

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DOMENIUL: Calculatoare și Tehnologia Informației  
SPECIALIZAREA: Calculatoare

## **LUCRARE DE DIPLOMĂ**

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

Absolvent:  
Cristian-Rafael Tudusciuc

**Iași, 2024**



UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IASI  
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE  
DOMENIUL: Calculatoare și Tehnologia Informației  
SPECIALIZAREA: Calculatoare

# **Accelerator hardware de criptare și decriptare implementat pe FPGA.**

**LUCRARE DE DIPLOMĂ**

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

Absolvent:  
Cristian-Rafael Tudusciuc

**Iași, 2024**



## **DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ**

Subsemnatul Tudusciuc Cristian Rafael,  
legitimat cu Cl seria VS nr. 964296, CNP 5020210374529  
autorul lucrării Accelerator hardware de criptare și decriptare implementat pe FPGA

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul  
de studii Calculatoare organizat de către Facultatea de  
Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din  
Iași, sesiunea Iulie a anului universitar 2024, luând în  
considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice  
„Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea  
Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este  
rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele  
bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a  
convențiilor internaționale privind drepturile de autor.

Data

29.06.2024

Semnătura





# Cuprins

<b>Introducere</b>	<b>1</b>
<b>1 Fundamentarea teoretică și documentarea bibliografică</b>	<b>3</b>
1.1 Domeniul și contextul abordării temei . . . . .	3
1.1.1 Ce este criptografia . . . . .	3
1.1.2 Criptografia în IoT . . . . .	4
1.1.3 Tehnologia FPGA . . . . .	5
1.1.4 Criptografia light-weight . . . . .	5
1.2 Tema propusă . . . . .	6
1.3 Realizările actuale pe aceeași temă . . . . .	6
1.3.1 Cifruri bloc studiate . . . . .	7
1.3.2 Analiza Comparativă . . . . .	8
1.3.3 Securitatea cifrurilor . . . . .	8
1.3.4 Flexibilitatea și adevararea aplicațiilor . . . . .	8
1.4 Elaborarea specificațiilor . . . . .	9
<b>2 Proiectarea aplicației</b>	<b>11</b>
2.1 Platforma utilizată . . . . .	11
2.2 Arhitectura sistemului . . . . .	11
2.2.1 Moduri de Implementare a Algoritmului Present . . . . .	12
2.2.2 Schema modulară . . . . .	13
2.3 Avantajele și dezavantajele metodei . . . . .	15
2.4 Componente software . . . . .	16
2.5 Componente hardware . . . . .	17
<b>3 Implementarea aplicației</b>	<b>19</b>
3.1 Modulele de accelerare hardware . . . . .	19
3.2 Împachetarea cu strat AXI . . . . .	20
3.3 Mediu de verificare în UVM . . . . .	21
3.4 Programul de test . . . . .	23
3.4.1 Operații cu numere mari . . . . .	24
<b>4 Testarea aplicației și rezultate experimentale</b>	<b>27</b>
4.1 Testarea prin interfață grafică . . . . .	27
4.2 Rezultate experimentale . . . . .	28
4.3 Elemente de configurare și instalare . . . . .	31
<b>Concluzii</b>	<b>33</b>
<b>Bibliografie</b>	<b>35</b>

<b>Anexe</b>	<b>37</b>
1 Componentele interfeței grafice . . . . .	37
2 Operații cu numere mari . . . . .	54
3 Mediu de verificare . . . . .	60
4 Componentele modulelor hardware . . . . .	80
5 Rezultate experimentale . . . . .	84

# **Accelerator hardware de criptare și decriptare implementat pe FPGA.**

**Cristian-Rafael Tudusciuc**

## **Rezumat**

În era digitală modernă, securitatea datelor este crucială, influențând direct integritatea și confidențialitatea informațiilor. Proiectul „Accelerator hardware de criptare și decriptare implementat pe FPGA” demonstrează eficiența implementării algoritmului de criptare Present pe FPGA-uri pentru a asigura securitatea datelor în dispozitive IoT și sisteme embedded. Alegerea algoritmului Present, cunoscut pentru eficiența sa în medii cu resurse limitate, a permis obținerea unor performanțe superioare în ceea ce privește viteza de procesare și consumul redus de energie comparativ cu soluțiile software tradiționale.

Proiectul a adoptat o abordare modulară, care a facilitat dezvoltarea și testarea eficientă a fiecărui component. Implementarea hardware a algoritmului Present pe FPGA a demonstrat că această soluție poate oferi un nivel înalt de securitate fără a compromite performanța. Mediul de verificare UVM a asigurat identificarea și corectarea problemelor potențiale, garantând conformitatea sistemului cu specificațiile inițiale.

Testele experimentale au confirmat robustețea și eficiența sistemului, evidențiind avantajele FPGA-urilor în domeniul criptografic. Soluția propusă se remarcă prin integrarea eficientă a componentelor hardware și software, utilizând un algoritm consacrat și oferind un echilibru optim între securitate, performanță și consum de resurse.

Proiectul subliniază potențialul FPGA-urilor în dezvoltarea soluțiilor criptografice eficiente și sigure, confirmând relevanța acestora în protejarea datelor în lumea digitală interconectată. Rezultatele obținute demonstrează aplicabilitatea soluțiilor FPGA în asigurarea securității în sistemele cu resurse limitate, evidențiind beneficiile acestora în contextul IoT.



## Introducere

În era digitală modernă, securitatea datelor reprezintă un pilon esențial al tehnologiei informației, influențând direct integritatea și confidențialitatea informațiilor. Cu expansiunea rapidă a Internet of Things (IoT) și a sistemelor embedded, necesitatea unor soluții criptografice eficiente și sigure a devenit mai evidentă ca niciodată. Dispozitivele IoT, fiind integrate în numeroase aspecte ale vieții cotidiene și industriale, operează adesea în medii cu resurse limitate, având nevoie de soluții care să nu compromită performanța sau durata de viață a bateriei.

În acest context, eficiența și viteza algoritmilor de criptare pot fi semnificativ îmbunătățite prin utilizarea circuitelor integrate specifice aplicațiilor (ASIC - Application Specific Integrated Circuits). Aceste circuite sunt proiectate pentru a realiza funcții specifice cu o eficiență maximă, fiind adaptate exact la cerințele aplicației pentru care sunt dezvoltate. Prin personalizarea hardware-ului pentru a implementa direct algoritmii de criptare se obțin performanțe superioare în ceea ce privește viteza de procesare și consumul energetic comparativ cu soluțiile software implementate pe hardware generalist.

Crearea unui ASIC implică mai mulți pași complecsi și interdependent, fiecare contribuind la dezvoltarea unui circuit eficient și fiabil. Procesul începe cu stabilirea specificațiilor, unde se definesc cerințele precise pe care trebuie să le îndeplinească circuitul. Acestea includ detalii legate de funcționalitate, performanță, consum de energie, dimensiuni și alte constrângeri tehnice.

Odată ce specificațiile sunt clare, următorul pas este definirea unei arhitecturi adecvate, pas ce implică elaborarea unei structuri de ansamblu a circuitului care să îndeplinească specificațiile stabilite. Arhitectura determină modul în care vor fi organizate și interconectate diferitele componente ale circuitului pentru a asigura funcționalitatea dorită.

Implementarea designului logic este etapa în care arhitectura definită este transpusă într-o descriere logică detaliată. Etapa include dezvoltarea schemelor și codului necesar pentru a descrie comportamentul circuitului la nivel logic. Designul logic trebuie să fie eficient și optimizat pentru a îndeplini cerințele de performanță.

Verificarea designului este un pas crucial care asigură că implementarea logică corespunde specificațiilor inițiale și că nu există erori. Verificarea implică simularea comportamentului circuitului în diverse scenarii și condiții de operare pentru a identifica și corecta eventualele probleme.

Sinteză este procesul prin care designul logic verificat este transformat într-un netlist, un set de conexiuni care descriu modul în care componente fizice ale circuitului vor fi interconectate.

Etapele ulterioare, cum ar fi layout-ul fizic, fabricarea propriu-zisă și testarea post-fabricare, nu fac parte din obiectivul acestei lucrări. După sinteză, netlist-ul rezultat este implementat pe FPGA pentru a valida și optimiza designul în condiții reale de operare.

FPGA-urile (Field-Programmable Gate Array) oferă un mediu flexibil și reconfigurabil care permite dezvoltatorilor să implementeze, testeze și optimizeze diferite module înainte de fabricarea finală a circuitului integrat. Aceasta reduce semnificativ timpul de dezvoltare, permitând ajustări rapide și eficiente în fazele inițiale de proiectare.

Acest proces asigură că produsul final va îndeplini toate cerințele de performanță și securitate, reducând riscurile și costurile asociate cu eventualele etape ulterioare.

Pentru a răspunde nevoilor specifice ale sistemelor embedded și IoT, se pot utiliza și algoritmi de criptare light-weight. Acești algoritmi sunt proiectați pentru a funcționa eficient în medii cu resurse limitate, asigurând în același timp un nivel adecvat de securitate. Algoritmii light-weight sunt esențiali pentru dispozitivele care operează cu putere redusă și au constrângeri stricte de memorie și procesare.

Algoritmul ales, în urma analizei soluțiilor existente, pentru această lucrare este Present, un algoritm simetric de criptare bloc pe 64 de biți cu o cheie de 80 biți. Present este un exemplu

excelent de algoritm light-weight, fiind conceput pentru a oferi un nivel ridicat de securitate cu un consum minim de resurse. Datorită arhitecturii sale eficiente și a necesităților reduse de hardware, Present este ideal pentru implementarea în circuitele integrate și pentru utilizarea în mediile cu resurse limitate caracteristice IoT și sistemelor embedded.

Lucrarea este structurată în patru capitole principale, fiecare abordând aspecte esențiale ale proiectului.

Primul capitol, Fundamentarea Teoretică, explorează concepțele fundamentale ale criptografiei, concentrându-se pe algoritmii simetриci și asimetrici, cu un accent special pe algoritmul Present. De asemenea, se discută despre importanța criptografiei light-weight în contextul IoT și al sistemelor embedded.

În al doilea capitol, Proiectarea și Implementarea Sistemului, se detaliază procesul de dezvoltare a modulelor de criptare și decriptare în Verilog, împachetarea acestora cu wrapper AXI, și integrarea cu procesorul MicroBlaze și alte periferice destinate transferului de date cu alte dispozitive. Se prezintă, de asemenea, arhitectura generală a sistemului și interconectarea componentelor.

Capitolul trei, Dezvoltarea Software-ului și Testarea Sistemului, descrie implementarea funcțiilor software pe procesorul MicroBlaze, incluzând algoritmul Diffie-Hellman și funcțiile de operare cu numere mari. Acest capitol detaliază și programul C++ dezvoltat pentru PC, care facilitează prezentarea funcționalității, testarea și validarea sistemului.

În final se prezintă Concluzii și Lucrări Viitoare, în care se sintetizează rezultatele obținute, evaluând performanța și eficiența sistemului. De asemenea, se discută despre posibile îmbunătățiri și direcții viitoare de cercetare, subliniind potențialul extinderii proiectului în alte aplicații de securitate.

Prin structura sa clară și detaliată, această lucrare oferă o abordare comprehensivă a implementării criptografiei în sisteme embedded, demonstrând relevanța și aplicabilitatea soluțiilor propuse în protejarea datelor iar utilizarea FPGA-urilor permite nu doar configurarea specifică a hardware-ului pentru cerințele particulare ale algoritmului Present, ci și o scalabilitate crescută, care facilitează adaptarea rapidă la evoluțiile tehnologice și la noi amenințări de securitate.

## Capitolul 1. Fundamentarea teoretică și documentarea bibliografică

În era digitală, protejarea informațiilor sensibile a devenit o prioritate majoră în toate domeniile, de la comunicații și finanțe, până la sănătate și apărare. Această protecție se realizează în mare parte prin utilizarea criptografiei, o ramură a matematicii și informaticii ce se ocupă cu tehniciile de securizare a datelor prin transformarea acestora într-o formă neinteligibilă pentru persoanele neautorizate. Criptografia este fundamentată pe principii matematice complexe și joacă un rol esențial în asigurarea confidențialității, integrității și autenticității informațiilor.

Acest capitol prezintă conceptele de bază ale criptografiei, contextul în care acestea sunt aplicate și importanța lor în diverse domenii, în special în cel al Internetului Lucrurilor (IoT). Se explorează diferitele tipuri de criptografie, avantajele și dezavantajele acestora, precum și aplicabilitatea lor în securizarea datelor mobile. În plus, se discută despre tehnologia FPGA (Field-Programmable Gate Array) și modul în care poate fi utilizată pentru implementarea accelerată a algoritmilor criptografici. Această tehnologie oferă flexibilitatea necesară pentru a testa și optimiza soluțiile criptografice încă dinainte de producția în masă, asigurând astfel performanță și eficiență energetică superioare.

### 1.1. Domeniul și contextul abordării temei

Criptografia nu este doar o metodă de a ascunde informații, ci un set complex de tehnici care, atunci când sunt implementate corect, pot asigura securitatea datelor într-o lume tot mai interconectată. În secțiunile următoare, se analizează în detaliu principiile fundamentale ale criptografiei, aplicațiile sale practice și modul în care acestea pot fi îmbunătățite prin utilizarea tehnologiilor moderne. Lucrarea se concentrează pe implementarea unui accelerator hardware de criptare și decriptare pe FPGA, o soluție ce promite să îmbunătățească semnificativ viteza și securitatea operațiunilor criptografice, aducând beneficii imediate în aplicațiile embedded și IoT.

#### 1.1.1. Ce este criptografia

Criptografia este știința și practica de protejare a informațiilor prin transformarea acestora într-o formă neinteligibilă pentru cei care nu au autorizație să le accesze. Aceasta implică utilizarea unor tehnici matematice complexe pentru a codifica sau decodifica date, asigurând astfel confidențialitatea, integritatea și autentificarea informațiilor. Criptografia nu este singura metodă de a asigura securitatea informațiilor, ci doar un set de tehnici utilizate în acest scop [1, Chapter 1].

Principalele tipuri includ criptografia simetrică, criptografia asimetrică și criptografia fără chei (vezi figura 1.1). Fiecare dintre aceste metode are propriile sale avantaje și dezavantaje, adaptându-se diferitelor nevoi de securitate.

Criptografia simetrică, cunoscută sub denumirea de criptografie cu cheie secretă, utilizează o singură cheie pentru atât criptarea, cât și decriptarea datelor. Algoritmii AES (Advanced Encryption Standard) sau DES (Data Encryption Standard) sunt exemple proeminente în acest domeniu. Principalul avantaj al criptografiei simetrice este viteza sa de procesare, ceea ce o face ideală pentru criptarea volumelor mari de date. Totuși, securitatea sa poate fi compromisă dacă cheia este interceptată de părți terțe. Astfel, managementul și distribuirea cheilor reprezintă o provocare majoră în utilizarea eficientă și sigură a criptografiei simetrice.

În contrast, criptografia asimetrică utilizează un set de două chei: una publică alături de una privată. Cheia publică este folosită pentru criptare, iar cheia privată pentru decriptare, exemplul clasic fiind algoritmul RSA (Rivest–Shamir–Adleman). Aceasta metodă oferă un nivel ridicat de securitate, deoarece chiar dacă cheia publică este cunoscută, cheia privată rămâne secretă și este necesară pentru decriptare. Criptografia asimetrică este mai sigură decât cea simetrică în ceea ce privește distribuția cheilor, însă are un dezavantaj major în ceea ce privește viteza de procesare,

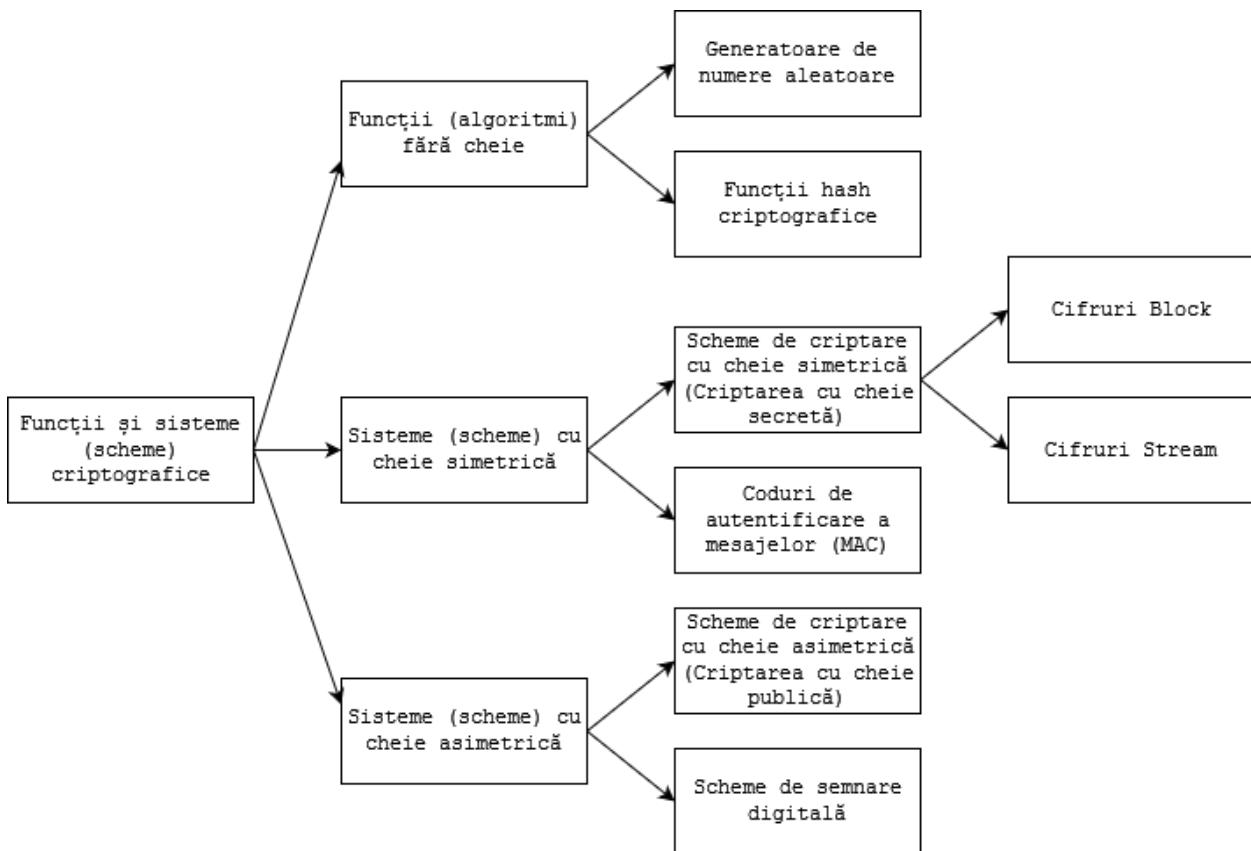


Figura 1.1. Primitive de ansamblu asupra sistemelor criptografice.[1]

fiind semnificativ mai lentă.

Criptografia fără chei reprezintă o abordare diferită, nefiind bazată pe utilizarea cheilor criptografice tradiționale pentru a cifra și descifra datele. Un exemplu de acest tip de criptografie este utilizarea funcțiilor de hash, care transformă datele originale într-o amprentă ireversibilă. Această metodă este folosită pentru verificarea integrității datelor, asigurându-se că nu au fost modificate. Un alt exemplu este criptografia bazată pe curbe eliptice, folosită în generarea de semnături digitale și acorduri de chei, exploatând proprietățile matematice ale curbelor eliptice pentru a asigura securitatea. Deși criptografia fără chei nu este utilizată pe scară largă pentru criptarea și decriptarea directă a datelor, joacă un rol crucial în asigurarea integrității și autenticității informațiilor.

### 1.1.2. Criptografia în IoT

Criptografia în dispozitivele mobile utilizează tehnici și tehnologii pentru a asigura securitatea și confidențialitatea datelor stocate și transmise. Aceasta protejează informațiile personale, precum contactele și mesajele, împotriva accesului neautorizat și a atacurilor cibernetice. Criptarea mesajelor și a apelurilor asigură că numai destinatarii intenționați pot accesa conținutul. De asemenea, criptarea memoriei interne și a cardurilor de memorie previne accesul neautorizat dacă dispozitivul este pierdut sau furat.

Aplicațiile mobile folosesc criptografia pentru autentificarea utilizatorilor, prin metode precum parole, amprente digitale și recunoaștere facială, asigurând că doar utilizatorii legitimi pot accesa aplicațiile și datele sensibile. Criptografia este esențială și în securizarea tranzacțiilor mobile, asigurând integritatea și confidențialitatea datelor financiare în timpul transferurilor de bani și a platilor mobile.

Pentru a asigura fluiditatea utilizării dispozitivelor mobile, este nevoie de algoritmi de criptare eficienți și rapizi. Acești algoritmi trebuie să aibă un consum redus de energie, astfel încât să

nu afectează durata de viață a bateriei dispozitivelor. La fel de esențial este ca timpul de calcul al acestor algoritmi să fie nesesizabil, pentru a nu îintrerupe sau încetini utilizarea normală a dispozitivului. O altă tehnică pentru a îmbunătăți performanța este accelerarea hardware prin intermediul circuitelor integrate specifice aplicațiilor (ASIC - application-specific integrated circuit), a căror prototipuri pot fi create folosind FPGA-uri. Acestea permit simularea și testarea eficientă a circuitelor specifice înainte de producția în masă, asigurând că soluțiile hardware sunt optimizate pentru sarcini criptografice specifice. Rezultatul nu doar îmbunătățește viteza de execuție, dar și reduce consumul de energie.

### *1.1.3. Tehnologia FPGA*

Tehnologia FPGA (Field-Programmable Gate Array) se referă la un tip de circuit integrat care poate fi programat după fabricare pentru a realiza diverse funcții logice. FPGA-urile sunt foarte flexibile și pot fi configurate să efectueze sarcini specifice prin reprogramare, făcându-le ideale pentru aplicații unde cerințele hardware se pot schimba sau pentru prototipuri. Totuși, măsurările empirice au cuantificat diferențele dintre FPGA-uri și ASIC-uri. S-a observat că pentru circuitele implementate exclusiv folosind LUT-uri și flip-flops (logic-only), un FPGA este în medie de 40 de ori mai mare și de 3,2 ori mai lent decât o implementare standard cell.

De asemenea, un FPGA consumă de 12 ori mai multă putere dinamică decât un ASIC echivalent în medie. S-a confirmat că utilizarea multiplicatorilor hardware și a memoriilor dedicate permite o reducere substanțială a ariei și a consumului de energie, dar aceste blocuri au un impact relativ minor asupra diferențelor de întârziere dintre ASIC-uri și FPGA-uri[2]. Această tehnologie are dezavantajul imposibilității de a oferi măsurători precise dar oferă o soluție robustă pentru testarea și optimizarea circuitelor integrate specifice aplicațiilor înainte de a trece la producția în masă, oferind astfel o eficiență sporită în termeni de performanță și consum de energie.

Majoritatea algoritmilor de criptare sunt inițial implementați în software datorită ușurinței și vitezei de dezvoltare. Această abordare permite iterarea rapidă, testarea și modificarea algoritmilor fără necesitatea de a investi timp și resurse în designul hardware complex. Cu toate acestea, implementările software sunt în general mai lente și mai costisitoare în ceea ce privește ciclurile de execuție, fiind limitate de performanțele unităților centrale de procesare. Pe măsură ce algoritmii devin maturi și cerințele de performanță cresc, aceștia sunt apoi implementați în hardware.

### *1.1.4. Criptografia light-weight*

Criptografia light-weight se concentrează pe dezvoltarea algoritmilor criptografici care sunt extrem de eficienți din punct de vedere al consumului de resurse. Acest domeniu este esențial pentru dispozitivele cu constrângeri de memorie și putere de calcul, cum ar fi cele din Internet of Things (IoT) și sistemele embedded. Dispozitivele IoT sunt omniprezente și cresc exponential în număr, necesitând soluții de securitate adecvate pentru a proteja datele sensibile transmise între dispozitive. În acest context, criptografia light-weight oferă un echilibru optim între securitate și utilizarea eficientă a resurselor limitate.

Implementarea acestor algoritmi ultra-ușori în hardware oferă mai multe avantaje. Accelerarea hardware poate îmbunătăți semnificativ operațiunile criptografice, reduce latența și scade consumul de energie prin efectuarea mai multor operații pe ciclu comparativ cu implementările software. Acest lucru este deosebit de important în aplicații precum procesarea datelor în timp real și comunicațiile securizate în rețelele de senzori, unde eficiența și capacitatea de reacție sunt critice. În plus, implementările hardware pot oferi securitate sporită prin reducerea opțiunilor de atac și îngreunarea tentativelor adversarilor de a manipula procesele criptografice.

Adoptarea criptografiei ultra-ușoare accelerate de hardware este esențială în diverse domenii, inclusiv automatizarea industrială, sănătate, orașe inteligente și sisteme auto, unde dispozitivele trebuie să comunice în mod securizat și eficient. Pe măsură ce numărul dispozitivelor conectate continuă să crească, importanța asigurării securității acestora fără a compromite performanța

devine tot mai critică. Prin valorificarea capabilităților criptografiei ultra-ușoare accelerate de hardware, dezvoltatorii pot crea soluții sigure, eficiente și scalabile care să răspundă cerințelor lumii moderne interconectate.

### 1.2. Tema propusă

Tema propusă pentru această lucrare este "Accelerator hardware de criptare și decriptare implementat pe FPGA". Acest subiect implică dezvoltarea și evaluarea unui dispozitiv dedicat, capabil să execute operațiuni de criptare și decriptare la viteze mult mai mari decât soluțiile software tradiționale și integrarea acestor module într-un sistem criptografic (vezi fig1.2). Implementarea pe FPGA permite o personalizare profundă a proceselor criptografice, oferind astfel un nivel înalt de performanță și securitate pentru diverse aplicații embedded.

Lucrarea de față nu își propune crearea unui nou algoritm de criptare, ci se concentrează pe adaptarea unui algoritm deja existent pentru a fi implementat eficient pe un FPGA. Alegerea unui algoritm consacrat asigură un nivel de securitate testat și verificat, permitând în același timp optimizarea acestuia pentru a profita de avantajele arhitecturale ale FPGA-ului. Astfel, obiectivul principal este de a demonstra cum un algoritm criptografic cunoscut poate fi adaptat și implementat într-un mod care să îmbunătățească semnificativ performanța și eficiența unui sistem embedded, fără a compromite securitatea acestuia. Această abordare pragmatică permite atât valorificarea cercetărilor anterioare, cât și realizarea unor implementări practice cu beneficii imediate.

Stabilirea algoritmului de criptare ce urmează a fi implementat implică o analiză atentă și detaliată a mai multor factori critici. În primul rând, este necesară evaluarea eficienței fiecărui algoritm din perspectiva resurselor necesare și a performanței. De asemenea, este esențială compararea nivelului de securitate oferit de fiecare algoritm pentru a asigura protecția eficientă a datelor împotriva atacurilor cibernetice. În cadrul acestei analize, sunt luate în considerare avantajele și dezavantajele fiecărui algoritm, inclusiv complexitatea implementării și consumul energetic.

