπότε η τιμή του σήματος εξόδου o_message_over γίνεται ίση με 1 και το εξάρτημα μεταβαίνει στην κατάσταση wait_for_message κάνοντας τις τιμές των μετρητών 0. Αν η τιμή του σήματος εισόδου i_new_message_flag γίνει 1 τότε ο μετρητής counter γίνεται ίσως με το 0 και το εξάρτημα μεταβαίνει στην κατάσταση **wait_for_message**.

3.2.4 Plaintext Generator



Εικόνα 3.2.4.1 Εξάρτημα Plaintext Generator

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_clk	Είσοδος	1	Ρολόι εξαρτήματος συχνότητας 250MHz.
i_rst	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα διακόπτει την λειτουργία του και επιστρέφει στην αρχική του κατάσταση.
i_rd_valid	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε μπορούν να διαβαστούν δεδομένα από την FIFO.
i_fifo_empty	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν μπορούν να διαβαστούν δεδομένα από την FIFO.
i_uart_data	Είσοδος	8	Διάνυσμα δεδομένων εισόδου από την FIFO.
i_message_flag	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε έχει ληφθεί μήνυμα μεγέθους μη διαιρέσιμο με το 128.
i_aes_busy	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα AES Encoder/Decoder κρυπτογραφεί δεδομένα τηλεμετρίας.

Πίνακας 3.2.9 Είσοδοι Plaintext Generator

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_rd_en	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε ξεκινάει το εξάρτημα να διαβάζει δεδομένα από την FIFO.
o_plaintext	Έξοδος	128	Διάνυσμα δεδομένων εξόδου που περιέχει δεδομένα τηλεμετρίας προς κρυπτογράφηση.
o_aes_start	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 θα ξεκινήσει η κρυπτογράφηση δεδομένων τηλεμετρίας.

Πίνακας 3.2.10 Έξοδοι Plaintext Generator

Το Plaintext Generator εξάρτημα είναι υπεύθυνο για την κατασκευή διανυσμάτων μεγέθους 128 bit για να κρυπτογραφηθούν/αποκρυπτογραφηθούν από τον AES. Δέχεται σαν είσοδο δεδομένα μεγέθους 8 bit και τα προσθέτει σε ένα διάνυσμα μεγέθους 128 bit, όταν γεμίσει τότε θα στείλει το διάνυσμα στο εξάρτημα AES Encoder/Decoder προκειμένου να κρυπτογραφηθούν ή να αποκρυπτογραφηθούν. Σε περίπτωση που το μέγεθος του μηνύματος από τον ελεγκτή πτήσης ή από τον σταθμό βάσης δεν διαιρείται τέλεια από το 128, το Plaintext Generator θα λάβει ένα flag από το εξάρτημα με όνομα message handler προκειμένου να στείλει τα δεδομένα που έχει λάβει για κρυπτογράφηση ή αποκρυπτογράφηση χωρίς να έχει γεμίσει απαραίτητα το διάνυσμα. Στην αρχιτεκτονική μας υπάρχουν λοιπόν δύο στιγμιότυπα του Plaintext Generator, ένα για κρυπτογράφηση και μία για αποκρυπτογράφηση.

Το εξάρτημα αποτελείται από μία διαδικασία η οποία ενεργοποιείται σε κάθε θετικό παλμό του ρολογιού εισόδου i_clk. Μέσα σε αυτήν υπάρχει μια μηχανή πεπερασμένων καταστάσεων για την λειτουργία του εξαρτήματος η οποία έχει πέντε καταστάσεις με ονόματα reset,start,fill_buffer,send_data.

1. **reset** : Σε αυτήν την κατάσταση φτάνει το εξάρτημα όταν το σήμα εισόδου i_rst είναι ίσο με 1 και επιστρέφει στην κατάσταση **fill_buffer** αφού όμως πρώτα αλλάξει τις τιμές των σημάτων του στην αρχική μορφή τους .

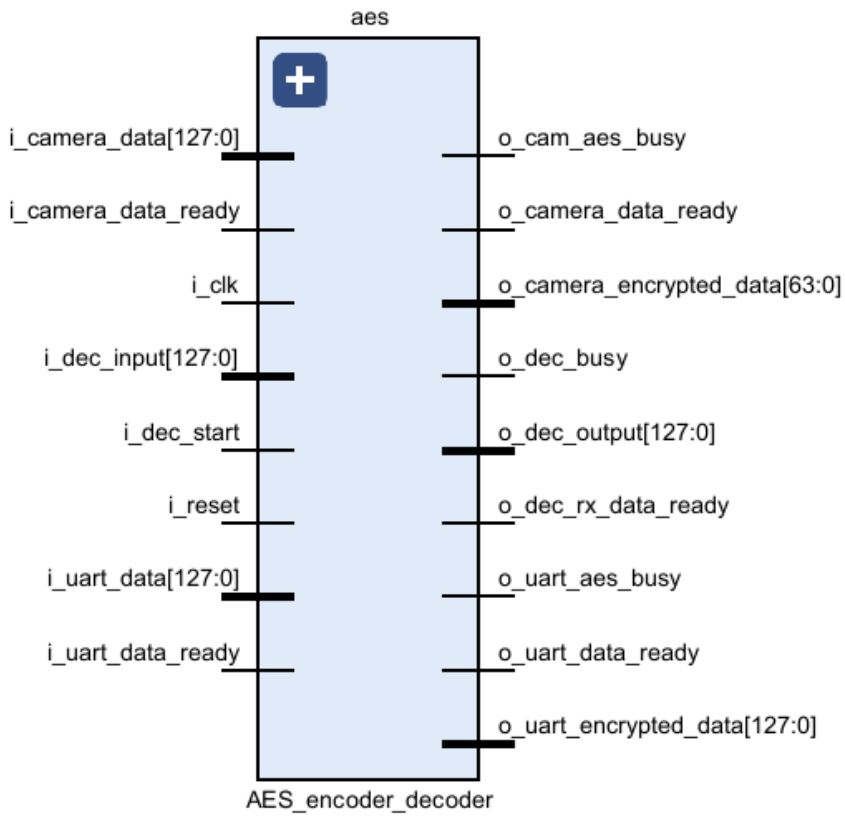
2. **start** : Η αρχική κατάσταση του εξαρτήματος, σε αυτήν η τιμή του σήματος εξόδου o_rd_en γίνεται ίση με 1 προκειμένου να μπορέσει το εξάρτημα να διαβάσει τα περιεχόμενα της FIFO που συνδέεται με αυτό, η FIFO αυτή περιέχει μέσα τα bytes που έλαβε το εξάρτημα UART RX. Μετά το εξάρτημα πηγαίνει στην κατάσταση **fill_buffer**.

3. **fill_buffer** : Σε αυτή την κατάσταση το εξάρτημα θα ξεκινήσει να γεμίζει ένα διάνυσμα με όνομα data_buffer με τις τιμές του διανύσματος εισόδου i_uart_data οι οποίες λαμβάνονται από την FIFO η οποία είναι συνδεδεμένη με το UART RX και το εξάρτημα αυτό. Προκειμένου να διαβαστούν οι τιμές αυτές θα πρέπει το σήμα εισόδου i_rd_valid να είναι ίσο με 1, αυτό συμβαίνει όταν τα δεδομένα που έχουν σταλθεί από το UART RX στην FIFO έχουν αποθηκευτεί με επιτυχία και είναι διαθέσιμα στην έξοδό της. Όταν τα δεδομένα αυτά είναι διαθέσιμα και ληφθούν από το εξάρτημα θα μπουν μέσα στο διάνυσμα data_buffer. Για να γίνει αυτό χρησιμοποιείται μια μεταβλητή με όνομα buffer_index σαν δείκτης που δείχνει σε μία θέση του διανύσματος data_buffer και η αρχική της τιμή είναι ίση με 7 , που σημαίνει ότι δείχνει στην έβδομη θέση του διανύσματος. Κάθε φορά που βρισκόμαστε σε αυτήν την κατάσταση και υπάρχουν διαθέσιμα δεδομένα θα τοποθετούνται στις θέσεις του διανύσματος ξεκινώντας από την θέση με νούμερο ίδιο με αυτό της μεταβλητής buffer_index μέχρι την θέση buffer_index-7 και στην συνέχεια αυξάνεται η τιμή της κατά 8.Όταν η τιμή της buffer_index ξεπεράσει το 127 τότε θα σταλθούν τα δεδομένα του data_buffer στο διάνυσμα εξόδου o_plaintext ,τα δεδομένα που βρίσκονται στην είσοδο i_uart_data θα μπουν στις εφτά πρώτες θέσεις του data_buffer , η τιμή του buffer_index θα γίνει ίση με 15 και η κατάσταση θα αλλάξει στην **send_data**. Στην περίπτωση όμως που η τιμή του σήματος εισόδου i_message_flag είναι ίση με ένα τότε σημαίνει πως το εξάρτημα message_handler ανίχνευσε πως λήφθηκε από το σύστημα μήνυμα μεγέθους που δεν διαιρείτε τέλεια με το 128, συνεπώς έχουν τοποθετηθεί στο data_buffer όλα τα διαθέσιμα δεδομένα. Σε αυτήν την περίπτωση τα δεδομένα του data_buffer θα σταλθούν στην έξοδο o_plaintext και η κατάσταση θα αλλάξει στην **send_data**.

4. **send_data** : Σε αυτή την κατάσταση το εξάρτημα θα ελέγξει αρχικά την τιμή του σήματος εισόδου i_aes_busy. Το σήμα αυτό προέρχεται από το εξάρτημα AES Encoder/Decoder και αν η τιμή του είναι ίση με 1 σημαίνει

ότι δεν έχει τελειώσει η κρυπτογράφηση δεδομένων ακόμα. Όταν η τιμή του γίνει 0 τότε το Plaintext_generator θα κάνει την τιμή του σήματος εξόδου o_aes_start ίση με 1 προκειμένου τα δεδομένα που βρίσκονται στο διάνυσμα εξόδου o_plaintext να ληφθούν από το AES Encoder/Decoder και να κρυπτογραφηθούν η αποκρυπτογραφηθούν. Με το πέρας αυτής της διαδικασίας , το εξάρτημα θα επιστρέψει στην κατάσταση **fill_buffer**.

3.2.5 AES Encoder/Decoder



Εικόνα 3.2.5.1 Εξάρτημα AES Encoder/Decoder

Όνομα pin	**Τύπος**	**Μήκος (bits)**	**Περιγραφή**
i_clk	Είσοδος	1	Ρολόι συχνότητας 250MHz
i_reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει reset στην αρχική του κατάσταση.
i_camera_data_ready	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 υπάρχουν διαθέσιμα δεδομένα βίντεο για κρυπτογράφηση.
i_uart_data_ready	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 υπάρχουν διαθέσιμα δεδομένα τηλεμετρίας για κρυπτογράφηση.
i_camera_data	Είσοδος	128	Δεδομένα βίντεο.
i_uart_data	Είσοδος	128	Δεδομένα τηλεμετρίας.
i_dec_input	Είσοδος	128	Δεδομένα σταθμού βάσης.
i_dec_start	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 υπάρχουν διαθέσιμα δεδομένα σταθμού βάσης για αποκρυπτογράφηση.

Πίνακας 3.2.11 Είσοδοι AES Encoder/Decoder

Όνομα pin	**Τύπος**	**Μήκος (bits)**	**Περιγραφή**
o_camera_data_ready	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 υπάρχουν διαθέσιμα κρυπτογραφημένα δεδομένα βίντεο.
o_uart_data_ready	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 υπάρχουν διαθέσιμα κρυπτογραφημένα δεδομένα τηλεμετρίας.
o_cam_aes_busy	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν έχει τελειώσει ακόμα η κρυπτογράφηση δεδομένων βίντεο.
o_uart_aes_busy	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν έχει τελειώσει ακόμα η κρυπτογράφηση δεδομένων τηλεμετρίας.
o_camera_encrypted_data	Έξοδος	64	Διάνυσμα εξόδου κρυπτογραφημένων δεδομένων βίντεο.
o_uart_encrypted_data	Έξοδος	128	Διάνυσμα εξόδου κρυπτογραφημένων δεδομένων τηλεμετρίας.
o_dec_busy	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν έχει τελειώσει ακόμα η αποκρυπτογράφηση δεδομένων εντολών.
o_dec_output	Έξοδος	128	Διάνυσμα εξόδου αποκρυπτογραφημένων δεδομένων εντολών.
o_dec_rx_data_ready	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 υπάρχουν διαθέσιμα αποκρυπτογραφημένα δεδομένα εντολών.

Πίνακας 3.2.12 Έξοδοι AES Encoder/Decoder

Το εξάρτημα **AES Encoder/Decoder** είναι υπεύθυνο για την κρυπτογράφηση δεδομένων τηλεμετρίας και βίντεο καθώς και για την αποκρυπτογράφηση δεδομένων εντολών που προέρχονται από τον σταθμό βάση. Η διαδικασία αυτή γίνεται μέσο του αλγορίθμου **AES-128** σε **CBC** mode. Προκειμένου να επιτευχθεί αυτό χρησιμοποιούνται δύο εξαρτήματα ,

το **encrypt_128_opt** και **decrypt_128**[15] .Το πρώτο είναι υπεύθυνο για την κρυπτογράφηση και το δεύτερο για την αποκρυπτογράφηση. Προκειμένου να καταλάβει το εξάρτημα από πού προέρχονται τα δεδομένα προς κρυπτογράφηση χρησιμοποιούνται δύο σήματα με όνομα **i_camera_data_ready** για τα δεδομένα της κάμερας και **i_uart_data_ready** για τα δεδομένα τηλεμετρίας ,τα οποία προέρχονται από τα εξαρτήματα **Plaintext Generator** και **Camera Buffer** αντίστοιχα. Όταν τελειώσει η κρυπτογράφηση τα κρυπτογραφημένα δεδομένα θα βγουν από τα διανύσματα εξόδου με όνομα **o_camera_encrypted_data**(δεδομένα κάμερας) και **o_uart_encrypted_data**(δεδομένα τηλεμετρίας).Στην περίπτωση της κάμερας τα δεδομένα θα βγούνε σε δύο κύκλους ρολογιού όπου σε κάθε κύκλο θα βγαίνουν 64 bits.

Το εξάρτημα αποτελείται από τρία υπο-εξαρτήματα και τρείς διαδικασίες οι οποίες τα ελέγχουν. Τα υπο-εξαρτήματα αυτά είναι δύο πυρήνες κρυπτογράφησης με ονόματα cam_enc_inst και uart_enc_inst οι οποίοι είναι στιγμιότυπα του εξαρτήματος **encrypt_128_opt** και ένας πυρήνας αποκρυπτογράφησης με όνομα dec_inst ο οποίος είναι στιγμιότυπο του εξαρτήματος **decrypt_128**.

3.2.5.1 Διαδικασίες κρυπτογράφησης

Το εξάρτημα όπως αναφέρθηκε και προηγουμένως έχει δύο διαδικασίες για την κρυπτογράφηση δεδομένων. Η πρώτη ονομάζεται uart_encode_proc και είναι υπεύθυνη για την κρυπτογράφηση δεδομένων τηλεμετρίας μέσω του εξαρτήματος uart_enc_inst.Η δεύτερη ονομάζεται cam_encode_proc και είναι υπεύθυνη για την κρυπτογράφηση δεδομένων βίντεο μέσω του εξαρτήματος cam_enc_inst. Προκειμένου να ελέγξουν τα εξαρτήματά τους οι διαδικασίες αυτές χρησιμοποιούν δύο μηχανές πεπερασμένων καταστάσεων των οποίων οι καταστάσεις ονομάζονται wait_for_start,wait_for_aes,hold και transmit. Οι πρώτες τρεις καταστάσεις είναι ίδιες για την κάθε διαδικασία με μόνη διαφορά τα ονόματα των σημάτων που συνδέονται με τους πυρήνες κρυπτογράφησης. Η τελευταία κατάσταση είναι διαφορετική καθώς στην περίπτωση της κρυπτογράφησης δεδομένων βίντεο τα δεδομένα αποστέλλονται στην έξοδο του fpga ενώ στην περίπτωση των δεδομένων τηλεμετρίας τα δεδομένα αποστέλλονται στο εξάρτημα uart transmitter buffer προκειμένου να αποθηκευτούν σε μία FIFO για να σταλθούν στον uart tranciever μέσο UART. Πριν από κάθε κατάσταση το εξάρτημα θα ελέγξει την τιμή του σήματος i_reset, αν η τιμή του είναι ίση με 1 τότε θα κάνει επανεκκίνηση και οι διαδικασίες θα επιστρέψουν στην αρχική τους κατάσταση. Παρακάτω παρουσιάζονται αναλυτικά οι καταστάσεις για τις δύο διαδικασίες.

cam_encode_proc :

1. **wait_for_start** : Η αρχική κατάσταση της διαδικασίας. Όσο βρίσκεται σε αυτήν η διαδικασία cam_encode_proc κάνει τις τιμές των σημάτων εξόδου o_cam_aes_busy και o_camera_data_ready ίσες με 0 και ελέγχει την τιμή του σήματος εισόδου i_camera_data_ready. Αν η τιμή του είναι ίση με 1 τότε οι τιμές των σημάτων o_cam_aes_busy και cam_aes_start γίνονται ίσες με 1 και η κατάσταση αλλάζει στην **hold**.

2. **hold** : Σε αυτήν η διαδικασία περιμένει για ένα κύκλο ρολογιού προκειμένου τα δεδομένα προς κρυπτογράφηση να ληφθούν από το εξάρτημα cam_enc_inst και το εξάρτημα μεταφέρεται στην κατάσταση **wait_for_aes**.

3. **wait_for_aes** : Σε αυτήν την κατάσταση η διαδικασία θα ελέγξει την τιμή του σήματος cam_aes_done το ο οποίο προέρχεται από το εξάρτημα cam_enc_inst. Αν η τιμή του είναι ίση με 1 τότε η κρυπτογράφηση των δεδομένων βίντεο έχει τελειώσει και η κατάσταση αλλάζει στην **transmit**.

4. **transmit** : Σε αυτήν την κατάσταση η διαδικασία θα στείλει στο διάνυσμα εξόδου o_camera_encrypted_data τα κρυπτογραφημένα δεδομένα τηλεμετρίας που έχουν αποθηκευτεί στο τοπικό διάνυσμα cam_aes_output. Τα δεδομένα θα σταλθούν σε δύο κύκλους ρολογιού όπου σε κάθε κύκλο θα αποστέλλονται 64 bit ενώ παράλληλα το σήμα εξόδου o_camera_data_ready γίνεται ίσο με 1 προκειμένου να καταλάβει ο tranceiver πως υπάρχουν διαθέσιμα δεδομένα βίντεο. Για να επιτευχθεί αυτό χρησιμοποιείται ένας μετρητής με όνομα counter ο οποίος αυξάνεται κατά 1 σε κάθε κύκλο ρολογιού έως ότου φτάσει στην τιμή 1. Με το πέρας αυτού η διαδικασία θα μηδενίσει τον μετρητή, η τιμή του σήματος o_camera_data_ready θα γίνει ίση με το 0 και η διαδικασία θα μεταβεί στην κατάσταση **wait_for_start**.

uart_encode_proc :

1. **wait_for_start** : Η αρχική κατάσταση της διαδικασίας. Όσο βρίσκεται σε αυτήν η διαδικασία uart_encode_proc κάνει τις τιμές των σημάτων εξόδου o_uart_aes_busy και o_uart_data_ready ίσες με 0 και ελέγχει την τιμή του σήματος εισόδου i_ uart_data_ready. Αν η τιμή του είναι ίση με 1 τότε οι τιμές των σημάτων o_ uart _aes_busy και uart _aes_start γίνονται ίσες με 1 και η κατάσταση αλλάζει στην **hold**.

2. **hold** : Σε αυτήν η διαδικασία περιμένει για ένα κύκλο ρολογιού προκειμένου τα δεδομένα προς κρυπτογράφηση να ληφθούν από το εξάρτημα uart_enc_inst και το εξάρτημα μεταφέρεται στην κατάσταση **wait_for_aes**.

3. **wait_for_aes** : Σε αυτήν την κατάσταση η διαδικασία θα ελέγξει την τιμή του σήματος uart _aes_done το οποίο προέρχεται από το εξάρτημα uart_enc_inst. Αν η τιμή του είναι ίση με 1 τότε η κρυπτογράφηση των δεδομένων βίντεο έχει τελειώσει, η τιμή του σήματος εξόδου o_uart_data_ready γίνεται ίση με 1, τα κρυπτογραφημένα δεδομένα τηλεμετρίας που βρίσκονται στο τοπικό διάνυσμα uart_aes_output αποθηκεύονται στο διάνυσμα εξόδου o_uart_encrypted_data και η διαδικασία επιστρέφει στην αρχική κατάσταση **wait_for_start**.

3.2.5.2 Διαδικασία αποκρυπτογράφησης

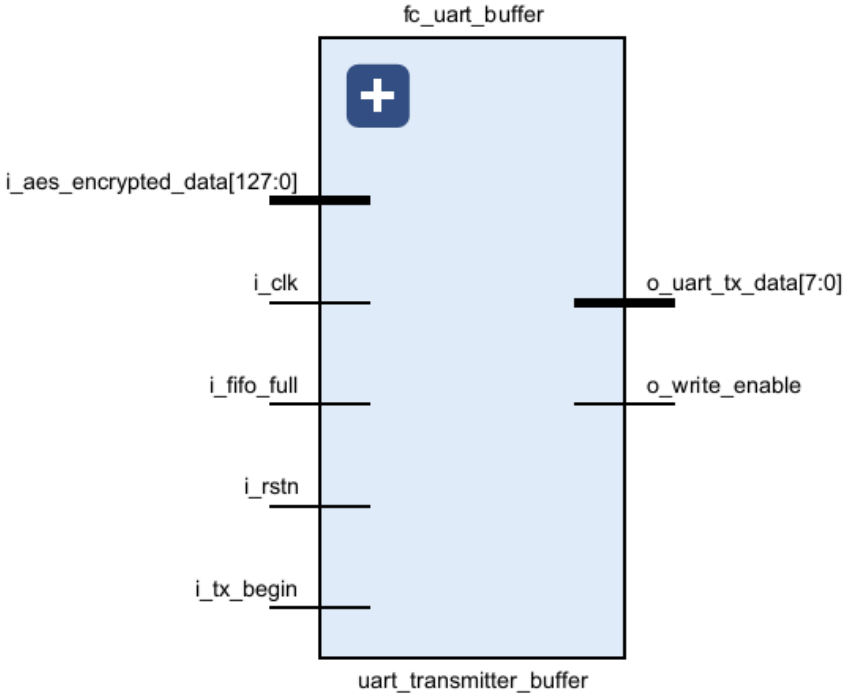
dec_inst :

1. **wait_for_start** : Η αρχική κατάσταση της διαδικασίας. Όσο βρίσκεται σε αυτήν την κατάσταση, η διαδικασία decode_proc κάνει τις τιμές των σημάτων εξόδου o_dec_busy και o_dec_rx_data_ready ίσες με 0 και ελέγχει την τιμή του σήματος εισόδου i_dec_start. Αν η τιμή του είναι ίση με 1 τότε οι τιμές των σημάτων o_dec_busy και decode_start γίνονται ίσες με 1 και η κατάσταση αλλάζει στην **hold**.

2. **hold** : Σε αυτήν η διαδικασία περιμένει για ένα κύκλο ρολογιού προκειμένου τα δεδομένα προς κρυπτογράφηση να ληφθούν από το εξάρτημα **uart_enc_inst** και το εξάρτημα μεταφέρεται στην κατάσταση **wait_for_aes**.

3. **wait_for_aes** : Σε αυτήν την κατάσταση η διαδικασία θα ελέγξει την τιμή του σήματος decode_done το οποίο προέρχεται από το εξάρτημα dec_inst. Αν η τιμή του είναι ίση με 1 τότε η αποκρυπτογράφηση των δεδομένων εντολών έχει τελειώσει, η τιμή του σήματος εξόδου o_dec_rx_data_ready γίνεται ίση με 1, τα αποκρυπτογραφημένα δεδομένα που περιέχονται στο τοπικό διάνυσμα decode_output αποθηκεύονται στο διάνυσμα εξόδου o_dec_output και η διαδικασία επιστρέφει στην αρχική κατάσταση **wait_for_start**.

3.2.6 UART Transmitter Buffer



Εικόνα 3.2.6.1 Εξάρτημα UART Transmitter Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_clk	Είσοδος	1	Ρολόι συχνότητας 250MHz.
i_rstn	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει επανεκκίνηση στην αρχική του κατάσταση.
i_tx_begin	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε υπάρχουν έτοιμα κρυπτογραφημένα δεδομένα τηλεμετρίας.
i_fifo_full	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO που συνδέεται με το εξάρτημα είναι γεμάτη.
i_aes_data	Είσοδος	128	Διάνυσμα εισόδου που περιέχει κρυπτογραφημένα/αποκρυπτογραφημένα δεδομένα.

Πίνακας 3.2.13 Είσοδοι UART Transmitter Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_write_enable	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα θα γράψει δεδομένα στην FIFO που συνδέεται με αυτό.
o_uart_tx_data	Έξοδος	8	Διάνυσμα εξόδου που περιέχει κρυπτογραφημένα/αποκρυπτογραφημένα δεδομένα.

Πίνακας 3.2.14 Έξοδοι UART Transmitter Buffer

Το εξάρτημα αυτό έχει δύο στιγμιότυπα , το ένα συνδέεται με την FIFO του UART TX στην πλευρά του ελεγκτή πτήσης και το άλλο με την FIFO του UART TX στην πλευρά του transceiver. Στην πρώτη περίπτωση δέχεται σαν είσοδο 128 bit αποκρυπτογραφημένων δεδομένων εντολής που έρχονται από τον σταθμό βάσης και επεξεργάζονται από το AES Encoder/Decoder και αφού τα λάβει τότε τα σπάει σε διανύσματα μήκους 8 bit τα οποία προωθεί στην FIFO που συνδέεται με το UART TX . Στην δεύτερη περίπτωση δέχεται στην είσοδο 128 bit κρυπτογραφημένων δεδομένων τηλεμετρίας που έχουν έρθει από τον ελεγκτή πτήσης και επεξεργαστεί από το AES Encoder/Decoder και τα στέλνει με τον ίδιο τρόπο όπως και το πρώτο στιγμιότυπο στην FIFO που συνδέεται στο UART TX στην πλευρά του transceiver.

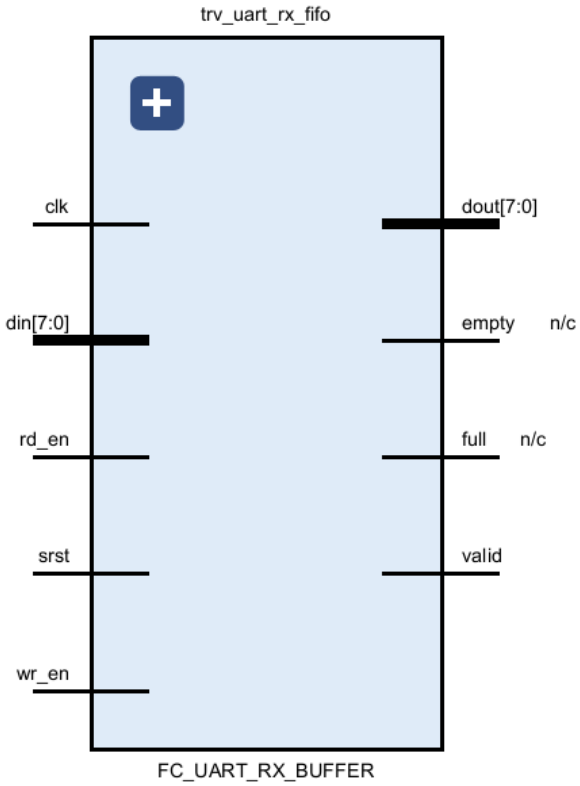
Η λειτουργία του εξαρτήματος αυτού βασίζεται πάνω σε μια διαδικασία που ενεργοποιείται σε κάθε θετικό παλμό του ρολογιού εισόδου i_clk. Μέσα σε αυτήν υπάρχει μία μηχανή πεπερασμένων καταστάσεων τριών καταστάσεων με ονόματα reset , wait_for_data και fill_fifo.

1. **reset** : Σε αυτή την κατάσταση το εξάρτημα μηδενίζει κάθε σήμα και αδειάζει κάθε διάνυσμα. Έπειτα το εξάρτημα μεταβαίνει στην κατάσταση **wait_for_data.**

2. **wait_for_data** : Σε αυτήν την κατάσταση το εξάρτημα περιμένει τιμή του σήματος εισόδου i_tx_begin να γίνει ίση με 1. Σε αυτήν την περίπτωση σημαίνει ότι έχουν κρυπτογραφηθεί επιτυχώς τα δεδομένα τηλεμετρίας από το εξάρτημα aes encoder/decoder ή έχουν αποκρυπτογραφηθέι επιτυχώς τα δεδομένα εντολών από τον σταθμό βάσης και το εξάρτημα θα μεταβεί στην κατάσταση **fill_fifo** αφού πρώτα αποθηκεύσει τα κρυπτογραφημένα/αποκρυπτογραφημένα δεδομένα τηλεμετρίας/εντολών που έχει λάβει στο διάνυσμα εισόδου i_aes_data σε ένα τοπικό διάνυσμα με όνομα tx_data_buffer.

3. **fill_fifo** : Σε αυτήν την κατάσταση το εξάρτημα θα στείλει σταδιακά, 8 bit την ανά κύκλο ρολογιού, τα δεδομένα που έχει λάβει στο διάνυσμα εισόδου i_aes_data στην FIFO. Η αποστολή των δεδομένων αυτών γίνεται με την εξής διαδικασία : Αρχικά ελέγχεται η τιμή του σήματος εισόδου i_fifo_full. Αν η τιμή του είναι 0 τότε σημαίνει ότι η FIFO δεν είναι γεμάτη οπότε μπορεί να αποθηκεύσει δεδομένα, αν όμως είναι ίση με 1 τότε το εξάρτημα θα περιμένει σε αυτήν την κατάσταση έως ότου γίνει 0. Στην συνέχεια το εξάρτημα θα κάνει την τιμή του σήματος εξόδου o_write_enable ίση με 1 προκειμένου η FIFO να μπορεί να δεχτεί δεδομένα και θα ελέγξει την τιμή μιας μεταβλητής με όνομα tx_index. Η μεταβλητή αυτή χρησιμοποιείται προκειμένου να διαβαστούν τα περιεχόμενα του διανύσματος tx_data_buffer, μέσω αυτής παίρνουμε 8 bit ανα κύκλο από το tx_data_buffer διαβάζοντας από αυτό 8 θέσεις ξεκινώντας από την θέση με νούμερο ίδιο με το tx_index μέχρι και την θέση tx_index-7, κάθε φορά που γίνεται αυτό η μεταβλητή αυξάνεται κατά 8 έως ότου η τιμή της γίνει μεγαλύτερη από το 127 .Όταν συμβεί αυτό το εξάρτημα θα κάνει την τιμή του σήματος εξόδου o_write_enable ίση με 0 προκειμένου η FIFO να μην δεχτεί δεδομένα, θα αλλάξει την τιμή της tx_index και θα την κάνει ίση με 7 και θα μεταβεί στην κατάσταση **wait_for_data**.

3.2.7 Μνήμες FIFO



Εικόνα 3.2.7.1 Παράδειγμα εξαρτήματος FIFO με όνομα FC UART RX Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
srst	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει επανεκκίνηση.
clk	Είσοδος	1	Ρολόι συχνότητας 250MHz.
din	Είσοδος	8	Διάνυσμα δεδομένων προς αποθήκευση.
wr_en	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα μπορεί να διαβάσει τα δεδομένα προς αποθήκευση.
rd_en	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα μπορεί να στείλει στην έξοδό του τα αποθηκευμένα δεδομένα του.

