VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2017 František Németh

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SIMULACE ŠIFROVACÍCH ALGORITMŮ POMOCÍ FPGA
SIMULATION OF CRYPTOGRAPHIC ALGORITHMS USING FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **František Németh**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. David Smékal**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: František Németh ***ID:*** 164766
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Simulace šifrovacích algoritmů pomocí FPGA

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s programovacím jazykem VHDL a vývojovým prostředím Xilinx Vivado. Analyzujte možnosti FPGA karet. Nastudujte a popište princip symetrických blokových šifer včetně jejich režimů.
Odsimulujte navržené šifrovací algoritmy pro platformu FPGA. Zaměřte se na šifrování provozu pomocí AES, implementaci různých operačních módů a práci s šifrovacím klíčem.

DOPORUČENÁ LITERATURA:

[1] STALLINGS, William. Cryptography and network security: principles and practice. 6. vyd. Upper Saddle River: Prentice Hall, 2013, 752 s. ISBN 01-333-5469-5.

[2] PINKER, Jiří, Martin POUPA. Číslicové systémy a jazyk VHDL. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-7300-198-5.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. David Smékal
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Bakalárska práca sa zaoberá so šifrovacím štandardom AES a následným návrhom šifrovacej a dešifrovacej komponenty vybratých operačných módoch v jazyku VHDL. V teoretickej časti práce je podrobnejšie rozoberatý fungovanie šifry a jednotlivých operačných módov. Teoretická časť ešte zahrňuje doseba stručný popis programovacieho jazyku VHDL, FPGA kariet a frameworku NetCOPE. Výstupom praktickej časti je stvorení návrh vo vývojovom prostredí Vivado od firmy Xilinx. Naprogramované sú šifrovacie a dešifrovacie komponenty pre operačné módy ECB, CBC, CTR, CFB. Výsledné simulácie a syntézne výsledky sú shrnuté v tabuľkách.

KLÍČOVÁ SLOVA

Kryptográfia, šifra, šifrovanie, dešifrovanie, šifrovací kľúč, symetrická kryptográfia, blokové šifry, AES, operačný mód, ECB, CBC, CFB, CTR, GCM, kryptogram, xorovanie, bit, bajt, inicializačný vektor, Galoisové pole, Substitucia bajtov, Rotácia riadkov, Zmiešanie stĺpcov, Pridanie iteračného kľuča, stavov, SBOX, rotácia, iteračný kľúč, Inverzná Substitúcia bajtov, Inverzná Rotácia riadkov, Inverzné Zmiešanie stĺpcov, VHDL, Xilinx, FPGA karty, komponent, COMBO-80G, NetCOPE, generic, buffer, FIFO buffer, Inverse SBOX, pomocný register, Vivado, simulácia, rozhranie, port, syntéza

ABSTRACT

Bachelor thesis is dealing with a cipher standard AES and with a design of encryption and decryption components for AES in special modes of operation. Programming language is VHDL. In theoretical part of thesis is a further descriptions of AES and behaviour of block cipher operation modes. Furthermore the brief description of VHDL, FPGA and NetCOPE framework is a piece of theoretical part as well. The practical part contains designs which are made in developing environment Vivado from Xilinx. Programmed modes of operation are ECB, CBC, CTR and CFB. Simulation outputs and synthesis results are summerized in tables.

KEYWORDS

Cryptography, cipher, encryption, decryption, cipher key, symetric cryptography, block cipher, AES, modes of operation, ECB, CBC, CFB, CTR, GCM, cryptogram, xor, bit, byte, initialization vector, Galois Field, Substitute bytes, Shift rows, Mix columns, Add Round Key, state, SBOX, rotate, iteration key, Inverse Substitute bytes, Inverse, Shift rows, Inverse Mix columns, VHDL, Xilinx, FPGA board, component, COMBO-80G, NetCOPE, generic, buffer, Vivado, simulation, interface, port, syntesis

NÉMETH, František. *Simulace šifrovacích algoritmů pomocí FPGA*. Brno, Rok, 69 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. David Smékal

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Simulace šifrovacích algoritmů pomocí FPGA“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. David Smékal, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

ÚVOD

Utajenie informácií v dnešnej dobe je nevyhnutnou záležitosťou. Vývojom internetovej komunikácie je stále potrebné utajiť čoraz väčšie množstvo informácií za kratšiu a kratšiu dobu. Vývojom sieťových kariet s FPGA, sa otvorila brána ku výraznému zrýchleniu šifrovania, ale aj iných hardwerovo akcelerovaných aplikácií. Táto bakalárská práca sa zaoberá s princípom symetrickej blokovej šifry AES a s návrhom šifry do programovacieho jazyka VHDL, pre testovanie chovania šifry na FPGA karte. Je popísaní princíp celého šifrovacieho a dešifrovacieho procesu pri 128 bitovej dĺžke kľúča a znázornené chovanie jednotlivých častí. Sú vysvetlené operačné módy ECB, CBC, CFB, CTR a GCM. Cieľom práce je oboznámenie sa s FPGA kartami a programovacím jazykom VHDL. Následne naprogramovanie syntetizovatelného návrhu šifrovania a dešifrovania AESu pre operačné módy ECB, CBC, CTR a CFB vo vývojovom prostredí Vivado od firmy Xilinx. Výsledný návrh odsimulovať, vysyntetizovať a naimplementovať na FPGA kartu COMBO-80G od spoločnosti NetCOPE.

Bakalárska práca je rozdelená na sedem základných kapitol. V prvej kapitole je predstavená stručná história kryptográfie a následné rozdelenie na asymetrickú a symetrickú kryptográfiu. Celá druhá kapitola sa zaoberá s blokovými symetrickými šiframi a operačnými módmi. Následná kapitola je venovaný ku podrobnejšiemu popisaniu šifrovacieho štandardu AES. Používané časti sú uvedené pre šifrovanie a dešifrovanie. Popis jazyka VHDL, FPGA kariet a frameworku NetCOPE je uvedené v kapitole štyri. Piatá kapitola slúži pre vysvetlenie návrhu pre jednotlivé operačné módy a popísanie účelu a chovania stvorených logických blokov a komponentov. V posledných kapitolách sú shrnuté výsledky simulácie a syntézi jednotlivých módov.

1 ÚVOD DO KRYPTOGRÁFIE

Kryptografia je veda, ktorá pomocou matematických operácií prevedie správu do takej neznámej podoby, ktorá je čitateľná iba zo špeciálnymi znalosťami. Umožňuje nám tak prenášať citlivé správy cez nezabezpečených prenosových kanálov napr. internet. Za správu v šifrovaní rozumieme informácie skladané z postupnosti symbolov. K utajeniu (šifrovanie) správy máme k dispozícií šifrovacie algoritmy (šifry). Šifrovací algoritmus je funkcia poskladaná na matematických operáciach. K tomu aby šifra fungovala správne, potrebuje jedineční parameter. To je šifrovací klúč. Šifrovanie kladie veľkú pozornosť na ochranu súkromia, bezpečnosť elektronických systémov a na bezpečnosť elektronickej komunikácie.

Začiatok kryptografie je stanovené na starovek, keď už sa používalo rovnaký vyjadrovací systém ako napr. písmo. Na začiatku boli používané mechanické šifry, ale potom kvalita šifrovania sa vyvíjalo rástom teoretických znalostí a technických pokrokov. Históriu šifrovania sa dá rozdeliť na dve časti. Prvá je klasická kryptográfia. Od začiatku 20. storočia sa začali vyvíjať sofistikovanejšie prístroje, ktoré umožnili zložitejšie postupy, a tak sa začala druhá časť, moderná kryptografia.

1.1 Klasická kryptográfia

Prvá časť trvala do poloviny 20. storočia. Charakteristické bolo pre začiatok tohto obdobia, že k šifrovaniu bolo potreba iba pero a papier. Hlavné klasické typy šifer boli tranzpozičné, substitučné a adaptívne. Tranzpozičné šifry spočívali v zamiešania poradia písmov ľubovoľného textu (napr. výraz „BUDEME ÚTOČIŤ“ zmodifikovalo sa na „EDEMUB ČITOTÚ“.

Susbtitučné šifry spočívali vo výmene symbolov na dopredu určený symbol. Medzi najstaršie substitučné šifry patrí Cézarová šifra. Pracuje na princípe, že každé písmeno v abecede je reprezentované s písmenom posunuté o tri miest(napr. správa „HELLO“ bude zašifrované ako „KHOOR“). V dnešnej dobe tento typ šifry už z bezpečnostných dôvodov nie je postačujúce.

Medzi ďalšie staršie kryptografické systémy, ktoré sú viac-menej pomerne známe patrí aj Scytale, Vigenerova a Vernamova šifra.

1.2 Moderná kryptográfia

Od druhej poloviny 20. storočia až k dnešnej dobe elektronická komunikácia sa rozvíjala veľmi rýchlo. K tomu, aby niekto nepovolený sa dostal k zachyteniu správ, ktorým nemá právo bolo relatívne ľahké. Práve kvôli tomu sa musela kryptografia zmo-

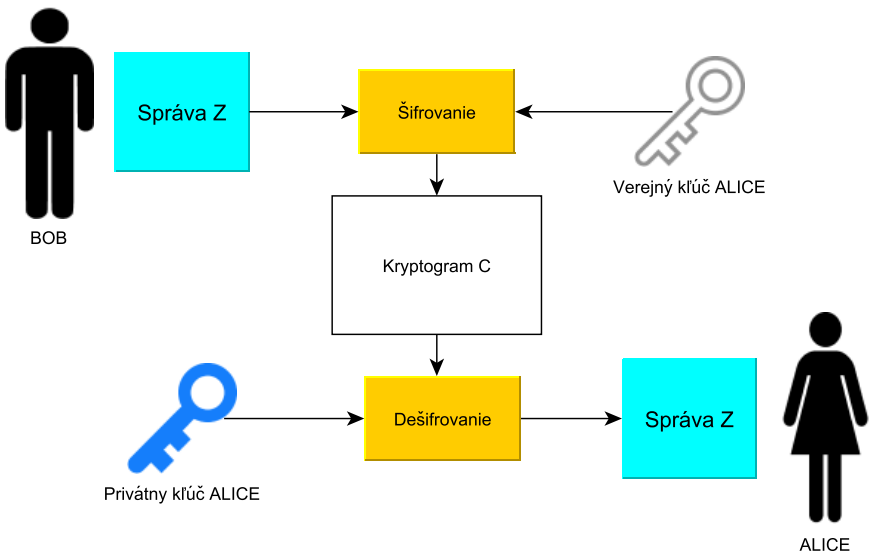
dernizovať a rozvíjať sa spolu s technikou. Dnešnú kryptografiu tak rozdelujeme na dve veľké skupiny. Na symetrickú a na asymetrickú kryptografiu.

1.2.1 Asymetrická kryptográfia

Kryptografia už má za sebou veľkú históriu vývoja, ale doposiaľ to bol vývoj iba symetrického kryptosystému. Asymetrická kryptografia sa začala vyvíjať začiatkom sedemdesiatych rokov minulého storočia, a veľmi rýchlo sa rozšírila. Dôvodom boli nasledujúce nedostatky internetovej komunikácie. Príjemca a odosielateľ nemal záruku, či prijaté informácie sú od pravého odosielateľa a zároveň ani jeden nemal istotu ohľadom toho, či neexistuje nejaká tretia strana, ktorá odpočúva komunikáciu medzi nimi. Riešením sa stala asymetrická kryptografia, ktorá zaručuje prenos dát a pravosť identity odosielateľa.

Asymetrické kryptosystémy pracujú s rozlišními kľúčami. Používa sa soukromný a verejný kľúč. Sú rozlišné a je medzi nimy matematická závislosť. V matematickej teórií sú stále nevyriešené problémy, na princípe ktorých sú postavené asymetrické kryptosystémy. Takým problémom je faktorizácia čísiel (RSA), problém diskrétneho logaritmu (Diffie-Hellman) a problém eliptického diskrétneho logaritmu (DSA). Bezpečnosť zasieľaných správ závisí na vyriešení týchto problémov. Pri šifrovaní odosielateľ zašifruje svoju správu pomocou verejného kľúča príjemcu a príjemca si to dešifruje pomocou svojho súkromného kľúča. Tento postup je znázornené na obrázku 1.1.

Výhodou asymetrických kryptosystémov je, že sú veľmi bezpečné. Ako všetky

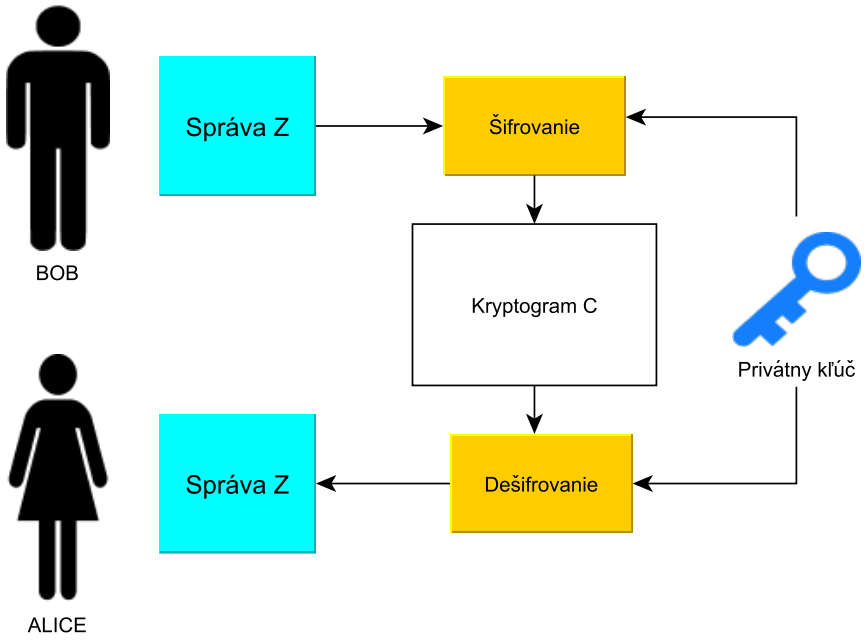


Obr. 1.1: Príklad asymetrického šifrovania

systémy majú svoje výhody a aj nevýhody. Nevýhodou je nízká rýchlosť kvôli zložitých matematických operácií. V praxi slúžia najmä na prenos symetrického kľúča medzi účastníkmi.

1.2.2 Symetrická kryptográfia

Účastníci pred šifrovaním alebo dešifrovaním sa dohodnú nad rovnakým kľúčom a ten istý kľúč používajú až do konca procesu. To znamená, že $K_E = K_D = K$, kde K_E je kľúč pre šifrovanie, K_D je kľúč pre dešifrovanie. Utajenie kľúča je veľmi dôležitá podmienka ohľadom bezpečnosti. Na obrázku 1.2 je znázornený princíp symetrického šifrovania.



Obr. 1.2: Príklad symetrického šifrovania

Symetrické kryptosystémy sa rozdelujú na dve skupiny. Na prúdovú šifru a na blokovú šifru.

Prúdová šifra šifruje dáta po jednotlivých bitoch čo môžeme vyjadriť podľa [1] nasledovne:

$$C_i = E(Z_i, K), \tag{1.1}$$

kde C_i je i-tý bit kryptogramu, Z_i je i-tý bit správy, K je kľúč a E je funkcia šifrovania. Napriek tomu, že v dnešnej počítačovej technike, ktorá je veľmi výkonná a rýchla, výhoda prúdových šifier je zanedbatelný kvôli jednoduchosti. Prúdové šifry sú menej odolné proti kryptoanalýze ako blokové šifry.

2 BLOKOVÉ SYMETRICKÉ ŠIFRY

S rýchlym vývojom telekomunikácie a internetu bolo potrebné utajiť čoraz väčšie a väčšie množstvo dát. Hlavnými podmienkami sa stali najmä bezpečnosť a rýchlosť celého procesu. Asymetrické kryptosystémy sa dali použiť pre šifrovanie, ale zahrňovali v sebe zložitejšie operácie a kvôli tomu boli veľmi pomalé pre tento účel. Prúdové symetrické šifry už boli rýchlejšie, ale vzhľadom na bezpečnosť neboli veľmi spoľahlivé. Napokon v 70-tých rokoch prišli prvé požiadavky od National Bureau of Standards (Národný Úrad Štandardov) na vývoj štandardného kryptosystému. Americká firma IBM a NSA (National Security Asociation) spoločne začali rozvoj blokových šifier. V roku 1975 prišla prvá publikácia s názvom DES (Data Encryption Standard) a v 1976 bola schválená ako celosvetový štandard. Kľúč mal dlhý 64 bitov z ktorých 8 bitov boli kontrolné a 56 efektívne. S vývojom infomatiky 64 bitový kľúč DESu sa stal nepostačujúcim k dostatočnej bezpečnosti.

Pomocou DES cracker (alebo Deep Crack) bolo možne prelomiť DES s útokom hrubou silou za necelý deň. Za náhradu by bolo možne použiť 3DES, ale tento algoritmus bol pomalý a stále používal iba bloky s dĺžkou 64 bitov. Tak v roku 1997 NIST (National Institute of Standards) vyhlásila konkurz na nový šifrovací algoritmus. Celý konkurz prešiel v troch častiach. Do poslednej časti sa dostalo iba 5 navrhovaných algoritmusov ako RC6, Serpent, Twofish, MARS a Rijndael. V roku 2000 bol vyhlásený ako víťazní algoritmus Rijndael. Ako medzinárodný štandard pre šifrovanie bola zvolená v roku 2001 a bola publikovaná ako Advanced Encryption Standard(AES), ktorý je zatiaľ hlavným zastupiteľom blokových symetrických šifier.

Hlavná charakteristika blokových symetrických šifier je na rozdiel od prúdových symetrických šifer, že nepracujú so samotnými bitmi ale s bitovými blokmi. Rozdeľujú danú správu na jednotlivé bloky, ktorých dĺžka n je dopredu definovaná. Typické hodnoty n sú 64 alebo 128 bitov.

Vo väčšine prípadov správy, ktoré je potrebné zašifrovať sú oveľa dlhšie ako dopredu definovaná dĺžka blokov. Aby bolo možne prevedenie šifrovania celej správy je potrebné používať tzv. operačné módy.

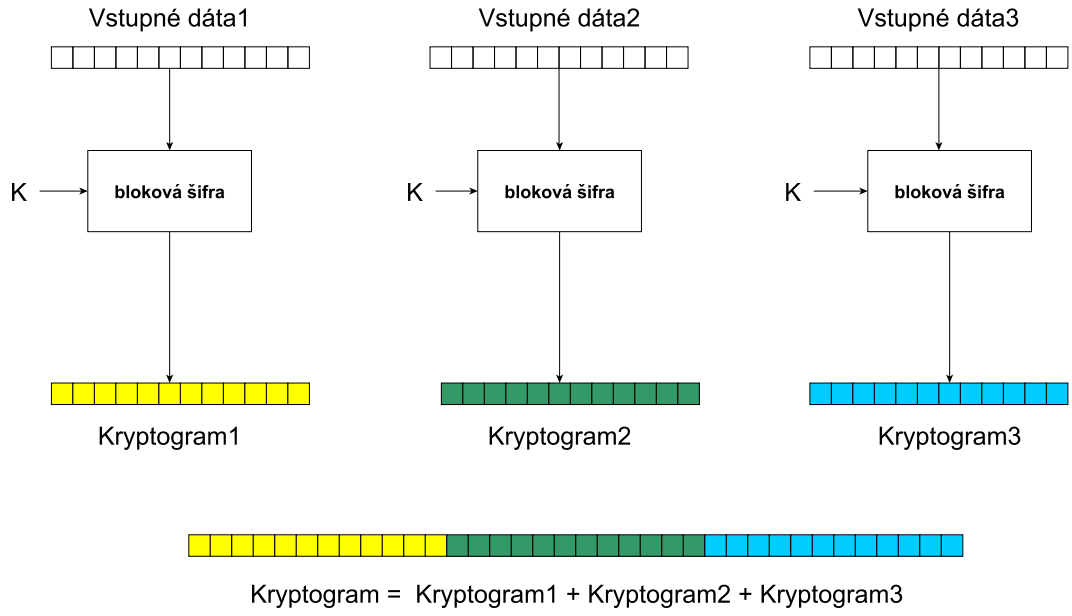
2.1 Operačné módy

Operačné módy slúžia pre flexibilné využitie blokových šifier. Jednotlivé módy pridávajú väčšiu dôvernosť šifry a autentičnosť k zašifrovaním správam. Podľa [1] a [4] sú popisáné chovania jednotlivých operačných módov.

