VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE ŠIFROVACÍCH ALGORITMŮ V JAZYCE VHDL

IMPLEMENTATION OF ENCRYPTION ALGORITHMS IN VHDL LANGUAGE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **LUKÁŠ FRUNĚK**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. JAN KOŘENEK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Fruněk Lukáš**
Program: Informační technologie
Název: **Implementace šifrovacích algoritmů v jazyce VHDL**
Implementation of Encryption Algorithms in VHDL Language
Kategorie: Počítačová architektura

Zadání:

1. Seznamte se s jazykem VHDL a algoritmy DES a AES pro šifrování a dešifrování dat.
2. Pro obě šifry navrhněte obvodovou realizaci šifrování a dešifrování dat. Návrh optimalizujte z pohledu dosažení maximální propustnosti.
3. Proveďte implementaci navržených řešení v jazyku VHDL.
4. Nad vytvořenou implementací proveďte funkční verifikaci a syntézu do FPGA.
5. Vyhodnoťte navržená řešení z pohledu využití zdrojů FPGA a urychlení oproti softwarovým implementacím a diskutujte možnosti jejich využití.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kořenek Jan, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Tato práce se zabývá návrhem a implementací šifrovacích algoritmů DES a AES, operující v režimu CTR. Navržené moduly jsou implementovány v jazyce VHDL a slouží k umístění do programovatelných hradlových polí FPGA, konkrétně pro zařízení Intel Arria 10 SX 480. Algoritmy jsou optimalizovány s cílem dosáhnout maximální propustnosti za použití rozvinutí a vnitřního zřetězení iterací algoritmů. Navržený šifrovací modul DES dosahuje propustnosti 26.2 Gbit/s při obvodové frekvenci 410 MHz, a modul AES dosahuje propustnosti 34.6 Gbit/s při obvodové frekvenci 271 MHz, což je řádově tisícinásobné zrychlení oproti softwarovým implementacím stejných algoritmů pro vestavěné procesory.

Abstract

The thesis deals with the design and implementation of the encryption algorithms DES and AES, operating in the CTR mode. The designed modules are implemented in the VHDL language and are mapped in the FPGA Intel Arria 10 SX 480. Algorithms are optimized for maximum throughput using loop unrolling and inner pipelining. The encryption module for DES reaches throughput of 26.2 Gbit/s with the circuit operating 410 MHz, and the module for AES reaches throughput of 34.6 Gbit/s with the circuit operating at 271 MHz. The reached throughput is in the order of thousand times faster than of the same encryption algorithms implemented in software for built-in microprocessors.

Klíčová slova

šifrování, symetrická šifra, bloková šifra, DES, AES, režím čítače, CTR, FPGA, VHDL

Keywords

encryption, symmetric cipher, block cipher, DES, AES, counter mode, CTR, FPGA, VHDL

Citace

FRUNĚK, Lukáš. *Implementace šifrovacích algoritmů v jazyce VHDL*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Jan Kořenek, Ph.D.

Implementace šifrovacích algoritmů v jazyce VHDL

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Jana Kořenka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Lukáš Fruněk
5. května 2021

Poděkování

Děkuji panu doc. Ing. Janu Kořenkovi, Ph.D. za poskytnutí cenných rad, literatury a pomoci, které mi umožnily vytvořit tuto bakalářskou práci.

Obsah

Kapitola 1

Úvod

V dnešní době je stále větší množství informací přenášeno digitálně přes počítačové sítě, a rostou požadavky jak na rychlost přenosu dat, tak na jejich zabezpečení. Kvůli stále rostoucímu množství útoků na počítačové sítě se zabezpečení dat stalo samozřejmostí při vývoji téměř jakýchkoliv aplikací, které ukládají nebo odesílají citlivá data. Pro zabezpečení přenášených dat se díky jejich rychlosti používají symetrické šifrovací algoritmy. V roce 1976 byl přijat veřejný standard pro symetrický šifrovací algoritmus DES. S rostoucím výpočetní silou počítačových systému přestával DES dostačovat z hlediska bezpečnosti kvůli příliš krátkému šifrovacímu klíči, což umožňovalo prolomení šifry hrubou silou. V roce 2002 byl přijat nový standard AES, který specifikoval dostatečně bezpečný algoritmus s rozšiřitelnou délkou šifrovacího klíče, který nahradil standard DES.

