UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” DIN IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
Domeniul: Calculatoare și Tehnologia Informației
Programul de studii: Calculatoare

PROIECT DE DIPLOMĂ

Coordonator științific:
ș.l. dr.ing. Mircea-Călin MONOR

Absolvent:
Rareș-Flavian ASOFRONIE

Iași, 2025

UNIVERSITATEA TEHNICĂ „GHEORGHE ASACHI” DIN IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
Domeniul: Calculatoare și Tehnologia Informației
Programul de studii: Calculatoare

Sistem de criptare/decriptare(AES) implementat pe FPGA

PROIECT DE DIPLOMĂ

Coordonator științific:
ș.l. dr.ing. Monor Mircea-Călin

Absolvent:
Asofronie Rareș-Flavian

Iași, 2025

Cuprins

Sistem de criptare/decriptare(AES) implementat pe FPGA

Asofronie Rareș-Flavian

Rezumat

Lucrarea de față prezintă implementarea completă a unei soluții ce îmbină partea de hardware cu cea de software pentru criptarea și decriptarea AES (Advanced Encryption Standard) utilizând limbajul hardware Verilog, într-un mediu de dezvoltare HDL, Vivado cu o rulare pe o placă FPGA Boolean Boad cu Spartan-7, iar utilizatorul va putea să trimită datele dintr-o interfață în urma rulării unui script Pyhton.Codul Verilog a fost structurat în module distincte (SubBytes, ShiftRows, MixColumns, AddRoundKey și generatorul de chei de rundă), reunite într-un fișier de design top-level. Pentru a adapta proiectul la placa FPGA, am creat un fișier de constrângeri (AES.xdc) care mapează semnalele către acei pini fizici ai plăcii de care avem nevoie. S-a realizat un testbench în care s-a verificat treptat funcționalitatea codului, iar apoi s-a trecut la etapa de comunicare cu placa când toate valorile au corespuns cu ce trebuia să se afișeze. După sintetizare, implementare și generare a bitstream-ului în Vivado, am încărcat direct configurația pe FPGA, eliminând necesitatea rulării unui testbench în mediu simulat. Interfața dintre PC și FPGA se realizează prin module UART dedicate (uart_tx.v și uart_rx), conexiunea fiind serială. Pe partea software am realizat un script Python care iniţializează comunicaţia serială cu FPGA-ul. Am construit o interfaţă prietenoasă în Tkinter, unde utilizatorul introduce portul serial (de exemplu: COM5), alege modul de funcţionare (criptare sau decriptare) prin butoane radio şi completează câte două câmpuri text pentru cheia AES şi blocul de date, fiecare reprezentat prin 32 de caractere hexazecimale. În acest fel, stările intermediare ale algoritmului AES apar în consolă într-un format clar, inspirat din FIPS PUB 197, permiţând utilizatorului să urmărească pas cu pas evoluţia criptării sau decriptării. Rezumatul subliniază combinarea eficientă a design-ului hardware (Verilog și FPGA) cu dezvoltarea software (Python și Tkinter), creând un sistem complet și modular.

Această lucrare de licență este specială prin caracterul său low-level, în care criptarea și decriptarea AES sunt implementate direct la nivel de biți în Verilog, pe un FPGA Spartan-7, oferind control fin asupra fiecărei operaţii hardware. În paralel, am dezvoltat o interfață prietenoasă în Python/Tkinter ce permite utilizatorului să introducă ușor parametrii şi să urmărească vizual stările intermediare ale algoritmului. Tema criptării devine astfel un proiect complet, de la descrierea circuitelor logice şi încărcarea bitstream-ului, până la comunicarea serială UART şi prezentarea rezultatelor în timp real. Combinația între design-ul hardware performant şi experiența utilizatorului face din acest proiect o realizare integrată și aplicabilă în medii de securitate.

Capitolul 1. Introducere

Lucrarea de față își propune implementarea completă a algoritmului de criptare și decriptare AES (Advanced Encryption Standard) utilizând limbajul hardware Verilog, într-un mediu de dezvoltare HDL, respectiv Vivado. Aceasta acoperă implementarea modulelor Verilog în vederea operațiilor prinicipale și generarea cheilor de rundă, realizarea modulelor de comunicație între PC și FPGA prin sincronizarea corectă de semnale și de asemenea proiectarea unui script din care Pyhton din care utilizatorul poate trimite seturi de date, să primească rezultatele înapoi și să vadă toate etapele de pe parcursul algoritmului. Acest proiect explorează utilizarea ca extensie a plăcii FPGA[1], îmbinând o implementare hardware complexă în Verilog pe FPGA Spartan-7 cu o interfață software prietenoasă dezvoltată în Python cu Tkinter. Problema securității datelor este una actuală, se fac eforturi în mediile de cercetare și industriale pentru a obține un grad cât mai mare de încredere la transmisia datelor. Prin această lucrare se demonstrează fezabilitatea implementării criptării și decriptării AES în logica hardware reconfigurabilă, deschinzând perspective pentru dezvoltări în domeniul securității embedded pe o placă FPGA. Un obiectiv general pentru acest proiect îl reprezintă demonstrarea unei soluții integrate hardware–software, care combină implementarea pe FPGA a algoritmului AES în Verilog cu dezvoltarea unei interfețe prietenoase pentru utilizator în Python cu Tkinter, permițând atât încărcarea şi configurarea parametrilor de criptare/decriptare, cât şi vizualizarea pas cu pas a stărilor intermediare ale algoritmului.

1.1. Contextul și importanța temei alese

Într-un context global în care securitatea informațiilor devine pe zi ce trece din ce în ce mai critică, proiectarea și implementarea algoritmilor criptografici în hardware oferă soluții rapide și greu de compromis. Algoritmul AES este standardul actual pentru criptare simetrică, fiind folosit la scară largă pe întreaga piață de un număr din ce în ce mai mare de oameni. Realizarea sa pe un FPGA permite execuție paralelă și optimizare maximă a performanței.

Proiectul de față se încadrează în acest context, vizând implementarea completă a algoritmului AES în logica reconfigurabilă a unui circuit FPGA. Alegerea acestei direcții de cercetare nu este întâmplătoare: sistemele bazate pe FPGA devin din ce în ce mai populare în aplicații embedded, criptografice sau IoT, datorită flexibilității, performanței ridicate și posibilității de a adapta arhitectura hardware la cerințe specifice.

Într-un peisaj digital în continuă expansiune, cantitatea de date transmise și stocate a crescut exponențial, ceea ce face ca protejarea lor să fie o necesitate, nu doar o opțiune. Importanța temei prezentate este susținută și de nevoia tot mai mare de soluții hardware sigure și rapide pentru criptare, în special în contexte în care resursele sunt limitate (de ex: dispozitive portabile sau comunicații wireless). Implementarea AES pe FPGA asigură nu doar performanțe superioare față de soluțiile software, dar și o barieră suplimentară împotriva unor atacuri cibernetice, inclusiv de tip side-channel[1], dacă designul este realizat corespunzător. Lucrarea își propune să ajute prin a înțelege și aplica o practică a criptografiei în mediile embedded, începând de la nivelul logicii de bază (Verilog HDL), până la a pregăti pe o platformă fizică (placă FPGA Boolean Board cu Spartan-7) în care utilizatorul să aibă un acces mai simplu și prin care să observe rezultatele obținute în urma rulării unui script, după ce codul Verilog a fost rulat pe acea placă FPGA.

[1] Atacul de tip side-channel reprezintă o modalitate prin care un atacator nu încearcă neapărat să spargă algoritmul în sine, ci încearcă să profite de informațiile colaterale scurse în timpul de execuție al algoritmului. Acest canal lateral apare în timpul execuției și include factori precum: timpul de execuție, consumul de energie, radiații electromagnetice - https://en.wikipedia.org/wiki/Side-channel_attack

1.2. Obiectivele generale și specifice ale lucrării

Lucrarea își propune ca obiectiv general să dezvolte și să valideze o implementare completă a algoritmului AES (Advanced Encryption Standard) pe o platformă hardware reccare combină implementarea pe FPGA a algoritmului AES în Verilog cu dezvoltarea unei interfețe prietenoase pentru utilizator în Python cu Tkinter. Proiectul se încadrează în sfera criptografiei aplicate și a proiectării digitale, cu accent pe performanță, modularitate și extensibilitate.

Din acest obiectiv general, putem afirma că există o serie de obiective specifice pe care ni le-am propus, structurate pe etape clare de proiectare și testare pentru a face proiectul reușit și cât mai bine structurat logic:

- **Analiza teoretică a algoritmului AES-128**: înțelegerea etapelor fundamentale ale criptării (AddRoundKey, SubBytes, ShiftRows, MixColumns) și a logicii inverse pentru decriptare, conform standardului FIPS PUB 197 publicat de oficialii de la NIST[2].

- **Proiectarea modulară a componentelor AES în Verilog**: acesta a reprezentat un aspect esențial, unde a avut loc implementarea fiecărei etape sub forma unui modul individual, cu semnale logice bine definite, pentru a facilita instanțierea, reutilizarea și testarea în plan îndepărtat atât independent, cât și în ansamblu.

- **Integrarea modulelor într-un design complet**: construirea modulului de top care coordonează fluxul de date prin toate rundele AES și controlează secvențele de criptare și decriptare gestionându-le comutarea.

- **Dezvoltarea unui testbench pentru verificarea simulării funcționale**: scrierea unui scenariu de test care folosește vectori de test oficiali NIST pentru a valida comportamentul logic al designului. În cadrul acestuia, s-au introdus afișările în consolă ce permit o vizibilitate a corectitudinii valorilor față de cele așteptate în final pentru a spune dacă procesele s-au desfășurat cu succes sau eronat.

- **Pregătirea arhitecturii pentru o implementare viitoare fizică pe FPGA ca o extensie a acestui proiect**: adaptarea structurii logice pentru a putea fi sintetizată și încărcată pe o placă Spartan-7 (Boolean Board).

- **Implementarea comunicării UART între PC și FPGA**: aici s-a realizat integrarea modulelor uart_tx și uart_rx în design pentru trimiterea și recepția datelor.

- **Crearea unui script Python pentru toate rundele AES**: parsarea intrărilor hex, deschidere deschiderea portului serial, trimiterea datelor către FPGA, afișarea în format FIPS PUB 197.

- **Dezvoltarea unei interfețe grafice cu Tkinter**: proiectarea unei ferestre cu câmpuri pentur port, modul de lucru ales, cheie și date, zonă de afișare a rezultatelor.

- **Documentarea clară și coerentă a întregului proces**: elaborarea unei lucrări care să servească drept ghid pentru proiecte viitoare în criptografie și sisteme embedded, de la teorie la implementare și testare.

Realizarea acestor obiective contribuie la consolidarea cunoștințelor teoretice în criptografie a AES și proiectare digitală, precum și la dobândirea de abilități practice în programarea atât hardware, cât și cea software și utilizarea platformelor de simulare FPGA.

1.3. Metodologia de cercetare utilizată

Metodologia adoptată în această lucrare a fost una structurată în mai multe etape consecutive și iterative. Procesul a început cu o etapă de documentare teoretică aprofundată asupra criptării simetrice și a algoritmului AES-128, utilizând ca referință principală documentul oficial FIPS-197 publicat de NIST.

Pe baza acestor tehnici, s-a trecut la proiectarea sistemului, utilizând limbajul Verilog HDL în mediul Vivado, urmată de testarea fiecărui modul individual în simulare. Validarea funcțională a fost realizată cu ajutorul unor vectori oficiali de test NIST, integrați într-un testbench personalizat.

Pe tot parcursul procesului, s-au urmărit bune practici de dezvoltare hardware, sincronizarea cu semnale de ceas, respectarea principiilor de modularitate și utilizarea semnalelor de control (start, done) pentru gestionarea criptării și decriptării. Etapele au fost parcurse iterativ, fiecare versiune a designului fiind îmbunătățită pe baza rezultatelor din simulare.

Această abordare a permis nu doar o implementare funcțională, ci și o înțelegere detaliată a modului în care algoritmii criptografici pot fi transpusi cu succes într-o arhitectură digitală hardware.

1.4. Structura generală a lucrării

Lucrarea este structurată în cinci capitole principale, fiecare având un rol bine definit în conturarea, dezvoltarea și documentarea soluției propuse:

- **Capitolul 1 – Introducere:** prezintă contextul și motivația alegerii temei, obiectivele generale și specifice ale lucrării, metodologia de cercetare utilizată și organizarea generală a conținutului teoretic și practic.
- **Capitolul 2 – Fundamentarea teoretică și soluții similare:** oferă o analiză a conceptelor esențiale din criptografie, cu accent pe criptarea simetrică și algoritmul AES. Se prezintă exemple de implementări din literatura de specialitate, tipuri de aplicații existente, precum și limitările acestora, evidențiindu-se necesitatea unei soluții proprii.
- **Capitolul 3 – Soluția propusă:** detaliază arhitectura sistemului dezvoltat, strategiile de proiectare hardware, modulele Verilog implementate, fișierul de constrângeri pentru FPGA Spartan-7, integrarea modulelor UART, uart_tx și uart_rx[3], descrierea scriptului Pyhton și a interfeței Tkinter[4] pentru introducerea parametrilor și afișarea rundelor AES.
- **Capitolul 4 – Testarea soluției și rezultate experimentale:** descrie scenariile de testare utilizate, modul de validare funcțională a criptării și decriptării AES cu comunicare UART, rezultatele obținute, analiza performanței și eventuale limitări observate.
- **Capitolul 5 – Concluzii:** rezumă realizările proiectului, evidențiază contribuțiile personale și oferă direcții posibile pentru extinderea viitoare a aplicației, cum ar fi suportul pentru moduri de operare avansate AES și integrarea într-un mediu embedded complet.

La final, lucrarea include o bibliografie relevantă și o serie de anexe care conțin capturi de ecran, porțiuni de cod relevante și exemple de rulare, pentru a sprijini înțelegerea soluției implementate.

Capitolul 2. Fundamentarea teoretică și documentarea bibliografică

2.1. Prezentarea conceptelor și teoriilor relevante pentru tema abordată

Când vorbim despre criptografie, gândul ni se duce automat la asigurarea securității unui anumit proiect unde datele transmise nu trebuie să fie vizibile de oricine. În criptografie, procesul de mascare a datelor si informațiilor este reprezentată de criptare, fiind folosită pentru a proteja o gamă de date de accesul neautorizat, asigurând confidențialitate. Folosirea criptării asigură protecția de a copia informații împotriva pirateriei software sau aplicațiilor neautorizate. Rolul acesteia este de a salva informații valoroase și a le transfera prin canale de comunicare, transferul de date reprezentând 2 procese inverse: criptarea și decriptarea. Până a trimite date pe linia de comunicație, acestea sunt criptate. Totodată, restaurarea informațiilor originale de la acele date criptate constă în procesul de decriptare. Există o varietate de modalități de a cripta, metodele respective fiind împărțite în funcție de cheile folosite în metode simetrice și metode asimetrice. Principala diferențiere între aceste metode constă în structura cheilor, criptarea simetrică folosind aceeași cheie atât pentru criptare, cât și pentru decriptare, iar cea asimetrică depinzând de 2 chei diferite, una ce se ocupă de criptare (cheie publică), iar cealaltă de procesul invers (cheie privată).

Procedeul de criptare simetrică se bazează doar pe o singură cheie pentru criptare/decriptare, ceea ce evidențiază rapiditatea cu care are loc. Aceștia sunt ideali când vorbim despre aplicații care necesită criptări la viteze mari sau utilizări în a cripta baze de date sau în a stoca diverse fișiere. Un set de exemple de algoritmi simetrici extrem de frecventați în viața de zi cu zi ar fi: DES[2], 3DES[3], AES[4], Blowfish[5], Twofish[6]

Procedeul de criptare asimetrică folosește 2 chei: o cheie publică pentru a cripta și o cheie privată pentru decriptarea datelor. Acest fapt îl determină să fie mai sigur fiindcă nu este nevoit să partajeze doar o cheie între utilizatorii acestuia. Un set de exemple de algoritmi asimetrici utilizați pentru securitatea informațiilor sunt: RSA[7], ECC[8] , DSA[9]. Mai multe detalii despre toți algoritmii simetrici și asimetrici se pot găsi aici[5].

În această lucrare de licență s-a utilizat un algoritm de criptare simetric, mai exact AES, această alegere venind datorită uzului pe scară largă în aplicații, fiind adoptat ca standard de orgzanizația americană NIST.

Povestea apariției acestui algoritm este important de știut. În trecut, odată cu trecerea timpului, DES ajunsese să fie vulnerabil din cauza unei lungimi a cheii mult prea mici, de 56 de biți. NIST a venit cu recomandarea de a se folosi 3DES, ce constă în aplicarea de 3 ori a algoritmului DES. Acesta a arătat ca este puternic, dar dezavantajul folosirii acestuia a fost faptul că este cam lent în implementări. Aici și-a făcut din nou apariția NIST care a lansat în 1997 propuneri pentru o posibilă înlocuire a acestui algoritm . Inițial s-a plecat cu 21 de propuneri ce au fost reduse odată cu trecerea timpului, ajungându-se să se aleagă algoritmul ce a fost propus de 2 belgieni criptografi,

[2]DES (Data Encryption Standard) unul dintre algoritmii cu cea mai mare vechime, fiind dezvoltat de IBM în anii 1970, în prezent fiind destul de vulnerabil, eliminându-se din uz din cauza cheii de 56 de biți

[3]3DES introdus pentru a rezolva vulnerabilitățile din DES, fiind o execuție triplă a algoritmului DES, cauzând o aplicare mai lentă decât alți algoritmi

[4]AES (Advanced Encryption Standard) considerată una din cele mai sigure metode, standardizat de NIST, adaptabil pentru diverse niveluri de securizare, aceasta fiind folosită în această lucrare de licență

[5]Blowfish algoritm simetric pe 64 de biți, cu o lungime a cheii flexibilă (32-448 de biți), cu o performanță rapidă

[6]Twofish succesorul lui Blowfish, cu o criptare mai puternică algoritm cu bloc pe 128 biți

[7]RSA (Rivest-Shamir-Adleman) una din cele mai cunoscute tehnici de criptare asimetrică, utilizează perechi de chei mari(1024-4096 de biți).