2.1.1 Operačný mód Electronic Codebook (ECB)

Najjednoduchší operačný mód je Electronic Codebook(ECB). Pracuje na princípe, že danú správu po zarovnaní rozdelí na istý počet blokov. Každý blok je zašifrovaný s daným tajným kľúčom K. Výsledný kryptogram C nám bude udané z následného skladania zašifrovaných blokov. Šifrovanie pomocou ECB je znázornená na obrázku 2.1. Výhodou režimu je, že rýchlosť šifrovania a dešifrovania sa rovná rýchlosti šifrátora. S tým že v režimu ECB nenavázajú sa na seba bloky dát, umožňuje prístup ku každému bloku správy. To znamená, že nie potrebné dešifrovať celý kryptogram. Stačí si dešifrovať tú časť kryptogramu, ktorá je pre nás potrebná. Tento režim je vhodné pre šifrovania databáz.

Veľkou nevýhodou podľa [1] je, že každý blok zašifruje rovnakým kľúčom. V prí-

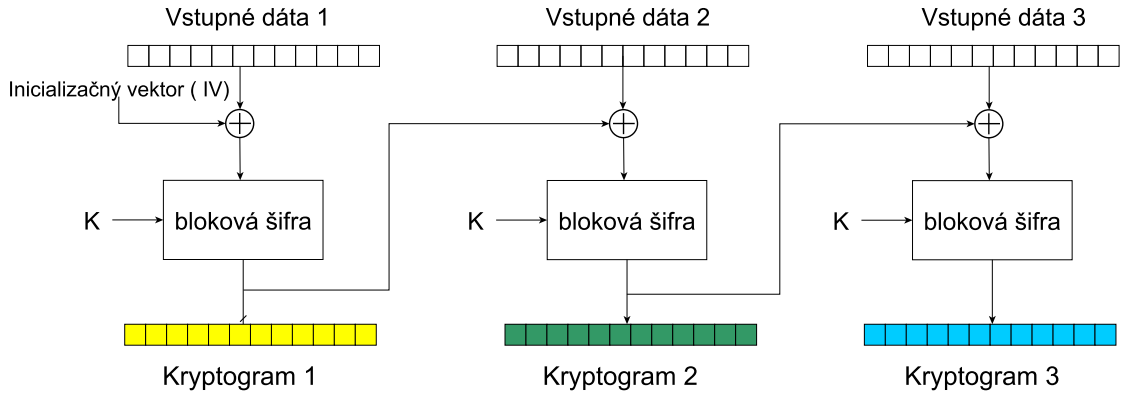


Obr. 2.1: Blokové schéma šifrovania *ECB*

pade, že medzi zašifrovanými blokmi je viac rovnakých blokov dát, tak vo výslednom bloku kryptogramu tie bloky budú rovnaké. Túto vlastnosť môže útočník využiť pri prelomení šifry a k získaniu tajného kľúča K. Zvyčajne štruktúry správ sú vybudované symetricky. Obsahujú redundancie ako napr. často sa opakujúcu postupnosť charakterov 0 a medzier. Preto iba vtedy sa využíva tento mód, keď nehrozí výskyt rovnakých blokov alebo pre šifrovanie jediného bloku. K tomu aby sa dalo vyhýbať tohto problému, bol navrhovaný mód Cipher Block Chaining (CBC).

2.1.2 Operačný mód Cipher Block Chaining (CBC)

CBC mód je už bezpečnejší, ako mód ECB, lebo šifrované bloky na rozdiel od ECB módu sú na seba závislé. Táto vlastnosť eliminuje výskyt rovnako zašifrovaných blokov. Pred šifrovaním sa logicky sčíta daľej iba xoruje, predošlí blok kryptogramu a šifrovaný blok správy. Po xorovania sa dôjde k zašifrovaniu získaného bloku dát, a celý proces sa začína odznova. Pre zašifrovanie prvého bloku, kde ešte nie je k dispozícií predošlí blok kryptogramu, je použitý inicializačný vektor. Tento vektor je unikátny, a musí byť zdieľaný medzi stranami ešte pred šifrovaním. Toto zdieľanie je možné urobiť v operačnom móde ECB. Vektor musí mať rovnakú dĺžku bitov ako majú bloky šifry. Na obrázku 2.2 je znázornené šifrovanie správy v režimu CBC. Pri dešifrovaní postup je obrátený. Blok kryptogramu je najprv dešifrovaný

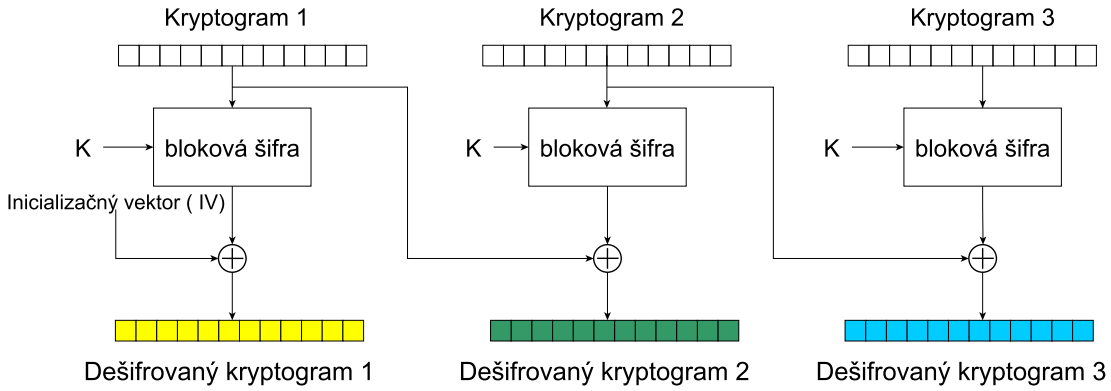


Obr. 2.2: Blokové schéma šifrovania v CBC

a následne xorovaný s predošlím blokom kryptogramu. Keď sa jedná o prvý blok kryptogramu rovnako sa dešifruje, ale následne sa xoruje s inicializačním vektorom, ktorý je zdieľaný pred šifrovacím a dešifrovacím procesom. Schéma na obrázku 2.3 popisuje princíp módy CBC pri dešifrovaní.

Hlavnou vlastnosťou operačného módu CBC je fakt, že hodnota zašifrovaného bloku P_i je závislá na predošlých zašifrovaných blokoch P_{i-1}, P_{i-2},...,P_1 a na hodnote inicializačného vektora. S tým je zvýšená entropia šifry, čo umožňuje obťaženie prípadných útokov. Výhodou CBC je jeho zvýšená bezpečnosť. Avšak veľkou nevýhodou je to, keď pri zmodifikovaní jedného bloku dát je potrebné znova spraviť šifrovanie od zmodifikovaného bloku.

Podľa [1] veľkou nevýhodu režimov ECB, CBC je šírenie chyby a preto sú veľmi citlivé na prenos správy. Chybný prenos jediného bitu môže urobiť zmenu hodnôt takmer v polovine preposlaných bitov bloku. Tieto režimy (ECB, CBC) sú používané vtedy, keď chybovosť prenosových kanálov je nízka. Takýto kanál môže byť metalický alebo optický. Pre prenosové kanály v ktorých je väčší pravdepodobnosť



Obr. 2.3: Blokové schéma dešifrovania v CBC

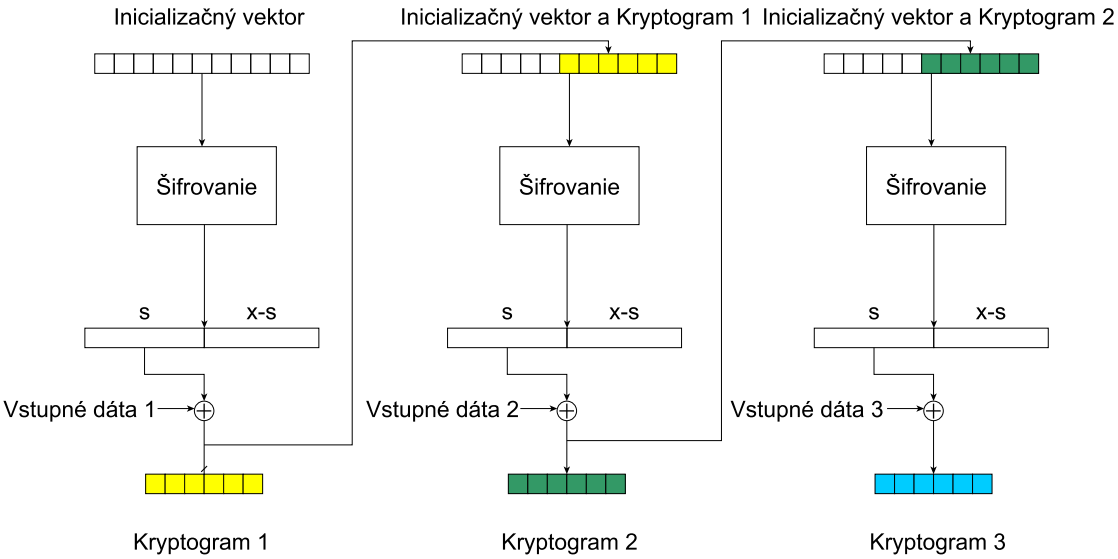
výskytu chyby sa používa režim Counter (CTR).

2.1.3 Operačný mód Cipher Feed Back (CFB)

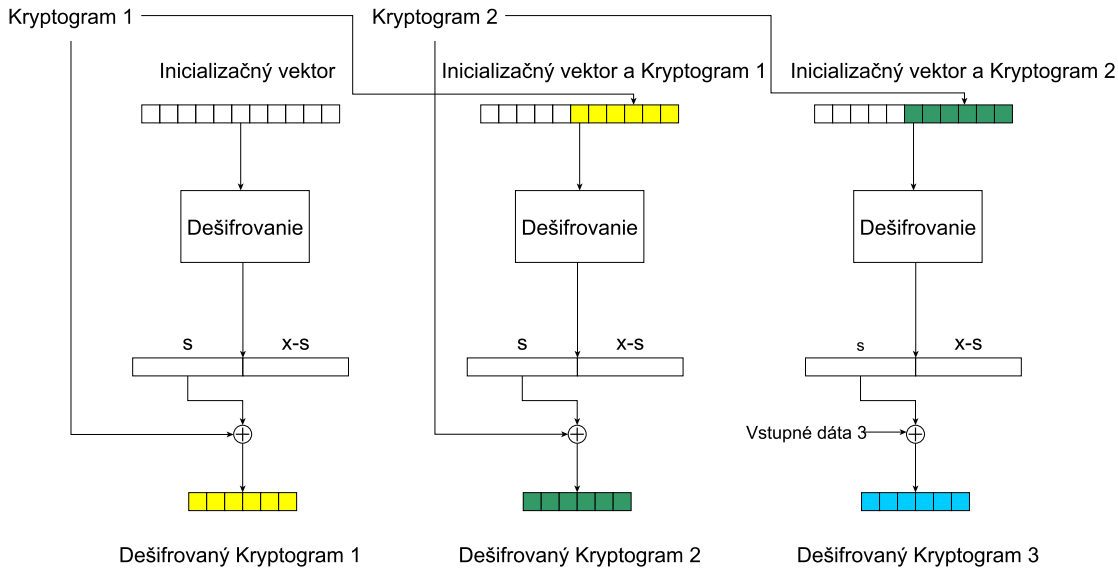
Vstupné dáta nemusia byť 128 bitov dlhé, ale keď sú menšie, musia deliť 128 bez zbytku.

Na začiatku šifrovania sa šifruje inicializačný vektor. Po šifrovaní zašifrované dáta sa rozdelujú na dve časti podľa dĺžky vstupných dát. Prvá zašifrovaná časť (od MSB do MSB − dlžka dát) sa xoruje so vstupnými dátami. Výsledkom xorovania je kryptogram prvého bloku dát. Obdobne ako v CBC, aj tu sú vstupy šifrátora závislé na predchádzajúcich výstupoch. Avšak AES pracuje so 128 bitovými vstupnými blokmi. Keď dlžka vstupných dát je menšia, nasledujúci inicializačný vektor sa upraví. Rotujú sa prvky inicializačného vektora do ľava toľko krát kolko bitov maju vstupné dáta. Rotácia je urobená tak, že na vzniknuté volné miestá sú napísané log. 0. Na miestá vzniknutý log. 0 sú pridané bity kryptogramu predchádzajúcich dát. Bloková schéma šifrovania je na obrázku 2.4. Parameter s je dĺžka vstupného bloku dát a x je potrebná dĺžka vstupu pre blokovú šifru. V AESu je pevne 128.

Pri dešifrovania je potrebné upraviť každý inicializačný vektor podľa potreby. Vstupné dáta znová vstupujú do komponenty šifrovania, ale teraz už nie sú závislé na predchádzajúcich dátoch. Chovanie módy pri dešifrovania je paralelizovateľná. Blokové schéma dešifrovania je na obrázku 2.5.



Obr. 2.4: Bloková schéma šifrovania v CFB



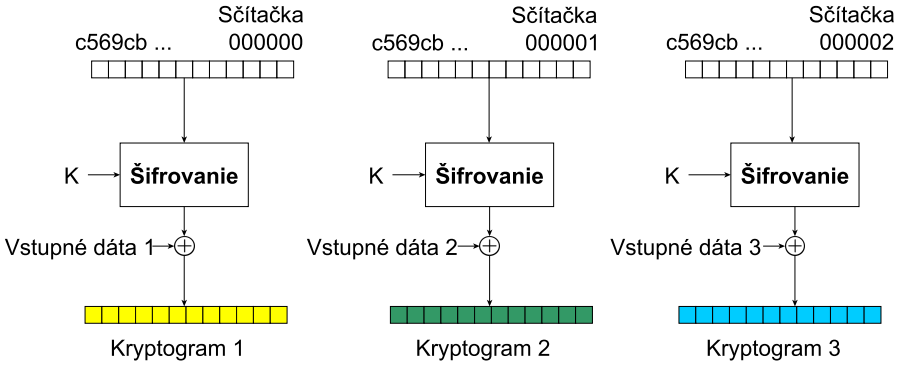
Obr. 2.5: Bloková schéma šifrovania v CFB

Operačný mód CFB má rovnaké výhody a nevýhody ako CBC. Zo strani dôverivosti je vylepšená, ale je výrazne najpomalšia medzi vybratými operačnými módmi.

2.1.4 Operačný mód Counter (CTR)

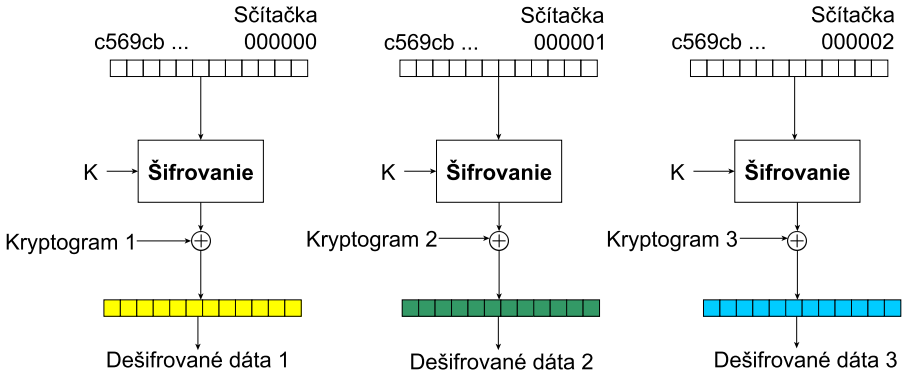
V operačnom móde Counter nie sú spojené vstupy a výstup blokov. Každý blok je zašifrovaný a dešifrovaný zvlášť. Na začiatku celého procesu je obdobne generovaný

nejaký inicializačný vektor IV ako u operačnom móde CBC. Posledných X bitov IV, ktoré sú zvolené pracujú ako sčítačka. Po každom šifrovaný alebo dešifrovaní hodnota sčitačky sa zvyšuje o jedno. CTR pracuje na princípe, že IV je zašifrovaný a následne je xorovaný buď s vstupným blokom dát, keď sa jedná o šifrovanie, alebo pri dešifrovaní s kryptogramom. Inicializačný vektor po dešifrovania ukazuje kolkáty blok celej správy je dešifrovaný kryptogram. Na obrázku 2.6 je uvedené blokové schéma operačnej módy Counter pre šifrovanie a na obrázku 2.7 pre dešifrovanie.



Obr. 2.6: Blokové schéma šifrovania v CTR

Významnou výhodou CTR módu je, že šifrovanie a dešifrovanie je možne spraviť paralelne a pri dešifrovaní nie je potrebné používať dešifrovací komponent. Náročnosť celého systému sa sníži. Nevýhoda je podľa [1] možnosť útoku na autentičnosť správy.