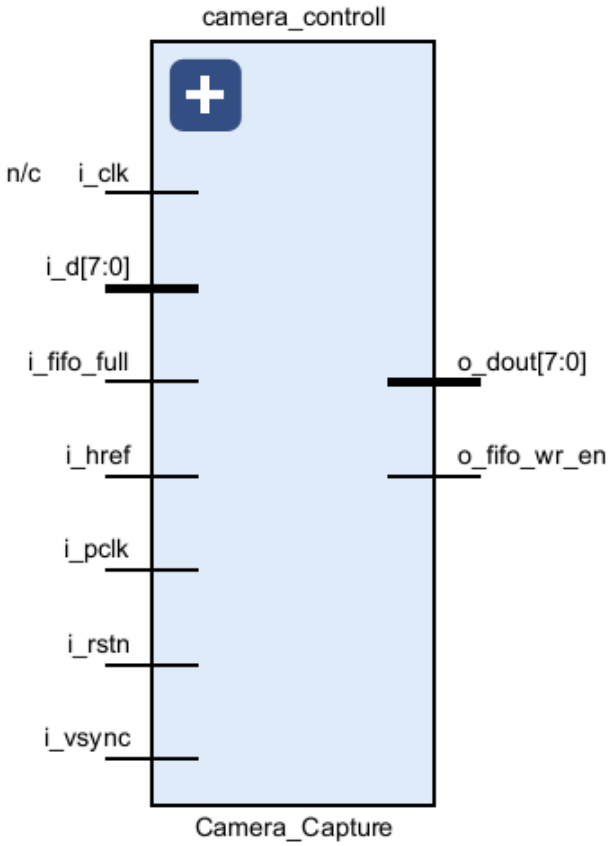
Πίνακας 3.2.15 Είσοδοι FIFO

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
dout	Έξοδος	8	Διάνυσμα εξόδου αποθηκευμένων δεδομένων.
full	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν μπορούν να αποθηκευτούν καινούργια δεδομένα.
empty	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν υπάρχουν αποθηκευμένα δεδομένα.
valid	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε μπορούν να διαβαστούν αποθηκευμένα δεδομένα.

Πίνακας 3.2.16 Έξοδοι FIFO

Στο σύστημα υπάρχουν συνολικά τέσσερις FIFO. Η κάθε FIFO συνδέεται είτε με το UART RX είτε με το UART TX και λειτουργούν σαν buffers για τα δεδομένα προς λήψη ή αποστολή. Το πλάτος της κάθε λέξης είναι ίσο με 8 bit και το βάθος τους είναι ίσο με το μέγεθος της μεγαλύτερης εντολής που μπορεί να στείλει ο ελεγκτής πτήσης ή ο σταθμός βάσης. Οι FIFO που συνδέονται με το UART RX δέχονται σαν είσοδο 8 bit που προέρχονται από το UART RX και τα στέλνουν στην έξοδό τους η οποία συνδέεται με το Plaintext Generator και οι FIFO που συνδέονται με το UART TX έχουν στην είσοδο τους δεδομένα που προέρχονται από το UART Buffer και τα στέλνουν στο UART TX. Το εξάρτημα αυτό και ο κώδικας που το συνοδεύει είναι έτοιμα από την Xilinx[26].

3.2.8 Camera Capture



Εικόνα 3.2.8.1 Εξάρτημα Camera Capture

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_clk	Είσοδος	1	Ρολόι συχνότητας 250MHz.
i_rstn	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει επανεκκίνηση.
i_pclk	Είσοδος	1	Ρολόι από την κάμερα συχνότητας 24MHz.
i_vsync	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε υπάρχει νέα εικόνα.
i_href	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε υπάρχουν διαθέσιμα δεδομένα εικόνας.
i_d	Είσοδος	8	Διάνυσμα δεδομένων εικόνας.
i_fifo_full	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε δεν αποστέλλονται δεδομένα εικόνας στην FIFO που συνδέεται με το εξάρτημα.

Πίνακας 3.2.17 Είσοδοι Camera Capture

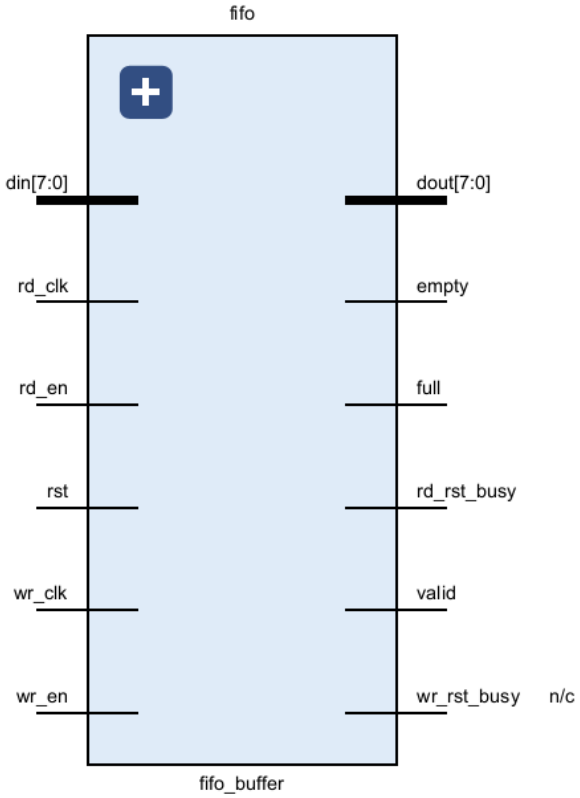
Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_fifo_wr_en	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε αποστέλλονται δεδομένα στην FIFO που συνδέεται το εξάρτημα.
o_dout	Έξοδος	8	Διάνυσμα δεδομένων εικόνας.

Πίνακας 3.2.18 Έξοδοι Camera Capture

Το εξάρτημα[12] αυτό δέχεται σαν είσοδο δύο σήματα ελέγχου (i_href και i_vsync), ένα διάνυσμα μήκους 8 bit και ένα σήμα ρολογιού από την κάμερα. Ανάλογα με τις τιμές των σημάτων ελέγχου το εξάρτημα θα αποθηκεύσει τα 8 bit που έλαβε από την κάμερα και θα τα προωθήσει σε μια FIFO. Η χρήση της FIFO είναι αναγκαία καθώς τα δεδομένα αυτά που θα σχηματίσουν ένα διάνυσμα μήκους 128 bit προκειμένου να κρυπτογραφηθούν από το εξάρτημα AES Encoder/Decoder βρίσκονται σε διαφορετικό clock domain από το AES Encoder/Decoder και συνεπώς έχουμε metastability. Με το να βάλουμε μια FIFO η οποία μεταφέρει τα bit από το ένα clock domain στο άλλο λύνουμε αυτό το πρόβλημα.

Η λειτουργία του εξαρτήματος αυτού περιλαμβάνεται μέσα σε μια διαδικασία που ενεργοποιείται σε κάθε θετικό παλμό του ρολογιού εισόδου i_pclk. Προκειμένου να λάβει δεδομένα από την κάμερα το εξάρτημα πρώτα θα διαβάσει τις τιμές των σημάτων εισόδου i_href,i_vsync και θα τις αποθηκεύσει στα τοπικά σήματα latched_href, latched_vsync κάθε φορά που πέφτει ο παλμός του ρολογιού. Στην συνέχεια αν latched_href = 1 ,latched_vsync = 0 και fifo_full = 0 τότε σημαίνει πως η FIFO που συνδέεται με το εξάρτημα μπορεί να λάβει δεδομένα και η κάμερα μπορεί να στείλει δεδομένα στο εξάρτημα, συνεπώς τότε το εξάρτημα θα μεταφέρει τα δεδομένα που βρίσκονται στο διάνυσμα εισόδου d στο διάνυσμα εξόδου o_dout και η τιμή του σήματος εξόδου o_fifo_wr_en θα γίνει ίση με 1 προκειμένου η FIFO να μπορέσει να διαβάσει και να αποθηκεύσει τις τιμές του o_dout. Αν latched_vsync = 1 τότε η κάμερα ετοιμάζεται να στείλει το επόμενο frame οπότε το εξάρτημα δεν λαμβάνει δεδομένα και η τιμή του o_fifo_wr_en γίνεται 0. Σε περίπτωση που το σήμα εισόδου i_rstn είναι ίσο με 0 το εξάρτημα θα κάνει επανεκκίνηση.

3.2.9 FIFO Buffer



Εικόνα 3.2.9.1 Εξάρτημα FIFO Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
rst	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει επανεκκίνηση.
wr_clk	Είσοδος	1	Ρολόι συχνότητας 24MHz
rd_clk	Είσοδος	1	Ρολόι συχνότητας 250MHz.
din	Είσοδος	8	Διάνυσμα δεδομένων εισόδου.
wr_en	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO μπορεί να αποθηκεύσει δεδομένα.
rd_en	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO μπορεί να γράψει στην έξοδό της δεδομένα.

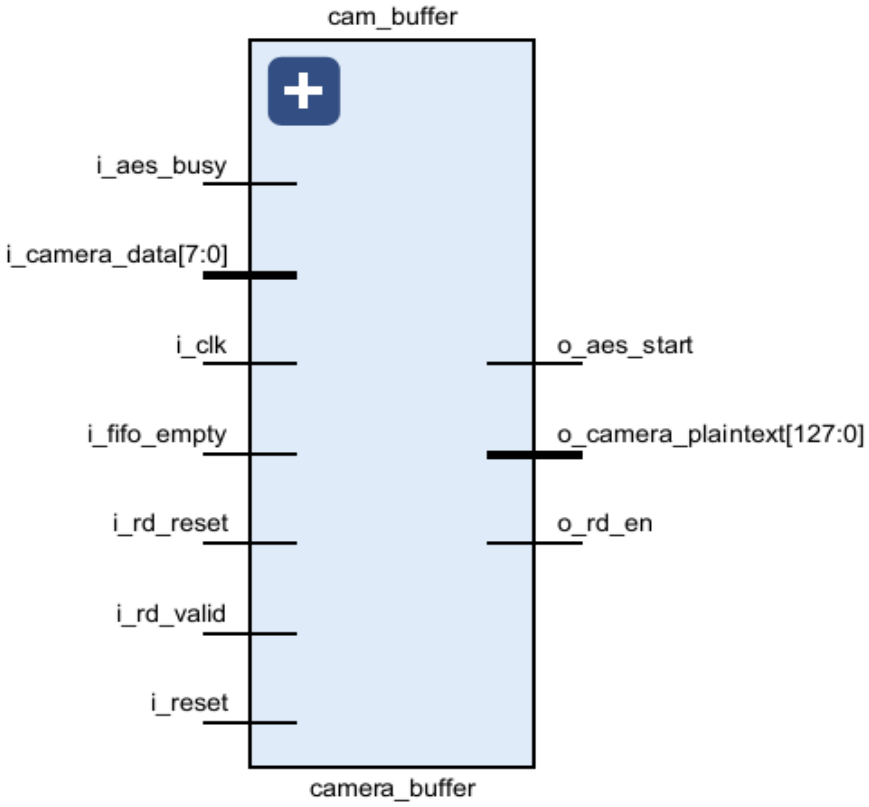
Πίνακας 3.2.19 Είσοδοι FIFO Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
dout	Έξοδος	8	Διάνυσμα δεδομένων εξόδου.
full	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO είναι γεμάτη και δεν μπορεί να δεχτεί δεδομένα.
empty	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO δεν μπορεί να στείλει στην έξοδό της δεδομένα.
valid	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO έχει δεδομένα στην έξοδό της.
wr_rst_busy	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO βρίσκεται σε κατάσταση επανεκκίνησης.
rd_rst_busy	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO βρίσκεται σε κατάσταση επανεκκίνησης.

Πίνακας 3.2.20 Έξοδοι FIFO Buffer

Η FIFO αυτή συνδέεται με το εξάρτημα Camera Capture στην είσοδό της και με το εξάρτημα Camera Buffer στην έξοδό της. Σκοπός της είναι να μπορέσουν να περάσουν τα δεδομένα από το clock domain του Camera Capture στο clock domain του Camera Buffer και του AES Encoder/Decoder χωρίς να υπάρχει η πιθανότητα να χαθούν δεδομένα λόγω των διαφορετικών ρολογιών(metastability). Το εξάρτημα αυτό και ο κώδικας που το συνοδεύει είναι έτοιμα από την Xilinx[26].

3.2.10 Camera Buffer



Εικόνα 3.2.10.1 Εξάρτημα Camera Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
i_clk	Είσοδος	1	Ρολόι συχνότητας 250MHz.
i_reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα κάνει επανεκκίνηση.
i_rd_reset	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO που συνδέεται με το εξάρτημα βρίσκεται σε κατάσταση επανεκκίνησης
i_rd_valid	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO που συνδέεται με το εξάρτημα μπορεί να του στείλει δεδομένα.
i_fifo_empty	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε η FIFO που συνδέεται με το εξάρτημα είναι άδεια.
i_camera_data	Είσοδος	8	Διάνυσμα δεδομένων εισόδου.
i_aes_busy	Είσοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα δεν στέλνει δεδομένα στην έξοδό του.

Πίνακας 3.2.21 Είσοδοι Camera Buffer

Όνομα pin	Τύπος	Μήκος (bits)	Περιγραφή
o_rd_en	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα διαβάζει δεδομένα από την FIFO που συνδέεται με αυτό.
o_camera_plaintext	Έξοδος	128	Διάνυσμα δεδομένων εξόδου.
o_aes_start	Έξοδος	1	Όταν η τιμή του είναι ίση με 1 τότε το εξάρτημα έχει διαθέσιμα δεδομένα προς κρυπτογράφηση.

Πίνακας 3.2.22 Έξοδοι Camera Buffer

Το εξάρτημα αυτό δέχεται στην είσοδό του τα δεδομένα από την FIFO που αναφέραμε προηγουμένως σαν ένα διάνυσμα 8 bit και φτιάχνει ένα διάνυσμα μήκους 128 bit προκειμένου να το στείλει στην έξοδό του που συνδέεται με το AES Encoder/Decoder για να το κρυπτογραφήσει. Επίσης δέχεται σαν είσοδο ένα σήμα από το AES Encoder/Decoder με όνομα i_aes_busy, σκοπός αυτού είναι να περιμένει το εξάρτημα προτού στείλει τα δεδομένα σε περίπτωση που γίνεται ήδη μια άλλη κρυπτογράφηση στα δεδομένα τηλεμετρίας.

Η λειτουργία του εξαρτήματος αυτού περιέχεται μέσα σε μία διαδικασία η οποία αποτελείται από μία μηχανή πεπερασμένων καταστάσεων τεσσάρων καταστάσεων με ονόματα start,fill_buffer και send_data.

1. **start** : Σε αυτήν την κατάσταση το εξάρτημα θα περιμένει έως ότου η τιμή του σήματος εισόδου i_rd_reset γίνει ίση με 0, όταν γίνει αυτό τότε θα αλλάξει την τιμή του σήματος εξόδου o_rd_en σε 1 προκειμένου να μπορέσει να διαβάσει δεδομένα από την FIFO που συνδέεται με αυτό και θα μεταβεί στην κατάσταση **fill_buffer**.

2. **fill_buffer** :Σε αυτή την κατάσταση το εξάρτημα θα κάνει την τιμή του σήματος εξόδου o_aes_start ίση με 0 προκειμένου να βεβαιώσει πως ο AES δεν θα ξεκινήσει ακόμα την κρυπτογραφία και στην συνέχεια θα ελέγξει την τιμή του σήματος εισόδου i_rd_valid αν είναι ίση με 1, στην περίπτωση που είναι τότε έχουν σταλθεί δεδομένα από την FIFO στο διάνυσμα εισόδου i_camera_data και θα αποθηκευτούν στο τοπικό διάνυσμα data_buffer. Προκειμένου να αποθηκευτούν στην κατάλληλη θέση του διανύσματος αυτού θα χρησιμοποιηθεί μια μεταβλητή με όνομα buffer_index έτσι ώστε κάθε φορά που υπάρχουν διαθέσιμα δεδομένα να αποθηκεύονται στις θέσεις buffer_index εως buffer_index-7,με το πέρας αυτής της διαδικασίας η τιμή του Buffer_index θα αυξηθεί κατά 8.Όταν η τιμή του buffer_index γίνει μεγαλύτερη από 127 τότε τα δεδομένα του data_buffer θα σταλθούν στο διάνυσμα εξόδου o_camera_plaintext , η τιμή του buffer_index θα γίνει ίση με 15 ,θα αποθηκευτούν στις πρώτες 8 θέσεις του data_buffer οι τιμές που βρίσκονται στην είσοδο i_camera_data και το εξάρτημα θα μεταβεί στην κατάσταση **send_data**.

3. **send_data** :Σε αυτήν την κατάσταση το εξάρτημα θα ελέγξει την τιμή του σήματος εισόδου i_aes_busy. Αν είναι ίση με 1 τότε ο AES Encoder/Decoder εκτελεί κρυπτογράφηση οπότε το εξάρτημα θα πρέπει να περιμένει έως ότου γίνει 0. Αν είναι 0 τότε θα γίνει η τιμή του σήματος εξόδου o_aes_start ίση με 1 προκειμένου να ξεκινήσει η διαδικασία της κρυπτογράφησης και το εξάρτημα θα μεταβεί στην κατάσταση **fill_buffer**.

4. Μεθοδολογία

Σε αυτό το κεφάλαιο θα παρουσιάσουμε το/τα λογισμικά που χρησιμοποιήθηκαν για τον σχεδιασμό και την επαλήθευση του συστήματος καθώς και τα βήματα/μεθοδολογία που ακολουθήσαμε προκειμένου να σχεδιάσουμε και να επαληθεύσουμε το σύστημά μας.

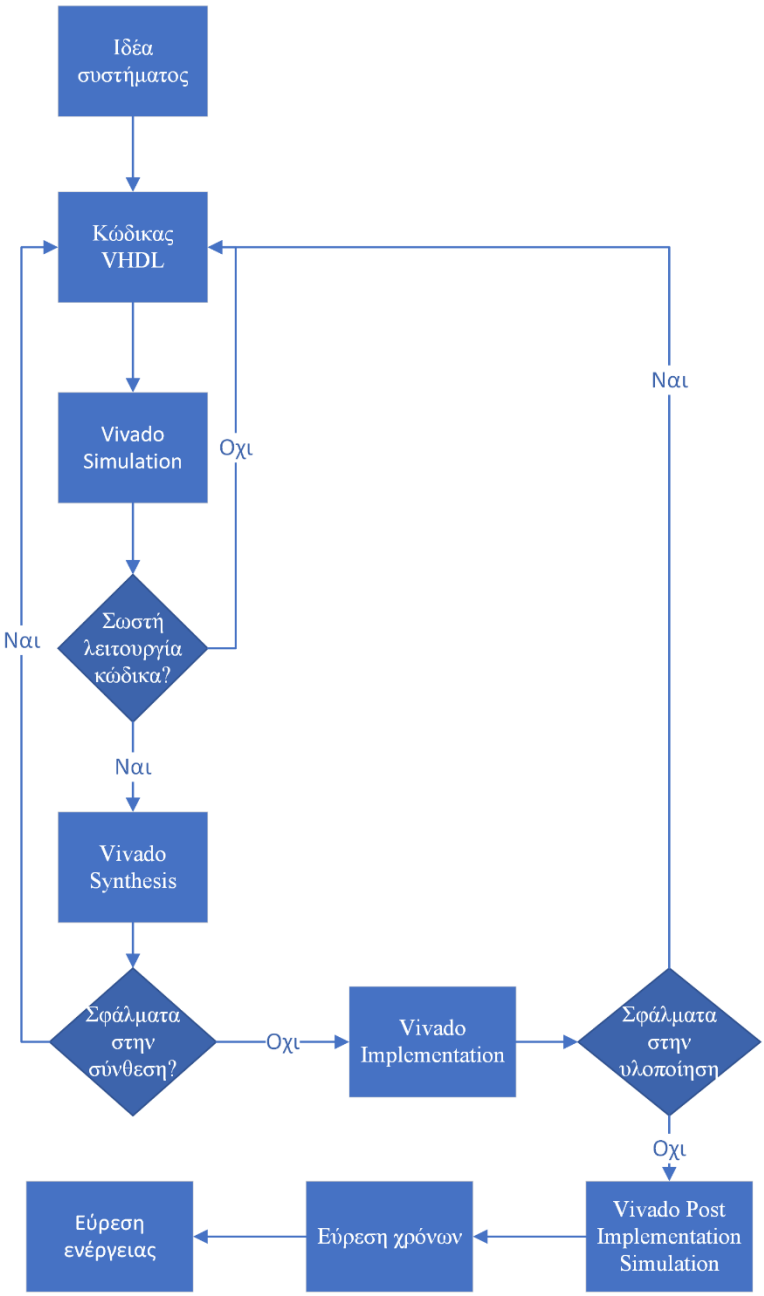
4.1 Λογισμικό

Για την υλοποίηση της παρούσας διπλωματικής επιλέχθηκε σαν λογισμικό το Vivado Design Suite της εταιρίας Xilinx[29].

Το Vivado αποτελεί ένα από τα πιο καινοτόμα λογισμικά για τον σχεδιασμό, υλοποίηση και επαλήθευση συστημάτων για FPGA's. Αξίζει να αναφέρουμε ότι για την υλοποίηση ενός συστήματος σε FPGA πέρα από τις γλώσσες περιγραφής υλικού όπως η VHDL και η Verilog μας παρέχεται από το vivado και η επιλογή να σχεδιαστεί με HLS(High Level Synthesis), δηλαδή να σχεδιαστεί με γλώσσες προγραμματισμού υψηλού επιπέδου όπως η C , C++ και System C .

4.2 Βήματα σχεδιασμού και επαλήθευσης

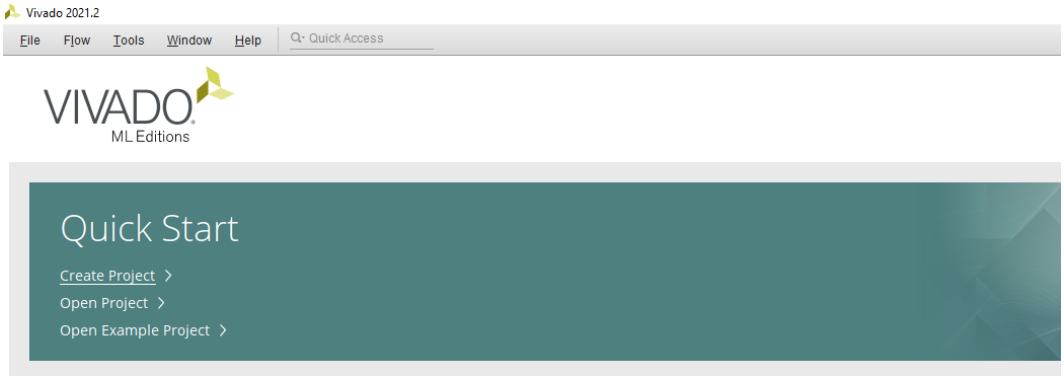
Στην εικόνα 4.2.1 παρουσιάζετε το διάγραμμα ροής για την κατασκευή, υλοποίηση και επαλήθευση του συστήματος.



Εικόνα 4.2.1 Διάγραμμα ροής σχεδιασμού συστήματος

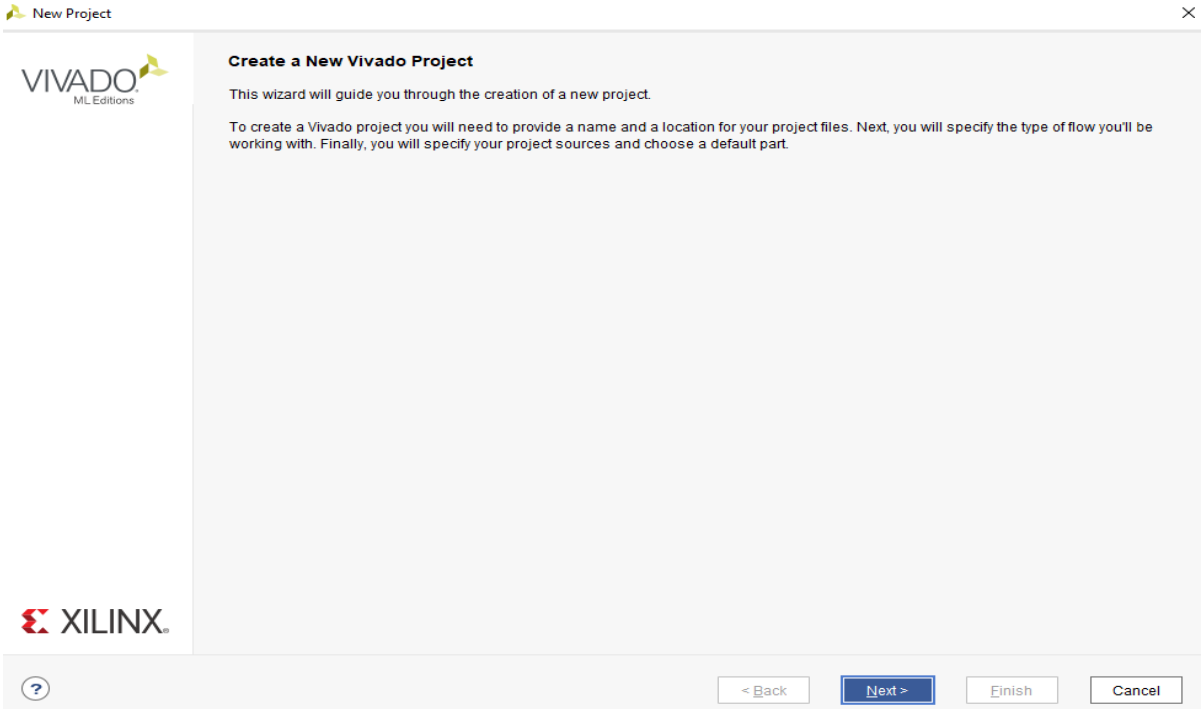
4.2.1 Δημιουργία project

Για να φτιάξουμε ένα project στο Vivado διαλέγουμε την επιλογή “Create Project ” στην αρχική σελίδα του Vivado Design Suite.

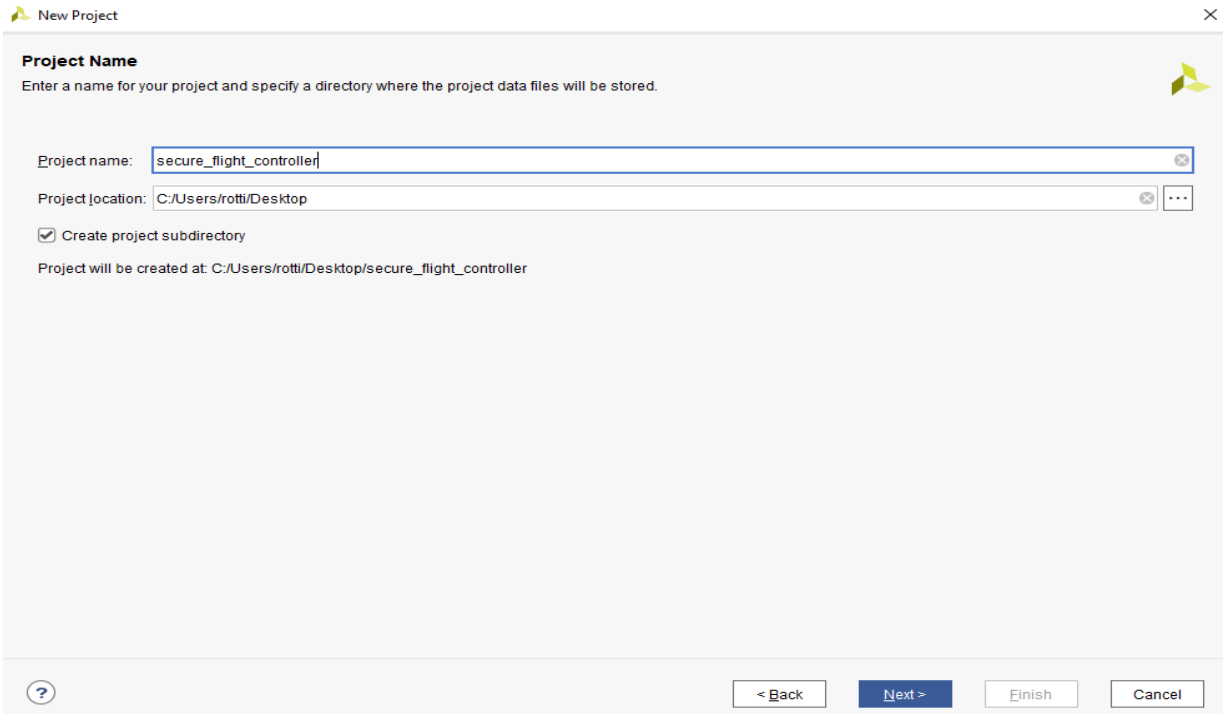


Εικόνα 4.2.1.1 Αρχικό παράθυρο του Vivado

Στην συνέχεια στο παράθυρο που θα εμφανιστεί επιλέγουμε “Next” και θα μεταβούμε σε ένα καινούργιο παράθυρο όπου θα πρέπει να πληκτρολογήσουμε το όνομα του project δίπλα από την ετικέτα με τίτλο “Project Name” και την τοποθεσία όπου θα αποθηκευτούν τα αρχεία του δίπλα από την ετικέτα με τίτλο “Project Location”.