Šifrovací algoritmy lze realizovat jako software běžící na mikroprocesorech, nebo pomocí obvodů implementovaných v hardware. Algoritmus realizovaný v software musí být jádrem mikroprocesoru prováděn sekvenčně, pokud neuvažujeme implementace využívající vícejádrové procesory. Nejrychlejší implementace šifrovacích algoritmů DES a AES pro vestavěné mikroprocesory dosahují propustnosti řádově desítek Mbit/s, což je pro vysokorychlostní síťovou komunikaci zcela nedostačující. Při obvodové realizaci šifrovacích algoritmů lze dosáhnout výrazně vyšší propustnosti s využitím paralelního zpracování a zřetězením datových cest. Také samotné operace šifrovacího algoritmu lze obvodově realizovat velmi efektivně, zejména bitové posuny, rotace a fixní permutace, které v hardware nespotřebují prakticky žádné zdroje. Nevýhodou takového řešení jsou vyšší náklady na vývoj a je tedy vhodné pro sériovou výrobu s velkým množstvím vyrobených kusů. Kompromis zde tvoří technologie programovatelných hradlových polí FPGA, která poskytuje jak výhody softwaru, zejména programovatelnost a snadnou modifikaci a opravu chyb, a tím nižší náklady na vývoj, tak i výhody hardwaru, zejména možnost paralelizace a zřetězení operací. Implementace šifrovacích algoritmů určené pro FPGA dosahují řádově stonásobně až tisícinásobně vyšší propustnosti řádově v desítkách Gbit/s.

Cílem této práce je návrh architektury šifrovacích algoritmů DES a AES pro FPGA zařízení Intel Arria 10 SX 480, a jejich implementace v jazyce VHDL. Šifrování bude probíhat v režimu čítače a datová komunikace bude probíhat nad rozhraním AXI4-Stream, Nad vytvořenou implementací bude provedena funkční verifikace a syntéza do cílového zařízení. Výsledná řešení budou poté srovnány se softwarovými implementacemi. Cílem je u obou algoritmů dosáhnout minimálně propustnosti 10 Gbit/s pro využití ve vysokorychlostní síťové komunikaci.

Práce je rozdělena do celkem 7 kapitol, které jsou organizovány následovně: V kapitole 2 je představena kryptografie a symetrické blokové šifrování, a je detailně popsán proces šifro-

vání a dešifrování algoritmů DES a AES. Nakonec jsou představeny základní režimy operace blokových šifer. Kapitola 3 se věnuje návrhu obvodových realizací šifrovacího procesu představených algoritmů DES a AES a optimalizaci návrhu z hlediska propustnosti. Je vytvořeno několik variant, které jsou mezi sebou po syntéze porovnány. Kapitola 4 představuje vstupní a výstupní AXI4-Stream rozhraní výsledného šifrovacího modulu, který je představen v následující kapitole 5. Tato kapitola se zabývá návrhem obvodové realizace modulu šifrování v režimu CTR pro algoritmy DES a AES, za použití komponent z kapitoly 3. V kapitole 6 je popsáno provedení funkční verifikace a syntézy šifrovacího modulu v režimu CTR pro DES i AES pro cílový obvod FPGA. Dosažené výsledky jsou pak porovnány se softwarovým řešením pro vestavěný mikroprocesor. Poslední kapitola 7 obsahuje shrnutí celé práce a nastiňuje možnosti dalšího pokračování.

Kapitola 2

Teoretický rozbor

Kryptografie [9] je obor, který se historicky zabýval pouze šifrováním, tedy utajením zpráv. Až do druhé poloviny 20. století se při vytváření šifer nebo jejich prolomení spoléhalo na kreativitu a nápaditost, a jednalo se tedy spíše o umění. Takovou kryptografii nazýváme jako klasická kryptografie. Šifry klasické kryptografie pracovaly se znaky určité abecedy, a proces šifrování a dešifrování spočíval v posunutí nebo nahrazení znaků zprávy jinými znaky. Ve druhé polovině 20. století se formuje moderní kryptografie - objevují se teoretické a matematické základy, které umožňují studovat kryptografii jako vědní obor. Kromě utajování dat se zabývá jinými problémy jako je integrita (detekce změny či poškození dat), autentizace (ověření původce zprávy), nebo digitální podpis. Moderní kryptografie pracuje se zprávou jako s posloupností bitů, tedy číslicově, a operace s těmito daty jsou čistě matematické.