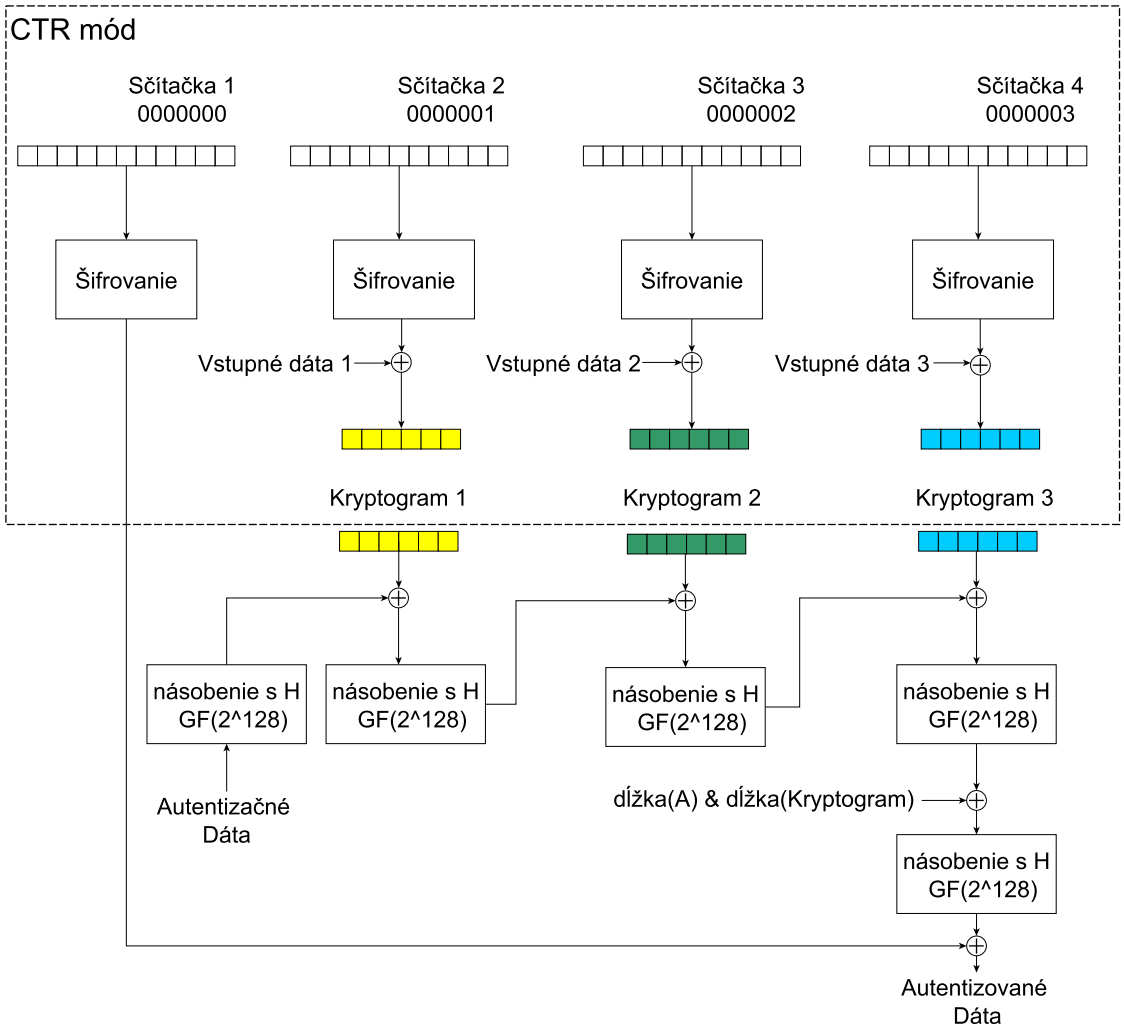


Obr. 2.7: Blokové schéma dešifrovanie v CTR

2.1.5 Operačný mód Galois Counter Mode (GCM)

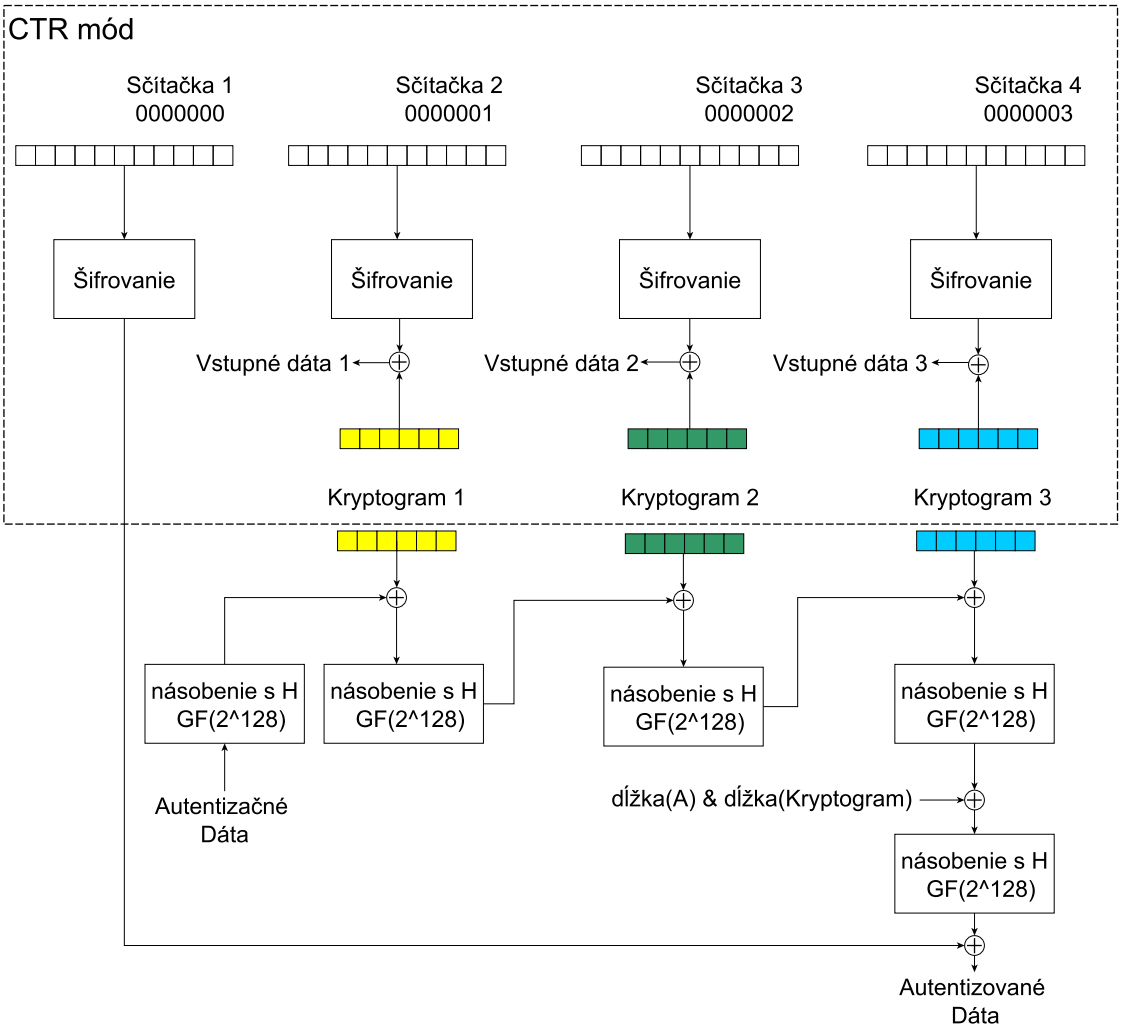
Operačný mód GCM zaručuje utajenie a autentičnosť zašifrovaných dát. Ako je znázornené v [3] pre samotné šifrovanie a dešifrovanie používa CTR mód. Autentizácia

je založená na násobenia v Galoisovom poli (2^{128}). Autentizačné dáta (napr. ip adresy), nie sú šifrované a násobujú sa s kľúčom H. Tento kľúč vzikne zašifrovaním nulového bloku dát so šifrovacím kľúčom. Po násobení sa xoruju násobené autentizačné dáta s patričním kryptogramom. Tento krok sa opakuje kým sa nexoruje posledný blok kryptogramu. Výsledok je vynásobená s H. Potom prixoruje sa dlžka autentizačných dát a kryptogramu. Ešte raz sa násobujú dáta. Posledným krokom je prixorovanie kľúča H. Pre lepšie porozumenie na obrázku 2.8 je znázornená blokova schéma šifrovania.



Obr. 2.8: Bloková schéma šifrovania v GCM

Pred začiatkom dešifrovania sa vypočíta hodnota autentizovaného dáta, aby bola overená validita zachytených dát. Keď vypočítaná hodnota sa zhoduje so získanými, začne sa dešifrovanie jednotlivých kryptogramov. V opačnom prípade dáta sú zahodené. Na obrázku 2.9 je bloková schéma dešifrovania.



Obr. 2.9: Bloková schéma dešifrovania v GCM

3 POPIS ŠIFROVACIEHO ŠTANDARDU AES

Advanced Encryption Standard (AES) [2] je medzištátným štandardom pre šifrovanie dát. Štandard AES ako šifrovací algoritmus používa Rijndael. Tvorcovia algoritmu sú dvaja vedci z Belgicka Joan Rijmen a Vincent Daemen. Rijandel je šifrovací algoritmus pre symetrické blokové šifry. Takže používa dopredu nadefinované bloky. Algoritmus bol stvorený tak, aby bolo možne zvoliť bloky vstupných dáta a kľúča rozlišné medzi 128 a 256 bitom. Podmienkou bolo to, že zvolená dĺžka bloku muselo byť deliteľné s 32. V štandarde AES je jednoznačne zvolená dĺžka bloku 128 bitov. Hlavným charakterom symetrických šifier je jediný tajný kľúč K. Tento tajný kľúč v štandarde AES má dopredu nadefinované dĺžky kľúča ako 128, 192 a 256 bitov.

3.1 Šifrovanie

Algoritmus sa začíná s inicializáciou. Vstupný blok dáta je xorované s vstupným kľúčom. Následne sú prevedené iterácie deväť krát. Jedna iterácia sa skladá zo štyroch častí:

- Substitúcia bajtov (Substitute Bytes)
- Rotácia riadkov (Shift Rows)
- Zmiešanie stĺpcov (Mix Columns)
- Pridanie iteračného kľuča (Add Round Key)

AES pracuje s dvojrozmerným blokom dát (maticou). Bitový tok vstupného dáta uloží po inicializácií do stavovej matice vertikálne. Rozmer stavovej matice je 4 krát 4 bajtov (4 krát 4 stavov). AES je kaskádová bloková šifra, čo znamená že jednotlivé časti sú na seba závislé.

3.1.1 Substitúcia bajtov

V tejto časti je prevedená jednoduchá substitúcia stavov rešp. bajtov stavovej matice. K pomoci substitúcie je dopredu nadefinovaná špeciálna substitučná tabuľka (3.1). Táto tabuľka je veľmi jedinečná. Má rozmer 16 krát 16 bajtov. Počet bajtov tabuľky je 256. Každý bajt sa nachádza v ňom práve jeden krát. Bola navrhovaná s tvorcami veľmi dôsledne tak, aby odolala k všetkým vtedy známym útokom. Je napr. veľmi neliniárna. Veľká výhoda tabuľky spočíva vtom, že sa dá jednoducho nájsť substitučný prvok daného bajtu. Daný bajt pri hexadecimálnom vyjadrenia sa skladá z dvoch prvkov. Prvý a druhý prvok môže udávať hodnotu v decimálnom

tvaru od 0 až 15. Hodnota, ktorú udáva prvá časť rozkladaného bajtu ukazuje na riadky (osa x) a druhá časť na stĺpce (osa y). Výstupom substitíce bajtov je vstupom rotácie riadkov.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	b7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1b	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	D1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Tab. 3.1: Substitučná tabuľka

3.1.2 Rotácia riadkov

Tu sa robí permutácia prvkov medzi jednotlivými stĺpcami matice. Na vstupu je výstup predošlej časti. Princípom tejto časti je rotácia jednotlivých riadkov matice. Prvý riadok ostáva rovnaká. V druhom riadku sa prevedie rotácia bajtov o jedno doľava. V treťom riadku sa rotuje o dva bajty doľava a v poslednom riadku sa rotuje o tri bajti doľava. S tým, že stavovú maticu skladáme zo stĺpcov vo výslednom matici každý stĺpec obdržal nejaký iný bajt (stav) od ostatných stĺpcov.

3.1.3 Zmiešanie stĺpcov

Táto časť je najnáročnejšia v celom algoritmu. Ide o násobenie matice s vektorom. Matica je tzv. mixovacia matica, ktorá je dopredu definovaná s tvorcami šifrovacieho algoritmu. Princíp procesu spočíva vtom, že sa postupne vynásobí stĺpce stavovej matice s mixovacou maticou(tab. 3.2). Násobené bajty sú navzájom xorované. Po skončení zmiešania posledného stĺpca je získané to, že každý bajt výslednej stavovej matice je závislá na ostatné bajty, s ktorými spolu tvoria stĺpec.

Náročnosť celého spočíva v násobení prvkov. Je potrebné pri násobenia dodržať

$$\begin{vmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{vmatrix}$$

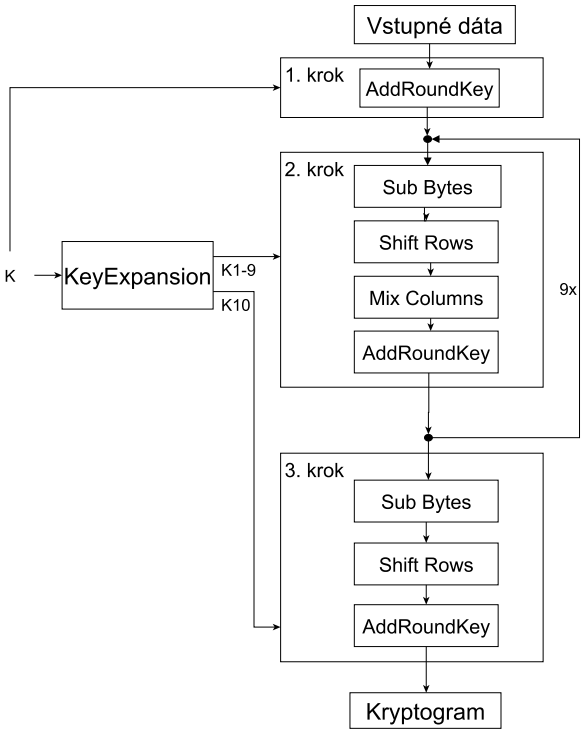
Tab. 3.2: Mixovacia matica

princíp toho, že vynásobený prvok musí mať dĺžku 8 bitov, vtedy musí byť bajt. Pri násobení s dvoma a s troma môžu sa nastať prípady, keď výslední násobok bude 9 bitov dlhý. Ku eliminovaniu tomuto problému, násobenie a aj sčítanie je zrealizované v Galoisovom poli (2^8). Násobuje sa podľa rovnice 3.1 z [1]. Rovnica je uvedená v polynomickom reprezentácií.

$$p_k(x) = [p_i(x) * p_j(x)] \ mod \ r(x) \tag{3.1}$$

3.1.4 Sčítanie iteračným kľúčom

Vstupné dáta sú xorované s príslušným iteračným kľúčom. Všetky iteračné kľúče sú vygenerované v komponente Expanzia iteračných kľúčov. Tento popis je napísané v sekci 3.2. Blokové schéma na obrázku 3.1 znázorňuje chovanie algoritmu AES.



Obr. 3.1: Blokové schéma šifrovania

Po deviatej iterácie sa prevedie ešte posledný krok algoritmu. Používané časti sú Substitúcia bajtov, Rotácia riadkov a Sčítanie s iteračným kľúčom. Výstupom z časti Sčítania s iteračným kľúčom je kryptogram vstupného bloku dát.

3.2 Expanzia iteračných kľúčov(Key Expansion)

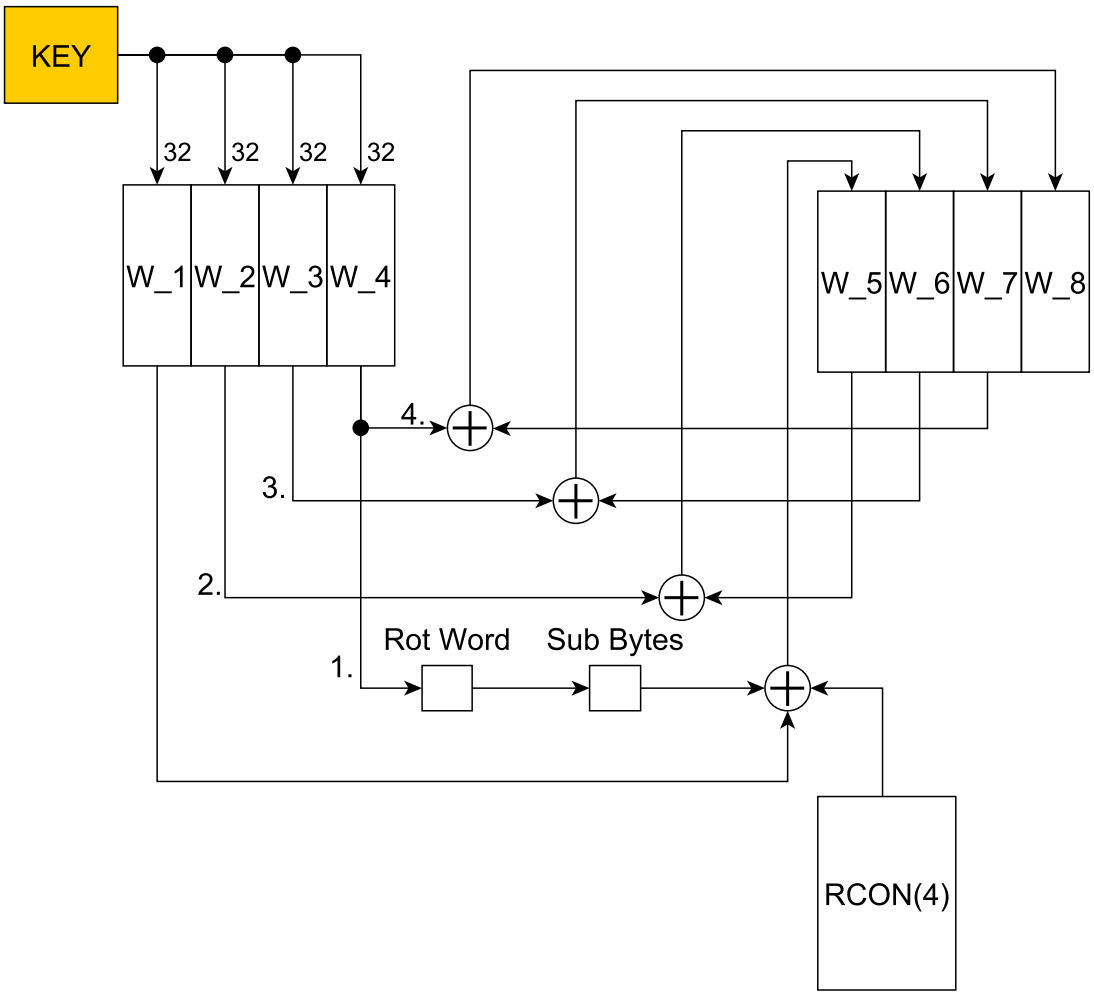
V časti Key Expansion sa generujú všetky iteračné kľúče z kľúča K. Ako v iteráciach aj tu sa pracuje so stavovými maticami. Tajný kľúč K je ukladaný vertikálne do stĺpcov stavovej matice. Tak sa získa kľúč K_0, s ktorým sa bude naďalej pracovať. Keďže dĺžka K je 128 bitov, podľa [2] potrebný počet vygenerovaných iteračních kľúčov je desať. Tieto kľúče sa generujú postupne po stĺpcoch na základe K_0. Pre ukladanie a generovanie kľúčov sa používá tzv. matica kľúčov. Rozmery má 44 stĺpcov a 4 riadkov. Stĺpce matice kľúčov sú označené ako W_i. Do prvých štyroch sĺpcov je uložený K_0. Stĺpce pre hodnoty i od 5 až do 44 sú výhradené pre jednotlivé iteračné kľúče. Tieto kľúče sú odvodené najprv z K_0 a nasledovne z predošlého kľúča. Pracovní postup sa rozkladá na dve hlavné podmienky. Prvá podmienka je, keď index stĺpca i $mod\ 4 \neq 0$ a druhá podmienka je, keď i $mod\ 4 = 0$. Pri prvej podmienke platý vzorec 3.2 zo [1]

$$\mathrm{W}_i = \mathrm{W}_{i-1} + \mathrm{W}_{i-4}. \tag{3.2}$$

Sčíta sa pomocou logického sčítania. Pracovný postup pri druhej podmienke je zložitejšia. Najprv sa rotuje stĺpec W_{i-1}. Potom pomocou Substitučnej tabuľky (viz Obr. 3.1) sa urobí substitúcia prvkov stĺpca. V poslednom kroku sa xorujú stĺpce W_{i-4} a W_{i-1} s patričným stĺpcom rundovej konštanty(RCON, tab. 3.3). Na obrázku 3.2 je bloková schéma vygenerovania jedného iteračného kľúča. Poradie operácií je znázornené.

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Tab. 3.3: Rundovná konštanta (RCON)



Obr. 3.2: Bloková schéma vygenerovania jedného iteračného kľúča

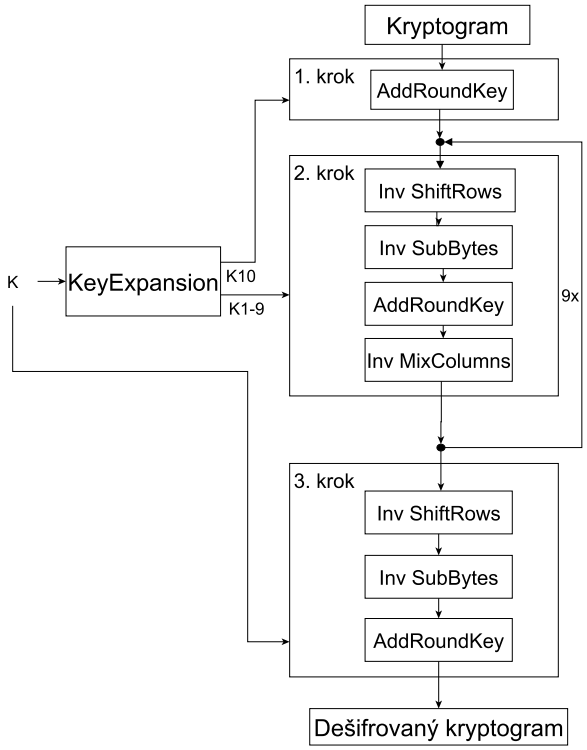
3.3 Dešifrovanie

Dešifrovanie je opačným postupom šifrovania. Jediný rozdiel medzi šifrovaním a dešifrovaním je postup vygenerovania iteračných kľúčov. Pred dešifrovaním je potrebné dopredu vygenerovať všetky iteračné kľúče, len potom sa môže začínať dešifrovanie s inicializáciou. Vygenerovanie iteračných kľúčov sa robí rovnako ako u šifrovania (v sekci 3.2). Pri inicializácií sa xoruje desiatý iteračný kľúč so vstupným blokom kryptogramu. Nasledovaných deväť iterácií sa skladá z častí,

- Inverzná Substitúcia bajtov (Inverse Substitute Bytes)
- Inverzná Rotácia riadkov (Inverse Shift Rows)
- Pridanie iteračného kľuča (Add Round Key).
- Inverzné Zmiešanie stĺpcov (Inverse Mix Columns)

Výstup inicializácie je vstupom časti InvShift Rows, potom nasleduje Inverse SubBytes, Add Round Key s k_i, kde $i = 9, \ldots, 1$ a Inverse Mix Columns.

Z poslednej iterácie je vynechané Inverse Mix Columns. Výstupom posledého xorovania, kde sa xoruje s kľúčom K je dešifrovaný blok kryptogramu. Bloké schéma dešifrovania je na obrázku 3.3.



Obr. 3.3: Blokové schéma dešifrovania

Ako bolo vyššie písané, dešifrovanie má opačný charakter voči šifrovania. Inv Sub Bytes (Inverzná substitúcia bajtov) pracuje na rovnakom princípe ako pri šifrovania, iba s inverznou Substitučnou tabuľkou (tab. 3.5).

Pri Inv Shift Rows v jednotlivých riadkoch sa posúvajú bajty (stavy) doprava. Inv Mix Columns násobuje s inverznou mixovaciou maticou 3.4. Podrobnejší rozbor

14	11	13	9
9	14	11	13
13	9	14	11
11	13	9	14

Tab. 3.4: Inverzná Mixovacia matica

Inv Mix Columns je v sekci 5.2.6.

52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Tab. 3.5: Inverzná substitučná tabuľka

4 JAZYK VHDL A FPGA KARTY

Pre úspešný návrh systémov a jednoduchú implementáciu na fyzické zariadenie, bolo potreba rozvinúť programovací jazyk, ktorý by navrhoval špeciálny popis a návrh číslicových obvodov a systémov. Pre tieto účely bol vyvinutý jazyk VHDL.