[8]ECC (Criptografia cu curbe eliptice) utilizează proprietăți matematice ale curbelor eliptice pentru a furniza chei de criptare

[9]DSA (Digital Signature Algorithm) algoritm standardizat pentru semnături digitale asigurând autenticitate

Joan Daemen și Vincent Rijmen, care și-au denumit algoritmul Rijndael. Printre criteriile pe baza cărora s-au evaluat toate propunerile pentru AES și-au făcut apariția rezistența la atacuri criptanalitice (securitatea), costurile (complexitatea, licențierea gratuită și liberă) și particularitățile algoritmului (simplitatea, flexibilitatea, ușurința de implementare software și hardware).

AES propune utilizarea algoritmului de criptare pe blocuri în care lungimea blocului este de 128 de biți , iar cheia folosită este variabilă, de 128 de biți, 192 de biți sau de 256 de biți. Toate operațiile din acest algoritm sunt definite sub forma unor operații pe matrice, astfel atât blocul, cât și cheia sunt scrise sub forma de unor matrice.

2.1.1. Implementarea logicii AES pentru criptare și decriptare a datelor la nivel teoretic

Procedeul constă în trecerea unei matrice date ca intrare printr-o serie de operații complexe. La startul rulării cifrului, blocul dat ca intrare se copiază într-un nou tablou ce este denumit stare (în eng. state), primii 4 octeți sunt scriși pe prima coloană, apoi următorii sunt transmiși pe a doua coloană și se continuă cu această logică până la completarea definitivă a tabloului. Acest tablou denumit state este modificat la fiecare pas și se tot furnizează ca ieșire pentru a putea fi preluat la următoarele etape.

Un exemplu concret de pseudocod al algoritmului utilizat în acest proiect poate fi vizualizat în

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

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

   out = state
end
```

Figura 2.1. Algoritmul de criptare AES-128 conform celor de la NIST

Algoritmul de decriptare AES este conceput într-o ordine inversă a operațiilor, folosindu-se și de alte module ce sunt gândite ca fiind inverse intenționat pentru a se obține rezultatul inițial trimis spre criptare, exact cum se poate observa în pseudocodul de mai jos:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                     // See Sec. 5.3.1
      InvSubBytes(state)                      // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                    // See Sec. 5.3.3
   end for

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

   out = state
end
```

Figura 2.2. Algoritmul de decriptare AES-128 conform celor de la NIST

Pe lângă arhitectura internă a algoritmului AES, proiectul îmbină și elemente fundamentale de comunicație serială și interfaționare hardware–software:

2.1.2. Protocolul UART (Universal Asynchronous Receiver/Transmitter)

- Datele sunt transmise bit cu bit pe un singur fir (TX/RX), fără semnal de ceas dedicat.
- Detalii despre frecvența operațiilor și tot ce ține de configurare se poate găsi aici[6]

2.1.3. Modulele Verilog de comunicație UART

- **uart_rx.v**: aici se primesc asincron datele seriale pe lina RX și transformă în octeți validați printr-o mașină de stări finite(FSM). După ce se detectează bitul de start, eșantionează fiecare din cei 8 biți de date , construiește registrul r_Rx_Byte și semnalează cu o_Rx_DV când byte-ul este recepționat complet. La detectarea bitului de stop, se revine în starea de așteptare.
- **uart_tx.v**: emite octeți pe linia TX, utilizând tot un FSM. Când se primește semnalul i_Tx_DV, trece din starea s_IDLE în s_TX_START_BIT pentru a trimite bitul de start, apoi trimite consecutiv toți cei 8 biți de date și un bit de stop în final, iar apoi marchează cu o_Tx_Done sfârșitul transmisiei. Semnalul o_Tx_Active arată perioada în care linia serială este ocupată cu transmiterea de date.

Codul integral folosit la implementarea celor 2 module de comunicație poate fi vizualizat în anexe(vezi Anexa 6.

2.1.4. Scriptul Python de comunicare și control

- Folosește PySerial pentru a deschide portul serial la 9600 baud și timeout 5 s.
- La trimitere, construiește un pachet de 33 octeți (1 octet mod + 16 octeți cheie + 16 octeți date) și îl trimite odată către FPGA.
- Primește secvențe de 16 octeți pentru fiecare rundă AES cu "ser.read()", elimină blocurile nule sau duplicate și le forwardează către interfață.
- Actualizează consola GUI cu etichete ‘round[n].<field>‘ și date hex pentru fiecare pas al algoritmului pentru ușurarea vizualizării rezultatelor și pentru a fi sigur cu certitudine că rezultatul a fost calculat în urma unui proces corect de la început până la final.

2.1.5. Interfața cu utilizatorul în Tkinter

Interfața grafică a fost realizată în Tkinter și organizată pe un `ttk.Frame` cu grid, după cum se poate vedea în capturile de ecran din figurile A.1 A.2 Anexa 3, alături de alte detalii ale componentelor folosite.

2.2. Prezentarea succintă și comparativă privind realizările actuale pe aceeași temă din literatura de specialitate

Dacă trebuie să ne referim la o lucrare relevantă în care se discută implementarea algoritmului AES în Verilog, atunci un exemplu complex este lucrarea prezentată de Chitta Sreevall, Krishna Dharvathu și Dr.M.S.Anuradh, unde se propune la discuție o arhitectură explicită pentru AES, utilizându-se limbajul Verilog. Se analizează 3 versiuni ale acestui algoritm, AES-128, AES-192, AES-256, în care se pune un accent pe ce performanță se obține în final. Luăm drept exemplu implementarea AES-128 în varianta compactă ce a utilizat cu aproximare 2% din LUT-urile disponibile obținându-se un randament de 38,26 Mbps. Totodată, varianta structurală pentru același algoritm a atins un randament de

350 Mbps, deși a utilizat până la 46%. Se evidențiază un anumit compromis între optimizarea pentru dimensiune și cea pentru viteză.

De asemenea, o altă lucrare reprezentativă este cea publicată de Yogendra Singh Sikarwar și N.S. Murty ce subliniază o implementare a algoritmului AES în Verilog, fiind destinată platformei FPGA Spartan-3E

2.3. Analiza tipurilor de soluții/aplicații existente din respectiva categorie a temei

Algoritmul AES este folosit într-o gamă extrem de largă de aplicații de securitate, atât hardware, cât și software. Aceste soluții existente sunt clasificate în 2 categorii relevante: implementări hardware dedicate, în special pe FPGA și implementări software general-purpose.

Soluțiile software includ biblioteci criptografice precum: OpenSSL, PyCryptodome, implementările din limbaje, drept exemplu fiind Java sau Python care sunt folosite în aplicațiile mobile și web.

Există numeroase "calculatoare" AES online care simplifică maxim fluxul de lucru: user-ul introduce cheia AES pe 128 de biți şi blocul de date tot pe 128 de biți, apasă „Encrypt" sau „Decrypt" şi primeşte imediat rezultatul final, fără niciun detaliu despre stările intermediare ale algoritmului. Un exemplu este AES Calculator de la TestProtect [10], unde se introduc cele 32 de cifre hex pentru cheie şi cele 32 de cifre pentru date şi se obţine direct textul criptat sau decriptat

Deşi aceste unelte online sunt foarte utile pentru validări rapide sau demonstraţii punctuale, ele funcţionează precum "cutii negre" şi nu permit nicio vizualizare pas-cu-pas a transformărilor SubBytes, ShiftRows, MixColumns şi AddRoundKey. În contrast, soluţia propusă în această lucrare integrează un front-end Python/Tkinter care nu doar transmite datele către FPGA prin UART, ci şi citeşte şi afişează în timp real fiecare stare intermediară a algoritmului. Mai mult, utilizarea unui FPGA Spartan-7 implică o sincronizare extrem de precisă a semnalelor la nivel de cronometru intern, ceea ce asigură că fiecare bit este procesat în fereastra de timp corectă — un aspect esenţial pentru validarea corectă a logicii AES în hardware.

Soluțiile hardware se bazează direct pe implementarea în circuite logice, folosindu-se FPGA și ASIC. Dacă ar fi să realizăm o comparație între aceste 2 categorii de soluții, am putea lua în calcul câteva criterii:

- Securitate: soluțiile hardware sunt rezistente la atacuri externe, în timp ce soluțiile software sunt destul de vulnerabile.
- Viteză: viteza soluțiilor hardware este una foarte mare datorită execuției paralele, iar în cealaltă categorie viteza este medie/mare, depinde de la caz la caz, fiind secvențială.
- Consumul de energie: destul de redus pentru cele hardware, în comparație cu cele software unde este variabil în funcție de CPU.

Aplicațiile includ o serie de componente:

- Dispozitivele IoT ce necesită criptarea în timp real cu un minim de consum de energie, un exemplu concret ar fi colectarea unor date printr-un senzor și criptarea acestora local folosind AES-128.
- Criptarea de a stoca : SSD-uri, controlere criptate, smart cards
- Asigurarea securizării comunicațiilor în rețele industriale: module de criptare embedded, VPN hardware
- Sisteme militare unde ideea de a cripta hardware date reprezintă o adevărată necesitate, chiar o cerință critică

[10] TestProtect - https://testprotect.com/appendix/AEScalc

Prin ideea implementării AES, putem observa faptul că lucrarea aceasta se încadrează în categoria hardware a soluțiilor, fiind realizată în Verilog pentru FPGA Spartan-7. În principal, ea este destinată aplicațiilor embedded unde securitatea criptografică este extrem de valoroasă, iar datorită structurii modulare cu vectori standard NIST, soluția este gata să fie inclusă în aplicații precum:

- Platforme de cercetare sau educaționale ce simulează atacuri și apărare criptografică
- Sistemele de control industriale
- Dispozitivele IoT ce integrează o nevoie de securitate

2.4. Identificarea lacunelor în stadiul actual și cele referitoare la soluțiile existente

În ciuda faptului că există un număr extrem de mare de lucrări ce vizează algoritmul AES pe FPGA, analiza evidențiază limitări care pot afecta calitatea acestor soluții. Lacunele respective explică nevoia de a dezvolta arhitecturi hardware mai complete și mai ușor de extins pentru aplicațiile reale.

Una din cele mai frecvente aspecte problematice o reprezintă lipsa componentei de decriptare în majoritatea lucrărilor. Majoritatea acestora se axează exclusiv pe procesul de criptare, astfel ignorându-se procesul invers. Se limitează evident aplicabilitatea din scenariile reale, unde o comunicare bidirecțională depinde clar atât de criptare, cât și de decriptare. Totodată, testarea decriptării este esențială în privința unei verificări pentru algoritm pentru a vedea dacă rezultatul procesului invers corespunde cu data trimisă ca dată de intrare.

Puține implementări implică mecanisme pentru extinderea practică, precum integrarea cu protocolul UART îmbinând partea de hardware cu cea de software. Deși se demonstrează o funcționalitate a criptării, lipsa conectivităților externe reduce evident aplicabilitatea în scenariile embedded. Nu sunt tratate pentru observare sincronizarea transmisiei, controlul criptării sau o alocare optimă a resurselor pentru integrarea, care sunt doar câteva din punctele de referință când ne gândim la posibile aspecte din domeniul aplicativ.

O altă chestiune problematică destul de frecvent întâlnită constă în lipsa unui testbench bine definit care să fie capabil să valideze o bună funcționare pentru AES pe FPGA. Cele mai multe dintre implementări se bazează pe verificări vizuale prin folosirea unor semnale în simulator, cum este în cazul Vivado. Fără a avea un testbench riguros, validarea devine destul de subiectivă ce poate conduce la o formarea a unor opinii greșite despre unele semnale.

De aceea, am creat un testbench complet de verificare a logicii algoritmului ce poate fi văzut la Anexa 7, folosit pentru a verifica corectitudinea algorimului pe parcursul procesului, făcut special pentru 2 tipuri de algoritmi, AES-128 și AES-256, care ar putea fi o extensie în viitor cu o comparație între acestea, iar rezultatul simulării poate fi vizualizat în Anexa 8, unde sunt reprezentate toate semnalele folosite pe parcursul codului, iar valorile semnalelor corespund cu valorile corecte pentru procesele de criptare și decriptare.

În majoritatea proiectelor întâlnite, proiectarea AES este destul de slab modularizată , ceea ce face grea înțelegerea codului. Tot acest set de transformări (SubBytes, ShiftRows, MixColumns) sunt, de obicei, integrate direct în fișierul principal. Claritatea este astfel pusă în pericol fără o separare logică adecvată.

Lucrarea de față reușește să reziste în fața acestor limitări printr-o implementare completă în Verilog, contribuind cu un cadru clar, ce include:

- un suport consistent atât pentru criptare, cât și pentru decriptare
- o arhitectură modularizată cu fișiere separate pentru toate transformările
- un design special gândit pentru o extindere cu o interfață UART
- un testbench complex, ce validează rezultatele criptării și decriptării pe baza vectorilor NIST

- un fișier de constrângeri (.xdc) care mapează toate semnalele AES și UART pe pinii plăcii Spartan-7, care poate fi urmărit la Anexa 9
- o interfață prietenoasă în Tkinter (logo-uri, radio-buttoane, entry-uri validate, ScrolledText) ce afișează în timp real fiecare stare de rundă AES, urmând ca model academic de afișaj articolul publicat de cei de la NIST.

2.5. Elaborarea specificațiilor privind caracteristicile așteptate de la aplicație

Proiectarea acestei aplicații hardware pentru criptarea și decriptarea AES pe FPGA implică o definire a unor cerințe clare ce pot asigura o funcționare corectă[7].

Aplicația trebuie să fie capabilă să implementeze complet AES-128, utilizând ca puncte de plecare o cheie de 128 de biți și un bloc de date de 128 de biți. Este necesară respectarea structurii de 10 runde, așa cum se definește în documentul emis de NIST. Fiecare modul trebuie verificat individual pe etape și implementat logic ca tot proiectul integral să funcționeze la capacitate maximă.

Posibilitatea de extindere depinde de claritatea aplicației, fiind nevoie de o proiectare modulară. Fiecare etapă din acest algoritm (transfomările: AES_SubBytes, AES_ShiftRows, AES_MixColumns, AES_AddRoundKey, AES_KeyExpansion) are nevoie de câte un fișier separat pentru a se înțelege exact ce se petrece în cadrul fiecărei operații. Modulul principal de top va instanția fiecare modul corespunzător, existând și un control al fluxului de date.

Pentru o verificare profesională a corectitudinii a acestui algoritm trebuie ca această aplicație să poată fi testată pe baza vectorilor furnizați de NIST, fiind realizate printr-un testbench în Vivado, iar atât rezultatul final, cât și fiecare matrice de valori din fiecare etapă vor trebui să corespundă exact cu acele valori afișate in acel document oficial. După ce s-a realizat această simulare de verificare a tuturor semnalelor, s-a trecut la transpunerea codului pentru a efectua protocolul UART și pentru a începe procesul de schimb de informații între PC și placa FPGA.

Designul aplicației trebuie să fie capabil să fie sintetizat și implementat pe o placă FPGA Boolean Board cu Spartan-7, verificându-se consumul de resurse (LUT-uri, flip-flop-uri) să fie în limite realistice, astfel încât implementarea să fie o posibilitate de utilizare pentru aplicații viitoare embedded reale. Comunicația este configurată cu baud-rate de 9600, iar fiecare transfer constă dintr-un bloc de 33 de octeți: 1 octet pentru modul ales(0 pentru criptare, 1 pentru decriptare), 16 octeți pentru cheie și încă 16 octeți pentru data de intrare.

În urma rulării codului Verilog pe FPGA, utilizatorul va avea ocazia de a trimite date în urma rulării unui script Python și de a primi rezultatele calculate de algoritm. User-ul poate introduce cheia și blocul de date direct din interfața Python realizată cu Tkinter, scriptul trimite modul ales (criptare sau decriptare), pachetul de cheie și date către FPGA, iar acesta răspunde cu stările intermediare și rezultatul final, astfel putând fi văzut tot procesul de calcul, nu doar rezultatul final, printr-o interfață prietenoasă. Întreg acest flux de schimb de date între PC și FPGA se gestionează total automat, oferindu-se astfel o experiență completă pentru utilizator, de la a trimite date, prin procesarea hardware a rundelor AES, iar în final se vor afișa rezultatele în interfața realizată.

Capitolul 3. Soluția propusă

3.1. Reiterarea problemei pe care o rezolvă soluția propusă și a strategiei de rezolvare

Putem să spunem că proiectul implementat răspunde unei probleme în criptografia aplicată, majoritatea soluțiilor algoritmilor AES existente fie rulează exclusiv în mediul software unde nu există niciun instrument de vizualizare a stărilor intermediare, fie se folosesc IP-uri din mediul hardware ce ascund complet logica internă. Totodată, simulările HDL nu reflectă în totalitate condițiile reale de pe FPGA, unde întârzierile sau nesincronizările de semnale pot determina eșecuri de recepție și de transmitere a datelor dorite de a fi furnizate.

Prin urmare, problema rezolvată de acest proiect este crearea unei soluţii care:

- Implementează complet AES-128 în Verilog, modular şi extensibil pentru eventuale AES-192/AES-256.
- Asigură o comunicare serială bidirecţională robustă între PC şi FPGA.
- Oferă, printr-un script Python şi o interfaţă Tkinter prietenoasă, vizibilitate pas-cu-pas asupra fiecărei transformări din cadrul fiecărei runde AES pentru a urmări întreg fluxul de date transmise.

În această eră a comunicațiilor digitale, securitatea tuturor datelor ce sunt transmise prin rețele deschise a devenit o cerință extrem de importantă în multe domenii, de la cele bancare, aplicații bancare: plățile contactless cu cardul unde sunt folosite criptări pentru autentificare și de a proteja datele tranzacțiilor în momentul schimbului cu banca, ATM-uri unde se folosește criptarea la procesarea datelor PIN și pentru tranzacțiile dintre ATM și serverul central al respectivei bănci, până la aplicații uzuale casnice, precum: routerele Wi-Fi pentru criptarea conexiunilor prin protocoale WPA2/WPA3, asigurându-se că acele date trimise între dispozitivele din casă și rețea sunt în siguranță împotriva interceptării, sistemele de supraveghere video ce transmit date video criptate pentru a asigura prevenirea accesului neautorizat la acele fluxuri video, dispozitivele de stocare externă precum stick-uri USB folosite pentru a proteja automat fișierele aflate pe respectivul dispozitiv. Majoritatea acestor date transmise conțin date personale, chiar sensibile, din acest motiv criptarea lor este o rezolvare pentru a se asigura confidențialitatea.