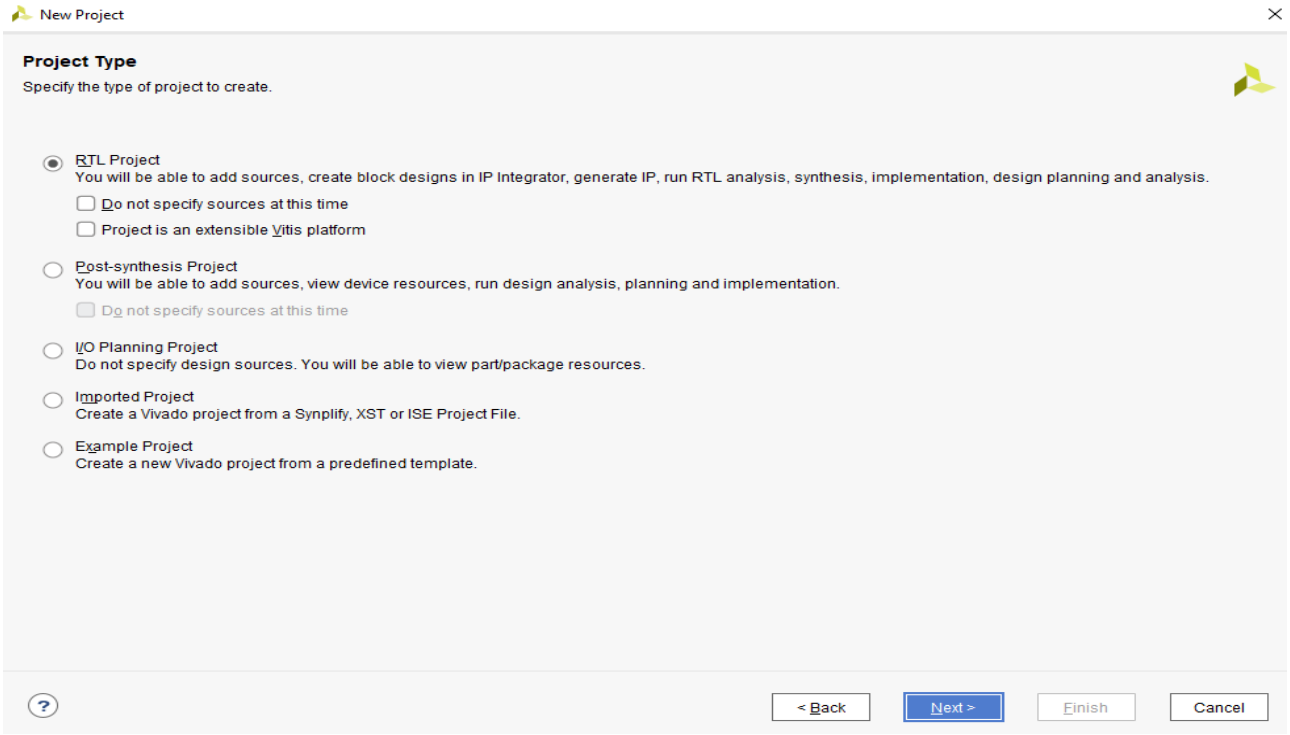


Εικόνα 4.2.1.2 Εισαγωγικό παράθυρο δημιουργίας project



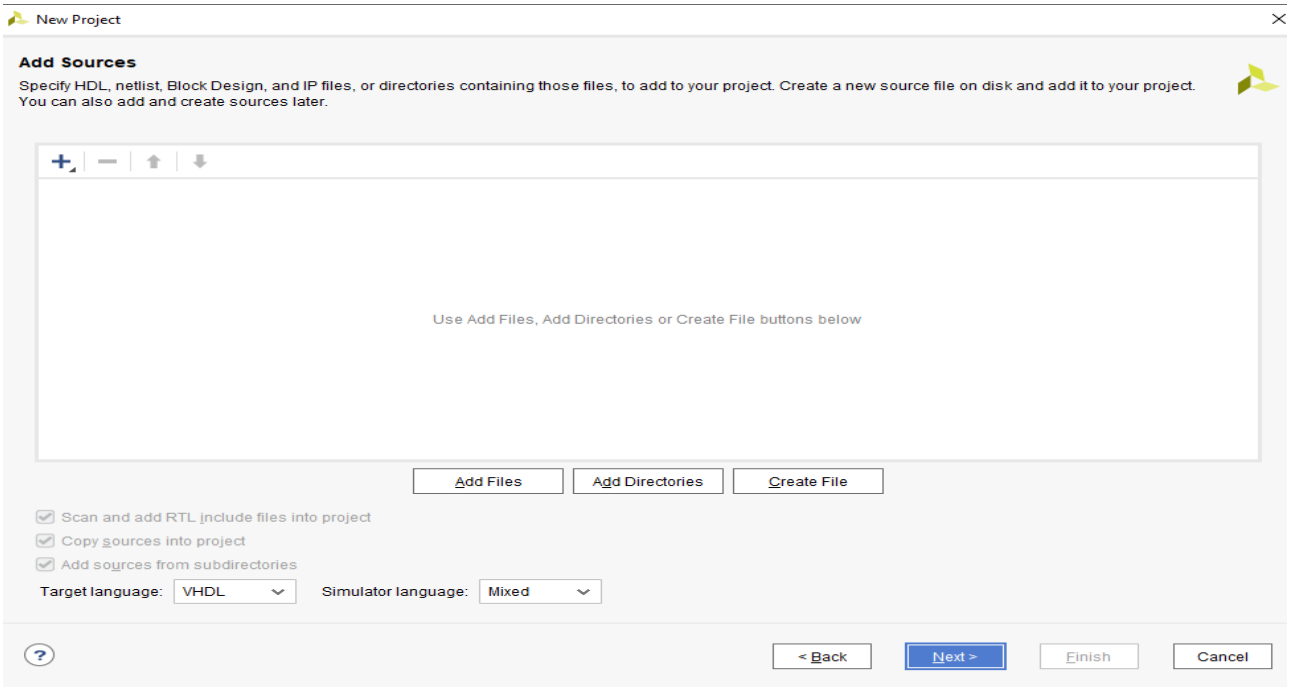
Εικόνα 4.2.1.3 Επιλογή ονόματος και τοποθεσίας αποθηκεύσεις

Αφού τελειώσουμε με την επιλογή ονόματος και τοποθεσίας αποθήκευσης πατάμε το κουμπί “Next” όπου και θα μεταφερθούμε σε ένα νέο παράθυρο προκειμένου να επιλέξουμε τον τύπο του project. Στην περίπτωσή μας θα επιλέξουμε την πρώτη επιλογή με όνομα “RTL Proj ect” και θα πατήσουμε “Next”.

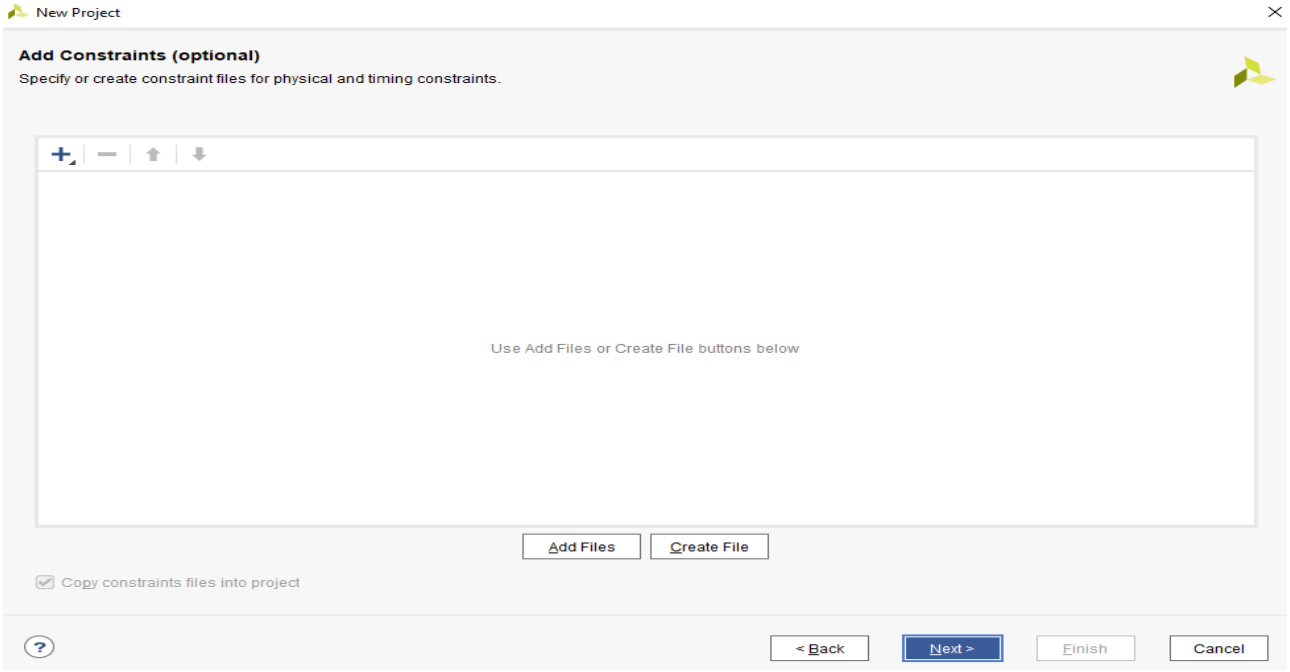


Εικόνα 4.2.1.4 Επιλογή τύπου project

Τα επόμενα δύο παράθυρα που θα εμφανιστούν μας προτρέπουν στο να εισάγουμε στο project “sources”, στην δικιά μας περίπτωση θα επιλέξουμε απλά την γλώσσα του project (VHDL) και θα τα προσπεράσουμε καθώς θα φτιάξουμε τα αρχεία μας με το πέρας της διαδικασίας δημιουργίας project. Σε περίπτωση που είχαμε αρχεία που μπορούν να φανούν χρήσιμα θα τα προσθέταμε πατώντας το πλήκτρο “Add Files” και επιλέγοντας τα από τον φάκελο που βρίσκονται. Σημειώνεται ότι το πρώτο παράθυρο αφορά τα αρχεία κώδικα του project ενώ το δεύτερο παράθυρο αφορά τα “constrains files” που είναι τα αρχεία για το πώς συνδέονται οι είσοδοι-έξοδοι του FPGA με τον κώδικά μας.

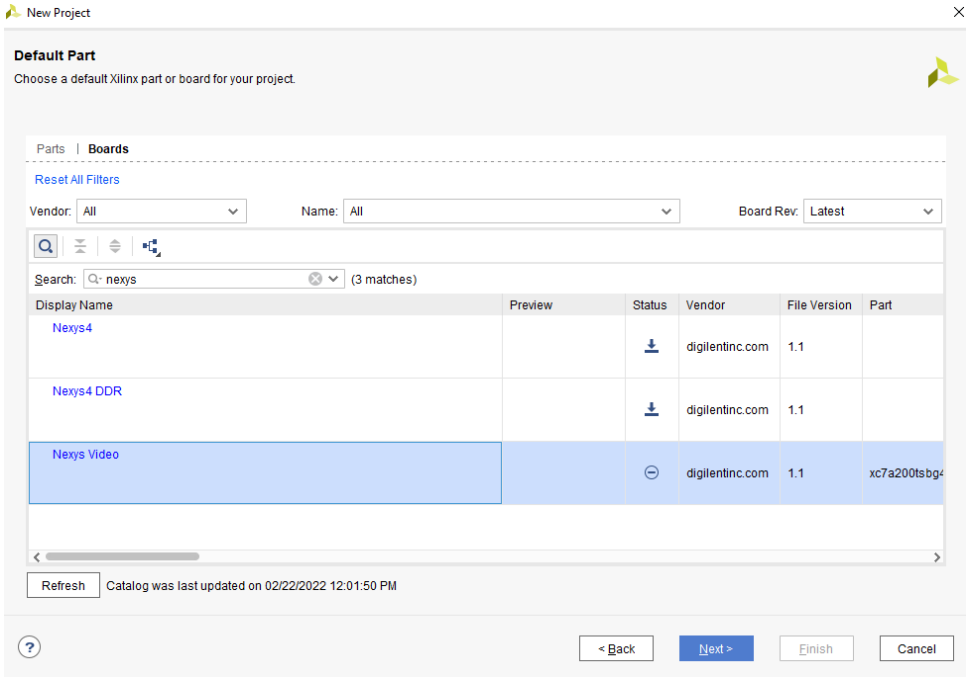


Εικόνα 4.2.1.5 Παράθυρο επιλογής και δημιουργίας αρχείων του project

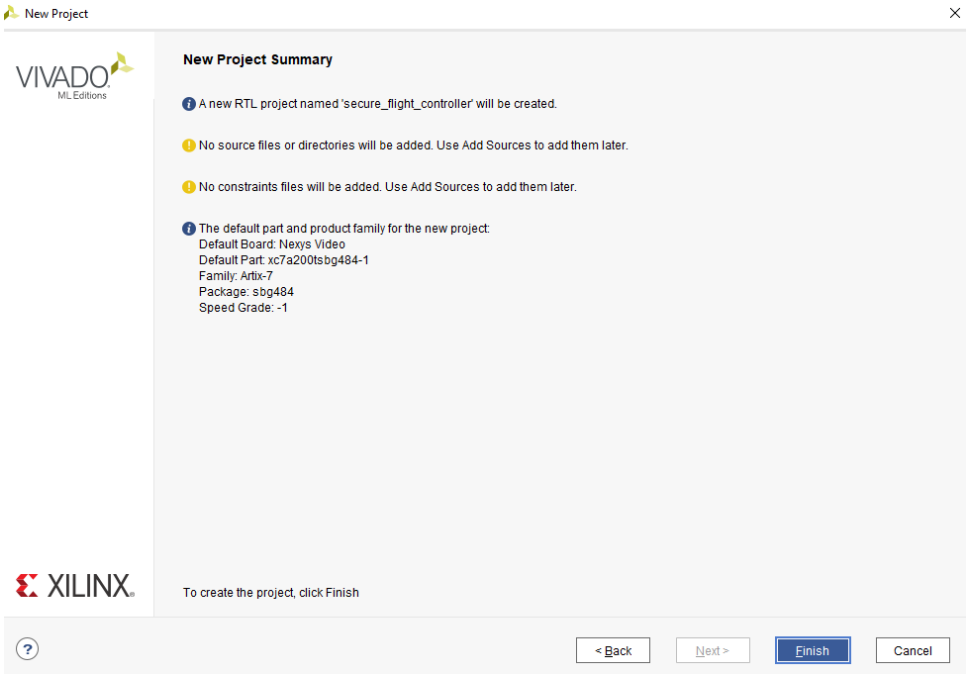


Εικόνα 4.2.1.6 Παράθυρο επιλογής και δημιουργίας αρχείου Contsraints

Στην συνέχεια, αφού περάσουμε τα δύο παράθυρα, πρέπει να επιλέξουμε το FPGA ή το board όπου θα ενσωματωθεί το project. Στην περίπτωσή μας επιλέγουμε το “Boards”(δίπλα από το “Parts”), πληκτρολογούμε στο πεδίο “search” nexys και επιλέγουμε το “Nexys Video”. Στην περίπτωσή μας είναι είδη κατεβασμένα τα αρχεία του, αν δεν ήταν τότε θα επιλέγαμε το εικονίδιο με το βέλος και την παύλα στο πεδίο status και θα το κατεβάζαμε. Έπειτα πατάμε “Next” και μεταφερόμαστε στο επόμενο παράθυρο το οποίο είναι και το τελευταίο. Εκεί βλέπουμε μία περίληψη του project και αφού βεβαιωθούμε ότι είναι OK πατάμε το κουμπί “Finish” .



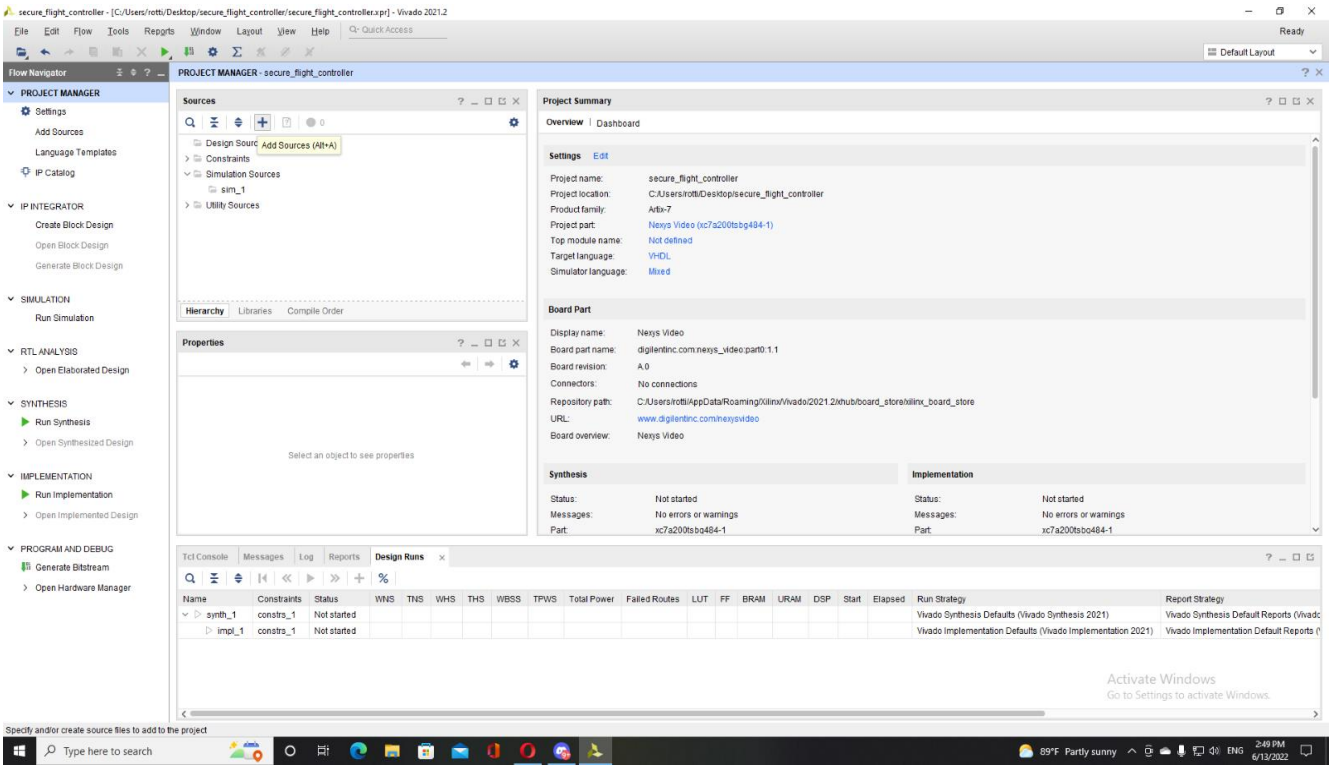
Εικόνα 4.2.1.7 Επιλογή FPGA



Εικόνα 4.2.1.8 Τελικό παράθυρο δημιουργίας project

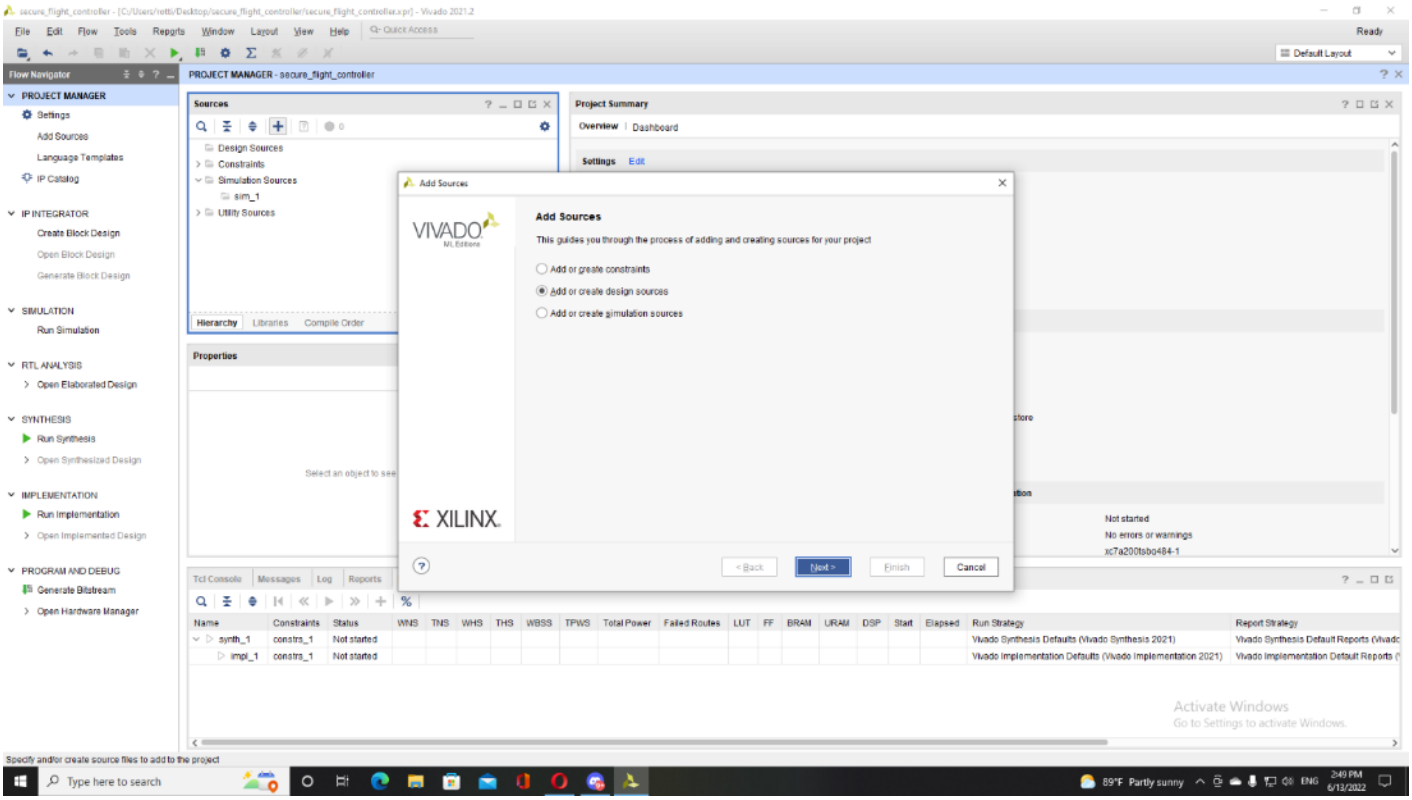
4.2.2 Δημιουργία πηγαίων αρχείων και αρχείων προσομοίωσης

Με το πέρας της διαδικασίας δημιουργίας project θα βρεθούμε στο παρακάτω παράθυρο. Εδώ είναι η αρχική σελίδα του project όπου μπορούμε να δούμε τα αρχεία που το απαρτίζουν καθώς και διάφορα στοιχεία όπως τι προβλήματα υπάρχουν, πόσο χώρο στο chip καταλαμβάνει ή αρχιτεκτονική μας και άλλα. Τώρα το συγκεκριμένο project είναι άδειο οπότε και κατευθυνόμαστε πάνω αριστερά στο σήμα “+” για να ξεκινήσουμε να προσθέτουμε αρχεία.



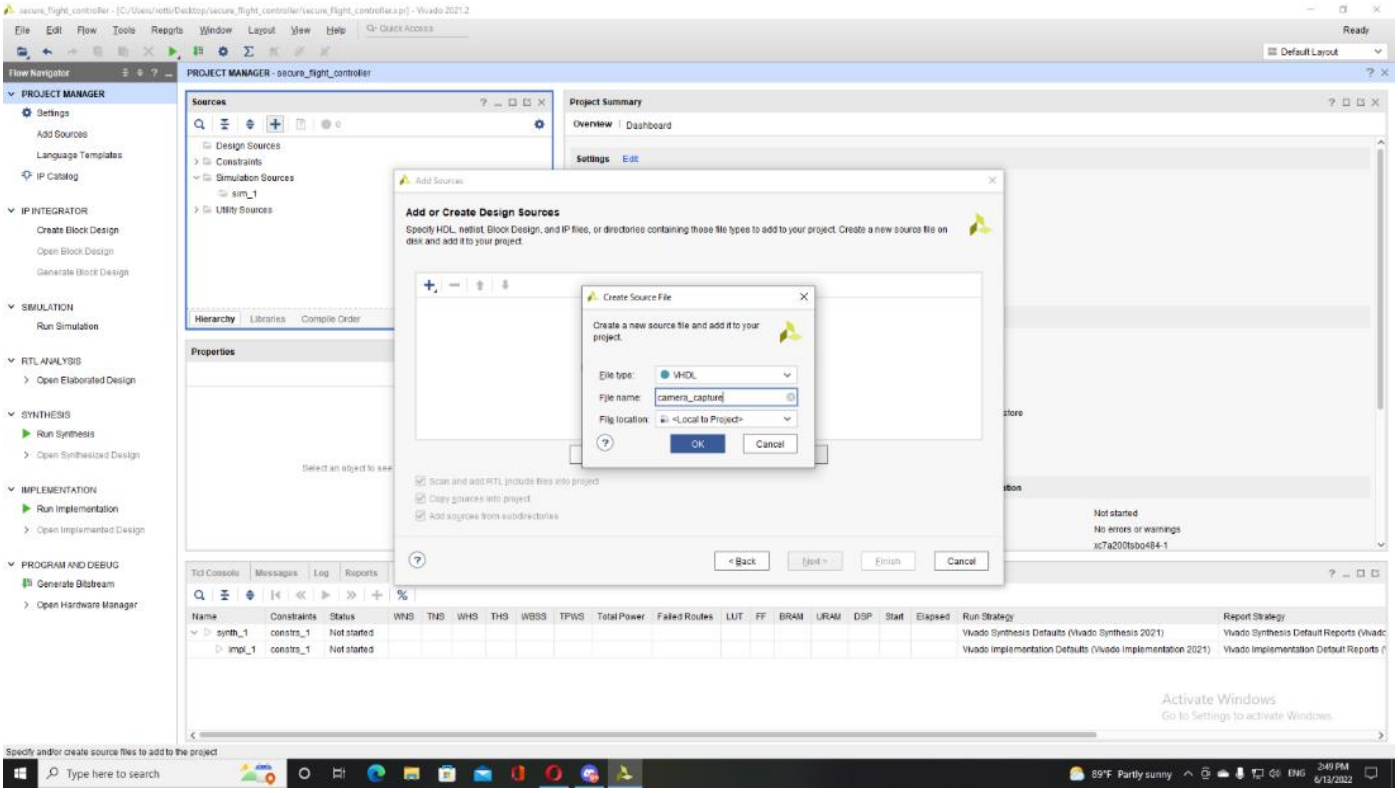
Εικόνα 4.2.2.1 Αρχικό παράθυρο του project

Στην συνέχεια στο νέο παράθυρο που άνοιξε θα διαλέξουμε την επιλογή με όνομα “Add or create design sources” και θα πατήσουμε το κουμπί “Next”.

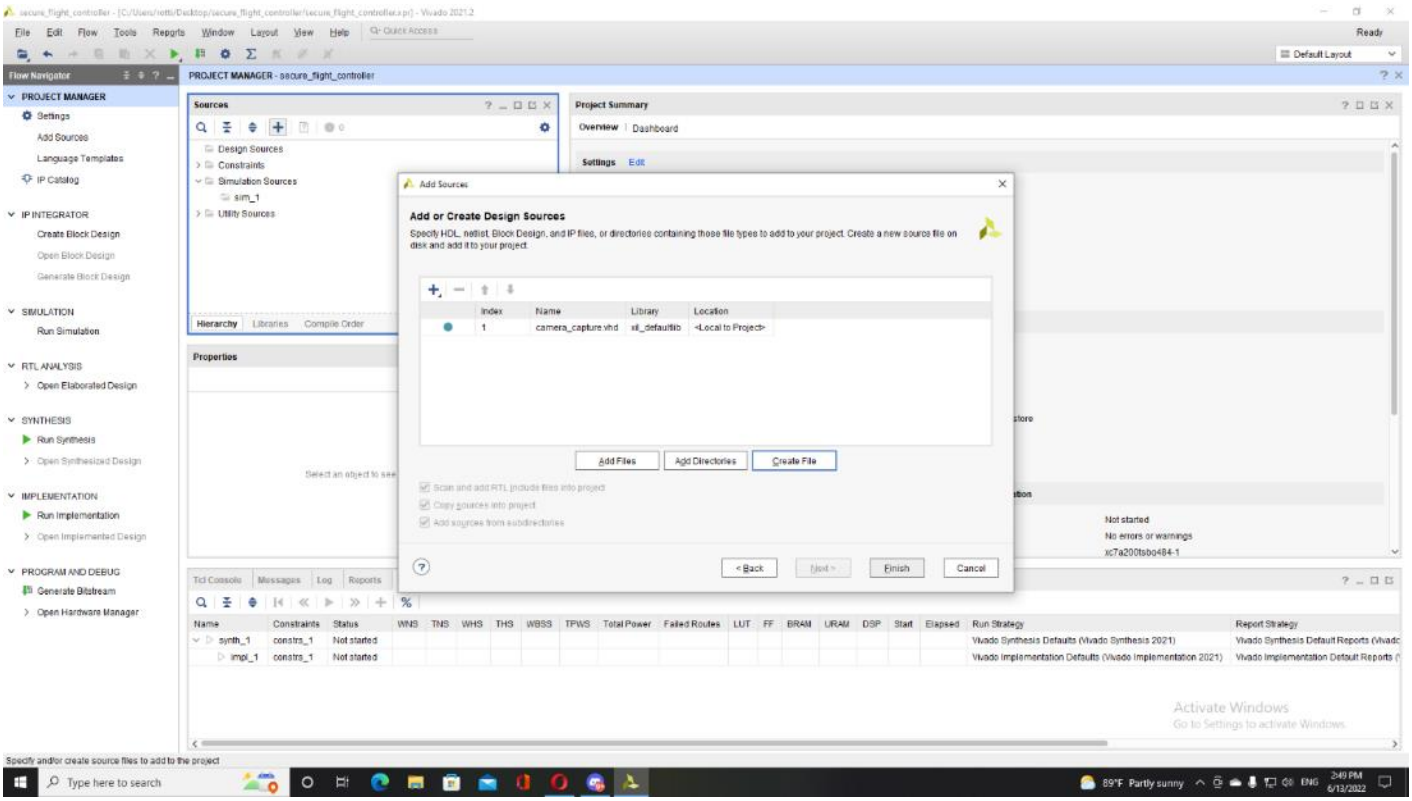


Εικόνα 4.2.2.2 Παράθυρο δημιουργίας αρχείων

Έπειτα ,στο επόμενο παράθυρο μας δίνεται η ικανότητα να προσθέσουμε αρχεία με τρεις διαφορετικούς τρόπους, “Add Files”,“Add Directories” και “Create File”. Στην δικιά μας περίπτωση θα επιλέξουμε το “Create File” καθώς δεν έχουμε κάποια έτοιμα αρχεία και θα μεταβούμε στο επόμενο παράθυρο όπου και θα μας ζητηθεί να ονομάσουμε το αρχείο που θέλουμε να φτιάξουμε, να επιλέξουμε τον τύπο του (VHDL,Verilog) καθώς και το πού θέλουμε να αποθηκευτεί. Αφού τελειώσουμε με αυτό θα πατήσουμε το κουμπί “Ok” και θα μεταφερθούμε πάλι στο αρχικό παράθυρο με τις επιλογές για δημιουργία αρχείων. Η διαδικασία αυτή θα επαναληφθεί έως ότου έχουμε προσθέσει στο project όλα τα αρχεία που θέλουμε.