4.1 Jazyk VHDL

Jazyk VHDL (Very High Speed Integrated Circuit Hardware Description Language) [5] je jeden z dvoch najpoužívanejších programovacích jazykov pre popis hardwerov. V roku 1987 bol definovaný štandardom IEEE 1076 (v roku 1993 bola revidovaná a v 1997 rozšírená). VHDL slúži pre vývoj systémov vysokou spoľahlivosťou. Je dobre čitateľný a má symetrickú štruktúru. Používa oveľa viac kľúčových slov ako ostatné jazyky. Programovací jazyk VHDL v tom zmysle slova výrazne sa líši od programovacých jazykov ako Java, C alebo Pascal. Je to viac menej niaký prístroj pomocou sa opisujú digitálne obvody, systémy. Používa sa najmä na simulovanie a syntetizovanie logických obvodov.

Pri programovania hociakého obvodu, je potrebné nadefinovať jeho rozhranie a samotné chovanie, arcitektúra obvodu. Bez toho naprogramovaný kód nie je možné syntetizovať. Podľa [5] pre popísanie vstupnej a výstupnej rozhrania slúži entita. Signálmi rozhrania sú porty. Nadefinujú smer vstupných a výstupných signálov z obvodu. Je možné v tejto časti nadefinovať generické parametry.

```
entity názov_entity is
  generic(
          názov_generického_parametru : typ_parametru := hodnota;
          )
  port (
          názov_portu: smer_portu typ_portu
        );
end entity;
```

Výpis 4.1: Štruktúra entity

Pre popis chovania obvodu slúži architektúra. Architektúra je návrhová jednotka závislá na entite. Nadefinuje vlastné chovanie obvodu. Popisuje vzťahy medzi jednotlivými portami. Každá entita musí mať aspoň jednu architektúru. Keď má viac architektúr, jednotlivé architektúry sú pomenované inak. VHDL má bohaté vyjadrovacie schopnosti, preto je možné jednu entitu napísať podľa rôznych štýlov popisu.

```
architecture názov_architektúry of názov_entity
  deklarácia konštant, typov, signálov atď.;
begin
  príkazy;
end názov_architektúry;
```

Výpis 4.2: Štruktúra architektúry

Ďalšiou možnosťou jazyku VHDL je naprogramovanie testovacieho komponentu (testbench). Slúži pre otestovania funkčnosti kódu a ku eliminovaniu nastávajúcich chýb.

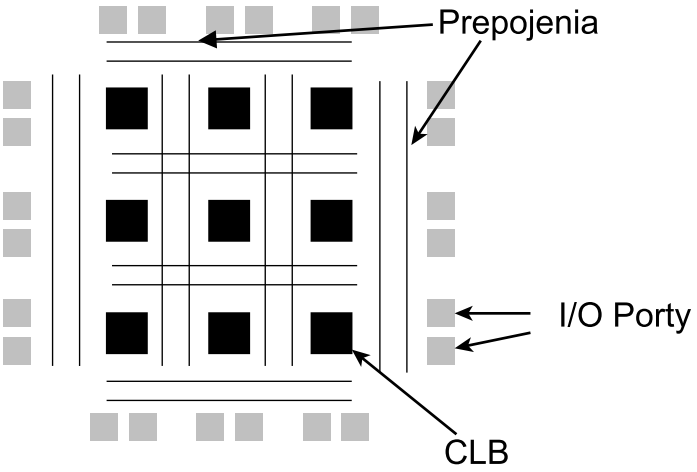
Všetky napísané príkazy, procesy, procedúry atď. ktoré sú napísané v architektúre sú realizované paralelne. Je to veľkou výhodou pri implementácií do FPGA (Field Programmable Gate Array).

4.2 FPGA karty

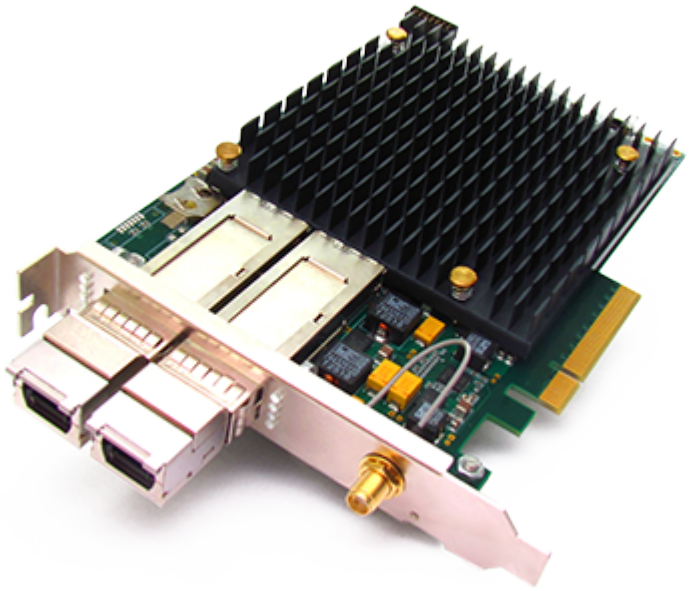
FPGA je programovatelné hradlové pole. Skladá sa z 2D matice programovateľných logických blokov (CLB), ktoré sú navzájom spojené s nakonfigurovatelnými prepojeniami [7]. Práve kvôli programovateľnosti sa uplatňujú čoraz v širšej škále aplikácií. FPGA nie je potrebné naprogramovať počas výroby. Logické bloky sú preprogramovatelné a tak umožnujú hocikedy podľa potreby zmeniť chovanie systému. Sú flexibilné a ľahko sa navrhuje na ne. Druhou veľkou výhodou majú možnosť paralelizácie. To znamená, že je možné si nahrať na kartu viac návrhov, ktoré môžu, ale nemusia byť spojené na seba. Tie jednotlivé návrhy sú schopné splňiť svoje úkoly navzájom bez toho, aby jeden na druhého čakal. Pri náročnom počítaní, keď je možné rozdeliť počítanie na jednotlivé časti, tak tie časti sú vypočítané paralelne. S týmto sa zvýši rýchlosť celého výpočtu.

Na obrázku 4.1 je znázornená zjednodušená štruktúra obvodu. Logické bloky CLB (Configurable Logic Block) ešte sú rozdelené na menšie logické elementy, ako napr. SLICE. SLICE môže obsahovať LUTy (Look Up Table), RAM , registry, multiplexory a pomocné logiky. LUTy sú zvyčajne 16 bitové pamäte, ktoré sú schopné realizovať všetky logické funkcie štyroch vstupov [7]. Programovateľná matica je obklopená so vstupnými a výstupným porty. U praktickom využitia návrhu bude používaná FPGA karta COMBO-80G a pomocný framework od firmy NetCope.

Karta podporuje sieťové rozhranie 10 Gbits/s Ethernet. Má dve fyzické rozhranie (konektory) QSFP+, ktoré umožnujú podporovať až 4×10G Ethernet. Používaným čipom je výkonný Virtex-7 od firmy XILINX. Samotná karta je na obrázku 4.2



Obr. 4.1: Štruktúra FPGA obvodu

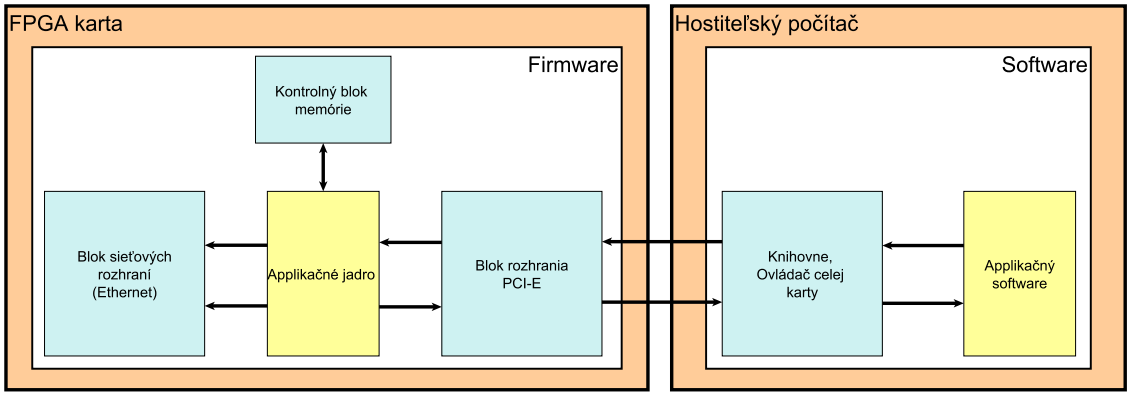


Obr. 4.2: Karta COMBO-80G od spoločnosti NetCope

4.3 Framework NetCOPE

Framework je možné chápať ako fixný program, ktorý interne riadi, alebo kontroluje rôzne elektronické zariadenie. NetCOPE [8] je konfigurovateľný framework, ktorý pomože k rýchlejšiému vývoji návrhov na FPGA karty a umožnuje lahšie využitie celého systému. Celý framework je rozdelený na dve časti podľa [8]. Na firmware, ktorý zahrňuje do seba jednotlivých komponentov na karte a aplikačné jádro, kam je nahraný užívateľom stvorený návrh. Na softwarovú čášt, kde je ovládač

karty a aplikačný software. Tieto dve časti sú spojené s rozhraním PCI. Štruktúra frameworku je na obrázku 4.3.



Obr. 4.3: Štruktúra framworku NetCOPE [8]

5 NAPROGRAMOVANÁ ŠIFRA AES

Výstupom praktickej časti práce je naprogramovaný AES pre šifrovanie a dešifrovanie. V tejto časti je popísaný princíp fungovania vytvoreného návrhu a jednotlivých komponentov. Naprogramovanie bolo zrealizované vo vývojovom prostredí Vivado od firmy XILINX v jazyku VHDL.

AES podporuje viac rozmerov kľúča K. Zvolená dĺžka kľúča je 128 bitov. Celý návrh je riadený pomocou hodinového a resetovacieho signálu. Blokové schéma návrhu šifrovania je na obrázku 5.3.

5.1 Šifrovanie

5.1.1 Top modul Encryption.vhd

Hlavným zdrojovým súborom pre šifrovanie je `Encryption.vhd`. Tento súbor je motorom celej šifry. Postupne ovláda celý návrh, aby šifrovací proces prebehla správne. Zvolené rozhranie je znázornené v zdrojovom kóde 5.1.

```
Port (
  RX_DATA     : in std_logic_vector(127 downto 0);
  RX_DATA_VLD : in std_logic;
  KEY         : in std_logic_vector(127 downto 0);

  CLK         : in std_logic;
  RESET       : in std_logic;

  TX_DATA     : out std_logic_vector(127 downto 0);
  TX_DATA_VLD : out std_logic
    );
```

Výpis 5.1: Rozhranie šifrovacej komponenty

Na vstupnom porte `RX_DATA` chodia vstupné dáta pre šifrovanie. Na `RX_DATA_VLD` prichádza signál ukazujúci validitu vstupných dát. Port `KEY` je vstupom kľúča K. Na porte `CLK` chodí hodinový signál, a na `RESET` resetovací signál.

Z teórie AESu zo sekci 3 plyne, že AES je iteračná šifra. Pri 128 bitovej dĺžke kľúča K počet iterácií je desať. Naprogramovaná šifra, je schopná za jeden takt hodinového signálu zašifrovať vstupné dáta, ale je to veľmi náročné pre hardware a znížil by sa jeho výkon. Pri takej náročnosti by karta musela pracovať na malej frekvencií, aby stihľa všetky procesy a zabránila voči vzniknutiu kritických ciest. V tomto prípade by nebola využitá jedna najvýznamnejšia vlastnosť karty, rýchlosť. Preto bola pridaná možnosť vkladať pomocné registry do návrhu na princípe z [7]. Pomocou toho je možné znížiť výpočtové nároky hardweru a výrazne zvýšiť výkon

a rýchlosť celého návrhu. To tak, že logika celého návrhu je rozdelení na menšie časti a medzi jednotlivé časti sú uložené pomocné registry viz obrázok 5.3. V každom hodinovom takte sa do pomocných registrov zapisujú aktuálne hodnoty rozdelených častí. V `Encription.vhd` je to vyriešené pomocou pevne zadefinovaných konštant.

```
constant STAGE_0  : boolean := false;
constant STAGE_1  : boolean := true;
constant STAGE_2  : boolean := false;
constant STAGE_3  : boolean := true;
constant STAGE_4  : boolean := false;
constant STAGE_5  : boolean := true;
constant STAGE_6  : boolean := false;
constant STAGE_7  : boolean := false;
constant STAGE_8  : boolean := true;
constant STAGE_9  : boolean := false;
constant STAGE_10 : boolean := true;
```

Výpis 5.2: Konštanti pre povolenie mezdiiteračných registrov

Keď majú hodnotu „true“ tak v každom hodinovom takte sa ukladajú výstupy iterácií do zvoleného pomocného registra. V zdrojovom kóde 5.3 je znázornené časť kódu zaregistrovania.

```
Round_2 : Round_1_9     -- Komponenta pre prvých 9 iterácií
Port map(  round_data_in  => reg_d_1,
           roundkey_in    => reg_k_1,
           roundconstant  => roundconst(287 downto 256),
           roundkey_out   => roundkey2,
           round_data_out => round2out);

stage_2_reg_gen : if (STAGE_2 = true) GENERATE

reg_d_2_we <= reg_en_1;   -- zapisovací signál pre dáta
reg_k_2_we <= reg_en_1;   -- zapisovací signál pre klúc
reg_en_we_2 <='1';
reg_round2_p : process(CLK)
  begin
    if rising_edge(CLK) then
       if RESET = '1'  then
          reg_d_2 <= (others => '0');
       elsif reg_d_2_we = '1' then
          reg_d_2 <= round2out; -- zaregistrovanie dáta
       end if;
    end if;
    end process;

reg_round2_p_k : process(CLK)
  begin
```

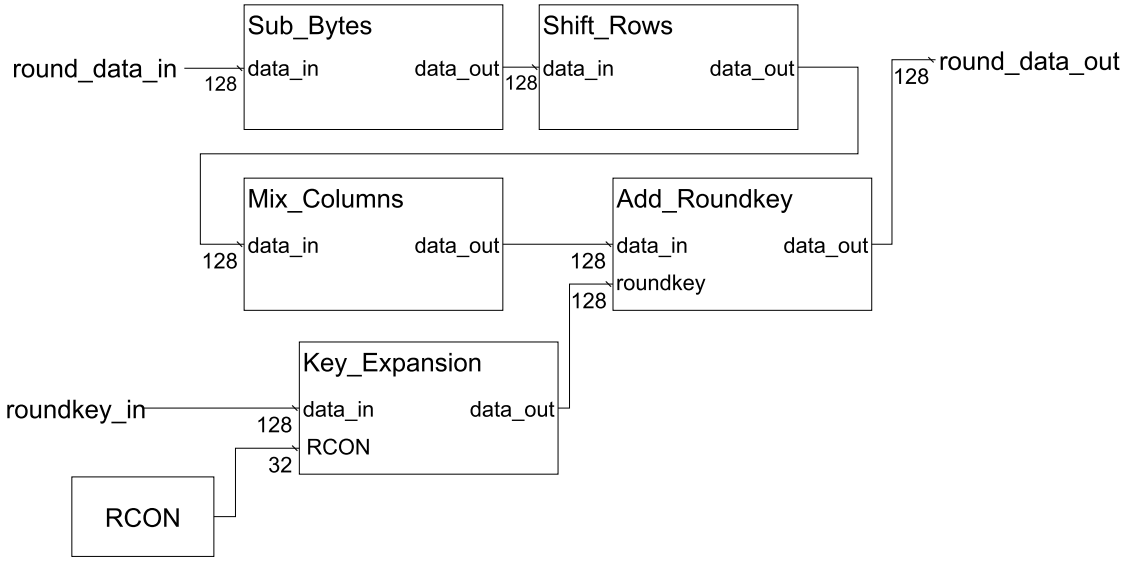
```
    if rising_edge(CLK) then
       if RESET = '1'  then
          reg_k_2 <= (others => '0');
       elsif reg_k_2_we = '1' then
          reg_k_2 <= roundkey2; -- zaregistrovanie klúča
       end if;
    end if;
  end process;

reg_round2_p_en : process(CLK)
 begin
   if rising_edge(CLK) then
      if RESET = '1'  then
         reg_en_2 <= '0';
      elsif reg_en_we_2 = '1' then
         reg_en_2 <= reg_en_1; -- registrovanie validného signálu
       end if;
    end if;
  end process;
end GENERATE;

stage_2_gen : if (STAGE_2 = false) GENERATE
    reg_d_2 <= round2out;
    reg_k_2 <= roundkey2;   -- bez registrovania
    reg_en_2 <= reg_en_1;
end GENERATE;
```

Výpis 5.3: Zdrojový kód pre zaregistrovanie po druhej iterácie

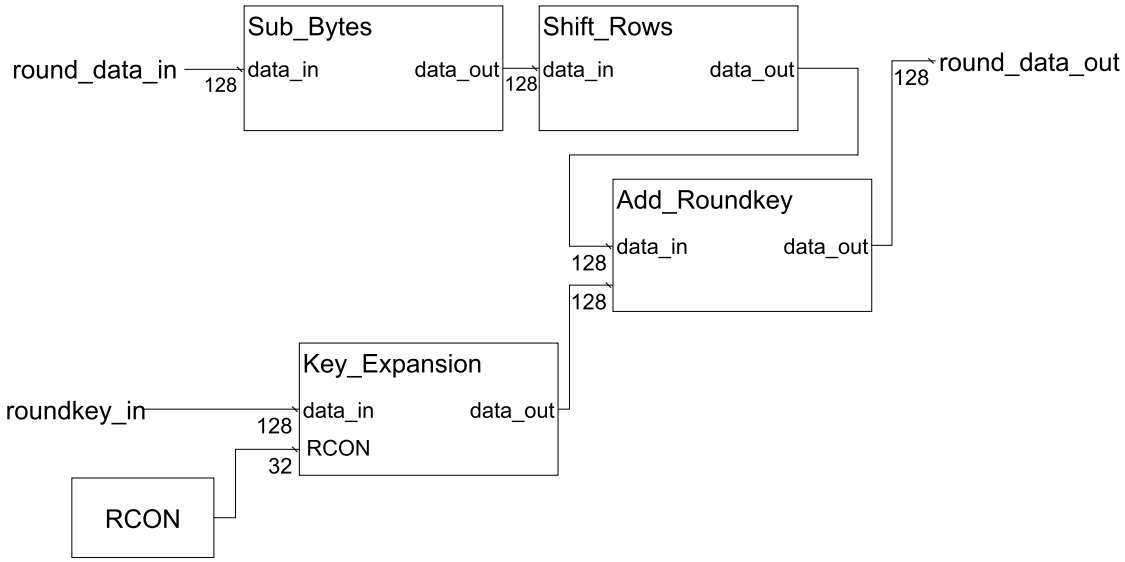
Komponenta Round_1_9 zahrňuje do seba naprogramované jednotlivé časti iterácií. SubBytes, ShiftRows, MixColumns, AddRoundKey a KeyExpansion je používané deväť krát. Komponenta Round_10 sa používa pre poslednú iteráciu. Výstupom tohto komponenty je kryptogram a je spojený s výstupným portom `TX_DATA`. Výstupný validný signál je spojený s výstupným portom `TX_DATA_VLD`.

5.1.2 Komponenty iterácií

V komponente Round_1_9 a Round_10 je namapovanie jednotlivých častí. Komponenty boli navrhované tak, aby iteračné kľúče neboli vygenerované pred začiatkom šifrovania, ale paralelne počas šifrovania. Na obrázku 5.1 a 5.2 je blokové schéma jednotlivých iteračných komponentov.



Obr. 5.1: Bloková schéma iterácií



Obr. 5.2: Bloková schéma poslednej iterácie

5.1.3 Substitúcia bajtov

V zdrojovom súboru `SubBytes.vhd` je naprogramované prevedenie substitúcie bajtov. Má jeden vstup `Sub_in` a jeden výstup `Sub_out`. Substitučná tabuľka (SBOX) je pevne daná ako konštanta. Je to definované ako jednorozmerné pole s 256 prvkami. Každý prvok je 8 bitov dlhý. Substitúcia je urobená v generickom for cyklu. Genericky for cyklus vygeneruje všetky výpočty pre všetky i paralelne. Algoritmus naprogramovanej substitúcie spočíva v tom, že hodnota bajtu v decimálnom tvaru udáva na koľkátom mieste sa nachádza ekvivalentný bajt v Substitučnej tabuľke

(SBOX).

```
SubBytes_operation : for i in 15 downto 0 generate
 begin
  sub_d((i+1)*8-1 downto i*8) <=
          SBOX(conv_integer(Sub_Bytes_in((i+1)*8-1 downto (i*8));
 end generate;
```

Výpis 5.4: Substitúcia bajtov

V zdrojovom kóde namiesto signálu sub_d je Sub_bytes_data.