Printre toți algoritmii de criptare, AES este considerat ca fiind un standard internațional datorită rezistenței sale la diverse atacuri și simplității relative de implementare. Cu toate că multe din aplicații utilizează implementări software ale acestuia, toate aceste soluții au unele limitări: de viteză, consumul de resurse sau vulnerabilitatea la atacuri de tip side-channel.

Gândindu-ne la toate aceste limitări, problema propriu-zisă ce este tratată în această lucrare de licență o reprezintă implementarea completă pentru algoritmul AES direct în hardware, folosind un limbaj de descriere hardware Verilog cu o simulare clară pentru a vedea evoluția tuturor valorilor la fiecare etapă trecută.

Strategia propusă presupune:

- desfacerea algoritmului în componente independente: am realizat descompunerea în blocuri diferite, fiecare reprezentând o etapă din procesul de criptare/decriptare (AES_SubByes, AES_ShiftRows, AES_MixColumns, AES_AddRoundKey, AES_KeyExpansion)
- componenta de implementare a tuturor etapelor algoritmului AES în module Verilog: am organizat codul în fișiere separate ce permit o înțelegere și adaptare ușoară pentru posibile extensii pe viitor, precum alte lungimi de chei pentru AES-192 sau AES-256

- validarea funcțională pentru fiecare modul în parte pentru a observa comportamentul adecvat: am construit un set de cazuri de test la simulare pentru a verifica specificațiile AES. Un exemplu concret aplicabil în această situație îl constituie formarea AES_KeyExpansion, unde am verificat dacă toate cele 11 chei de rundă sunt generate corect și ordinea acestora corespunde cu setul de chei care trebuie să rezulte pe parcursul procesului.

- integrarea tuturor acestor module într-un top logic central (AES_TOP): am instanțiat într-un fișier design de top toate modulele necesare procesului, unde sunt controlate fluxurile de date prin transformările AES, sincronizând operațiile și semnalele conform specificațiilor oficiale.

- testarea pentru tot sistemul utilizând un testbench care respectă vectorii declarați de NIST: am conceput acest test pentru a verifica integritatea criptării și decriptării la simulare. Rezultatele ce se obțin se compară cu cele pe care le așteptăm să rezulte în urma simulării.

- testarea designului prin simulare: în urma integrării tuturor componentelor, am testat proiectul în simulatorul Vivado cu ideea de a observa corectitudinea rezultatelor și modul în care reacționează semnalele. Totodată, am realizat un afișaj al stării interne pentru proces, pentru a compara rezultatele intermediare din fiecare rundă, fiecare transformare, utilizând semnale de tip display.

- pregătirea pentru o extensie în viitor, de implementare finală pe FPGA Boolean Board cu Spartan-7: am urmat tot procesul aferent acestei implementări, am sintetizat design-ul. Printr-o mapare a semnalelor și analizând resursele, am dedus că algoritmul AES poate funcționa în condiții reale. Astfel, prin această etapă, consider că am pregătit terenul pentru o conectare fizică și testează comunicația CU PC-ul prin portul serial (în acest caz se folosește COM5).

- sincronizarea semnalelor: în hardware real, schimbările de ceas sau lipsa sincronizărilor pot genera erori de transmisie.

- flexibilitate pentru extindere: pe lângă AES-128, arhitectura modulară permite adăugarea rapidă a suportului pentru AES-192 sau AES-256

- automatizarea testării: am integrat un script Python care trebuie doar să primească modul, cheia şi data de la utilizator, să ruleze toate rundele şi să raporteze automat erorile în cazul în care este nevoie, reducând timpul necesar pentru validarea conservatoare a design-ului sau să afișeze toate valorile rundelor în caz de succes.

- interfaţă prietenoasă pentru utilizator: lipsa unei interfețe în multe proiecte înseamnă o îndepărtare a utilizatorului față de pro respectiv.

3.2. Idei originale, soluții noi

Această lucrare nu se limitează doar la reproducerea specificațiilor algoritmului AES, ci aduce un set de elemente originale ce contribuie la valoarea practică a acestei soluții propuse. Toate aceste contribuții reflectă abordarea atent gândită pentru studiere, integrare embedded, chiar și posibile extensii ce pot fi realizate pentru extinderea acestui proiect.

- Implementare bidirecțională pentru criptare/decriptare: soluția prezentată oferă un suport complet pentru cele 2 operații, cu un control logic prin modulele de legătură corespunzătoare fiecărei transformări.

- Arhitectură modulară prin care toate componentele algoritmului au fost proiectate ca module Verilog separate, ce permit testarea independentă , reutilizarea acestora în alte proiecte viitoare.

- Prezentarea unui design pregătit pentru o extensie embedded cu o placă FPGA: inițial, acest proiect a fost gândit și implementat pentru o simulare a acestui algoritm pentru ca ulterior, în urma funcționării soluției, am realizat că o extensie interesantă ar fi comunicarea între PC și FPGA Spartan-7. Astfel, am adăugat semnale logice noi (în acest caz: start, data_in, data_out, key_in, done) ce pot fi conectate ușor la o interfață UART pentru a accepta comunicația dintre PC și FPGA. Toate aceste adăugări clasifică soluția potrivită pentru o integrare într-o posibilă extensie din viitor în dispozitive embedded ce decriptează informații în timp real sau în proiecte educaționale ce testează protecția din sistemele IoT.

- Validare completă cu vectorii oficiali FIPS Pub 197 și alte seturi de date: pe lângă rularea simulată cu vectorii oficiali NIST, am luat diverse seturi de date (input și cheie) în mod aleatoriu pe care le-am verificat pe mai multe surse de calcul, iar rezultatele corespundeau cu cele din urma simulării din Vivado. Astfel, se oferă o imagine a validării pe mai multe seturi de date, verificându-se dacă implementarea este completă.

- Comutarea celor 2 moduri acordându-i utilizatorului opțiunea de a alege ce proces dorește să folosească, criptarea sau decriptarea.

- Afișarea tuturor valorilor din fiecare rundă și nu doar a rezultatului final, pentru a acorda user-ului șansa de a vizualiza corectitudinea sau dacă are nevoie de o anumită valoare dintr-o anumită rundă pentru a putea calcula altceva.

- Am adăugat opțiunea de a alege tema dorită în interfața Tkinter în urma rulării scriptului Python prin 2 moduri: Light/Dark Mode, oferind utilizatorului șansa de a alege cum vrea să vizualizeze interfața.

- Combinația hardware-software este are ca scop obținerea unui mediu complet de calculare a unui algoritm AES-128. Această colaborare bidirecțională asigură o platformă completă, de la bit-level chiar până la user-experience, fiind ideală pentru a testa avansat.

3.3. Cerințe ale utilizatorului (en., user requirements): așteptările utilizatorilor finali de la sistemul sau aplicația dezvoltată

Cu toate că soluția propusă prezintă un caracter tehnic și academic, putem contura un set de cerințe funcționale pe care acest sistem implementat are obligația de a le îndeplini. Toate aceste cerințe evidențiază așteptările utilizatorului ce dorește să integreze un modul hardware de criptare/-decriptare prin algoritmul AES într-o aplicație embedded.

Cerințele funcționale considerate pentru acest proiect ar fi cele de mai jos:

- Sistemul prezentat trebuie să permită într-un mod bidirecțional criptarea și decriptarea blocurilor de date de 128 biți, așa cum informează standardul AES-128.

- Acest sistem trebuie să prezinte o arhiectură modulară cu fișiere separate pentru fiecare transformare.

- User-ul are obligația de a furniza intrarea și cheia de 128 biți ca intrări externe ale programului, fiind declarate ca semnale de input, nefiind calculate pe parcursul programului, astfel acceptându-se schimbul cheii de criptare sau al datei de intrare oricând se dorește, fiind obligat să introducă date valide pentru a putea efectua procesul dorit.

- Datele rezultate criptate sau decriptate trebuie să fie disponibile pe un semnal de ieșire logică data_out /cipher_out decrypted_out, fără a avea întârzieri mari, acestea fiind folosite în testbench pentru a simula și a verifica corectitudinea programului.

- Design-ul prezentat trebuie să poată fi sincronizat cu un semnal de ceas (clk) de sistem.

- Existența unui fișier de constrângeri adecvat componentelor folosite.
- Crearea unui script Python ce parsează intrările hex, se ocupă cu deschiderea portului serial și începe transferul de date.
- Interfața de control trebuie să fie compatibilă cu un controller extern, putând să fie controlat ușor de către un alt dispozitiv extern , cu semnale clare, ușor de conectat și înțeles de alte componente hardware sau software.
- Sistemul nu trebuie să aibă scurgeri de informații din cauza unei implementări defectuoase a algoritmului, prin oferirea unor rezultate eronate, ci să ofere claritate și un set de rezultate corecte.
- La final, rezultatul va fi afișat clar către utilizator.

Tot acest set de cerințe a fost respectat în momentul proiectării acestui sistem, iar arhitectura permite o extindere a funcționalităților fără modificări considerabile.

3.4. Arhitectura/Modelarea sistemului: descrierea componentelor principale și a interacțiunilor dintre acestea, e.g., prin scheme bloc

Arhitectura generală pentru acest sistem prezentat este modelat după un model destul de clar și structurat, unde fiecare etapă a algoritmului este implementată într-un modul separat pentru a oferi o ușurință de utilizare ulterioară. Dacă ar trebui să alcătuim un plan al componentelor arhitecturale, acestea ar fi: Operația de multiplicare în $\mathrm{GF}(2^8)$ este utilizată în etapa `MixColumns` a algoritmului AES.

- Modulul AES_TOP: Acesta este modulul principal, top-ul, include atât logica de criptare, cât și cea de decriptare comutându-se dinamic între cele 2 moduri pe baza unui semnal "mode". Este componenta cea mai importantă deoarece ajută la gestionarea fluxului de date din procesele de criptare și decriptare.
- Modulele utilizate pentru procesul de criptare
  - Modulul AES_KeyExpansion: Pleacă de la primirea cheii de 128 de biți pentru transformarea inițială din runda 0 și continuă cu cele 10 chei de rundă, câte una pentru fiecare din următoarele 10 runde.
  - Modulul AES_SubBytes: Aici se aplică substituția fiecărui octet din stare folosind S_box-ul din fișierul sbox_mem.txt. Scopul acestei operații este ca fiecare byte să nu fie înlocuit de el însuși.
  - Modulul AES_ShiftRows: Scopul este de a permuta rândurile din matricea de stare astfel: primul rând nu este afectat, celui de-al doilea rând i se aplică o permutare cu o poziție la stânga și cu acest raționament se acționează si pentru următoarele 2 rânduri.
  - Modulul AES_MixColumns: Aplică o transformare liniară folosindu-se multiplicarea în câmpul Galois ($\mathrm{GF}(2^8)$), fiind o multiplicare matriceală. Fiecare coloană se înmulțește cu o matrice specifică, modificându-se ca rezultat poziția fiecărui byte.
  - Modulul AES_AddRoundKey: Se efectuează operația de XOR între starea de la acel moment și cheia de rundă corespunzătoare.
- Modulele de decriptare folosite pe parcursul programului:
  - Modulul AES_KeyExpansion utilizat este același precum cel din procesul de criptare.
  - Modulul AES_InvSubBytes: Se înlocuiește fiecare octet cu o valoare inversă din inv_sbox_mem.txt.

  - Modulul AES_InvShiftRows: Se realizează permutarea rândurilor printr-o rotire inversată spre dreapta pentru a se obține pozițiile dinaintea procesului de criptare.
  - Modulul AES_InvMixColumns: Printr-o aplicare a transformării inverse asupra coloanelor se anulează astfel efectul din urma criptării a operației de AES_MixColumns fiindcă acesta utilizează un polinom diferit în câmpul Galois ($GF(2^8)$)
  - Modulul AES_AddRoundKey: Acest modul rămâne neschimbat deoarece operația XOR este simetrică.

- Pentru a se realiza cu succes legătura între PC și placa Boolean Board, sunt folosite modulele uart_tx și uart_rx, fiind instanțiate în modulul de top al programului. Sunt sincronizate cu ceasul de sistem pentru a evita riscul ca fiecare bit să fie procesat greșit.

- Controllerul intern: Constă în gestiunea celor 10 cicluri de runde, activarea modulelor într-o ordine corectă și generarea semnalului logic done la finalul procesului.

- Scriptul Python cu Tkinter comunică cu hardware-ul ce a fost încărcat pe placă prin portul serial configurabil, trimite pachetele de 33 octeți, iar pe măsură ce FPGA-ul emite date, scriptul le afișează cu efect imediat în consolă, ceea ce reflectă o sincronizare exactă între cele 2 medii.

Arhitectura prezentată este una modulară și extensibilă care permite o trecere de la criptare la decriptare. Spre deosebire de alte lucrări ce proiectează doar AES. Nu se folosește un semnal de control, dar am 2 module diferite pentru criptare și decriptare activate separat. Aici se dovedește suport complet pentru ambele procese, fiecare proces fiind gestionat individual, prin folosirea de semnale de control (done_enc și done_dec). Corectitudinea ambelor se validează prin simulare prin comparația cu vectorii de test recunoscuți de NIST.

3.5. Alegerea tehnologiilor și justificarea pe scurt a alegerii

Pentru a realiza această propunere de proiect a fost necesară selectarea cu atenție a tehnologiilor de dezvoltare, acestea fiind făcute pe baza unor criterii precum performanța, ușurința în procesul de simulare și testare.

Limbajul de descriere hardware, Verilog HDL, este utilizat pentru o implementare a logicii digitale a algoritmului AES, alegerea acestui limbaj fiind motivată de mai multe aspecte. Verilog a permis o proiectare modulară a fiecărei etape din AES, precum și realizarea unui modul de control care are ca rol gestiunea rundei curente, semnalele de activare și semnalul de finalizare.

- Reprezintă un limbaj standard, fiind folosit pe scară largă în proiectarea de circuite digitale.

- Prezintă o sintaxă apropiată de C, motiv pentru care îl face mai accesibil pentru utilizatoricu experiență.

- Este extrem de bine susținut în mediul educațional cu multe exemple pentru proiecte pe FPGA.

Mediul de dezvoltare, Vivado Design Suite, permit simularea, sintetizarea și implementarea circuitului pe FPGA cu Spartan-7 și au fost realizate în Xilinx Vivado Design Suite, versiunea compatibilă cu familia Spartan-7. Alegerea mediului este explicată prin:

- Acordă suport pentru FPGA-uri și plăci educaționale precum Boolean Board.

- Se oferă spre utilizare un simulator logic integrat(Vivado Simulator) ce este esențial pentru a valida comportamentul tuturor modulelor înainte de încărcare pe placă.

- Se prezintă o interfață destul de intuitivă pentru realizarea conectivității modulelor, analizarea semnalelor folosite pe parcursul procesului și generarea fișierelor de configurare (fișierele bistream).

Standardul de referință, NIST AES (FIPS PUB 197), este folosit pentru a valida funcționarea implementării printr-un set de vectori de test, asigurându-se:

- o credibilitate științifică a testării
- o bază solidă pentru a compara rezultatele obținute în urma simulării

Python 3 fost ales pentru a folosi cu claritate sintaxa și tot ansamblul extrem de bogat în biblioteci,iar PySerial s-a dovedit a fi esențial în comunicația serială între PC și FPGA.

Utilizarea Tkinter-ului a fost importantă pentru a construi o interfață grafică prietenoasă, ce oferă câmpuri validate pentru portul serial, modul ales prin butoanele radio, cheie, data de intrare, un buton de "Send" intuitiv, opțiunea de a comuta toată fereastra între 2 moduri de vizualizare Light/Dark Mode și o zonă în care utilizatorul urmărește în timp real toată evoluția tuturor rundelor AES ale algoritmului.

3.6. Identificarea avantajelor și a dezavantajelor metodei alese

Punerea în practică a algoritmului AES direct în hardware, oferă o serie de avantaje extrem de importante față de o posibilă alternativă software. În același timp, toată această abordare vine, de asemenea, și cu anumite provocări ce trebuie analizate cu multă grijă.

Dacă ar fi să alcătuim un set de avantaje ale metodei alese, acesta ar arăta cam așa:

- Modularitate completă: fiecare etapă a algoritmului este proiectată ca modul separat, oferind testare și modificabilitate mai ușoare.
- Extensibilitate viitoare pregătită: tot acest design este gata să fie integrat ulterior cu interfețe externe, ceea ce face posibilă conectarea la aplicații PC sau microcontrollere. Arhitectura modulară în Verilog și structura codului Python permit extinderea ușurată spre implementarea AES-192 sau AES-256.
- Validare riguroasă: Testbench-ul folosit în procesul de simulare include vectori de test furnizați de NIST. Prin această metodă se garantează o corectitudine logică pentru criptare/decriptare, conform specificațiilor FIPS PUB 197. Totodată, această validitate este confirmată și prin afișarea matriceală de la fiecare pas pentru a ușura etapa de debug de pe parcursul întregului proces.
- Execuție paralelă: Un FPGA acceptă execuția în mod paralel a mai multor operații logice. În contextul acestui algoritm, etapele pot fi realizate simultan sau pipeline, iar timpul de procesare rezultat ar fi mult mai mic decât dacă am implementa software secvențial, ceea ce evidențiază o performanță ridicată prin pipeline.
- Combinarea HDL-ului cu un limbaj de nivel înalt precum Python face proiectul flexibil și accesibil la nivel de interfață. Tema devine mult mai interesantă prin această îmbinare a controlului bit-level al FPGA-ului cu dezvoltarea software, ceea ce ajută la vizualizarea tuturor etapelor AES.
- Folosind Tkinter pentru interfață și PySerial pentru protocolul UAR, am obținut un mediu de testare ideal pentru scopuri educaționale.