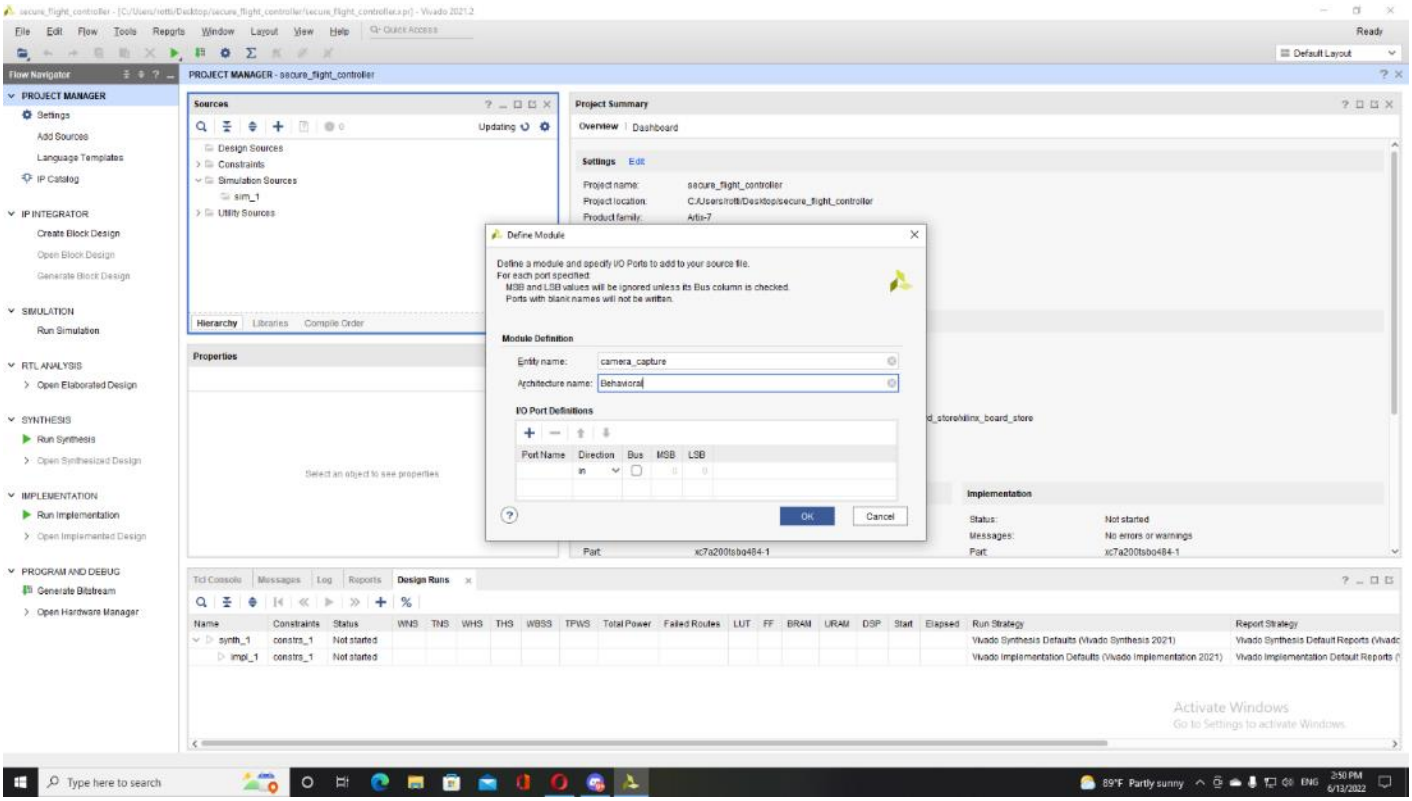


Εικόνα 4.2.2.3 Επιλογή ονόματος αρχείου

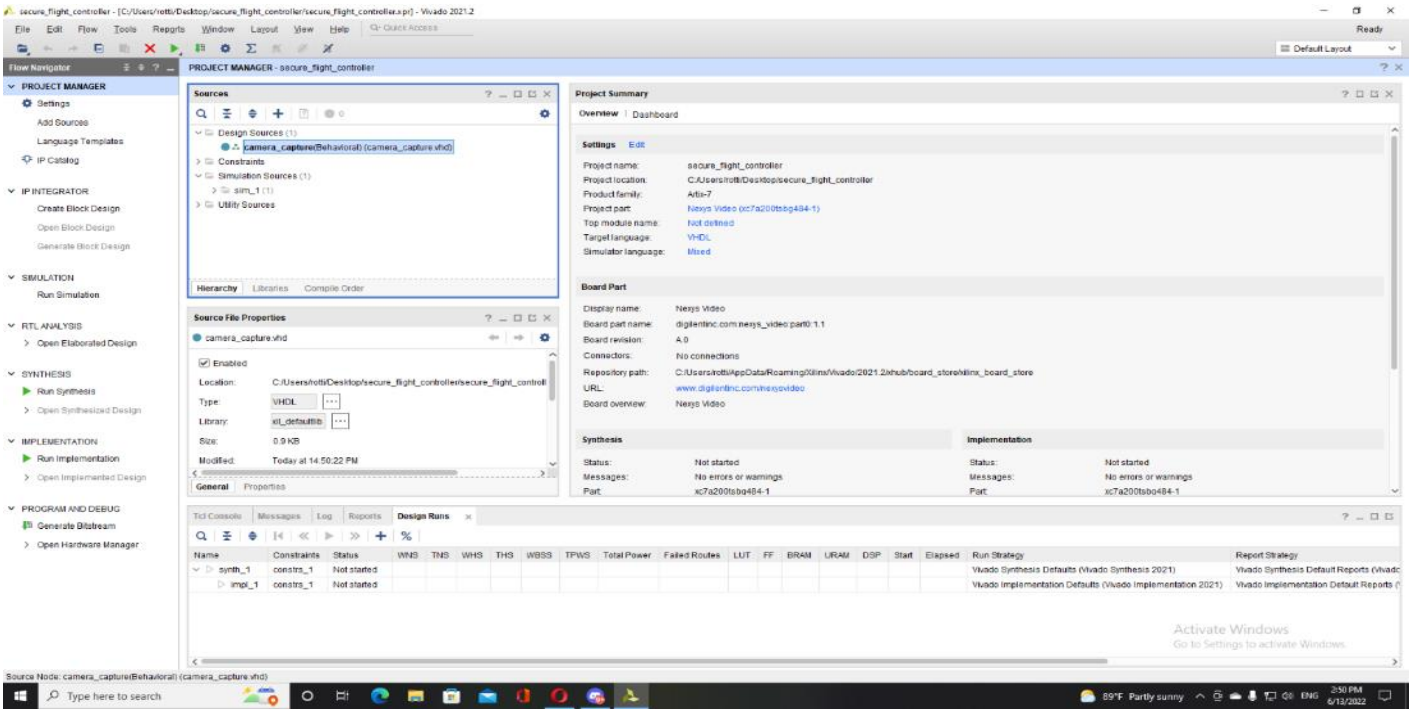


Εικόνα 4.2.2.4 Παράθυρο δημιουργίας αρχείων

Στην συνέχεια, αφού τελειώσουμε με την πρόσθεση αρχείων πατάμε το κουμπί “Finish” και θα μεταβούμε στο επόμενο παράθυρο όπου μας δίνεται η δυνατότητα να ονομάσουμε το εξάρτημα που φτιάξαμε, να ορίσουμε το όνομα της αρχιτεκτονικής του καθώς και να επιλέξουμε τις εισόδους/εξόδους του. Αφού επαναλάβουμε την διαδικασία αυτή για όλα τα αρχεία που φτιάξαμε θα πατήσουμε το κουμπί “Ok” και θα μεταβούμε στο αρχικό παράθυρο του project.



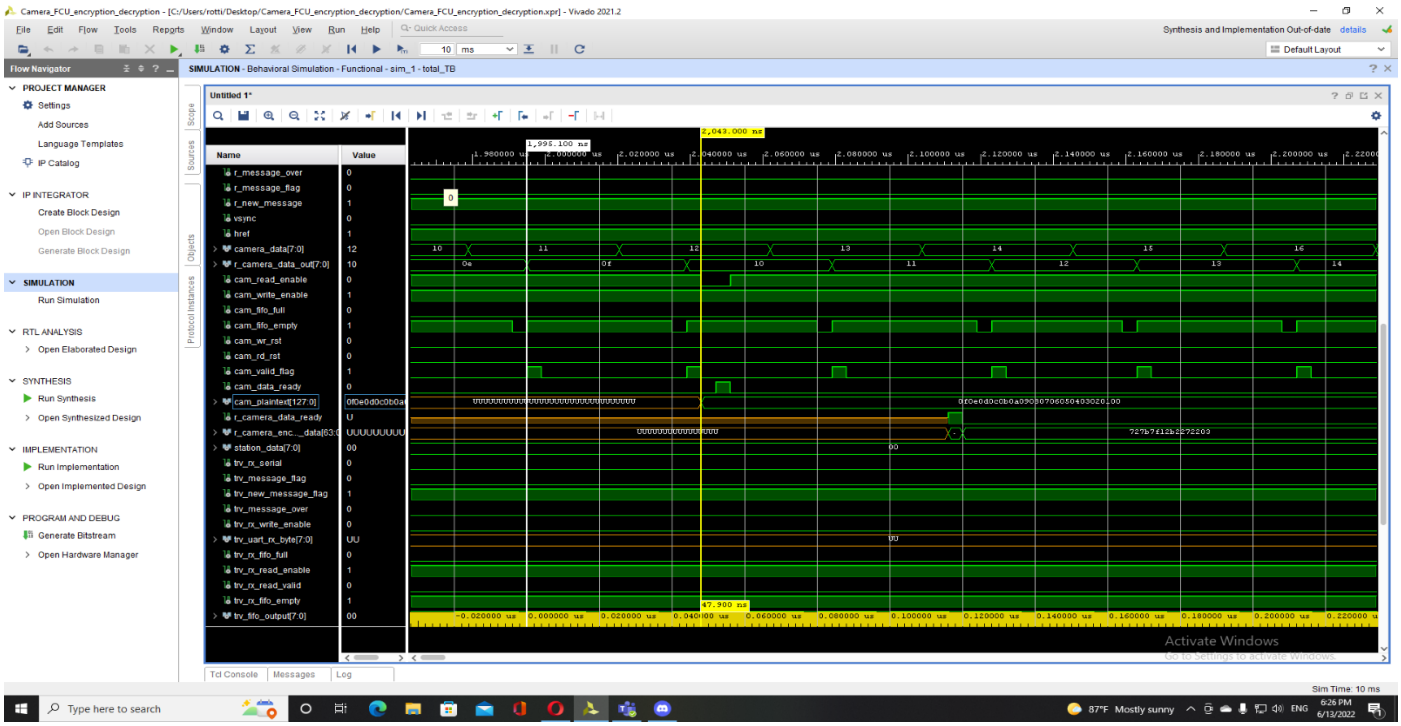
Εικόνα 4.2.2.5 Επιλογή ονόματος και αρχιτεκτονικής



Εικόνα 4.2.2.6 Αρχικό παράθυρο του project με πηγαία αρχεία

4.2.3 Διαδικασία προσομοίωσης ,επαλήθευσης και εκτέλεσης του project

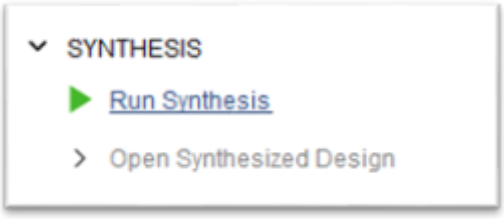
Με το πέρας των προηγουμένων διαδικασιών δημιουργίας project και αρχείων καθώς και με την ολοκλήρωση του κώδικα, φτάνουμε στο σημείο όπου πρέπει να προσομοιώσουμε των κώδικα προκειμένου να βεβαιωθούμε ότι δουλεύει καθώς και να τον φορτώσουμε στο fpga. Για να το κάνουμε αυτό πρέπει να φτιάξουμε ένα testbench αρχείο. Το αρχείο αυτό περιέχει μέσα μία αρχικοποίηση των εξαρτημάτων που έχουμε φτιάξει καθώς και συναρτήσεις χρόνου για το πότε θα ξεκινήσουν να δουλεύουν τα διάφορα εξαρτήματα αυτά. Όταν το φτιάξουμε τότε θα επιλέξουμε στο αρχικό παράθυρο του project την επιλογή “Run Simulation” και στο υποπαράθυρο που θα εμφανιστεί θα διαλέξουμε “Run behavioral simulation”. Τότε θα εμφανιστεί στην οθόνη μας ένα παράθυρο με τις διάφορες κυματομορφές των αρχείων του project των οποίων τα entities έχουν οριστεί στο testbench που φτιάξαμε.



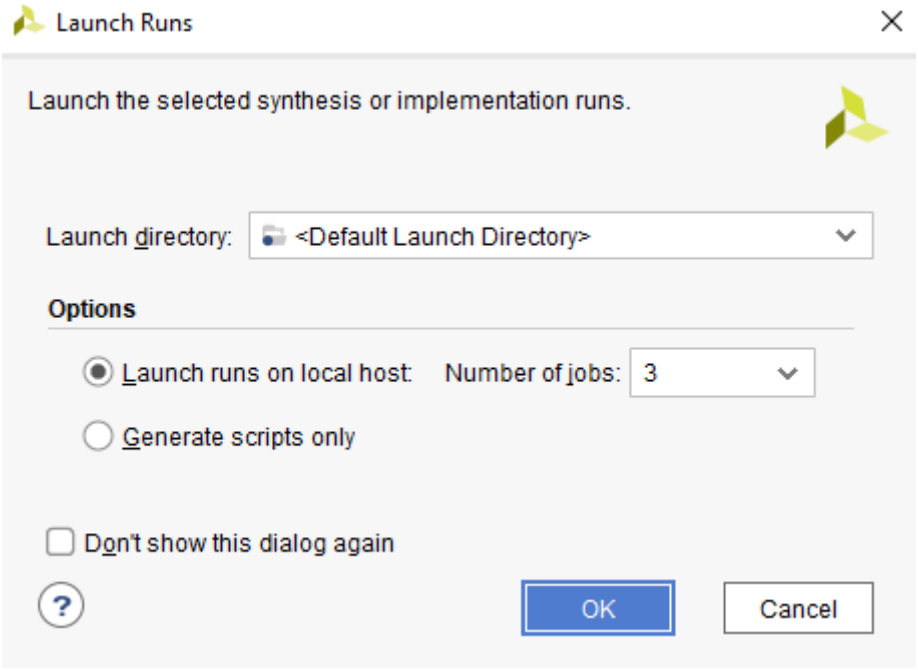
Εικόνα 4.2.3.1 Παράδειγμα προσομοίωσης

Αφού βεβαιωθούμε ότι τα αποτελέσματά μας είναι σωστά τότε θα πατήσουμε την επιλογή “Run synthesis” έτσι ώστε

να συνθέσει το Vivado το project προκειμένου να μπορέσει μετά να υλοποιηθεί πάνω στο FPGA.

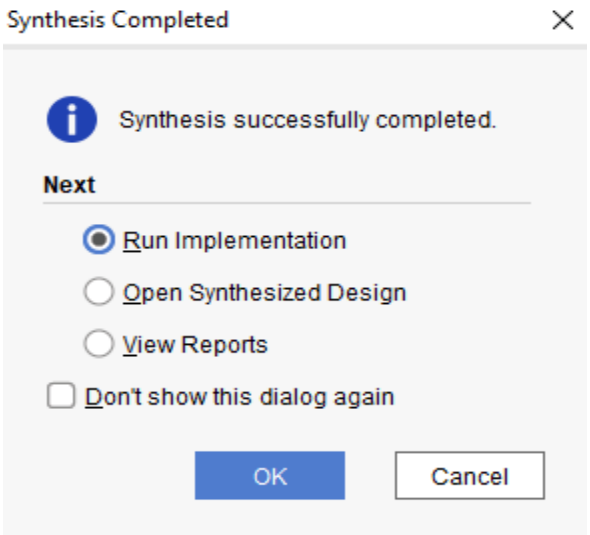


Εικόνα 4.2.3.2 Επιλογή σύνθεσης του project



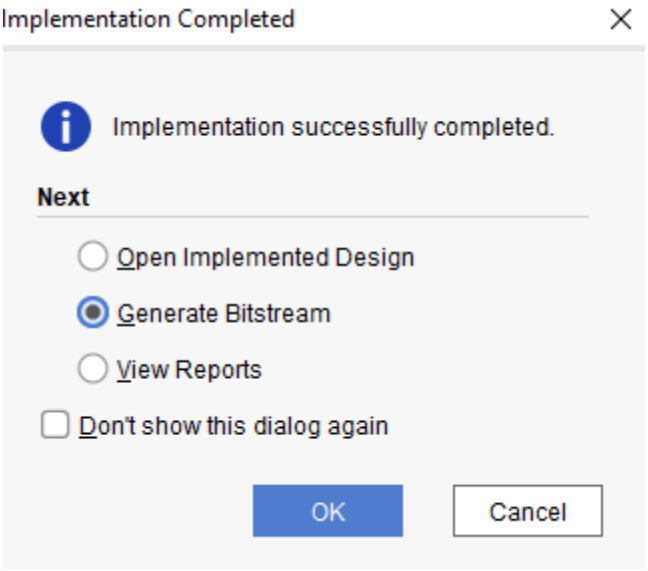
Εικόνα 4.2.3.3 Επιλογές σύνθεσης.

Όταν η σύνθεση τελειώσει επιτυχώς θα εμφανιστεί στην οθόνη το εξής παράθυρο.



Εικόνα 4.2.3.4 Επιλογή εκκίνησης υλοποίησης

Εδώ θα επιλέξουμε την πρώτη επιλογή με όνομα “Run Implementation” προκειμένου να ξεκινήσει η διαδικασία της υλοποίησης του project προκειμένου να φορτωθεί στο FPGA. Όταν τελειώσει επιτυχώς θα εμφανιστεί στην οθόνη το ακόλουθο παράθυρο.



Εικόνα 4.2.3.4 Τέλος διαδικασίας σύνθεσης και υλοποίησης

5. Αποτελέσματα

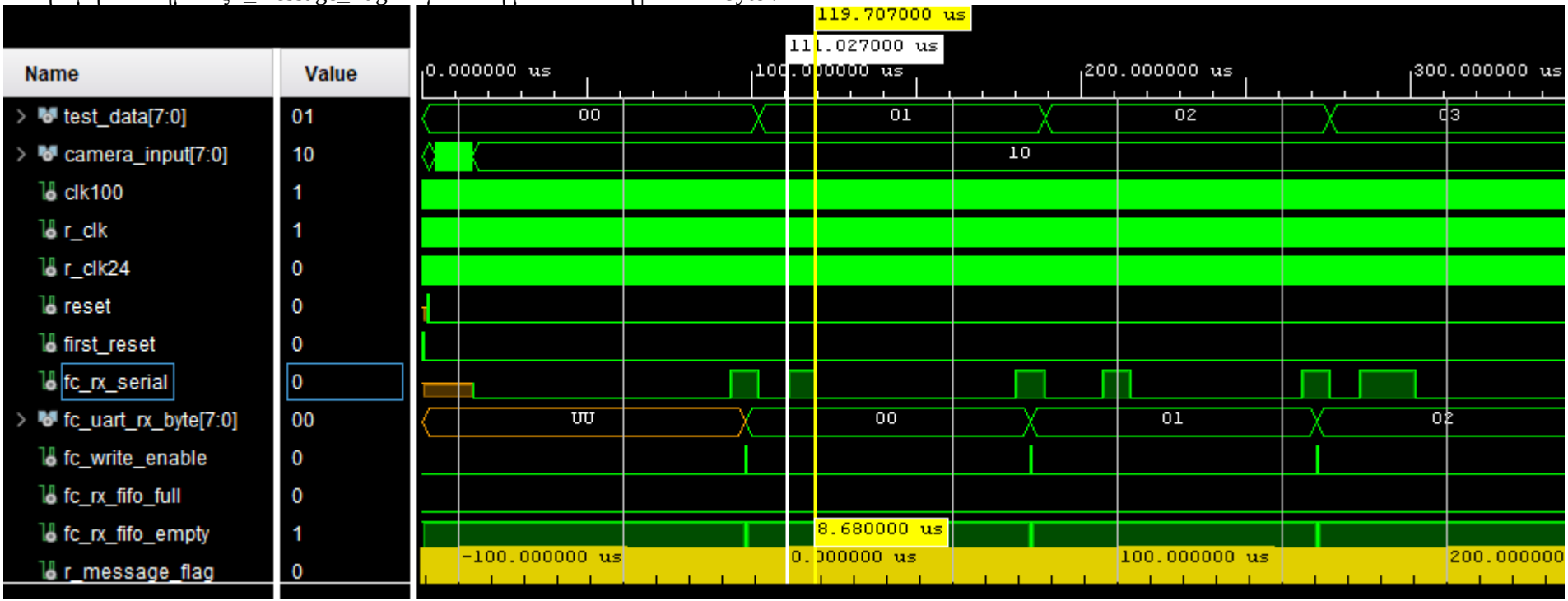
Στην παράγραφο αυτήν θα δούμε και θα αναλύσουμε τα αποτελέσματα της προσομοίωσης που λάβαμε από το vivado καθώς και τα αποτελέσματα για την κατανάλωση ενέργειας του project όταν ενσωματώνεται στο FPGA. Παρακάτω παρουσιάζεται ένας πίνακας με τα χαρακτηριστικά του εξαρτήματος κρυπτογράφησης/αποκρυπτογράφησης AES Encoder/Decoder. Τα χαρακτηριστικά αυτά είναι το κλειδί για την κρυπτογράφηση/αποκρυπτογράφηση δεδομένων τηλεμετρίας, το κλειδί για την κρυπτογράφηση δεδομένων βίντεο καθώς και το διάνυσμα αρχικοποίησης(IV, απαραίτητο για την λειτουργία του AES σε CBC mode).

Κλειδί Τηλεμετρίας	00000000000000000000000000000000
Κλειδί Βίντεο	2b7e151628aed2a6abf7158809cf4f3c
Διάνυσμα Αρχικοποίησης	ffffffffffffffffffffffffffffffffffffffffffffffff

Πίνακας 5.1 Κλειδιά AES και διάνυσμα αρχικοποίησης

5.1 Μετάδοση δεδομένων από τον ελεγκτή πτήσης στο FPGA μέσω UART (Εξάρτημα UART RX)

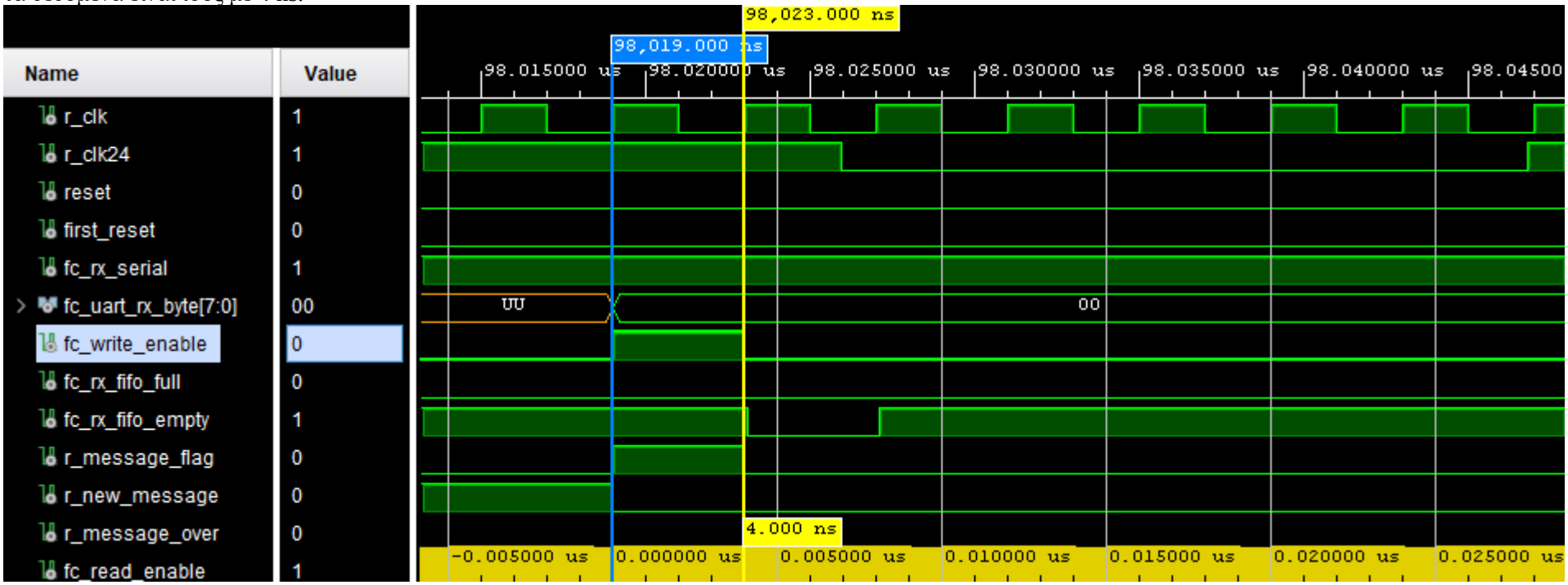
Στην εικόνα 5.1.1 παρουσιάζεται η μετάδοση των δεδομένων από τον ελεγκτή πτήσης στο fpga μέσο του UART. Παρατηρούμε ότι η λειτουργία είναι σωστή καθώς προκειμένου να ξεκινήσει η μετάδοση η γραμμή επικοινωνίας fc_rx_serial θα γίνει από ένα μηδέν προκειμένου να ξεκινήσει η μετάδοση δεδομένων μέσω UART. Ύστερα, ανάλογα με το byte που θα μεταδοθεί η τιμή της γραμμής δεδομένων θα γίνει 1 ή 0 για χρόνο ίσο με αυτόν που έχει εφαρμοστεί ανάλογα με το baudrate του UART,η διαδικασία αυτή θα πραγματοποιηθεί για το κάθε bit του byte δεδομένων δηλαδή για 8 φορές όπου και στο τέλος η τιμή της γραμμή δεδομένων θα γίνει ίση με 1 για να καταλάβει το FPGA ότι τελείωσαν τα δεδομένα. Για παράδειγμα στην εικόνα 5.1.1 όταν στέλνεται το byte με τιμή 01 αφού ξεκινήσει η επικοινωνία θα γίνει η τιμή της γραμμής δεδομένων ίση με 1 για 8680 ns και μετά θα παραμείνει 0 για τα υπόλοιπα bits. Με το πέρας της διαδικασίας αυτής ενημερώνεται η τιμή του διανύσματος fc_uart_rx_byte με την τιμή που λήφθηκε και η τιμή του σήματος fc_write_enable γίνεται ίση με 1 προκειμένου να γραφτούν τα δεδομένα στην μνήμη FIFO. Επίσης η τιμή του σήματος r_new_message θα γίνει ίση με 1 κατά την διάρκεια λήψης μηνύματος προκειμένου να καταλάβει το εξάρτημα Message Handler ότι ένα νέο μήνυμα έρχεται από τον ελεγκτή πτήσης και η τιμή του σήματος και η τιμή του σήματος r_message_flag θα γίνει ίση με 1 όταν ληφθεί ένα byte .



Εικόνα 5.1.1 Προσομοίωση εξαρτήματος UART RX .

5.2 Αποθήκευση δεδομένων τηλεμετρίας σε μνήμη (Εξάρτημα FIFO)

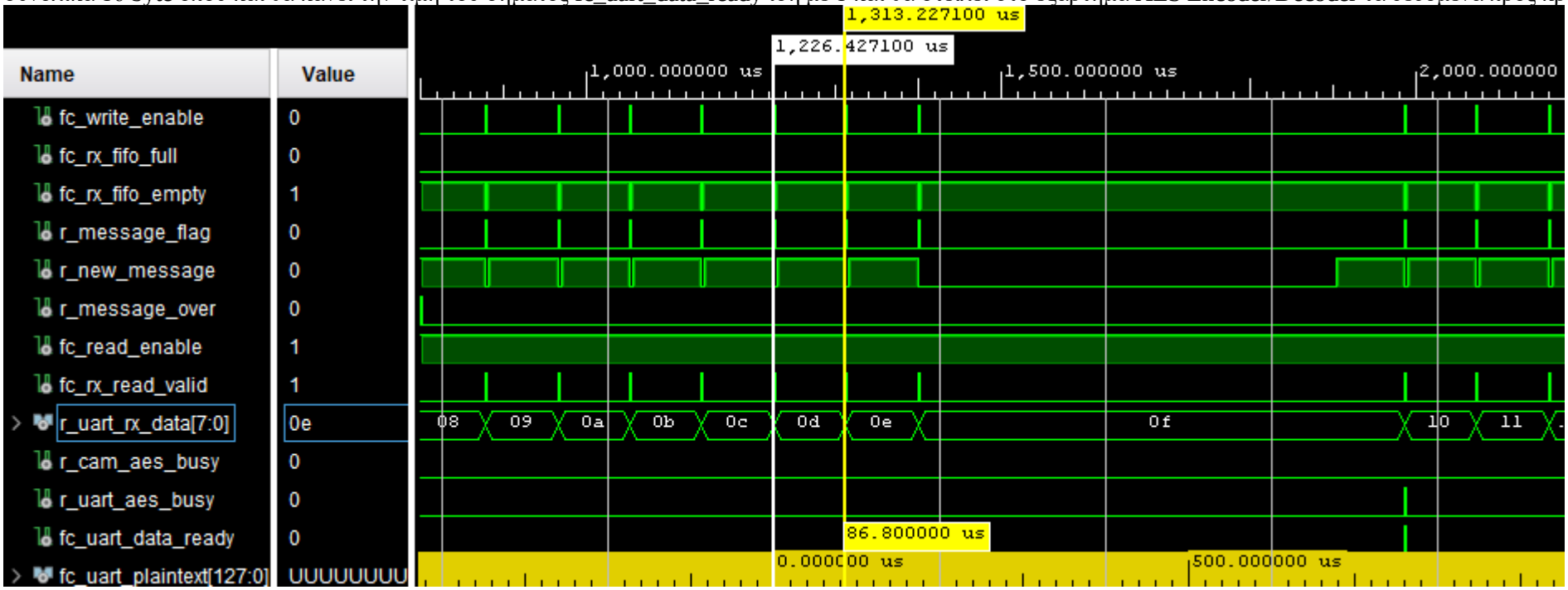
Στην εικόνα 5.2.1 παρουσιάζεται η προσομοίωση αποθήκευσης δεδομένων τηλεμετρίας στην FIFO που συνδέεται με το εξάρτημα UART RX. Όταν η τιμή του σήματος fc_write_enable γίνει ίση με 1 τότε θα ξεκινήσει η αποθήκευση δεδομένων. Με το που αποθηκευτεί το πρώτο byte η τιμή του σήματος fc_rx_fifo_empty θα γίνει ίση με 0 καθώς τώρα θα υπάρξουν δεδομένα στην FIFO, με το που βγουν από την FIFO αυτά τα δεδομένα η τιμή του θα ξαναγίνει ίση με 1. Ο χρόνος που χρειάζεται για να αποθηκευτούν τα δεδομένα είναι ίσος με 4 ns.



Εικόνα 5.2.1 Προσομοίωση εξαρτήματος FIFO .

5.3 Σχηματισμός μπλοκ δεδομένων τηλεμετρίας προς κρυπτογράφηση(Εξάρτημα Plaintext Generator)

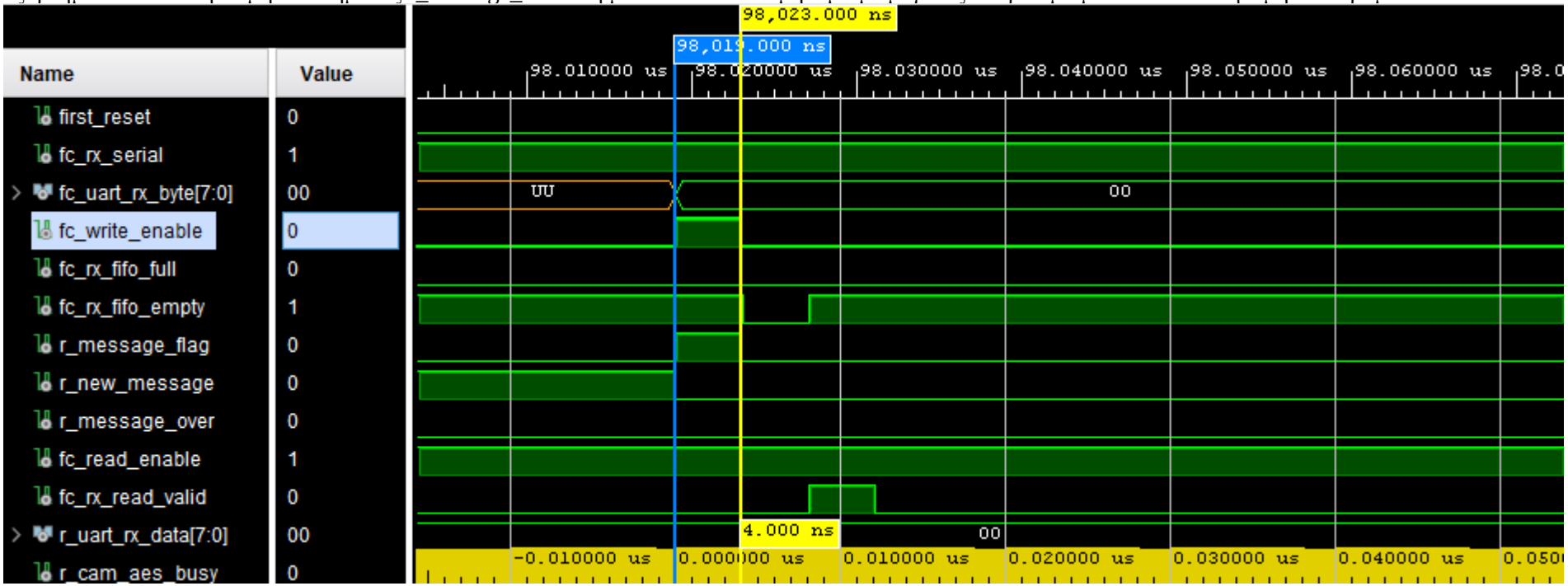
Στην εικόνα 5.3.1 παρουσιάζεται η προσομοίωση του σχηματισμού μπλοκ δεδομένων τηλεμετρίας μεγέθους 128 προς κρυπτογράφηση. Η διαδικασία ξεκινάει όταν η τιμή του σήματος fc_rx_read_valid γίνει ίση με 1, τότε σημαίνει πως υπάρχουν δεδομένα στην FIFO που συνδέεται το εξάρτημα Plaintext Generator. Όταν γίνει αυτό το εξάρτημα θα διαβάσει τα περιεχόμενα του διανύσματος r_uart_rx_data και θα τα αποθηκεύσει σε ένα δικό του τοπικό διάνυσμα. Αυτή η διαδικασία θα επαναληφθεί έως ότου διαβάσει συνολικά 16 byte όπου και θα κάνει την τιμή του σήματος fc_uart_data_ready ίση με 1 και θα στείλει στο εξάρτημα AES Encoder/Decoder τα δεδομένα προς κρυπτογράφηση.