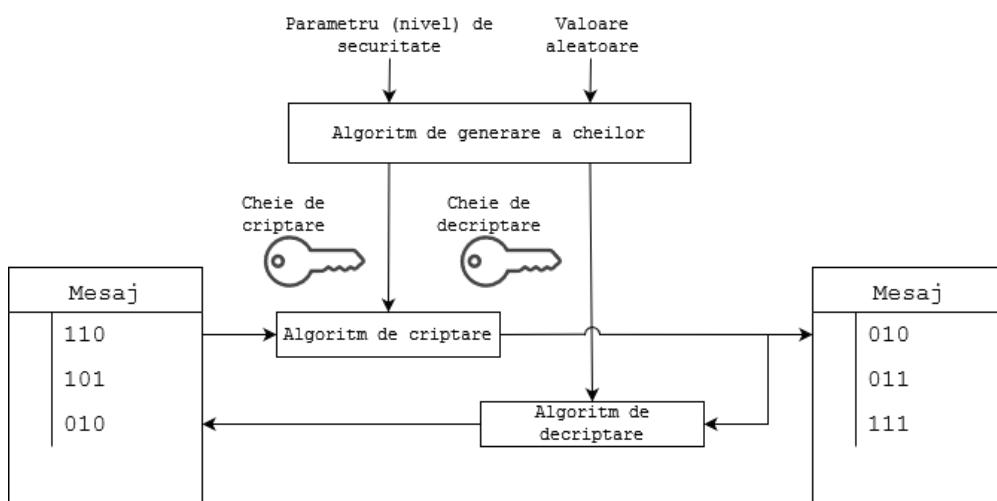


Figura 1.2. Sistem criptografic.[3]

### 1.3. Realizările actuale pe aceeași temă

În domeniul algoritmilor criptografici ultra-ușori, au fost propuse mai multe cifruri bloc pentru a răspunde la nevoile mediilor constrâns, cum ar fi etichetele RFID și rețelele de senzori. Dintre acestea, PRESENT, SKINNY, CAMELLIA, MANTIS, PICCOLO și SIMON se remarcă prin contribuțiile și caracteristicile lor de performanță unice. Această analiză comparativă explorează filozofile lor de design, caracteristicile de securitate și eficiențele de implementare, evidențiind realizările și avantajele lor relative.

### 1.3.1. Cifruri bloc studiate

- Cifrul Bloc PRESENT

PRESENT[4], conceput pentru eficiență hardware, este un cifru bloc compact cu o dimensiune a blocului de 64 de biți și suportă chei de 80 sau 128 de biți. Filozofia sa de design pune accentul pe simplitate și securitate, ceea ce îl face deosebit de potrivit pentru mediile cu resurse limitate. Cifrul utilizează o rețea de substituție-permutare (SPN) cu 31 de runde. Fiecare rundă constă dintr-un strat de permutare pe biți și un strat de substituție neliniară folosind un singur S-box de 4 biți aplicat în paralel de 16 ori pe rundă. Acest design asigură un nivel ridicat de securitate menținând în același timp un număr redus de porti, implementarea necesitând doar 1570 GE (Gate Equivalents).

- Cifrul Bloc SKINNY

SKINNY[5] este un alt cifru bloc ușor remarcabil, conceput pentru a atinge un nivel înalt de securitate cu un consum minim de resurse. Oferă versiuni cu dimensiuni diferite ale blocului (64 și 128 de biți) și suportă multiple dimensiuni ale cheilor. Designul SKINNY încorporează cadrul cifrului bloc modificabil, permitând o flexibilitate și securitate sporite. Structura cifrului include un program de chei minimalist și o funcție de rundă, rezultând o implementare foarte eficientă. În termeni de performanță, SKINNY depășește mulți dintre concurenții săi, inclusiv binecunoscutul cifru SIMON, în special în implementările ASIC bazate pe runde. Acest lucru îl face deosebit de potrivit pentru aplicații care necesită schimbări frecvente de chei și debit mare.

- Cifrul Bloc CAMELLIA

CAMELLIA[6] este un cifru bloc versatil, conceput pentru a oferi un nivel înalt de securitate pe diverse platforme. Suportă o dimensiune a blocului de 128 de biți cu chei de 128, 192 și 256 de biți. Designul este influențat de structurile Feistel și SPN, încorporând S-box-uri puternice și un program complex de chei pentru a asigura o securitate robustă împotriva atacurilor criptografice. Deși este mai intens din punct de vedere al resurselor comparativ cu PRESENT și SKINNY, caracteristicile de securitate cuprinzătoare ale CAMELLIA îl fac o alegere preferată pentru mediile unde securitatea este esențială și resursele sunt mai puțin constrânsă.

- Cifrul Bloc MANTIS

MANTIS[5] este un cifru bloc modificabil optimizat pentru implementări cu latență redusă, în special pentru criptarea memoriei. Dispune de o dimensiune a blocului de 64 de biți cu o cheie de 128 de biți și un modificador de 64 de biți. Designul se inspiră puternic din cifrele PRINCE și MIDORI, încorporând proprietatea de reflectie  $\alpha$ , care simplifică procesul de decriptare făcându-l practic reversul procesului de criptare. Această alegere de design reduce semnificativ costurile pentru decriptare. Cifrul MANTIS atinge un nivel înalt de securitate cu o latență minimă, folosind o versiune modificată a S-box-ului MIDORI și o permutare specifică a modificadorului, asigurând cel puțin 16 S-box-uri active pe rundă.

- Cifrul Bloc PICCOLO

PICCOLO[7] este un cifru bloc ușor conceput pentru medii cu constrângeri stricte de resurse. Are o dimensiune a blocului de 64 de biți și suportă chei de 80 și 128 de biți. Designul se bazează pe o rețea Feistel generalizată cu o structură de 25 de runde. Utilizează S-box-uri mici de 4 biți și un strat de permutare pentru a obține difuzie, rezultând o implementare extrem de compactă și eficientă. PICCOLO este cunoscut pentru cerințele sale reduse de spațiu, fiind potrivit pentru aplicații în etichete RFID și rețele de senzori. Securitatea sa a

fost bine studiată, arătând rezistență la atacuri criptografice comune, menținând în același timp o amprentă foarte mică în termeni de echivalenți de porti (GE).

- Cifrul Bloc SIMON

SIMON[8] face parte dintr-o familie de cifruri bloc ultra-ușoare, concepute de NSA pentru performanță optimă în implementări hardware și software. SIMON suportă diverse dimensiuni ale blocului și cheilor, versiunea cea mai comună având o dimensiune a blocului de 64 de biți și chei de la 64 la 256 de biți. Cifrul utilizează o structură Feistel simplă și eficientă cu o funcție de rundă minimalistă care se bazează pe AND bit cu bit, XOR și deplasări circulare. Această simplitate se traduce într-un număr excepțional de mic de porti și o eficiență ridicată, în special în implementările hardware. Cu toate acestea, securitatea SIMON a fost subiect de dezbatere, unele versiuni fiind susceptibile la criptanaliza diferențială și liniară.

#### *1.3.2. Analiza Comparativă*

În ceea ce privește eficiența hardware a cifrelor bloc ultra-ușoare, fiecare algoritm oferă avantaje distincte. PRESENT este optimizat pentru utilizarea minimă a hardware-ului, făcându-l o alegere excelentă pentru mediile extrem de constrânsă ocupând doar 1570 GE, semnificativ mai puțin decât multe alte cifre. Similar, SKINNY excelează în eficiență hardware, în special în implementările bazate pe runde, depășind SIMON în metricile de performanță și demonstrându-și adecvarea pentru aplicații care necesită atât securitate, cât și eficiență. În contrast, CAMELLIA, deși mai intensivă în resurse, oferă caracteristici de securitate robuste, potrivite pentru o gamă largă de platforme, făcându-l preferabil pentru aplicații care necesită securitate ridicată. MANTIS, optimizat pentru implementări cu latență redusă, atinge un echilibru între securitate și eficiență cu un număr de porti relativ mai mare comparativ cu PICCOLO, dar oferă o latență mai redusă. PICCOLO se remarcă prin designul său extrem de compact și eficient, cu un număr foarte mic de echivalenți de porti, fiind ideal pentru mediile extrem de constrânsă. În final, SIMON este cunoscut pentru numărul său excepțional de mic de porti și eficiență ridicată în hardware, făcându-l unul dintre cele mai prietenoase cifre în termeni de hardware disponibile.

#### *1.3.3. Securitatea cifrurilor*

În termeni de securitate, cifrele bloc prezintă grade variante de robustețe. PRESENT oferă un echilibru bun între securitate și eficiență cu structura sa SPN și designul compact, deși este recomandat în principal pentru aplicații cu securitate redusă. SKINNY oferă caracteristici de securitate sporite cu cadrul său de cifru bloc modificabil, făcându-l versatil pentru diverse aplicații, inclusiv cele care necesită schimbări frecvente de chei. CAMELLIA se remarcă prin caracteristicile sale puternice de securitate, inclusiv rezistență la o gamă largă de atacuri criptografice, făcându-l potrivit pentru aplicații în care securitatea nu poate fi compromisă. MANTIS oferă garanții puternice de securitate cu designul său modificabil și utilizarea eficientă a S-box-ului MIDORI, asigurând rezistență împotriva atacurilor diferențiale și liniare. PICCOLO oferă o securitate bună cu structura sa Feistel generalizată și S-box-urile bine analizate, potrivit pentru aplicații cu securitate redusă

spre medie. Cu toate acestea, deși SIMON este foarte eficient, unele versiuni au arătat vulnerabilități la criptanaliza diferențială și liniară, ridicând întrebări despre utilizarea lor în aplicații cu securitate ridicată.

#### *1.3.4. Flexibilitatea și adekvarea aplicațiilor*

În ceea ce privește flexibilitatea și adekvarea aplicațiilor, PRESENT este cel mai potrivit pentru aplicații ultra-ușoare unde resursele hardware sunt extrem de limitate. SKINNY oferă flexibilitate cu designul său modificabil și multiple dimensiuni ale cheilor, făcându-l potrivit pentru o gamă largă de aplicații, inclusiv cele care necesită gestionarea eficientă a cheilor. CAMELLIA

este ideal pentru aplicațiile care necesită securitate robustă pe diverse platforme, deși cu o disponibilitate mai mare de resurse. MANTIS este bine adaptat pentru aplicații de criptare cu latență redusă, cum ar fi criptarea memoriei, unde criptarea modificabilă este avantajoasă. PICCOLO este perfect pentru aplicații ultra-ușoare unde minimizarea spațiului hardware este crucială. În cele din urmă, SIMON este potrivit pentru o gamă largă de aplicații, în special unde eficiența hardware este primordială, dar limitările sale de securitate în anumite configurații trebuie luate în considerare.

Fiecare dintre aceste cifre bloc are puncte forte unice, iar alegerea între ele depinde în mare măsură de cerințele specifice ale aplicației, inclusiv echilibrul între securitate, eficiența hardware și flexibilitate. PRESENT este preferat pentru mediile extrem de constrânsse, SKINNY oferă o soluție versatilă și eficientă, CAMELLIA oferă securitate robustă pentru aplicațiile mai puțin constrânsse de resurse, MANTIS excellează în scenarii cu latență redusă, PICCOLO este ideal pentru utilizări ultra-ușoare, iar SIMON oferă o eficiență hardware neegalată, cu unele rezerve privind securitatea.

#### **1.4. Elaborarea specificațiilor**

Elaborarea specificațiilor reprezintă un pas crucial în orice proiect de dezvoltare, fie că vorbim de software, inginerie, construcții sau orice alt domeniu tehnic. Importanța acestui proces nu poate fi subliniată îndeajuns, deoarece specificațiile clar definite servesc drept fundament pentru întregul ciclu de viață al proiectului. Pe lângă beneficiile legate de gestionarea proiectului, specificațiile detaliate joacă un rol esențial și în asigurarea calității. Ele servesc drept referință pentru testarea și validarea produsului final.

Specificațiile acestui proiect sunt următoarele:

1. Aplicația dezvoltată trebuie să includă o capacitate robustă de criptare și decriptare a blocurilor de date pe 64 de biți. Acest aspect este esențial pentru asigurarea securității și integrității datelor gestionate de aplicație. Performanța algoritmului trebuie să fie optimizată pentru a asigura procesarea rapidă și eficientă a datelor, fără a compromite securitatea.
2. Pentru a facilita integrarea și utilizarea aplicației în diverse sisteme și medii, este crucial ca aceasta să ofere o interfață ușoară prin intermediul unei interfețe AXI (Advanced eXtensible Interface). Interfața AXI este cunoscută pentru flexibilitatea și performanța sa, permitând transferul rapid și eficient al datelor între diferite module și componente hardware. Utilizatorii aplicației vor beneficia de o conectivitate simplificată și de o interoperabilitate crescută, ceea ce va contribui la o adoptare mai rapidă și la o utilizare mai facilă a aplicației în diverse scenarii.
3. De asemenea, trebuie creată o bibliotecă software care să abstractizeze procesele complexe de criptare și decriptare, precum și interacțiunea cu interfața AXI. Această bibliotecă va facilita utilizarea modulelor aplicației, oferind funcții și metode clare și intuitive pentru utilizatori. Abstractia oferită de bibliotecă va ascunde detaliile tehnice și va permite dezvoltatorilor să se concentreze pe integrarea și utilizarea aplicației în proiectele lor, fără a fi necesar să înțeleagă toate aspectele tehnice ale implementării de bază. Prin simplificarea interacțiunii cu modulele aplicației, biblioteca software va contribui semnificativ la eficiența și productivitatea dezvoltatorilor.
4. Întreg ansamblul trebuie să fie scalabil întrucât scalabilitatea este o caracteristică esențială a aplicației, menită să asigure că sistemul poate crește și se poate adapta în funcție de creșterea volumului de date. Aceasta implică atât capacitatea de a procesa volume mari de date fără a compromite performanța sau fiabilitatea sistemului. Pentru a realiza acest lucru, aplicația trebuie să fie proiectată pe o arhitectură modulară, care să permită adăugarea și eliminarea de resurse în funcție de cerințele curente.

În contextul criptării și decriptării blocurilor de date pe 64 de biți, scalabilitatea înseamnă că aplicația trebuie să poată efectua aceste operațiuni rapid și eficient, chiar și atunci când

volumul de date crește semnificativ. Utilizarea unor algoritmi optimizați și implementarea unor tehnici de paralelizare vor fi esențiale pentru a asigura că performanța rămâne constantă.

Pe lângă scalabilitatea pe orizontală (adăugarea de noi instanțe ale aplicației pentru a gestiona sarcini ridicate), trebuie luată în considerare și scalabilitatea pe verticală, adică îmbunătățirea capacitaților hardware ale modulelor existente. Aceasta poate include actualizarea procesorului, memoriei și a capacitații de stocare pentru a sprijini volumele mai mari de procesare și stocare a datelor.

5. Un alt aspect important de luat în considerare este crearea unui mediu de verificare adecvat pentru simularea și testarea modulelor. Un astfel de mediu permite evaluarea riguroasă a fiecărui modul în parte, asigurând că toate funcționalitățile sunt verificate înainte de a fi integrate în sistemul principal. Prin simulări detaliate și teste exhaustive, se pot identifica și corecta eventualele deficiențe sau neconformități. În absența unui mediu de testare bine pus la punct, există riscul ca modificările aduse să nu fie pe deplin compatibile cu restul sistemului, ducând la erori de funcționare.

## Capitolul 2. Proiectarea aplicației

În realizarea unui accelerator hardware de criptare și decriptare pe FPGA, este esențial să se efectueze o planificare detaliată și riguroasă a tuturor aspectelor implicate, de la selecția platformei hardware până la implementarea componentelor software și hardware necesare. În acest capitol, este explorat în detaliu procesul de proiectare al aplicației, subliniind fiecare pas esențial, justificând alegerile făcute pentru a asigura o soluție optimă și eficientă.

### 2.1. Platforma utilizată

În urma analizei specificațiilor proiectului și a platformelor hardware disponibile, alături de limitările acestora, s-a decis utilizarea plăcii de dezvoltare Nexys A7-100T, bazată pe FPGA-ul Xilinx Artix-7. Această platformă oferă performanță și flexibilitate în ceea ce privește perifericele disponibile dar și a capabilității de implementare a unei arhitecturi hardware de dimensiuni medii.

Totodată, prin intermediul bibliotecilor de module predefinite (IP), precum AXI Uartlite, AXI EthernetLite, AXI Interconnect, MicroBlaze sau AXI GPIO, integrate în Vivado, se poate crea o arhitectură modulară și adaptabilă. Aceste IP-uri predefinite permit configurarea rapidă și eficientă a interfețelor de comunicație și control, asigurând astfel o interconectare optimă între diferitele componente ale sistemului. Flexibilitatea oferită de aceste module IP facilitează dezvoltarea unei arhitecturi hardware personalizate, capabile să răspundă cerințelor specifice de performanță și funcționalitate ale diverselor medii ce necesită soluții de criptare și decriptare.

Un alt aspect important ce nu trebuie ignorat în implementarea unui accelerator hardware de criptare și decriptare este frecvența de funcționare a sistemului criptografic. În cazurile în care se pot folosi platforme de scop general, cum ar fi procesoarele cu arhitectură x86 sau ARM, care pot atinge frecvențe de operare în ordinul gigahertzilor, variantele software ale algoritmilor criptografici devin o opțiune viabilă și eficientă. Aceste procesoare sunt capabile să execute rapid instrucțiuni complexe, oferind performanțe ridicate în criptare și decriptare, chiar și pentru volume mari de date.

În contrast, sistemele dedicate bazate pe FPGA-uri, cum este placa de dezvoltare Nexys A7-100T, operează de obicei la frecvențe mai reduse, în ordinul megahertzilor. Deși FPGA-urile oferă o flexibilitate considerabilă și pot fi optimizate pentru sarcini specifice, inclusiv criptografia, limitările de frecvență pot influența performanța generală a sistemului criptografic. În acest context, FPGA-urile pot să nu fie la fel de rapide ca procesoarele de scop general pentru anumite aplicații de criptare și decriptare, în special când se procesează cantități mari de date într-un timp scurt.

Cu toate acestea, sistemele FPGA prezintă avantaje semnificative în ceea ce privește personalizarea și optimizarea pentru aplicații specifice. De exemplu, algoritmii criptografici pot fi implementați direct în hardware, eliminând necesitatea interpretării instrucțiunilor software și reducând astfel latența. Această caracteristică poate compensa parțial limitările de frecvență, oferind o performanță consistentă și predictibilă în criptare și decriptare.

### 2.2. Arhitectura sistemului

Deși una dintre abordările posibile în implementarea unui accelerator hardware de criptare și decriptare ar putea fi crearea unui modul monolitic capabil să gestioneze atât criptarea și decriptarea, cât și interfațarea cu exteriorul, gestionarea și procesarea mesajelor și memoriei, această metodă prezintă anumite dezavantaje semnificative. Într-adevăr, un modul monolitic ar putea reduce dimensiunea finală ocupată pe FPGA, consolidând toate funcționalitățile într-un singur bloc logic. Totuși, complexitatea crescută a designului monolitic poate introduce dificultăți considera-

bile în procesul de implementare și modificare ulterioară a sistemului.

Un design monolitic complică semnificativ procesul de dezvoltare, deoarece integrează multiple funcționalități într-un singur modul. Acest lucru poate face mai dificilă identificarea și rezolvarea problemelor care apar în timpul testării și depurării. Orice modificare necesară pentru adaptarea la un alt context sau pentru îmbunătățirea performanței trebuie realizată în cadrul unui sistem complex și interdependent, ceea ce poate duce la creșterea timpului și a costurilor de dezvoltare.

În contrast, o abordare modulară, în care funcțiile de criptare, decriptare, interfațarea cu exteriorul și gestionarea și procesarea mesajelor și memoriei sunt implementate în module separate, oferă mai multe beneficii. Această metodă simplifică procesul de dezvoltare și testare, permitând izolarea și optimizarea individuală a fiecărui modul. Modificările necesare pentru adaptarea la diferite contexte pot fi realizate mai ușor, fără a afecta întregul sistem.

Astfel, deși un modul monolitic ar putea părea avantajos din perspectiva dimensiunii finale ocupate și a eficienței energetice, complexitatea și dificultățile asociate în dezvoltare și întreținere nu justifică întotdeauna această abordare în contextul unui accelerator hardware de criptare și decriptare. O strategie modulară oferă o soluție mai flexibilă, eficientă și ușor de întreținut, care se aliniază mai bine cu cerințele sistemelor embedded moderne.

### 2.2.1. Moduri de Implementare a Algoritmului Present

În teorie, algoritmul Present poate fi implementat în mai multe moduri, fiecare având avantaje și dezavantaje specifice. Primul mod presupune efectuarea calculelor în 31 de cicluri pe blocuri de 64 de biți, folosind în mod repetat rețeaua SP (Substituție-Permutare) (Fig. 2.2). Acest mod este detaliat în documentul de referință pentru Present, unde fiecare ciclu implică aplicarea secvențială a operațiunilor de adăugare a cheii de rundă, stratul de substituție și stratul de permutare. Această abordare utilizează un singur set de hardware pentru rețeaua SP, care este folosit repetat pentru fiecare dintre cele 31 de runde, rezultând într-o implementare eficientă din punct de vedere al resurselor.

Algoritm 2.1. Pseudo cod Present

```

1: GenereazaChei()
2: for i = 1 to 31 do
3:   adaugaCheie(STARE, Ki)
4:   stratSubstitutie(STARE)
5:   stratPermutare(STARE)
6:   adaugaCheie(STARE, K32)

```

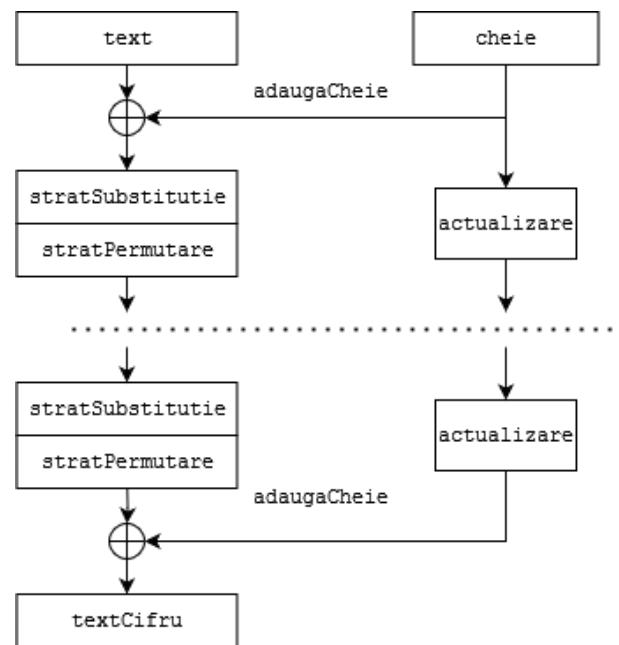


Figura 2.1. Descrierea algoritmului Present.

Al doilea mod de implementare presupune înșiruirea rețelei SP pentru toți cei 31 de pași ai algoritmului. În această abordare, Present poate fi reprezentat sub forma unui circuit combinațional, unde toate runde sunt desfășurate simultan. Astfel, calculele pentru un bloc de 64 de biți sunt reali-

zate într-un singur ciclu. Această metodă, deși foarte rapidă în termeni de număr de cicluri de ceas necesari pentru criptare sau decriptare, poate necesita o frecvență de operare mai scăzută datorită vitezei de propagare a semnalelor prin multiplele porți logice implicate. Complexitatea crescută a circuitului și cerințele mai mari de resurse hardware pot reprezenta dezavantaje semnificative ale acestei abordări.

Deși metoda de înșiruire a rețelei SP poate oferi un timp de procesare mai rapid, trebuie să se ia în considerare impactul asupra frecvenței de operare și utilizării resurselor. Viteza de propagare a informațiilor prin porțile logice devine un factor limitativ, ceea ce poate necesita o frecvență mai scăzută pentru a asigura stabilitatea și corectitudinea operațiunilor. Aceasta poate contrabalașa beneficiul obținut prin reducerea numărului de cicluri necesari pentru procesarea unui bloc de date.

Este important de menționat că, precum a fost stabilit în primul capitol, aspectele legate de metoda de fabricație a circuitului integrat nu sunt considerate în această lucrare. Aceasta înseamnă că implementarea practică și optimizarea la nivel de siliciu a acestor două metode nu sunt discutate aici, concentrându-ne în schimb pe arhitectura logică și performanțele teoretice ale algoritmului Present în contextul sistemelor embedded și FPGA-urilor. Astfel, alegerea metodei de implementare depinde de constrângerile specifice ale aplicației și de resursele hardware disponibile, fiecare abordare oferind compromisuri între complexitate, performanță și utilizarea resurselor.

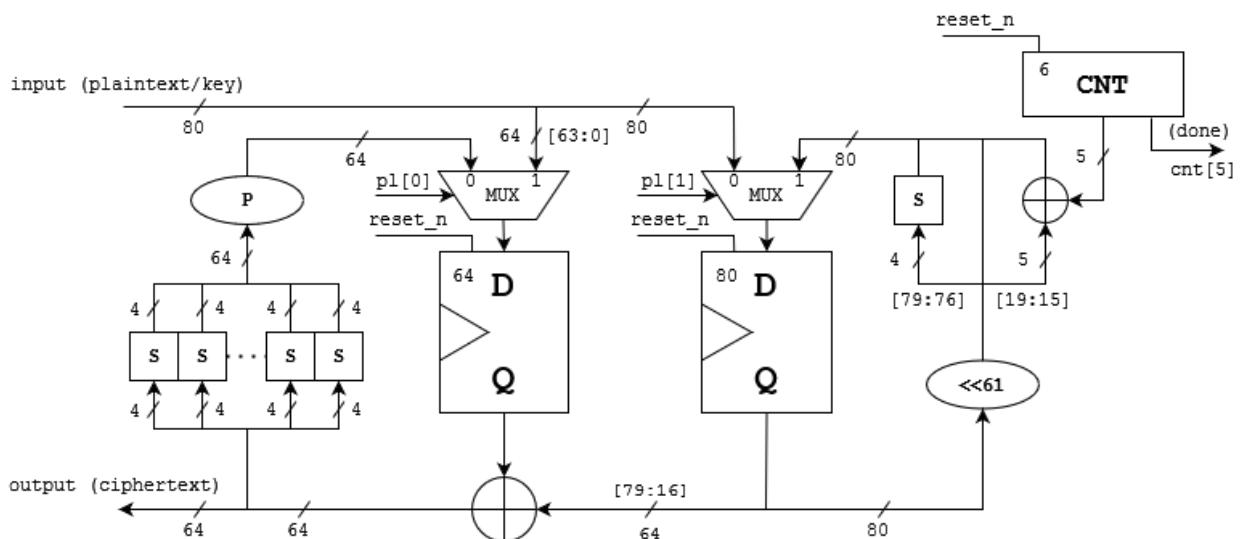


Figura 2.2. Calea de date a algoritmului de criptare Present, versiunea optimizată pentru suprafață.