Dezavantajele metodei alese constau în următorul set:

- Prezența unui debugging mai dificil decât în software: Toate diagnosticurile erorilor în HDL se fac prin simulări și prin a interpreta semnalele logice, ceea ce este mai dificil comparativ cu simpla afișare a mesajelor pe ecran în mediul software.

- Necesitarea unor instrumente specializate: Toate simulările și implementările design-ului au nevoie de un software dedicat precum în acest caz, Vivado, care poate consuma multe resurse ale sistemului (ex: spațiu pe disc, timp de sintetizare).
- Complexitatea de dezvoltare: Proiectarea algoritmului prezent în această temă de licență presupune o bună înțelegere a logicii digitale, arhitecturii hardware, dar și a sincronizării de semnale. Dezvoltarea HDL este mai dificilă și mai amplă decât scrierea unui algoritm în C sau Python, mai ales în faza de debug.
- Procesarea efectivă a algoritmului pe FPGA, sintetizarea, implementarea și încărcarea bitstream-ului poate dura câteva minute până când user-ul poate să introducă date și să primească rezultatele din urma calculării algoritmului rulat pe placă.

Avantajele algoritmului în ceea ce privește performanța, securitatea și predictibilitatea execuției explică alegerea metodei, chiar dacă implementarea în HDL implică o complexitate mai mare în dezvoltare. Un modul AES hardware oferă o soluție mai rapidă și mai sigură decât mare parte din alternativele software.

3.7. Descrierea implementării soluției

Implementarea soluției din această temă s-a format într-un mod progresiv, pas cu pas, structurat, cu o validare riguroasă și un accent pe modularitate. Scopul proiectului este de a reuși criptarea și decriptarea completă a unui bloc de 128 de biți, în conformitate cu specificația AES-128[8]din documentul oficial NIST, prin încărcarea codului Verilog pe o placă FPGA și apoi la rularea unui script Python să se poată adăuga date de utilizator și de a vedea tot procesul cum evoluează de la o etapă la alta. A

La nivel de ansamblu al implementării, sunt prezente următoarele componente:

- un modul de top (AES_TOP.v) care leagă toate celelalte etape din algoritmul AES, cu rol de a îmbina partea de criptare cu cea de decriptare folosind sincronizări de semnale în funcție de modul ales de utilizator.
- un modul de expansiune pentru cheie (AES_KeyExpansion.v) ce generează cele 10 chei de rundă necesare, pe lângă cheia inițială de plecare dată ca intrare
- module special implementate pentru fiecare etapă logică:
  - AES_SubBytes.v
  - AES_ShiftRows.v
  - AES_MixColumns.v
  - AES_AddRoundKey.v
  - AES_InvSubBytes.v
  - AES_InvShiftRows.v
  - AES_InvMixColumns.v
- un controller de runde ce gestionează trecerea dintre runde și astfel se sincronizează modulele
- semnale logice utilizate pentru a controla criptarea (start, done_enc) și decriptarea (done_dec), cu semnale individuale de ieșire (cipher_out, decrypted_out).
- scriptul Python/Tkinter ce comunică cu hardware-ul a fost încărcat pe placă prin portul serial configurabil, trimite pachetele de 33 octeți (mode + key + data_in), iar pe măsură ce FPGA-ul emite date, scriptul le afișează cu efect imediat în consolă, ceea ce reflectă o sincronizare exactă între cele 2 medii

Fluxul de procesare pentru cele 2 operații:

3.7.1. Criptarea

- Când se primește semnalul start, modulul AES pornește procesul de criptare.
- Inițializare și aplicarea AddRoundKey:
  - blocul dat ca intrare este copiat în matricea state
  - acum se aplică prima cheie de rundă asupra datei de intrare din state, printr-o operație de XOR pe fiecare coloană
  - această etapă reprezintă etapa AES_AddRoundKey, unde datele sunt amestecate cu cheia
- Rundele 1-9
  - AES_SubBytes: Fiecare byte din matrice este schimbat cu o valoare corespunzătoare dintr-un tabel de substituție oferind o confuzie în criptare.
  - AES_ShiftRows: Fiecare rând aflat în matrice este deplasat la stânga astfel: rândul 0 este neschimbat, rândul 1 cu 1 poziție la stânga si tot așa cu celelalte 2 rânduri după această regulă.
  - AES_MixColumns: Coloanele sunt tratate ca vectori și sunt înmulțite cu o matrice fixă din câmpul Galois ($GF(2^8)$), astfel crescând securitatea criptării.
  - AES_AddRoundKey: Se aplică cheia de rundă curentă din acel moment prin operația XOR, această cheie fiind diferită pentru fiecare rundă.
- Această rundă din final este aproximativ identică cu cele anterioare, fiind o excepție de care trebuie neapărat să se țină cont, neaplicarea AES_MixColumns. Se execută:
  - AES_SubBytes
  - AES_ShiftRows
  - AES_AddRoundKey

  Aceasta ajută la păstrarea simetriei și facilitează criptarea.
- După ce se finalizează runda finală, conținutul matricei state este copiat în vectorul de ieșire out, reprezentând blocul criptat cipher_out ce conține valoarea criptată finală.
- Rezultatul se livrează prin semnalul cipher_out, iar done_enc este activat.

3.7.2. Decriptarea

- Aceasta folosește în mod similar o cheie extinsă RoundKeys[10] până la RoundKeys[0] într-o ordine inversă.
- Sunt aplicate transformările inverse: AES_InvShiftRows, AES_InvSubBytes, AES_AddRoundKey și AES_InvMixColumns.
- Ultima rundă inversează doar AES_ShiftRows, AES_SubBytes și AES_AddRoundKey
- Rezultatul obținut este memorat în decrypted_out, iar done_dec se activează.

Toată testarea funcțională de pe parcurs s-a format în modulul AES_TB.v, fiind testbench-ul din acest proiect cu rol în verificarea treptată a codului Verilog, iar ulterior totul a trecut la faza de implementare pe placă după ce ne-am asigurat că totul a funcționat complet în simulare. Acesta simulează cu un set de plecare format dintr-o dată și o cheie, comparând rezultatele finale cu cele afișate de vectorii de test oficiali NIST.

Testarea urmată prin simulare confirmă o funcționalitate pentru ambele procese, iar astfel face ca toată această proiectare a algoritmului să fie potrivită pentru extensii viitoare ale aplicațiilor embedded ce se pot realiza.

3.7.3. Comunicația UART

- Au fost folosite 2 module Verilog dedicate, uart_tx și uart_rx, care se ocupă cu trimiterea și recepția serială de date.

- Este un protocol asincron, deci nu necesită un semnal de ceas separat, condiția este ca emițătorul și receptorul să aibă același baud-rate configurat (în acest proiect, baud-rate-ul ales a fost 9600).

- UART începe să trimită câte un bit pe rând, totul fiind serial. Majoritatea plăcilor FPGA au un modul UART incorporat, astfel încât totul a fost ușurat pentru a realiza legătura dintre cele 2 componente.

3.7.4. Python/Tkinter

- Scriptul Python folosește PySerial pentru a deschide și configura portul serial și pentru a reuși citirea/scrierea pachetelor UART.

- Interfața grafică Tkinter include câmpuri pentru fiecare componentă necesară de a fi introdusă de utilizator pentru port, modul de operare, cheia și blocul de date, plus butonul "Send" ce lansează comunicația.

- Rezultatele și stările intermediare ale rundelor AES sunt afișate într-o fereastră ce se actualizează după ce se recepționează fiecare byte.

3.8. Exemple și alte elemente de ajutor

3.8.1. Componente software:

Am ales Python 3 ca limbaj principal pentru scriptul de control şi interfaţă datorită simplităţii sintaxei și prin ecosistemul bogat de biblioteci.

Tkinter a fost folosit pentru GUI deoarece acesta face parte din librăria standard Python, nu necesită instalări suplimentare şi permite realizarea rapidă a unei interfeţe funcţionale, cu câmpuri validate şi widget-uri care să aibă un comportament "prietenos" față de utilizator.

Pentru comunicaţia serială, PySerial s-a impus natural ca standard în Python, oferind o interfaţă clară pentru deschiderea portului, configurarea baud-rate-ului şi citire/scriere de octeţi.

Biblioteca Pillow (PIL) a fost integrată pentru încărcarea şi redimensionarea logo-urilor, contribuind la aspectul profesional al aplicaţiei fără a creşte semnificativ dimensiunea pachetului final.

Dacă ar fi să identificăm anumite limitări ale aplicației create, în partea de software, totul rulează în Python/Tkinter la 9600 baud, ceea ce înseamnă că interfaţa poate arăta stări noi doar la câteva zeci de mesaje pe secundă şi poate părea uşor înţepenită dacă scriptul preia multe date odată. Dacă se încarcă un log foarte mare (mii de linii pe fiecare rundă), memoria scriptului poate crește semnificativ și interfața începe să dea rateuri.

3.8.2. Componente hardware:

Pentru realizarea prototipului am folosit placa Boolean Board cu Spartan-7, care oferă un oscilator intern de 100 MHz şi o interfaţă USB directă pentru comunicaţia serială. Am inclus pe parcursul rulărilor pe placă câteva LED-uri de status conectate la semnalele start, done, tx şi rx, pentru a oferi feed-back vizual asupra stării FPGA-ului. Cablul micro-USB asigură atât alimentarea, cât şi transmisiunea datelor între PC și FPGA fără componente adiționale, așa cum se poate observa din Figura 3.1.

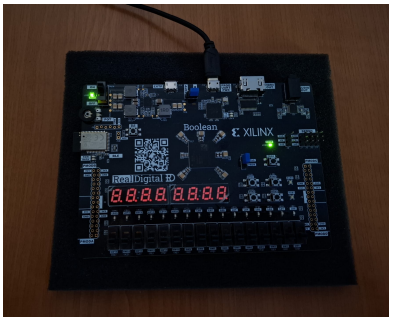


Figura 3.1. Alimentarea plăcii Boolean Board cu Spartan-7[11]

Schema bloc prezintă fluxul end-to-end: aplicația Python/Tkinter de pe PC inițiază un pachet serial prin portul USB, care ajunge la uart_rx în FPGA. Apoi, datele sunt preluate de modulul AES_TOP, unde trec prin etapele SubBytes, ShiftRows, MixColumns, AddRoundKey și KeyExpansion, controlate de FSM-ul intern. După procesare, uart_tx preia fiecare octet de rundă şi uşa trimite înapoi către PC, iar LED-urile de status indică progresul. Acest design modular facilitează adăugarea rapidă a altor periferice sau a unor extensii ale protocolului UART. Tot procesul reprezentat de schimbul de informații dintre PC și FPGA se poate găsi la Anexa 5.

Pe parcursul implementării proiectului de licență, au fost realizate simulării în interiorul Vivadoului pentru a fi siguri de corectitudinea modulelor folosite, iar abia apoi s-a trecut la etapa de rulare a codului pe FPGA și prin a începe tot procesul de comunicare și de transfer de date. Rezultatul simulării poate fi vizualizat în Anexa 8, unde sunt reprezentate toate semnalele folosite pe parcursul codului, iar valorile semnalelor corespund cu valorile corecte pentru procesele de criptare și decriptare.

Ca limitări, procesul de sintetizare și încărcare a bitstream-ului pe FPGA durează câteva minute deoarece conține extrem de multe semnale de sincronizat, ceea ce încetinește foarte mult ciclul de dezvoltare și testare comparativ cu simple scripturi software. FPGA-ul Spartan-7 oferă resurse suficiente pentru AES-128, însă spațiul rămas (LUT, BRAM) este limitat, îngreunând extinderea rapidă la AES-256 sau adăugarea de pipeline-uri suplimentare. Fără un analizor logic dedicat conectat la pini, erorile interne de semnalizare și de sincronizare sunt greu de diagnosticat, fiind nevoie de recunoaștere manuală în log-urile Python sau in log-urile Vivado dacă erorile sunt din urma rulării codului pe FPGA.

Capitolul 4. Testarea aplicației și rezultate experimentale

4.1. Punerea în funcțiune/lansarea aplicației,elemente de configurare sau instalare

Vom dezvălui pașii întregului proces de rulare al aplicației prezentate din acest proiect de licență. Pentru a programa FPGA-ul și a lansa apoi aplicația software, următoarea secțiune descrie pas cu pas întregul flux de lucru în Vivado Design Suite, de la creare proiect până la generarea şi descărcarea bitstream-ului pe placa Spartan-7.

4.1.1. Deschiderea Vivado și crearea unui proiect nou

La pornirea Vivado, din pagina de start alegeţi "Create Project". Vi se va deschide un wizard în câțiva pași:

- Project Name Location: introduceţi un nume descriptiv, de exemplu AES_TOP, și alegeți un folder de lucru (de ex: C:<user>_TOP).
- Project Type: selectaţi „RTL Project" și debifaţi opțiunea „Create project subdirectory" dacă doriţi structura implicită.
- Add Sources: apăsaţi pe „Add Files...", navigaţi la directorul cu fișierele Verilog (AES_TOP.v, AES_SubBytes.v, uart_rx.v etc.) și adăugaţi-le. Nu bifaţi testbench-ul aici, doar codul de design.
- Add Constraints: apăsaţi „Add Files...", selectați fișierul de constrângeri .xdc în care aţi mapat pinii UART și semnalele AES, apoi asiguraţi-vă că sunt la „Top" și apăsaţi „Finish".
- Add Simulation Sources: în Flow Navigator, sub Project Manager, daţi click pe Add Sources, alegeţi „Add or Create Simulation Sources", apoi apăsaţi Add Files... şi selectaţi fişierul vostru de test (de ex. AES_TB.v). Asiguraţi-vă că tipul este setat pe Simulation (nu pe RTL), apoi finalizaţi cu Finish. Aici am folosit un fișier de testbench pentru verificarea treptată a corectitudinii codului.
- Simulation Settings: din aceeaşi fereastră, verificaţi sub Simulation Settings că top-level-ul de simulare este AES_TB şi că toate modulele necesare apar în lista de Simulation Sources.

4.1.2. Sintetizarea design-ului

După închiderea wizard-ului, în panoul din stânga (Flow Navigator) faceţi click pe "Run Synthesis". Vivado va afișa progresul în jos, în console log.

La final, va apărea o notificare „Synthesis Completed".

Puteţi deschide raportul de utilizare a resurselor (Reports → Utilization) și cel de timing (Reports → Timing Summary) pentru a verifica dacă toate constrângerile sunt satisfăcute.

4.1.3. Implementarea design-ului

Imediat sub etapa de sintetizare, selectaţi "Run Implementation". Aceasta aliniază logica sintetizată la efectivele elemente fizice ale FPGA-ului și optimizează traseele de semnal.

La sfârșit, veţi primi mesajul „Implementation Completed".

Puteți să deschideţi raportul de „Post-Implementation Timing" pentru a confirma că toţi timpii critici (setup, hold) sunt în limite acceptabile.

4.1.4. Generarea bitstream-ului

După ce implementarea s-a încheiat cu succes, faceţi click pe “Generate Bitstream”.
Vivado va concura să construiască fișierul .bit care conține configurația completă a FPGA-ului.
Când procesul se termină, aplicația va oferi butonul „Open Hardware Manager”.

4.1.5. Programarea FPGA-ului

Deschideţi Hardware Manager (Flow Navigator → Open Hardware Manager) şi conectați placa prin cablul micro-USB. Vivado detectează automat dispozitivul Spartan-7 conectat.

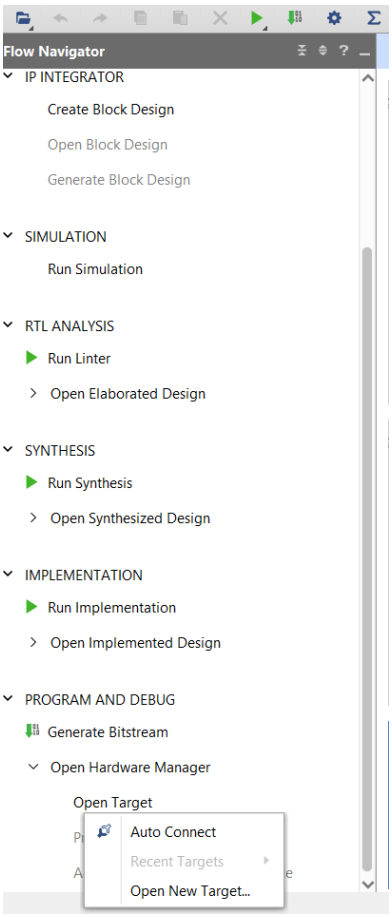
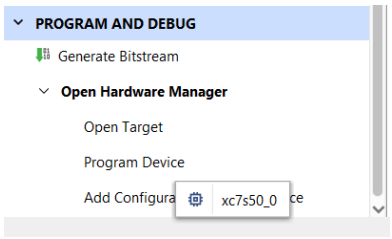


Figura 4.1. Conectare automată cu placa[12]

Click pe “Program Device”.



În fereastra care apare, alegeţi bitstream-ul generat (AES_TOP.runs/impl_1/top.bit) şi apăsaţi “Program”.

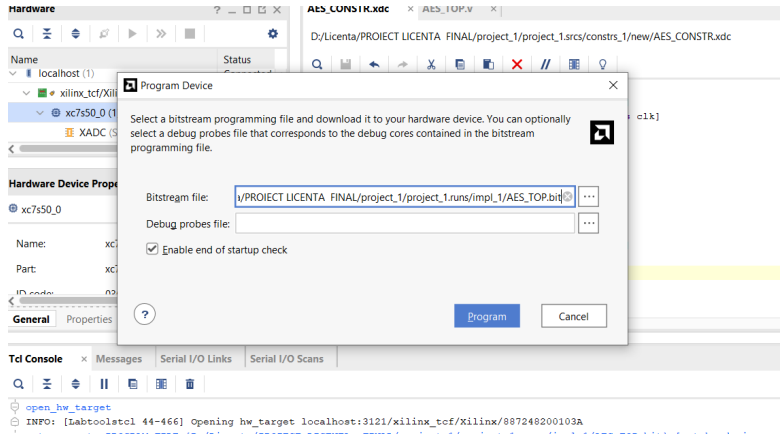
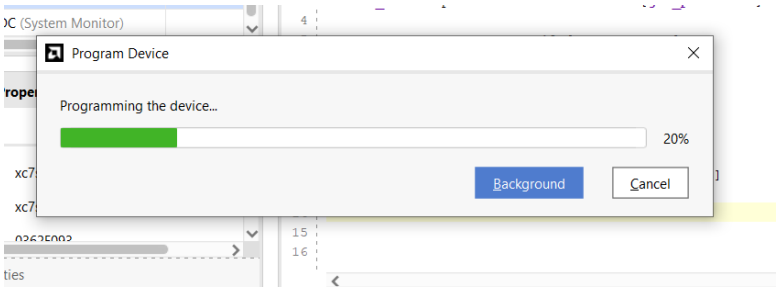


Figura 4.2. Dacă avem deja locația fișierului .bit la Bitstream file, începe transferul spre FPGA.