Εικόνα 5.3.1 Προσομοίωση εξαρτήματος Plaintext Generator .

5.4 Έλεγχος μήκος ολικού μηνύματος (Εξάρτημα Message handler)

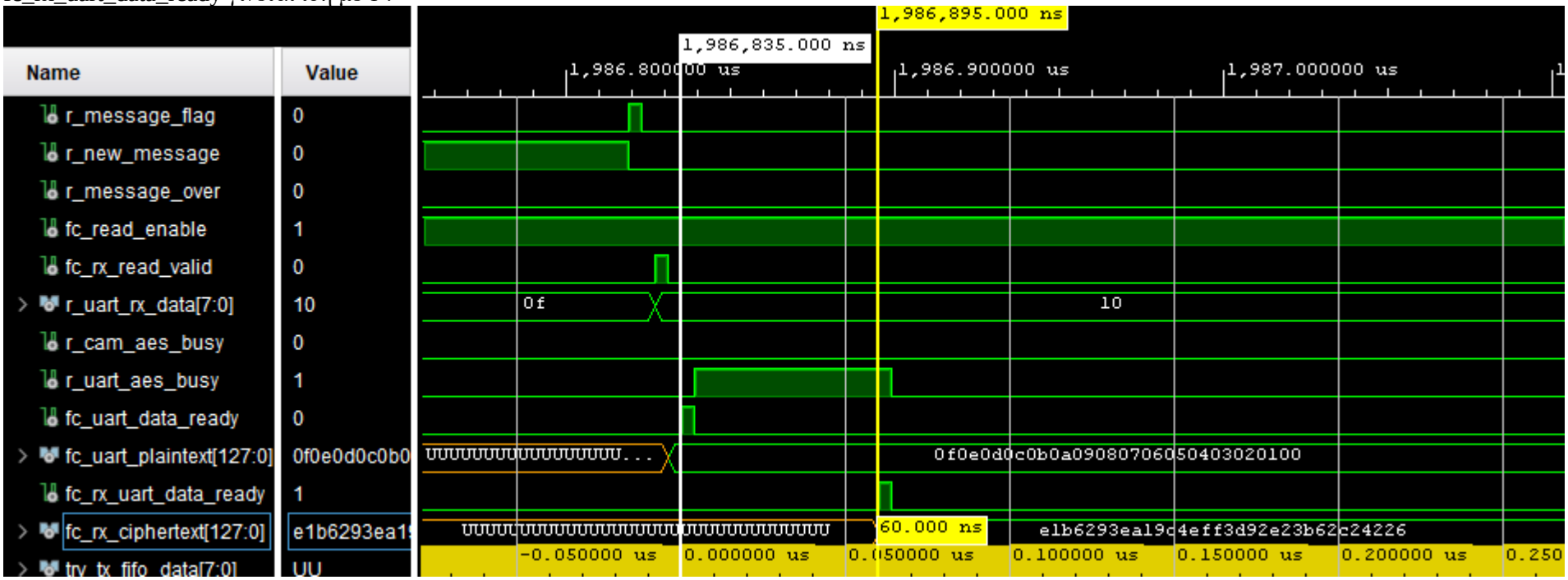
Στην εικόνα 5.4.1 παρουσιάζεται η προσομοίωση λειτουργίας του εξαρτήματος Message Handler. Όταν ληφθούν από το σύστημα 8 bit δεδομένων τηλεμετρίας τότε σημαίνει πως ένα νέο μήνυμα έρχεται από τον ελεγκτή πτήσης. Με το που γίνει αυτό η τιμή του σήματος r_message_flag θα γίνει ίση με 1 προκειμένου να καταλάβει το εξάρτημα ότι έφτασε ένα byte. Αν περάσει χρόνος ίσος με 26.040 ns τότε σημαίνει ότι το μήνυμα έχει σταλθεί ολόκληρο και το μέγεθος του δεν είναι ακέραια διαίρεση του 128 οπότε το εξάρτημα θα κάνει την τιμή του σήματος r_message_over ίση με 1. Αν σταλθεί μήνυμα με μέγεθος διαιρέσιμο με το 128 τότε η τιμή θα παραμείνει 0.



Εικόνα 5.4.1 Προσομοίωση εξαρτήματος Message Handler .

5.5 Κρυπτογράφηση δεδομένων τηλεμετρίας (Εξάρτημα AES Encoder/Decoder)

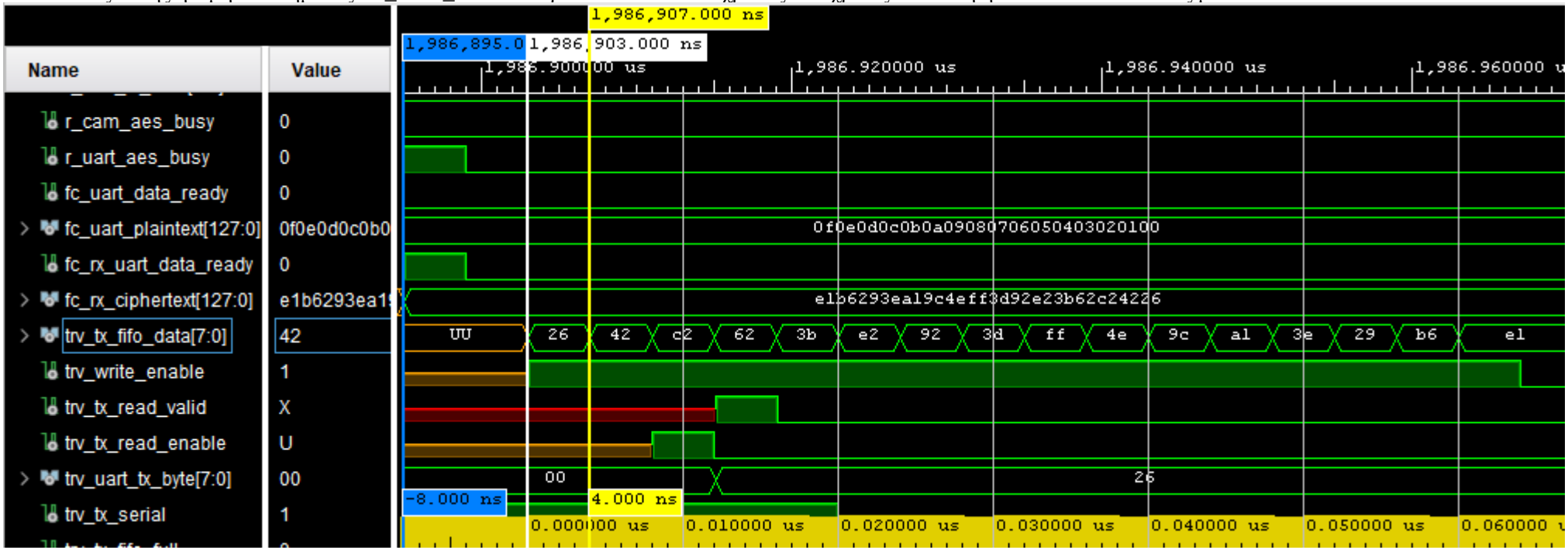
Στην εικόνα 5.5.1 παρουσιάζεται η προσομοίωση της κρυπτογράφησης δεδομένων τηλεμετρίας. Όταν η τιμή του σήματος fc_uart_data_ready γίνει ίση με 1 τότε τα δεδομένα που βρίσκονται στο διάνυσμα fc_uart_plaintext θα αποθηκευτούν από το εξάρτημα AES Encoder/Decoder και θα κρυπτογραφηθούν. Όσο το εξάρτημα τα κρυπτογραφεί η τιμή του σήματος r_uart_aes_busy θα είναι ίση με 1. Ο χρόνος που χρειάζεται για να βγουν τα κρυπτογραφημένα δεδομένα από την στιγμή που μπουν φτάσουν στο εξάρτημα AES Encoder/Decoder είναι ίσος με 60 ns. Με το πέρας του χρόνου αυτού τα κρυπτογραφημένα δεδομένα αποθηκεύονται στο διάνυσμα fc_rx_ciphertext και η τιμή του σήματος fc_rx_uart_data_ready γίνεται ίση με 1 .



Εικόνα 5.5.1 Προσομοίωση εξαρτήματος AES Encoder/Decoder .

5.6 Κατακερματισμός και αποθήκευση κρυπτογραφημένων δεδομένων τηλεμετρίας (Εξάρτημα UART Transmitter Buffer και FIFO)

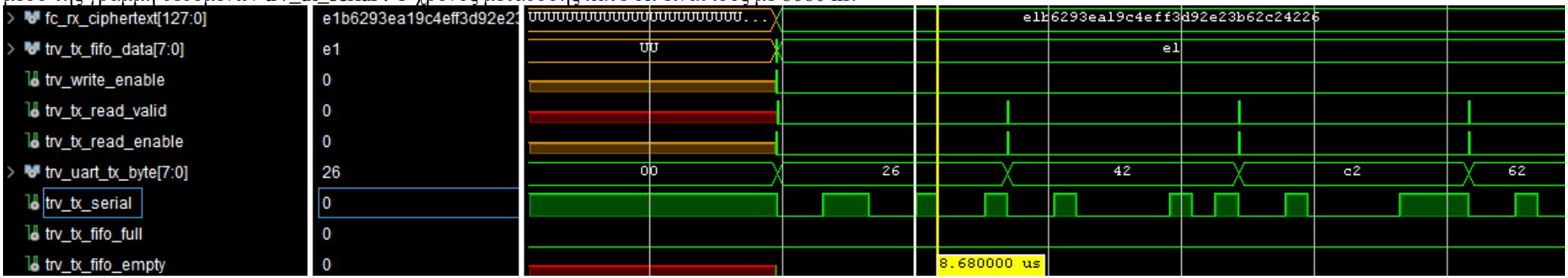
Στην εικόνα 5.6.1 παρουσιάζεται η προσομοίωση του κατακερματισμού των κρυπτογραφημένων δεδομένων τηλεμετρίας και η αποθήκευσή τους σε μνήμη FIFO. Όταν η τιμή του σήματος fc_rx_uart_data_ready γίνει ίση με 1 το εξάρτημα UART Transmitter Buffer θα αποθηκεύσει τα δεδομένα του διανύσματος fc_rx_ciphertext. Στην συνέχεια θα κάνει την τιμή του σήματος trv_write_enable ίση με 1 προκειμένου να αποθηκευτούν τα μπλοκ 8 bit δεδομένων του διανύσματος trv_tx_fifo_data στην FIFO. Με το πέρας της διαδικασίας αυτής η τιμή του σήματος trv_write_enable θα γίνει και πάλι 0. Ο χρόνος που χρειάζεται αυτή η διαδικασία είναι ίσος με 72 ns .



Εικόνα 5.6.1 Προσομοίωση εξαρτήματος UART Transmitter Buffer και FIFO .

5.7 Αποστολή κρυπτογραφημένων δεδομένων τηλεμετρίας από το FPGA στον UART transceiver(Εξάρτημα UART TX)

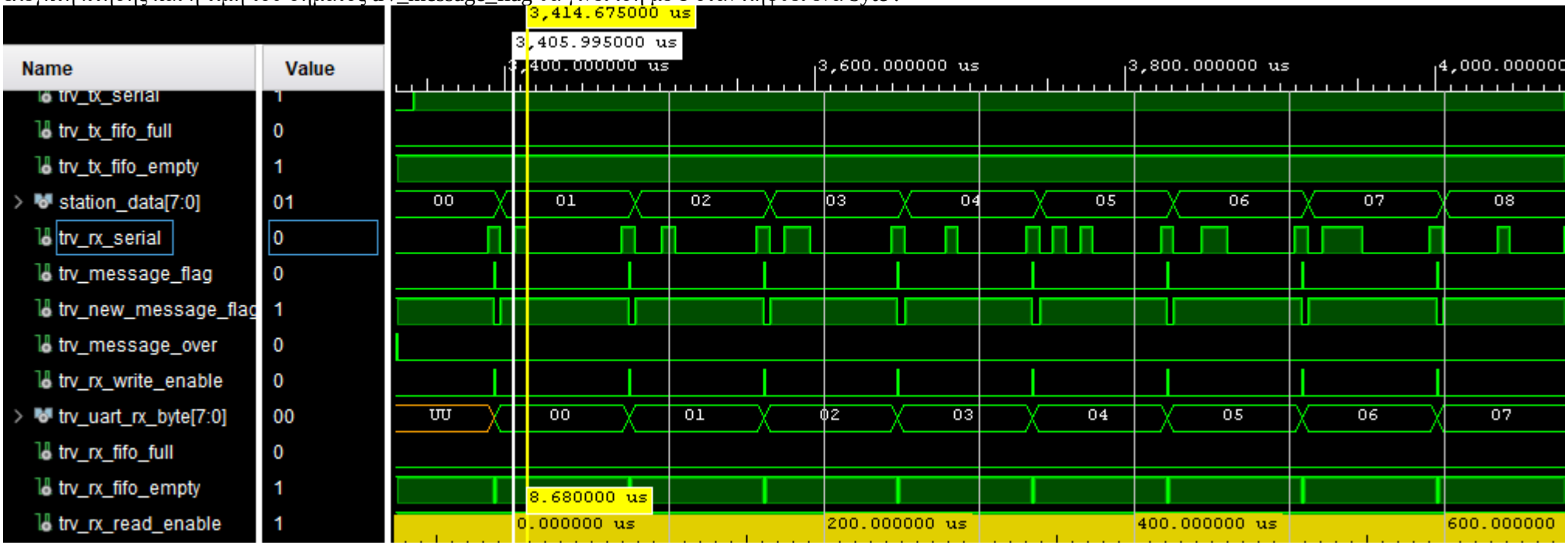
Στην εικόνα 5.7.1 παρουσιάζεται η προσομοίωση μετάδοσης κρυπτογραφημένων δεδομένων τηλεμετρίας μέσω του εξαρτήματος UART TX. Το εξάρτημα θα κάνει την τιμή του σήματος trv_tx_read_enable ίση με 1 όταν η τιμή του σήματος trv_tx_fifo_full είναι ίση με 0 προκειμένου να διαβάσει τα αποθηκευμένα δεδομένα από την FIFO. Μετά από ένα κύκλο ρολογιού (4 ns) το εξάρτημα θα κάνει την τιμή του σήματος trv_tx_read_enable ίση με 0 και θα ξεκινήσει να ελέγχει την τιμή του σήματος trv_tx_read_valid. Αν η τιμή του σήματος αυτού είναι ίση με 1 τότε σημαίνει ότι υπάρχουν διαθέσιμα δεδομένα, το εξάρτημα θα αποθηκεύσει αυτα τα δεδομένα και θα ξεκινήσει να τα στέλνει μέσο της γραμμή δεδομένων trv_tx_serial . Ο χρόνος μετάδοσης κάθε bit είναι ίσος με 8680 ns.



Εικόνα 5.1.1 Προσομοίωση εξαρτήματος UART TX .

5.8 Μετάδοση δεδομένων από τον UART transceiver στο FPGA μέσω UART (Εξάρτημα UART RX)

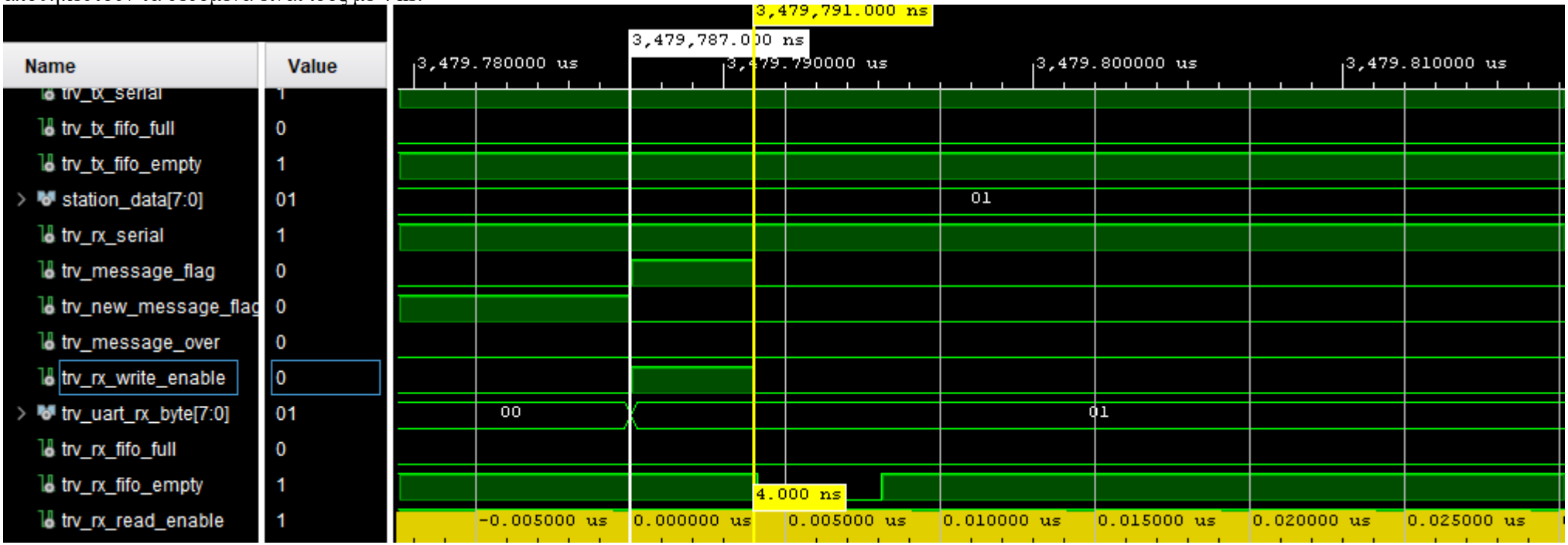
Στην εικόνα 5.8.1 παρουσιάζεται η μετάδοση των δεδομένων από τον UART Transceiver στο FPGA μέσω του UART. Παρατηρούμε ότι η λειτουργία είναι σωστή καθώς προκειμένου να ξεκινήσει η μετάδοση η γραμμή επικοινωνίας trv_rx_serial θα γίνει από ένα μηδέν προκειμένου να ξεκινήσει η μετάδοση δεδομένων μέσω UART. Ύστερα, ανάλογα με το byte που θα μεταδοθεί η τιμή της γραμμής δεδομένων θα γίνει 1 ή 0 για χρόνο ίσο με αυτόν που έχει εφαρμοστεί ανάλογα με το baudrate του UART, η διαδικασία αυτή θα πραγματοποιηθεί για το κάθε bit του byte δεδομένων δηλαδή για 8 φορές όπου και στο τέλος η τιμή της γραμμής δεδομένων θα γίνει ίση με 1 για να καταλάβει το FPGA ότι τελείωσαν τα δεδομένα. Για παράδειγμα στην εικόνα 5.8.1 όταν στέλνεται το byte με τιμή 01 αφού ξεκινήσει η επικοινωνία θα γίνει η τιμή της γραμμής δεδομένων ίση με 1 για 8680 ns και μετά θα παραμείνει 0 για τα υπόλοιπα bits. Με το πέρας της διαδικασίας αυτής ενημερώνεται η τιμή του διανύσματος trv_uart_rx_byte με την τιμή που λήφθηκε και η τιμή του σήματος trv_rx_write_enable γίνεται ίση με 1 προκειμένου να γραφτούν τα δεδομένα στην μνήμη FIFO. Επίσης η τιμή του σήματος trv_new_message_flag θα γίνει ίση με 1 κατά την διάρκεια λήψης μηνύματος προκειμένου να καταλάβει το εξάρτημα Message Handler ότι ένα νέο μήνυμα έρχεται από τον ελεγκτή πτήσης και η τιμή του σήματος trv_message_flag θα γίνει ίση με 1 όταν ληφθεί ένα byte .



Εικόνα 5.8.1 Προσομοίωση εξαρτήματος UART RX (Δεδομένα εντολών).

5.9 Αποθήκευση δεδομένων εντολών σε μνήμη (Εξάρτημα FIFO)

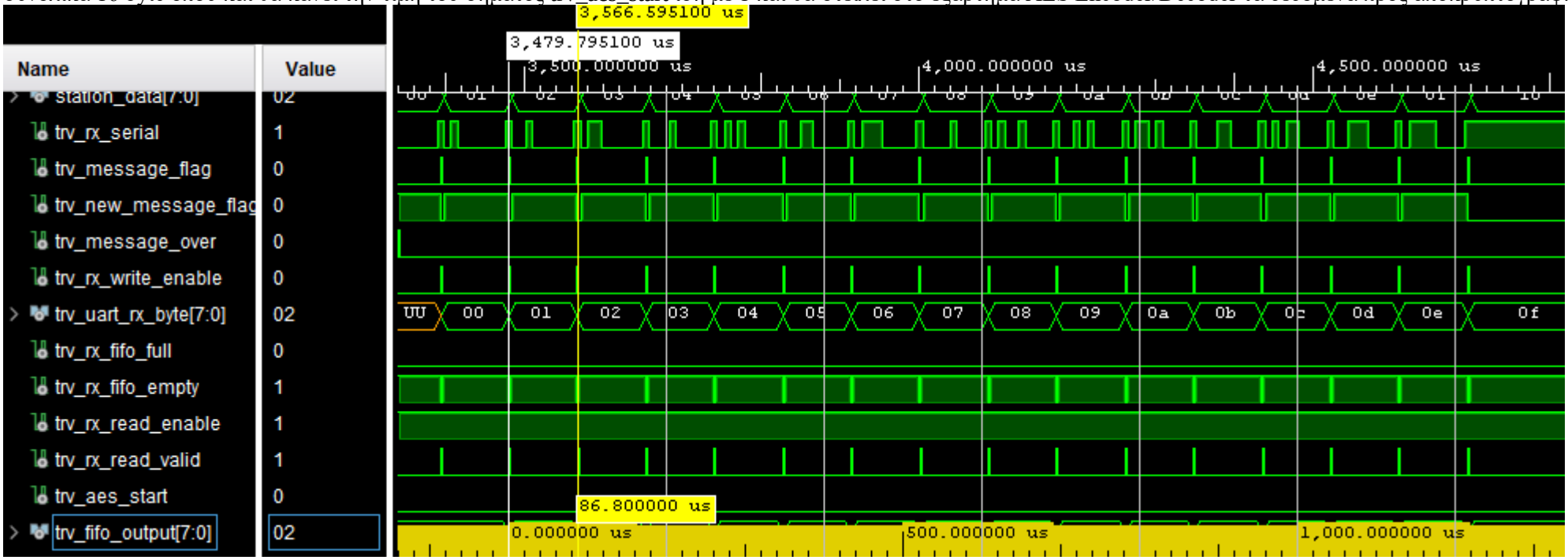
Στην εικόνα 5.9.1 παρουσιάζεται η προσομοίωση αποθήκευσης εντολών τηλεμετρίας στην FIFO που συνδέεται με το εξάρτημα UART RX. Όταν η τιμή του σήματος trv_rx_write_enable γίνει ίση με 1 τότε θα ξεκινήσει η αποθήκευση δεδομένων. Με το που αποθηκευτεί το πρώτο byte η τιμή του σήματος trv_rx_fifo_empty θα γίνει ίση με 0 καθώς τώρα θα υπάρξουν δεδομένα στην FIFO, με το που βγουν από την FIFO αυτά τα δεδομένα η τιμή του θα ξαναγίνει ίση με 1. Ο χρόνος που χρειάζεται για να αποθηκευτούν τα δεδομένα είναι ίσος με 4 ns.



Εικόνα 5.9.1 Προσομοίωση εξαρτήματος FIFO .

5.10 Σχηματισμός μπλοκ δεδομένων εντολών προς αποκρυπτογράφηση(Εξάρτημα Plaintext Generator)

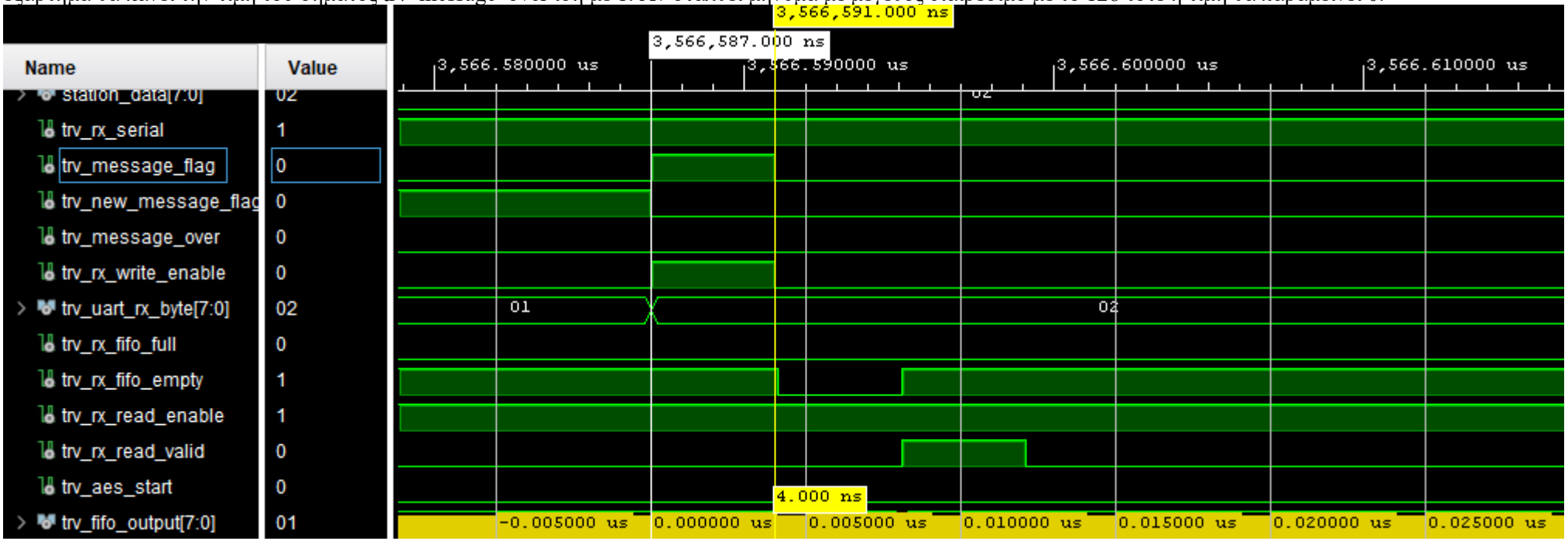
Στην εικόνα 5.10.1 παρουσιάζεται η προσομοίωση του σχηματισμού μπλοκ δεδομένων εντολών μεγέθους 128 προς αποκρυπτογράφηση. Η διαδικασία ξεκινάει όταν η τιμή του σήματος trv_rx_read_valid γίνει ίση με 1, τότε σημαίνει πως υπάρχουν δεδομένα στην FIFO που συνδέεται το εξάρτημα Plaintext Generator. Όταν γίνει αυτό το εξάρτημα θα διαβάσει τα περιεχόμενα του διανύσματος trv_uart_rx_byte και θα τα αποθηκεύσει σε ένα δικό του τοπικό διάνυσμα. Αυτή η διαδικασία θα επαναληφθεί έως ότου διαβάσει συνολικά 16 byte όπου και θα κάνει την τιμή του σήματος trv_aes_start ίση με 1 και θα στείλει στο εξάρτημα AES Encoder/Decoder τα δεδομένα προς αποκρυπτογράφηση.



Εικόνα 5.10.1 Προσομοίωση εξαρτήματος Plaintext Generator .

5.11 Έλεγχος μήκος ολικού μηνύματος (Εξάρτημα Message handler)

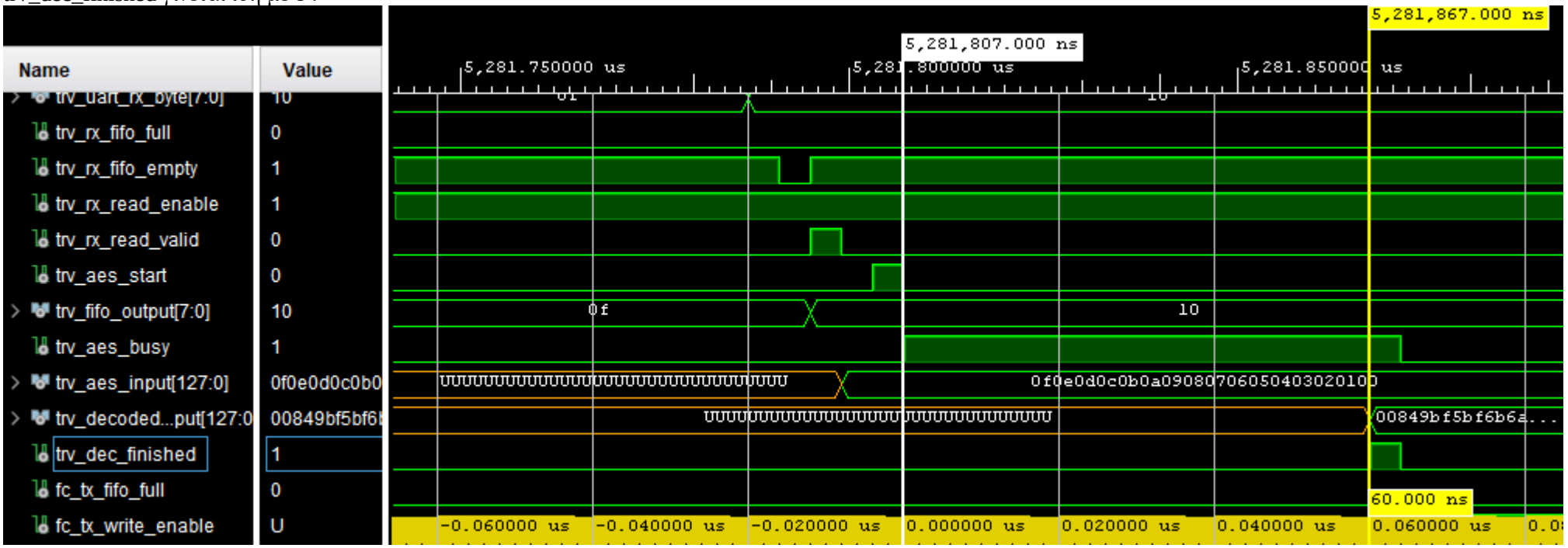
Στην εικόνα 5.11.1 παρουσιάζεται η προσομοίωση λειτουργίας του εξαρτήματος Message Handler. Όταν ληφθούν από το σύστημα 8 bit δεδομένων τηλεμετρίας τότε σημαίνει πως ένα νέο μήνυμα έρχεται από τον ελεγκτή πτήσης. Με το που γίνει αυτό η τιμή του σήματος trv_message_flag θα γίνει ίση με 1 προκειμένου να καταλάβει το εξάρτημα ότι έφτασε ένα byte. Αν περάσει χρόνος ίσος με 26.040 ns τότε σημαίνει ότι το μήνυμα έχει σταλθεί ολόκληρο και το μέγεθος του δεν είναι ακέραια διαίρεση του 128 οπότε το εξάρτημα θα κάνει την τιμή του σήματος trv_message_over ίση με 1. Αν σταλθεί μήνυμα με μέγεθος διαιρέσιμο με το 128 τότε η τιμή θα παραμείνει 0.



Εικόνα 5.11.1 Προσομοίωση εξαρτήματος Message Handler .