### 2.2.2. Schema modulară

Sistemul modular este construit în jurul unui procesor MicroBlaze, care joacă un rol crucial în coordonarea eficientă a componentelor principale. Acest procesor RISC soft, specific arhitecturii FPGA, asigură o flexibilitate remarcabilă și o performanță optimizată, facilitând integrarea și gestionarea diferitelor module ale sistemului. Datorită MicroBlaze, sistemul beneficiază de o capacitate sporită de adaptare și scalabilitate, permitând actualizări și modificări rapide în funcție de nevoile specifice ale aplicației, fără a compromite stabilitatea și eficiența operațională.

Componentele funcționale necesită prezenta unui modul de gestionare a semnalelor de ceas precise și stabile pentru întregul sistem. Acesta trebuie să se ocupe de resetarea și sincronizarea ceasurilor interne, asigurându-se că toate componentele funcționează sincron și la frecvență corectă. Pe de altă parte, este necesar un modul care gestionează semnalele de resetare necesare pentru inițializarea corectă a sistemului. Acesta va coordona resetarea procesorului MicroBlaze și a componentelor periferice, garantând că întregul sistem pornește într-o stare definită și stabilă, prevenind astfel comportamentele neașteptate sau erorile de inițializare.

La fel de esențial pentru buna funcționare a sistemului este un modul de gestionare a între-ruperilor generate de diferitele componente periferice ce trebuie redirecționate către procesorul MicroBlaze. Prin intermediul acestui controller de întreruperi, sistemul poate răspunde rapid și eficient la evenimentele externe, prioritizând și gestionând multiple surse de întreruperi. Modulul asigură că întreruperile sunt tratate în ordine și conform importanței lor, prevenind astfel conflictele și asigurând o operare fluidă a întregului sistem.

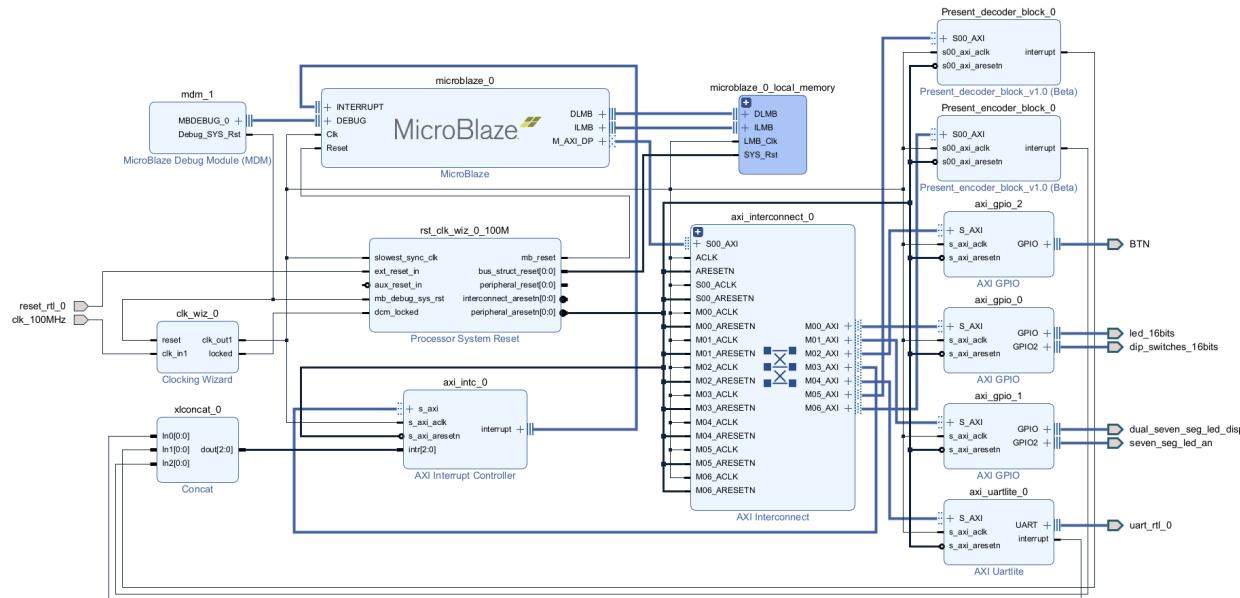


Figura 2.3. Arhitectura modulara a sistemului.

Conecțarea tuturor modulelor într-o topologie totală conectată ar prezenta o complexitate inutilă, având în vedere că fiecare modul ar trebui să fie conectat direct la fiecare alt modul. Aceasta ar crește exponential numărul de conexiuni și ar face dificilă gestionarea sistemului, atât din punct de vedere al costurilor, cât și al performanței. Într-o astfel de rețea, ar fi nevoie de resurse considerabile pentru a asigura comunicarea eficientă între module, ceea ce ar putea duce la o suprasolicitare a infrastructurii hardware și la întârzieri în procesarea datelor.

Chiar și o topologie de tip stea, în care toate modulele sunt conectate la un procesor central, poate fi îmbunătățită. Într-o topologie stea, deși complexitatea este redusă față de o rețea totală conectată, procesorul central devine rapid aglomerat de interfețe de conectare, fapt ce limitează scalabilitatea sistemului.

Pentru a aborda această problemă, o topologie de tip crossbar poate oferi o soluție mai eficientă. În contrast cu topologia stea, topologia crossbar nu presupune conectarea directă a tuturor modulelor între ele. În schimb, crossbar-ul permite selectarea dinamică a rutelor de comunicare între module. Fiecare modul poate comunica direct cu un alt modul prin intermediul interconectărilor configurabile, fără a trece printr-un nod central.

Pentru comunicarea cu exteriorul, se poate alege între un modul Ethernet sau un modul UART. Ambele asigură transmiterea serială de informații într-un mod robust, fiecare având avantaje și dezavantaje specifice. Modulul Ethernet oferă viteze mari de comunicare, ceea ce îl face ideal pentru aplicațiile care necesită transfer rapid de date. Cu toate acestea, acest beneficiu vine la pachet cu un nivel mai ridicat de complexitate în implementare și utilizare, necesitând o cunoaștere avansată a protocolelor de retea și a configurației corespunzătoare.

Pe de altă parte, modulul UART (Universal Asynchronous Receiver-Transmitter) este cunoscut pentru simplitatea sa, fiind mult mai ușor de implementat și utilizat în comparație cu modulul Ethernet. Deși nu dispune de beneficiul vitezei mari de transmitere, acest modul compensează

printr-o complexitate redusă fiind mai accesibil pentru proiectele de testare și pentru demonstrarea conceptelor de bază în comunicarea serială.

În scop de testare și demonstrare a conceptului, s-a optat pentru utilizarea unui modul UART. Această alegere este justificată prin necesitatea de a menține simplitatea implementării, permitând o concentrare pe aspectele esențiale ale lucrării. Modulul UART facilitează atât trimiterea, cât și recepționarea datelor, oferind flexibilitate în configurarea vitezei de comunicare și a opțiunilor de paritate pentru corectarea erorilor de transmisie.

Tot în scopuri demonstrative, este de dorit atașarea unui modul general de intrare-ieșire (GPIO) pentru controlul unui display cu șapte segmente, care să afișeze informații utile direct de pe platforma de dezvoltare. Acest aranjament permite vizualizarea în timp real a datelor și a stării sistemului, facilitând înțelegerea modului de funcționare a circuitului și oferind un feedback vizual imediat pentru utilizator.

Modulele principale care constituie tema acestei lucrări sunt cele de accelerare a criptării și decriptării, care necesită integrarea cu o interfață AXI pentru a putea fi configurate și utilizate ca dispozitive de intrare și ieșire. Aceste module sunt esențiale pentru asigurarea unei securități sporite și a unei eficiențe superioare în manipularea datelor sensibile, oferind o soluție robustă pentru criptarea și decriptarea rapidă a informațiilor.

Din perspectiva procesorului care le coordonează, aceste module sunt abstractizate și percepute ca locații de memorie unde se pot scrie date pentru a fi prelucrate și de unde se pot citi datele criptate sau, invers, decriptate. Această abordare simplifică gestionarea modulelor și permite procesorului să interacționeze cu ele într-un mod eficient, tratându-le ca pe niște resurse standard de memorie.

Pentru a evita blocarea procesorului în așteptarea finalizării operațiilor de criptare sau decriptare, este necesară implementarea unui semnal de întrerupere către procesor. Acest semnal de întrerupere va notifica procesorul imediat ce operațiunea a fost finalizată, permitând astfel procesorului să își continue alte sarcini fără a fi nevoie să verifice constant starea modulelor. Prin această metodă, se asigură o utilizare optimă a resurselor de procesare și se îmbunătățește considerabil performanța globală a sistemului.

### **2.3. Avantajele și dezavantajele metodei**

Schema modulară adoptată pentru implementarea acceleratorului hardware de criptare și decriptare pe FPGA oferă o serie de avantaje semnificative. Flexibilitatea și modularitatea reprezintă beneficii majore, deoarece separarea funcționalităților în module distințe pentru criptare, decriptare, interfațarea cu exteriorul și gestionarea mesajelor permite un design mai ușor de întreținut. Fiecare modul poate fi dezvoltat, testat și optimizat independent, facilitând adaptarea rapidă la noi cerințe sau îmbunătățiri ale sistemului.

În ceea ce privește eficiența în dezvoltare și depanare, designul modular simplifică identificarea și rezolvarea problemelor. Erorile pot fi izolate și abordate fără a afecta întregul sistem, ceea ce face ca procesul de testare și validare să fie mai riguros și detaliat. Fiecare modul poate fi testat separat, permitând o validare mai precisă a fiecărei funcții individuale înainte de integrarea lor în sistemul complet.

Performanța și securitatea sunt, de asemenea, avantaje notabile ale metodei alese. Fiecare modul poate fi optimizat pentru sarcinile specifice pe care le îndeplinește, asigurând performanțe ridicate și eficiență energetică. Utilizarea algoritmului Present în module dedicate pentru criptare și decriptare asigură un nivel înalt de securitate, potrivit pentru aplicațiile embedded și IoT. În plus, designul modular permite adăugarea ușoară de noi funcționalități sau module, ceea ce face ca sistemul să fie scalabil și adaptabil la evoluțiile tehnologice și noi cerințe de securitate.

Cu toate acestea, metoda modulară prezintă și câteva dezavantaje. Dezvoltarea unei arhitecturi modulare poate fi mai complexă inițial, necesitând o planificare detaliată a interacțiunilor dintre module și a fluxurilor de date. Asigurarea unei comunicări eficiente și coordonate între module

poate adăuga un nivel suplimentar de complexitate în designul sistemului. Utilizarea interfețelor AXI pentru comunicarea dintre module poate introduce un surplus de semnale echivalent unui surplus în utilizarea resurselor FPGA ce duce la o cantitate mai mare a resurselor totale comparativ cu un design monolitic.

Pe lângă complexitatea de design, performanța în timp real poate fi afectată. Descompunerea funcționalităților în module separate poate introduce latență suplimentară datorită necesității de transfer de date între module prin interfața AXI. De asemenea, frecvența de operare poate fi limitată de viteza de propagare a semnalelor prin multiple module și interfețe. În final, dezvoltarea și testarea fiecărui modul separat poate duce la costuri și timp suplimentar de dezvoltare inițială. Integrarea modulelor separate într-un sistem complet și validarea acestuia poate necesita eforturi suplimentare pentru a asigura compatibilitatea și funcționarea corectă a întregului sistem.

#### **2.4. Componente software**

Acest sub-capitol explorează aspectele esențiale ale componentelor software necesare pentru implementarea acceleratorului hardware de criptare și decriptare. Dezvoltarea software-ului joacă un rol crucial în asigurarea funcționării corecte și eficiente a sistemului, permitând gestionarea comunicării între modulele hardware, implementarea algoritmilor criptografici și asigurarea interacțiunii cu utilizatorii și dispozitivele externe.

În cadrul lucrării, au fost utilizate diverse componente software pentru a asigura capabilitatea completă a sistemului. Fiecare componentă a fost selectată și integrată cu atenție pentru a garanta performanța optimă. De la dezvoltarea inițială până la testarea finală, a fost asigurată o colaborare eficientă între toate modulele software pentru a crea un sistem coerent și robust.

- Vivado

Una dintre componentele esențiale a fost Vivado, un instrument puternic dezvoltat de Xilinx, utilizat pentru sinteza și implementarea limbajului de descriere hardware Verilog în fișierul bitstream. Vivado oferă un mediu integrat de dezvoltare (IDE) care simplifică semnificativ procesul de proiectare a circuitelor integrate. Fișierul bitstream conține toate datele necesare pentru configurarea FPGA-ului.

Deși Vivado permite simularea designului, nu oferă suport deplin pentru funcționalitățile limbajului de descriere și verificare SystemVerilog pe care este bazată Metodologia Universală de Verificare (UVM). Din această cauză, mediul de verificare bazat pe UVM a trebuit să fie creat prin alte metode pentru a se asigura că toate cerințele proiectului sunt îndeplinite.

- EDA Playground

Soluția aleasă a fost platforma online EDA Playground. Aceasta oferă diverse configurații în funcție de limbajul de descriere și compilatorul dorit, precum și de biblioteci disponibile. Pentru limbaj, s-a ales SystemVerilog pentru testbench și Verilog pentru design, utilizând simulatorul Synopsys VCS 2023.03 și biblioteca UVM 1.2. Această combinație permite beneficierea de performanțe semnificative și flexibilitate în proiectare, oferind un mediu robust pentru verificarea designului hardware.

În ciuda avantajelor considerabile oferite de platforma EDA Playground, există și anumite limitări. Una dintre acestea este cantitatea de date care se poate simula, fiind restricționată de capacitatea platformei online. Aceasta impune unele constrângeri asupra dimensiunii și complexității testelor care pot fi executate. Cu toate acestea, platforma furnizează un mediu adecvat pentru dezvoltarea inițială și verificarea prototipurilor, permitând identificarea și remedierea problemelor critice într-un stadiu timpuriu al dezvoltării.

Astfel, utilizând EDA Playground și instrumentele sale avansate, verificarea designului a fost completată, asigurându-se că acesta respectă toate specificațiile și cerințele. În viitor,

pentru simulări mai complexe și pentru a depăși limitările platformei online, ar putea fi necesară utilizarea unor resurse hardware și software mai avansate, care să ofere o capacitate de simulare mai mare și să sprijine pe deplin metodologia UVM.

- Vitis

Pe lângă Vivado a fost utilizat și Vitis, un alt instrument dezvoltat de Xilinx, care oferă funcționalități extinse pentru programarea în limbajul C a microprocesorului MicroBlaze integrat în FPGA. Acest instrument puternic permite dezvoltatorilor să creeze aplicații software complexe direct pe platformele FPGA, facilitând astfel gestionarea eficientă a resurselor hardware.

Vitis a jucat un rol crucial în dezvoltarea și implementarea codului software necesar pentru gestionarea funcțiilor de criptare și decriptare pe placa Nexys A7. Utilizarea acestui mediu integrat de dezvoltare a permis programarea facilă a memoriei flash a plăcii, asigurând astfel persistența configurației și a aplicației indiferent de alimentarea plăcii la o sursă de curent.

Acest mediu integrat de dezvoltare oferă un set complet de unelte pentru dezvoltarea de aplicații software pe platformele FPGA Xilinx. De asemenea simplifică procesul de dezvoltare prin abstractizarea detaliilor complexe ale gestionării adreselor resurselor hardware. Aceasta duce la o dezvoltare mai rapidă și mai eficientă a soluțiilor inovatoare, maximizând astfel potențialul tehnologiilor FPGA.

- Microsoft Visual Studio

Pentru dezvoltarea programului de testare și interfațare cu hardware-ul prin UART, a fost utilizat mediul integrat de dezvoltare (IDE) Microsoft Visual Studio. Aceasta reprezintă o soluție completă pentru programatorii C++, oferind un set de instrumente avansate care simplifică procesul de scriere, compilare și depanare a codului.

În cadrul proiectului, Microsoft Visual Studio a fost esențial pentru gestionarea eficientă a comunicației seriale prin UART, folosind Windows API. Această metodă a permis programului C++ să interacționeze direct cu porturile seriale ale sistemului, asigurând un control precis și fiabil al transferului de date. Funcționalitățile avansate ale IDE-ului, cum ar fi IntelliSense și debugger-ul grafic, au facilitat identificarea și remedierea rapidă a problemelor, îmbunătățind calitatea generală a codului și reducând timpul de dezvoltare.

Programul C++ dezvoltat în Visual Studio a fost conceput pentru a facilita interacțiunea cu acceleratorul hardware, permitând testarea și validarea funcționalităților de criptare și decriptare prin intermediul interfeței UART. Aceasta a inclus implementarea unor rutine eficiente de citire și scriere, optimizate pentru a funcționa în timp real, asigurând astfel performanțe ridicătoare și fiabilitate în comunicarea cu hardware-ul specializat.

În plus, utilizarea extensiilor WinForms din Visual Studio poate permite dezvoltarea unei interfețe grafice prietenoase, care să oferă utilizatorilor posibilitatea de a monitoriza statusul comunicațiilor seriale și de a ajusta parametrii de funcționare fără a necesita modificări directe în codul sursă. Această abordare nu doar că îmbunătățește experiența utilizatorului, dar și adaugă un nivel suplimentar de flexibilitate și eficiență în gestionarea și testarea sistemelor hardware.

## 2.5. Componente hardware

Componentele hardware utilizate pentru implementarea acceleratorului de criptare și decriptare pe FPGA sunt esențiale în asigurarea performanței și eficienței necesare pentru realizarea operațiunilor criptografice complexe.

Placa Nexys A7-100T este compatibilă cu suitele de design Xilinx Vivado și ISE, permitând utilizarea unor unelte avansate pentru dezvoltarea și simularea proiectelor FPGA. În plus, dispune

de multiple dispozitive I/O, cum ar fi 16 comutatoare, 16 LED-uri, două LED-uri RGB și două afișaje cu șapte segmente de câte patru cifre. Aceste caracteristici fac ca Nexys A7 să fie potrivită pentru o gamă largă de aplicații, de la circuite combinaționale introductive până la procesoare embedded puternice.

Nexys A7-100T este echipată cu FPGA-ul Xilinx Artix-7, care dispune de 15.850 de logic slices, fiecare având patru LUT-uri (Look-Up Tables) de 6 intrări și 8 flip-flop-uri. Aceasta permite realizarea unor designuri logice complexe și optimizate pentru criptare și decriptare. De asemenea, FPGA-ul include 1.188 Kb de RAM bloc, 240 de DSP slices și șase unități de management al ceasului (clock management tiles), fiecare având un PLL (Phase-Locked Loop). Aceste caracteristici asigură suportul necesar pentru operațiuni rapide și eficiente, esențiale pentru implementarea algoritmului Present. Memorie și Stocare

Placa Nexys A7-100T este dotată cu o memorie externă DDR2 de 128 MiB, care oferă spațiu suficient pentru stocarea temporară a datelor necesare în timpul procesării criptografice. De asemenea, placa include un slot pentru carduri microSD, oferind opțiuni aditionale de stocare și acces la date, ceea ce poate fi util în aplicațiile care necesită manipularea unor volume mari de date criptate. Conectivitate

Placa dispune de multiple opțiuni de conectivitate, inclusiv porturi USB și Ethernet, care facilitează programarea și comunicația cu alte dispozitive. Portul USB este utilizat pentru programarea FPGA-ului și pentru interacțiunea cu software-ul de dezvoltare, cum ar fi Vivado și Vitis. Conectivitatea Ethernet permite integrarea plăcii în rețele mai mari, facilitând transferul de date criptate și decriptate într-un mediu conectat. Periferice Integrate

Nexys A7-100T include diverse periferice integrate care sunt esențiale pentru interacțiunea cu utilizatorul și monitorizarea sistemului. Printre acestea se numără un accelerometru, un senzor de temperatură, un microfon digital MEMS, un amplificator pentru difuzor și multiple dispozitive I/O, cum ar fi 16 comutatoare, 16 LED-uri RGB și două afișaje cu șapte segmente de câte patru cifre. Aceste periferice permit dezvoltarea unor aplicații interactive și oferă feedback vizual și auditiv în timp real asupra stării și funcționării sistemului.

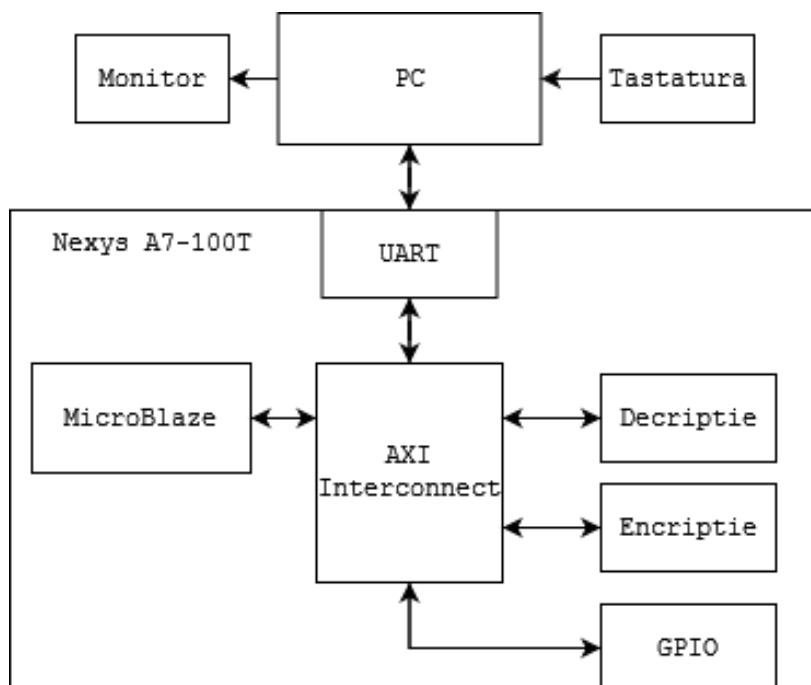


Figura 2.4. Schema bloc a componentelor hardware.

## Capitolul 3. Implementarea aplicației

În acest capitol se descrie modul de implementare a lucrării, pornind de la modulele hardware criptografice descrise în Verilog și până la modul de funcționare a programelor concepute atât pentru procesorul Microblaze, cât și pentru un computer care comunică cu sistemul criptografic. Procesul de implementare include detalii despre proiectarea și integrarea modulelor hardware, precum și despre dezvoltarea software-ului necesar pentru a asigura o comunicare eficientă între componentele sistemului. Se pune accent pe descrierea arhitecturii generale, pe interfațele de comunicare și pe metodele utilizate pentru testarea și validarea funcționării corecte a întregului sistem criptografic.

### 3.1. Modulele de accelerare hardware

Principalul obiectiv urmărit este dimensiunea circuitului rezultat, să că dintre cele două metode de implementare prezентate în capitolul 2, pentru aceasta s-a ales folosirea în mod repetat a aceleiași rețele SP. Modulul de criptare folosește calea de date prezentată în figura 2.1, ceea ce permite utilizarea eficientă a resurselor hardware și menținerea dimensiunii circuitului la un nivel minim. Această abordare asigură o utilizare repetată a componentelor existente, reducând necesitatea de a include componente suplimentare și, astfel, contribuind la un design mai compact și mai eficient.

În schimb, modulul de decriptare necesită logică adițională de gestionare a cheii. Dacă algoritmul de criptare pornește de la cheia  $K_0$  ce trece prin cei 31 de pași ajungând la  $K_{32}$ , la fiecare pas  $K_i$  contribuie la criptarea textului prin confuzie. Acest lucru înseamnă că pentru decriptare, trebuie să se revină în mod corect la fiecare cheie intermediară până la cheia inițială,  $K_0$ . Algoritmul de decriptare primește ca intrare tot  $K_0$  și necesită actualizarea cheii până la  $K_{32}$  pentru a putea începe decriptarea, ceea ce adaugă un nivel suplimentar de complexitate în gestionarea cheilor și, totodată, poate dubla timpul de răspuns. Totuși, este posibilă paraleлизarea modului în care se gestionează cheia necesară pentru următorul pas și cheia folosită în decriptare.

Din momentul în care utilizatorul a introdus o cheie, se începe actualizarea acesteia, indiferent dacă se dorește sau nu începerea unei decriptări. În același moment în care începe o decriptare, începe și actualizarea în paralel a următoarei chei. Prin această optimizare, este posibilă executarea operațiunilor de decriptare fără întârzieri provocate de actualizarea cheii. Singurul caz în care nu se poate evita întârzierea este cazul în care un text este introdus imediat după ce a fost introdusă o cheie. Numărul maxim de perioade de ceas de întârziere este de 32 perioade, întrucât decriptarea pornește fără alte intervenții ale utilizatorului.

Astfel, deși complexitatea gestionării cheilor crește, această abordare permite optimizarea performanței sistemului. Paraleлизarea proceselor de actualizare și decriptare asigură fluența operațiunile desfășurate, reducând timpul de așteptare și menținând eficiența întregului circuit. Această metodă combină eficiența spațială cu îmbunătățiri semnificative în ceea ce privește timpul de răspuns, oferind un echilibru între complexitate și performanță.

În cadrul operațiilor de actualizare a cheii, cât și a rețelei SP, este necesară prezența blocurilor de substituție. Acestea sunt reprezentate în hardware ca blocuri de memorie ce sunt inițializate prin citirea unui fișier cu extensia ".mem". Pentru modulul de decriptare este necesară prezența unui bloc de substituție pentru avansarea cheii, cât și de un bloc invers pentru operația criptografică. În ambele cazuri, ele sunt reprezentate ca blocuri de memorie.

Complementar căii de date descrise anterior, se află calea de control care coordonează datele introduse de utilizator și semnalele care controlează elementele de execuție. Aceasta include mecanisme de resetare a modulului, precum și generarea de semnale de control pentru inițierea și monitorizarea pașilor de criptare și decriptare.

### 3.2. Împachetarea cu strat AXI

Pentru interfațarea cu restul modulelor din sistem, a fost adăugat un strat AXI (AXI wrapper) care conține 8 registre interne ce pot fi citite sau scrise de orice modul ce joacă rolul unui moderator AXI. Deoarece procesorul MicroBlaze poate accesa numai 32 de biți într-o singură operație, registrele interne ai stratului AXI vor avea dimensiunea de 32 biți. Pentru scrierea unei chei este nevoie de 3 operații de scriere, întrucât cheia ocupă 80 de biți, utilizând doar jumătate din ultimul registru scris. Același principiu se aplică și pentru operațiile de citire. Dacă o cheie este stocată în trei registre (K1, K2, K3), citirea acesteia va necesita tot 3 operații de citire. Primele două registre vor fi citite integral, iar ultimul registru va fi citit parțial.

Numărul de registre se poate optimiza prin îmbinarea diverselor părți din tranzacții pe același registru, dar pe jumătăți diferite ale acestuia. Această abordare permite utilizarea eficientă a spațiului disponibil în registre și reduce necesitatea de a adăuga mai multe registre pentru stocarea cheilor sau a altor date.

De exemplu, dacă o cheie de 80 de biți este divizată în trei părți (K1, K2, K3), primele două părți (K1 și K2) pot fi stocate în registre separate, fiecare ocupând un registru întreg de 32 de biți. Ultima parte (K3), care ocupă doar jumătate dintr-un registru, poate fi combinată cu alte date sau poate rămâne în registrul său, dar cu posibilitatea de a fi combinată cu date din alte tranzacții viitoare.