LED-urile de status de pe placă vor clipi, indicând descărcarea fișierului de configurare.

La acest punct, FPGA-ul este gata să primească comenzi prin UART și să execute AES-ul în hardware. În continuare veţi trece la mediul software (PyCharm sau linie de comandă) pentru a lansa scriptul Python care comunică cu FPGA-ul și afișează rezultatele pas cu pas în interfața Tkinter.

4.1.6. Deschide proiectul existent în PyCharm Community Edition

Structura proiectului realizat în PyCharm este evidențiată în Figura 4.3,

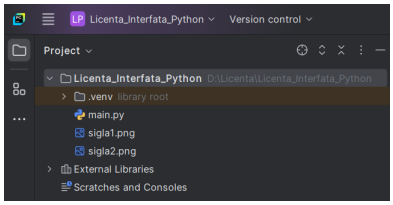


Figura 4.3. Structura proiect[13]

4.1.7. Rulează aplicația

Apasă butonul verde “Run Main”. Se va deschide fereastra realizată cu Tkinter, iar în GUI completează câmpul "Serial port", selectează modul (Criptare/Decriptare), introdu cheia și blocul de date în hex și apasă "Send". Această fereastră poate fi analizată în figurile A.1 A.2 Anexa 3.

4.1.8. Rezultatul final în urma rulării

Plecând de la setul de date oficial postat de cei de la FIPS PUB 197, cu exemplul de set de date (key = 000102030405060708090a0b0c0d0e0f, plaintext = 00112233445566778899aabbccddeeff), am realizat o rulare în interfață pentru a observa toate valorile obținute pe parcursul transformărilor.

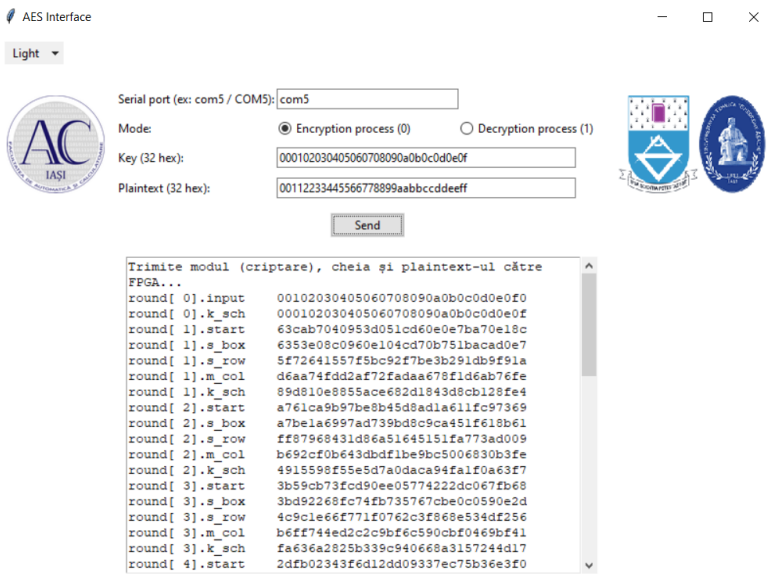


Figura 4.4. Exemplu de rulare cu date oficiale conform NIST în Light Mode[14]

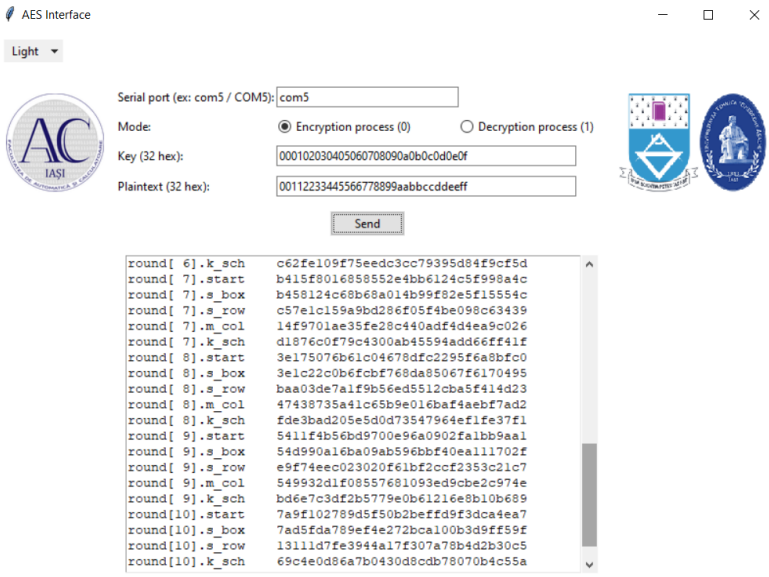


Figura 4.5. Exemplu de rulare cu date oficiale conform NIST în Light Mode[15]

Se poate observa că la finalul algoritmului se obține valoarea dorită, conform valorilor oficiale luate din articol postat de cei de la NIST din Anexa 4.

4.2. Testarea sistemului (hardware/software)

În cadrul testării complete a sistemului, s-a pornit de la verificarea componentei software înainte de orice interacţiune cu FPGA-ul. Am rulat codul Python în PyCharm, cod pe care îl găsiţi integral în Anexa 1, am verificat funcţionarea interfeţei Tkinter: validarea câmpurilor de intrare, comutarea între modurile "Criptare” şi "Decriptare” şi afişarea în timp real a rundelor din fereastră. Am testat mai întâi cu un port serial virtual (de exemplu cu un emulator COM) şi am comparat rezultatele obținute cu cele generate de o bibliotecă standard Python (PyCryptodome), pentru a mă asigura că logica de transmitere este una stabilă și că nu vom avea probleme când se va ajunge să se realizeze și legătura cu componenta Verilog.

Paralel, în Vivado am folosit testbench-ul AES_TB.v, atașat în Anexa 7, pentru simulare comportamentală: am încărcat vectorii oficiali NIST, am urmărit semnalele din interior şi am confirmat

că fiecare ieşire de rundă se potriveşte exact cu valorile aşteptate. Capturile de forme de undă le-am inclus tot în Anexa 8, pentru a avea șansa de a demonstra că design-ul HDL respectă constrângerile și funcţionează corect până la nivelul de granulaţie pe biţi. În final, cu bitstream-ul rulat pe placa cu Spartan-7, am conectat cablul micro-USB şi am trimis prin interfaţa Python mai multe teste decriptate şi criptate succesiv.

4.3. Aspecte legate de încărcarea procesorului, memoriei, limitări în ce privește transmisia datelor/comunicarea

În cadrul lucrării de licență, sistemul dezvoltat pentru interfața AES între PC și FPGA implică o serie de aspecte legate de încărcarea procesorului, memoriei și limitările asociate comunicării. Procesorul PC-ului este utilizat pentru a gestiona interfața grafică bazate pe Tkinter și pentru procesarea minimă a datelor (validări hexazecimale, conversii și afișări). Încărcarea procesorului rămâne scăzută, deoarece majoritatea operațiilor de calcul intensive (algoritmul AES) sunt delegate FPGA-ului, reducând astfel cerințele de procesare pe partea software. Memoria utilizată este redusă, fiind limitată la stocarea datelor de intrare (cheie de 32 de caractere hex și text de 32 de caractere hexazecimale, reprezentând 16 octeți fiecare) și a rezultatelor intermediate afișate în interfață.

Limitările în ceea ce privește transmiterea datelor și comunicarea se referă, în principal, din utilizarea unei conexiuni seriale cu o viteză de 9600 ca frecvență a operațiilor. Această rată de transfer impune o latență semnificativă, estimată la aproximativ 13.3 ms per bloc de 16 octeți, ceea ce duce la un timp total de aproximativ 146 ms pentru cele 11 blocuri citite în procesul de decriptare. De asemenea, stabilitatea conexiunii seriale poate fi afectată de erori de sincronizare sau deconectări, așa cum s-a observat în cazurile de timeout raportate (ex. "Timeout or connection lost").

Pentru a evalua performanța sistemului, a fost implementată o modificare în cod prin adăugarea măsurării timpului de execuție al funcției send_data(). Această metrică, afișată în interfață ca "Timp de execuție", reflectă durata totală a procesului de transmitere a datelor către FPGA, procesarea acestora și primirea rezultatelor, inclusiv latența serială și overhead-ul Tkinter. Rezultatul experimental obținut, de exemplu 0.928 secunde pentru decriptare, indică faptul că comunicarea serială domină timpul total, cu contribuții minore din procesarea software.

4.4. Datele de test

În această secţiune sunt prezentate datele de test folosite pentru implementarea hardware–software a AES-128. Pentru transparenţă şi reproducibilitate, în Anexa 4 am inclus setul complet de vectori oficiali NIST, alături de ieşirile generate pentru fiecare rundă AES, astfel încât cititorul să poată verifica exactitatea şi consistenţa rezultatelor pe parcursul fiecărei runde.

4.5. Aspecte legate de fiabilitate/securitate/scalabilitate

Sistemul dezvoltat pentru implementarea algoritmului AES pe o platformă FPGA, însoțit de o interfață grafică bazată pe Python (folosind Tkinter) și o conexiune serială pentru comunicare, ridică câteva aspecte legate de fiabilitate, securitate și scalabilitate. Aceste aspecte sunt esențiale pentru evaluarea performanței și aplicabilității practice a proiectului, bazat pe documentele furnizate: codul Verilog (AES_TOP.v), constrângerile hardware (AES_CONSTR.xdc) și scriptul Python (main.py).

Fiabilitatea acestui sistem prezentat depinde în mare măsură de stabilitatea comunicării seriale dintre PC și FPGA, configurată la 9600 baud conform parametrului CLKS_PER_BIT = 10416 din modulul AES_TOP.v (100MHz / 9600 = 10416.66, aprox. 10416). Eroarea de timeout raportată în main.py (ex: "Timeout or connection lost") indică o vulnerabilitate a conexiunii, care poate fi cauzată de linia serială, deconectări accidentale sau sincronizarea defectuoasă între ceasul FPGA (100 MHz) și rata de baud. De asemenea, sincronizarea ieșirilor intermediare ale rundelor AES în registrele Verilog asigură consistența rezultatelor, însă lipsa unui mecanism de verificare a integrității datelor primite (ex: compararea cu un hash) limitează fiabilitatea în scenarii cu interferențe.

Securitatea este un punct forte al proiectului, datorită implementării hardware a algoritmului AES pe FPGA, care reduce riscul expunerii cheilor sau a datelor în memoria software. Modulul aes_key_expansion.v generează cele 11 chei de rundă, cheia inițială plus încă alte 10 chei folosite pe parcursul rundelor (11 * 128 = 1408 biți) folosind un document S-box stocat local (sbox_mem.txt) și constantele Rcon, minimizând manipularea cheii în afara hardware-ului. Deoarece FPGA-ul este un dispozitiv hardware dedicat, datele procesate (cheie de 128 biți și text de 128 biți) rămân izolate de alte procese ale sistemului, oferind un nivel suplimentar de protecție împotriva atacurilor software. Folosirea AES pe FPGA, optimizată pentru securitate, face proiectul potrivit pentru aplicații precum criptarea datelor sensibile în dispozitive IoT sau sisteme embedded. Timpul de execuție raportat (ex: 0.928 secunde) poate fi redus prin accelerarea comunicării seriale, permițând procesarea în timp real a datelor criptate. Sistemul poate deveni un exemplu de utilizare a criptării hardware, demonstrând cum FPGA-urile pot fi exploatate pentru a atinge standarde de securitate ridicate. Aceste îmbunătățiri valorifică pe deplin scopul criptării AES, transformând proiectul într-o soluție potrivită pentru scenarii în care securitatea datelor este prioritară.

Scalabilitatea proiectului tău, care implementează algoritmul AES pe o platformă FPGA cu o interfață grafică bazată pe Python (main.py) și comunicare serială (AES_TOP.v), se referă la capacitatea de a gestiona volume mai mari de date sau de a extinde funcționalitatea fără modificări majore. Aceasta este limitată de mai mulți factori. Creșterea vitezei seriale (ex: 115200 baud) sau trecerea la o interfață mai rapidă ar putea îmbunătăți performanța, dar necesită modificări ale constrângerilor hardware (AES_CONSTR.xdc) și a modulelor UART (uart_rx.v, uart_tx.v). De asemenea, arhitectura actuală procesează doar un bloc de 128 biți pe ciclu, ceea ce limitează scalabilitatea la volume mari de date fără paralelizare.

Proiectul, așa cum este configurat, este adecvat pentru prototipuri și aplicații educaționale, demonstrând funcționalitatea AES pe FPGA cu o interfață grafică prietenoasă, inclusiv suport pentru dark/light mode și afișarea timpului de execuție. Totuși, pentru utilizări reale, îmbunătățirile în fiabilitate (ex. gestionare avansată a erorilor), securitate (ex. criptare a datelor în tranzit) și scalabilitate (ex. interfețe mai rapide) sunt esențiale pentru a răspunde cerințelor aplicațiilor practice.

4.6. Rezultate experimentale

Analiza experimentală a sistemului dezvoltat pentru implementarea algoritmului AES pe o platformă FPGA, cu interfață grafică bazată pe Python (main.py) și comunicare serială (AES_TOP.v), se bazează pe testele efectuate cu date standard și măsurătorile obținute. Tabelul de mai jos evidențiază valorile obținute ca performanță pentru fiecare rulare, randamentul fiind 100% deoarece se obțin de fiecare dată valorile exacte.

Numărul încercării	**Durata (secunde)**	**Randamentul**
1	0.913	100%
2	0.927	100%
3	0.925	100%
4	0.928	100%
5	0.923	100%
6	0.922	100%
7	0.926	100%
8	0.915	100%
9	0.921	100%
10	0.920	100%
Medie	**0.922**	**100%**
Deviația standard	**0.005**	-

Tabelul 4.1. Rezultatele simulării procesului de decriptare (10 rulări)

Am realizat o analiză de performanță pentru seturi de date introduse manual și pentru a observa care este timpul mediu de performanță a datelor transmise. Coloana "Numărul încercării" conține indicii numerici de la 1 la 10, care reprezintă ordinea fiecărei rulări individuale a procesului de decriptare. Coloana "Durata (secunde)" prezintă valorile timpilor de execuție măsurați pentru fiecare încercare, exprimate în secunde cu trei zecimale, reflectând intervalul de timp scurs de la inițierea procesului până la finalizarea afișării. Coloana "Randamentul" indică procentul de succes al fiecărei decriptări, stabilit la 100% pentru toate încercările, sugerând că fiecare rulare a fost completă și corectă. Am inclus media calculată (0.922 secunde) ca ultim rând, pentru a reflecta performanța generală a celor 10 rulări, iar deviația standard include valoarea aproximată de 0.005 secunde, calculată pentru a evidenția variația minimă între rulări, demonstrând consistența performanței, tot acest calcul putând fi vizualizat la Anexa 2.

Datele reflectă performanța, stabilitatea și comportamentul sistemului în scenarii de criptare și decriptare.

4.7. Utilizarea sistemului

În modul de utilizare al sistemului, totul începe prin deschiderea aplicaţiei Python în mediul PyCharm. Odată lansată fereastra grafică, user-ul este întâmpinat de două logo-uri care marchează intrarea în lumea criptografiei hardware-software. Primul pas este alegerea portului serial corespunzător plăcii FPGA, de obicei „COM7” sau „COM5”, apoi setează modul de operare: **Encryption process** sau **Decryption process**.

După introducerea cheii şi a blocului de date în format hexazecimal (32 de caractere fiecare), simpla apăsare a butonului **Send** iniţiază comunicarea cu FPGA-ul. În câteva clipe, interfaţa începe să listeze în fereastra de log stările fiecărei runde AES: de la valorile iniţiale (input şi cheia generată din runda 0), prin SubBytes, ShiftRows, MixColumns şi cheile de rundă intermediare, până la rezultatul final. În acelaşi timp, sub toată evoluția fiecărei runde, apare timpul total de execuţie, oferind feedback instantaneu asupra performanţei.

Dacă utilizatorul activează Dark Mode, interfaţa se transformă într-un fundal întunecat iar textul rămâne lizibil, iar dacă revin la Light Mode, totul se întoarce la schema clasică alb-negru.

Pentru un flux de lucru complet, după finalizarea sesiunii în Python, utilizatorul poate reveni în Vivado Hardware Manager pentru a monitoriza semnalele FPGA-ului în timp real sau poate rula simulări suplimentare în Vivado Simulator folosind testbench-ul inclus. În bara de instrumente PyCharm, o simplă comandă python main.py reporneşte ciclul, iar interacţiunea devine rapidă și intuitivă.

Așadar, printr-o combinaţie echilibrată de hardware de ultimă generaţie şi interfaţă software prietenoasă, utilizatorul are la dispoziţie un instrument complet pentru studierea, testarea şi validarea algoritmului AES, cu vizualizare pas cu pas şi măsurători de performanţă integrate.

Proiectul reunește într-un mod inedit două lumi aparent separate: forța de procesare paralelă a FPGA-ului și flexibilitatea unui mediu software modern, oferind o experiență în care fiecare pas al algoritmului AES este transparent și accesibil. Majoritatea instrumentelor folosite pe piață afișează exclusiv rezultatul final fără să acorde atenție asupra stărilor de pe parcursul procesului.