5.1.4 Rotácia riadkov

Signály row1 až row4 sú manuálne poskladané tak, aby vyhovovali požiadavkám algoritmu rotácie riadkov.

```
row1 <= in(127 downto 120) & in(87 downto 80) & in(47 downto 40)
                                                & in(7 downto 0);
row2 <= in(95 downto 88) & in(55 downto 48) & in(15 downto 8)
                                             & in(103 downto 96);
row3 <= in(63 downto 56) & in(23 downto 16) & in(111 downto 104)
                                             & in(71 downto 64);
row4 <= in(31 downto 24) & in(119 downto 112) & in(79 downto 72)
                                                & in(39 downto 32);
```

Výpis 5.5: Manualná rotácia riadkov

Vstupný port `Shift_Rows_in` kvôli šírke zdrojového kódu je zmenený na in.

5.1.5 Zmiešanie stĺpcov

Pre Zmiešanie stĺpcov bolo potrebné naprogramovať štyri zdrojové súbory:

- `MixColumns.vhd`,
- `MixColumns.vhd`,
- `Multiply_2.vhd`,
- `Multiply_3.vhd`.

Prvý zdrojový súbor v našom prípade `MixColumn.vhd` spraví metódu pre univerzálny stĺpec s počtom prvkov štyri. Jednotlivé bajty stĺpca sú rozdelené do signálov. Nasledovne sú používane komponenty Multiply_2 a Multiply_3. Teória násobenia s hodnotou 02_{16} a 03_{16} je uvedené v kapitole 3.1.3. Pri implementácií do programovacieho jazyka sa dá jednoducho vyhýbať popísaných javov. S tým, že sa násobí v Galoisovom poli, boli používane dve algoritmy. Prvý algoritmus je využívaný v `Multiply_2.vhd`. Algoritmus používa jednu podmienku. Keď MSB (Most Significant Bit), vtedy bit najväčšou váhou násobeného bajtu je v log. 1, tak bity sú logicky

posunuté o jedno doľava a xorované s bajtom $1B_{16}$. V opačnom prípade keď MSB je v log. 0, bity sú iba logicky posunuté o jedno.

```
if multiply_x2_in(7) = '1' then
 multiply_x2_out <= (multiply_x2_in(6 downto 0) & '0') xor X"1B";
else
 multiply_x2_out <= (multiply_x2_in(6 downto 0) & '0');
end if;
```

Výpis 5.6: Násobenie s 2

V `Multiply_3.vhd` je používaná rozšírená verzia algoritmu z predošlej časti. Bajt násobení s 03_{16}, je rovná hodnote toho istého bajtu po násobenia s 02_{16} a následne xorované s tým istým bajtom.

Po násobenia je používaný posledný proces. Každý prvok stĺpca je vynásobená s príslušným riadkom mixovacej matice a násobky sú následne xorované. Výstup celého zdrojového súboru je poskladaný z bajtov a tak je získaný výsledný stĺpec.

```
byte1out <= byte1x2 xor byte2x3 xor byte3 xor byte4;
byte2out <= byte1 xor byte2x2 xor byte3x3 xor byte4;
byte3out <= byte1 xor byte2 xor byte3x2 xor byte4x3;
byte4out <= byte1x3 xor byte2 xor byte3 xor byte4x2;

multiply_column_out <= byte1out & byte2out & byte3out & byte4out;
```

Výpis 5.7: Skladanie bajtov

V `MixColumns.vhd` sú namapované jednotlivé stĺpce na komponentu MixColumn.

5.1.6 Sčítanie s iteračním kľúčom

K výstupu z MixColumns je prixorovaný v každej iterácií iteračný kľúč. Vygenerovanie iteračních kľúčov je prevedené v zdrojovom súbore `KeyExpansion.vhd`.

5.1.7 Generovanie iteračních kľúčov

V zdrojovom súbore `KeyExpansion.vhd` je naprogramované vygenerovania iteračného kľúča. Na vstup key_in je privedený kľúč, s ktorým sa bude pracovať. Po rozdelení vstupného kľúča do jednotlivých stĺpcov nasleduje proces rot_p.

```
rot_p : process(column_4)
    begin
    rotword <= column_4(23 downto 0) & column_4(31 downto 24);
    end process;
```

Výpis 5.8: Rotácia bajtov

Tento proces si vezme posledný stĺpec kľúča a urobí aritmetický posun bajtov o jedno doľava.

Druhý proces sub_p si urobí substitúciu bajtov stĺpca presne takým istým princípom ako je urobené v zdrojovom súbore `SubBytes.vhd`.

```
sub_p : process(rotword)
    begin
    sub_byte1 <= SBOX(conv_integer(rotword(31 downto 24)));
    sub_byte2 <= SBOX(conv_integer(rotword(23 downto 16)));
    sub_byte3 <= SBOX(conv_integer(rotword(15 downto 8)));
    sub_byte4 <= SBOX(conv_integer(rotword(7 downto 0)));
    end process;
    subbytescolumn <= sub_byte1 & sub_byte2 & sub_byte3 & sub_byte4
```

Výpis 5.9: Výsledný iteračný kľúč

V nasledujúcom zdrojovom kóde je uvedené vypočítanie nového kľúča, a xorovanie vypočítaných častí.

```
round_key_column1 <= column_1 xor sub_bytes_column xor RConstant;
round_key_column2 <= column_2 xor round_key_column1;
round_key_column3 <= column_3 xor round_key_column2;
round_key_column4 <= column_4 xor round_key_column3;

RKey <= round_key_column1 & round_key_column2
                          & round_key_column3 & round_key_column4;
```

Výpis 5.10: Výsledný iteračný kľúč

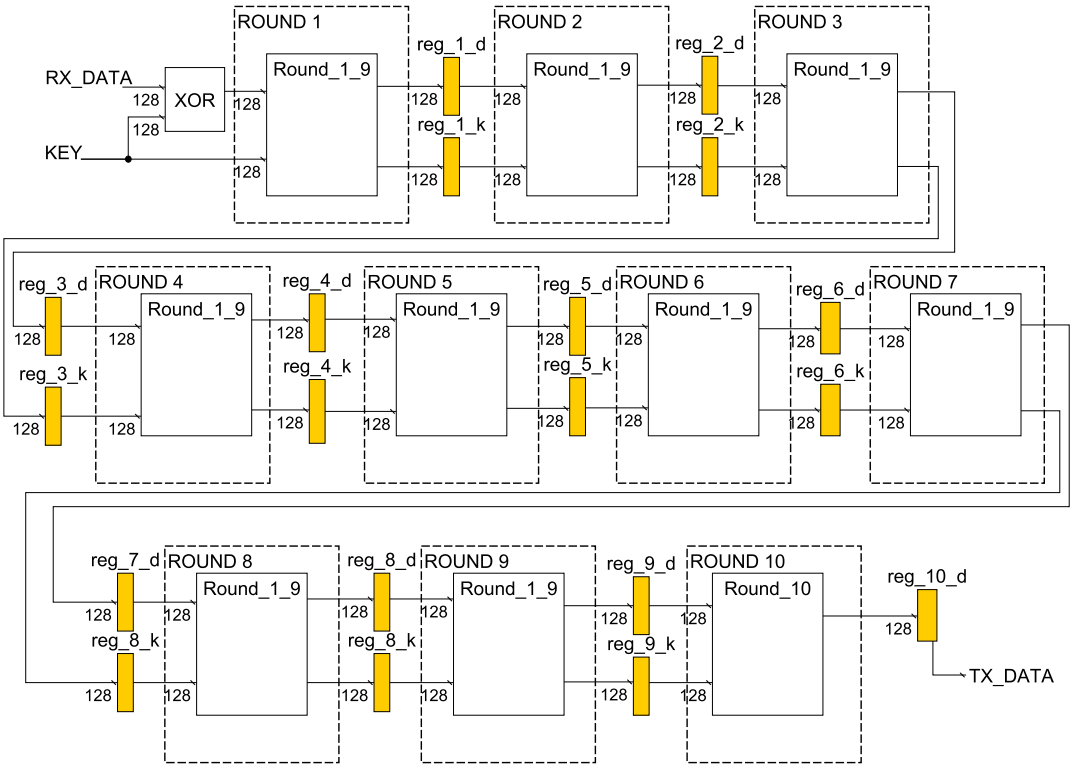
5.2 Dešifrovanie

`Decryption.vhd` je hlavným komponentom dešifrovania. Naprogramovanie dešifrovania bolo uskutočnené podľa teórií zo sekcie 3.3. Vstupné a výstupné rozhranie je rovnaké ako u šifrovania. Zahrňuje do seba pomocné komponenty DATA_BUFF, key_table, Round1_9_Decryption, Round_10_Decryption a invezné iteračné časti.

Na základe teoretických poznatkov dešifrovania, je potrebné vygenerovať pred začitakom dešifrovacieho procesu všetky iteračné kľúče. K tomu slúží komponenta key_table. Najprv sú popísané komponenty key_table a DATA_BUFF.

5.2.1 Key_table

Na vstupy key a key_vld prichádza kľúč a validní signál pre kľúč. Na výstupe rkey1 až rkey10 sú všetky iteračné kĺúče. K vygenerovaniu samotných kľúčov je používaný



Obr. 5.3: Blokové schéma šifrovania

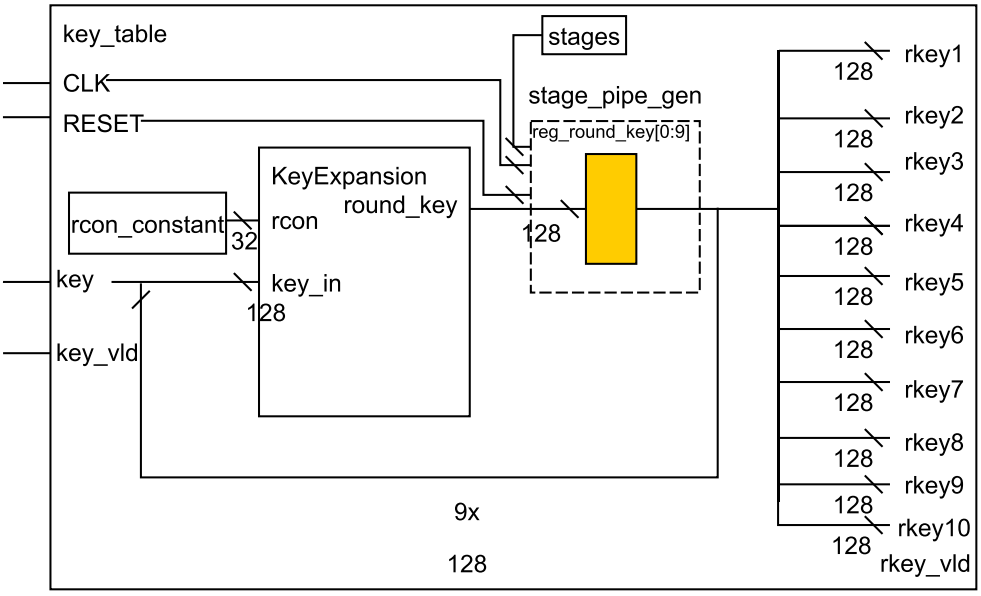
vyššie popísaný komponent Key_Expansion. Aby sa dali dosiahnúť vyššie frekvencie a nevznikli kritické cesty pri vygenerovaní kľúčov je pridaná možnosť rozsekania a ukladania hodnôt do pomocných registrov. Na obrázku 5.4 je bloková schéma komponenty key_table.

Stages je dopredu nadefinovaná 10 bitov dlhá konštanta. Nastavením jednotlivých bitov na log. 1 docielíme to, aby v tej iterácií bol používaný pomocný register. Napr. keď všetky bity sú v log.1, po každom vygenerovaní iteračný kľúč sa uloží do pomocného registra reg_round_key[0:9]. Reg_round_key je pole, ktoré sa skladá z desať 128 bitových registrov. S takým nastaveným vygenerovanie všetkých kľúčov bude trvať desať hodinových taktov. Komponent je naprogramovaný v zdrojovom súbore `key\_table.vhd`.

5.2.2 Data_BUFF

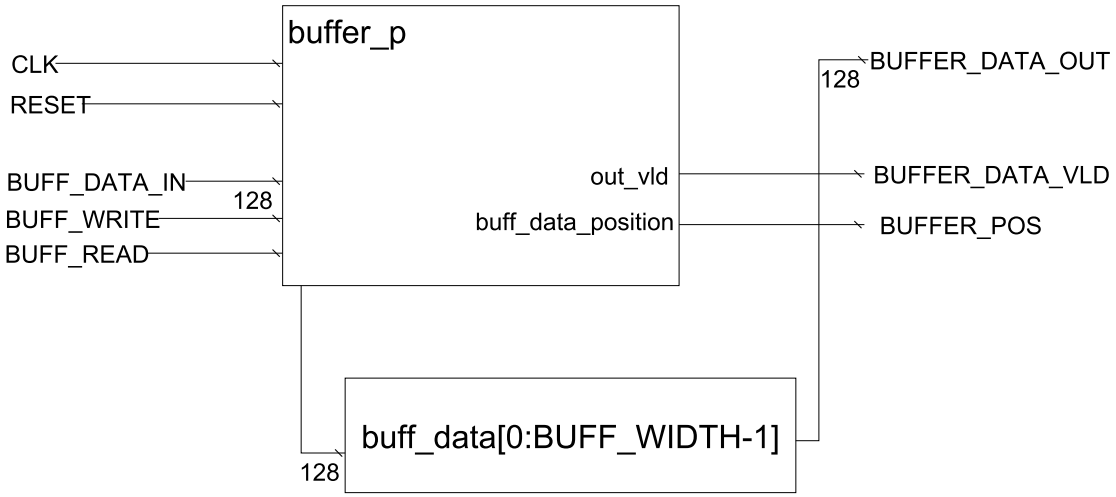
Môže sa nastať prípad, keď nie sú vygenerované všetky iteračné kľúče, ale na vstupe už sú validné dáta. Ku eliminovaniu tohto problému slúži komponent Data_BUFF. Chová sa ako jednoduchý First In First Out (FIFO) buffer. Uchová a vyčítáva dáta v takom poradí v akom prišli.

Na vstupnom porte `BUFF_DATA_IN` prichádzajú dáta. Na portoch `BUFF_WRITE` a



Obr. 5.4: Blokové schéma key_table

`BUFF_READ` prichádzajú zapisovacie a čítacie signály. Bloková schéma komponentu je znázornená na obrázku 5.5.



Obr. 5.5: Blokové schéma komponenty Data_Buffer

Ukladacou memóriou komponentu je signál buff_data. Pomocou generických konštant `DATA_WIDTH` a `BUFF_WIDTH` je nadefinovaná dĺžka a maximální počet adries.

```
type data_buff is array (0 to BUFF_WIDTH-1) of
                        std_logic_vector(DATA_WIDTH-1 downto 0);
signal buff_data     : data_buff;
```

Výpis 5.11: Nadefinovanie memórie data_buff

Process buffer_p ovláda zapísanie a vyčítanie z bufferu. Pri každom zapisovaní sa ukladajú dáta na nultú adresu memórie. Aby komponent fungoval ako FIFO buffer, po každom zapísaní sú posunuté prvky na nasledujúcu adresu. Adresu prvých zapísaných bitov, ktoré majú najväčšiu prioritu, ukazuje signál buff_data_pos. Aby ukazoval reálnu adresu prvých zapísaných bitov, pri každom zapisovaní hodnota signálu sa zvýši o jedno. Keď sa vyčítá tak sa zníži. V prípade, keď vyčítanie a zapisovanie je prevedené naraz hodnota ostáva rovnaká.

```
elsif BUFF_WRITE = '1' and BUFF_READ = '0' then
    buff_data(0) <= input;
    for i in 0 to BUFF_WIDTH - 2 loop
      buff_data(i + 1) <= buff_data(i);
    end loop;
    buff_data_position <= buff_data_position +1 ;
```

Výpis 5.12: Zdrojový kód procesu buffer_p pre zapisovanie

V prípade, keď sa iba vyčíta z bufferu, na adresu vyčítaných bitov v memórií sú naplnené nuly.

```
elsif BUFF_WRITE = '0' and BUFF_READ = '1' then
    if buff_data_position /= 0 then
      buffer_out_data <= buff_data(buff_data_position-1);
      out_vld <= '1';
      buff_data_position <= buff_data_position -1;
      buff_data(buff_data_position -1 ) <= (others => '0');
    else
      out_vld <= '0';
      buff_data_position <= 0;
      buff_data(buff_data_position ) <= (others => '0');
    end if;
```

Výpis 5.13: Zdrojový kód procesu buffer_p pre vyčítanie

```
elsif BUFF_WRITE = '1' and BUFF_READ = '1' then
    buffer_out_data <= buff_data(buff_data_position-1);
    buff_data(0) <= input;
    for i in 0 to BUFF_WIDTH - 2 loop
      buff_data(i + 1) <= buff_data(i);
    end loop;
    out_vld <= '1';
```

Výpis 5.14: Zdrojový kód procesu buffer_p pre vyčítanie a zapisovanie naraz

5.2.3 Top modul dešifrovania Decryption.vhd

Na začiatku celého dešifrovania je uložení kľúč a vstupné dáta do registrov reg_key, reg_data a reg_data_vld. Zaregistrovaný kľúč je namapovaní na vstup komponentu key_table. Koniec vygenerovania všetkých iteračných kľúčov ukazuje log. 1 na výstupnom porte `rkey_vld` u komponenty key_table. Vygenerovanie všetkých iteračných kľúčov trvá viac taktov. Medzitým na vstupe už môžu byť validné dáta. Aby sa zabránilo ku strate validných dát sa používa komponent DATA_BUFF.

Process write_read_buff_p riadí posieľanie zapisovacej a vyčítacej signálu do bufferu. Všetky ošetrené podmienky, podľa ktorých je naprogramované chovanie procesu sú založené na stavoch signálov rkey_vld, reg_data_vld a buff_pos.

Keď rkey_vld je v log. 1 a reg_data_vld v log. 0 to znamená, že už sú vygenerované všetky kľúče, ale na vstupe nie sú validné dáta. Vtedy proces si pozrie na hodnotu buff_pos. V prípade, keď sa nerovná nule, dáta z bufferu sú vyčítané po sebe v každom hodinovom takte, kým pozícia nebude nulová. Tento prípad je znázornení na nasledujúcom zdrojovom kóde.

```
elsif rkey_vld = '1' and reg_data_vld = '0' then
        if  buff_pos /= 0 then
          buff_w <= '0';
          buff_r <= '1';
        elsif buff_pos = 0 then
          buff_w <= '0';
          buff_r <= '0';
        end if;
```

Výpis 5.15: Signál rkey_vld je log. 1 a reg_data_vld log. 0

V prípade, keď validné dáta sú na vstupe (reg_data_vld je v log. 1), ale potrebné kľúče nie sú vygenerované (rkey_vld je v log. 0), proces pošle zapisujúci signál do bufferu.

```
elsif rkey_vld = '0' and reg_data_vld = '1' then
        buff_w <= '1';
        buff_r <= '0';
        buff_data <= reg_data;
end if;
```

Výpis 5.16: Signál rkey_vld je log. 0 a reg_data_vld log. 1

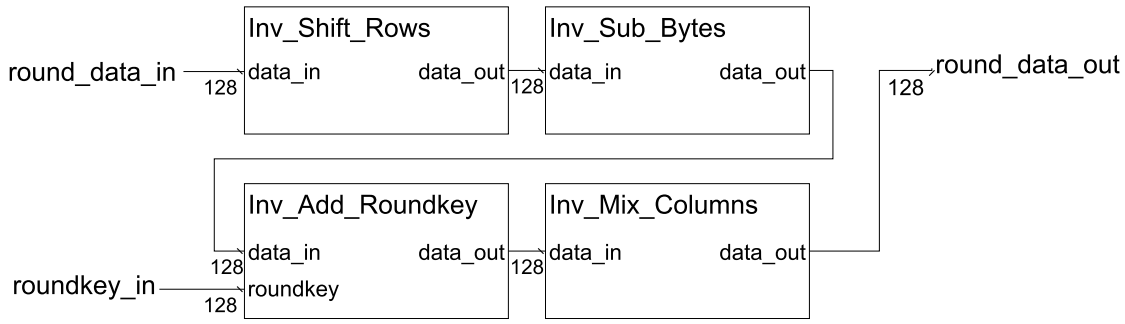
Keď vygenerovanie kľúčov sa skončilo a na vstupe sa nachádzajú validné dáta, sú ošetrené dva prípady. Hodnota signálu buff_pos sa nerovná nule. To znamená, že v bufferu ešte sú dáta, ktoré majú väčšiu prioritu pri vyčítaní. Vtedy proces pošle zapisovací a vyčítajúci signál do bufferu. Keď sa rovná nule, buffer je prázdný a dáta sú hneď privedené na vstup dešifrovania.

```
      elsif rkey_vld = '1' and reg_data_vld = '1' then
        if buff_pos /= 0 then
          buff_w <= '1';
          buff_data <= reg_data;
          buff_r <= '1';
        elsif buff_pos = 0 then
          buff_w <= '0';
          buff_r <= '0';
        end if;

init_data <= reg_data when rkey_vld = '1'
                       and reg_data_vld = '1'
                       and buff_pos = 0 else
             buff_data_out ;
```

Výpis 5.17: Signáli rkey_vld a reg_data_vld sú v log. 1

Obdobne ako u komponentu šifrovania, je možnosť používania pomocných registrov, pomocou dopredu nastavenou konštantou stages. Rovnako ako v komponente key_table. Pomocné registery sú ukladané medzi jednotlivé iterácie.