Modulele de criptare și decriptare utilizează aceeași structură de adresare a regisrelor furnizate de stratul AXI pentru a asigura consistență și eficiență. Registrul de la prima adresă este critic, deoarece conține semnalele de control și de stare esențiale pentru funcționarea modulului. Specific, bitul 0 este utilizat pentru resetarea modulului, în timp ce biții 1 și 2 servesc drept semnale de validare a datelor de intrare. Bitul 2 validează cheia, iar bitul 1 validează textul și inițiază logica de procesare. Funcția bitului 3 este de a dezactiva semnalul de întrerupere, iar bitul 4 indică finalizarea procesării. Este de remarcat faptul că toate aceste semnale sunt active atunci când sunt setate la valoarea logică 1, oferind o metodă simplă pentru a controla și monitoriza starea modulului.

Offset Adresă	Biți	Descriere
0h	0	Resetează modulul
	1	Validează textul și inițiază logica de procesare
	2	Validează cheia
	3	Dezactivează semnalul de întrerupere
	4	Indică finalizarea procesării
	31:5	Rezervat
1h	31:0	Registru cheii pentru depanare [31:0]
2h	31:0	Registru cheii pentru depanare [63:32]
3h	15:0 (L)	Registru cheii pentru depanare [80:64]
	31:16 (H)	Registru date de intrare [80:64]
4h	31:0	Registru date de intrare [63:32]
5h	31:0	Registru date de intrare [31:0]
6h	31:0	Registru date de ieșire [63:32]
7h	31:0	Registru date de ieșire [31:0]

Tabelul 3.1. Structura regisrelor pentru modulele de criptare și decriptare.

Rezultatele operațiunilor criptografice sunt stocate în ultimele două registre. Deși aceste registre pot fi citite în orice moment, ele conțin informații relevante doar odată ce semnalul de întrerupere este detectat sau când semnalul de finalizare este activ. Aceasta asigură că datele citite sunt rezultatul final al procesului criptografic. Registrele situate la adresele 3, 4 și 5 sunt destinate datelor de intrare. Datorită naturii secvențiale a citirii cheii și textului, aceleași regisre pot fi reu-

tilizate, cu doar două dintre cele trei registre alocate fiind utilizate pentru citirea textului. Această utilizare eficientă a spațiului de registre ajută la gestionarea datelor de intrare fără a necesita resurse suplimentare.

În scopuri de depanare, anumite registre (1 și 2, și jumătatea superioară a registrului 3) au fost desemnate pentru scrierea cheii din modul. Această cheie poate fi apoi citită de procesor și afișată pe un display cu șapte segmente. Această caracteristică este deosebit de utilă pentru verificarea corectitudinii cheii utilizate în timpul criptării sau decriptării și pentru diagnosticarea oricăror probleme care pot apărea în timpul procesării. Prin confirmarea vizuală a cheii se poate asigura utilizarea cheii corecte sau depăra mai ușor orice discrepanțe sau erori care apar în timpul operațiunilor criptografice. Această structură cuprinzătoare de utilizare a regisrelor susține astfel atât eficiența operațională, cât și capacitatele eficiente de depanare, făcând-o o parte integrantă a modulelor de criptare și decriptare.

### 3.3. Mediu de verificare în UVM

Universal Verification Methodology (UVM) este un cadru standardizat și extensibil dezvoltat pentru a aborda complexitatea crescândă a verificării designurilor digitale. Acesta oferă o suita de clase și funcții de bază care facilitează crearea, gestionarea și reutilizarea componentelor de verificare. Mediu de verificare UVM este esențial pentru asigurarea corectitudinii și performanței designurilor hardware, permitând dezvoltatorilor să identifice și să rezolve erori într-un mod sistematic și eficient. De asemenea sunt disponibile diverse materiale ce ajuta la înțelegerea funcționalității mediului și a metodelor de a-l crea. Una dintre cărțile studiate în acest sens este [9].

Mediul de verificare în UVM este structurat pentru a maximiza modularitatea și reutilizarea componentelor. Acesta include elemente precum agenți, monitoare, stimuli, comparatoare și raportori, care interacționează pentru a simula și verifica funcționarea designului. Fiecare componentă are un rol bine definit și colaborează în cadrul unei arhitecturi ierarhice pentru a efectua verificări complexe.

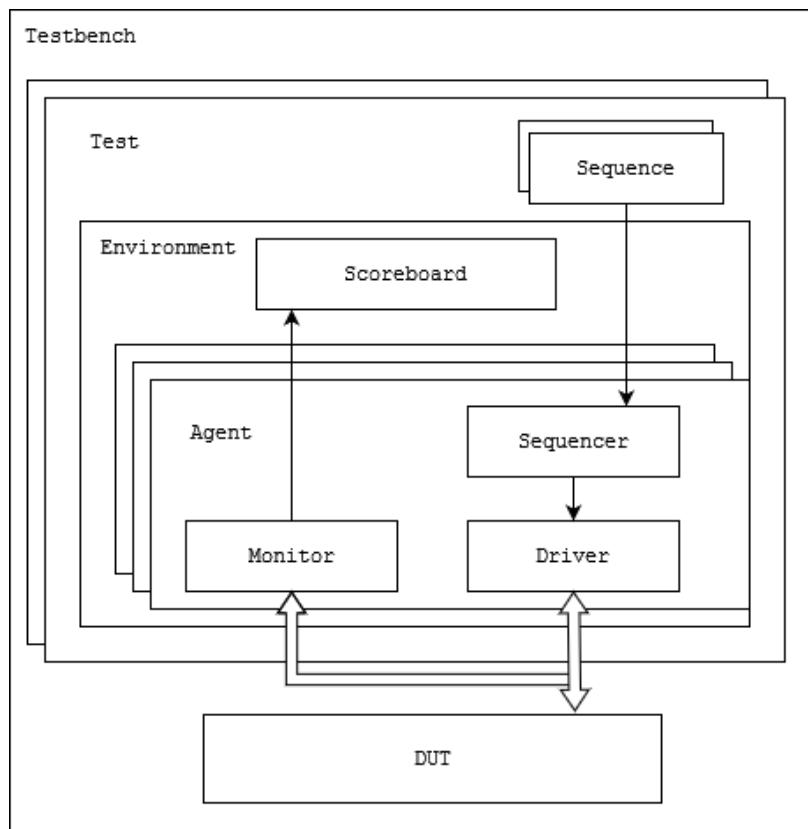


Figura 3.1. Structura mediului de verificare în UVM.

În cadrul acestei lucrări, a fost utilizat un singur mediu de verificare bazat pe biblioteca uvm distribuită de Accellera [10] pentru modulele de criptare și decriptare, ceea ce a permis o optimizare semnificativă a resurselor și a timpului de dezvoltare. Singura diferență între module a constat în componenta de comparare a rezultatelor, cunoscută sub numele de Scoreboard, care a fost specifică fiecărui modul în parte. Această abordare a asigurat un grad înalt de reutilizare a codului și a redus complexitatea procesului de verificare, permitând testarea eficientă a ambelor module în același mediu.

Procesul de dezvoltare și verificare a fost sesizabil mai scurt datorită acestei abordări eficiente. Utilizarea unui mediu comun de verificare a eliminat necesitatea creării și întreținerii mai multor medii distincte pentru fiecare modul, economisind astfel timp și resurse. În plus, prin specificarea componentelor Scoreboard pentru fiecare modul, s-a asigurat că rezultatele verificării sunt corecte și relevante pentru funcționalitatea specifică a fiecărui modul, îmbunătățind în același timp calitatea testelor.

Semnalele de intrare și ieșire au fost împărțite în trei interfețe distincte: control, comandă și output. Această structurare a impus necesitatea creării a trei agenți specifici pentru a gestiona fiecare dintre aceste interfețe. Agenții au fost proiectați să includă componente esențiale pentru funcționarea și verificarea corectă a semnalelor aferente fiecărei interfețe.

Agentul de control și agentul de comandă sunt ambele compuse din trei componente principale: monitor, secvențiator și driver. Monitorul are rolul de a supraveghea și captura activitatea semnalelor de pe interfață respectivă, asigurându-se că toate evenimentele sunt înregistrate corect pentru analiză ulterioară. Secvențiatorul este responsabil pentru generarea secvențelor de stimuli care vor fi trimise către driver. Aceste stimuli sunt proiectați pentru a testa diferitele condiții și scenarii de operare ale modulului. Driverul, la rândul său, primește aceste stimuli de la secvențiator și le transformă în semnale de intrare corespunzătoare, care sunt apoi trimise către modulul verificat.

În contrast, agentul de output este mai simplu și include doar un monitor. Aceasta se datorează faptului că interfața de output nu conține semnale de intrare și, prin urmare, nu necesită un driver sau un secvențiator. Rolul monitorului în acest context este de a supraveghea și înregistra toate semnalele de ieșire generate de modulul verificat. Astfel, se asigură că rezultatele obținute sunt conforme cu așteptările și specificațiile definite.

Această structurare în agenți și componente specifice a permis o gestionare mai eficientă a verificării semnalelor și a redus complexitatea generală a mediului de verificare. Fiecare agent, prin intermediul componentelor sale, își îndeplinește rolul specific în cadrul procesului de verificare, contribuind la un flux de lucru bine organizat și la rezultate de verificare precise și de încredere.

Agenții au fost încorporați într-un mediu (environment) alături de componenta Scoreboard, ceea ce a permis o integrare eficientă și o comunicare clară între toate componentele sistemului. În cadrul acestui mediu pot comunica prin intermediul unor porturi specifice componentele asigurând astfel un flux de date coerent și sincronizat.

Monitorul, driverul și secvențiatorul din fiecare agent comunică direct între ei pentru a asigura transmiterea și recepționarea corectă a semnalelor de control și comandă. Monitorul capturează semnalele de pe interfață, secvențiatorul generează stimuli de test, iar driverul interpretează și trimită aceste stimuli către modulul sub verificare. Această comunicare internă în cadrul fiecărui agent permite o gestionare eficientă a procesului de verificare pentru fiecare interfață în parte.

În ceea ce privește comunicarea între agenți și Scoreboard, aceasta se realizează prin intermediul environment-ului. Această structură permite ca datele capturate de monitoarele din agenți să fie transmise către Scoreboard pentru comparație și evaluare. Scoreboard-ul analizează aceste date pentru a verifica corectitudinea funcționării modulelor de criptare și decriptare, asigurând astfel validarea corectă a rezultatelor.

### 3.4. Programul de test

Pentru a testa funcționalitatea întregului ansamblu, nu doar a componentelor de criptare și decriptare, a fost creat un program în limbajul C care rulează pe procesorul MicroBlaze. Acest program a fost conceput pentru a verifica funcționalitatea fiecărei componente, de la modulul UART și cele de criptare și decriptare, până la modulele de intrare și ieșire specifice plăcii Nexys A7.

Programul de test a fost structurat astfel încât să inițializeze și să configureze toate componentele hardware implicate în sistem. În primul rând, programul stabilește conexiunea UART, asigurând astfel o comunicare stabilă și fiabilă pentru transmisia datelor între procesor și restul sistemului. Apoi, sunt verificate componentele de criptare și decriptare, programul generând și trimițând date specifice pentru a testa funcționalitatea corectă a acestor module.

Pe lângă componentele de criptare și decriptare, programul testează și modulele de intrare și ieșire specifice plăcii Nexys A7. Aceste module includ diverse periferice și interfețe care sunt esențiale pentru funcționarea corectă a sistemului. Programul C trimit și primește date prin aceste interfețe, verificând astfel integritatea și performanța acestora.

Programul funcționează într-o buclă infinită după inițializarea componentelor, asigurând astfel o operare continuă și receptivă. În cadrul acestei bucle, programul așteaptă să primească o comandă prin interfața UART. În funcție de comanda primită, programul poate realiza una dintre următoarele acțiuni:

1. Stabilirea unei chei comune secrete folosind algoritmul Diffie-Hellman:

La primirea unei comenzi specifice pentru inițierea procesului de stabilire a cheii comune, programul execută algoritmul Diffie-Hellman. Acest algoritm permite celor două părți să genereze o cheie secretă comună utilizând schimbul de chei publice, fără ca acestea să fie compromise.

2. Transmiterea mesajului înapoi nemodificat:

În cazul în care comanda primită indică doar retransmiterea mesajului, programul citește mesajul din buffer-ul UART și îl trimit înapoi exact cum a fost primit, asigurând astfel o funcționalitate de tip "echo".

3. Criptarea mesajului și transmiterea acestuia:

La primirea unei comenzi de criptare, programul preia mesajul din buffer-ul UART, aplică algoritmul de criptare și trimită mesajul criptat înapoi. Acest lucru asigură securitatea datelor în tranzit, făcându-le ilegibile pentru terți neautorizați.

4. Decriptarea mesajului și transmiterea acestuia:

Pentru comanda de decriptare, programul citește mesajul criptat din buffer-ul UART, aplică algoritmul de decriptare pentru a-l transforma în forma originală, și trimită mesajul decriptat înapoi. Aceasta permite destinatarului să recupereze informațiile originale dintr-un mesaj criptat.

În cazul unei comenzi invalide sau a lipsei unei comenzi, programul apelează funcția `get_key_on_segments()`. Această funcție are rolul de a citi cheia din interiorul modulului de criptare sau decriptare, în funcție de un switch, și de a o afișa pe display-ul cu 7 segmente

```

1  while(1)
2  {
3      while(!XUartLite_Recv(&uart0, &command, 1)) get_key_on_segments();
4
5      switch (command) {
6          case 'K': key_agreement(); break;

```

```

7     case 'N': echo(); break;
8     case 'E': enc_echo(); break;
9     case 'D': dec_echo(); break;
10    default : get_key_on_segments(); break;
11 }
12 }
```

Listing 3.1. Bucla principală de control

Funcția `key_agreement()` este responsabilă pentru stabilirea unei chei comune secrete între două părți folosind algoritmul Diffie-Hellman. Generatorul ( $g$ ) și modulul prim ( $p$ ) sunt predefinite, asigurând parametrii necesari calculului cheii comune. Datorită dimensiunii cheii de 80 de biți și a limitării procesorului la 32 de biți, este necesară utilizarea unor metode speciale pentru reprezentarea numerelor și implementarea unor funcții de calcul adecvate.

```

1 void key_agreement() {
2     uint8_t size = word_80 * 2;
3     uint32_t a[size];
4     set_rand(a, size);
5     uint32_t g[size];
6     uint32_t p[size];
7     set_zero(g, size);
8     set_zero(p, size);
9     g[0] = 2;
10    p[2] = 0x10000;
11    p[0] = 0xd;
12
13    pow_mod(g, a, p, size);
14    send_hex(g, "A", size/2);
15
16    receive_B(g, size/2);
17    pow_mod(g, a, p, size);
18
19    set_key(g);
20 }
```

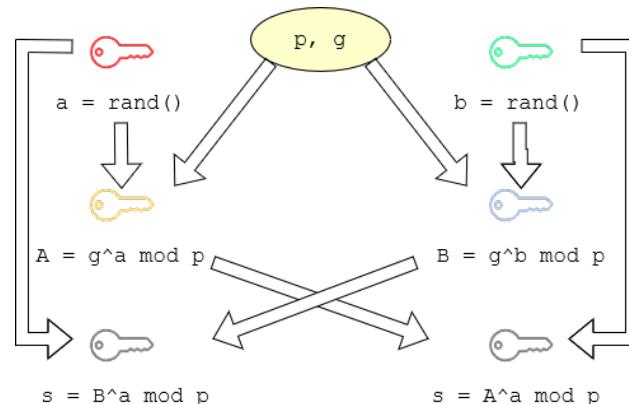


Figura 3.2. Algoritmul Diffie-Hellman.

Pentru a reprezenta numerele de 80 de biți pe un procesor de 32 de biți, se utilizează un vector de trei elemente, fiecare de 32 de biți. Această structură permite manipularea numerelor mari prin împărțirea acestora în segmente ușor de gestionat. Astfel, indexul 0 al vectorului conține cei mai puțin semnificativi 32 de biți, indexul 1 conține următorii 32 de biți, iar indexul 2 conține cei mai semnificativi 16 biți (restul celor 32 de biți rămân nefolosiți).

### 3.4.1. Operații cu numere mari

Operația de înmulțire se poate implementa în mai multe moduri, dar nu toate sunt optime. O primă opțiune poate fi folosirea de adunări repetitive, însă această metodă este limitată de valoarea numerelor înmulțite. Timpul de procesare este direct proporțional cu numărul de repetări (valoarea unuia dintre operanzi), ceea ce poate duce la o eficiență scăzută pentru numere mari. De exemplu, pentru a înmulți 15 cu 7, ar fi necesare 15 adunări ale valorii 7, ceea ce nu este practic pentru numere de mari dimensiuni.

O a doua opțiune constă în folosirea operației limitate de înmulțire oferită de procesor, aplicată pe fiecare parte și combinarea rezultatelor intermediare. Totuși, înmulțirea a două numere pe 32 de biți rezultă într-un număr pe 64 de biți, iar combinarea rezultatelor devine dificilă. Procesorul poate avea capacitați limitate de a gestiona direct aceste operații pe 64 de biți, necesitând astfel algoritmi suplimentari pentru gestionarea corectă a rezultatelor intermediare și pentru a asigura acuratețea operației finale.

Cea mai simplă soluție constă în folosirea unor operații de deplasare combinate cu adunare. În acest fel, se lucrează la nivel de bit, similar cu modul în care ar proceda un element de înmulțire hardware. Deplasarea unui bit la stânga este echivalentă cu înmulțirea numărului respectiv cu 2, iar adunarea permite combinarea parțială a rezultatelor pentru a obține produsul final. Această metodă este mai eficientă și mai rapidă, reducând complexitatea calculului și fiind discutată pe larg în lucrări de specialitate[11].

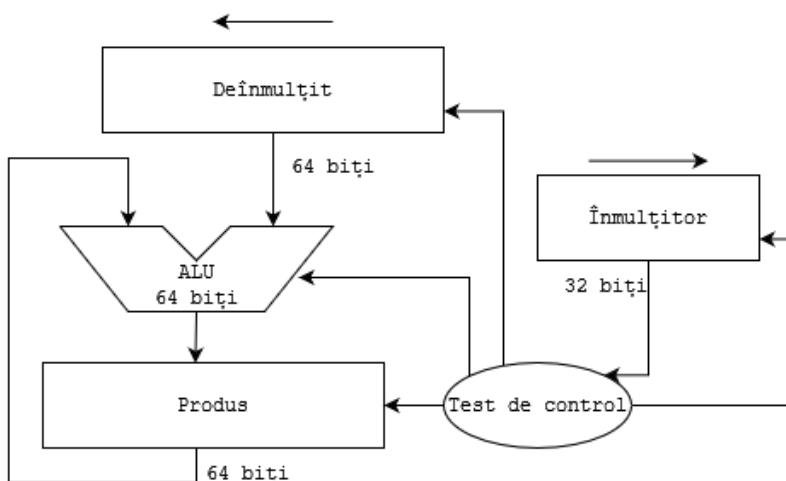


Figura 3.3. Element de înmulțire hardware.

Ridicarea la putere funcționează pe un principiu similar de operare cu biți, doar că în loc de a shifta primul operand la stânga (echivalentul înmulțirii cu 2), se efectuează operația de înmulțire cu el însuși (echivalentul ridicării la puterea a două). În acest proces, în locul adunării se efectuează înmulțirea între registri. Acest principiu de operare este esențial în multe aplicații criptografice și de calcul de înaltă performanță.

Conform algoritmului Diffie-Hellman, operația de ridicare la putere se realizează în modul, ceea ce înseamnă că după fiecare operație din cadrul ridicării la putere, rezultatele se normalizează folosind modulul. Aceasta implică faptul că, după fiecare înmulțire intermediară, rezultatul este redus prin împărțirea la numărul prim (modulul) și păstrarea restului. Acest proces de normalizare asigură că rezultatele rămân gestionabile ca dimensiune și previne depășirea limitei de biți a procesorului.

Cât despre împărțirea în modul, mai întâi se deplasează divizorul în jumătatea mai semnificativă a registrului. După se efectuează o serie de deplasări la dreapta, în care se testează dacă divizorul este mai mare decât împărțitorul caz în care se operează scădere. Acest proces de deplasare la dreapta și testare continuă până când toate pozițiile relevante ale registrului au fost procesate. Rezultatul final al acestui proces este restul împărțirii, care reprezintă partea rămasă din dividend după ce au fost extrase toate portiunile posibile de divizor.



## Capitolul 4. Testarea aplicației și rezultate experimentale

Platforma creată pentru efectuarea operațiilor de criptografie este testată atât fizic prin intermediul unei interfețe grafice, cât și prin simulare utilizând mediul de verificare UVM.

### 4.1. Testarea prin interfața grafică

Dezvoltarea aplicației grafice a fost simplificată semnificativ datorită funcționalităților oferite de biblioteca `cppClrWinForm`, care permite crearea unei logici de control bazate pe evenimentele generate de interacțiunea cu componentele grafice. Prin intermediul butoanelor de pe interfață, utilizatorii pot trimite comenzi către sistemul criptografic, configurându-l și controlându-l. Printre comenziile disponibile se numără trimiterea unui mesaj pentru a fi procesat de FPGA prin criptare sau decriptare, inițierea algoritmului de stabilire a unei chei comune secrete prin algoritmul Diffie-Hellman, sau simpla funcționalitate de echo, în care sistemul trimită înapoi orice mesaj primit pentru a testa canalul de comunicare stabilit prin UART.

Interfața grafică permite criptarea sau decriptarea mesajelor primite prin executarea algoritmului prezent în formă software. De asemenea, utilizatorii pot configura portul și viteza de transfer pentru a stabili conexiunea dorită. Mesajele primite pot fi afișate în format text sau hexazecimal, iar interfața oferă posibilitatea afișării cheii actuale calculate, pentru a putea fi comparată cu cheia afișată pe placă prin intermediul unui afișaj cu șapte segmente.

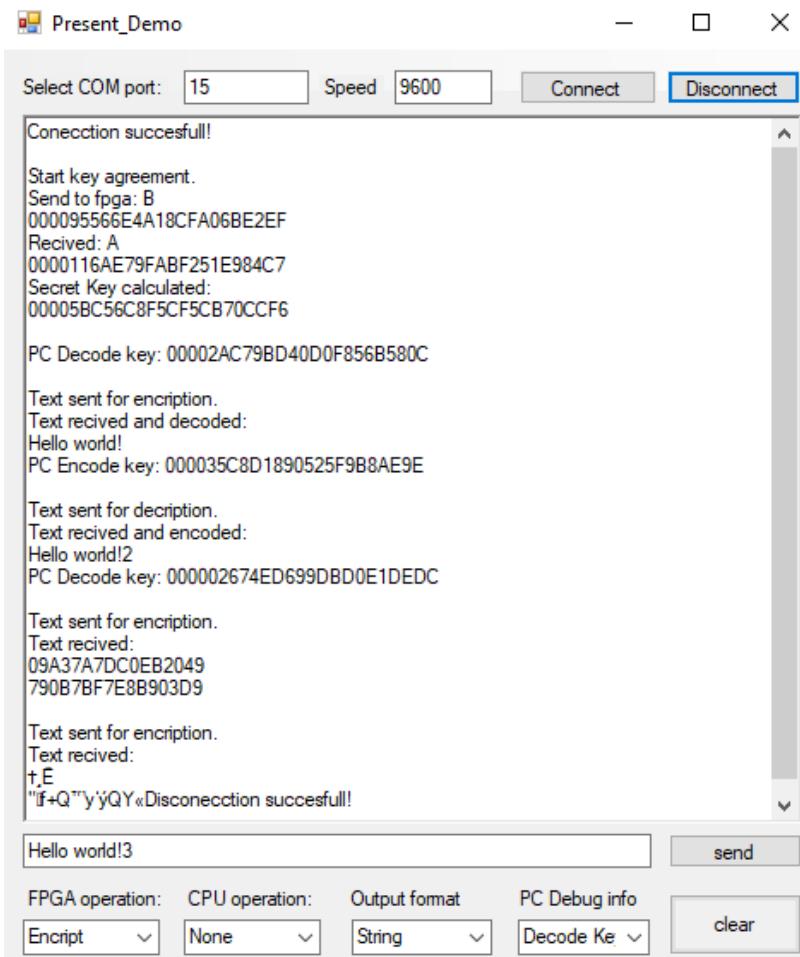


Figura 4.1. Interfața grafică. Legătură cu A.1 A.2 A.3 A.4

Atât aplicația software care rulează pe computer, cât și cea implementată pe FPGA folo-

sesc aceleasi functii pentru a efectua operații complexe cu numere mari. Acest lucru subliniază versatilitatea și consistența designului acestor funcții, permitându-le să fie utilizate eficient în ambele medii. Compatibilitatea aceasta asigură că algoritmii criptografici pot fi implementați și testați uniform pe ambele platforme, facilitând dezvoltarea și verificarea sistemului criptografic.

Pentru gestionarea afișării datelor în cazul programului grafic, funcția standard *xil\_printf* este suprascrisă astfel încât datele să fie stocate într-un buffer local înainte de a fi prelucrate și afișate pe consolă. Această suprascriere permite colectarea și manipularea datelor relevante într-o manieră organizată și eficientă, pregătindu-le pentru afișare ulterioară.

Pentru a valida comunicarea cu exteriorul și funcționalitatea generală a sistemului de criptare și decriptare, a fost creat un program de testare care trimitе comenzi prin UART și, în paralel, primește răspunsurile. Acest program utilizează Windows API pentru gestionarea comunicației seriale și oferă o interfață grafică ușor de utilizat, creată cu ajutorul WinForms.

```
1 #define xil_printf(fmt, ...) recv_index +=  
→ sprintf(&recv_buff[recv_index], fmt, ##__VA_ARGS__)
```

Acest fragment de cod redifineste funcția *xil\_printf* pentru a adăuga datele într-un buffer, *recv\_buff*, actualizând în același timp indexul de receptie, *recv\_index*. Astfel, datele colectate sunt păstrate într-un spațiu de memorie temporar, permitând prelucrarea lor ulterioară înainte de a fi afișate pe consolă. Această metodă asigură o gestionare mai bună a datelor și îmbunătățește capacitatea de a monitoriza și depana sistemul criptografic.

Interfața grafică este proiectată pentru a se putea redimensiona, oferind astfel mai mult spațiu pentru afișarea mesajelor și suportă navigarea prin derulare, permitând utilizatorilor să parcurgă mesajele afișate pe consolă cu ușurință. Aceasta este o caracteristică utilă pentru revizuirea mesajelor anterioare sau pentru monitorizarea fluxului continuu de date, oferind un control sporit asupra informațiilor vizualizate.

Pentru conectarea la sistemul criptografic, se utilizează protocolul UART, care poate fi manipulat prin intermediul Windows API. Setările legate de paritate și numărul de biți de date sunt prestable, simplificând astfel procesul de configurare și utilizare.