5.12 Αποκρυπτογράφηση δεδομένων εντολών (Εξάρτημα AES Encoder/Decoder)

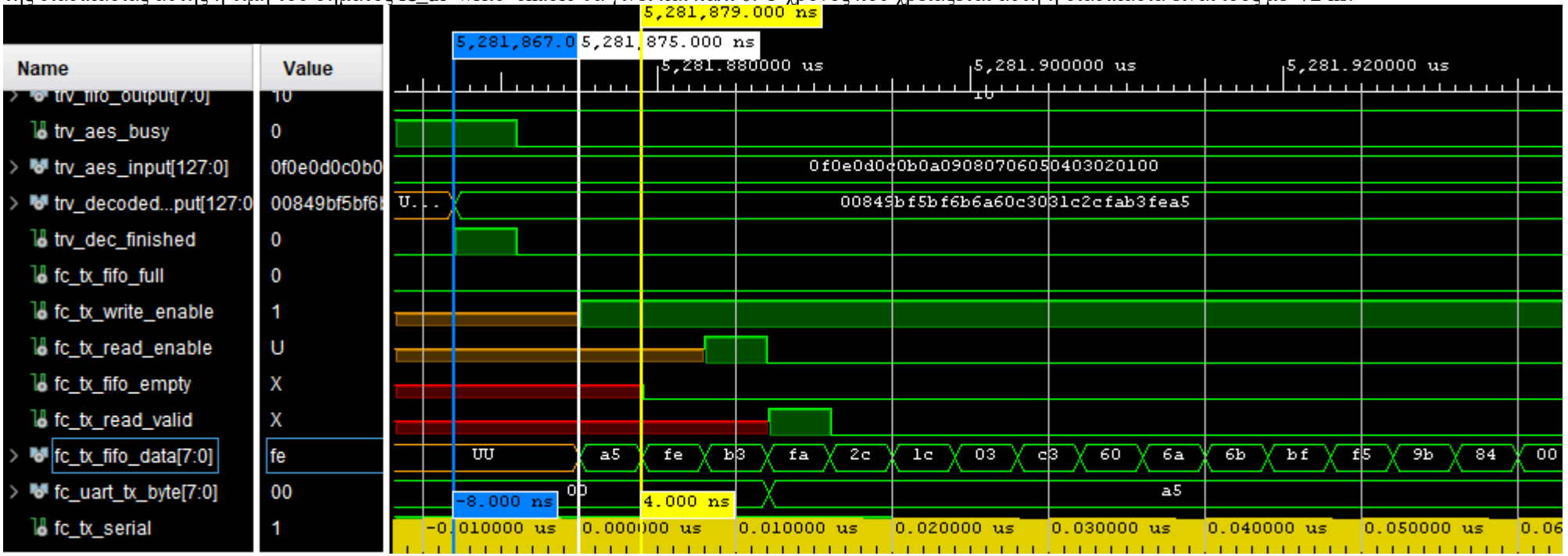
Στην εικόνα 5.12.1 παρουσιάζεται η προσομοίωση της αποκρυπτογράφησης δεδομένων τηλεμετρίας. Όταν η τιμή του σήματος trv_aes_start γίνει ίση με 1 τότε τα δεδομένα που βρίσκονται στο διάνυσμα trv_aes_input θα αποθηκευτούν από το εξάρτημα AES Encoder/Decoder και θα κρυπτογραφηθούν. Όσο το εξάρτημα τα κρυπτογραφεί η τιμή του σήματος trv_aes_busy θα είναι ίση με 1. Ο χρόνος που χρειάζεται για να βγουν τα κρυπτογραφημένα δεδομένα από την στιγμή που μπουν φτάσουν στο εξάρτημα AES Encoder/Decoder είναι ίσος με 60 ns . Με το πέρας του χρόνου αυτού τα κρυπτογραφημένα δεδομένα αποθηκεύονται στο διάνυσμα trv_decoded_output και η τιμή του σήματος trv_dec_finished γίνεται ίση με 1 .



Εικόνα 5.12.1 Προσομοίωση εξαρτήματος AES Encoder/Decoder .

5.13 Κατακερματισμός αποκρυπτογραφημένων δεδομένων εντολών (Εξάρτημα UART Transmitter Buffer)

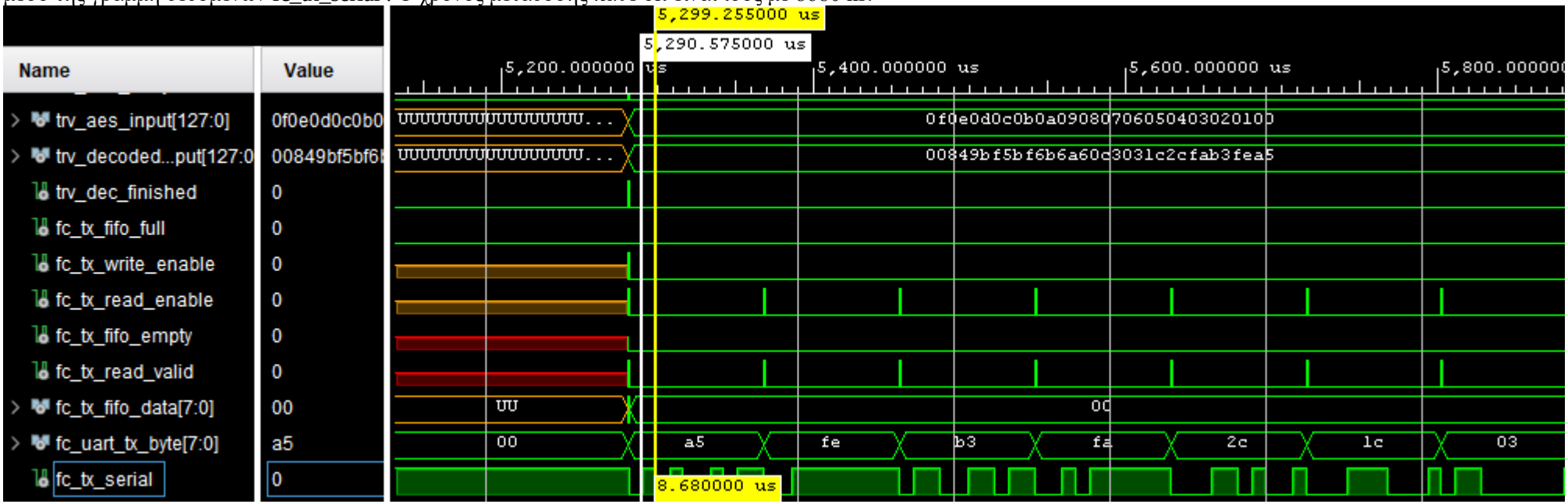
Στην εικόνα 5.13.1 παρουσιάζεται η προσομοίωση του κατακερματισμού των αποκρυπτογραφημένων δεδομένων τηλεμετρίας και η αποθήκευσή τους σε μνήμη FIFO. Όταν η τιμή του σήματος trv_dec_finsihed γίνει ίση με 1 το εξάρτημα UART Transmitter Buffer θα αποθηκεύσει τα δεδομένα του διανύσματος trv_decoded_output. Στην συνέχεια θα κάνει την τιμή του σήματος fc_tx_write_enable ίση με 1 προκειμένου να αποθηκευτούν τα μπλοκ 8 bit δεδομένων του διανύσματος fc_tx_fifo_data στην FIFO. Με το πέρας της διαδικασίας αυτής η τιμή του σήματος fc_tx_write_enable θα γίνει και πάλι 0. Ο χρόνος που χρειάζεται αυτή η διαδικασία είναι ίσος με 72 ns.



Εικόνα 5.13.1 Προσομοίωση εξαρτήματος UART Transmitter Buffer.

5.14 Αποστολή αποκρυπτογραφημένων δεδομένων εντολών από το FPGA στον ελεγκτή πτήσης (Εξάρτημα UART TX)

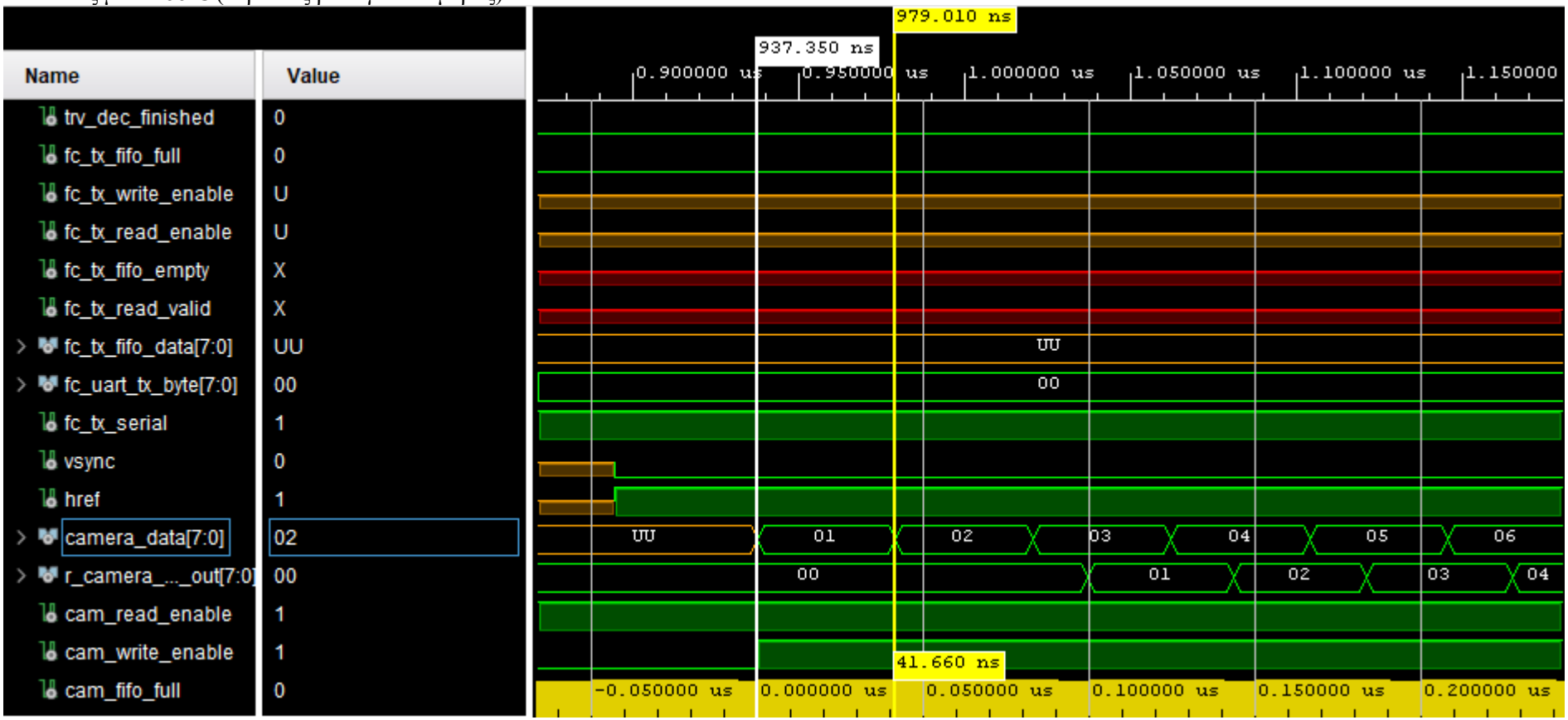
Στην εικόνα 5.14.1 παρουσιάζεται η προσομοίωση μετάδοσης αποκρυπτογραφημένων δεδομένων τηλεμετρίας μέσο του εξαρτήματος UART TX. Το εξάρτημα θα κάνει την τιμή του σήματος fc_tx_read_enable ίση με 1 όταν η τιμή του σήματος fc_tx_fifo_empty είναι ίση με 0 προκειμένου να διαβάσει τα αποθηκευμένα δεδομένα από την FIFO. Μετά από ένα κύκλο ρολογιού (4 ns) το εξάρτημα θα κάνει την τιμή του σήματος fc_tx_read_enable ίση με 0 και θα ξεκινήσει να ελέγχει την τιμή του σήματος fc_tx_read_valid. Αν η τιμή του σήματος αυτού είναι ίση με 1 τότε σημαίνει ότι υπάρχουν διαθέσιμα δεδομένα, το εξάρτημα θα αποθηκεύσει αυτα τα δεδομένα και θα ξεκινήσει να τα στέλνει μέσο της γραμμή δεδομένων fc_tx_serial . Ο χρόνος μετάδοσης κάθε bit είναι ίσος με 8680 ns.



Εικόνα 5.14.1 Προσομοίωση εξαρτήματος UART TX .

5.15 Λήψη δεδομένων βίντεο από το FPGA και αποθήκευση σε μνήμη FIFO(Εξάρτημα Camera Capture και FIFO Buffer)

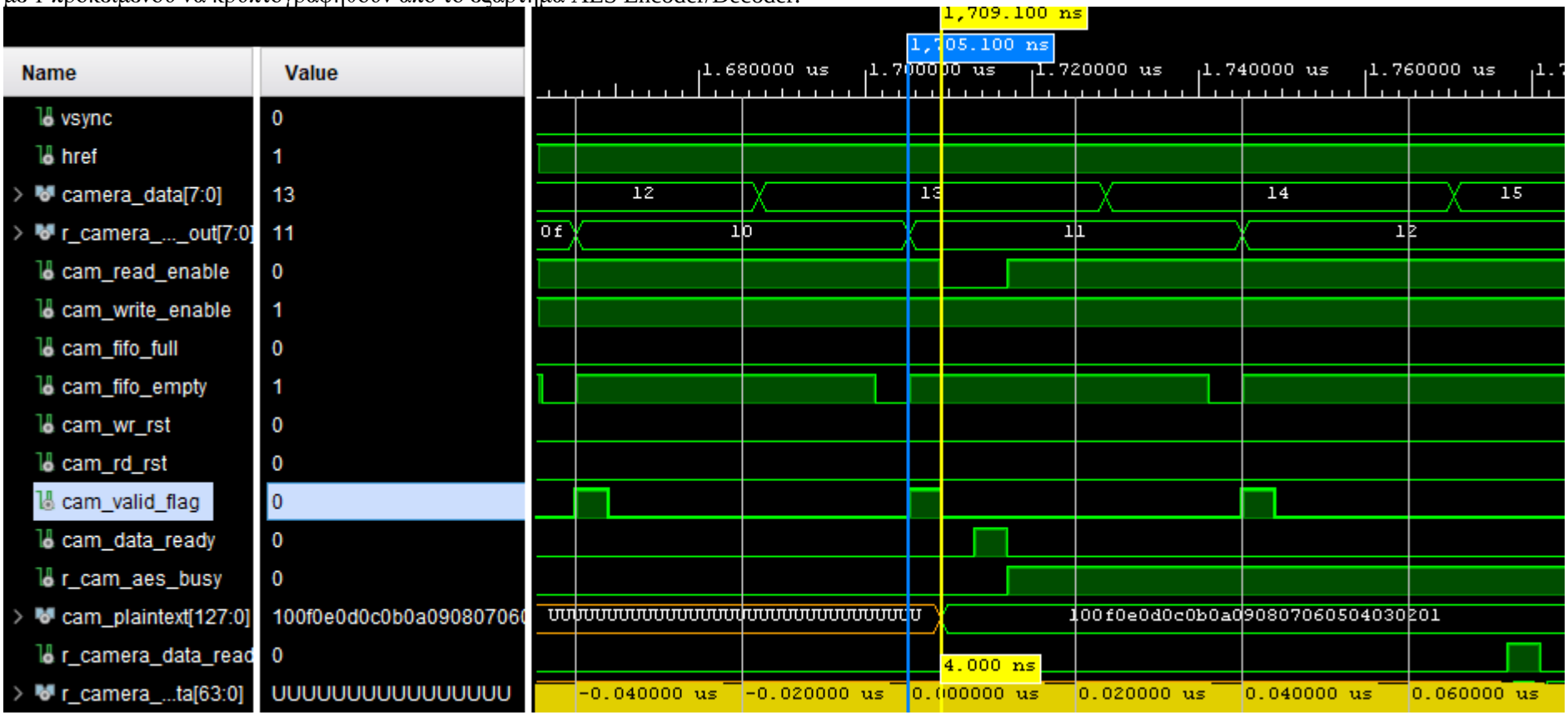
Στην εικόνα 5.15.1 παρουσιάζεται η προσομοίωση λήψης δεδομένων από την κάμερα. Η λήψη ξεκινάει όταν η τιμή του σήματος **href** είναι ίση με 1 και του σήματος **vsync** ίση με 0. Τα δεδομένα της κάμερας, που στέλνονται 8 bit ανά κύκλο ρολογιού της κάμερας αποθηκεύονται και στέλνονται στο διάνυσμα camera_data το οποίο είναι η είσοδός τους στην FIFO που συνδέεται με το εξάρτημα Camera Capture. Η αποθήκευσή τους γίνεται όταν η τιμή του σήματος cam_write_enable είναι ίση με 1. Ο χρόνος αποθήκευσής είναι ίσος με 41.66ns (περίοδος ρολογιού κάμερας).



Εικόνα 5.15.1 Προσομοίωση εξαρτήματος Camera Capture και FIFO Buffer.

5.16 Σχηματισμός διανύσματος δεδομένων προς κρυπτογράφηση (Εξάρτημα Camera Buffer)

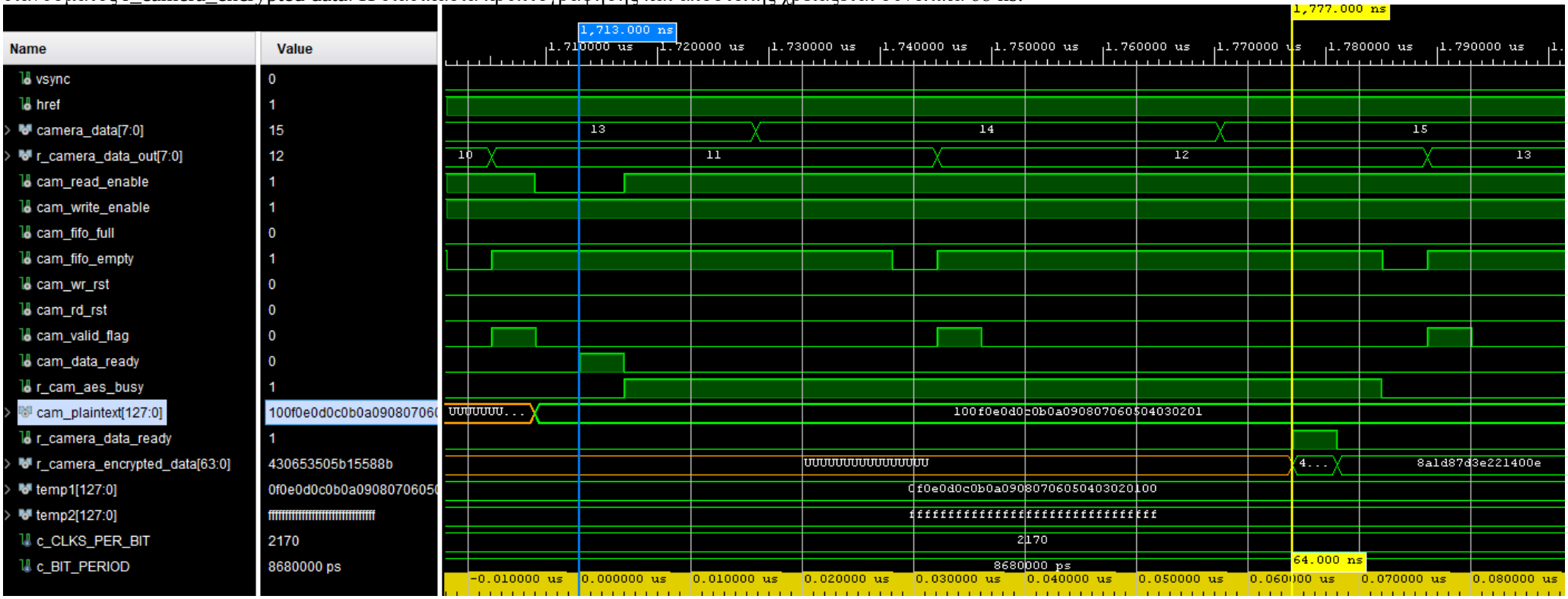
Στην εικόνα 5.16.1 παρουσιάζεται η προσομοίωση σχηματισμού διανύσματος δεδομένων κάμερας προς κρυπτογράφηση. Το εξάρτημα Camera Buffer κάνει την τιμή του σήματος cam_read_enable ίση με 1 προκειμένου να διαβάσει δεδομένα από την FIFO. Όταν η τιμή του σήματος cam_valid_flag είναι ίση με 1 τότε το εξάρτημα μπορεί να διαβάσει δεδομένα. Αφού διαβάσει συνολικά 16 byte από την FIFO θα τα αποθηκεύσει στο διάνυσμα cam_plaintext και θα κάνει την τιμή του σήματος cam_data_ready ίση με 1 προκειμένου να κρυπτογραφηθούν από το εξάρτημα AES Encoder/Decoder.



Εικόνα 5.16.1 Προσομοίωση εξαρτήματος Camera Buffer .

5.17 Κρυπτογράφηση και αποστολή δεδομένων βίντεο (Εξάρτημα AES Encoder/Decoder)

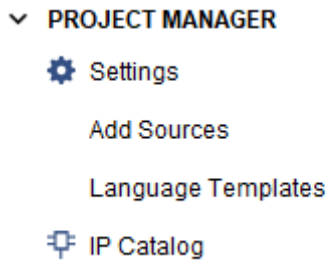
Στην εικόνα 5.17.1 παρουσιάζεται η προσομοίωση κρυπτογράφησης και αποστολής δεδομένων βίντεο. Όταν η τιμή του σήματος cam_data_ready είναι ίση με 1 τότε υπάρχουν διαθέσιμα δεδομένα προς κρυπτογράφηση στο διάνυσμα cam_plaintext. Όσο κρυπτογραφούνται τα δεδομένα η τιμή του σήματος r_cam_aes_busy είναι ίση με 1. Όταν τελειώσει η κρυπτογράφηση τους τα κρυπτογραφημένα δεδομένα βίντεο θα βγουν από το εξάρτημα σε 2 κύκλους ρολογιού συστήματος σε κομμάτια των 64 bits μέσω του διανύσματος r_camera_encrypted data. Η διαδικασία κρυπτογράφησης και αποστολής χρειάζεται συνολικά 68 ns.



Εικόνα 5.17.1 Προσομοίωση εξαρτήματος AES Encoder/Decoder για κρυπτογράφηση βίντεο .

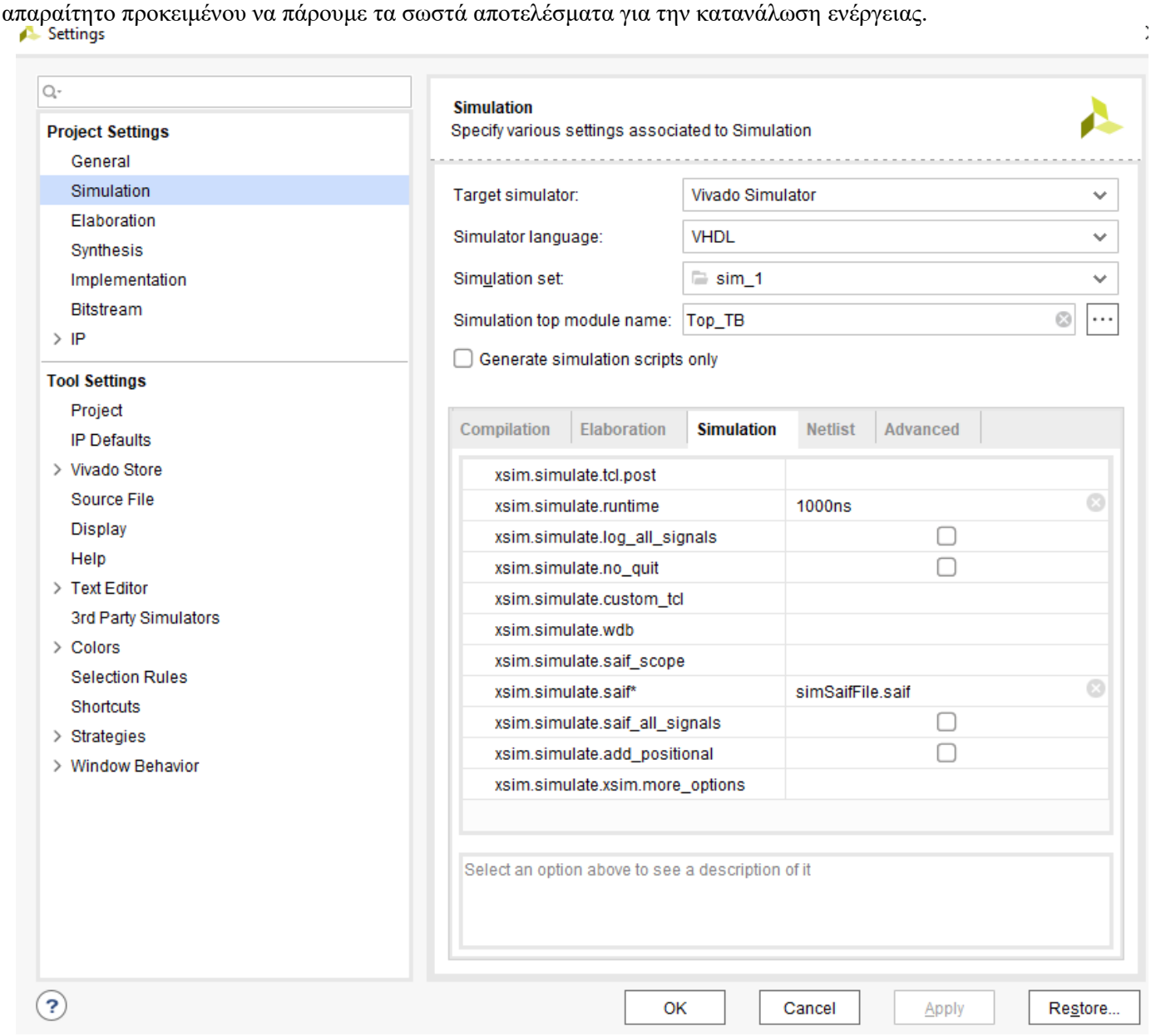
6. ‘Εύρεση ενέργειας

Για να βρούμε την ενέργεια πρώτα θα πάμε στην αρχική σελίδα του project όπως παρουσιάζεται και στην ενότητα δημιουργία project στο vivado και θα επιλέξουμε στο κομμάτι του Project Manager την επιλογή “Settings”.



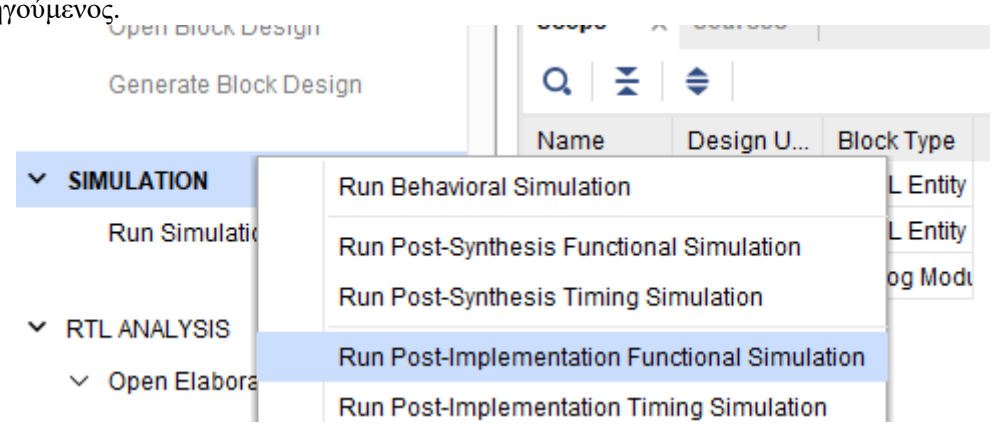
Εικόνα 6.1 Επιλογή “Settings”

Στην συνέχεια στο παράθυρο που άνοιξε θα μεταφερθούμε στο κομμάτι “Simulation”, εκεί θα πάμε στην επιλογή xsim.simulate.saif* όπου και θα γράψουμε το όνομα που θέλουμε να έχει το αρχείο saif. Το αρχείο αυτό είναι απαραίτητο προκειμένου να πάρουμε τα σωστά αποτελέσματα για την κατανάλωση ενέργειας.



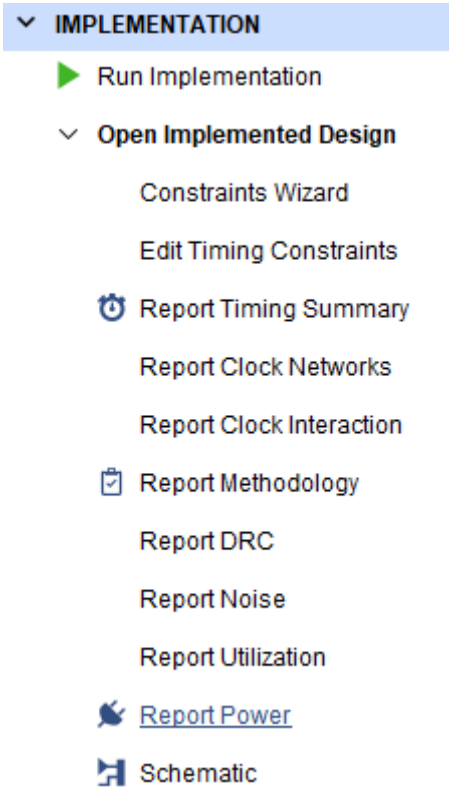
Εικόνα 6.2 Δημιουργία αρχείου Saif

Με το πέρας της διαδικασίας αυτής θα μεταφερθούμε πάλι στην αρχική σελίδα του project και στο κομμάτι Simulation θα επιλέξουμε Run Post_Implementation Functional Simulation προκειμένου να δημιουργηθεί το αρχείο που αναφέραμε προηγούμενος.