Naprogramované komponenty pre jednotlivých iterácií sú v zdrojových súboroch `Round_1_9_Decryption` a v `Round_10_Decryption`. Podľa teoretických poznatkov je namapované správne poradie jednotlivý iteračných častí.



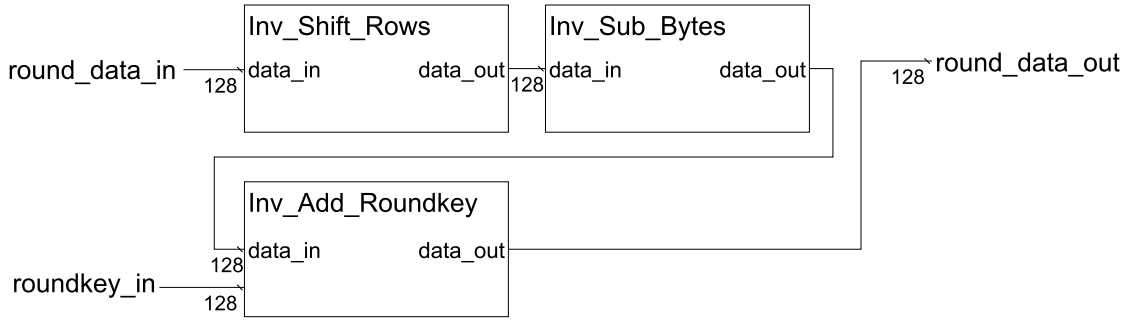
Obr. 5.6: Bloková schéma prvých deväť iterácií

5.2.4 Inverzná substitúcia bajtov

Celý komponent pracje rovnako ako u šifrovania. Jediným rozdieľom je, že pre substitúciu sa používa inverzná substitučná tabuľka (INV_SBOX).

5.2.5 Inverzná rotácia riadkov

Obdobne ako u šifrovania riadky sú manuálne poskladané.



Obr. 5.7: Bloková schéma poslednej iterácie

```
row1 <= in(127 downto 120) & in(23 downto 16) & in(47 downto 40)
                                              & in(71 downto 64) ;
row2 <= in(95 downto 88) & in(119 downto 112) & in(15 downto 8)
                                              & in(39 downto 32);
row3 <= in(63 downto 56) & in(87 downto 80) & in(111 downto 104)
                                            & in(7 downto 0);
row4 <= in(31 downto 24) & in(55 downto 48) & in(79 downto 72)
                                            & in(103 downto 96);
```

Výpis 5.18: Inverzná rotácia riadkov

5.2.6 Inverzné zmiešanie stĺpcov

Aj u inverznej zmiešaní stĺpcov je rovnaký princip ako u šifrovaní. S tým rozdieľom, že násobuje sa s inverznou mixovaciou maticou (tab. 3.4). Násobenia sú naprogramované v

- `Multiply9.vhd`,
- `Multiply11.vhd`,
- `Multiply13.vhd`,
- `Multiply14.vhd`.

Pri programovania bol používaný algoritmus pre násobenia s dvoma 5.1.5. Čísla, s ktorými sa násobuje sú rozdelené podľa nasledujúcich rovníc

$$\begin{aligned} x \times 9 &= (((x \times 2) \times 2) \times 2) + x, \\ x \times 11 &= ((((x \times 2) \times 2) + x) \times 2) + x, \\ x \times 13 &= ((((x \times 2) + x) \times 2) \times 2) + x, \\ x \times 14 &= ((((x \times 2) + x) \times 2) + x) \times 2. \end{aligned} \tag{5.1}$$

Kde x je násobené číslo, znak $\times$ je násobenie a $+$ je xorovanie.

Princíp komponenty Multiply9 je následovny. Vstupný bajt sa uloží do signálu

byte. Následne sa násobí s dvojkou tri rázy posebe a uloží sa do signálu value3. Hodnota signálu value3 je akoby násobený bajt bol vynásobený s osmyčkou. A ako posledné sa ktomu prixoruje násobené číslo. Stým sa docielio k násobeniu s devinou.

```
byte <= multiply9_in;
multiplyx9 : process(byte, value1, value2, value3)
begin
    if (byte(7) = '1') then
      value1 <= ((byte(6 downto 0) & '0') xor X"1B");
    else
      value1 <= byte(6 downto 0) & '0' ;
    end if;
    if (value1(7) = '1') then
      value2 <= ((value1(6 downto 0) & '0') xor X"1B");
    else
      value2 <= value1(6 downto 0) & '0' ;
    end if;
    if (value2(7) = '1') then
      value3 <= ((value2(6 downto 0) & '0') xor X"1B");
    else
      value3 <= value2(6 downto 0) & '0' ;
    end if;
    bytex9 <= value3 xor byte;
end process;
```

Výpis 5.19: Násobenie jedného bajtu s devinou

Ostatné násobenia sú naprogramované na rovnakom princípe podľa rovíc 5.1.

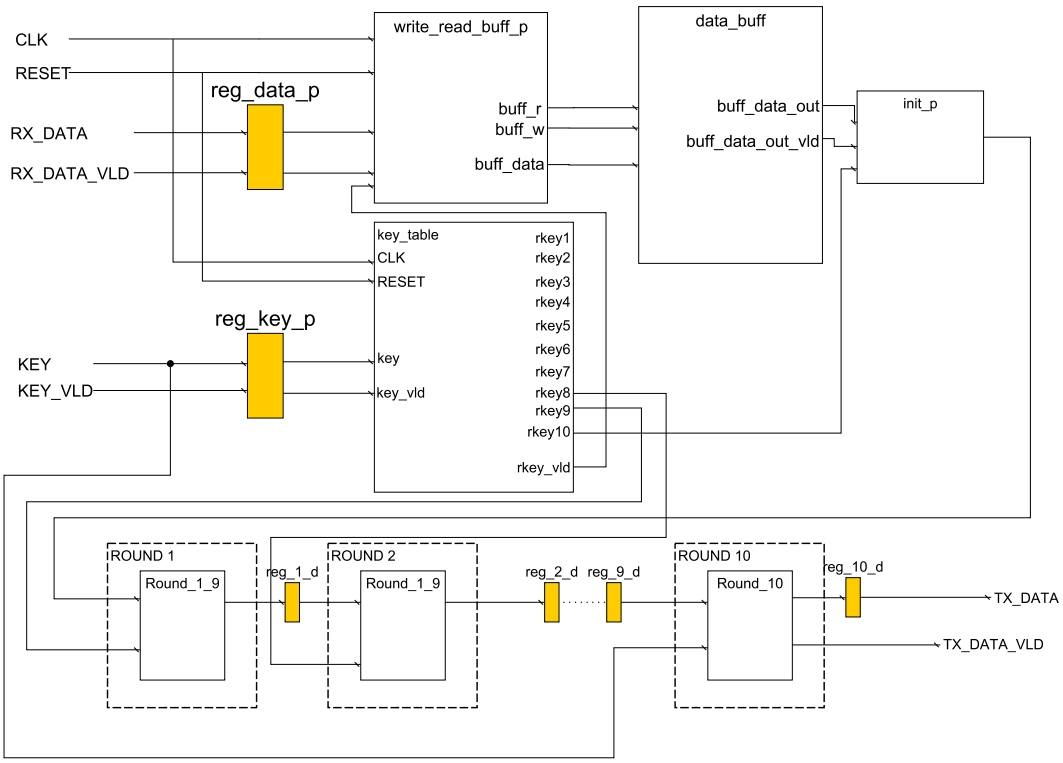
Komponenta AddRoundKey_Decryption robí sčítanie s iteračným kľúčom. Na obrázku 5.8 je blokové schéma návrhu dešifrovania.

5.3 Naprogramovaný mód ECB

Naprogramovaná samotná šifrovacia a dešifrovacia komponenta sa chová v operačnom móde ECB.

5.4 Naprogramovaný mód CBC

Šifrovanie je naprogramované v zdrojovom súbore `cbc_encryption.vhd` a dešifrovanie v `cbc_decryption.vhd`



Obr. 5.8: Blokové schéma dešifrovania

5.4.1 Šifrovanie v CBC

Vstupné dáta sú závislé na predchádzajúcich zašifrovaných dátoch. V prípade, keď šifrovanie trvá viac taktov, je potrebné si ukladať validné dáta na vstupu, kým sa šifrovanie neskončí. Na tento účel bolo možné použiť naprogramovaný buffer zo sekci 5.2.2. Problém je vtom, že v prípade maximálneho zaťaženia kapacitné nároky bufferu by boli veľmi veľké. Maximálne zaťaženie znamená, že na vstupe sú v každom hodinovom takte validné dáta. Predpokladajme, že naimplementovaná šifra používa všetky pomocné registery, aby dosiahnutá frekvencia bola čo najväčšia. V móde CBC každé vstupné dáta okrem prvých sú xorovaný pred vstupom do šifry s predchádzajúcimi zašifrovanými dátami. To znamená, že zašifrovanie 128 bitov by trvalo v tomto prípade desať hodinových taktov. Dôvtedy všetky validné dáta by boli ukladané do bufferu. Keď v každom hodinovom takte sú na vstupe validné dáta, po 100 taktov by už bolo zapísaných do bufferu 90 dátových blokov. S takými nasteveniami sice návrh by pracoval na vysokých frekvenciach, ale nezvýšila by mimoriadne celkový výkon šifrovania. Zo získaných teoretických výsledkov naprogramovalo sa šifrovanie bez možnosti použitia pomocných registrov. Sice tak dosiahnutá frekvencia je nižšia, ale zašifrované množstvo dát za sekundu nie je výrazne menšia než s pomocnými registrami. Avšak nároky na čerpatelných zdrojov karty sú oveľa nižšie.

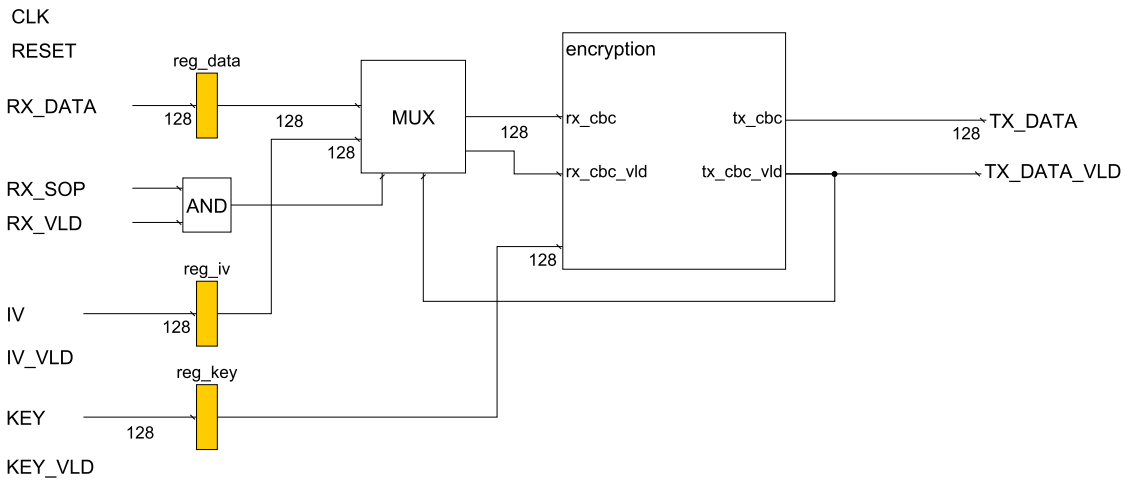
Vstupné a výstupné rozhranie komponenty je znázornené na obrázku 5.9. Pomo-

cou dvoch podmienok je ošetrené priraďovanie vhodných dát do šifrovania. V momente, keď na vstupnom porte `RX_VLD` a `RX_SOP` je log. 1, dáta sa xorujú s inicializačným vektorom. V druhom prípade, keď `RX_VLD` je v log. 1, `RX_SOP` v log. 0 a out_vld v log. 1 na vstup šifry sú privedené, validné dáta xorované s výstupom šifry.

```
rx_cbc <= IV xor RX_DATA when RX_VLD = '1' and RX_SOP = '1' else
          tx_cbc xor RX_DATA when RX_VLD = '1'
                                and RX_SOP = '0'
                                and out_vld = '1' else nulls;

rx_cbc_vld <='1' when RX_VLD = '1' and RX_SOP = '1' else
             '1' when RX_VLD = '1' and RX_SOP = '0'
                                   and out_vld = '1' else '0';
```

Výpis 5.20: Priraďovanie vhodných dát na vstup šifry

Na výstpny port `TX_DATA` a `TX_VLD` sú namapované výstupy šifrovacej komponenty. Blokové schéma šifrovania je na obrázku 5.9.



Obr. 5.9: Blokové schéma komponenty CBC_encryption

5.4.2 Dešifrovanie v CBC

Pri dešifrovaní vstupné dáta nie sú závislé na predchádzajúcich dátoch. Táto vlastnosť dovoluje používania pomocných registrov. Podobne ako vo väčšine prípadov, vstupné dáta a validné signáli sú zaregistrované do registrov. Následne sú namapované na vstupy dešifrovacej komponenty. Získané dešifrované dáta z komponentu Decryption, je potrebné xorovať buď s inicializačným vektorom, keď sa jedná o prvý blok dát, alebo s predchádzajúcimý vstupnými dátami. Pre ukladanie validných

dát je používaná komponenta DATA_BUFF. Zapisovnie je výkonané v momente, keď zaregistrované dáta sú validné. Zapisujú sa vstupné dáta so signálom ukazujúci prvý blok dát. Vyčítanie je vykonané, keď na výstupe komponenty Decryption sú validné dáta.

```
buff1_w <= '1' when reg_rx_vld = '1' else
           '0';

buff1_r <= '1' when tx_cbc_vld = '1' else
           '0';
```

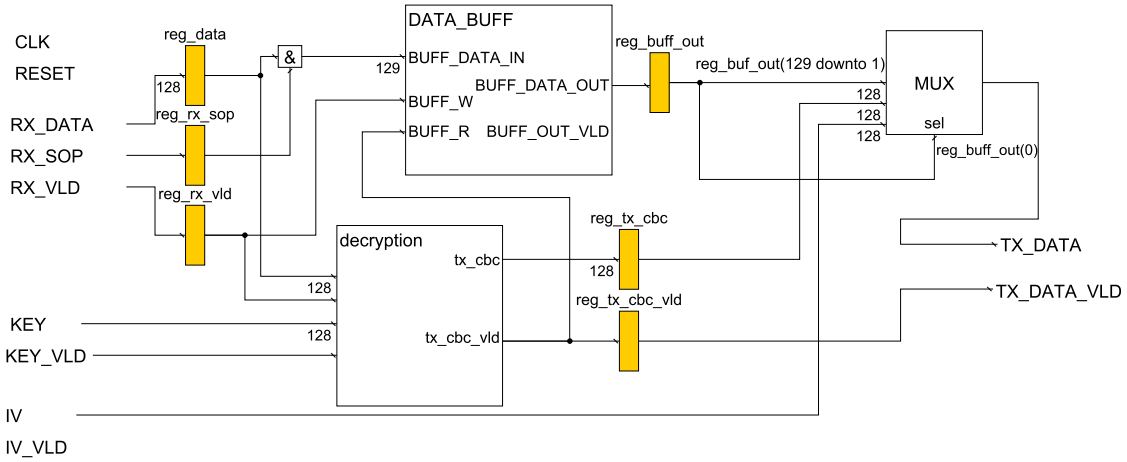
Výpis 5.21: Zapisovanie a vyčítanie z bufferu

Kvôli správnemu synchronizovaniu, výstupy dešifrovacieho komponentu a bufferu sú ukladané do registrov. Následne je vykonaný posledný krok. Keď vyčítané dáta z bufferu sú prvý blok dát, tak sa xorujú s inicializačným vektorom. Inak s vyčítanými dátami z bufferu.

```
out_data <= reg_tx_cbc xor IV when buff1_data_out(0) = '1' else
  reg_tx_cbc xor reg_buff_out when buff1_data_out(0) = '0' else
  nulls;

out_data_vld <= reg_tx_cbc_vld;
```

Výpis 5.22: Posledný krok dešifrovania v CBC

Na nasledujúcom obrázku je blokové schéma celého dešifrovania v CBC móde.



Obr. 5.10: Blokové schéma dešifrovania v CBC móde

5.5 Naprogramovaný mód CFB

U šifrovaní a dešifrovaní v tomto móde je používaná komponenta `Encryption.vhd` a `data_buff.vhd`. Šifrovacia a dešifrovacia komponenta celého módu je naprogramovaný tak, aby bol schopný pracovať s rôznymi dovolenými dĺžkami vstupných dát. Pri návrhu šifrovaní a dešifrovaní sú používané pomocné registry.

5.5.1 Šifrovanie

Šifrovanie je naprogramované v zdrojovom súbore `cfb_encryption.vhd`. Vstupné a výstupné rozhranie je rovnaký ako u komponentu operačného módu CBC. Validné dáta su hneď zapísané do bufferu pre neskoršie používanie. Proces reg_shift_iv_p posieľá na vstup šifrovacieho komponentu Encryption inicializačný vektor v potrebnom stave a validný signál iv_rdy. Keď vstupné dáta sú prvý validný blok dát, tak šifruje sa inicializačný vektor.

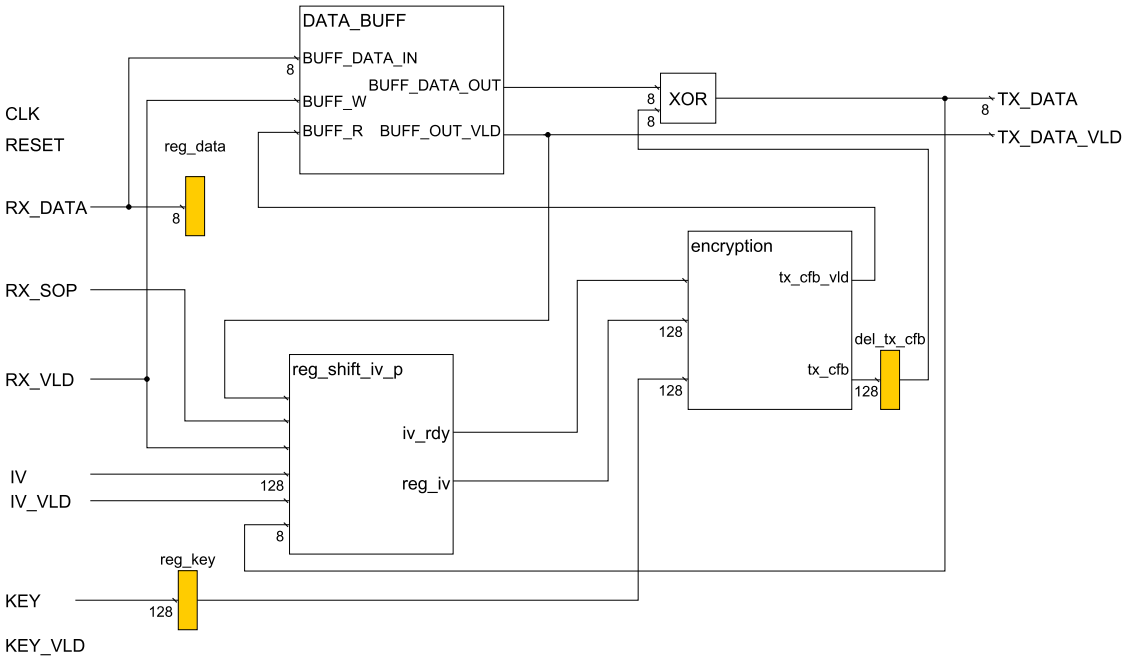
```
elsif IV_VLD = '1' and RX_SOP = '1' and RX_VLD = '1' then
       reg_iv <= IV;
       iv_rdy <= '1';
elsif buff_out_vld = '1' then
       reg_iv <= reg_iv(IV_WIDTH-DATA_WIDTH-1 downto 0) & out_data;
       iv_rdy <= '1';
```

Výpis 5.23: Časť zdrojového kódu procesu reg_shift_iv_p

Keď šifrovanie sa skončilo, dáta sú vyčítané z bufferu. Vyčítané dáta sú následne xorované s výstupom šifry. Kvôli dobrému časovaniu jednotlivých častí, zašifrované dáta sú zadržaný jeden hodinový takt. Keď na výstupu sú validné dáta, inicializačný vektor sa rotuje doľava a na miestá kam po rotácie boli napísané nuly pridajú sa výstupné dáta módu. Takto upravený inicializačný vektor je následne privedený do šifrovacej komponenty Encryption. Na zdrojovom kóde je znázornené ošetrenie týchto stavov. Na obrázku 5.11 je blokové schéma šifrovania v módu CFB.