#### 4.2. Rezultate experimentale

În ceea ce privește simularea sistemului, cea mai practică abordare constă în crearea unui mediu de verificare datorită similitudinilor dintre modulul de criptare și cel de decriptare. Utilizarea bibliotecii UVM (Universal Verification Methodology) oferă un cadru robust pentru gestionarea mecanismelor de transfer a pachetelor între componente și pentru organizarea testelor și secvențelor necesare. Avantajele utilizării UVM

Crearea unui mediu de verificare în UVM permite exploatarea similitudinilor dintre criptare și decriptare, facilitând reutilizarea componentelor de verificare și reducând timpul și efortul necesare pentru dezvoltarea testelor. Biblioteca UVM asigură o structură modulară și scalabilă, care poate fi ușor extinsă și adaptată pentru diferite scenarii de testare. Gestionarea transferului de pachete

Unul dintre principalele avantaje ale utilizării UVM este capacitatea sa de a gestiona eficient mecanismele de transfer a pachetelor între diferitele componente ale sistemului. Acest lucru este esențial pentru simularea precisă și fiabilă a operațiunilor criptografice, unde pachetele de date trebuie transferate corect între modulele de criptare și decriptare. UVM facilitează implementarea și monitorizarea acestor transferuri, asigurând integritatea și consistența datelor pe parcursul întregului proces de verificare. Organizarea testelor și secvențelor

Biblioteca UVM oferă instrumente puternice pentru organizarea și gestionarea testelor și secvențelor de verificare. Prin utilizarea secvențelor UVM, testele pot fi create într-o manieră structurată, asigurând acoperirea completă a funcționalităților sistemului. Acest cadru permite definirea clară a stimulilor și a așteptărilor pentru fiecare test, facilitând identificarea și diagnosticarea

problemelor potențiale.

```

1 virtual function void write_cmd(cmd_packet trans);
2   if(counter == 0) begin
3     case(trans.data_type)
4       KEY: expected_key = trans.data;
5       TEXT: begin
6         expected_text = trans.data;
7         counter = 1;
8         start_timer(current_time + 38);
9         $uvm_info(get_type_name(), $sformatf("start timer=%0d",
10           → get_time()), UVM_LOW);
11      end
12    endcase
13  end
14 endfunction : write_cmd

```

Figura 4.2. Write\_cmd pentru verificarea criptării

```

1 virtual function void write_cmd(cmd_packet trans);
2   case(trans.data_type)
3     KEY: begin
4       stop_timer();
5       expected_key = next_expected_key;
6       next_expected_key = trans.data;
7       key_time = get_time();
8       key_counter = get_time() + 31;
9       counter = get_time();
10    end
11    TEXT: begin
12      if(get_time() <= key_counter) begin //done key update
13        expected_text = trans.data;
14        counter = key_counter + 32;
15        start_timer(key_counter + 38);
16        $uvm_info(get_type_name(), $sformatf("was start timer=%0d",
17          → get_time()), UVM_LOW);
18      end else
19        if(get_time() > counter) begin
20          expected_text = trans.data;
21          counter = get_time() + 32;
22          start_timer(get_time() + 38);
23          $uvm_info(get_type_name(), $sformatf("start timer=%0d,
24            → counter=%0d", get_time(), counter), UVM_LOW);
25        end else
26          $uvm_info(get_type_name(), $sformatf("ignore packet time=%0d,
27            → counter=%0d", get_time(), counter), UVM_LOW);
28    end
29  endcase
30 endfunction : write_cmd

```

Figura 4.3. Write\_cmd pentru verificarea decriptării

Pentru a adapta mediul de verificare de la un modul la altul, este necesară doar modificarea Design Under Test (DUT) și a scoreboard-ului, deoarece interfetele, monitoarele și generatoarele sunt identice. Diferențele apar doar în modul în care implementările porturilor de analiză gestioneză și verifică datele rezultate față de cele așteptate. Procesul de testare a implicat utilizarea

```

UVM_INFO test_lib.sv(46) @ 46720: uvm_test_top [simple_test] ** UVM TEST PASSED **
UVM_INFO /apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_report_server.svh(904) @ 46720: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO : 584
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
** Report counts by id
[RNTST] 1
[UVM/RELNOTES] 1
[scoreboard] 274
[simple_test] 2
[start_phase] 3
[uvm_test_top.env0.cmd_agt.monitor] 212
[uvm_test_top.env0.ctrl_agt.monitor] 2
[uvm_test_top.env0.out_agt.monitor] 89

$finish called from file "/apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_root.svh", line 527.
$finish at simulation time 46720
      V C S   S i m u l a t i o n   R e p o r t
Time: 46720 ns
CPU Time: 1.080 seconds;      Data structure size: 0.4Mb

```

Figura 4.4. Raport de simulare pentru criptare.

unor teste randomizate pe o perioadă îndelungată, acoperind o gamă largă de scenarii și parametru. Acestă abordare a permis identificarea unor neconformități atât în implementarea sistemului criptografic, cât și în mediul de verificare. Problemele descoperite au variat de la erori de logică în algoritmi până la disfuncționalități în mecanismele de transfer de date și comparare. Odată identificate, aceste neconformități au fost analizate și rezolvate sistematic.

```

UVM_INFO test_lib.sv(46) @ 46720: uvm_test_top [simple_test] ** UVM TEST PASSED **
UVM_INFO /apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_report_server.svh(904) @ 46720: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---

** Report counts by severity
UVM_INFO : 556
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
** Report counts by id
[RNTST] 1
[UVM/RELNOTES] 1
[scoreboard] 274
[simple_test] 2
[start_phase] 3
[uvm_test_top.env0.cmd_agt.monitor] 212
[uvm_test_top.env0.ctrl_agt.monitor] 2
[uvm_test_top.env0.out_agt.monitor] 61

$finish called from file "/apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_root.svh", line 527.
$finish at simulation time 46720
      V C S   S i m u l a t i o n   R e p o r t
Time: 46720 ns
CPU Time: 1.040 seconds;      Data structure size: 0.4Mb

```

Figura 4.5. Raport de simulare pentru decriptare.

Utilizarea testelor randomizate a fost esențială pentru identificarea problemelor subtile care ar fi putut trece neobservate în cazul unor teste deterministe. Testarea randomizată a permis acoperirea unui spectru mai larg de condiții și a contribuit la asigurarea unei robuste și sporite a sistemului.

Această metodă de testare a fost crucială pentru garantarea faptului că sistemul criptografic poate face față unei varietăți de situații neprevăzute.

Prin aceste teste, s-a confirmat că implementarea hardware și software a sistemului de criptare și decriptare este robustă și eficientă, demonstrând capacitatea de a oferi securitate în aplicațiile embedded.

Din tabelul 4.1 se observă eficiența în resurse utilizate și energie consumată, eficiență dobândită în urma alegерilor arhitecturale și a comparației dintre opțiunile de algoritmi de criptare disponibili.

Tabelul 4.1. Raport de utilizare a resurselor și energiei.

Resursa din fpga	Utilizare (%)
LUT	4
LUTRAM	1
FF	2
BRAM	24
IO	27
BUFG	13
MMCM	17
Consum de energie	
Total On-Chip Power	0.26 W
Junction Temperature	26.2°C
Thermal Margin	58.8°C (12.8 W)
Power supplied to off-chip devices	0 W
Confidence level	Low

### 4.3. Elemente de configurare și instalare

În vederea recreării proiectului sau modificării acestuia sunt necesari diferiți pași, în funcție de tipul de modificări dorite. Fie că se dorește modificarea arhitecturii hardware sau ajustarea componentelor software, fiecare modificare necesită un set specific de instrumente și proceduri.

Dacă se dorește modificarea arhitecturii hardware, este necesar programul Vivado, care oferă o interfață grafică pentru manipularea modulelor sub forma unor diagrame. Vivado permite utilizatorului să adauge, să elimine sau să modifice modulele hardware conform noii arhitecturi dorite. După efectuarea modificărilor, noua arhitectură trebuie sintetizată și implementată, ceea ce implică generarea unui fișier bitstream care conține datele de configurare pentru FPGA.

După generarea fișierului bitstream, hardware-ul trebuie exportat incluzând acest bitstream. Acest lucru rezultă într-un fișier .xsa (Xilinx Shell Archive), care conține toate informațiile necesare despre configurația hardware-ului. Fișierul .xsa poate fi utilizat ulterior pentru a programa FPGA-ul și pentru a integra hardware-ul cu componenta software dezvoltată în Vitis.

Recrearea arhitecturii dezvoltate în această lucrare poate fi realizată printr-un script TCL, cu condiția să fie disponibile IP-urile necesare în sistem. Acestea pot fi recreate printr-un AXI wrapper pentru modulul de criptare și unul pentru modulul de decriptare. Registrele AXI trebuie conectate la porturile modulului aferent.

Modificarea programului care rulează pe MicroBlaze necesită crearea unei platforme în Vitis folosind fișierul .xsa. În Vitis, se creează o nouă aplicație pentru platforma generată, implementând codul dorit. După implementarea și compilarea codului, se programează placa FPGA și se generează un nou fișier bitstream care include atât noua configurație hardware, cât și aplicația software actualizată. Acest fișier bitstream poate fi ulterior scris în memoria flash a placii Nexys A7-100T, asigurând stocarea nevolatilă a configurației și a aplicației software. Acești pași pot fi

```

1  if (slv_reg_wren)
2  begin
3  ...
4  end
5  else
6  begin
7      slv_reg1    <= pe.key[C_S_AXI_DATA_WIDTH - 1 :0];
8      slv_reg2    <= pe.key[C_S_AXI_DATA_WIDTH * 2 -
9          ↳ 1:C_S_AXI_DATA_WIDTH];
9      slv_reg0[C_S_AXI_DATA_WIDTH - 1 : C_S_AXI_DATA_WIDTH/2] <= pe.key[
10         ↳ C_S_AXI_DATA_WIDTH * 5 / 2 - 1 : C_S_AXI_DATA_WIDTH * 2];
10     slv_reg7    <= out_text[C_S_AXI_DATA_WIDTH * 2 -
11         ↳ 1:C_S_AXI_DATA_WIDTH];
11     slv_reg6    <= out_text[C_S_AXI_DATA_WIDTH - 1 :0];
12     slv_reg0[4] <= done;
13 end
14
15 assign interrupt = slv_reg0[3] ? 1'b1 : done;
16 assign reset     = ~slv_reg0[0] & S_AXI_ARESETN;
17 assign pl      = slv_reg0[2:1];
18 assign in_text   = {slv_reg5, slv_reg4, slv_reg3};
19
20 // Add user logic here
21 present_encoder pe(S_AXI_ACLK, reset, pl, in_text, out_text, done);
22 // User logic ends

```

Figura 4.6. Conectare registre AXI la porturile modulului.

studiați mai detaliat în documentația oferită de Xilinx[12].

Interfața grafică creată în WinForms oferă un mediu ușor de utilizat și de adaptat datorită multitudinii de componente grafice prestabilite. Aceste componente, cum ar fi butoanele, casetele de text, etichetele și altele, pot fi rapid integrate și configurate în cadrul interfeței, oferind utilizatorilor un mod intuitiv de a interacționa cu sistemul sau prin folosirea anexei1.

Unul dintre principalele avantaje ale utilizării WinForms este flexibilitatea oferită prin funcțiile de callback asociate fiecărei componente grafice. Funcțiile de callback permit definirea unor comportamente specifice care sunt declanșate de acțiunile utilizatorilor, cum ar fi apăsarea unui buton sau modificarea textului într-o casetă. Acest mecanism face posibilă adaptarea rapidă a interfeței la nevoile specifice ale aplicației, fără a necesita modificări extensive ale codului.

De exemplu, pentru trimiterea de comenzi prin UART, butoanele din interfață pot avea asociate funcții de callback care inițializează și gestionează comunicarea serială. Astfel, când utilizatorul apasă un buton pentru a trimite un mesaj criptat, funcția de callback corespunzătoare va prelua textul din caseta de input, va efectua operațiunile necesare de criptare și va trimite datele către sistemul FPGA.

## Concluzii

Proiectul „Accelerator hardware de criptare și decriptare implementat pe FPGA” a demonstrat cu succes cum implementarea unui algoritm de criptare light-weight pe un circuit FPGA poate aduce multiple beneficii, inclusiv eficiență ridicată și consum redus de resurse. Alegerea algoritmului Present, cunoscut pentru eficiență să în medii cu resurse limitate, a fost justificată prin necesitatea de a asigura securitatea datelor în cadrul dispozitivelor IoT și a sistemelor embedded.

Un aspect esențial al acestui proiect a fost abordarea modulară a designului sistemului. Această metodă a permis o flexibilitate crescută în dezvoltare și testare, facilitând izolare și optimizarea fiecărui modul în parte. În locul unui design monolitic, care ar fi complicat semnificativ procesul de implementare și întreținere, soluția modulară a oferit o structură mai ușor de gestionat și extins. Acest lucru este deosebit de important în contextul sistemelor embedded, unde cerințele și specificațiile pot varia considerabil.

Testele și rezultatele experimentale au evidențiat performanțele superioare ale sistemului implementat pe FPGA în comparație cu soluțiile software traditionale. Algoritmul Present a fost capabil să proceseze datele rapid și eficient, reducând semnificativ timpul necesar pentru criptare și decriptare. Acest lucru s-a datorat în mare măsură utilizării arhitecturii FPGA, care permite execuția paralelă a operațiunilor și optimizează consumul de energie. Implementarea hardware a algoritmului Present pe FPGA a demonstrat că acest tip de soluție poate oferi un nivel înalt de securitate fără a compromite performanța sau eficiența energetică.

De asemenea, mediul de verificare UVM (Universal Verification Methodology) utilizat în acest proiect a jucat un rol crucial în asigurarea funcționalității corecte a sistemului. Prin intermediul simulărilor detaliate și testelor exhaustive, s-au identificat și corectat problemele potențiale înainte ca sistemul să fie implementat în medii reale de operare. Această metodologie de verificare a asigurat că sistemul respectă specificațiile și cerințele inițiale, minimizând riscurile de erori în funcționare și asigurând un nivel ridicat de fiabilitate.

Comparativ cu alte proiecte și soluții similare, acest proiect se remarcă prin integrarea eficiență a componentelor hardware și software și utilizarea unui algoritm de criptare bine cunoscut și testat, cum este Present. Alegerea FPGA-ului ca platformă de implementare a permis obținerea unui echilibru optim între securitate, performanță și consum de resurse. Pe lângă avantajele evidente de performanță și eficiență energetică, utilizarea FPGA-urilor oferă și o scalabilitate crescută, permitând adaptarea rapidă la evoluțiile tehnologice și la noile amenințări de securitate.

În viitor, direcțiile de dezvoltare ale acestui proiect pot include extinderea funcționalităților criptografice prin implementarea unor algoritmi suplimentari, optimizarea suplimentară a consumului de energie și explorarea altor tehnologii de criptare light-weight. De asemenea, adaptarea sistemului la noi tipuri de atacuri cibernetice și îmbunătățirea capacitaților de reacție și adaptare la amenințările emergente vor fi esențiale pentru menținerea relevanței și eficienței soluției propuse.

În concluzie, acest proiect oferă o contribuție semnificativă în domeniul criptografiei embedded, demonstrând aplicabilitatea și eficiența soluțiilor FPGA pentru asigurarea securității datelor în contextul IoT și al sistemelor cu resurse limitate. Implementarea algoritmului Present și utilizarea metodologiilor moderne de verificare subliniază potențialul acestor tehnologii în dezvoltarea de soluții criptografice eficiente, sigure și scalabile. Aceste rezultate confirmă că FPGA-urile reprezintă o platformă robustă și flexibilă pentru implementarea soluțiilor de securitate, având capacitatea de a răspunde cerințelor tot mai complexe ale lumii digitale interconectate.



## Bibliografie

- [1] A. Menezes, P. Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996, ISBN: 0849385237.
- [2] I. Kuon and J. Rose, “Measuring the gap between fpgas and asics,” *The Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto*, 2007, {ikuon,jayar}@eecg.utoronto.ca.
- [3] B. Groza, *Introducere în Criptografie: Funcții Criptografice, Fundamente Matematice și Computaționale*. Bucharest, Romania: MatrixRom, 2013.
- [4] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vinkelsoe, “PRESENT: An Ultra-Lightweight Block Cipher,” in *Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007)*. Berlin, Heidelberg: Springer, 2007, pp. 450–466. [Online]. Available: [https://doi.org/10.1007/978-3-540-74735-2\\_31](https://doi.org/10.1007/978-3-540-74735-2_31)
- [5] C. Beierle, J. Jean, M. Khairallah, S. Lucks, T. Peyrin, Y. Sasaki, Y. Todo, and L. Wang, “The skinny family of block ciphers and its low-latency variant mantis,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 9665, Springer. Springer, 2016. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-3-662-49890-3\\_5](https://link.springer.com/chapter/10.1007/978-3-662-49890-3_5)
- [6] K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita, “Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms — Design and Analysis,” in *Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT 2001)*. Berlin, Heidelberg: Springer, 2001, pp. 39–56. [Online]. Available: [https://doi.org/10.1007/3-540-45682-1\\_3](https://doi.org/10.1007/3-540-45682-1_3)
- [7] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, “Piccolo: An Ultra-Lightweight Blockcipher,” in *Proceedings of the 13th International Conference on Cryptographic Hardware and Embedded Systems (CHES 2011)*, ser. Lecture Notes in Computer Science, B. Preneel and T. Takagi, Eds., vol. 6917. Berlin, Heidelberg: Springer, 2011, pp. 342–357. [Online]. Available: [https://doi.org/10.1007/978-3-642-23951-7\\_23](https://doi.org/10.1007/978-3-642-23951-7_23)
- [8] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, “The simon and speck families of lightweight block ciphers,” National Security Agency, Tech. Rep. Cryptology ePrint Archive, Report 2013/404, 2013. [Online]. Available: <https://eprint.iacr.org/2013/404>
- [9] R. Salemi, *UVM Cookbook*. Verilog Pro, 2013. [Online]. Available: <https://www.verilogpro.com/uvm-cookbook/>
- [10] A. S. Initiative, *Universal Verification Methodology (UVM)*. Accellera Systems Initiative, 2013. [Online]. Available: <https://accellera.org/downloads/standards/uvm>
- [11] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. San Francisco, CA: Morgan Kaufmann, 2004.
- [12] Xilinx, *Embedded Design Tutorials: MicroBlaze System*, 2022, accessed: 2024-06-25. [Online]. Available: [https://xilinx.github.io/Embedded-Design-Tutorials/docs/2022.1/build/html/docs/Feature\\_Tutorials/microblaze-system/README.html](https://xilinx.github.io/Embedded-Design-Tutorials/docs/2022.1/build/html/docs/Feature_Tutorials/microblaze-system/README.html)



## Anexe

### Anexa 1. Componentele interfeței grafice

```

1 #pragma once
2
3 using namespace System::Runtime::InteropServices;
4
5 //Variabile globale
6 char cpu_op = 'K';
7 char fpga_op = 'K';
8 char out_op = 'S';
9
10 uint32_t decript_key[6];
11 uint32_t encrypt_key[6];
12
13 HANDLE port;
14 char com_port_buff[100];
15 uint32_t baud_rate = 9600;
16 int conected = 0;
17 char* recv_buff;
18 int recv_index = 0;
19
20 namespace CppCLRWinFormsProject {
21
22     using namespace System;
23     using namespace System::ComponentModel;
24     using namespace System::Collections;
25     using namespace System::Windows::Forms;
26     using namespace System::Data;
27     using namespace System::Drawing;
28
29     public ref class Form1 : public System::Windows::Forms::Form
30     {
31     public:
32         Form1(void)
33         {
34             InitializeComponent();
35             recv_index = 0;
36             //alocare memorie pentru buffer
37             recv_buff = new char[10000];
38
39             cpuComboBox->Items->Add("None");
40             cpuComboBox->Items->Add("DF key");
41             cpuComboBox->Items->Add("Encrypt");
42             cpuComboBox->Items->Add("Decrypt");
43
44             fpgaComboBox->Items->Add("None");
45             fpgaComboBox->Items->Add("DF key");
46             fpgaComboBox->Items->Add("Encrypt");
47             fpgaComboBox->Items->Add("Decrypt");
48

```

```

49         outputComboBox->Items->Add( "Hex" );
50         outputComboBox->Items->Add( "String" );
51
52         dbgComboBox->Items->Add( "Decode Key" );
53         dbgComboBox->Items->Add( "Encode Key" );
54     }
55
56 protected:
57     /// <summary>
58     /// Clean up any resources being used.
59     /// </summary>
60     ~Form1()
61     {
62         if (components)
63         {
64             delete[] recv_buff;
65             delete components;
66         }
67     }
68 //////////////////////////////////////////////////////////////////
69 //Crearea componentelor grafice
70 //////////////////////////////////////////////////////////////////
71 private: System::Windows::Forms::Button^ sendButton;
72 protected:
73
74 protected:
75 private: System::Windows::Forms::MenuStrip^ menuStrip1;
76 private: System::Windows::Forms::TextBox^ inputTextBox;
77 private: System::Windows::Forms::TextBox^ comTextBox;
78 private: System::Windows::Forms::Button^ connectButton;
79 private: System::Windows::Forms::Label^ label1;
80 private: System::Windows::Forms::Label^ label2;
81 private: System::Windows::Forms::TextBox^ speedTextBox;
82 private: System::Windows::Forms::ComboBox^ cpuComboBox;
83 private: System::Windows::Forms::Label^ label3;
84 private: System::Windows::Forms::Label^ label4;
85 private: System::Windows::Forms::ComboBox^ fpgaComboBox;
86 private: System::Windows::Forms::Button^ clearButton;
87 private: System::Windows::Forms::RichTextBox^ richTextBox1;
88 private: System::Windows::Forms::ComboBox^ outputComboBox;
89 private: System::Windows::Forms::Label^ label5;
90 private: System::Windows::Forms::Label^ label6;
91 private: System::Windows::Forms::ComboBox^ dbgComboBox;
92 private: System::Windows::Forms::Button^ disconnectButton;
93 private:
94     /// <summary>
95     /// Required designer variable.
96     /// </summary>
97     System::ComponentModel::Container^ components;
98
99 #pragma region Windows Form Designer generated code
100    /// <summary>
101    /// Required method for Designer support - do not modify

```

```

102     /// the contents of this method with the code editor.
103     /// </summary>
104     void InitializeComponent(void)
105     {
106         this->sendButton = (gcnew System::Windows::Forms::Button());
107         this->menuStrip1 = (gcnew System::Windows::Forms::MenuStrip());
108         this->inputTextBox = (gcnew System::Windows::Forms::TextBox());
109         this->comTextBox = (gcnew System::Windows::Forms::TextBox());
110         this->connectButton = (gcnew System::Windows::Forms::Button());
111         this->label1 = (gcnew System::Windows::Forms::Label());
112         this->label2 = (gcnew System::Windows::Forms::Label());
113         this->speedTextBox = (gcnew System::Windows::Forms::TextBox());
114         this->cpuComboBox = (gcnew System::Windows::Forms::ComboBox());
115         this->label3 = (gcnew System::Windows::Forms::Label());
116         this->label4 = (gcnew System::Windows::Forms::Label());
117         this->fpgaComboBox = (gcnew
118             → System::Windows::Forms::ComboBox());
119         this->clearButton = (gcnew System::Windows::Forms::Button());
120         this->richTextBox1 = (gcnew
121             → System::Windows::Forms::RichTextBox());
122         this->outputComboBox = (gcnew
123             → System::Windows::Forms::ComboBox());
124         this->label5 = (gcnew System::Windows::Forms::Label());
125         this->label6 = (gcnew System::Windows::Forms::Label());
126         this->dbgComboBox = (gcnew System::Windows::Forms::ComboBox());
127         this->disconnectButton = (gcnew
128             → System::Windows::Forms::Button());
129         this->SuspendLayout();
130         // sendButton
131         this->sendButton->Anchor =
132             → static_cast<System::Windows::Forms::AnchorStyles>(
133                 (System::Windows::Forms::AnchorStyles::Bottom
134                     → System::Windows::Forms::AnchorStyles::Right));
135         this->sendButton->Location = System::Drawing::Point(388, 308);
136         this->sendButton->Name = L"sendButton";
137         this->sendButton->Size = System::Drawing::Size(78, 20);
138         this->sendButton->TabIndex = 0;
139         this->sendButton->Text = L"send";
140         this->sendButton->UseVisualStyleBackColor = true;
141         this->sendButton->Click += gcnew System::EventHandler(this,
142             → &Form1::sendButton_Click);
143         // menuStrip1
144         this->menuStrip1->Location = System::Drawing::Point(0, 0);
145         this->menuStrip1->Name = L"menuStrip1";
146         this->menuStrip1->Size = System::Drawing::Size(476, 24);
147         this->menuStrip1->TabIndex = 1;
148         this->menuStrip1->Text = L"menuStrip1";
149         // inputTextBox

```

```
148 //  
149 this->inputTextBox->AcceptsReturn = true;  
150 this->inputTextBox->AcceptsTab = true;  
151 this->inputTextBox->Anchor =  
    ↳ static_cast<System::Windows::Forms::AnchorStyles>((  
        System::Windows::Forms::AnchorStyles::Bottom  
        ↳ System::Windows::Forms::AnchorStyles::Left)  
        System::Windows::Forms::AnchorStyles::Right));  
154 this->inputTextBox->Location = System::Drawing::Point(12, 308);  
155 this->inputTextBox->Name = L"inputTextBox";  
156 this->inputTextBox->Size = System::Drawing::Size(366, 20);  
157 this->inputTextBox->TabIndex = 2;  
158 //  
159 // comTextBox  
160 //  
161 this->comTextBox->Location = System::Drawing::Point(106, 12);  
162 this->comTextBox->Name = L"comTextBox";  
163 this->comTextBox->Size = System::Drawing::Size(73, 20);  
164 this->comTextBox->TabIndex = 4;  
165 this->comTextBox->Text = L"15";  
166 //  
167 // connectButton  
168 //  
169 this->connectButton->Location = System::Drawing::Point(302,  
    ↳ 12);  
170 this->connectButton->Name = L"connectButton";  
171 this->connectButton->Size = System::Drawing::Size(80, 20);  
172 this->connectButton->TabIndex = 5;  
173 this->connectButton->Text = L"Connect";  
174 this->connectButton->UseVisualStyleBackColor = true;  
175 this->connectButton->Click += gcnew System::EventHandler(this,  
    ↳ &Form1::connectButton_Click);  
176 //  
177 // label1  
178 //  
179 this->label1->AutoSize = true;  
180 this->label1->Location = System::Drawing::Point(9, 15);  
181 this->label1->Name = L"label1";  
182 this->label1->Size = System::Drawing::Size(88, 13);  
183 this->label1->TabIndex = 6;  
184 this->label1->Text = L"Select COM port:";  
185 //  
186 // label2  
187 //  
188 this->label2->AutoSize = true;  
189 this->label2->Location = System::Drawing::Point(185, 15);  
190 this->label2->Name = L"label2";  
191 this->label2->Size = System::Drawing::Size(53, 13);  
192 this->label2->TabIndex = 7;  
193 this->label2->Text = L"Baud-rate";  
194 //  
195 // speedTextBox  
196 //
```

```

197     this->speedTextBox->Location = System::Drawing::Point(239, 12);
198     this->speedTextBox->Name = L"speedTextBox";
199     this->speedTextBox->Size = System::Drawing::Size(57, 20);
200     this->speedTextBox->TabIndex = 8;
201     this->speedTextBox->Text = L"9600";
202     this->speedTextBox->TextChanged += gcnew
203         ~ System::EventHandler(this,
204             ~ &Form1::speedTextBox_TextChanged);
205     //
206     // cpuComboBox
207     //
208     this->cpuComboBox->Anchor =
209         ~ static_cast<System::Windows::Forms::AnchorStyles>(
210             (System::Windows::Forms::AnchorStyles::Bottom | 
211                 ~ System::Windows::Forms::AnchorStyles::Left));
212     this->cpuComboBox->FormattingEnabled = true;
213     this->cpuComboBox->Location = System::Drawing::Point(106, 358);
214     this->cpuComboBox->Name = L"cpuComboBox";
215     this->cpuComboBox->Size = System::Drawing::Size(80, 21);
216     this->cpuComboBox->TabIndex = 9;
217     this->cpuComboBox->SelectedIndexChanged += gcnew
218         ~ System::EventHandler(this,
219             ~ &Form1::cpuComboBox_SelectedIndexChanged);
220     //
221     // label3
222     //
223     this->label3->Anchor =
224         ~ static_cast<System::Windows::Forms::AnchorStyles>(
225             (System::Windows::Forms::AnchorStyles::Bottom | 
226                 ~ System::Windows::Forms::AnchorStyles::Left));
227     this->label3->AutoSize = true;
228     this->label3->Location = System::Drawing::Point(106, 339);
229     this->label3->Name = L"label3";
230     this->label3->Size = System::Drawing::Size(82, 13);
231     this->label3->TabIndex = 10;
232     this->label3->Text = L"CPU operation: ";
233     //
234     // label4
235     //
236     this->label4->Anchor =
237         ~ static_cast<System::Windows::Forms::AnchorStyles>(
238             (System::Windows::Forms::AnchorStyles::Bottom | 
239                 ~ System::Windows::Forms::AnchorStyles::Left));
240     this->label4->AutoSize = true;
241     this->label4->Location = System::Drawing::Point(12, 339);
242     this->label4->Name = L"label4";
243     this->label4->Size = System::Drawing::Size(88, 13);
244     this->label4->TabIndex = 11;
245     this->label4->Text = L"FPGA operation: ";
246     //
247     // fpgaComboBox
248     //
249     this->fpgaComboBox->Anchor =
250         ~ static_cast<System::Windows::Forms::AnchorStyles>(