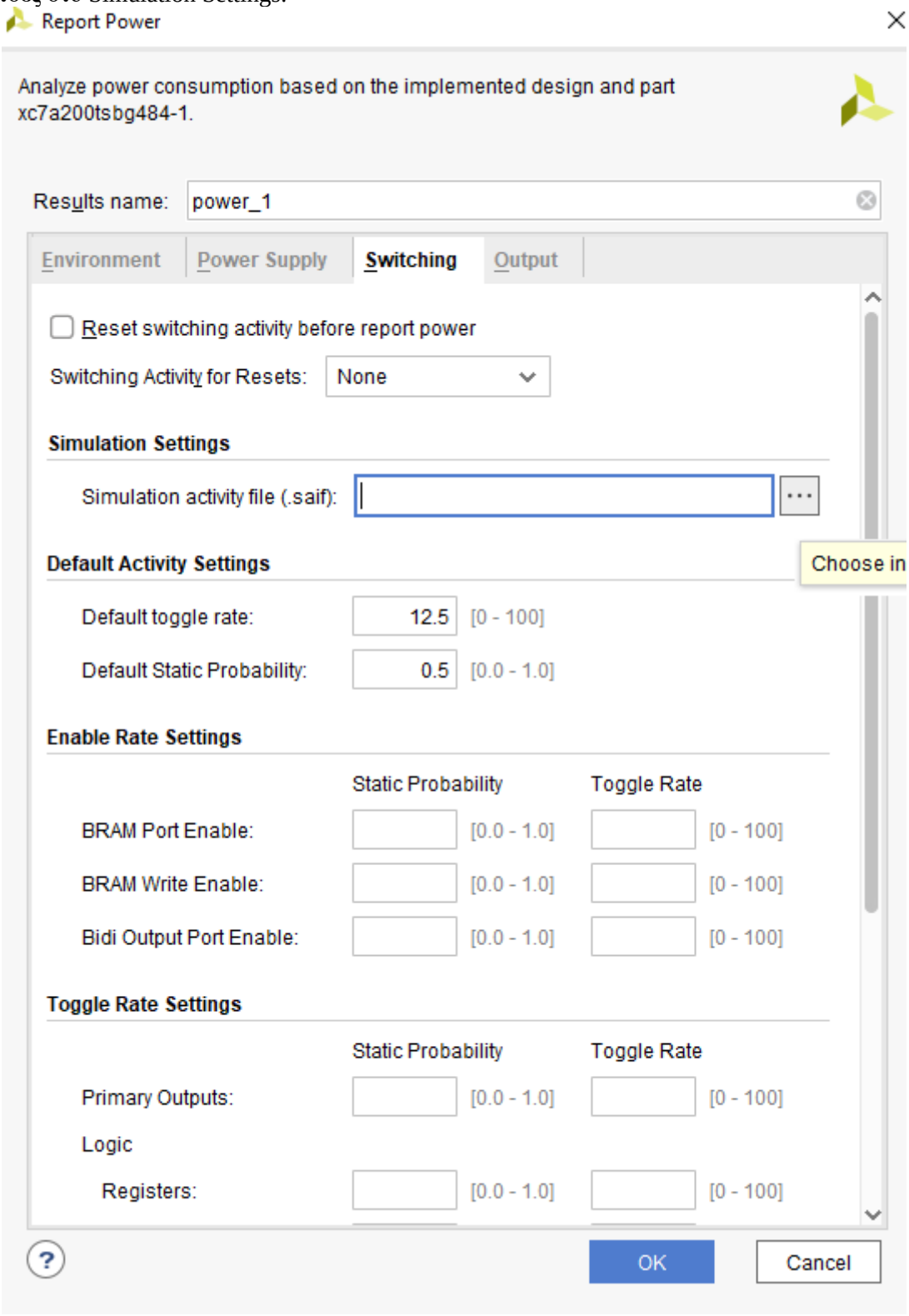
Εικόνα 6.3 Εκκίνηση προσομοίωσης

Με το πέρας της προσομοίωσης θα μεταφερθούμε για άλλη μια φορά στην αρχική σελίδα και θα επιλέξουμε στο κομμάτι “Implementation” την επιλογή “Report Power”



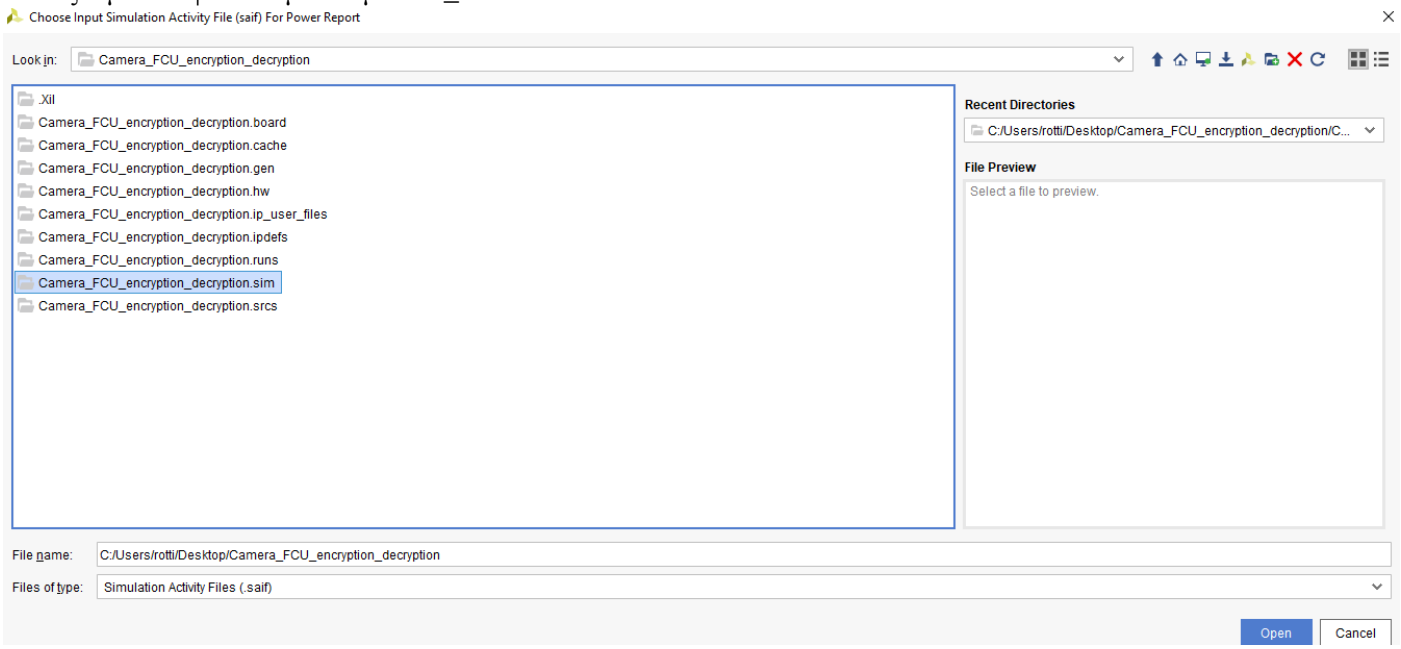
Εικόνα 6.4 Επιλογή “Report Power”

Στην συνέχεια στο παράθυρο που θα ανοίξει θα μεταφερθούμε στο κομμάτι που γράφει Switching και θα πατήσουμε τις τρείς τελίτσες στο Simulation Settings.

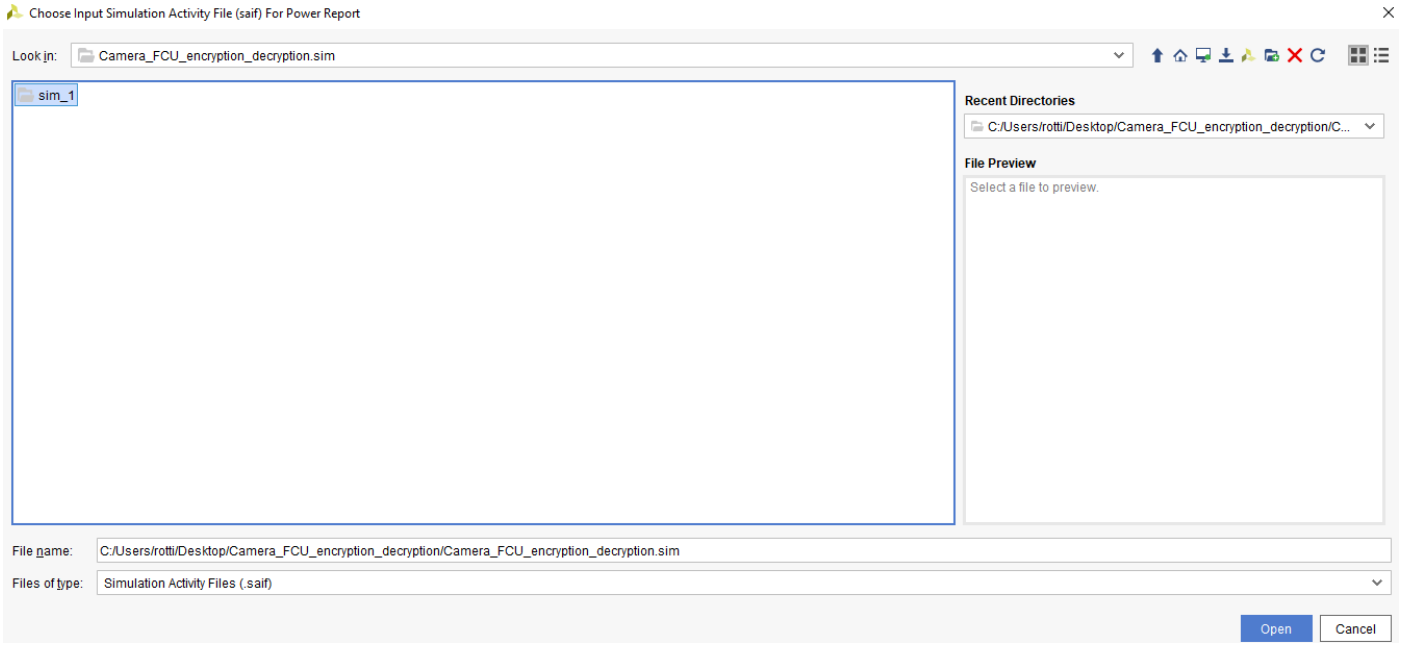


Εικόνα 6.5 Επιλογή αρχείου Saif

Στην συνέχεια θα επιλέξουμε τον φάκελο του project και θα μεταφερθούμε στον φάκελο με κατάληξη .sim. Έπειτα θα επιλέξουμε τον φάκελο με όνομα sim_1.

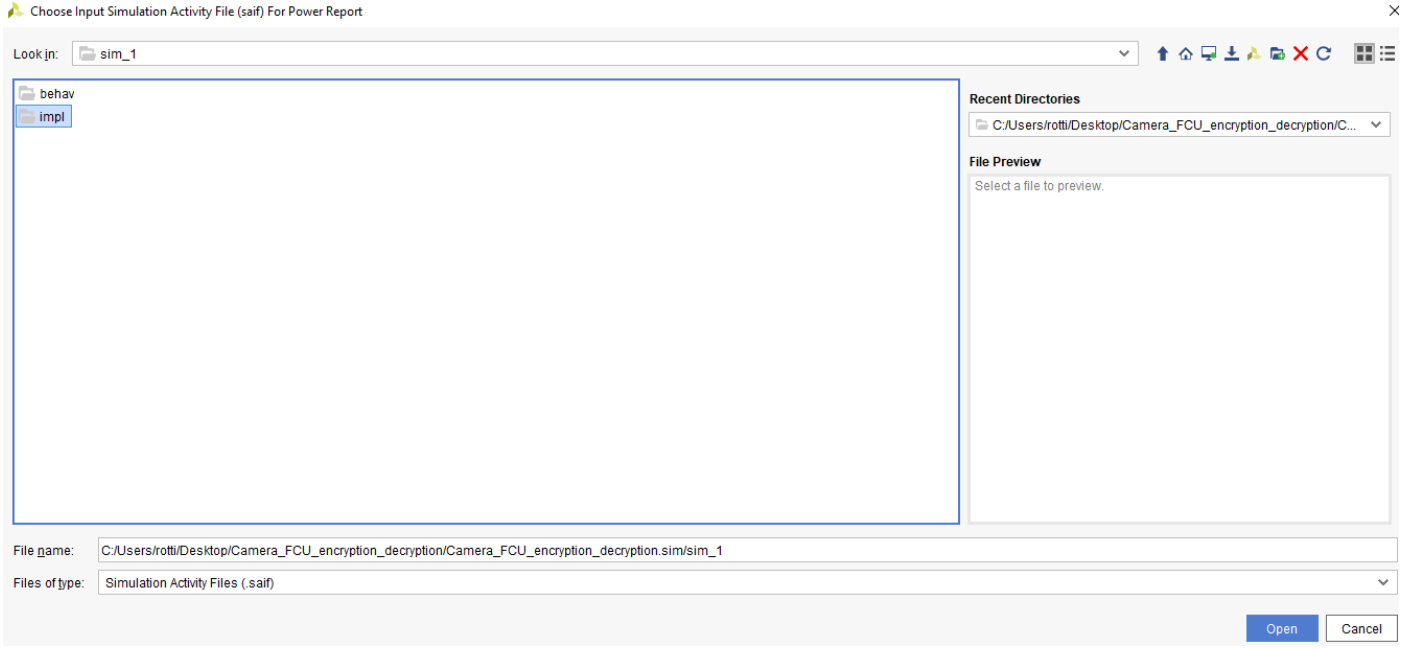


Εικόνα 6.6 Επιλογή φακέλου sim

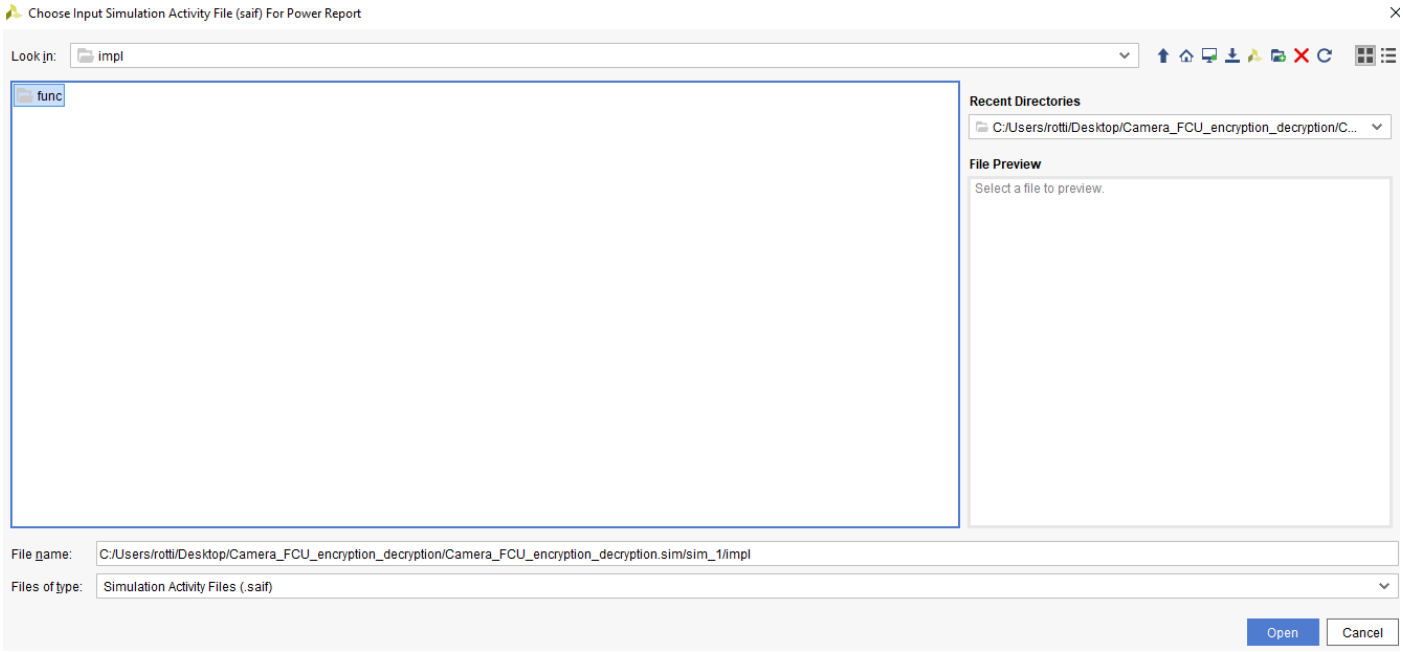


Εικόνα 6.7 Επιλογή φακέλου sim_1

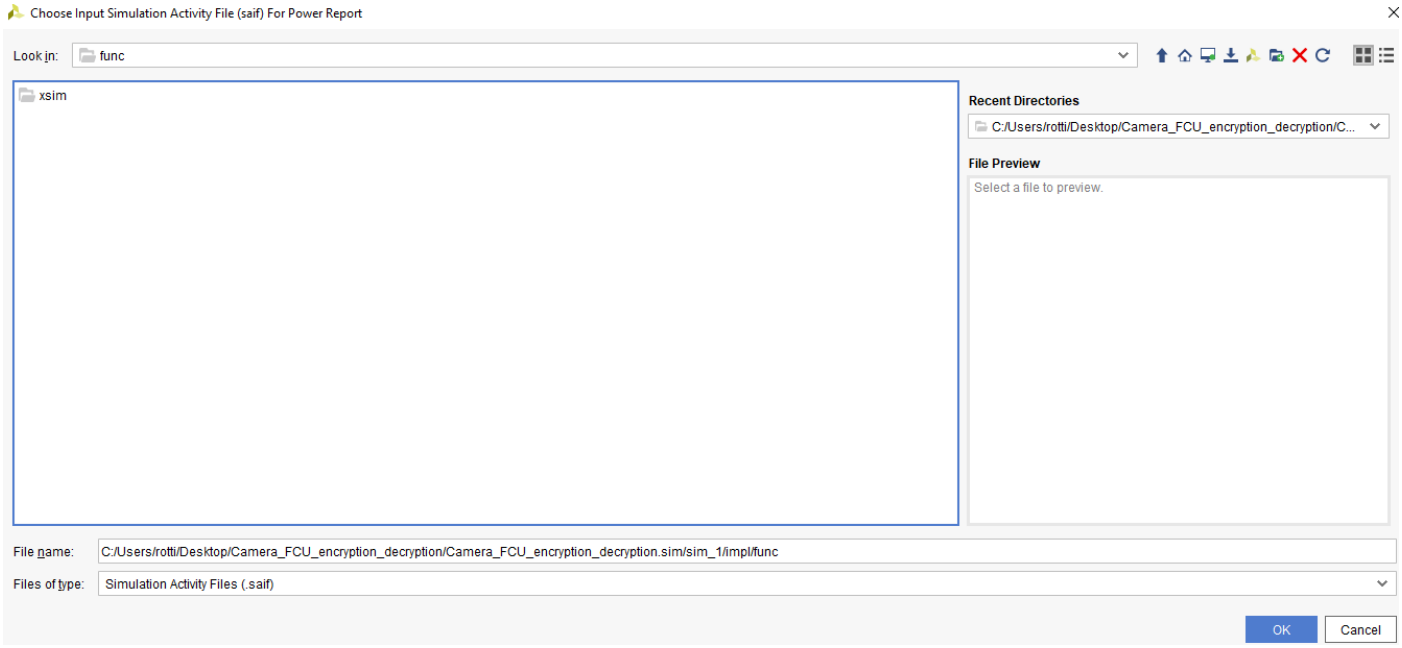
Συνεχίζοντας θα επιλέξουμε τον φάκελο με όνομα impl και θα προχωρήσουμε ανοίγοντας τους φακέλους εώς ότου φτάσουμε στον τελευταίο φάκελο με όνομα xsim. Μέσα σε αυτόν θα επιλέξουμε το .saif αρχείο.



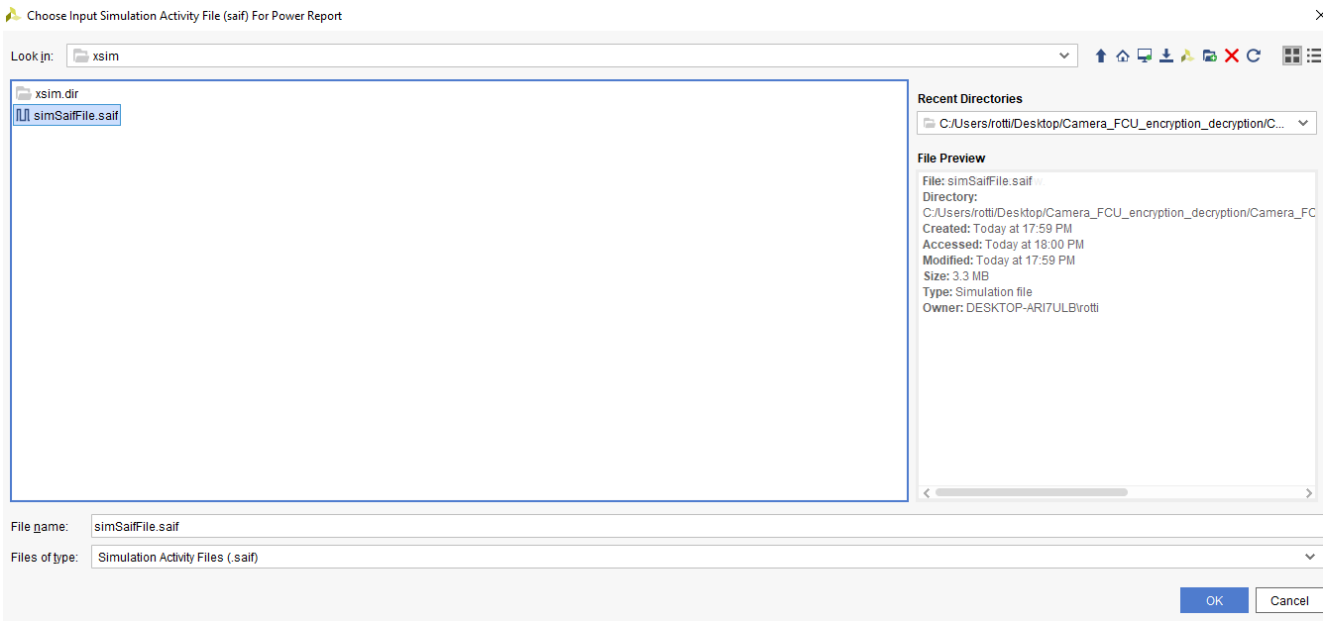
Εικόνα 6.8 Επιλογή φακέλου impl



Εικόνα 6.9 Επιλογή φακέλου func

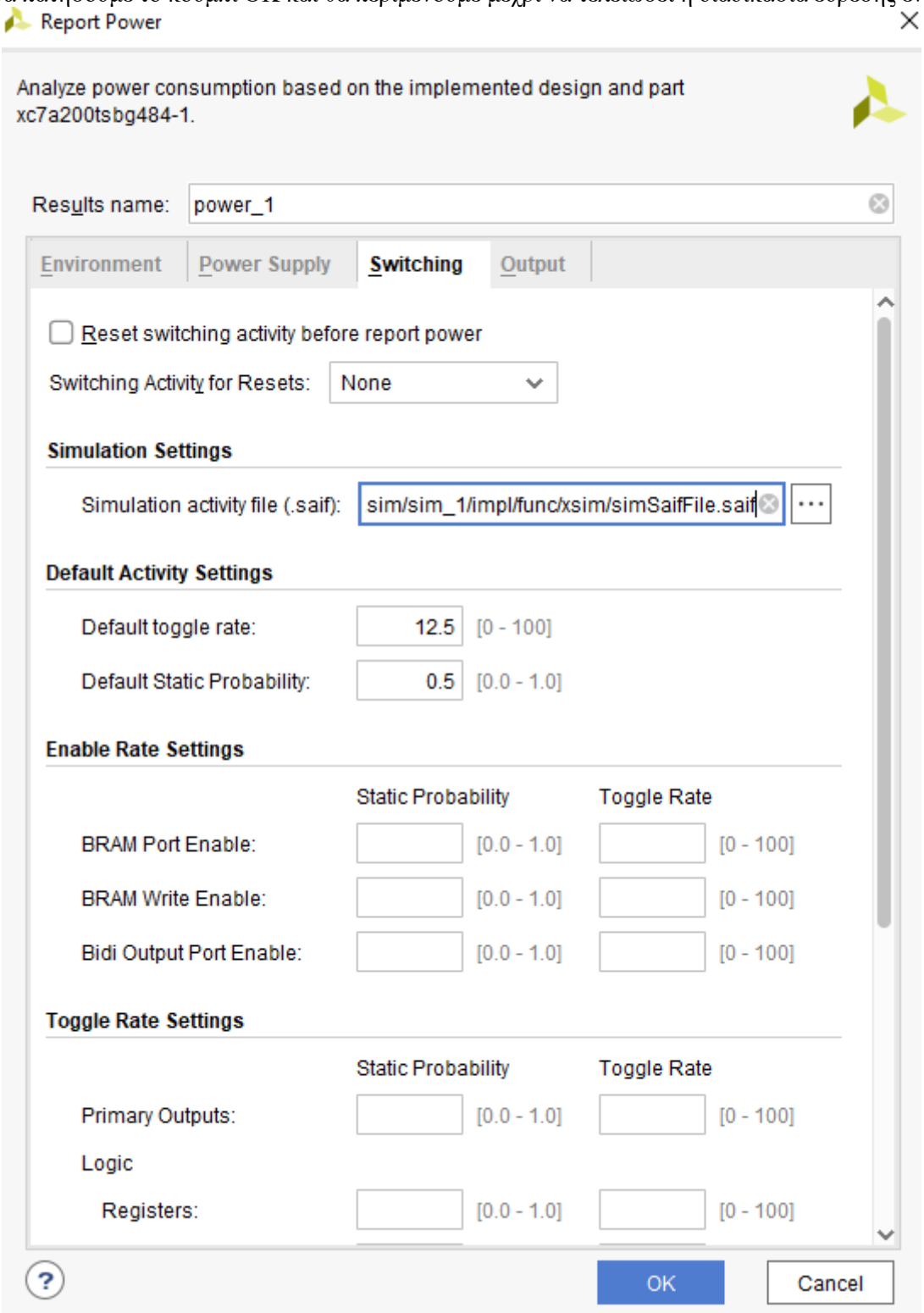


Εικόνα 6.10 Επιλογή φακέλου xsim



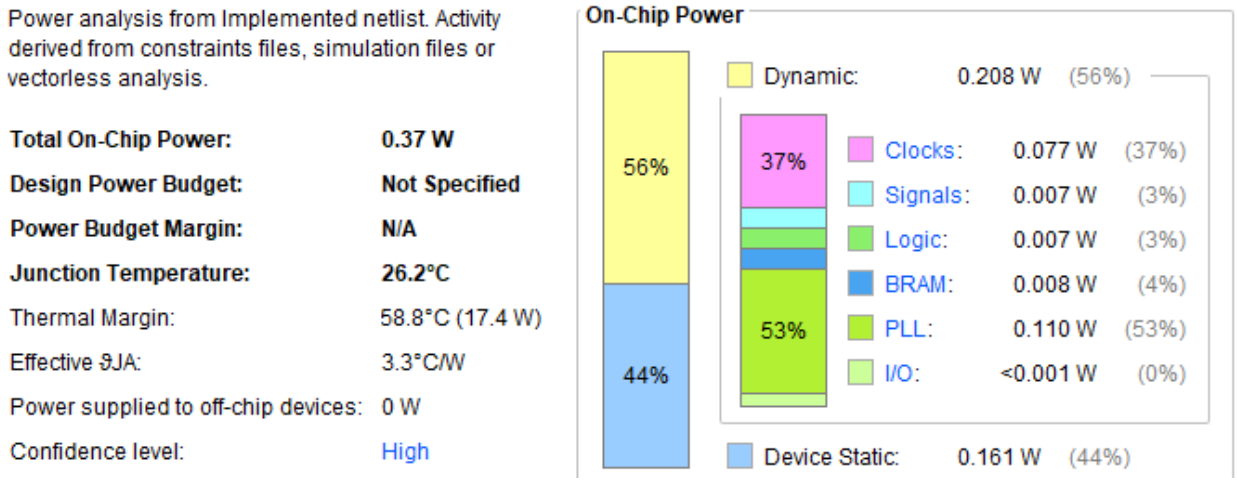
Εικόνα 6.11 Επιλογή αρχείου Saif

Με το που διαλέξουμε το αρχείο saif θα πατήσουμε το κουμπί OK και θα μεταβούμε στο παράθυρο του Report Power.Εκεί θα πατήσουμε το κουμπί OK και θα περιμένουμε μέχρι να τελειώσει η διαδικασία εύρεσης ενέργειας.



Εικόνα 6.12 Επιλεγμένο αρχείο Saif

Με το που τελειώσει θα δούμε στο κάτω μέρος της οθόνης την ενέργεια που θα καταναλώσει το σύστημά μας.



Εικόνα 6.13 Ενέργεια συστήματος

Στην εικόνα 6.14 παρουσιάζεται και η αξιοποίηση των πόρων του FPGA του συστήματός μας.

Resource	Utilization	Available	Utilization %
LUT	15630	133800	11.68
LUTRAM	3	46200	0.01
FF	4922	269200	1.83
BRAM	4.50	365	1.23
IO	83	285	29.12
BUFG	5	32	15.63
PLL	1	10	10.00

Εικόνα 6.14 Πόροι συστήματος

7. Συμπεράσματα

Στους παρακάτω πίνακες παρουσιάζονται ο χρόνος που χρειάζεται κάθε εξάρτημα για να ολοκληρώσει την λειτουργία του και ο συνολικός χρόνος του συστήματος. Αξίζει να αναφέρουμε ότι ο συνολικός χρόνος που χρειάζεται το σύστημα δεν είναι απαραίτητα το άθροισμα των χρόνων των εξαρτημάτων, αυτό συμβαίνει καθώς τα εξαρτήματα λειτουργούν παράλληλα και οι μνήμες FIFO έχουν μια αρχική καθυστέρηση στην αποθήκευση/αποστολή δεδομένων κατά την εκκίνηση του συστήματος και κατά την διάρκεια της πρώτης αποθήκευσης/αποστολής δεδομένων.

Λειτουργία	Κύκλοι	Χρόνος
Λήψη 17 byte από το UART RX	367.820	1.471.280 ns
Αποστολή 16 byte από το UART TX	347.260	1.389.040 ns
Αποθήκευση 8 bit από την FIFO(UART RX/ TX)	1	4 ns
Αποστολή 8 bit από την FIFO(UART RX/ TX)	1	4 ns
Δημιουργία διανύσματος 128 bit προς κρυπτογράφηση/ αποκρυπτογράφηση από το Plaintext Generator	367.822	1.471.288 ns
Έλεγχος μηνύματος από το Message Handler	6510	26.040 ns
Κρυπτογράφηση/αποκρυπτογράφηση δεδομένων τηλεμετρίας και εντολών από το Aes Encoder/Decoder(UART)	15	60 ns
Κατακερματισμός δεδομένων τηλεμετρίας και εντολών από το UART Transmitter Buffer	18	72 ns
Λήψη και αποστολή δεδομένων βίντεο από το Camera Capture	10	41,6 ns
Αποθήκευση δεδομένων βίντεο από την FIFO(Camera)	10	40 ns
Αποστολή δεδομένων βίντεο από την FIFO	1	4 ns
Δημιουργία διανύσματος 128 bit δεδομένων βίντεο από το Camera Buffer	169	675,9 ns
Κρυπτογάφηση και αποστολή δεδομένων βίντεο από το Aes Encoder/Decoder	17	68 ns

Πίνακας 7.1 Χρόνοι και κύκλοι λειτουργίας εξαρτημάτων

Λειτουργία	Κύκλοι	Χρόνος
Κρυπτογράφηση και αποστολή δεδομένων τηλεμετρίας.	715.105	2.860.420 ns
Αποκρυπτογράφηση και αποστολή δεδομένων σταθμού βάσης.	715.105	2.860.420 ns
Κρυπτογράφηση και αποστολή δεδομένων βίντεο.	210.912	843.650 ns

Πίνακας 7.2 Συνολικός χρόνος συστήματος για κρυπτογράφηση δεδομένων τηλεμετρίας, αποκρυπτογράφηση δεδομένων εντολών και κρυπτογράφηση δεδομένων βίντεο

Ο χρόνος που χρειάζεται το σύστημά μας για την κρυπτογράφηση/αποκρυπτογράφηση δεδομένων τηλεμετρίας, εντολών και βίντεο είναι αρκετά μικρός με αποτέλεσμα να μην επιβαρύνει την μετάδοση των δεδομένων. Επίσης η ενεργειακή κατανάλωση του συστήματος είναι αρκετά μικρή με αποτέλεσμα να μην επιβαρύνει την κατανάλωση ενέργειας του UAV . Συνεπώς το σύστημα που φτιάξαμε μπορεί να εφαρμοστεί στον πραγματικό κόσμο για την ασφάλιση των δεδομένων που μεταδίδει ένα UAV.