5.5.2 Dešifrovanie

Dešifrovanie je naprogramované v zdrojovom súbore `cfb_decryption.vhd`. Chovanie návrhu je to isté ako u šifrovania. Rozdiel je v úprave inicializačného vektora. S tým, že mód je paralelizovateľný vstupy možu byť spracované hneď. Upravený inicializačný vektor je posieľaný do šifrovacieho komponentu. Kým sa šifrovanie neskončí vstupné kryptogramy sú uchované v bufferu. Po rovnakom synchronizovaní ako u šifrovaní v móde CFB sa xoruje zašifrovaný inicializačný vektor a vyčítané kryptogrami z bufferu. Na obrázku 5.12 je blokové schéma dešifrovania v módu CFB.



Obr. 5.11: Blokové schéma šifrovania v CFB móde

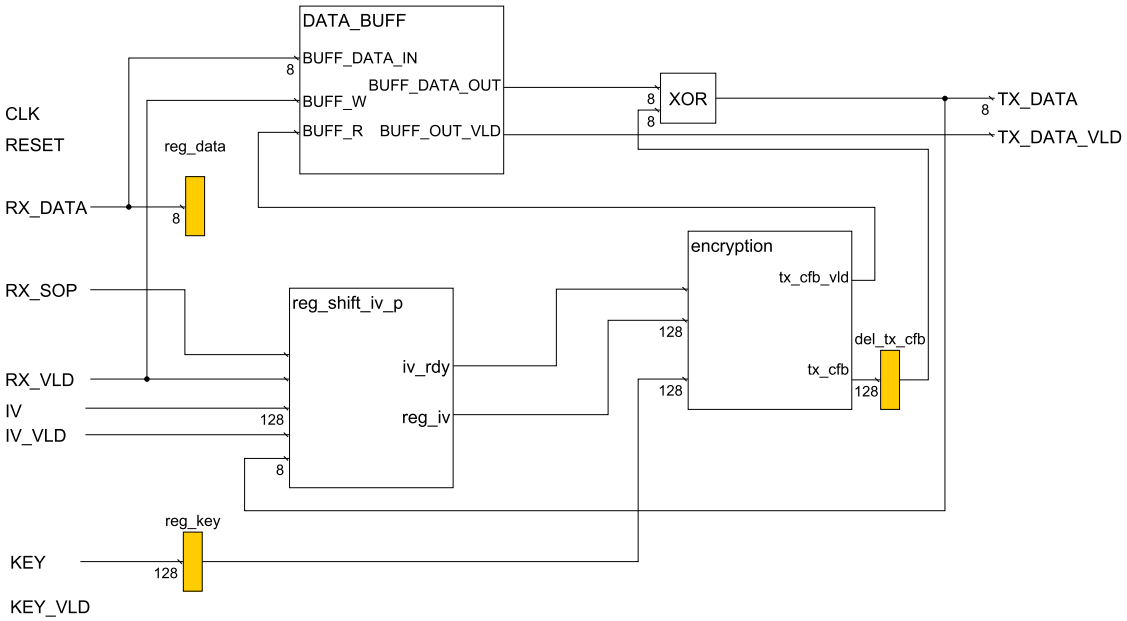
```
elsif IV_VLD = '1' and RX_SOP = '1' and RX_VLD = '1' then
       reg_iv <= IV;
       iv_rdy <= '1';
elsif RX_VLD = '1' and RX_SOP = '0' then
       reg_iv <= reg_iv(IV_WIDTH-DATA_WIDTH-1 downto 0) & reg_data;
       iv_rdy <= '1';
```

Výpis 5.24: Časť zdrojového kódu procesu reg_shift_iv_p pri dešifrovania

5.6 Naprogramovaný mód CTR

Operačný mód Counter dovoluje, aby šifrovanie a dešifrovanie bolo zrealizované rovnakou komponentou. Využíva celý návrh šifrovania a pomocnú komponentu bufferu zo sekcie 5.2.2. Na vstupné rozhranie vstupuje hodinový signál (CLK), resetujúci signál (RESET), inicializačný vektor (IV), kľúč (KEY), dáta pre šifrovanie (RX_DATA), validný signál dát (RX_VLD), kľúča (KEY_VLD), inicializačného vektora (IV_VLD) a ukazovateľ prvého bloku dát (RX_SOP).

Zachytené validné dáta na vstupoch RX_DATA, KEY a IV sú ukladané do registrov. Proces counter_p riadí šifrovanie prípadne dešifrovanie. Sú vňom ošetrené dva prípady. Prvý, keď signáli reg_vld a reg_sop majú hodnotu log. 1. To znamená, že zaregistrované vstupné dáta sú prvé validné dáta. Vtedy podľa teórie módu CTR inicializačný vektor je privedený na vstup šifrovania.



Obr. 5.12: Blokové schéma dešifrovania v CFB móde

```
elsif reg_vld = '1' and reg_sop = '1' and reg_iv_vld = '1'  then
        rx_ctr     <= reg_iv;
        rx_ctr_vld <= '1';
```

Výpis 5.25: Prvý ošetrený prípad v procese counter_p

Druhý ošetrený prípad je, keď vstupné dáta sú validné, ale už nie prvé. Vtedy sa inkrementuje hodnota signálu rx_ctr o jedno.

```
elsif reg_vld = '1' and reg_sop = '0' and reg_iv_vld = '1'  then
        rx_ctr <= std_logic_vector(unsigned(rx_ctr) + 1);
        rx_ctr_vld <= '1';
```

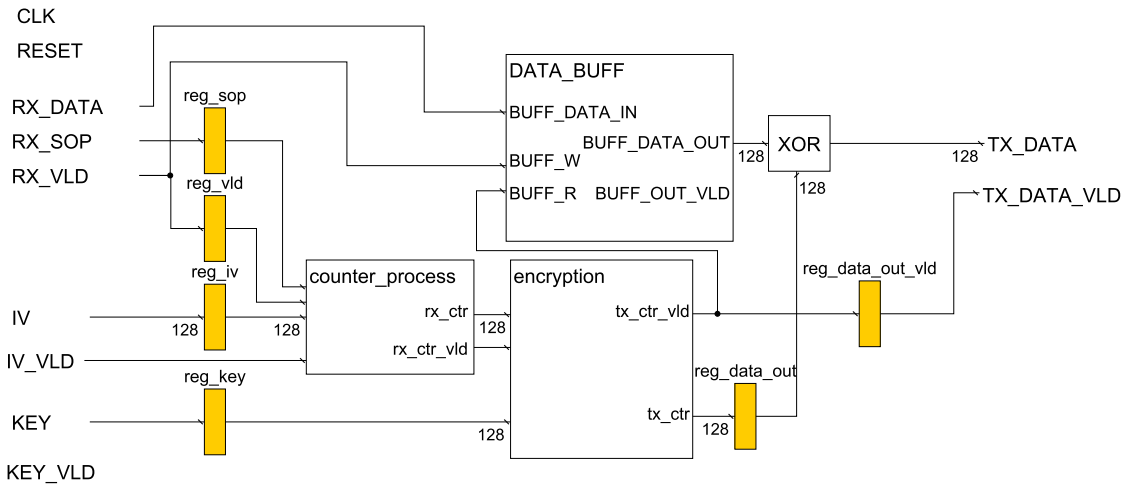
Výpis 5.26: Druhý ošetrený prípad v procese counter_p

U oboch prípadoch je veľmi dôležité, aby inicializačný vektor bol validný. Pri všetkých možných ostatných prípadoch validita signálu rx_ctr je nastavené na log. 0.

Účelom bufferu je zadržanie vstupných validných dát, kým zašifrovanie inicializačného vektora nie je hotové. Zapisovanie do bufferu je prevedené, keď na vstupe sú validné dáta. Vyčítanie, keď šifovanie sa skončilo rešp. signál tx_ctr_vld je v log. 1. S tým, že vstupné dáta do šifrovacieho komponentu, nie sú závislé na predchádzajúcich zašifrovaných dátoch, zníži vysoké kapacitné nároky bufferu. Pri maximálnom zaťažení komponentu módy CTR stačí buffer s kapacitou štrnásť 128 bitových registrov. Maximálne zaťaženie znamená, že na vstupe sú v každom hodinovom takte validné dáta a šifrovacia komponenta používa všetky pomocné registre.

Buffer na povel vyčítajúceho signála privedie na svoj výstupný port potrebné

dáta iba v nasledujúcom hodinovom takte. Aby buffer a šifrovacia komponenta bola správne synchronizovaná, je potrebné zaregistrovať validný výstup šifrovacej komponenty. Na výstupný port TX_DATA je privedný vyčítaný signál z bufferu xorováne s registrovaným výstupom šifry. Na obrázku 5.13 je blokové schéma návrhu pre CTR mód.



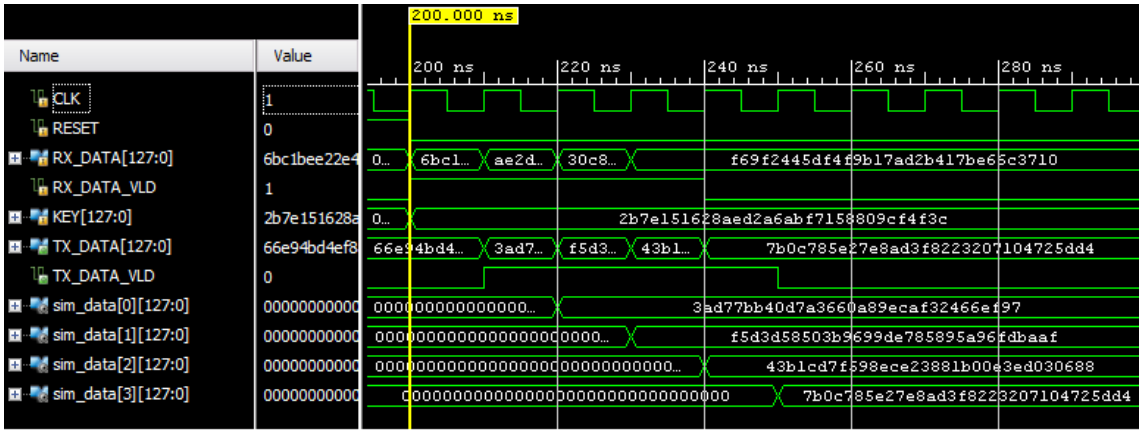
Obr. 5.13: Blokové schéma counter módu

6 SIMULÁCIA NÁVRHU ŠIFRY

Simulácie boli uskutočnené vo verziach Vivado 2013.4 a Vivado 2015.3. Pre správne otestovania funkčnost všetkých operačných módov sa používal testbench viz sekcia 4.1. V testbenchi je možné posielanie dát na jednotlivé vstupy testovanej komponenty a zachytiť výstupné dáta. Používané testovacie vektory sú stanovené v dokumente [4]. Pre šifrovanie sú používané vždy nasledujúce štyri bloky dát. Všetky dáta v texte sú vyjadrené v hexadecimálnom vyjadrení.

- 6b c1 be e2 2e 40 9f 96 e9 3d 7e 11 73 93 17 2a,
- ae 2d 8a 57 1e 03 ac 9c 9e b7 6f ac 45 af 8e 51,
- 30 c8 1c 46 a3 5c e4 11 e5 fb c1 19 1a 0a 52 ef,
- f6 9f 24 45 df 4f 9b 17 ad 2b 41 7b e6 6c 37 10.

Na nasledujúcich obrázkoch sú výsledky simulácií programu Vivado pre všetky komponenty. Do zdrojového kódu boli pridané registery sim_data_0 . . . sim_data_3, aby bol lepšie znázornení každý výstup. Program Vivado doposiaľ nepodporuje zväčšenie fontu písmen v simulovacom okne. Získané šifrované a dešifrované dáta sa zhodovali s predpokľadný dátami z [4].

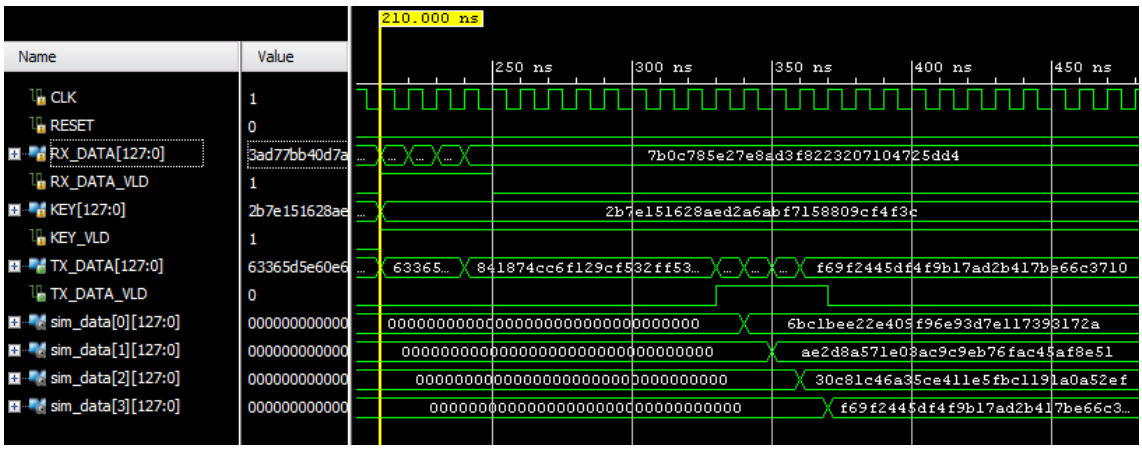


Obr. 6.1: Simulácia šifrovania v móde ECB

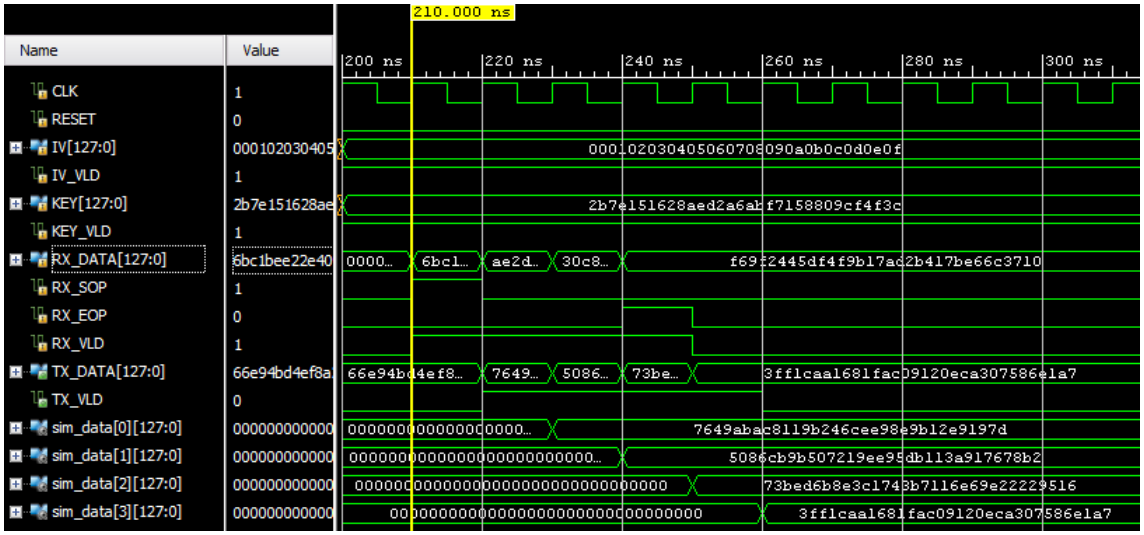
Získané zašifrované dáta v móde ECB

- `3a d7 7b b4 0d 7a 36 60 a8 9e ca f3 24 66 ef 97`,
- `f5 d3 d5 85 03 b9 69 9d e7 85 89 5a 96 fd ba af`,
- `43 b1 cd 7f 59 8e ce 23 88 1b 00 e3 ed 03 06 88`,
- `7b 0c 78 5e 27 e8 ad 3f 82 23 20 71 04 72 5d d4`,

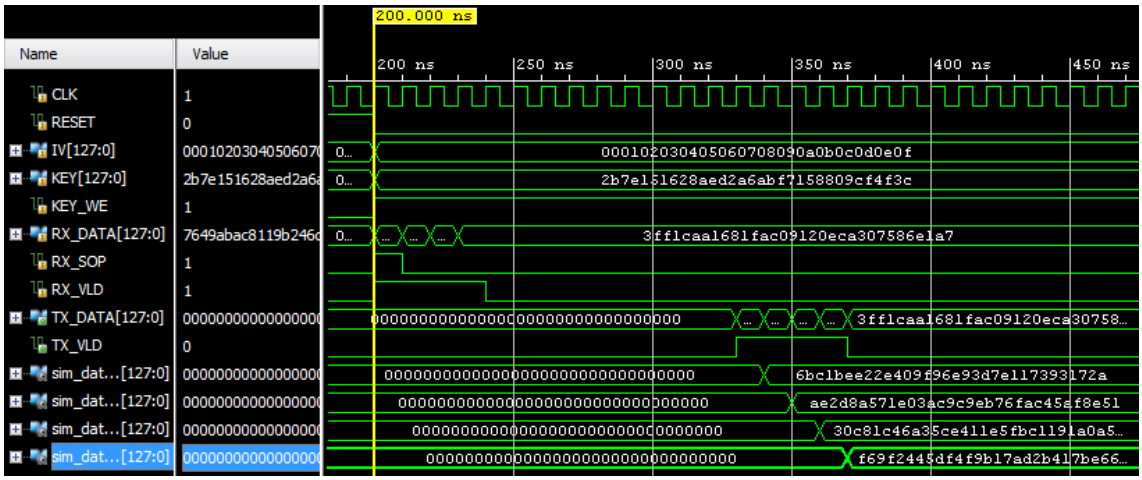
boli poslané pojednom na vstup dešifrovacej komponenty. Simulácia dešifrovania v móde ECB je znázornená na obrázku 6.2. Na nasledujúcích obrázkoch 6.3, 6.4, 6.5, 6.6, 6.7 sú znázornené simulácie pre módy CBC, CTR a CFB. Všetky zašifrované a dešifrované hodnoty dát boli správne.