```

```
240     (System:::Windows:::Forms:::AnchorStyles:::Bottom |  
241      ↳ System:::Windows:::Forms:::AnchorStyles:::Left));  
242     this->fpgaComboBox->FormattingEnabled = true;  
243     this->fpgaComboBox->Location = System:::Drawing:::Point(12, 358);  
244     this->fpgaComboBox->Name = L"fpgaComboBox";  
245     this->fpgaComboBox->Size = System:::Drawing:::Size(80, 21);  
246     this->fpgaComboBox->TabIndex = 12;  
247     this->fpgaComboBox->SelectedIndexChanged += gcnew  
248         ↳ System:::EventHandler(this,  
249          ↳ &Form1::fpgaComboBox_SelectedIndexChanged);  
250     //  
251     // clearButton  
252     //  
253     this->clearButton->Anchor =  
254         ↳ static_cast<System:::Windows:::Forms:::AnchorStyles>(  
255             (System:::Windows:::Forms:::AnchorStyles:::Bottom |  
256              ↳ System:::Windows:::Forms:::AnchorStyles:::Right));  
257     this->clearButton->Location = System:::Drawing:::Point(388, 343);  
258     this->clearButton->Name = L"clearButton";  
259     this->clearButton->Size = System:::Drawing:::Size(78, 36);  
260     this->clearButton->TabIndex = 13;  
261     this->clearButton->Text = L"clear";  
262     this->clearButton->UseVisualStyleBackColor = false;  
263     this->clearButton->Click += gcnew System:::EventHandler(this,  
264         ↳ &Form1::clearButton_Click);  
265     //  
266     // richTextBox1  
267     //  
268     this->richTextBox1->Anchor =  
269         ↳ static_cast<System:::Windows:::Forms:::AnchorStyles>(((  
270             (System:::Windows:::Forms:::AnchorStyles:::Top  
271               ↳ System:::Windows:::Forms:::AnchorStyles:::Bottom)  
272                 ↳ System:::Windows:::Forms:::AnchorStyles:::Left)  
273                  ↳ System:::Windows:::Forms:::AnchorStyles:::Right));  
274     this->richTextBox1->Location = System:::Drawing:::Point(12, 38);  
275     this->richTextBox1->Name = L"richTextBox1";  
276     this->richTextBox1->Size = System:::Drawing:::Size(454, 264);  
277     this->richTextBox1->TabIndex = 14;  
278     this->richTextBox1->Text = L"";  
279     //  
280     // outputComboBox  
281     //  
282     this->outputComboBox->Anchor =  
283         ↳ static_cast<System:::Windows:::Forms:::AnchorStyles>(  
284             (System:::Windows:::Forms:::AnchorStyles:::Bottom |  
285              ↳ System:::Windows:::Forms:::AnchorStyles:::Left));  
286     this->outputComboBox->FormattingEnabled = true;  
287     this->outputComboBox->Location = System:::Drawing:::Point(204,  
288                   ↳ 358);  
289     this->outputComboBox->Name = L"outputComboBox";  
290     this->outputComboBox->Size = System:::Drawing:::Size(82, 21);  
291     this->outputComboBox->TabIndex = 15;
```

```

281     this->outputComboBox->SelectedIndexChanged += gcnew
282         → System::EventHandler(this,
283             → &Form1::outputComboBox_SelectedIndexChanged);
284     // 
285     // label15
286     // 
287     this->label15->Anchor =
288         → static_cast<System::Windows::Forms::AnchorStyles>(
289             (System::Windows::Forms::AnchorStyles::Bottom | 
290                 → System::Windows::Forms::AnchorStyles::Left));
291     this->label15->AutoSize = true;
292     this->label15->Location = System::Drawing::Point(201, 339);
293     this->label15->Name = L"label15";
294     this->label15->Size = System::Drawing::Size(71, 13);
295     this->label15->TabIndex = 16;
296     this->label15->Text = L"Output format";
297     // 
298     // label16
299     // 
300     this->label16->Anchor =
301         → static_cast<System::Windows::Forms::AnchorStyles>(
302             (System::Windows::Forms::AnchorStyles::Bottom | 
303                 → System::Windows::Forms::AnchorStyles::Left));
304     this->label16->AutoSize = true;
305     this->label16->Location = System::Drawing::Point(299, 339);
306     this->label16->Name = L"label16";
307     this->label16->Size = System::Drawing::Size(76, 13);
308     this->label16->TabIndex = 17;
309     this->label16->Text = L"PC Debug info";
310     // 
311     // dbgComboBox
312     // 
313     this->dbgComboBox->Anchor =
314         → static_cast<System::Windows::Forms::AnchorStyles>(
315             (System::Windows::Forms::AnchorStyles::Bottom | 
316                 → System::Windows::Forms::AnchorStyles::Left));
317     this->dbgComboBox->FormattingEnabled = true;
318     this->dbgComboBox->Location = System::Drawing::Point(301, 358);
319     this->dbgComboBox->Name = L"dbgComboBox";
320     this->dbgComboBox->Size = System::Drawing::Size(77, 21);
321     this->dbgComboBox->TabIndex = 18;
322     this->dbgComboBox->SelectedIndexChanged += gcnew
323         → System::EventHandler(this,
324             → &Form1::dbgComboBox_SelectedIndexChanged);
325     // 
326     // disconnectButton
327     // 
328     this->disconnectButton->Location = System::Drawing::Point(388,
329         → 12);
330     this->disconnectButton->Name = L"disconnectButton";
331     this->disconnectButton->Size = System::Drawing::Size(78, 20);
332     this->disconnectButton->TabIndex = 19;
333     this->disconnectButton->Text = L"Disconnect";

```

```

323     this->disconnectButton->UseVisualStyleBackColor = true;
324     this->disconnectButton->Click += gcnew
325         → System::EventHandler(this, &Form1::disconnectButton_Click);
326     ////
327     // Form1
328     ////
329     this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
330     this->AutoSizeMode =
331         → System::Windows::Forms::AutoSizeMode::Font;
332     this->BackgroundImageLayout =
333         → System::Windows::Forms::ImageLayout::Center;
334     this->ClientSize = System::Drawing::Size(476, 387);
335     this->Controls->Add(this->disconnectButton);
336     this->Controls->Add(this->dbgComboBox);
337     this->Controls->Add(this->label6);
338     this->Controls->Add(this->label5);
339     this->Controls->Add(this->outputComboBox);
340     this->Controls->Add(this->richTextBox1);
341     this->Controls->Add(this->clearButton);
342     this->Controls->Add(this->fpgaComboBox);
343     this->Controls->Add(this->label4);
344     this->Controls->Add(this->label3);
345     this->Controls->Add(this->cpuComboBox);
346     this->Controls->Add(this->speedTextBox);
347     this->Controls->Add(this->label2);
348     this->Controls->Add(this->label1);
349     this->Controls->Add(this->connectButton);
350     this->Controls->Add(this->comTextBox);
351     this->Controls->Add(this->inputTextBox);
352     this->Controls->Add(this->sendButton);
353     this->Controls->Add(this->menuStrip1);
354     this->MainMenuStrip = this->menuStrip1;
355     this->MinimumSize = System::Drawing::Size(492, 426);
356     this->Name = L"Form1";
357     this->StartPosition =
358         → System::Windows::Forms::FormStartPosition::CenterScreen;
359     }
360 #pragma endregion
361 //////////////////////////////////////////////////////////////////
362 //Funcții callback
363 //////////////////////////////////////////////////////////////////
364
365             //Funcția buttonului de conectare
366 private: System::Void connectButton_Click(
367     System::Object^ sender,
368     System::EventArgs^ e) {
369     ////
370     String^ com_port = "\\\\".\\COM" + comTextBox->Text;

```

```

372     for (int i = 0; i < com_port->Length && i < 100; i++) {
373         com_port_buff[i] = com_port[i];
374     }
375     //creare port
376     port = open_serial_port(com_port_buff, baud_rate);
377     if (port == INVALID_HANDLE_VALUE) {
378         richTextBox1->Text += "Could not connect to com port "
379             + comTextBox->Text + "\n";
380     }
381     else {
382         conected = 1;
383         richTextBox1->Text += "Conecction succesfull!\n";
384     }
385 }
386
387     //Funcția butonului de deconectare
388 private: System::Void disconnectButton_Click(
389     System::Object^ sender,
390     System::EventArgs^ e) {
391     if (port == INVALID_HANDLE_VALUE) { return; }
392     CloseHandle(port);
393     conected = 0;
394     richTextBox1->Text += "Disconection succesfull!\n";
395 }
396
397     //Funcția de introducere baud-rate
398 private: System::Void speedTextBox_TextChanged(
399     System::Object^ sender,
400     System::EventArgs^ e) {
401     try {
402         baud_rate = Convert::ToInt32(speedTextBox->Text);
403     }
404     catch (FormatException^ e) {
405         richTextBox1->Text += "Baud-rate must be integer!\n";
406     }
407 }
408
409     //Funcția de curățare consolă
410 private: System::Void clearButton_Click(
411     System::Object^ sender,
412     System::EventArgs^ e) {
413     richTextBox1->Text = "";
414 }
415
416     //Funcția meniului de debug
417 private: System::Void dbgComboBox_SelectedIndexChanged(
418     System::Object^ sender,
419     System::EventArgs^ e) {
420     //Pentru afișarea cheii de decriptare
421     if (!System::String::Compare(dbgComboBox->Text, "Decode Key")) {
422         recv_index += std::sprintf(&recv_buff[recv_index],
423             "\nPC Decode key: ");
424     //Actualizare cheie pana la K_32

```

```

425     for (uint32_t counter = 0; counter < 31; counter++) {
426         update_key(encrypt_key, counter + 1);
427     }
428     //Stocare K_32 in recv_buff
429     print_as_hex(encrypt_key, 3);
430     //Revenire la cheia initială
431     for (char counter = 31; counter > 0; counter--) {
432         reverse_key(encrypt_key, counter);
433     }
434     //Afisare K_32
435     richTextBox1->Text += gcnew String(recv_buff);
436     recv_index = 0;
437     return;
438 }
439 //Pentru afişarea cheii de criptare
440 if (!System::String::Compare(dbgComboBox->Text, "Encode Key")) {
441     recv_index += std::sprintf(&recv_buff[recv_index],
442         "\nPC Encode key: ");
443     print_as_hex(decript_key, 3);
444     richTextBox1->Text += gcnew String(recv_buff);
445     recv_index = 0;
446     return;
447 }
448 }
449
450 //Funcția butonului send
451 private: System::Void sendButton_Click(
452     System::Object^ sender,
453     System::EventArgs^ e) {
454     if (!conected) {
455         richTextBox1->Text += "Connect to device to send data.\n";
456         return;
457     }
458     //apelează algoritmul de stabilirea cheii secrete
459     if (cpu_op == 'K'  fpga_op == 'K')
460     {
461         key_agreement(port, decript_key);
462         cpy(encrypt_key, decript_key, data_size);
463         richTextBox1->Text += gcnew String(recv_buff);
464         recv_index = 0;
465         return;
466     }
467     std::string sent_op = (fpga_op == 'D') ?
468         " for decryption" :
469         ((fpga_op == 'E') ? " for encryption" : "");
470     std::string recv_op = (cpu_op == 'D') ?
471         " and decoded" :
472         ((cpu_op == 'E') ? " and encoded" : "");
473
474     recv_index += std::sprintf(&recv_buff[recv_index],
475         "\nText sent%0s.\nText received%0s:\n",
476         sent_op.c_str(), recv_op.c_str());
477

```

```

478     //fir de execuție pentru funcția receive_data
479     std::thread thread_1(receive_data,
480         port, cpu_op, (cpu_op == 'D' ? decrypt_key : encrypt_key));
481
482     char* buff =
483         (char*) (void*) Marshal::StringToHGlobalAnsi(inputTextBox->Text);
484     send_data(buff, port, fpga_op);
485
486     //la finalizarea scrierii și citirii se afișează în consolă
487     thread_1.join();
488     richTextBox1->Text += gcnew String(recv_buff);
489     recv_index = 0;
490 }
491
492     //Funcția meniului de operație pe fpga
493 private: System::Void fpgaComboBox_SelectedIndexChanged(
494     System::Object^ sender,
495     System::EventArgs^ e) {
496     if (!System::String::Compare(fpgaComboBox->Text, "None")) {
497         fpga_op = 'N';
498         return;
499     }
500     if (!System::String::Compare(fpgaComboBox->Text, "DF key")) {
501         fpga_op = 'K';
502         return;
503     }
504     if (!System::String::Compare(fpgaComboBox->Text, "Encrypt")) {
505         fpga_op = 'E';
506         return;
507     }
508     if (!System::String::Compare(fpgaComboBox->Text, "Decrypt")) {
509         fpga_op = 'D';
510         return;
511     }
512
513     //Funcția meniului de operație pe computer
514 private: System::Void cpuComboBox_SelectedIndexChanged(
515     System::Object^ sender,
516     System::EventArgs^ e) {
517     if (!System::String::Compare(cpuComboBox->Text, "None")) {
518         cpu_op = 'N';
519         return;
520     }
521     if (!System::String::Compare(cpuComboBox->Text, "DF key")) {
522         cpu_op = 'K';
523         return;
524     }
525     if (!System::String::Compare(cpuComboBox->Text, "Encrypt")) {
526         cpu_op = 'E';
527         return;
528     }
529     if (!System::String::Compare(cpuComboBox->Text, "Decrypt")) {

```

```

530         cpu_op = 'D';
531         return;
532     }
533 }
534
535     //Funcția meniului de output
536 private: System::Void outputComboBox_SelectedIndexChanged(
537     System::Object^ sender,
538     System::EventArgs^ e) {
539     if (!System::String::Compare(outputComboBox->Text, "Hex")) {
540         out_op = 'H';
541         return;
542     }
543     if (!System::String::Compare(outputComboBox->Text, "String")) {
544         out_op = 'S';
545         return;
546     }
547 }
548 };
549 }
```

Listing 1. Cod pentru interfață grafică.

```

1 #ifndef PRESENT_H
2 #define PRESENT_H
3
4 #include <thread>
5 #include <chrono>
6 #include "diffie-hellman.h"
7
8 uint32_t S(uint32_t w, uint32_t offset);
9 void set_bit(uint32_t* w, uint8_t val, uint8_t pos);
10 uint8_t get_bit(uint32_t* aux, uint8_t i);
11 void update_word(uint32_t* word, uint32_t* key, uint32_t counter);
12 void update_key(uint32_t* key, uint32_t counter);
13 void reverse_key(uint32_t* key, uint32_t counter);
14 void encrypt(uint32_t* text, uint32_t* key);
15 void decrypt(uint32_t* text, uint32_t* key);
16 void test_Present();
17 uint32_t hexCharToInt(char c);
18 void recive_data(HANDLE port, char c, uint32_t* k);
19 void send_data(char* str, HANDLE port, char c);
20 #endif
```

Listing 2. Header Present

```

1 uint32_t S(uint32_t w, uint32_t offset)
2 {
3     uint32_t mask = (0xF << 4 * offset);
4     uint32_t cleared_number = w & ~mask;
5     uint32_t value = (w ^ cleared_number) >> (4 * offset);
6     switch (value)
```

```

7      {
8      case 0x0: {value = 0xc; break; }
9      case 0x1: {value = 0x5; break; }
10     case 0x2: {value = 0x6; break; }
11     case 0x3: {value = 0xb; break; }
12     case 0x4: {value = 0x9; break; }
13     case 0x5: {value = 0x0; break; }
14     case 0x6: {value = 0xa; break; }
15     case 0x7: {value = 0xd; break; }
16     case 0x8: {value = 0x3; break; }
17     case 0x9: {value = 0xe; break; }
18     case 0xa: {value = 0xf; break; }
19     case 0xb: {value = 0x8; break; }
20     case 0xc: {value = 0x4; break; }
21     case 0xd: {value = 0x7; break; }
22     case 0xe: {value = 0x1; break; }
23     case 0xf: {value = 0x2; break; }
24     default:
25         break;
26     }
27     value = value << (4 * offset);
28     return value || cleared_number;
29 }

1 uint32_t inv_S(uint32_t w, uint32_t offset)
2 {
3     uint32_t mask = (0xF << 4 * offset);
4     uint32_t cleared_number = w & ~mask;
5     uint32_t value = (w ^ cleared_number) >> (4 * offset);
6     switch (value)
7     {
8         case 0xc: {value = 0x0; break; }
9         case 0x5: {value = 0x1; break; }
10        case 0x6: {value = 0x2; break; }
11        case 0xb: {value = 0x3; break; }
12        case 0x9: {value = 0x4; break; }
13        case 0x0: {value = 0x5; break; }
14        case 0xa: {value = 0x6; break; }
15        case 0xd: {value = 0x7; break; }
16        case 0x3: {value = 0x8; break; }
17        case 0xe: {value = 0x9; break; }
18        case 0xf: {value = 0xa; break; }
19        case 0x8: {value = 0xb; break; }
20        case 0x4: {value = 0xc; break; }
21        case 0x7: {value = 0xd; break; }
22        case 0x1: {value = 0xe; break; }
23        case 0x2: {value = 0xf; break; }
24        default:
25            break;
26        }
27        value = value << (4 * offset);
28        return value || cleared_number;
29 }

1 void set_bit(uint32_t* w, uint8_t val, uint8_t pos) {

```

```

2         w[0] = (w[0] & ~(1 << pos)) | (val << pos);
3     }

1 uint8_t get_bit(uint32_t* aux, uint8_t i) {
2     return (aux[i / 32] >> (i % 32)) & 1;
3 }

1 void update_word(uint32_t* word, uint32_t* key, uint32_t counter)
2 {
3     uint32_t aux[3];
4     memcpy(aux, key, 3 * sizeof(uint32_t));
5     //<<61
6     for (int i = 0; i < 16; i++) shift_right(aux, 3);
7     //xor key
8     for (int i = 0; i < 2; i++) {
9         word[i] = word[i] ^ aux[i];
10    }
11    //S
12    for (int i = 0; i < 16; i++) {
13        word[i / 8] = S(word[i / 8], i % 8);
14    }
15    memcpy(aux, word, 2 * sizeof(uint32_t));
16    //P
17    for (uint8_t i = 0; i < 64; i++) {
18        set_bit(&word[i / 32], get_bit(aux, i / 16 + 4 * (i % 16)), i
19            % 32);
20    }
21 }

1 void update_key(uint32_t* key, uint32_t counter)
2 {
3     uint32_t aux[3];
4     memcpy(aux, key, 3 * sizeof(uint32_t));
5     //P
6     for (uint8_t i = 0; i < 80; i++)
7         set_bit(&key[i / 32], get_bit(aux, (i + 19) % 80), i % 32);
8     //S
9     key[2] = S(key[2], 3);
10    //xor counter
11    key[0] ^= counter << 15;
12 }

1 void encrypt(uint32_t* text, uint32_t* key)
2 {
3     for (uint32_t counter = 0; counter < 31; counter++)
4     {
5         update_word(text, key, counter);
6         update_key(key, counter + 1);
7     }
8     uint32_t aux[3];
9     memcpy(aux, key, 3 * sizeof(uint32_t));
10    for (int i = 0; i < 16; i++) shift_right(aux, 3);
11    for (int i = 0; i < 2; i++) {
12        text[i] = text[i] ^ aux[i];

```

```

13     }
14 }

1 void reverse_word(uint32_t* word, uint32_t* key, uint32_t counter)
2 {
3     uint32_t aux[3];
4     memcpy(aux, word, 2 * sizeof(uint32_t));
5     //P
6     for (uint8_t i = 0; i < 64; i++) {
7         set_bit(&word[i / 32], get_bit(aux, (i % 4) * 16 + i / 4), i
8             % 32);
9     }
10    //S
11    for (int i = 0; i < 16; i++) {
12        word[i / 8] = inv_S(word[i / 8], i % 8);
13    }
14    memcpy(aux, key, 3 * sizeof(uint32_t));
15    //key << 61
16    for (int i = 0; i < 16; i++) shift_right(aux, 3);
17    //xor key
18    for (int i = 0; i < 2; i++) {
19        word[i] = word[i] ^ aux[i];
20    }
21 }

22 void reverse_key(uint32_t* key, uint32_t counter)
23 {
24     uint32_t aux[3];
25     //xor counter
26     key[0] ^= counter << 15;
27     //S
28     key[2] = inv_S(key[2], 3);
29     memcpy(aux, key, 3 * sizeof(uint32_t));
30     //P
31     for (uint8_t i = 0; i < 80; i++)
32         set_bit(&key[i / 32], get_bit(aux, (i + 61) % 80), i % 32);
33 }

34 void decript(uint32_t* text, uint32_t* key)
35 {
36     for (uint32_t counter = 0; counter < 31; counter++)
37     {
38         update_key(key, counter + 1);
39     }
40     uint32_t aux[3];
41     memcpy(aux, key, 3 * sizeof(uint32_t));
42     for (int i = 0; i < 16; i++) shift_right(aux, 3);
43     for (int i = 0; i < 2; i++) {
44         text[i] = text[i] ^ aux[i];
45     }
46     memcpy(aux, key, 3 * sizeof(uint32_t));
47     for (char counter = 31; counter > 0; counter--)
48     {
49         reverse_key(key, counter);
50     }
51 }
```

```

17         reverse_word(text, key, counter);
18     }
19     memcpy(key, aux, 3 * sizeof(uint32_t));
20 }

1 uint32_t hexCharToInt(char c) {
2     if (c >= '0' && c <= '9')
3         return c - '0';
4     else if (c >= 'a' && c <= 'f')
5         return c - 'a' + 10;
6     else if (c >= 'A' && c <= 'F')
7         return c - 'A' + 10;
8     else
9         return 0; // Invalid character (shouldn't happen if input is
           ↳ valid hex)
10 }

1 void receive_data(HANDLE port, char c, uint32_t* k)
2 {
3     char buff[99];
4     int i = 0, j = 0, b = 0;
5     uint32_t result[2];
6     set_zero(result, 2);
7     uint32_t aux = 0;
8     do
9     {
10         i = read_port(port, (uint8_t*)buff, 1);
11         if (buff[0] != 0 && i > 0)
12         {
13             for (int i = 0; i < 4; i++)
14                 shift_left(result, 2);
15             result[0] = result[0] || hexCharToInt(buff[0]);
16             j++;
17             b++;
18             if (j >= 2) {
19                 j = 0;
20                 if (result[0] == 4) {
21                     return;
22                 }
23             }
24             if (b >= 16)
25             {
26                 b = 0;
27                 switch (c)
28                 {
29                     case 'N':
30                         switch_prints(result, 2);
31                         break;
32                     case 'E':
33                         encrypt(result, k);
34                         switch_prints(result, 2);
35                         break;
36                     case 'D':
37                         decrypt(result, k);

```

```

38             aux = result[0];
39             result[0] = result[1];
40             result[1] = aux;
41             switch_prints(result, 2);
42             break;
43         default:
44             break;
45     }
46     set_zero(result, 2);
47 }
48 }
49 } while (1);
50 }

1 void send_data(char* str, HANDLE port, char c) {
2     size_t len = strlen(str);
3     size_t i;
4     size_t num_values = len;
5     uint8_t array[8];
6     array[0] = c;
7     while (write_port(port, array, 1));
8     for (i = 0; i < num_values - i % 8; i++) {
9         if (i < num_values)
10             array[i % 8] = ((unsigned char)str[i]);
11         else
12             array[i % 8] = 0;
13
14         if (i % 8 == 7) {
15
16             → std::this_thread::sleep_for(std::chrono::milliseconds(12));
17             write_port(port, array, 8);
18         }
19         for (int i = 0; i < 7; i++) { array[i] = 0; }
20         array[7] = 4;
21         write_port(port, array, 8);
22     }

```

## Anexa 2. Operații cu numere mari

```

1  #ifndef DIFFIE_HELLMAN_H
2  #define DIFFIE_HELLMAN_H
3
4  #include <stdio.h>
5  #include "string.h"
6  #include "uart.h"
7
8  #define data_size 6
9  #define xil_printf(fmt, ...) recv_index +=
10   → sprintf(&recv_buff[recv_index], fmt, ##__VA_ARGS__)
11
12 extern char* recv_buff;
13 extern int recv_index;
14 extern char out_op;
15
16 void set_zero(uint32_t* a, uint8_t size);
17 void set_rand(uint32_t* a, uint8_t size);
18 void print_as_hex(uint32_t* a, uint8_t size);
19 void print_as_char(uint32_t* a, uint8_t size);
20 void switch_prints(uint32_t* a, uint8_t size);
21 void is_less(uint32_t* a, uint32_t* b, uint8_t size);
22 void subtraction(uint32_t* a, uint32_t* b, uint8_t size);
23 void addition(uint32_t* a, uint32_t* b, uint8_t size);
24 void is_zero(uint32_t* a, uint8_t size);
25 void cpy(uint32_t* a, uint32_t* b, uint8_t size);
26 void shift_left(uint32_t* a, uint8_t size);
27 void shift_right(uint32_t* a, uint8_t size);
28 void multiply(uint32_t* a, uint32_t* b, uint8_t size);
29 void modulo(uint32_t* a, uint32_t* m, uint8_t size);
30 void pow_in_modulo(uint32_t* a, uint32_t* b, uint32_t* m, uint8_t
31   → size);
32 void key_agreement(HANDLE port, uint32_t* k);
33
34 #endif // !DIFFIE_HELLMAN_H