2.1 Šifrování

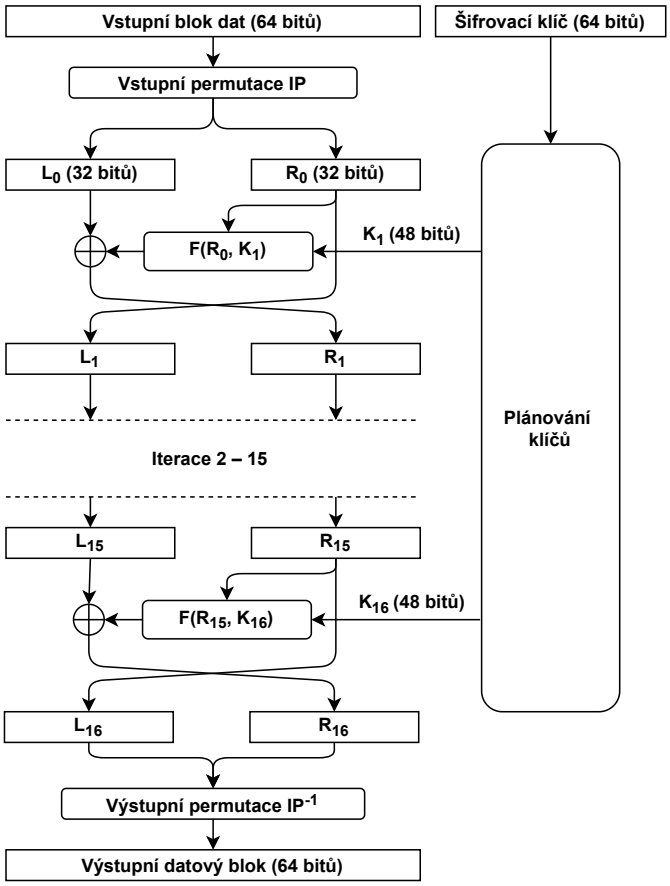
Šifrování je proces, kdy je vstupní zpráva pomocí určitého algoritmu a klíče převedena na zašifrovaný text, který je možné převést na původní text opět pomocí určitého algoritmu a klíče. Historicky spočívala bezpečnost šifry často pouze v algoritmu (např. Cézarova šifra) - znalost šifrovacího algoritmu znamenalo prolomení šifry. V 19. století formuloval Auguste Kerckhoff svůj názor na bezpečnost šifer (dnes známý jako Kerchoffův princip), který říká, že bezpečnost šifry by měla spočívat pouze v klíči, nikoliv v algoritmu. Algoritmus tedy může být veřejný, a přitom nebude možné rozšifrovat komunikaci, pokud není znám použitý klíč. Tento princip je doporučením, kterým se řídí moderní šifrovací algoritmy.

V moderní kryptografii rozlišujeme dle použitého klíče symetrické a asymetrické šifrování. Symetrické šifrování, také nazývané šifrování se soukromým klíčem, používá k šifrování i k dešifrování komunikace stejný klíč, který musejí znát pouze účastníci komunikace. Asymetrické šifrování, také šifrování s veřejným klíčem, využívá pár matematicky spojeným klíčů - soukromého a veřejného, přičemž z veřejného není reálně možné odvodit klíč soukromý. Veřejný klíč obdrží kdokoliv, kdo chce zašifrovat zprávu a poslat ji vlastníkovi klíče. Takto zašifrovaná zpráva lze rozšifrovat pouze pomocí soukromého klíče vlastníka.

2.2 Algoritmus DES

Algoritmus DES (specifikován ve standardu DES [2]) je symetrický blokový šifrovací algoritmus operující s bloky o délce 64 bitů a délkou klíče 64 bitů. Bity v bloku jsou indexovány

zleva doprava (bit nejvíce vlevo je bit 1). Algoritmus je definován šifrovací funkcí F (tzv. Feistelova funkce [7]) a funkcí plánování klíčů KS (Key Scheduling), která je podrobně popsána v kapitole 2.2.2. Plánování klíčů ze vstupního klíče vytvoří 16 podklíčů, z nichž každý slouží jako vstup odpovídající iteraci Feistelovy funkce. Na vstupní blok je aplikována úvodní permutace, poté 16 iterací Feistelovy funkce a nakonec výstupní permutace. Pro šifrování i pro dešifrování je použit stejný algoritmus, pouze podklíče se použijí v obráceném pořadí. Schéma algoritmu je zakresleno v obrázku 2.1 a podrobně popsáno v následující kapitole 2.2.1.