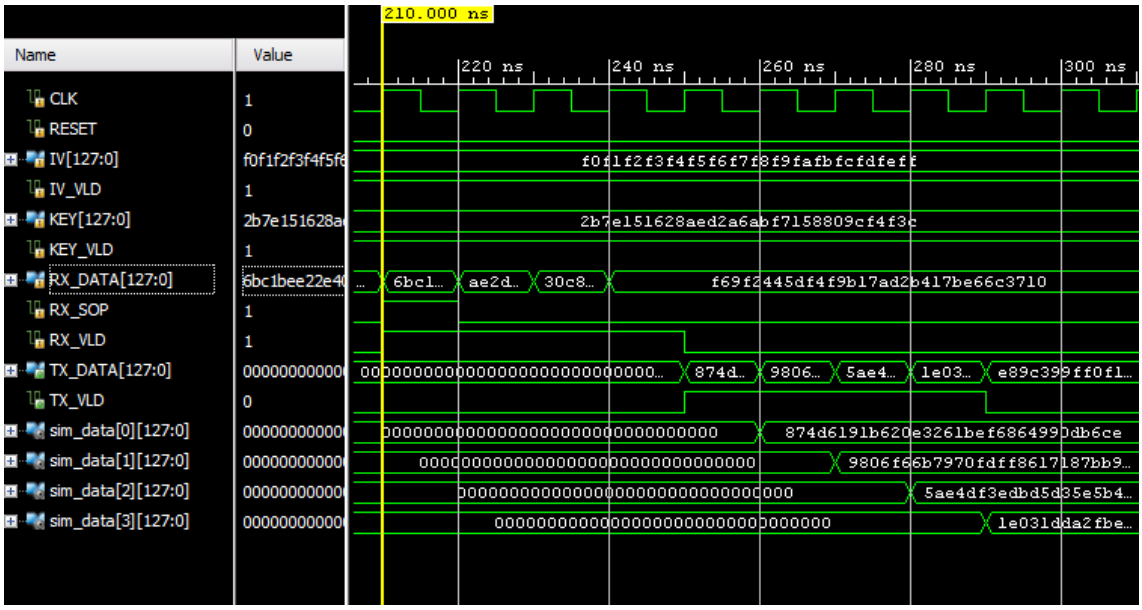
Obr. 6.2: Simulácia dešifrovania v móde ECB



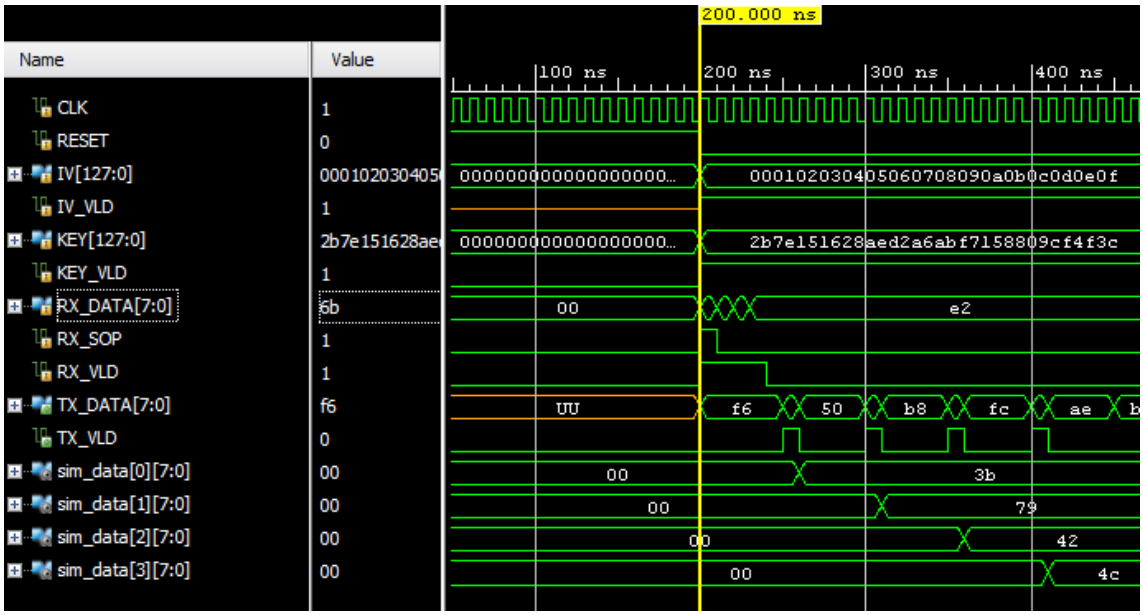
Obr. 6.3: Simulácia šifrovania v móde CBC



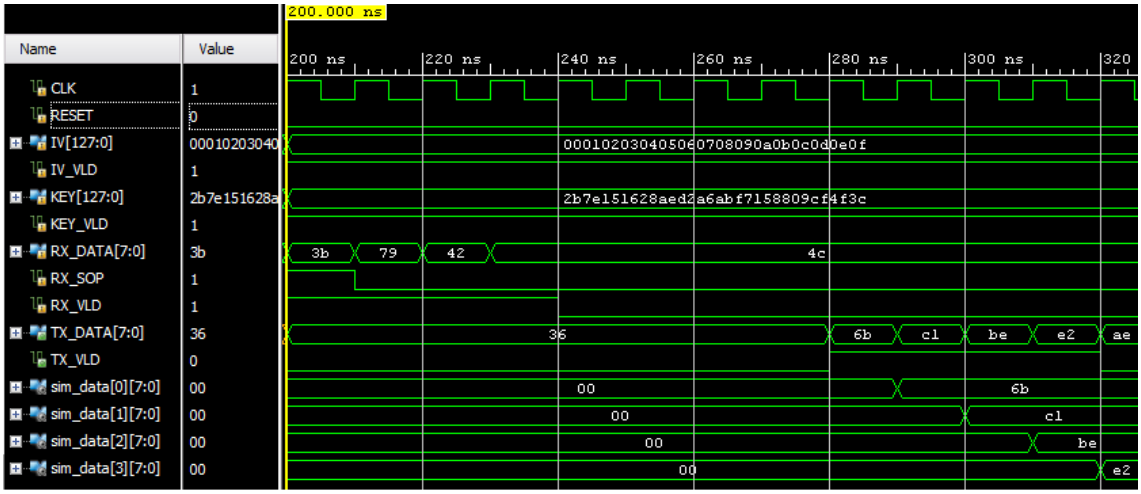
Obr. 6.4: Simulácia dešifrovania v móde CBC



Obr. 6.5: Simulácia šifrovania v móde CTR



Obr. 6.6: Simulácia šifrovania v móde CFB



Obr. 6.7: Simulácia dešifrovania v móde CFB

7 VÝSLEDKY SYNTÉZI

Podľa [5] pri popisu číslicového systému jazykom VHDL je potrebné si uvedomiť, že popisujeme číslicový systém, ktorý vo väčšine prípadov chceme realizovat v hardware. To znamená, že vytvorený kód bude potrebné vysyntetizovať. Syntéza udáva vyśledné zapojenie hradlov a klopných obvodov. Namapuje cesty medzi využitými zdrojami a vyhľadá kritické cesty logiky, kde sa nespĺňajú časové požiadávky na určenom programovateľnom logickom obvode. Nie všetky konštrukcie kódov sú syntetizovateľné. Bez chybnej syntézi nie je možné návrh implementovať na programovateľný logický obvod a práve preto je veľmi potrebné aby návrh bol syntetizovaný bez chýb.

Všetky naprogramované operačné módy boli vysyntetizované pre dva rôzne čipy FPGA. V tabuľke 7.1 sú uvedené čipy a jejich parametry.

FPGA čipy	Artix-7	Virtex-7
LUT(ks)	134600	433200
FF(ks)	269200	866400
IO(ks)	400	850
BUFG(ks)	32	32

Tab. 7.1: Parametry FPGA

V tabuľkách 7.2 a 7.3 sú uvedené výsledky syntézi pre čipy ARTIX-7 a VIRTEX-7. Znázornená je maximálna dosiahnutá frekvencia f_{max}, teoretická rýchlosť a počet používaných Look-Up Table (LUT), klopných obvodov (FF) a pomocných registrov (PIPE). Pre každý jeden operačný mód okrem šifrovania v CBC a CFB sú znázornené výsledky v dvoch riadkoch. Je uvedené zapôsobenie počtu pomocných registrov na využité zdroje čipu a hlavne na dosianhutú frekvenciu logiky. V módoch CBC a CFB šifrovanie nie je paralelizovateľný. V tomto prípade využivania rozsekania logiky na menšie časti nemá zmyseľ. Každý jeden vstup je závislý na predchádzajúcim výstupom, teda zašifrovanie 128 alebo menej bitov bude trvať stále istých počet taktov.

Z hodnôt maximálných frekvencí, je možné vypočítať teoretickú maximálnú rýchlosť komponentov podľa rovnice 7.1.

$$\textit{rýchlosť} = f_{max} \times 128 - \left(\text{PIPE} \times \frac{1}{f_{max}} \times 128\right). \tag{7.1}$$

Frekvencia vynásobená so 128 udáva rýchlosť, keď v každom hodinovom takte prichádajú na vstupu validné dáta. Vo väčšine prípadov sú používané pomocné registery. Aby výpočet bol presnejší je potrebné odpočítať prvých 10-12 taktov, kým ešte nie sú na výstupu validné dáta. Výsledky sú uvedené v merítku gigabity za sekundu.

MÓDY	ARTIX-7						
	f_{max} [MHz]	LUT	%	FF	%	PIPE	rýchlosť[Gb/s]
ECB_enc	263	9842	7,31	2828	1,05	10	33
ECB_enc	144	9474	7,04	1543	0,57	6	18,4
ECB_dec	163	12435	9,24	7245	2,69	10	20,8
ECB_dec	103	13290	9,87	7245	3,38	5	13,1
CBC_enc	28	13554	10,07	385	0,14	0	3,5
CBC_dec	153	14360	10,67	11934	4,43	10	19,5
CBC_dec	103	14456	10,74	11425	4,24	5	13,1
CFB_enc	166	9006	6,69	3485	1,29	10	0,81
CFB_dec	170	8977	6,67	3622	0,41	10	10,5
CFB_dec	107	8890	6,6	2507	0,28	5	6,8
CTR	162	11394	8,47	7610	2,83	10	20,7
CTR	105	11651	8,66	5946	2,21	4	13,4
CTR	33	12830	9,38	5046	1,87	0	4,2

Tab. 7.2: Výsledky syntézi pre FPGA ARTIX-7

MÓDY	VIRTEX-7						
	f_{max} [MHz]	LUT	%	FF	%	PIPE	rýchlosť[Gb/s]
ECB_enc	384	10040	2,32	2828	0,32	10	49,1
ECB_enc	217	9716	2,24	1543	0,18	6	27,7
ECB_dec	212	15377	3,55	7467	0,86	10	27,1
ECB_dec	133	14635	3,38	6830	0,79	5	17
CBC_enc	42	12418	2,87	385	0,04	0	5,3
CBC_dec	228	17319	4,00	12146	1,40	10	29,1
CBC_dec	136	16488	3,81	11509	1,33	5	17,4
CFB_enc	217	8743	2,01	3484	0,4	10	1,068
CFB_dec	217	8744	2,02	3492	0,41	10	13,5
CFB_dec	196	8599	1,98	2631	0,29	5	12,2
CTR	234	11214	2,59	7602	0,88	10	29,9
CTR	152	11551	2,67	5931	0,68	4	19,4
CTR	42	13811	3,19	5031	0,58	0	5,3

Tab. 7.3: Výsledky syntézi pre FPGA VIRTEX-7

7.1 Shrnutie výsledkov syntézi

Najrýchlejšiou komponentou je šifrovanie v ECB móde. Na čipe VIRTEX-7 teoreticky by dokázal zašifrovať 49,1 gigabitov za sekundu v prípade, keď počas jednej sekundy v každom hodinovom takte by prišlo na vstup 128 bitov. Na menej výkonnom čipe ARTIX-7 33 gigabitov. Pri dešifrovania dosiahnutá rýchlosť je o zhruba 45 % menšia. To spôsobuje náročnejšia logika komponentu, čo je aj vidno v riadkoch zabraných zdrojov. Mód ECB nie je postačujúce pre dňešné bezpečnostné požiadavky. Šifrovacia a dešifrovacia komponenta tohto módu tvorí jádro ostatných operačných módov.

Šifrovanie v CBC a CFB móde je výrazne pomalšia ako u ostatných komponentov. Príčinou je neparalelizovateľnosť chovania módu pri šifrovaní. U operačného módu CFB získané výsledky sú pri vstupnej dĺžke 64 bitov. Šifrovanie tohto módu je najpomaľšia zo všetkých. Výsledok z rovnice 7.1 je ešte potrebené deliť s počtom používaných pomocných registrov. Komponenta nie je schopná v každom takte priniesť na výstup zašifrované dáta. Šifrovacia rýchlosť 1,068 gigabitov za sekundu je pomerne pomalé. U dešifrovanie je už všetko počítano ako u ostatných módov.

Najrýchlejší medzi naprogramovanými módami je CTR. Uvedená rýchlosť je rovnaká pre šifrovanie a dešifrovanie.

Pri používania menšieho počtu pomocných registrov je vidno výrazný pokles rýchlosti komponentov. Pre používanie pomocných registrov je zanedbateľné, aby návrh bol paralelizovateľný. Sice kým sú na výstupe prvé zašifrované bity trvá oveľa viac taktov, ale následne v každom hodinovom takte sú na výstupe validné výsledky.

8 ZÁVER

Cieľom bakalárskej práce bolo urobiť syntetizovateľné návrhy pre jednotlivé operačné módy v jazyku VHDL. Pochopiť princíp FPGA kariet, chovanie programovacieho jazyka VHDL a ovládanie vývojového prostredia Vivado. Najprv boli preštudované symetrické blokové šifry a následne sa vybrala šifra AES. AES je zatiaľ neprelomiteľným medzinárodným štandardom pre šifrovania dát. Postupom práce je popísaný algoritmus šifry a jednotlivých častí, z ktorých sa skladá. Po teoretickej analýze boli vybraté operačné módy ECB, CBC, CFB a CTR pre návrhy v jazyku VHDL. Najprv sa overila funčnosť navrhnutého módu ECB pomocou testovaciého súboru v simulácií. ECB mód je samotná šifra a tvorí jadro ostatných operačných módov. Po naprogramovania a eliminovania všetkých chýb pomocou simulácií operačných módov v programe Vivado sa začala syntéza jednotlivých návrhov. Pre syntézu boli používané FPGA čipy Artix-7 a Virtex-7. Výsledky sú shrnuté v tabuľkách.

Z dosiahnutých výsledkov plyne, že čip Virtex-7 je výkonnejšia než Artix-7. Pri šifrovaní okrem CBC a CFB a dešifrovaní bola použitá optimalizácia pomocou pomocných registrov pre rozsekanie celej logiky na menšie časti. Najrýchlejším operačným módom medzi naprogramovanými je ECB. V prípade, keď v každom stanovenom maximálnom hodinovom takte (384 MHz pri šifrovania a 212 MHz pri dešifrovania), čo by návrh vydržal, teoretická rýchlosť šifrovania bola 49 gigabitov za sekundu a pri dešifrovania 27,1 gigabitov za sekundu. Šifrovanie v módoch CBC a CFB je výrazne pomalšia ako u ostatných. Príčinou je neparalelizovateľnosť chovania módu pri šifrovaní. Vstupy sú závislé na výstupoch šifry. To je aj vidno u teoretickej rýchlosti 5,3 gigabitov za sekundu.

Najrýchlejši operačný mód je CTR. Teoretická rýchlosť návrhu je 29,9 gigabitov za sekundu pri frekvencií 234 MHz. Tento mód využil najlepšie rýchlosť naprogramovaného AESu.

LITERATURA

[1] BURDA, Karel. *Aplikovaná kryptografie.* Brno: VUTIUM, 2013, 255 stran. ISBN 978-80-214-4612-0.

[2] FIPS-197. ADVANCED ENCRYPTION STANDARD (AES). USA: NIST, 2001. Dostupné z URL <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>

[3] DWORKIN, Morris. *Recommendation for Block Cipher Modes of Operation: Special Publication 800-38D - Galois/Counter Mode (GCM) and GMAC* [online]. National Institute of Standards and Technology, 2007, 37 stran. Dostupné z URL: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>

[4] DWORKIN, Morris. *Recommendation for Block Cipher Modes of Operation: Special Publication 800-38A - Methods and Techniques* [online]. WASHINGTON: National Institute of Standards and Technology, 2001, 66 stran. Dostupné z URL: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf>

[5] PINKER, Jiří, POUPA, Martin. *Číslicové systémy a jazyk VHDL.* Praha: BEN - technická literatura, 2006. ISBN 80-7300-198-5

[6] SMÉKAL, David. *Zabezpečení vysokorychlostních komunikačních systémů*: diplomová práce. Brno: VUT, FEKT, UTKO, 2015. 55 stran

[7] ŠŤASTNÝ, Jakub. *FPGA prakticky: realizace číslicových systémů pro programovatelná hradlová pole.* Praha: BEN - technická literatura, 2010. ISBN 9788073002619.

[8] INVEA-TECH a.s. *FPGA Framework for Rapid Development of Network Applications: Users Manual*, 2015. 86 stran.

[9] Rijndael Animation. [online], 2015 [cit. 2015-12-10]. Dostupné z URL: <http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

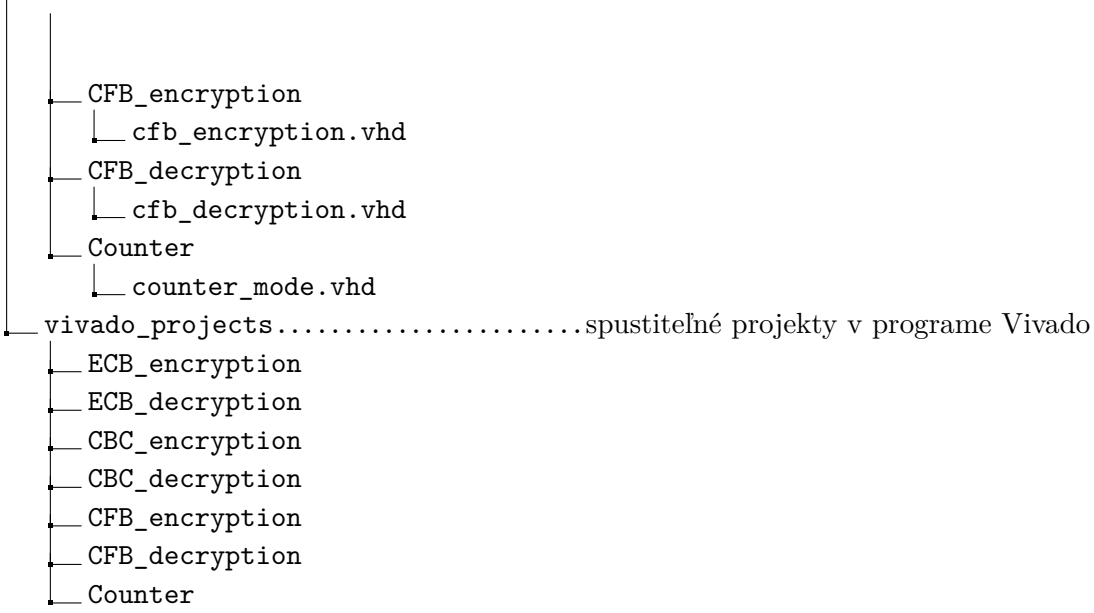
3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
CBC	operačný mód Cipher Block Chaining
CFB	operačný mód Cipher Feedback
C_i	kryptogram
CLB	Configurable Logic Block
CTR	operačný mód Counter
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
E	šifrovací proces
ECB	operačný mód Electronic Codebook
FIFO	First In First Out
FPGA	Field Programmable Gate Array
GCM	operačný mód Galois Counter Mode
GF	Galois Field
H	autentizačný kľúč
IBM	International Business Machines Corporation
IV	inicializačný vektor
K	kľúč
K_D	dešifrovací kľúč
K_E	šifrovací kľúč
log.	logická hodnota
LUT	Look Up Table
MSB	Most Significant Bit
NSA	National Security Asociation
NIST	National Institute of Standards
PCI	Peripheral Component Interconnect
PIPE	pomocný register
QSFP	Quad Small Form-factor Pluggable
RAM	Random-access memory
RSA	iniciáli autorov Rivest, Shamir, Adleman
RX	vstupný port
SBOX	Substitute BOX
SLICE	malý logický element
SOP	Start of Packet
VHDL	Very High Speed Integrated Circuit Hardware Description Language

VLD	valid
W	stĺpce kľúča
XOR	exkluzívne logické sčítanie
Z	správa

SEZNAM PŘÍLOH

A OBSAH PRILOŽENÉHO CD

- /CD_Németh ..... kořenový adresář přiloženého CD
  - constraints ..... nastavenie pariódy hodinového signálu
    - top.xdc
  - sim ..... testbench pre jednotlivé módy
    - tb_ecb_enc.vhd
    - tb_ecb_dec.vhd
    - tb_cbc_enc.vhd
    - tb_cbc_dec.vhd
    - tb_cfb_enc.vhd
    - tb_cfb_dec.vhd
    - tb_ctr.vhd
  - src ..... zdrojové súbory pre všetky operačné módy
    - ECB_encryption
      - Encryption.vhd
      - Round_1_9.vhd
      - Round_10.vhd
      - SubBytes.vhd
      - ShiftRows.vhd
      - MixColumns.vhd
      - MixColumn.vhd
      - Multiply_2.vhd
      - Multiply_3.vhd
      - KeyExpansion.vhd
      - AddRoundKey.vhd
    - ECB_decryption
      - Decryption.vhd
      - key_table.vhd
      - KeyExpansion.vhd
      - data_buff.vhd
      - Round_1_9_Decryption.vhd
      - Round_10_Decryption.vhd
      - Sub_Bytes_Decryption.vhd
      - Shift_Rows_Decryption.vhd
      - Mix_Columns_Decryption.vhd
      - Multiply_Column_Decryption.vhd
      - Multiply9.vhd
      - Multiply11.vhd
      - Multiply13.vhd
      - Multiply14.vhd
      - Add_Round_Key_Decryption.vhd
    - CBC_encryption
      - cbc_encryption.vhd
    - CBC_decryption
      - cbc_decryption.vhd


CFB_encryption
cfb_encryption.vhd
CFB_decryption
cfb_decryption.vhd
Counter
counter_mode.vhd
vivado_projects.....spustiteľné projekty v programe Vivado
ECB_encryption
ECB_decryption
CBC_encryption
CBC_decryption
CFB_encryption
CFB_decryption
Counter