Capitolul 5. Concluzii

5.1. Gradul de realizare pentru tema propusă

Tema propusă ce vizează implementarea algoritmului AES pe o platformă cu o interfață grafică bazată pe Python pentru control si afișare demonstează o funcționalitate completă a procesului de criptare și decriptare pe un bloc de 128 biți, conform specificațiilor standard AES. Implementarea a inclus dezvoltarea unui modul Verilog care gestionează cele 10 runde ale algoritmului, integrat cu o interfață serială UART pentru comunicare cu un script Python ce oferă suport pentru afișarea rundelor intermediare, a timpului de execuție și o interfață grafică cu moduri dark/light.

Inițial, această lucrare a fost gândită strict pentru a realiza un algoritm AES-128 de criptare/decriptare care să fie verificat prin simulare și așa să demonstreze corectitudinea algoritmului. Ulterior, am gândit câteva extensii alte prototipului de plecare și am venit cu ideea de a realiza o comunicare între PC și FPGA, astfel putând realiza o combinație între limbajul HDL Verilog și Python. Acest lucru a fost gândit cu scopul de a evidenția flexibilitatea subliniată printr-o relație între aceste 2 limbaje. Totodată, m-am gândit cum aș putea să realizez o experiență de utilizare cât mai facilă pentru cel ce va folosi acest proiect și am decis să implementez o interfață cu Tkinter pentru a acorda user-ului o derulare cât mai plăcută a algoritmului, în loc să apară toate rezultatele în consola din PyCharm.

Totul a fost implementat treptat, pentru a fi sigur de funcționalitatea aplicației generale și mi-am salvat de fiecare dată task-urile importante realizate pentru a le refolosi pe viitor și a lucra eficient. Am început tot acest proces prin proiectarea fiecărui modul utilizat din procesul de criptare, apoi pentru cel de decriptare. Verificarea acestora am creat-o prin simularea în Vivado separat a fiecărui modul, utilizând testbench-uri bine definite, iar apoi integrându-le pentru a vedea cum se comportă acele semnale când sunt grupate, iar ieșirea unui semnal devine intrare pentru un alt semnal și tot așa pentru fiecare caz în parte. După ce am dus la final toate aceste task-uri și am fost sigur că valorile semnalelor de ieșire sunt corecte, am trecut la implementarea relației dintre PC și FPGA. Cu ajutorul modulelor UART din Anexa 6, am reușit să transpun o comunicație între cele 2 medii diferite. Pe parcurs a fost nevoie de unele momente de gândire și de documentație pentru a înțelege cum funcționează totul, de asemenea au existat semnale nesincronizat până să ajung să primesc în consolă ce semnal doream. Prin nesincronizarea și suprascrierea unor semnale, rezultatul ce ajungea să fie ieșire era eronat, astfel am realizat multe debug-uri pe parcurs pentru a putea observa ce trebuia modificat pentru a corecta. În urma acestor task-uri realizate, am implementat codul Python pentru a genera o legătură completă între cele 2 medii și pentru a proiecta o interfață cât mai profesională și plăcută vizual în același timp.

Succesul proiectului a fost susținut de o arhitectură bine definită, care include generarea locală a cheilor prin modulul de expansiune a cheii și utilizarea unei S-box stocate local, asigurând conformitatea cu standardul AES și o performanță stabilă, cu o deviație standard de doar 0.005 secunde, ceea ce reflectă consistența procesului. Totuși, au fost identificate anumite limitări, în special legate de comunicarea serială la 9600 baud, care contribuie semnificativ la latența totală (aproximativ 230-250 milisecunde din timpul de execuție), sugerând o oportunitate de optimizare, dar fără a afecta scopul fundamental al proiectului. Obiectivele inițiale nu au fost modificate, deoarece toate cerințele esențiale – implementarea hardware a AES, interfața grafică și comunicarea bidirecțională – au fost atinse, deși ritmul de procesare a fost influențat de constrângerile hardware și software existente, cum ar fi rata de transfer serială și resursele limitate ale plăcii FPGA utilizate.

De asemenea, testele experimentale au confirmat corectitudinea ieșirilor intermediare și a rezultatelor finale, validând implementarea pe parcursul celor 10 runde, ceea ce întărește gradul ridicat de realizare a temei. Cu toate acestea, erorile ocazionale de tip „Timeout or connection lost” raportate în scriptul Python indică o vulnerabilitate minoră a conexiunii seriale, care, deși nu a necesitat modificarea obiectivelor, sugerează necesitatea unor ajustări viitoare pentru îmbunătățirea fiabilității

în scenarii cu interferențe sau volume mai mari de date.

În concluzie, tema a fost realizată cu succes în parametrii propuși, cu un grad de îndeplinire excelent, iar eventualele limitări identificate sunt considerate oportunități de rafinare în plan viitor.

5.2. Direcții viitoare de dezvoltare

Una din direcțiile principale pe care intenționez să le explorez și să le analizez mai atent în viitor este realizarea unei comparații de performanță detaliate între AES-128 și AES-256. Acest demers va fi capabil să evalueze diferențele în ceea ce privește securitatea, timpul de execuție oferite de cele 2 variante ale algoritmului, ajutând astfel pentru înțelegerea aprofundată. Am început deja implementarea AES-256 ca parte a acestui obiectiv, iar un progres concret este reprezentat de dezvoltarea unui testbench prin care am realizat simulări pentru a testa atât AES-128, cât și AES-256, validând funcționalitatea ambelor implementări în mediu software și pregătind terenul pentru integrarea ulterioară, putând fi vizualizat în Anexele 7 și 8.

Pasul următor constă în rularea ambelor versiuni pe FPGA, unde voi dori să compar rezultatele experimentale obținute, să realizez o analiză pentru timpul de execuție în cazul AES-256 și apoi să compar rezultatele obținute cu AES-128 pentru a vedea care este mai optim de folosit și implementat.

5.3. Lecții învățate pe parcursul dezvoltării proiectului de diplomă

Pe parcursul acestui proiect, am acumulat cunoștințe esențiale despre realizarea conexiunii pentru un schimb de informații între PC și FPGA, să scriu codul cât mai modular pentru a putea fi refolosit în viitor în funcție de nevoie, învățând să configurez și să gestionez comunicarea serială la un nivel detaliat, ceea ce mi-a permis să înțeleg mai bine protocoalele de bază și să depășesc provocări precum erorile de tip „Timeout or connection lost”. Mi-am îmbogățit cunoștințele, am învățat, de asemenea, să sincronizez semnalele în mod eficient pentru a asigura transmiterea exactă a datelor prin portul serial, evitând primirea unor rezultate total diferite față de ce așteptam, ajustând parametri precum rata de baud (9600) și ceasul FPGA (100 MHz) pentru a obține o comunicare stabilă și precisă.

În plus, am îmbunătățit semnificativ abilitățile mele de a dezvolta interfețe grafice utilizând Tkinter, integrând funcționalități avansate precum afișarea rundelor intermediare, suportul pentru moduri dark/light și calculul timpului de execuție, ceea ce mi-a oferit o experiență practică în crearea de interfețe intuitive și eficiente. Nu în ultimul rând, am învățat să scriu și să structurez documentația în LaTeX, tehnici precum formatarea ecuațiilor, gestionarea tabelelor și organizarea anexelor, ceea ce mi-a permis să prezint rezultatele într-un mod profesional și coerent, consolidând astfel competențele tehnice și academice acumulate pe parcursul acestui proiect.

Bibliografie

[1] Real Digital, "Boolean Board Hardware Overview," https://www.realdigital.org/hardware/boolean, 2024.

[2] National Institute of Standards and Technology, "FIPS PUB 197: Advanced Encryption Standard (AES)," https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf, 2001.

[3] Nandland, "UART Serial Port Module," https://nandland.com/uart-serial-port-module/, Jun 2025, accessed: June 25, 2025.

[4] GeeksforGeeks, "Python | GUI - Tkinter," https://www.geeksforgeeks.org/python/python-gui-tkinter/, Jun 2025, accessed: June 24, 2025.

[5] SSL Dragon, "Encryption Types and Algorithms Explained," https://www.ssldragon.com/ro/blog/encryption-types-algorithms/, 2023.

[6] E. Peña and M. G. Legaspi, "UART: A Hardware Communication Protocol," https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html, Dec 2020, analog Dialogue, Vol. 54.

[7] G. Saini, "AES Algorithm and Its Hardware Implementation on FPGA: A Step-by-Step Guide," https://medium.com/@imgouravsaini/aes-algorithm-and-its-hardware-implementation-on-fpga-a-step-by-step-guide-2bef178db736, 2022.

[8] GeeksforGeeks, "Advanced Encryption Standard (AES)," https://www.geeksforgeeks.org/advanced-encryption-standard-aes/, 2023.

Anexe

Anexa 1. Codul Python implementat în acest proiect de licență

```
import serial
import serial.tools.list_ports
import time
import tkinter as tk
from tkinter import ttk, scrolledtext
from PIL import Image, ImageTk
from Crypto.Cipher import AES




def read_bytes(ser, n):
    data = b""
    while len(data) < n:
        chunk = ser.read(n - len(data))
        if not chunk:
            raise RuntimeError("Timeout or connection lost")
        data += chunk
    return data


def format_bytes(data):
    return "".join(f"{b:02x}" for b in data)


class AESApp:
    def __init__(self, root):
        self.root = root
        self.root.title("AES-128 Interface")
        self.root.geometry("800x600")

        self.mode_var = tk.StringVar(value="Light")
        theme_menu = ttk.OptionMenu(root, self.mode_var,
                                    "Light", "Light", "Dark",
                                    command=self.switch_theme)
        theme_menu.grid(row=0, column=0, sticky="nw",
                        padx=10, pady=10)

        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=1, column=0, sticky="nsew")
        self.root.grid_rowconfigure(1, weight=1)
        self.root.grid_columnconfigure(0, weight=1)

        self.sigla1 = ImageTk.PhotoImage(
            Image.open("sigla1.png").resize((100, 100),
             ↪  Image.Resampling.LANCZOS))
        self.left_logo = ttk.Label(main_frame, image=self.sigla1)
        self.left_logo.grid(row=0, column=0, rowspan=2,
```

```
48                              padx=(0, 10), pady=10, sticky="n")
49
50          self.sigla2 = ImageTk.PhotoImage(
51              Image.open("sigla2.png").resize((150, 100),
                ↪  Image.Resampling.LANCZOS))
52          self.right_logo = ttk.Label(main_frame, image=self.sigla2)
53          self.right_logo.grid(row=0, column=2, rowspan=2,
54                               padx=(10, 0), pady=10, sticky="n")
55
56          input_frame = ttk.Frame(main_frame)
57          input_frame.grid(row=0, column=1, sticky="ew")
58
59
60          ttk.Label(input_frame, text="Serial port:")\
61              .grid(row=0, column=0, sticky="w", pady=5)
62          ports = [p.device for p in serial.tools.list_ports.comports()]
63          self.port_cb = ttk.Combobox(input_frame, values=ports,
64                                      width=28, state="readonly")
65          self.port_cb.grid(row=0, column=1, sticky="w", pady=5)
66          if ports:
67              self.port_cb.set(ports[0])
68          ttk.Button(input_frame, text="Refresh",
69                     command=self.refresh_ports)\
70              .grid(row=0, column=2, padx=(5,0), pady=5)
71
72
73          ttk.Label(input_frame, text="Mode:")\
74              .grid(row=1, column=0, sticky="w", pady=5)
75          self.mode_var_radio = tk.StringVar(value="0")
76          ttk.Radiobutton(input_frame,
77                          text="Encryption process (0)",
78                          value="0",
79                          variable=self.mode_var_radio,
80                          command=self.update_text_label)\
81              .grid(row=1, column=1, sticky="w")
82          ttk.Radiobutton(input_frame,
83                          text="Decryption process (1)",
84                          value="1",
85                          variable=self.mode_var_radio,
86                          command=self.update_text_label)\
87              .grid(row=1, column=2, sticky="w")
88
89          ttk.Label(input_frame, text="Key (32 hex):")\
90              .grid(row=2, column=0, sticky="w", pady=5)
91          self.key_entry = ttk.Entry(input_frame, width=50)
92          self.key_entry.grid(row=2, column=1,
93                              columnspan=2, sticky="w", pady=5)
94
95          self.text_label = tk.StringVar(value="Plaintext (32 hex):")
96          ttk.Label(input_frame, textvariable=self.text_label)\
97              .grid(row=3, column=0, sticky="w", pady=5)
98          self.text_entry = ttk.Entry(input_frame, width=50)
99          self.text_entry.grid(row=3, column=1,
```

```
100                                 columnspan=2, sticky="w", pady=5)
101
102             ttk.Button(input_frame, text="Send",
103                        command=self.send_data)\
104                 .grid(row=4, column=1, pady=10)
105
106             self.output_text = scrolledtext.ScrolledText(
107                 main_frame, width=60, height=20, wrap=tk.WORD)
108             self.output_text.grid(row=1, column=1, padx=10,
109                                   pady=10, sticky="nsew")
110             main_frame.grid_columnconfigure(1, weight=1)
111
112             self.set_light_theme()
113
114
115         def refresh_ports(self):
116             ports = [p.device for p in serial.tools.list_ports.comports()]
117             self.port_cb['values'] = ports
118             if ports:
119                 self.port_cb.set(ports[0])
120
121
122
123         def set_light_theme(self):
124             style = ttk.Style()
125             self.root.configure(bg="#ffffff")
126             style.configure("TFrame", background="#ffffff")
127             style.configure("TLabel", background="#ffffff",
128                             foreground="#000000")
129             style.configure("TRadiobutton", background="#ffffff",
130                             foreground="#000000")
131             style.configure("TButton", background="#e0e0e0",
132                             foreground="#000000")
133             style.configure("TEntry", fieldbackground="#ffffff",
134                             foreground="#000000")
135             self.output_text.configure(bg="#ffffff", fg="#000000")
136
137
138
139         def set_dark_theme(self):
140             style = ttk.Style()
141             self.root.configure(bg="#333333")
142             style.configure("TFrame", background="#333333")
143             style.configure("TLabel", background="#333333",
144                             foreground="#ffffff")
145             style.configure("TRadiobutton", background="#333333",
146                             foreground="#ffffff")
147             style.configure("TButton", background="#555555",
148                             foreground="#ffffff")
149             style.configure("TEntry", fieldbackground="#cccccc",
150                             foreground="#000000")
151
152             self.output_text.configure(bg="#cccccc", fg="#000000")
```

```
153
154
155         def switch_theme(self, mode):
156             if mode == "Dark":
157                 self.set_dark_theme()
158             else:
159                 self.set_light_theme()
160
161
162         def update_text_label(self):
163             self.text_label.set(
164                 "Ciphertext (32 hex):"
165                 if self.mode_var_radio.get() == "1"
166                 else "Plaintext"
167             )
168
169
170         def read_and_filter(self, ser, field_name,
171                             round_num, prev_value):
172             data = read_bytes(ser, 16)
173             while data == b'\x00'*16 or data == prev_value:
174                 data = read_bytes(ser, 16)
175             self.output_text.insert(
176                 tk.END,
177                 f"round[{round_num:2}]."
178                 f"{field_name:<8} {format_bytes(data)}\n"
179             )
180             self.output_text.see(tk.END)
181             self.root.update()
182             return data
183
184         def send_data(self):
185             self.output_text.delete(1.0, tk.END)
186             start_time = time.time()
187
188             port = self.port_cb.get().strip()
189             mode = self.mode_var_radio.get()
190             key_hex = self.key_entry.get().strip().lower()
191             text_hex = self.text_entry.get().strip().lower()
192
193             if not port:
194                 self.output_text.insert(
195                     tk.END, "Eroare: Selecteaza un port serial.\n")
196                 return
197             if mode not in ['0','1']:
198                 self.output_text.insert(
199                     tk.END, "Eroare: Modul invalid! Foloseste 0 sau 1.\n")
200                 return
201             if len(key_hex)!=32 or len(text_hex)!=32:
202                 self.output_text.insert(
203                     tk.END,
204                     "Eroare: Cheia si textul trebuie sa aiba "
205                     "exact 32 caractere hex.\n"
```

```
206             )
207             return
208
209         try:
210             key_bytes = bytes.fromhex(key_hex)
211             text_bytes = bytes.fromhex(text_hex)
212             mode_byte = bytes([int(mode)])
213         except ValueError:
214             self.output_text.insert(
215                 tk.END,
216                 "Eroare: Intrari hex invalide.\n"
217             )
218             return
219
220         prev_value = None
221
222         try:
223             with serial.Serial(port, 9600,
224                                timeout=5) as ser:
225                 self.output_text.insert(
226                     tk.END,
227                     f"Trimite modul "
228                     f"({'criptare' if mode=='0' else 'decriptare'}), "
229                     f"cheia și "
230                     f"{'plaintext-ul' if mode=='0' else
                         ↪  'ciphertext-ul'} "
231                     f"către FPGA...\n"
232                 )
233                 ser.write(mode_byte + key_bytes + text_bytes)
234                 time.sleep(0.1)
235
236                 if mode=='0':
237                     prev_value = self.read_and_filter(
238                         ser, "k_sch", 0, prev_value
239                     )
240                     for r in range(1,11):
241                         stages = (["start","s_box","s_row",
242                                    "m_col","k_sch"]
243                                   if r<10 else
244                                   ["start","s_box","s_row",
245                                    "k_sch","output"])
246                         for st in stages:
247                             prev_value = self.read_and_filter(
248                                 ser, st, r, prev_value
249                             )
250                 else:
251                     prev_value = self.read_and_filter(
252                         ser, "ik_sch", 0, prev_value
253                     )
254                     for r in range(1,11):
255                         stages = (["istart","is_row","is_box",
256                                    "ik_sch","ik_add"]
257                                   if r<10 else
```

```
                                    ["istart","is_row","is_box",
                                     "ik_sch","output"])
                            for st in stages:
                                prev_value = self.read_and_filter(
                                    ser, st, r, prev_value
                                )

                        self.output_text.insert(
                            tk.END, "\nProces finalizat.\n"
                        )
                        exec_time = time.time() - start_time
                        self.output_text.insert(
                            tk.END,
                            f"Timp de executie: {exec_time:.3f} secunde\n"
                        )
                        self.output_text.see(tk.END)



                    cipher = AES.new(key_bytes, AES.MODE_ECB)

                    if mode=='0':
                        expected = cipher.encrypt(text_bytes)
                    else:
                        expected = cipher.decrypt(text_bytes)

                    if prev_value == expected:
                        self.output_text.insert(tk.END, "TEST PASSED: Output
                         ↪  corect.\n")
                    else:
                        self.output_text.insert(
                            tk.END,
                            f"TEST FAILED:\n"
                            f"  Expected: {expected.hex()}\n"
                            f"  Got:      {format_bytes(prev_value)}\n"
                        )
                    self.output_text.see(tk.END)


                except Exception as e:
                    self.output_text.insert(
                        tk.END, f"Eroare: {e}\n"
                    )
                    self.output_text.see(tk.END)


if __name__=="__main__":
    root = tk.Tk()
    app = AESApp(root)
    root.mainloop()
```

Listing 1. Cod Python folosit pentru conexiunea dintre PC și FPGA, plus generarea unei interfețe

Anexa 2. Timpul de execuție

În această secțiune sunt prezentate detaliile calculului mediei și al deviației standard pentru timpurile de execuție obținute în cele 10 rulări ale proceselor de criptare și decriptare.