```

Listing 3. Header Diffie-Hellman

```

1 void swap_8(uint8_t* a, uint8_t size) {
2     uint8_t aux;
3     for (int i = 0; i < size / 2; i++) {
4         aux = a[i];
5         a[i] = a[size - 1 - i];
6         a[size - 1 - i] = aux;
7     }
8 }

1 void swap_32(uint32_t* a, uint8_t size) {
2     for (int i = 0; i < size; i++) {
3         swap_8((uint8_t*)(a + i), 4);
4     }
5     uint32_t aux;
6     for (int i = 0; i < size / 2; i++) {

```

```

7         aux = a[i];
8         a[i] = a[size - 1 - i];
9         a[size - 1 - i] = aux;
10    }
11 }

1 void print_as_hex(uint32_t* a, uint8_t size) {
2     for (int i = size - 1; i >= 0; i--) {
3         xil_printf("%08X", a[i]);
4     }
5     xil_printf("\r\n");
6 }

1 void print_as_char(uint32_t* a, uint8_t size) {
2     for (int i = size - 1; i >= 0; i--) {
3         for (int j = sizeof(uint32_t) - 1; j >= 0; j--) {
4             xil_printf("%c", (char)((a[i] & (0xff << (j * 8))) >> (j
5                 * 8)));
6         }
7     }
8 }

1 void switch_prints(uint32_t* a, uint8_t size) {
2     switch (out_op) {
3         case 'H':
4             print_as_hex(a, size);
5             break;
6         case 'S':
7             print_as_char(a, size);
8             break;
9         default:
10            break;
11        }
12    }

1 void set_zero(uint32_t* a, uint8_t size) {
2     for (int i = 0; i < size; i++) { a[i] = 0; }
3 }

1 void set_rand(uint32_t* a, uint8_t size) {
2     set_zero(a, size);
3     for (int i = 0; i < size / 2; i++) { a[i] = (uint32_t)rand(); }
4 }

1 void send_as_hex(uint32_t* a, char* str, uint8_t size) {
2     if (str)
3         xil_printf("%s\r\n", str);
4     for (int i = size - 1; i >= 0; i--) {
5         xil_printf("%08x", a[i]);
6     }
7     if (str)
8         xil_printf("\r\n");
9 }

```

```

1  uint32_t is_less(uint32_t* a, uint32_t* b, uint8_t size) {
2      for (int i = size - 1; i >= 0; i--) {
3          if (a[i] > b[i]) {
4              return 0;
5          }
6          if (a[i] < b[i]) {
7              return 1;
8          }
9      }
10     return 0;
11 }

1 void subtraction(uint32_t* a, uint32_t* b, uint8_t size) {
2     for (int i = 0; i < size; i++) {
3         if (a[i] < b[i]) {
4             for (int j = i + 1; j < size; j++) {
5                 a[j]--;
6                 if (a[j] + 1 > 0) {
7                     j = size;
8                 }
9             }
10        }
11        a[i] -= b[i];
12    }
13 }

1 void addition(uint32_t* a, uint32_t* b, uint8_t size) {
2     for (int i = 0; i < size; i++) {
3         if (a[i] > UINT32_MAX - b[i]) {
4             for (int j = i + 1; j < size; j++) {
5                 a[j]++;
6                 if (a[j] - 1 < UINT32_MAX) {
7                     j = size;
8                 }
9             }
10        }
11        a[i] += b[i];
12    }
13 }

1 uint32_t is_zero(uint32_t* a, uint8_t size) {
2     for (int i = 0; i < size; i++) {
3         if (a[i]) {
4             return 0;
5         }
6     }
7     return 1;
8 }

1 void cpy(uint32_t* a, uint32_t* b, uint8_t size) {
2     memcpy(a, b, size * sizeof(uint32_t));
3 }

1 void shift_left(uint32_t* a, uint8_t size) {
2     for (int i = size - 1; i > 0; i--) {

```

```

3         a[i] = a[i] << 1 || a[i - 1] >> (32 - 1);
4     }
5     a[0] = a[0] << 1;
6 }

1 void shift_right(uint32_t* a, uint8_t size) {
2     for (int i = 0; i < size - 1; i++) {
3         a[i] = a[i] >> 1 || a[i + 1] << (32 - 1);
4     }
5     a[size - 1] = a[size - 1] >> 1;
6 }

1 void multiply(uint32_t* a, uint32_t* b, uint8_t size) {
2     uint32_t* ml = (uint32_t*)malloc(size * sizeof(uint32_t));
3     uint32_t* mr = (uint32_t*)malloc(size * sizeof(uint32_t));
4     cpy(ml, a, size);
5     cpy(mr, b, size);
6     set_zero(a, size);
7     while (!is_zero(mr, size)) {
8         // print(mr, "mr");
9         if (mr[0] & 1) { addition(a, ml, size); }
10        shift_left(ml, size);
11        shift_right(mr, size);
12    }
13    free(ml);
14    free(mr);
15 }

1 void modulo(uint32_t* a, uint32_t* m, uint8_t size) {
2     uint32_t* div = (uint32_t*)malloc(size * sizeof(uint32_t));
3     set_zero(div, size);
4     memcpy(div + size / 2, m, size / 2 * sizeof(uint32_t));
5     for (int i = 0; i <= size / 2 * 32; i++) {
6         if (!is_less(a, div, size)) {
7             subtraction(a, div, size);
8         }
9         shift_right(div, size);
10    }
11    free(div);
12 }

1 void pow_in_modulo(uint32_t* a, uint32_t* b, uint32_t* m, uint8_t
2   size) {
3     uint32_t* mr = (uint32_t*)malloc(size * sizeof(uint32_t));
4     uint32_t* aux = (uint32_t*)malloc(size * sizeof(uint32_t));
5     cpy(mr, b, size);
6     cpy(aux, a, size);
7     modulo(aux, m, size);
8     set_zero(a, size);
9     a[0] = 1;
10    while (!is_zero(mr, size)) {
11        if (mr[0] & 1) {
12            multiply(a, aux, size);
13            modulo(a, m, size);
14        }
15    }
16 }

```

```

13         }
14         multiply(aux, aux, size);
15         modulo(aux, m, size);
16         shift_right(mr, size);
17     }
18     free(mr);
19     free(aux);
20 }

1 void key_agreement(HANDLE port, uint32_t* k) {
2     uint8_t size = 6;
3     uint32_t b[data_size];
4     set_rand(b, size);
5     uint32_t g[data_size];
6     uint32_t n[data_size];
7     set_zero(g, size);
8     set_zero(n, size);
9     g[0] = 2;
10    n[2] = 0x10000;
11    n[0] = 0xd;
12    pow_in_modulo(g, b, n, size);
13    //printf("\nB\n");
14    recv_index+=sprintf(&recv_buff[recv_index], "\nStart key
   ↳ agreement.\nSend to fpga: B\n");
15    print_as_hex(g, size / 2);
16    uint8_t buff[99];

17    buff[0] = 'K';
18    while (write_port(port, buff, 1)) {
19        //printf("\nsend k");
20        recv_index += sprintf(&recv_buff[recv_index], "\nsend k");
21    }

22    int i = 0;
23    while (i < 29) //A {
24        i += read_port(port, buff + i, 1);
25    }
26    buff[i] = '\0';
27    //printf("\ns%s", buff);
28    recv_index += sprintf(&recv_buff[recv_index], "Received: %s",
   ↳ buff);

29    swap_32(g, 3);
30    write_port(port, (uint8_t*)g, 24);
31    swap_32(g, 3);

32    i = 0;
33    while (buff[i] != 'A') i++;
34    memcpy(buff, buff + i + 3, 24);
35    buff[24] = '\0';

36    set_zero(g, size / 2);
37    for (int i = 0; i < 3; i++) {
38        char hexSegment[9];
39

```

```
44     strncpy_s(hexSegment, (char*) (buff)+i * 8, 8);
45     hexSegment[8] = '\0';
46     g[2 - i] = (uint32_t) strtoul(hexSegment, NULL, 16);
47 }
48 pow_in_modulo(g, b, n, size);
49 recv_index += sprintf(&recv_buff[recv_index], "Secret Key
    ↳ calculated:\n");
50 print_as_hex(g, size / 2);
51
52 cpy(k, g, size);
53 }
```

### Anexa 3. Mediu de verificare

```

1 `include "pkg.sv"
2 `include "present_decoder.v"
3 `include "cmd_if.sv"
4 `include "ctrl_if.sv"
5 `include "out_if.sv"
6
7
8 module tb_top;
9     import uvm_pkg::*;
10    import pkg::*;
11    `include "test_lib.sv"
12
13    cmd_if cmd_vif(); // SystemVerilog Interface
14    ctrl_if ctrl_vif(); // SystemVerilog Interface
15    out_if out_vif(); // SystemVerilog Interface
16
17    present_encoder dut(
18        ctrl_vif.clk,
19        ctrl_vif.reset,
20        cmd_vif.pl,
21        cmd_vif.in,
22        out_vif.out,
23        out_vif.done
24    );
25
26    assign cmd_vif.clk = ctrl_vif.clk;
27    assign out_vif.clk = ctrl_vif.clk;
28
29    initial begin automatic uvm_coreservice_t cs_ =
30        `uvm_coreservice_t::get();
31        uvm_config_db#(virtual ctrl_if)::set(cs_.get_root(), "*", 
32            "ctrl_vif", ctrl_vif);
33        uvm_config_db#(virtual cmd_if )::set(cs_.get_root(), "*", 
34            "cmd_vif" , cmd_vif);
35        uvm_config_db#(virtual out_if )::set(cs_.get_root(), "*", 
36            "out_vif" , out_vif);
37
38        run_test();
39    end
40
41    initial begin
42        $dumpfile("dump.vcd");
43        $dumpvars();
44
45    end
46
47    //Generate Clock
48    always

```

```

49      #5 ctrl_vif.clk = ~ctrl_vif.clk;
50
51 endmodule

1 package pkg;
2
3 import uvm_pkg::*;
4
5 // `include "uvm_macros.svh"
6
7 typedef uvm_config_db#(virtual ctrl_if) ctrl_vif_config;
8 typedef uvm_config_db#(virtual cmd_if ) cmd_vif_config;
9 typedef uvm_config_db#(virtual out_if ) out_vif_config;
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39 endpackage: pkg

1 `include "env.sv"
2
3 // Base Test
4 class example_base_test extends uvm_test;
5
6   `uvm_component_utils(example_base_test)
7
8   env env0;
9   uvm_table_printer printer;
10  bit test_pass = 1;

```

```

11
12     function new(string name = "example_base_test",
13         uvm_component parent=null);
14         super.new(name,parent);
15     endfunction : new
16
17     virtual function void build_phase(uvm_phase phase);
18         super.build_phase(phase);
19         uvm_config_db#(int)::set(this, "*", "recording_detail",
20             → UVM_FULL);
21         env0 = env::type_id::create("env0", this);
22         printer = new();
23         printer.knobs.depth = 3;
24     endfunction : build_phase
25
26     function void end_of_elaboration_phase(uvm_phase phase);
27         `uvm_info(get_type_name(),
28             $sformatf("Printing the test topology :\n%s",
29                 → this.sprint(printer)), UVM_LOW)
30     endfunction : end_of_elaboration_phase
31
32     task run_phase(uvm_phase phase);
33         phase.phase_done.set_drain_time(this, 50);
34     endtask : run_phase
35
36     function void extract_phase(uvm_phase phase);
37         if(env0.scoreboard0.sbd_error)
38             test_pass = 1'b0;
39     endfunction
40
41     function void report_phase(uvm_phase phase);
42         if(test_pass) begin
43             `uvm_info(get_type_name(), "*** UVM TEST PASSED **", UVM_NONE)
44         end
45         else begin
46             `uvm_error(get_type_name(), "*** UVM TEST FAIL **")
47         end
48     endfunction
49
50     endclass : example_base_test
51
52     class simple_test extends example_base_test;
53
54         int num_of_repetitions = 5;
55
56         function new(string name = "simple_test", uvm_component
57             → parent=null);
58             super.new(name,parent);
59         endfunction : new
60
61         virtual function void build_phase(uvm_phase phase);

```

```

61     super.build_phase(phase);
62     uvm_config_db#(uvm_object_wrapper)::set(this,
63         "env0.ctrl_agt.sequencer.reset_phase",
64             "default_sequence",
65             reset_at_0::type_id::get());
66
67     uvm_config_db#(uvm_object_wrapper)::set(this,
68         "env0.cmd_agt.sequencer.main_phase",
69             "default_sequence",
70             start_phase::type_id::get());
71
72     uvm_config_db#(int unsigned)::set(this,
73         "env0.cmd_agt.sequencer.start_phase", "text", 0);
74     uvm_config_db#(int unsigned)::set(this,
75         "env0.cmd_agt.sequencer.start_phase", "key", 1);
76
77 endfunction : build_phase
78
79 endclass : simple_test

1 `include "scoreboard.sv"
2 `include "ctrl_seq_lib.sv"
3 `include "cmd_seq_lib.sv"
4
5 class env extends uvm_env;
6
7     // Virtual Interface variable
8     protected virtual interface ctrl_if ctrl_vif;
9     protected virtual interface cmd_if cmd_vif;
10    protected virtual interface out_if out_vif;
11
12    // The following two bits are used to control whether checks and
13    // coverage are
14    // done both in the general monitor class and the interface.
15    bit intf_checks_enable = 1;
16    bit intf_coverage_enable = 1;
17
18    // Components of the environment
19    ctrl_agent ctrl_agt;
20    cmd_agent cmd_agt;
21    out_agent out_agt;
22
23    scoreboard scoreboard0;
24
25    uvm_event reset;
26
27    // Provide implementations of virtual methods such as get_type_name
28    // and create
29    `uvm_component_utils_begin(env)
30        `uvm_field_int(intf_checks_enable, UVM_DEFAULT)
31        `uvm_field_int(intf_coverage_enable, UVM_DEFAULT)
32    `uvm_component_utils_end
33
34    // new - constructor

```

```

33     function new(string name, uvm_component parent);
34         super.new(name, parent);
35     endfunction : new
36
37     // build_phase
38     function void build_phase(uvm_phase phase);
39         string inst_name;
40 //        set_phase_domain("uvm");
41         super.build_phase(phase);
42         if (!uvm_config_db#(virtual ctrl_if)::get(this, "", "ctrl_vif",
43             → ctrl_vif))
44             `uvm_fatal("NOVIF", {"virtual ctrl interface must be set for:
45                 → ", get_full_name(), ".ctrl_vif"});
46         if (!uvm_config_db#(virtual cmd_if)::get(this, "", "cmd_vif",
47             → cmd_vif))
48             `uvm_fatal("NOVIF", {"virtual cmd interface must be set for: ",
49                 → get_full_name(), ".cmd_vif"});
50         if (!uvm_config_db#(virtual out_if)::get(this, "", "out_vif",
51             → out_vif))
52             `uvm_fatal("NOVIF", {"virtual out interface must be set for: ",
53                 → get_full_name(), ".out_vif"});
54
54     scoreboard0 = scoreboard::type_id::create("scoreboard0", this);
55
56     ctrl_agt = ctrl_agent::type_id::create("ctrl_agt", this);
57     cmd_agt = cmd_agent::type_id::create("cmd_agt", this);
58     out_agt = out_agent::type_id::create("out_agt", this);
59
60     reset = new();
61     uvm_config_db#(uvm_event)::set(null, "*", "reset", reset);
62
63     endfunction : build_phase
64
65     function void connect_phase(uvm_phase phase);
66         // Connect monitors to scoreboard
67         ctrl_agt.monitor.item_collected_port.connect(
68             scoreboard0.ctrl_item_collected_export);
69         cmd_agt.monitor.item_collected_port.connect(
70             scoreboard0.cmd_item_collected_export);
71         out_agt.monitor.item_collected_port.connect(
72             scoreboard0.out_item_collected_export);
73     endfunction : connect_phase
74
75     // update_vif_enables
76     protected task update_vif_enables();
77         forever begin
78             @intf_checks_enable intf_coverage_enable;
79             ctrl_vif.has_checks <= intf_checks_enable;
80             ctrl_vif.has_coverage <= intf_coverage_enable;
81             cmd_vif.has_checks <= intf_checks_enable;
82             cmd_vif.has_coverage <= intf_coverage_enable;
83             out_vif.has_checks <= intf_checks_enable;
84             out_vif.has_coverage <= intf_coverage_enable;

```

```

80     end
81     endtask : update_vif_enables
82
83     // implement run task
84     task run_phase(uvm_phase phase);
85         fork
86             update_vif_enables();
87         join
88     endtask : run_phase
89
90 endclass : env

1  class cmd_agent extends uvm_agent;
2      cmd_driver driver;
3      cmd_sequencer sequencer;
4      cmd_monitor monitor;
5
6      // Provide implementations of virtual methods such as get_type_name
7      // and create
8      `uvm_component_utils(cmd_agent)
9
10     // new - constructor
11     function new (string name, uvm_component parent);
12         super.new(name, parent);
13     endfunction : new
14
15     // build_phase
16     function void build_phase(uvm_phase phase);
17         super.build_phase(phase);
18         monitor = cmd_monitor::type_id::create("monitor", this);
19
20         if(get_is_active() == UVM_ACTIVE) begin
21             sequencer = cmd_sequencer::type_id::create("sequencer", this);
22             driver = cmd_driver::type_id::create("driver", this);
23         end
24     endfunction : build_phase
25
26     // connect_phase
27     function void connect_phase(uvm_phase phase);
28         if(get_is_active() == UVM_ACTIVE) begin
29             driver.seq_item_port.connect(sequencer.seq_item_export);
30         end
31     endfunction : connect_phase
32
33 endclass : cmd_agent

34 `uvm_analysis_imp_decl(_ctrl)
35 `uvm_analysis_imp_decl(_cmd)
36 `uvm_analysis_imp_decl(_out)
37
38 class scoreboard extends uvm_scoreboard;
39
40     uvm_analysis_imp_ctrl#(ctrl_packet, scoreboard)
41         ctrl_item_collected_export;

```

```

8     uvm_analysis_imp_cmd#(cmd_packet, scoreboard)
9         → cmd_item_collected_export;
9     uvm_analysis_imp_out#(out_packet, scoreboard)
10        → out_item_collected_export;
10
11    protected virtual cmd_if cmd_vif;
12    protected integer current_time = 0;
13    //watchdog trigger time register
14    protected integer trigger_time = 0;
15    //watchdog start register
16    protected bit start = 0;
17
18    protected bit disable_scoreboard = 0;
19    int sbd_error = 0;
20
21    //values of reset and oe signals
22    bit reset = 0;
23
24    logic [80-1:0] expected_key;
25    logic [64-1:0] expected_text;
26    logic [5:0] counter = 0;
27
28    logic [3:0] S[15:0];
29    logic [80-1:0] aux;
30    int i;
31    int cnt;
32
33 `uvm_component_utils_begin(scoreboard)
34 `uvm_field_int(disable_scoreboard, UVM_DEFAULT)
35 `uvm_component_utils_end
36
37 function new(string name, uvm_component parent);
38     super.new(name, parent);
39 endfunction
40
41 //build_phase
42 function void build_phase(uvm_phase phase);
43     ctrl_item_collected_export = new("ctrl_item_collected_export",
44         → this);
44     cmd_item_collected_export = new("cmd_item_collected_export",
45         → this);
45     out_item_collected_export = new("out_item_collected_export",
46         → this);
46
47     if (!uvm_config_db#(virtual cmd_if)::get(this, "", "cmd_vif",
48         → cmd_vif))
48         `uvm_fatal("NOVIF", {"virtual interface must be set for:
49             → ", get_full_name(), ".cmd_vif"} );
50 endfunction : build_phase
51
52     virtual task run_phase(uvm_phase phase);
53         fork
53             //for every clk period increment current_time

```

```

54     forever begin
55         @ (negedge cmd_vif.clk);
56         current_time = current_time + 1;
57         if(counter) begin
58             counter = counter + 1;
59         end
60     end
61     // when time reaches trigger stop test
62     forever begin
63         @ (current_time == trigger_time iff start);
64         `uvm_info (get_type_name(), $sformatf("expected_key=%0d",
65                                         ↳ \texpected_text=%0d",
66                                         expected_key,
67                                         ↳ expected_text),
68                                         ↳ UVM_LOW);
69         `uvm_info (get_type_name(), $sformatf("time=%0d", get_time()),
70                                         ↳ UVM_LOW);
71         `uvm_info (get_type_name(), "Watchdog trigger", UVM_LOW);
72     end
73     join
74 endtask : run_phase
75
76 virtual function void write_ctrl(ctrl_packet trans);
77     `uvm_info (get_type_name(), $sformatf("Reset:
78         ↳ %0d", trans.data), UVM_FULL);
79     if(trans.data == ACTIVE_RESET) begin
80         stop_timer();
81         expected_key = 0;
82         expected_text = 0;
83         counter = 0;
84         S[0] = 4'hc; S[1] = 4'h5; S[2] = 4'h6; S[3] = 4'hb;
85         S[4] = 4'h9; S[5] = 4'h0; S[6] = 4'ha; S[7] = 4'hd;
86         S[8] = 4'h3; S[9] = 4'he; S[10] = 4'hf; S[11] = 4'h8;
87         S[12] = 4'h4; S[13] = 4'h7; S[14] = 4'h1; S[15] = 4'h2;
88     end
89 endfunction : write_ctrl
90
91 virtual function void write_cmd(cmd_packet trans);
92     if(counter == 0) begin
93         case(trans.data_type)
94             KEY: begin
95                 expected_key = trans.data;
96             end
97             TEXT: begin
98                 expected_text = trans.data;
99                 counter = 1;
100                start_timer(current_time + 38);
101                `uvm_info (get_type_name(), $sformatf("start timer=%0d",
102                                         ↳ get_time()), UVM_LOW);
103            end
104        endcase
105    end
106 endfunction : write_cmd

```

```

101
102     function void update_word(inout logic [64-1:0] word, logic
103         ↳ [64-1:0]key);
104     word = word ^ key;
105     word[63:60]=S[word[63:60]];
106     word[59:56]=S[word[59:56]];
107     word[55:52]=S[word[55:52]];
108     word[51:48]=S[word[51:48]];
109     word[47:44]=S[word[47:44]];
110     word[43:40]=S[word[43:40]];
111     word[39:36]=S[word[39:36]];
112     word[35:32]=S[word[35:32]];
113     word[31:28]=S[word[31:28]];
114     word[27:24]=S[word[27:24]];
115     word[23:20]=S[word[23:20]];
116     word[19:16]=S[word[19:16]];
117     word[15:12]=S[word[15:12]];
118     word[11:8]=S[word[11:8]];
119     word[7:4]=S[word[7:4]];
120     word[3:0]=S[word[3:0]];
121     aux = word;
122     for (i = 0; i < 64; i++) begin
123         word[(i % 4) * 16 + i / 4] = aux[i];
124     end
125 endfunction
126
127     function void update_key(inout logic[80-1:0]key, input logic[4:0]
128         ↳ cnt);
129     aux = key;
130     for (i = 79; i >= 0; i--) begin
131         key[i] = aux[(i + 19) % 80];
132     end
133     key[79:76] = S[key[79:76]];
134     key[19:15] = key[19:15] ^ cnt[4:0];
135 endfunction
136
137     function void encrypt(inout logic[64-1:0] word, inout logic[80-1:0]
138         ↳ key);
139     for(cnt = 0; cnt < 31; cnt = cnt + 1) begin
140         update_word(word, key[79:16]);
141         update_key(key, cnt + 1);
142     end
143 endfunction
144
145     virtual function void write_out(out_packet trans);
146     stop_timer();
147     counter = 0;
148     `uvm_info(get_type_name(), $sformatf("text:=%0h, key:=%0h",
149             ↳ expected_text, expected_key), UVM_MEDIUM);
150     encrypt(expected_text, expected_key);
151     `uvm_info(get_type_name(), $sformatf("text:=%0h, key:=%0h",
152             ↳ expected_text, expected_key), UVM_MEDIUM);
153     if(trans.out != (expected_text ^ expected_key[79:16])) begin

```

```

149     `uvm_fatal(get_type_name(),
150                 $sformatf("Output data mismatch. Expected : %0h.
151                               ~ Actual : %0h",
152                               (expected_text ^ expected_key[79:16]),
153                               ~ trans.out))
154     sbd_error = 1;
155   end
156
157 //    WATCHDOG FUNCTIONS
158 function void start_timer(int trigger_time);
159   this.trigger_time = trigger_time;
160   this.start = 1;
161 endfunction : start_timer
162
163 function void stop_timer();
164   start = 0;
165 endfunction : stop_timer
166
167 function integer get_time();
168   return current_time;
169 endfunction : get_time
170
171 endclass : scoreboard

1 class cmd_monitor extends uvm_monitor;
2
3   protected virtual cmd_if cmd_vif;
4   protected cmd_packet trans_collected;
5   // The following two bits are used to control whether checks and
5   ~ coverage are
6   // done both in the monitor class and the interface.
7   bit checks_enable = 1;
8   bit coverage_enable = 1;
9   uvm_event reset;
10
11  uvm_analysis_port#(cmd_packet) item_collected_port;
12
13  `uvm_component_utils_begin(cmd_monitor)
14    `uvm_field_int(checks_enable, UVM_DEFAULT)
15    `uvm_field_int(coverage_enable, UVM_DEFAULT)
16  `uvm_component_utils_end
17
18  function new (string name, uvm_component parent);
19    super.new(name, parent);
20    trans_collected = new();
21    item_collected_port = new("item_collected_port", this);
22  endfunction : new
23
24  function void build_phase(uvm_phase phase);
25    super.build_phase(phase);
26    if(!uvm_config_db#(virtual cmd_if)::get(this, "", "cmd_vif",
27      ~ cmd_vif))

```



```

13
14     function void build_phase(uvm_phase phase);
15         super.build_phase(phase);
16         if (!uvm_config_db#(virtual cmd_if)::get(this, "", "cmd_vif",
17             cmd_vif))
18             `uvm_fatal("NOVIF", {"virtual interface must be set for:
19                 ", get_full_name(), ".cmd_vif"});
20     endfunction: build_phase
21
22     // run phase
23     virtual task run_phase(uvm_phase phase);
24         fork
25             get_and_drive();
26         join
27     endtask : run_phase
28
29     // get_and_drive
30     virtual protected task get_and_drive();
31         forever begin
32             @ (posedge cmd_vif.clk);
33             cmd_vif.pl <= 2'b0;
34             seq_item_port.get_next_item(req);
35             $cast(rsp, req.clone());
36             rsp.set_id_info(req);
37
38             drive_transfer(rsp);
39
40             seq_item_port.item_done();
41             seq_item_port.put_response(rsp);
42         end
43     endtask : get_and_drive
44
45     // drive_transfer
46     virtual protected task drive_transfer (cmd_transfer trans);
47         //TO DO//
48         if (trans.transmit_delay > 0) begin
49             repeat(trans.transmit_delay) @ (posedge cmd_vif.clk);
50         end
51         cmd_vif.in <= trans.data;
52         case (trans.data_type)
53             TEXT: cmd_vif.pl[TEXT] <= 1'b1;
54             KEY: cmd_vif.pl[KEY] <= 1'b1;
55         endcase
56
57     endtask : drive_transfer
58
59 endclass : cmd_driver
60
61 class cmd_sequencer extends uvm_sequencer #(cmd_transfer);
62     `uvm_component_utils(cmd_sequencer)
63
64     function new (string name, uvm_component parent);
65         super.new(name, parent);