8. Αρχεία κώδικα

8.1 UART RX

```
-- File Downloaded from http://www.nandland.com
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

entity UART_RX is
  generic (
    g_CLKS_PER_BIT : integer := 115     -- Needs to be set correctly
    );
  port (
    i_reset : in std_logic;
    i_Clk      : in  std_logic;
    i_RX_Serial : in  std_logic;
    o_message_finished     : out std_logic;
    o_new_message_flag : out std_logic;
    o_write_enable : out std_logic := '0';
    o_RX_Byte   : out std_logic_vector(7 downto 0)
    );
end UART_RX;


architecture rtl of UART_RX is

  type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
                     s_RX_Stop_Bit, s_Cleanup);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_RX_Data_R : std_logic := '0';
  signal r_RX_Data   : std_logic := '0';

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
  signal r_RX_DV     : std_logic := '0';

begin



  p_SAMPLE : process (i_Clk)
  begin
    if rising_edge(i_Clk) then
      r_RX_Data_R <= i_RX_Serial;
      r_RX_Data   <= r_RX_Data_R;
    end if;
  end process p_SAMPLE;


  -- Purpose: Control RX state machine
  p_UART_RX : process (i_Clk)
  begin
    if rising_edge(i_Clk) then
       if(i_reset = '1')then
       r_SM_Main <= s_Idle;
        o_message_finished     <= '0';
                          o_new_message_flag <= '0';
        o_write_enable <= '0';
        r_Clk_Count <= 0;
```

```
    r_Bit_Index <= 0;
end if;
 case r_SM_Main is

  when s_Idle =>
   o_message_finished    <= '0';
   o_write_enable <= '0';
   r_Clk_Count <= 0;
   r_Bit_Index <= 0;

   if r_RX_Data = '0' then      -- Start bit detected
    r_SM_Main <= s_RX_Start_Bit;
    o_new_message_flag <= '1';
    o_message_finished    <= '0';
   else
    r_SM_Main <= s_Idle;
   end if;


  -- Check middle of start bit to make sure it's still low
  when s_RX_Start_Bit =>
   if r_Clk_Count = (g_CLKS_PER_BIT-1)/2 then
    if r_RX_Data = '0' then
     r_Clk_Count <= 0;  -- reset counter since we found the middle
     r_SM_Main   <= s_RX_Data_Bits;
    else
     r_SM_Main   <= s_Idle;
    end if;
   else
    r_Clk_Count <= r_Clk_Count + 1;
    r_SM_Main   <= s_RX_Start_Bit;
   end if;


  -- Wait g_CLKS_PER_BIT-1 clock cycles to sample serial data
  when s_RX_Data_Bits =>
   if r_Clk_Count < g_CLKS_PER_BIT-1 then
    r_Clk_Count <= r_Clk_Count + 1;
    r_SM_Main   <= s_RX_Data_Bits;
   else
    r_Clk_Count          <= 0;
    r_RX_Byte(r_Bit_Index) <= r_RX_Data;

    -- Check if we have sent out all bits
    if r_Bit_Index < 7 then
     r_Bit_Index <= r_Bit_Index + 1;
     r_SM_Main   <= s_RX_Data_Bits;
    else
     r_Bit_Index <= 0;
     r_SM_Main   <= s_RX_Stop_Bit;
    end if;
   end if;


  -- Receive Stop bit.  Stop bit = 1
  when s_RX_Stop_Bit =>
   -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
   if r_Clk_Count < g_CLKS_PER_BIT-1 then
    r_Clk_Count <= r_Clk_Count + 1;
    r_SM_Main   <= s_RX_Stop_Bit;
   else
    o_message_finished    <= '1';
```

```
        o_new_message_flag <= '0';

        o_RX_Byte <= r_RX_Byte;
        o_write_enable <= '1';
        r_Clk_Count <= 0;
        r_SM_Main   <= s_Cleanup;
       end if;


      -- Stay here 1 clock
      when s_Cleanup =>
       r_SM_Main <= s_Idle;
       o_message_finished   <= '0';
       o_write_enable <= '0';


      when others =>
       r_SM_Main <= s_Idle;

    end case;
   end if;
 end process p_UART_RX;


end rtl;
```

8.2 UART TX

```
-- File Downloaded from http://www.nandland.com
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity UART_TX is
  generic (
    g_CLKS_PER_BIT : integer := 115     -- Needs to be set correctly
    );
  port (
    i_Clk      : in  std_logic;
    i_reset : in std_logic;
    i_TX_DV    : in  std_logic;
    i_TX_Byte  : in  std_logic_vector(7 downto 0);
    o_TX_Serial : out std_logic;
    i_fifo_empty : in std_logic;
    o_TX_read_enable : out std_logic
    );
end UART_TX;


architecture RTL of UART_TX is

  type t_SM_Main is (s_Idle, s_TX_Start_Bit, s_TX_Data_Bits,
                     s_TX_Stop_Bit, s_Cleanup,s_latch_byte);
  signal r_SM_Main : t_SM_Main := s_Idle;

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
  signal r_TX_Data   : std_logic_vector(7 downto 0) := (others => '0');
```

```
begin


 p_UART_TX : process (i_Clk)
 begin
  if rising_edge(i_Clk) then
      if(i_reset = '1')then
        r_SM_Main <= s_Idle;
          end if;
    case r_SM_Main is

     when s_Idle =>
      o_TX_Serial <= '1';
      r_Clk_Count <= 0;
      r_Bit_Index <= 0;
     if(i_fifo_empty = '0')then
        o_TX_read_enable <= '1';
        r_SM_Main <= s_latch_byte;
      else
        r_SM_Main <= s_Idle;
      end if;


     when s_latch_byte =>
     o_TX_read_enable <= '0';
        if i_TX_DV = '1' then
           r_TX_Data <= i_TX_Byte;
           o_TX_read_enable <= '0';
           r_SM_Main <= s_TX_Start_Bit;

        end if;

     -- Send out Start Bit. Start bit = 0
     when s_TX_Start_Bit =>
      o_TX_Serial <= '0';

      -- Wait g_CLKS_PER_BIT-1 clock cycles for start bit to finish
      if r_Clk_Count < g_CLKS_PER_BIT-1 then
        r_Clk_Count <= r_Clk_Count + 1;
        r_SM_Main   <= s_TX_Start_Bit;
      else
        r_Clk_Count <= 0;
        r_SM_Main   <= s_TX_Data_Bits;
      end if;


     -- Wait g_CLKS_PER_BIT-1 clock cycles for data bits to finish
     when s_TX_Data_Bits =>
      o_TX_Serial <= r_TX_Data(r_Bit_Index);

      if r_Clk_Count < g_CLKS_PER_BIT-1 then
        r_Clk_Count <= r_Clk_Count + 1;
        r_SM_Main   <= s_TX_Data_Bits;
      else
        r_Clk_Count <= 0;

        -- Check if we have sent out all bits
        if r_Bit_Index < 7 then
          r_Bit_Index <= r_Bit_Index + 1;
          r_SM_Main   <= s_TX_Data_Bits;
        else
```

```
          r_Bit_Index <= 0;
          r_SM_Main   <= s_TX_Stop_Bit;
        end if;
       end if;


      -- Send out Stop bit.  Stop bit = 1
      when s_TX_Stop_Bit =>
       o_TX_Serial <= '1';

       -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
       if r_Clk_Count < g_CLKS_PER_BIT-1 then
         r_Clk_Count <= r_Clk_Count + 1;
         r_SM_Main   <= s_TX_Stop_Bit;
       else
         r_Clk_Count <= 0;
         r_SM_Main   <= s_Cleanup;
       end if;


      -- Stay here 1 clock
      when s_Cleanup =>
       r_SM_Main   <= s_Idle;


      when others =>
       r_SM_Main <= s_Idle;

    end case;
   end if;
 end process p_UART_TX;


end RTL;
```

8.3 Message Handler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity message_handler is
 Port (
      i_clk : in std_logic;
      i_reset : in std_logic;
      i_message_flag : in std_logic;
      i_new_message_flag : in std_logic;
      o_message_over : out std_logic);
end message_handler;

architecture Behavioral of message_handler is

type fsm is (reset,start,wait_for_message);
```

```
signal state : fsm := wait_for_message;

begin


process(i_clk)
variable counter : integer := 0;
variable message_counter : integer := 0;
begin
  if(rising_edge(i_clk)) then
          if(i_reset = '1')then
       state <= reset;
          end if;
     case state is
       when reset =>
          counter := 0;
          message_counter := 0;
          o_message_over <= '0';
          state <= wait_for_message;

       when wait_for_message =>
        o_message_over <= '0';
          if(i_message_flag = '1')then
          message_counter := message_counter + 1;
             state <= start;
          else
             state <= wait_for_message;
          end if;

       when start =>
       if(message_counter = 16)then
          message_counter := 0;
          state <= wait_for_message;
       else
          if(i_new_message_flag = '0')then
             if(counter = 6510)then
                counter := 0;
                message_counter := 0;
                o_message_over <= '1';
                state <= wait_for_message;
             else
                counter := counter + 1;
                state <= start;
             end if;

          else
          counter := 0;
             state <= wait_for_message;
          end if;
       end if;
      end case;


  end if;


end process;


end Behavioral;
```

8.4 Plaintext Generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity aes_plaintext_generator is
   Port ( i_clk : in std_logic;
          i_rst : in std_logic;
         -- i_rd_reset : in std_logic;
          o_rd_en : out STD_LOGIC := '1';
          i_rd_valid : in std_logic;

          i_uart_data : in STD_LOGIC_VECTOR (7 downto 0);
          i_message_flag : in std_logic;
          i_aes_busy : in std_logic;
          o_plaintext : out std_logic_vector(127 downto 0);
          o_aes_start : out std_logic
          );
end aes_plaintext_generator;

architecture Behavioral of aes_plaintext_generator is
  signal data_buffer : std_logic_vector(127 downto 0);
  signal send_buffer : std_logic_vector(127 downto 0);
  TYPE machine IS(reset,start,fill_buffer,send_data); --needed states
   SIGNAL state      : machine := reset;                  --state machine
begin

   process(i_clk)
   variable buffer_index : integer := 7;
   begin
      if(rising_edge(i_clk))then
      if(i_rst = '1')then
         state <= reset;
            end if;
         case state is

         when reset =>
            o_aes_start <= '0';
            o_rd_en <= '0';
            data_buffer <=  (others => '0');
            buffer_index := 7;
            state <= fill_buffer;

         when start =>
            --if(i_rd_reset = '0')then
               o_rd_en <= '1';
               state <= fill_buffer;
            --end if;


         when fill_buffer =>
            o_aes_start <='0';
            o_rd_en <= '1';
```

```
        if(i_message_flag = '1')then
                --data_buffer(127 downto buffer_index-7) <= (others => '0');
                o_camera_plaintext <= data_buffer;
                buffer_index := 7;
                state <= send_data;
        end if;
        if(i_rd_valid = '1')then

          if(buffer_index > 127)then

              o_plaintext <= data_buffer;
              o_aes_start <= '0';
              data_buffer(7) <= i_uart_data(7);
              data_buffer(6) <= i_uart_data(6);
              data_buffer(5) <= i_uart_data(5);
              data_buffer(4) <= i_uart_data(4);
              data_buffer(3) <= i_uart_data(3);
              data_buffer(2) <= i_uart_data(2);
              data_buffer(1) <= i_uart_data(1);
              data_buffer(0) <= i_uart_data(0);
              buffer_index := 15;
              state <= send_data;
           else



              o_aes_start <='0';
              data_buffer(buffer_index) <= i_uart_data(7);
              data_buffer(buffer_index-1) <= i_uart_data(6);
              data_buffer(buffer_index-2) <= i_uart_data(5);
              data_buffer(buffer_index-3) <= i_uart_data(4);
              data_buffer(buffer_index-4) <= i_uart_data(3);
              data_buffer(buffer_index-5) <= i_uart_data(2);
              data_buffer(buffer_index-6) <= i_uart_data(1);
              data_buffer(buffer_index-7) <= i_uart_data(0);


            buffer_index := buffer_index + 8;
           end if;
            end if;


      when send_data =>
        if(i_aes_busy = '0')then
           o_aes_start <= '1';
           state <= fill_buffer;
        else
           o_aes_start <= '0';
           state <= send_data;

        end if;



       end case;

     end if;

  end process;
```

```
end Behavioral;
```

8.5 AES Encoder/Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;


entity AES_encoder_decoder is
 Port (

      i_clk : in std_logic;
      i_reset : in std_logic;
      i_camera_data_ready : in std_logic := '0';--Used to tell the aes interface that data is ready to encrypt
      i_uart_data_ready : in std_logic := '0' ;
      i_camera_data : in std_logic_vector(127 downto 0);--Input from the camera
      i_uart_data : in std_logic_vector(127 downto 0);
      o_camera_data_ready : out std_logic;--Output signal,used to tell the transmitter module that data is ready
      o_uart_data_ready : out std_logic;
      o_cam_aes_busy : out std_logic;
      o_uart_aes_busy : out std_logic;
      o_camera_encrypted_data : out std_logic_vector(63 downto 0);--Output data to the transmiter module
      o_uart_encrypted_data : out std_logic_vector(127 downto 0);

      i_dec_input : in std_logic_vector(127 downto 0);
      i_dec_start : in std_logic;
      o_dec_busy : out std_logic;
      o_dec_output : out std_logic_vector(127 downto 0);
      o_dec_rx_data_ready : out std_logic
 );
end AES_encoder_decoder;

architecture Behavioral of AES_encoder_decoder is

--Instance of the aes component
          component encrypt_128_opt                                    --When tesing encryption 128
                    generic (
                              key_len: integer := 128;
                              text_len: integer := 128;
                              nr: integer := 14;
                              nk: integer := 8
                    );
                    port (
                              d_clock : in std_logic;
                               i_reset : in std_logic;
                              d_run: in std_logic;
                              d_done: out std_logic;
                              key: in std_logic_vector(key_len-1 downto 0);
                              input_text: in std_logic_vector(text_len-1 downto 0);
                              output_text: out std_logic_vector(text_len-1 downto 0)
                    );
          end component;

          component decrypt_128 is
          generic (
                    key_len: integer :=128;
                    text_len: integer := 128;
```

```
                nr: integer := 10;
                nk: integer := 4
        );
        port (
                        d_clock : in std_logic;
                         i_reset : in std_logic;
                        d_run: in std_logic;
                        d_done: out std_logic := '1';
                        key: in std_logic_vector(key_len-1 downto 0);
                        input_text: in std_logic_vector(text_len-1 downto 0);
                        output_text: out std_logic_vector(text_len-1 downto 0)
                );
end component;

--Camera encoder signals
signal cam_aes_start : std_logic;
signal cam_aes_input : std_logic_vector(127 downto 0);
signal cam_aes_output : std_logic_vector(127 downto 0);
signal cam_aes_done : std_logic;

--Uart encoder signals
signal uart_aes_start :std_logic;
signal uart_aes_input : std_logic_vector(127 downto 0);
signal uart_aes_output : std_logic_vector(127 downto 0);
signal uart_aes_done : std_logic;

signal reset : std_logic := '0';--If the reset is 0 then the encryption does not start

signal cam_crypto_key : std_logic_vector(127 downto 0):= x"2b7e151628aed2a6abf7158809cf4f3c";
signal uart_crypto_key : std_logic_vector(127 downto 0):= (others => '0');

signal counter : integer :=0;


signal cam_data_buffer : std_logic_vector(127 downto 0);
signal uart_data_buffer : std_logic_vector(127 downto 0);


signal decode_start : std_logic;
signal decode_input : std_logic_vector(127 downto 0);
signal decode_output : std_logic_vector(127 downto 0);
signal decode_done : std_logic;

TYPE machine IS(wait_for_start,wait_for_aes,hold,transmit); --needed states
   SIGNAL cam_enc_state      : machine := wait_for_start;
   signal uart_enc_state : machine := wait_for_start;                --state machine


   SIGNAL dec_state      : machine := wait_for_start;                --state machine

begin

dec_inst : decrypt_128
port map(
        d_clock => i_clk,
        i_reset => i_reset,
                        d_run => decode_start,
                        d_done => decode_done,
                        key => uart_crypto_key,
                        input_text => i_dec_input,
                        output_text => decode_output
);
```

```
cam_enc_inst : encrypt_128_opt
                port map(
                        d_clock     => i_clk,
                        i_reset => i_reset,
                        d_run       => cam_aes_start,
                        key       => cam_crypto_key,
                        input_text  => i_camera_data,
                        output_text => cam_aes_output,
                        d_done      => cam_aes_done
                );

        uart_enc_inst : encrypt_128_opt
                port map(
                        d_clock     => i_clk,
                        i_reset => i_reset,
                        d_run       => uart_aes_start,
                        key       => uart_crypto_key,
                        input_text  => i_uart_data,
                        output_text => uart_aes_output,
                        d_done      => uart_aes_done
                );


 uart_encode_proc : process(i_clk)
 begin
 if(rising_edge(i_clk)) then
        if(i_reset = '1')then

        uart_enc_state <= wait_for_start;


        uart_aes_start <= '0';

          end if;
        case uart_enc_state is
          when wait_for_start =>
            o_uart_aes_busy <= '0';
            o_uart_data_ready <= '0';

            if(i_uart_data_ready = '1')then
              o_uart_aes_busy <= '1';
              uart_aes_start <= '1';
              uart_enc_state <= hold;

            end if;

          when hold =>--Used to treat metastability

            uart_enc_state <= wait_for_aes;


          when wait_for_aes =>
          if(uart_aes_done = '1') then
            uart_aes_start <= '0';
            o_uart_encrypted_data <= uart_aes_output;
            o_uart_data_ready <= '1';
            uart_enc_state <= wait_for_start;
         else
          uart_aes_start <= '0';
          uart_enc_state <= wait_for_aes;
          end if;
```

```
        when others =>


      end case;


    end if;


end process;

cam_encode_proc : process(i_clk)
begin
if(i_reset = '1')then
      cam_enc_state <= wait_for_start;

      cam_aes_start <= '0';

        end if;
   if(rising_edge(i_clk)) then


      case cam_enc_state is
        when wait_for_start =>
          o_cam_aes_busy <= '0';
          o_camera_data_ready <= '0';

          if(i_camera_data_ready ='1' )then
              o_cam_aes_busy <= '1';
              cam_aes_start <= '1';
              cam_enc_state <= hold;
          end if;



        when hold =>

          cam_enc_state <= wait_for_aes;


        when wait_for_aes =>
        if(cam_aes_done = '1') then
          cam_aes_start <= '0';
          cam_enc_state <= transmit;
         else
          cam_aes_start <= '0';
          cam_enc_state <= wait_for_aes;
        end if;

        when transmit =>
        cam_aes_start <= '0';

            o_camera_data_ready <= '1';
              if(counter = 1) then
                 o_camera_data_ready <= '0';
                 counter <= 0;
                 o_camera_encrypted_data <= cam_aes_output(127 downto 64);
```

```
                cam_enc_state <= wait_for_start;
              else
                o_camera_encrypted_data <= cam_aes_output(63 downto 0);
                counter <= counter +1;
                cam_enc_state <= transmit;
              end if;


      end case;



    end if;



end process;


decode_proc : process(i_clk)
begin
if(rising_edge(i_clk)) then
      if(i_reset = '1')then

      dec_state <= wait_for_start;
      decode_start <= '0';

        end if;

      case dec_state is
        when wait_for_start =>
          o_dec_busy <= '0';
          o_dec_rx_data_ready <= '0';

          if(i_dec_start = '1' )then

            o_dec_busy <= '1';
            decode_start <= '1';
            dec_state <= hold;

          end if;

        when hold =>--Used to treat metastability
          -- aes_input<= data_to_aes;
          --reset<='1';
          dec_state <= wait_for_aes;


        when wait_for_aes =>
        if(decode_done = '1') then
          o_dec_output <= decode_output;
          o_dec_rx_data_ready <= '1';

           decode_start <= '0';
           dec_state <= wait_for_start;
         else
          decode_start <= '0';
          dec_state <= wait_for_aes;
        end if;

        when others =>
```

```
        end case;


     end if;

 end process;

end Behavioral;
```

8.6 UART Transmitter Buffer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity uart_transmitter_buffer is
 Port (
      i_clk : in std_logic;
      i_rstn : in std_logic;

      i_tx_begin : in std_logic;
      i_fifo_full : in std_logic;
      o_write_enable : out std_logic;
      i_aes_data : in std_logic_vector(127 downto 0);
      o_uart_tx_data : out std_logic_vector(7 downto 0)
      );
end uart_transmitter_buffer;

architecture Behavioral of uart_transmitter_buffer is

signal tx_data_buffer : std_logic_vector(127 downto 0);


TYPE tx_fsm IS(reset,wait_for_data,hold,fill_fifo,start); --needed states
   SIGNAL tx_state      : tx_fsm := wait_for_data;                  --state machine


begin

tx_proc : process(i_clk)
variable tx_index : integer := 7;
begin
   if rising_edge(i_clk)then
      case tx_state is

         when reset =>
            tx_state <= wait_for_data;
  tx_index := 7;
  tx_data_buffer <= (others => '0');
            o_write_enable <= '0';
```

```
        when wait_for_data =>
          if(i_tx_begin ='1')then
             tx_data_buffer <= i_aes_data;

             tx_state <= fill_fifo;
          end if;

        when start =>
          tx_state <= fill_fifo;
        when hold =>

          tx_state <= start;


        when fill_fifo =>

          if(i_fifo_full = '0')then
            if(tx_index > 127)then
              o_write_enable <= '0';
               tx_index := 7;
               tx_state <= wait_for_data;
            else
               o_write_enable <= '1';
               o_uart_tx_data <= tx_data_buffer(tx_index downto tx_index - 7);
               tx_index := tx_index + 8;
               tx_state <= fill_fifo;

            end if;
          else
            tx_state <= fill_fifo;
          end if;




     end case;

  end if;

end process;



end Behavioral;
```

8.7 Camera Capture

```
-- Engineer: Mike Field <hamster@snap.net.nz>
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Camera_Capture is
  Port (
     --Signals from the camera,used to manage the flow of data to the fpga
      i_clk : in std_logic;
      i_rstn : in std_logic;
      i_pclk  : in   STD_LOGIC;
      i_vsync : in   STD_LOGIC;
```

```
        i_href : in  STD_LOGIC;
        i_d    : in  STD_LOGIC_VECTOR (7 downto 0);--Input byte from the  camera
        o_fifo_wr_en : out std_logic;
        i_fifo_full :in std_logic;
        o_dout  : out  STD_LOGIC_VECTOR (7 downto 0)
        );--Output data to the aes interface
end Camera_Capture;

architecture Behavioral of Camera_Capture is
--This signals are used to hold values from the camera so as to see if a new frame is ready or the rows change
  signal latched_vsync : STD_LOGIC := '0';
  signal latched_href  : STD_LOGIC := '0';



begin

--The main process of the camera control,in it we capture the bytes from the camera and send them out to the AES
interface
capture_process: process(i_pclk,i_rstn)
  begin
     if(i_rstn = '1')then
        latched_vsync <= '0';
        latched_href  <= '0';
        o_fifo_wr_en <= '0';
           end if;
    --data_buffer <= d;
    if rising_edge(i_pclk) then

      if (latched_href = '1' and i_fifo_full = '0') then
        o_fifo_wr_en <= '1';
        o_dout <= i_d;
      end if;

      if(latched_vsync = '1')then
        latched_vsync <= '0';
        latched_href  <= '0';
        o_fifo_wr_en <= '0';
      end if;

      end if;


      -- If vsync=1 then the module is reseted
      if latched_vsync = '1' then
        latched_href <= '0';

      end if;
    --end if;
    --On falling edge clock latch the href and vsync values to read on the rising edge
    if falling_edge(i_pclk) then
      latched_href  <= i_href;
      latched_vsync <= i_vsync;
    end if;
  end process;
end Behavioral;
```

8.8 Camera Buffer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity camera_buffer is
   Port ( i_clk : in std_logic;
          i_reset : in std_logic;
          i_rd_reset : in std_logic;
          o_rd_en : out STD_LOGIC;
          i_rd_valid : in std_logic;
          i_fifo_empty : in std_logic;
          i_camera_data : in STD_LOGIC_VECTOR (7 downto 0);
          i_aes_busy : in std_logic := '0';
          o_camera_plaintext : out std_logic_vector(127 downto 0);
          o_aes_start : out std_logic
          );
end camera_buffer;

architecture Behavioral of camera_buffer is
  signal data_buffer : std_logic_vector(127 downto 0);
  TYPE machine IS(start,fill_buffer,send_data); --needed states
   SIGNAL state       : machine := start;                    --state machine
begin



   process(i_clk)
   variable buffer_index : integer := 7;
   begin
      if(rising_edge(i_clk))then
         if(i_reset = '1')then
      o_aes_start <= '0';
      o_rd_en <= '0';
      data_buffer <=  (others => '0');
      state <= start;
   end if;
         case state is
```

```
        when start =>
          if(i_rd_reset = '0')then
            o_rd_en <= '1';
            state <= fill_buffer;
          end if;


        when fill_buffer =>
          o_aes_start <='0';
          o_rd_en <= '1';
          if(i_rd_valid = '1')then

            if(buffer_index > 127)then
              o_rd_en <= '0';
              o_camera_plaintext <= data_buffer;

              data_buffer(7) <= i_camera_data(7);
              data_buffer(6) <= i_camera_data(6);
              data_buffer(5) <= i_camera_data(5);
              data_buffer(4) <= i_camera_data(4);
              data_buffer(3) <= i_camera_data(3);
              data_buffer(2) <= i_camera_data(2);
              data_buffer(1) <= i_camera_data(1);
              data_buffer(0) <= i_camera_data(0);
              buffer_index := 15;
              state <= send_data;
            else

              data_buffer(buffer_index) <= i_camera_data(7);
              data_buffer(buffer_index-1) <= i_camera_data(6);
              data_buffer(buffer_index-2) <= i_camera_data(5);
              data_buffer(buffer_index-3) <= i_camera_data(4);
              data_buffer(buffer_index-4) <= i_camera_data(3);
              data_buffer(buffer_index-5) <= i_camera_data(2);
              data_buffer(buffer_index-6) <= i_camera_data(1);
              data_buffer(buffer_index-7) <= i_camera_data(0);

            buffer_index := buffer_index + 8;
            end if;

          end if;


        when send_data =>
          if(i_aes_busy = '0')then
            o_aes_start <='1';
            state <= fill_buffer;
          else
            o_aes_start <='0';
            state <= send_data;
          end if;


        end case;

      end if;

    end process;


end Behavioral;
```

9. Βιβλιογραφία

1. Cast, N., 2022. *How Drones Communicate With the Controller?*. [online] Remoteflyer. Available at: <https://www.remoteflyer.com/how-drones-communicate-with-the-controller/> [Accessed 8 June 2022].
2. 911security.com. 2022. *Drone Communication - Data Link*. [online] Available at: <https://www.911security.com/learn/airspace-security/drone-fundamentals/drone-communication-data-link#:~:text=Data%20link%20uses%20a%20radio,altitude%2C%20and%20many%20other%20parameters> [Accessed 1 June 2022].
3. Wiik, J., 2020. *Cybersecurity and cryptographic methods in unmanned systems*. [online] Available at: <https://publications.ffi.no/nb/item/asset/dspace:6791/20-01289.pdf> [Accessed 11 June 2022].
4. Unmanned Systems Technology. 2022. *Wireless Telemetry Systems | FPV Telemetry for Drones, UAV, USV, UGV*. [online] Available at: <https://www.unmannedsystemstechnology.com/expo/wireless-telemetry/#:~:text=Drone%20telemetry%20is%20data%20gathered,as%20the%20aircraft%27s%20power%20source> [Accessed 2 June 2022].
5. En.wikipedia.org. 2022. *Telemetry - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Telemetry> [Accessed 3 June 2022].
6. En.wikipedia.org. 2022. *Advanced Encryption Standard - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Advanced_Encryption_Standard> [Accessed 7 June 2022].
7. En.wikipedia.org. 2022. *Field-programmable gate array - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Field-programmable_gate_array> [Accessed 9 June 2022].
8. HardwareBee. n.d. *Field Programmable Gate Array (FPGA) History - HardwareBee*. [online] Available at: <https://hardwarebee.com/field-programmable-gate-array-fpga-history-applications/#:~:text=In%20the%20late%201980s%2C%20the,patented%20the%20creation%20in%2> [Accessed 3 June 2022].
9. Xilinx. n.d. *Nexys Video Artix-7 FPGA: Trainer Board for Multimedia Applications*. [online] Available at: <https://www.xilinx.com/products/boards-and-kits/1-cfdwic.html> [Accessed 4 June 2022].
10. Digilent.com. n.d. *Nexys Video - Digilent Reference*. [online] Available at: <https://digilent.com/reference/programmable-logic/nexys-video/start> [Accessed 20 May 2022].
11. 2015. *CMOS OV7670 Camera Module 1/6-Inch 0.3-Megapixel Module Datasheet*. [ebook] openhacks. Available at: <https://www.openhacks.com/uploadsproductos/ov7670_cmos_camera_module_revc_ds.pdf> [Accessed 18 May 2022].
12. Fpga4student.com. n.d. *Basys 3 FPGA OV7670 Camera*. [online] Available at: <https://www.fpga4student.com/2018/08/basys-3-fpga-ov7670-camera.html> [Accessed 10 June 2022].
13. Nandland. n.d. *UART in VHDL and Verilog for an FPGA*. [online] Available at: <https://nandland.com/uart-serial-port-module/> [Accessed 10 April 2022].
14. ARAMPATZIS, A., 2022. *Cybersecurity and Drones: How to Address the Security Threats*. [online] The State of Security. Available at: <https://www.tripwire.com/state-of-security/security-data-protection/cybersecurity-and-drones-how-to-address-the-security-threats/> [Accessed 1 May 2022].
15. GitHub. 2019. *GitHub - caiJiYiMei/AES-encryption-and-decryption-VHDL: AES 128 encryption/decryption, AES 128/192/256 encryption module*. [online] Available at: <https://github.com/caiJiYiMei/AES-encryption-and-decryption-VHDL> [Accessed 5 April 2022].
16. En.wikipedia.org. 2022. *Block cipher mode of operation - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation> [Accessed 10 June 2022].
17. Techplayon.com. 2017. [online] Available at: <https://www.techplayon.com/drone-wireless-video-transmission-technologies-challenges-probable-solutions/#Wireless_Technologies_for_Video_Transmission> [Accessed 10 June 2022].
18. Σφακιανάκης, Ν., 2022. *Κατασκευή μη επανδρωμένου αεροσκάφους με την πλακέτα STM32*. Undergraduate. University of West Attica.
19. Σερέμελης, Σ., 2022. *Ανάπτυξη και έλεγχος τετρακόπτερου*. Undergraduate. University of West Attica.
20. En.wikipedia.org. 2022. *List of unmanned aerial vehicle applications - Wikipedia*. [online] Available at:

<https://en.wikipedia.org/wiki/List_of_unmanned_aerial_vehicle_applications> [Accessed 10 June 2022].

21. En.wikipedia.org. 2022. *History of unmanned aerial vehicles - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/History_of_unmanned_aerial_vehicles> [Accessed 10 June 2022].

22. En.wikipedia.org. 2022. *Unmanned aerial vehicle - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle> [Accessed 10 June 2022].

23. Daly, D., n.d. *A Not-So-Short History of Unmanned Aerial Vehicles (UAV) - Consortiq*. [online] Consortiq.com. Available at: <https://consortiq.com/uas-resources/short-history-unmanned-aerial-vehicles-uavs> [Accessed 10 June 2022].

24. Rise Above. n.d. *Applications and Uses for Multirotor Drones*. [online] Available at: <https://riseabove.com.au/pages/uav-applications-and-uses> [Accessed 5 June 2022].

25. Joshi, N., 2017. [online] Allerin.com. Available at: <https://www.allerin.com/blog/10-stunning-applications-of-drone-technology> [Accessed 5 June 2022].

26. 2017. *https://docs.xilinx.com*. 13th ed. [ebook] Xilinx. Available at: <https://docs.xilinx.com/v/u/en-US/pg057-fifo-generator> [Accessed 10 May 2022].

27. *Clocking Wizard v6.0 LogiCORE IP Product Guide*. 6th ed. [ebook] Xilinx. Available at: <https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/clk_wiz/v6_0/pg065-clk-wiz.pdf> [Accessed 10 May 2022].

28. Larry Ewing, lewing@isc.tamu.edu,GIMP, http://www.isc.tamu.edu/~lewing/linux/.

29. Xilinx. n.d. *Vivado ML Overview*. [online] Available at: <https://www.xilinx.com/products/design-tools/vivado.html> [Accessed 5 June 2022].