2.1. Criptare

Valorile obținute: 0.920, 0.919, 0.918, 0.915, 0.913, 0.921, 0.904, 0.909, 0.921, 0.906.

2.1.1. Calculul mediei

Media (μ): $\mu = \frac{0.920+0.919+0.918+0.915+0.913+0.921+0.904+0.909+0.921+0.906}{10} = \frac{9.146}{10} = 0.9146$ secunde.

2.1.2. Calculul deviației standard

$\sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{n}}$, cu $\mu = 0.9146$, $n = 10$.

1. Diferențe ($x_i - \mu$): 0.0054, 0.0044, 0.0034, 0.0004, -0.0016, 0.0064, -0.0106, -0.0056, 0.0064, -0.0086.
2. Pătrate: 0.00002916, 0.00001936, 0.00001156, 0.00000016, 0.00000256, 0.00004096, 0.00011236, 0.00003136, 0.00004096, 0.00007396.
3. Sumă pătratelor: 0.00002916 + 0.00001936 + 0.00001156 + 0.00000016 + 0.00000256 + 0.00004096 + 0.00011236 + 0.00003136 + 0.00004096 + 0.00007396 = 0.0003624.
4. Varianța: $\frac{0.0003624}{10} = 0.00003624$.
5. Deviația standard: $\sigma = \sqrt{0.00003624} \approx 0.0060$ secunde.

Așadar, deviația standard a timpilor de execuție pentru criptare este **0.0060** secunde, indicând o variație minimă între rulări și o performanță stabilă a sistemului.

2.2. Decriptare

Valorile obținute: 1.215, 1.219, 1.221, 1.224, 1.216, 1.218, 1.226, 1.219, 1.226, 1.222.

2.2.1. Calculul mediei

Media (μ): $\mu = \frac{1.215+1.219+1.221+1.224+1.216+1.218+1.226+1.219+1.226+1.222}{10} = \frac{12.206}{10} = 1.2206$ secunde.

2.2.2. Calculul deviației standard

$\sigma = \sqrt{\frac{\sum(x_i-\mu)^2}{n}}$, cu $\mu = 1.2206$, $n = 10$.

1. Diferențe ($x_i - \mu$): -0.0056, -0.0016, 0.0004, 0.0034, -0.0046, -0.0026, 0.0054, -0.0016, 0.0054, 0.0014.
2. Pătrate: 0.00003136, 0.00000256, 0.00000016, 0.00001156, 0.00002116, 0.00000676, 0.00002916, 0.00000256, 0.00002916, 0.00000196.
3. Sumă pătratelor: 0.00003136 + 0.00000256 + 0.00000016 + 0.00001156 + 0.00002116 + 0.00000676 + 0.00002916 + 0.00000256 + 0.00002916 + 0.00000196 = 0.0001364.
4. Varianța: $\frac{0.0001364}{10} = 0.00001364$.
5. Deviația standard: $\sigma = \sqrt{0.00001364} \approx 0.0037$ secunde.

Așadar, deviația standard a timpilor de execuție pentru decriptare este **0.0037** secunde, indicând o variație minimă între rulări și o performanță stabilă a sistemului.

Numărul încercării	Durata (secunde)	Randamentul
1	0.920	100%
2	0.919	100%
3	0.918	100%
4	0.915	100%
5	0.913	100%
6	0.921	100%
7	0.904	100%
8	0.909	100%
9	0.921	100%
10	0.906	100%
Medie	**0.9146**	**100%**
Deviația standard	**0.0060**	-

Tabelul A.1. Rezultatele simulării procesului de criptare (10 rulări)

Numărul încercării	Durata (secunde)	Randamentul
1	1.215	100%
2	1.219	100%
3	1.221	100%
4	1.224	100%
5	1.216	100%
6	1.218	100%
7	1.226	100%
8	1.219	100%
9	1.226	100%
10	1.222	100%
Medie	**1.2206**	**100%**
Deviația standard	**0.0037**	-

Tabelul A.2. Rezultatele simulării procesului de decriptare (10 rulări)

Anexa 3. Interfața în Tkinter cu utilizatorul

În Figurile 4.4 și 4.5 este prezentată interfața cu diverse componente:

- **Serial port (ex: com5 / COM5)**: aici utilizatorul introduce numele portului UART la care este conectat FPGA-ul, de exemplu COM5.
- **Mode : Encryption process (0) / Decryption process (1)** (`Radiobutton`): butoane radio care setează variabila internă `mode_var` la `"0"` sau `"1"`, indicând scriptului dacă să cripteze sau să decripteze datele trimise de utilizator în ferestrele de mai jos.
- **Key (32 hex)**: câmp de text în care se tastează cheia AES de 128 biţi (32 de caractere hexazecimale); este validată la apăsarea butonului "Send”.
- **Plaintext / Ciphertext**: câmp de text unde se introduce blocul de date (32 hex); eticheta se schimbă automat în "Ciphertext” când utilizatorul alege decriptarea.
- **Butonul "Send”** (`Button`): la click validează intrările, construiește pachetul UART (mod + cheie + date) și îl transmite către FPGA, apoi inițiază recepția și afișarea rundelor.

- **ScrolledText** (`ScrolledText`): zonă de jurnal în care apar în timp real mesajele de stare, etichetele `round[n].field` și valorile hex ale fiecărui bloc AES recepționat din FPGA.

- **Siglele instituționale** (`Label` cu `ImageTk`): elemente vizuale plasate în stânga și dreapta sus, care oferă o exprimare academică și identifică aparținerea proiectului.

- **Moduri Light și Dark**: Am adăugat suport pentru moduri Light și Dark pentru a personaliza aspectului ferestrei, permițând utilizatorului să comute între o temă luminoasă (cu fundal alb și text negru) și una întunecată (cu fundal gri închis și text alb), îmbunătățind experiența vizuală și cea de acces în funcție de preferințe sau condițiile de lucru.

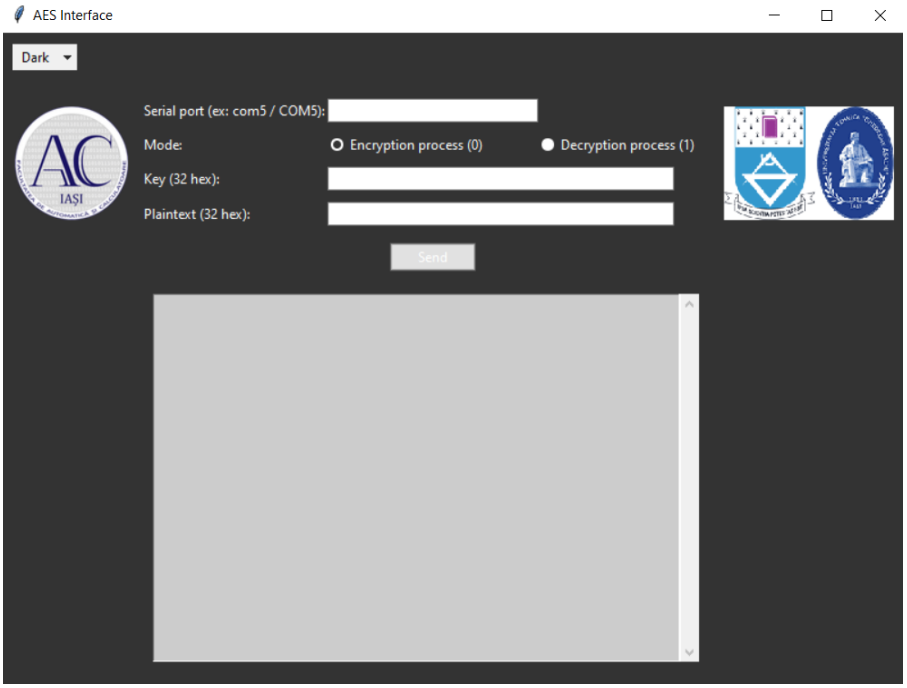


Figura A.1. Interfață în Dark Mode pentru AES-128[16]

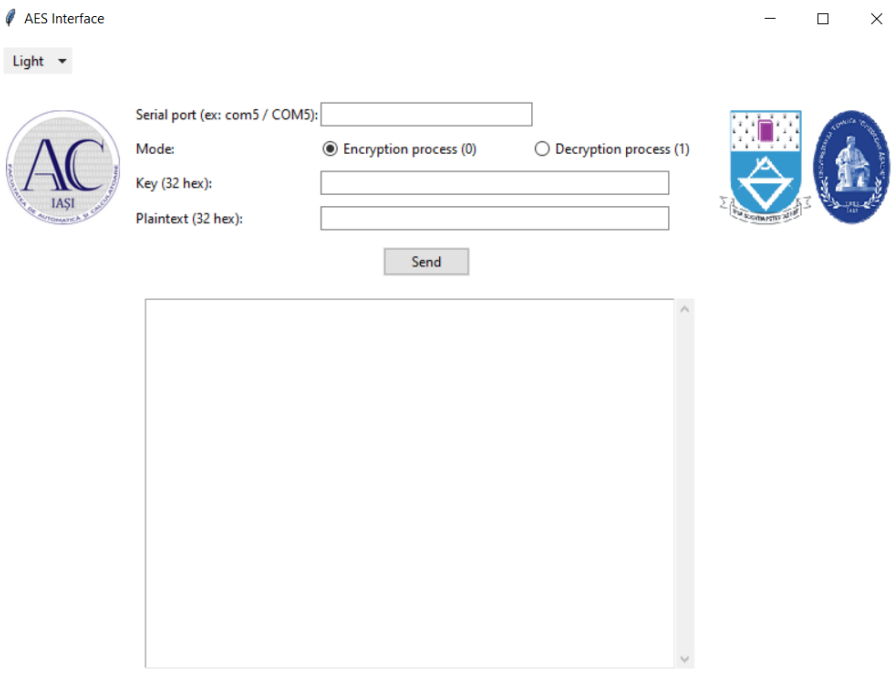


Figura A.2. Interfață în Light Mode pentru AES-128[17]

Anexa 4. Exemplu de ieșiri complete ale rundelor AES-128

Mai jos este prezentată ieșirea completă a scriptului Python pentru criptarea și decriptarea AES-128 folosind cheia și plaintext-ul specificate. Datele sunt afișate runda cu rundă, în conformitate cu FIPS PUB 197.

Tabelul A.3. Exemplu output runde AES-128 conform NIST

PLAINTEXT: 00112233445566778899aabbccddeeff
KEY: 000102030405060708090a0b0c0d0e0f

CIPHER (ENCRYPT):

Rundă	Etapă	Valoare hex
0	input	00112233445566778899aabbccddeeff
0	k_sch	000102030405060708090a0b0c0d0e0f
1	start	00102030405060708090a0b0c0d0e0f0
1	s_box	63cab7040953d051cd60e0e7ba70e18c
1	s_row	6353e08c0960e104cd70b751bacad0e7
1	m_col	5f72641557f5bc92f7be3b291db9f91a
1	k_sch	d6aa74fdd2af72fadaa678f1d6ab76fe
2	start	89d810e8855ace682d1843d8cb128fe4
2	s_box	a761ca9b97be8b45d8ad1a611fc97369
2	s_row	a7be1a6997ad739bd8c9ca451f618b61
2	m_col	ff87968431d86a51645151fa773ad009
2	k_sch	b692cf0b643dbdf1be9bc5006830b3fe
3	start	4915598f55e5d7a0daca94fa1f0a63f7
3	s_box	3b59cb73fcd90ee05774222dc067fb68
3	s_row	3bd92268fc74fb735767cbe0c0590e2d
3	m_col	4c9c1e66f771f0762c3f868e534df256
3	k_sch	b6ff744ed2c2c9bf6c590cbf0469bf41
4	start	fa636a2825b339c940668a3157244d17
4	s_box	2dfb02343f6d12dd09337ec75b36e3f0
4	s_row	2d6d7ef03f33e334093602dd5bfb12c7
4	m_col	6385b79ffc538df997be478e7547d691
4	k_sch	47f7f7bc95353e03f96c32bcfd058dfd
5	start	247240236966b3fa6ed2753288425b6c
5	s_box	36400926f9336d2d9fb59d23c42c3950
5	s_row	36339d50f9b539269f2c092dc4406d23
5	m_col	f4bcd45432e554d075f1d6c51dd03b3c
5	k_sch	3caaa3e8a99f9deb50f3af57adf622aa
6	start	c81677bc9b7ac93b25027992b0261996
6	s_box	e847f56514dadde23f77b64fe7f7d490
6	s_row	e8dab6901477d4653ff7f5e2e747dd4f
6	m_col	9816ee7400f87f556b2c049c8e5ad036
6	k_sch	5e390f7df7a69296a7553dc10aa31f6b
7	start	c62fe109f75eedc3cc79395d84f9cf5d
7	s_box	b415f8016858552e4bb6124c5f998a4c
7	s_row	b458124c68b68a014b99f82e5f15554c
7	m_col	c57e1c159a9bd286f05f4be098c63439
7	k_sch	14f9701ae35fe28c440adf4d4ea9c026
8	start	d1876c0f79c4300ab45594add66ff41f
8	s_box	3e175076b61c04678dfc2295f6a8bfc0

Tabela A.3 (continuare)

Rundă	Etapă	Valoare hex
8	s_row	3e1c22c0b6fcbf768da85067f6170495
8	m_col	baa03de7a1f9b56ed5512cba5f414d23
8	k_sch	47438735a41c65b9e016baf4aebf7ad2
9	start	fde3bad205e5d0d73547964ef1fe37f1
9	s_box	5411f4b56bd9700e96a0902fa1bb9aa1
9	s_row	54d990a16ba09ab596bbf40ea111702f
9	m_col	e9f74eec023020f61bf2ccf2353c21c7
9	k_sch	549932d1f08557681093ed9cbe2c974e
10	start	bd6e7c3df2b5779e0b61216e8b10b689
10	s_box	7a9f102789d5f50b2beffd9f3dca4ea7
10	s_row	7ad5fda789ef4e272bca100b3d9ff59f
10	k_sch	13111d7fe3944a17f307a78b4d2b30c5
10	output	69c4e0d86a7b0430d8cdb78070b4c55a
INVERSE CIPHER (DECRYPT):		
0	iinput	69c4e0d86a7b0430d8cdb78070b4c55a
0	ik_sch	13111d7fe3944a17f307a78b4d2b30c5
1	istart	7ad5fda789ef4e272bca100b3d9ff59f
1	is_row	7a9f102789d5f50b2beffd9f3dca4ea7
1	is_box	bd6e7c3df2b5779e0b61216e8b10b689
1	ik_sch	549932d1f08557681093ed9cbe2c974e
1	ik_add	e9f74eec023020f61bf2ccf2353c21c7
2	istart	54d990a16ba09ab596bbf40ea111702f
2	is_row	5411f4b56bd9700e96a0902fa1bb9aa1
2	is_box	fde3bad205e5d0d73547964ef1fe37f1
2	ik_sch	47438735a41c65b9e016baf4aebf7ad2
2	ik_add	baa03de7a1f9b56ed5512cba5f414d23
3	istart	3e1c22c0b6fcbf768da85067f6170495
3	is_row	3e175076b61c04678dfc2295f6a8bfc0
3	is_box	d1876c0f79c4300ab45594add66ff41f
3	ik_sch	14f9701ae35fe28c440adf4d4ea9c026
3	ik_add	c57e1c159a9bd286f05f4be098c63439
4	istart	b458124c68b68a014b99f82e5f15554c
4	is_row	b415f8016858552e4bb6124c5f998a4c
4	is_box	c62fe109f75eedc3cc79395d84f9cf5d
4	ik_sch	5e390f7df7a69296a7553dc10aa31f6b
4	ik_add	9816ee7400f87f556b2c049c8e5ad036
5	istart	e8dab6901477d4653ff7f5e2e747dd4f
5	is_row	e847f56514dadde23f77b64fe7f7d490
5	is_box	c81677bc9b7ac93b25027992b0261996
5	ik_sch	3caaa3e8a99f9deb50f3af57adf622aa
5	ik_add	f4bcd45432e554d075f1d6c51dd03b3c
6	istart	36339d50f9b539269f2c092dc4406d23
6	is_row	36400926f9336d2d9fb59d23c42c3950
6	is_box	247240236966b3fa6ed2753288425b6c
6	ik_sch	47f7f7bc95353e03f96c32bcfd058dfd
6	ik_add	6385b79ffc538df997be478e7547d691
7	istart	2d6d7ef03f33e334093602dd5bfb12c7

Tabela A.3 (continuare)