```

```

7  endfunction : new
8
9 endclass :cmd_sequencer
10
11 interface cmd_if;
12
13     // Control flags
14     bit has_checks = 1;
15     bit has_coverage = 1;
16
17     // Actual Signals
18     logic clk;
19     logic [1 : 0] pl;
20     logic [80-1 : 0] in;
21
22     // Coverage and assertions to be implemented here.
23     always @ (negedge clk)
24     begin
25
26         // Valid must not be X or Z
27         assert_cmd_valid_unknown:assert property (
28             disable iff (!has_checks)
29             !$isunknown(pl) )
30             else
31                 $error("ERR_CMD_VALID_XZ\n cmd_valid went to X or
32                     ↳ Z");
33
34         // Oe must not be X or Z
35         assert_input_unknown:assert property (
36             disable iff (!has_checks)
37             (pl[0]pl[1] -> (!$isunknown(in)) ) )
38             else
39                 $error("ERR_CMD_SIG\n Data went to X or Z during
40                     ↳ cmd_valid");
41
42     end
43
44 endinterface : cmd_if
45
46
47 class cmd_transfer extends uvm_sequence_item;
48
49     rand bit data_type;
50     rand logic [80-1 : 0] data;
51     rand int unsigned transmit_delay = 0;
52
53     constraint c_transmit_delay {
54         transmit_delay <= 100 ;
55     }
56
57     constraint c_transmit_data
58     {
59         solve data_type before data;
60         (data_type==TEXT)      -> data inside{ [0:(1<<64)-1] };
61         (data_type==KEY)       -> data inside{ [0:(1<<80)-1] };
62     }
63
64 endclass

```

```

17
18
19     `uvm_object_utils_begin(cmd_transfer)
20     `uvm_field_int      (data_type,           UVM_DEFAULT)
21         `uvm_field_int    (data,       UVM_DEFAULT)
22         `uvm_field_int    (transmit_delay,     UVM_DEFAULT)
23     `uvm_object_utils_end
24
25 // new - constructor
26 function new (string name = "cmd_transfer_inst");
27     super.new(name);
28 endfunction : new
29
30 endclass : cmd_transfer

1 class ctrl_agent extends uvm_agent;
2     ctrl_driver      driver;
3     ctrl_sequencer   sequencer;
4     ctrl_monitor     monitor;
5
6     // Provide implementations of virtual methods such as get_type_name
7     // and create
8     `uvm_component_utils(ctrl_agent)
9
10 // new - constructor
11 function new (string name, uvm_component parent);
12     super.new(name, parent);
13 endfunction : new
14
15 // build_phase
16 function void build_phase(uvm_phase phase);
17     super.build_phase(phase);
18     monitor = ctrl_monitor::type_id::create("monitor", this);
19
20     if(get_is_active() == UVM_ACTIVE) begin
21         sequencer = ctrl_sequencer::type_id::create("sequencer", this);
22         driver = ctrl_driver::type_id::create("driver", this);
23     end
24 endfunction : build_phase
25
26 // connect_phase
27 function void connect_phase(uvm_phase phase);
28     if(get_is_active() == UVM_ACTIVE) begin
29         driver.seq_item_port.connect(sequencer.seq_item_export);
30     end
31 endfunction : connect_phase
32
33 endclass : ctrl_agent

1 class ctrl_monitor extends uvm_monitor;
2
3     protected virtual ctrl_if ctrl_vif;
4     protected ctrl_packet trans_collected_reset;
5     protected ctrl_packet trans_collected_oe;

```

```

6     protected uvm_event reset;
7     bit coverage_enable = 1;
8
9     uvm_analysis_port#(ctrl_packet) item_collected_port;
10
11    `uvm_component_utils_begin(ctrl_monitor)
12      `uvm_field_int(coverage_enable, UVM_DEFAULT)
13    `uvm_component_utils_end
14
15    function new (string name, uvm_component parent);
16      super.new(name, parent);
17      cov_ctrl_trans = new();
18      cov_ctrl_trans.set_inst_name({get_full_name(),
19        → ".cov_ctrl_trans"});
20      trans_collected_reset = new();
21      trans_collected_oe = new();
22      item_collected_port = new("item_collected_port", this);
23    endfunction : new
24
25    function void build_phase(uvm_phase phase);
26      super.build_phase(phase);
27      if (!uvm_config_db#(virtual ctrl_if)::get(this, "", "ctrl_vif",
28        → ctrl_vif))
29        `uvm_fatal("NOVIF", {"virtual interface must be set for:
30          → ", get_full_name(), ".ctrl_vif"});
31      if (!uvm_config_db#(uvm_event)::get(null, "", "reset", reset))
32        `uvm_fatal("NOVIF", {"reset event must be set for:
33          → ", get_full_name(), ".reset"});
34    endfunction: build_phase
35
36
37    virtual task run_phase(uvm_phase phase);
38      `uvm_info(get_full_name(), "starting ctrl monitor", UVM_MEDIUM);
39      fork
40        collect_transactions();
41        detect_reset();
42      join
43    endtask : run_phase
44
45    virtual protected task collect_transactions();
46      forever begin
47        @(ctrl_vif.reset);
48        void'(this.begin_tr(trans_collected_reset));
49        trans_collected_reset.data = ctrl_vif.reset;
50        this.end_tr(trans_collected_reset);
51
52        `uvm_info(get_full_name(), $sformatf("packet collected
53          → :\n%s",
54            trans_collected_reset.sprint()), UVM_FULL)
55        if (coverage_enable)
56          perform_packet_coverage();
57        item_collected_port.write(trans_collected_reset);
58      end

```

```

54      endtask : collect_transactions
55
56      virtual protected task detect_reset();
57          fork
58              forever begin
59                  @(posedge ctrl_vif.reset == ACTIVE_RESET);
60                  reset.trigger();
61              end
62              forever begin
63                  @(negedge ctrl_vif.reset == ACTIVE_RESET);
64                  reset.reset();
65              end
66          join
67      endtask : detect_reset
68
69
70      covergroup cov_ctrl_trans;
71          option.per_instance = 1;
72          trans_reset : coverpoint trans_collected_reset.data;
73      endgroup : cov_ctrl_trans
74
75      virtual protected function void perform_packet_coverage();
76          cov_ctrl_trans.sample();
77      endfunction : perform_packet_coverage
78
79      virtual function void report_phase(uvm_phase phase);
80          `uvm_info(get_full_name(), $sformatf("Covergroup 'cov_ctrl_trans'",
81          " coverage: %2f",
82          cov_ctrl_trans.get_inst_coverage()), UVM_LOW)
83      endfunction
84
85  endclass : ctrl_monitor

1  class ctrl_driver extends uvm_driver #(ctrl_transfer);
2
3      // The virtual interface used to drive and view HDL signals.
4      protected virtual ctrl_if ctrl_vif;
5
6      // Provide implementations of virtual methods such as get_type_name
7      // and create
8      `uvm_component_utils(ctrl_driver)
9
10     // new - constructor
11     function new (string name, uvm_component parent);
12         super.new(name, parent);
13     endfunction : new
14
15     function void build_phase(uvm_phase phase);
16         super.build_phase(phase);
17         if(!uvm_config_db#(virtual ctrl_if)::get(this, "", "ctrl_vif",
18             ctrl_vif))
19             `uvm_fatal("NOVIF", {"virtual interface must be set for:
20                 ", get_full_name(), ".ctrl_vif"});
21     endfunction: build_phase

```

```

19
20    // run phase
21    virtual task run_phase(uvm_phase phase);
22        fork
23            get_and_drive();
24        join
25    endtask : run_phase
26
27    // get_and_drive
28    virtual protected task get_and_drive();
29        forever begin
30            seq_item_port.get_next_item(req);
31            $cast(rsp, req.clone());
32            rsp.set_id_info(req);
33
34            drive_transfer(rsp);
35
36            seq_item_port.item_done();
37        end
38    endtask : get_and_drive
39
40    // drive_transfer
41    virtual protected task drive_transfer (ctrl_transfer trans);
42        //TO DO//
43        if (trans.transmit_delay > 0) begin
44            repeat(trans.transmit_delay) @ (negedge ctrl_vif.clk);
45        end
46        toggle_signal(trans);
47
48        if (trans.transmit_hold > 0) begin
49            repeat(trans.transmit_hold) @ (negedge ctrl_vif.clk);
50            toggle_signal(trans);
51        end
52    endtask : drive_transfer
53
54
55    virtual protected task toggle_signal(ctrl_transfer trans);
56        if (trans.toggle) begin
57            if (trans.reset == ACTIVE_RESET) begin
58                ctrl_vif.reset <= ~ctrl_vif.reset;
59            end
60        end
61        else begin
62            ctrl_vif.reset <= trans.reset;
63        end
64    endtask : toggle_signal
65
66
67 endclass : ctrl_driver
1
2
3
4

```

```

5   function new (string name, uvm_component parent);
6     super.new(name, parent);
7   endfunction : new
8
9   endclass : ctrl_sequencer

10  interface ctrl_if;
11
12    // Control flags
13    bit has_checks = 1;
14    bit has_coverage = 1;
15
16    // Actual Signals
17    logic clk;
18    logic reset;
19
20    // Coverage and assertions to be implemented here.
21    always @(negedge clk)
22      begin
23
24        // Reset must not be X or Z
25        assert_reset_unknown:assert property (
26          disable iff(!has_checks)
27          !$isunknown(reset) )
28        else
29          $error("ERR_RESET_XZ\n Reset went to X or Z");
30      end
31
32
33      endinterface : ctrl_if

34  class out_agent extends uvm_agent;
35    out_monitor monitor;
36
37    // Provide implementations of virtual methods such as get_type_name
38    ↳ and create
39    `uvm_component_utils(out_agent)
40
41    // new - constructor
42    function new (string name, uvm_component parent);
43      super.new(name, parent);
44    endfunction : new
45
46    // build_phase
47    function void build_phase(uvm_phase phase);
48      super.build_phase(phase);
49      monitor = out_monitor::type_id::create("monitor", this);
50    endfunction : build_phase
51
52
53  endclass : out_agent

54  class out_monitor extends uvm_monitor;
55
56    protected virtual out_if out_vif;
57    protected out_packet trans_collected;

```

```

5      bit coverage_enable = 1;
6
7      uvm_analysis_port#(out_packet) item_collected_port;
8
9      `uvm_component_utils_begin(out_monitor)
10     `uvm_field_int(coverage_enable, UVM_DEFAULT)
11     `uvm_component_utils_end
12
13    function new (string name, uvm_component parent);
14        super.new(name, parent);
15        cov_out_trans = new();
16        cov_out_trans.set_inst_name({get_full_name(), ".cov_out_trans"});
17        trans_collected = new();
18        item_collected_port = new("item_collected_port", this);
19    endfunction : new
20
21    function void build_phase(uvm_phase phase);
22        super.build_phase(phase);
23        if (!uvm_config_db#(virtual out_if)::get(this, "", "out_vif",
24            → out_vif))
25            `uvm_fatal("NOVIF", {"virtual interface must be set for:
26                → ", get_full_name(), ".out_vif"});
27    endfunction: build_phase
28
29    virtual task run_phase(uvm_phase phase);
30        `uvm_info(get_full_name(), "starting out monitor", UVM_MEDIUM);
31        fork
32            collect_transactions();
33        join
34    endtask : run_phase
35
36    virtual protected task collect_transactions();
37        forever begin
38            @ (posedge out_vif.done);
39            @ (negedge out_vif.clk);
40            void'(this.begin_tr(trans_collected));
41            trans_collected.out = out_vif.out;
42            this.end_tr(trans_collected);
43
44            `uvm_info(get_full_name(), $sformatf("packet collected :\n%s",
45                trans_collected.sprint()), UVM_MEDIUM)
46            if (coverage_enable)
47                perform_packet_coverage();
48            item_collected_port.write(trans_collected);
49        end
50    endtask : collect_transactions
51
52    covergroup cov_out_trans;
53        option.per_instance = 1;
54        trans_cmd_out : coverpoint trans_collected.out;
55    endgroup : cov_out_trans

```

```

56     virtual protected function void perform_packet_coverage();
57         cov_out_trans.sample();
58     endfunction : perform_packet_coverage
59
60     virtual function void report_phase(uvm_phase phase);
61         `uvm_info(get_full_name(), $sformatf("Covergroup 'cov_out_trans'",
62             " coverage: %2f",
63             cov_out_trans.get_inst_coverage()), UVM_LOW)
64     endfunction
65
66 endclass : out_monitor
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2390
2391
2392
2393
2394
2395
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2488
2489
2490
2491
2492
2493
2494
2495
2495
2496
2497
2498
2499
2499
2500
250
```

**Listing 4.** Mediu de verificare în UVM.

#### Anexa 4. Componentele modulelor hardware

```

1 `define INITIAL_KEY 0
2 `define NR_ROUNDS 32
3
4 module present_encoder(clk,n_reset,pl,in_text,ciphertext,done);
5     input clk,
6         n_reset;
7     input [1:0] pl;
8     input [79:0]in_text;
9     output wire done;
10    output [63:0]ciphertext;
11
12    reg[63:0] word;
13    reg[79:0] key;
14    reg[5:0] counter;
15    reg      reseted;
16    reg[3:0]  S[0:15];
17
18    wire[63:0] x;
19    wire[63:0] s;
20    wire[63:0] p;
21
22    assign x = word ^ key[79:16];
23    assign s = {S[x[63:60]],
24                 S[x[59:56]],
25                 S[x[55:52]],
26                 S[x[51:48]],
27                 S[x[47:44]],
28                 S[x[43:40]],
29                 S[x[39:36]],
30                 S[x[35:32]],
31                 S[x[31:28]],
32                 S[x[27:24]],
33                 S[x[23:20]],
34                 S[x[19:16]],
35                 S[x[15:12]],
36                 S[x[11:8]],
37                 S[x[7:4]],
38                 S[x[3:0]]};
39
40    assign p =
41        {s[63],s[59],s[55],s[51],s[47],s[43],s[39],s[35],s[31],s[27],s[23],s[19],s[15],
42        s[11],s[7],s[3],s[1],s[0]};
43    assign done      = (counter == `NR_ROUNDS);
44    assign ciphertext = done ? ( word ^ key[79:16] ) : 64'hz;
45
46    always@ (posedge clk or negedge n_reset)

```

```

47      if(!n_reset)
48          begin
49              counter  <= 1;
50              word    <= 0;
51              reseted  <= 1;
52              key     <= `INITIAL_KEY;
53 //           S<=64'hc56b90ad3ef84712;
54             S[0]  <= 4'hc; S[1]  <= 4'h5; S[2]  <= 4'h6; S[3]  <= 4'hb;
55             S[4]  <= 4'h9; S[5]  <= 4'h0; S[6]  <= 4'ha; S[7]  <= 4'hd;
56             S[8]  <= 4'h3; S[9]  <= 4'he; S[10] <= 4'hf; S[11] <= 4'h8;
57             S[12] <= 4'h4; S[13] <= 4'h7; S[14] <= 4'h1; S[15] <= 4'h2;
58 //           $readmemh("smem.mem", S);
59         end
60     else
61         if(reseted done)
62             begin
63                 if(pl[1])
64                     begin
65                         key  <= in_text;
66                     end
67                 if(pl[0])
68                     begin
69                     reseted  <= 0;
70                     counter  <= 1;
71                     word    <= in_text[63:0];
72                 end
73             end
74         else
75             if(!done)
76                 begin
77                     counter  <= counter+1;
78                     key     <= {S[key[18:15]],
79                               key[14:0],key[79:39],
80                               key[38:34]^counter[4:0],
81                               key[33:19]
82                             };
83                     word    <= p;
84                 end
85     endmodule
86
87 `define INITIAL_KEY 0
88 `define NR_ROUNDS 32
89
90 module present_encoder(clk,n_reset,pl,in_text,ciphertext,done);
91     input clk,
92         n_reset;
93     input [1:0] pl;
94     input [79:0]in_text;
95     output wire done;
96     output [63:0]ciphertext;
97
98     reg[63:0] word;
99     reg[79:0] key;
100    reg[5:0]   counter;

```

```

15      reg      reseted;
16      reg[3:0]   S[0:15];
17
18      wire[63:0] x;
19      wire[63:0] s;
20      wire[63:0] p;
21
22      assign x = word ^ key[79:16];
23      assign s = {S[x[63:60]],
24                  S[x[59:56]],
25                  S[x[55:52]],
26                  S[x[51:48]],
27                  S[x[47:44]],
28                  S[x[43:40]],
29                  S[x[39:36]],
30                  S[x[35:32]],
31                  S[x[31:28]],
32                  S[x[27:24]],
33                  S[x[23:20]],
34                  S[x[19:16]],
35                  S[x[15:12]],
36                  S[x[11:8]],
37                  S[x[7:4]],
38                  S[x[3:0]]};
39
40      assign p =
41          {s[63],s[59],s[55],s[51],s[47],s[43],s[39],s[35],s[31],s[27],s[23],s[19],
42          s[62],s[58],s[54],s[50],s[46],s[42],s[38],s[34],s[30],s[26],s[22],
43          s[61],s[57],s[53],s[49],s[45],s[41],s[37],s[33],s[29],s[25],s[21],
44          s[5],s[1],
45          s[60],s[56],s[52],s[48],s[44],s[40],s[36],s[32],s[28],s[24],s[20],
46          s[4],s[0]};
47
48      assign done     = (counter == `NR_ROUNDS);
49      assign ciphertext = done ? ( word ^ key[79:16] ) : 64'hz;
50
51
52      always@(posedge clk or negedge n_reset)
53          if(!n_reset)
54              begin
55                  counter <= 1;
56                  word <= 0;
57                  reseted <= 1;
58                  key    <= `INITIAL_KEY;
59
60                  // S<=64'hc56b90ad3ef84712;
61                  S[0] <= 4'hc; S[1] <= 4'h5; S[2] <= 4'h6; S[3] <= 4'hb;
62                  S[4] <= 4'h9; S[5] <= 4'h0; S[6] <= 4'ha; S[7] <= 4'hd;
63                  S[8] <= 4'h3; S[9] <= 4'he; S[10] <= 4'hf; S[11] <= 4'h8;
64                  S[12] <= 4'h4; S[13] <= 4'h7; S[14] <= 4'h1; S[15] <= 4'h2;
65
66                  $readmemh("smem.mem", S);
67              end
68          else
69              if(reseted done)

```

```
62      begin
63          if(pl[1])
64          begin
65              key <= in_text;
66          end
67          if(pl[0])
68          begin
69              reseted <= 0;
70              counter <= 1;
71              word <= in_text[63:0];
72          end
73      end
74  else
75      if(!done)
76      begin
77          counter <= counter+1;
78          key <= {S[key[18:15]],
79                     key[14:0],key[79:39],
80                     key[38:34]^counter[4:0],
81                     key[33:19]
82                     };
83          word <= p;
84      end
85  endmodule
```

### Anexa 5. Rezultate experimentale

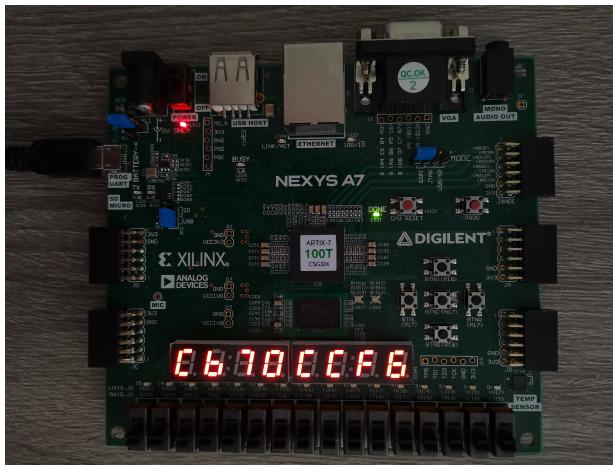


Figura A.1. Cheie de criptare stabilită prin Diffie-Hellman.

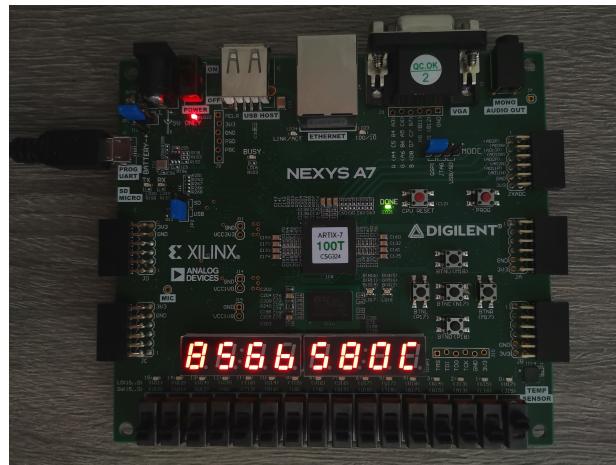


Figura A.2. Cheie de decriptare stabilită prin Diffie-Hellman

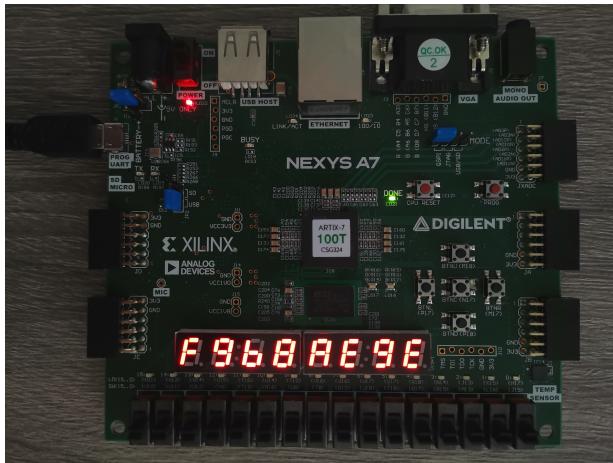


Figura A.3. Cheie actualizată după o criptare.

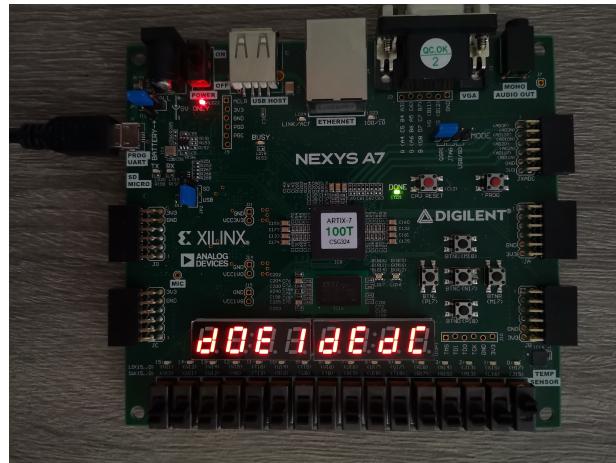


Figura A.4. Cheie actualizată după o decriptare.

plaintext	key	ciphertext
00000000 00000000	00000000 00000000 0000	5579C138 7B228445
00000000 00000000	FFFFFFFFFF FFFFFFFF FFFF	E72C46C0 F5945049
FFFFFFFF FFFFFFFF	00000000 00000000 0000	A112FFC7 2F68417B
FFFFFFFF FFFFFFFF	FFFFFFFFFF FFFFFFFF FFFF	3333DCD3 213210D2

Tabelul A.1. Test vectors for PRESENT with an 80-bit key are shown in hexadecimal notation.

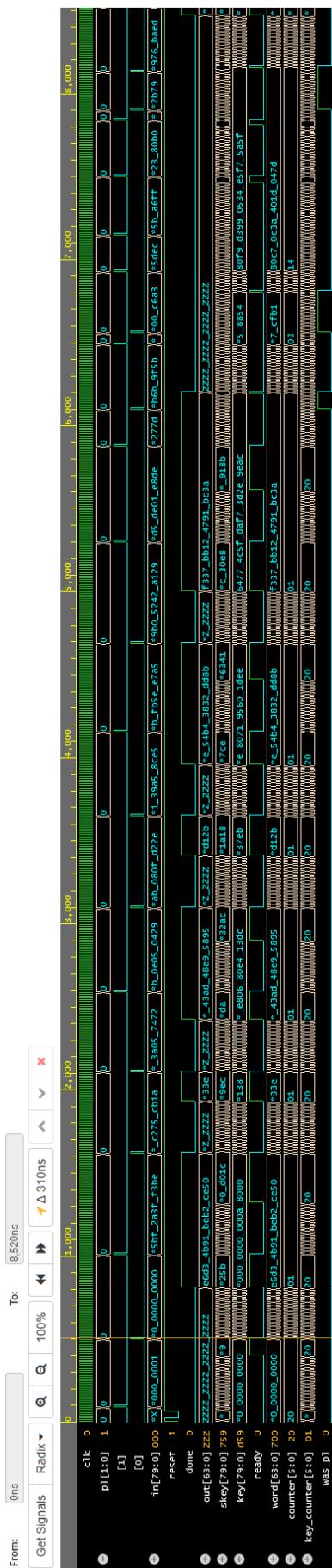


Figura A.5. Raport de simulare pentru decriptare.

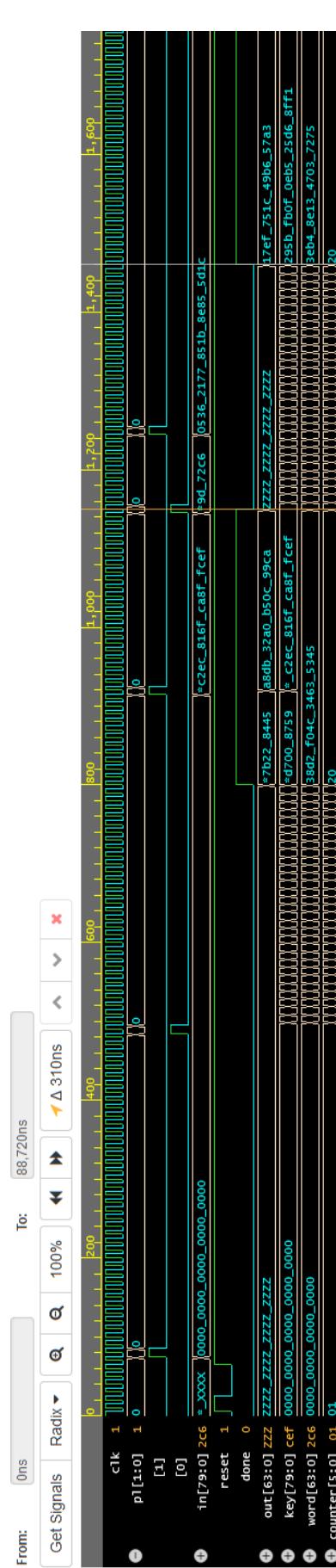


Figura A.6. Raport de simulare pentru criptare.