Rundă	**Etapă**	**Valoare hex**
7	is_row	2dfb02343f6d12dd09337ec75b36e3f0
7	is_box	fa636a2825b339c940668a3157244d17
7	ik_sch	b6ff744ed2c2c9bf6c590cbf0469bf41
7	ik_add	4c9c1e66f771f0762c3f868e534df256
8	istart	3bd92268fc74fb735767cbe0c0590e2d
8	is_row	3b59cb73fcd90ee05774222dc067fb68
8	is_box	4915598f55e5d7a0daca94fa1f0a63f7
8	ik_sch	b692cf0b643dbdf1be9bc5006830b3fe
8	ik_add	ff87968431d86a51645151fa773ad009
9	istart	a7be1a6997ad739bd8c9ca451f618b61
9	is_row	a761ca9b97be8b45d8ad1a611fc97369
9	is_box	89d810e8855ace682d1843d8cb128fe4
9	ik_sch	d6aa74fdd2af72fadaa678f1d6ab76fe
9	ik_add	5f72641557f5bc92f7be3b291db9f91a
10	istart	6353e08c0960e104cd70b751bacad0e7
10	is_row	63cab7040953d051cd60e0e7ba70e18c
10	is_box	000102030405060708090a0b0c0d0e0f0
10	ik_sch	0001020304050607080 90a0b0c0d0e0f
10	ioutput	00112233445566778899aabbccddeeff

Anexa 5. Tabel simplu

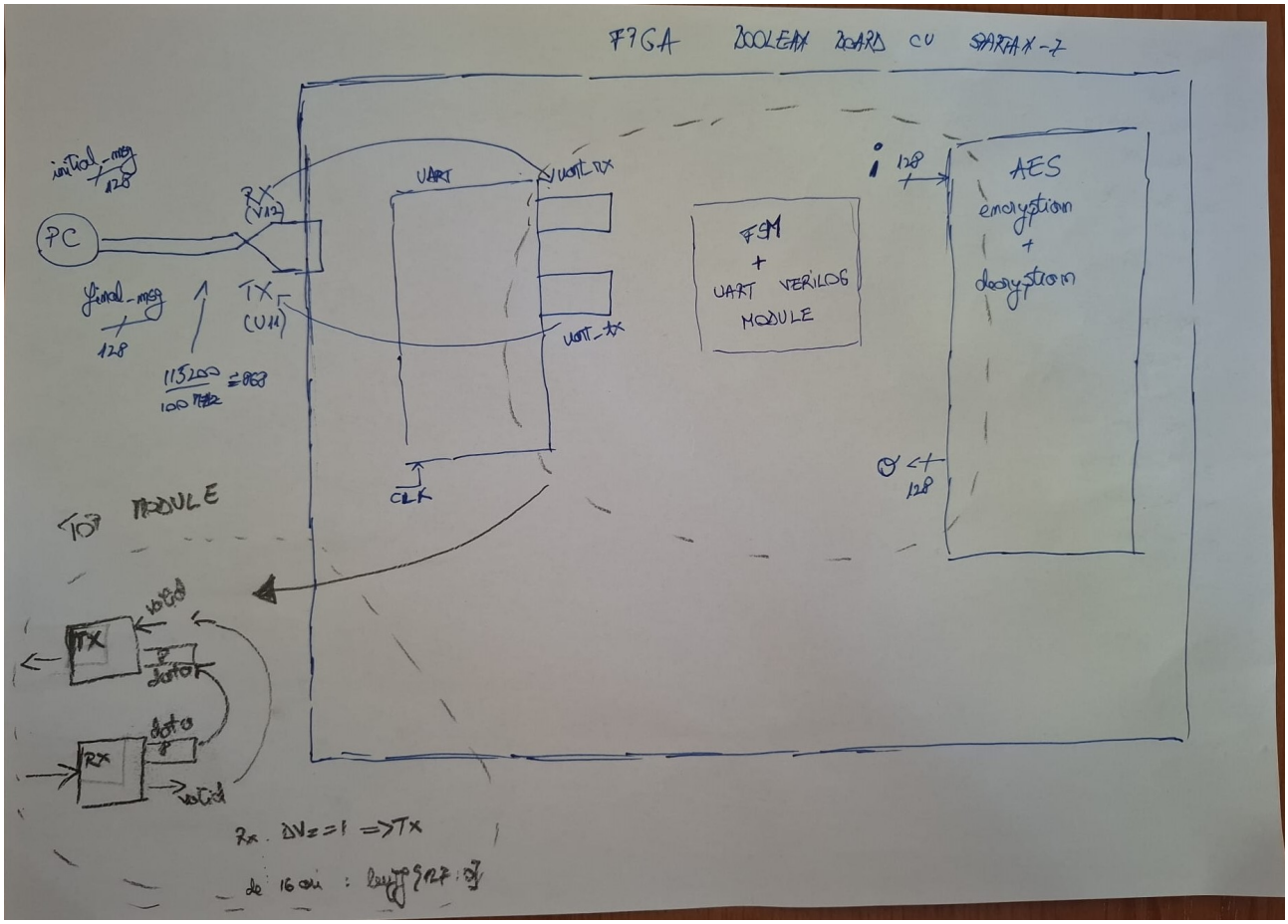


Figura A.3. Schemă reprezentativă pentru relația dintre PC și modulul ce ține de AES pe FPGA Boolean Board cu Spartan-7[18]

Anexa 6. Modulele Verilog pentru comunicația UART

```
1       module uart_rx
2     #(parameter CLKS_PER_BIT = 10416)
3     (
4      input        i_Clock,
5      input        i_Rx_Serial,
6      output       o_Rx_DV,
7      output [7:0] o_Rx_Byte
8      );
9
10    parameter s_IDLE         = 3'b000;
11    parameter s_RX_START_BIT = 3'b001;
12    parameter s_RX_DATA_BITS = 3'b010;
13    parameter s_RX_STOP_BIT  = 3'b011;
14    parameter s_CLEANUP      = 3'b100;
15
16    reg        r_Rx_Data_R = 1'b1;
17    reg        r_Rx_Data   = 1'b1;
18    reg [13:0] r_Clock_Count = 0;
19    reg [2:0]  r_Bit_Index   = 0;
20    reg [7:0]  r_Rx_Byte     = 0;
21    reg        r_Rx_DV       = 0;
22    reg [2:0]  r_SM_Main     = 0;
23
24    always @(posedge i_Clock)
25      begin
26        r_Rx_Data_R <= i_Rx_Serial;
27        r_Rx_Data   <= r_Rx_Data_R;
28      end
29
30    always @(posedge i_Clock)
31      begin
32        case (r_SM_Main)
33          s_IDLE :
34            begin
35              r_Rx_DV       <= 1'b0;
36              r_Clock_Count <= 0;
37              r_Bit_Index   <= 0;
38
39              if (r_Rx_Data == 1'b0)
40                r_SM_Main <= s_RX_START_BIT;
41            end
42
43          s_RX_START_BIT :
44            begin
45              if (r_Clock_Count == (CLKS_PER_BIT-1)/2)
46                begin
47                  if (r_Rx_Data == 1'b0)
48                    begin
49                      r_Clock_Count <= 0;
50                      r_SM_Main     <= s_RX_DATA_BITS;
51                    end
52                  else
```

```
53                    r_SM_Main <= s_IDLE;
54                end
55              else
56                r_Clock_Count <= r_Clock_Count + 1;
57            end
58
59          s_RX_DATA_BITS :
60            begin
61              if (r_Clock_Count < CLKS_PER_BIT-1)
62                r_Clock_Count <= r_Clock_Count + 1;
63              else
64                begin
65                  r_Clock_Count          <= 0;
66                  r_Rx_Byte[r_Bit_Index] <= r_Rx_Data;
67
68                  if (r_Bit_Index < 7)
69                    r_Bit_Index <= r_Bit_Index + 1;
70                  else
71                    begin
72                      r_Bit_Index <= 0;
73                      r_SM_Main   <= s_RX_STOP_BIT;
74                    end
75                end
76            end
77
78          s_RX_STOP_BIT :
79            begin
80              if (r_Clock_Count < CLKS_PER_BIT-1)
81                r_Clock_Count <= r_Clock_Count + 1;
82              else
83                begin
84                  r_Rx_DV       <= 1'b1;
85                  r_Clock_Count <= 0;
86                  r_SM_Main     <= s_CLEANUP;
87                end
88            end
89
90          s_CLEANUP :
91            begin
92              r_SM_Main <= s_IDLE;
93              r_Rx_DV   <= 1'b0;
94            end
95
96          default :
97            r_SM_Main <= s_IDLE;
98        endcase
99      end
100
101   assign o_Rx_DV   = r_Rx_DV;
102   assign o_Rx_Byte = r_Rx_Byte;
103
104 endmodule
105
```

```
106
107
108  // uart_tx.v
109  module uart_tx
110    #(parameter CLKS_PER_BIT = 10416)
111    (
112     input        i_Clock,
113     input        i_Tx_DV,
114     input  [7:0] i_Tx_Byte,
115     output       o_Tx_Active,
116     output reg   o_Tx_Serial,
117     output       o_Tx_Done
118     );
119
120    parameter s_IDLE         = 3'b000;
121    parameter s_TX_START_BIT = 3'b001;
122    parameter s_TX_DATA_BITS = 3'b010;
123    parameter s_TX_STOP_BIT  = 3'b011;
124    parameter s_CLEANUP      = 3'b100;
125
126    reg [2:0] r_SM_Main      = 0;
127    reg [13:0] r_Clock_Count = 0;
128    reg [2:0] r_Bit_Index    = 0;
129    reg [7:0] r_Tx_Data      = 0;
130    reg       r_Tx_Done      = 0;
131    reg       r_Tx_Active    = 0;
132
133    always @(posedge i_Clock)
134      begin
135        case (r_SM_Main)
136          s_IDLE :
137            begin
138              o_Tx_Serial   <= 1'b1;
139              r_Tx_Done     <= 1'b0;
140              r_Clock_Count <= 0;
141              r_Bit_Index   <= 0;
142
143              if (i_Tx_DV == 1'b1)
144                begin
145                  r_Tx_Active <= 1'b1;
146                  r_Tx_Data   <= i_Tx_Byte;
147                  r_SM_Main   <= s_TX_START_BIT;
148                end
149              else
150                r_SM_Main <= s_IDLE;
151            end
152
153          s_TX_START_BIT :
154            begin
155              o_Tx_Serial <= 1'b0;
156
157              if (r_Clock_Count < CLKS_PER_BIT-1)
158                r_Clock_Count <= r_Clock_Count + 1;
```

```
159             else
160               begin
161                 r_Clock_Count <= 0;
162                 r_SM_Main     <= s_TX_DATA_BITS;
163               end
164           end
165
166         s_TX_DATA_BITS :
167           begin
168             o_Tx_Serial <= r_Tx_Data[r_Bit_Index];
169
170             if (r_Clock_Count < CLKS_PER_BIT-1)
171               r_Clock_Count <= r_Clock_Count + 1;
172             else
173               begin
174                 r_Clock_Count <= 0;
175
176                 if (r_Bit_Index < 7)
177                   r_Bit_Index <= r_Bit_Index + 1;
178                 else
179                   begin
180                     r_Bit_Index <= 0;
181                     r_SM_Main   <= s_TX_STOP_BIT;
182                   end
183               end
184           end
185
186         s_TX_STOP_BIT :
187           begin
188             o_Tx_Serial <= 1'b1;
189
190             if (r_Clock_Count < CLKS_PER_BIT-1)
191               r_Clock_Count <= r_Clock_Count + 1;
192             else
193               begin
194                 r_Tx_Done     <= 1'b1;
195                 r_Clock_Count <= 0;
196                 r_SM_Main     <= s_CLEANUP;
197                 r_Tx_Active   <= 1'b0;
198               end
199           end
200
201         s_CLEANUP :
202           begin
203             r_Tx_Done <= 1'b1;
204             r_SM_Main <= s_IDLE;
205           end
206
207         default :
208           r_SM_Main <= s_IDLE;
209       endcase
210     end
211
```

```
212     assign o_Tx_Active = r_Tx_Active;
213     assign o_Tx_Done   = r_Tx_Done;
214
215   endmodule
```

Listing 2. Codul modulelor *uart_rx* și *uart_tx*

Anexa 7. Testbench creat pe parcurs pentru validarea codului verilog prin simulare

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   // Company:
4   // Engineer:
5   //
6   // Create Date: 03/18/2025
7   // Design Name:
8   // Module Name: AES_TB
9   // Project Name:
10  // Target Devices:
11  // Tool Versions:
12  // Description:
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21
22
23  module AES_TB;
24
25      reg clk;
26      reg rst;
27      reg start;
28      reg [127:0] data_in;
29      reg [127:0] key_128;
30      reg [255:0] key_256;
31
32      wire [127:0] cipher_out_128;
33      wire [127:0] decrypted_out_128;
34      wire [127:0] cipher_out_256;
35      wire [127:0] decrypted_out_256;
36      wire done_enc_128;
37      wire done_dec_128;
38      wire done_enc_256;
39      wire done_dec_256;
40
41      AES_TOP dut (
42          .clk(clk),
43          .rst(rst),
44          .start(start),
45          .data_in(data_in),
46          .key_128(key_128),
47          .key_256(key_256),
48          .cipher_out_128(cipher_out_128),
49          .decrypted_out_128(decrypted_out_128),
50          .cipher_out_256(cipher_out_256),
51          .decrypted_out_256(decrypted_out_256),
52          .done_enc_128(done_enc_128),
```

```
53          .done_dec_128(done_dec_128),
54          .done_enc_256(done_enc_256),
55          .done_dec_256(done_dec_256)
56      );
57
58      initial begin
59          clk = 0;
60          forever #5 clk = ~clk;
61      end
62
63      initial begin
64          rst = 1;
65          start = 0;
66          #20;
67          rst = 0;
68
69          // init date
70          data_in = 128'h00112233445566778899aabbccddeeff;
71          key_128 = 128'h000102030405060708090a0b0c0d0e0f;
72          key_256 =
             ↪  256'h000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
73
74         // start
75          #10;
76          start = 1;
77          #10;
78          start = 0;
79
80          // se asteapta terminarea criptarii si decriptarii
81          wait(done_enc_128 && done_enc_256);
82          wait(done_dec_128 && done_dec_256);
83
84          $display("AES-128");
85          $display("Plaintext  : %h", data_in);
86          $display("Encrypted  : %h", cipher_out_128);
87          $display("Decrypted  : %h", decrypted_out_128);
88          if(decrypted_out_128 == data_in)
89              $display("Decriptarea pentru AES-128 este OK!!!!");
90          else
91              $display("Decriptarea pentru AES-128 NU e ok!!");
92
93          $display("AES-256");
94          $display("Plaintext  : %h", data_in);
95          $display("Encrypted  : %h", cipher_out_256);
96          $display("Decrypted  : %h", decrypted_out_256);
97          if(decrypted_out_256 == data_in)
98              $display("Decriptarea pentru AES-256 este OK!!!!");
99          else
100             $display("Decriptarea pentru AES-256 NU e ok!!");
101
102         $finish;
103     end
104
```

```
105   endmodule
```

Listing 3. Testbench integrat pentru AES-128 și AES-256

Anexa 8. Simularea testbench-ului pentru AES-128 și AES-256

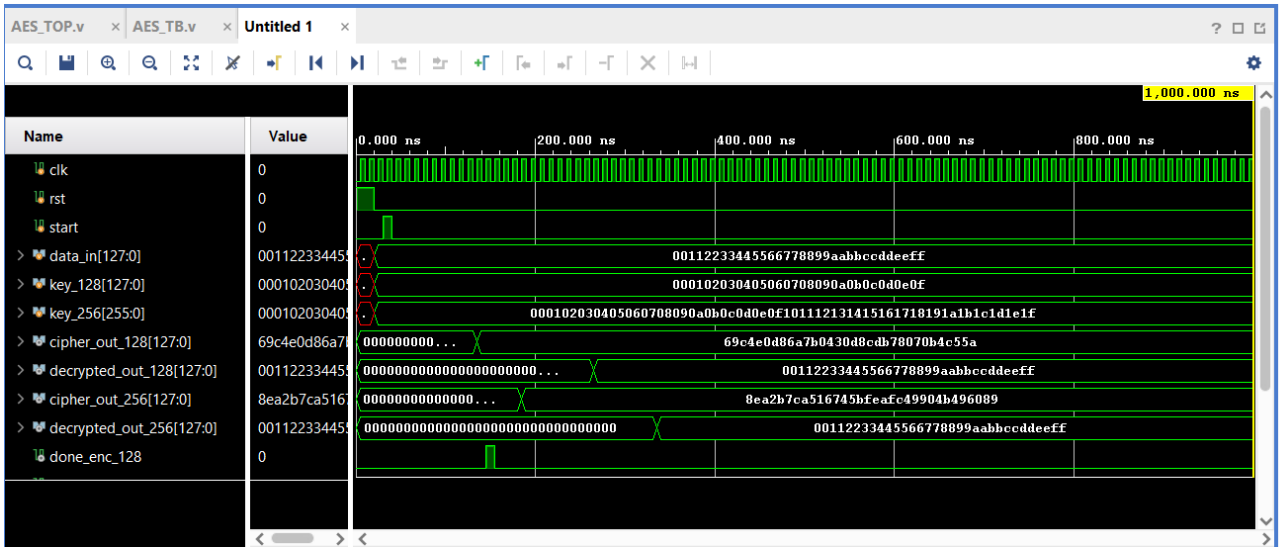


Figura A.4. AT[19]

Anexa 9. Fișier de constrângeri utilizate pentru Boolean Board cu Spartan-7

Pinii aleși pentru clk, tx, rx si rst au fost transpuși din documentul oficial schematic postat de cei de la RealDigital pe site-ul de prezentare al plăcii Boolean Board (https://www.realdigital.org/hardware/boolean)

```
1   set_property PACKAGE_PIN F14 [get_ports clk]
2   set_property IOSTANDARD LVCMOS33 [get_ports clk]
3   create_clock -period 10.000 -name clk [get_ports clk]
4
5   set_property PACKAGE_PIN V12 [get_ports rx]
6   set_property IOSTANDARD LVCMOS33 [get_ports rx]
7
8   set_property PACKAGE_PIN U11 [get_ports tx]
9   set_property IOSTANDARD LVCMOS33 [get_ports tx]
10
11  set_property PACKAGE_PIN J2 [get_ports rst]
12  set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

Listing 4. Constrângeri folosite pentru Boolean Board cu Spartan-7