Obrázek 2.1: Schéma algoritmu DES

2.2.1 Proces šifrování

Vstupní permutace

Proces šifrování začíná aplikací vstupní permutace IP (Initial permutation) na vstupní 64-bitový blok. Permutace je znázorněna v tabulce 2.1, kde 1. řádek odpovídá výstupním bitům 1 až 8, 2. řádek bitům 9 až 16, atd., a čísla v tabulce jsou indexy vstupních bitů. Aplikaci vstupní permutace lze vyjádřit jako $ip\_output[i] = input[IP[i]]; \quad i \in \{1, 2, ..., 64\}$, takže např. 1. bit výstupu z permutace je 58. bit ze vstupu, 2. bit výstupu je 50. bit ze vstupu, až 64. bit výstupu je 7. bit ze vstupu.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Tabulka 2.1: Permutace IP

Iterace

Výstup z úvodní permutace je dále rozdělen na dvě 32-bitové poloviny, ve schématu 2.1 označené jako L_0 a R_0. Tyto pak projdou 16 iteracemi Feistelovy funkce. Každá iterace Feistelovy funkce bere jako vstup dvě 32-bitové hodnoty, ve schématu označené jako L_{r-1} a R_{r-1}, a 48-bitový podklíč, označený jako K_r, kde $r \in \{1, 2, ..., 16\}$ udává číslo iterace. Výstupem každé iterace jsou opět dvě 32-bitové hodnoty L_r a R_r, které jsou získány následovně: Pravá polovina R_{r-1} je přímo přiřazena levé polovině výstupu L_r. Do pravé poloviny výstupu R_r je přiřazen exkluzivní logický součet (XOR) levé poloviny vstupu L_{r-1} a výstupu šifrovací funkce $F(R_{r-1}, K_r)$. Průběh iterace je vyjádřena následujícími vztahy 2.1 a 2.2 a ve schématu je zakreslena 1. a 16. iterace Feistelovy funkce.

$$L_r = R_{r-1} \tag{2.1}$$

$$R_r = L_{r-1} \oplus F(R_{r-1}, K_r) \tag{2.2}$$

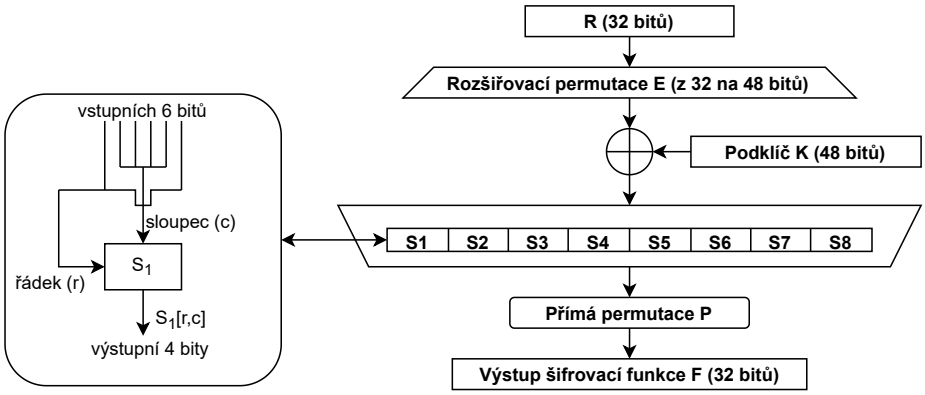
Šifrovací funkce $F(R, K)$

Šifrovací funkce F je znázorněna na obrázku 2.2 aává ze 3 částí. První část je rozšiřovací permutace E, která rozšíří vstup R ze 32 bitů na 48 bitů. Permutace je znázorněna obrázkem 2.3, kde nahoře jsou jednotlivé bity ze vstupu R, a dole jsou jednotlivé bity výstupu rozšiřovací permutace E. Permutace E je provedena tak, že každé 4 bity ze vstupu R se rozšíří na 6 bitů následovně: tyto 4 bity jsou zkopírovány na bity 2,3,4,5 v odpovídající výsledné šestici, a 1. bit v šestici je zkopírovaný 4. bit předchozí vstupní čtveřice a 6. bit v šestici je zkopírovaný 1. bit následující vstupní čtveřice. Pro vstupní čtveřici 1,2,3,4 je předchozí čtveřicí poslední vstupní čtveřice 29,30,31,31 a naopak, pro čtveřici 29,30,31,32 je následující čtveřice 1,2,3,4. Takže např. výstupním bitům 1 až 6 odpovídají po řadě vstupní bity 32,1,2,3,4,5.

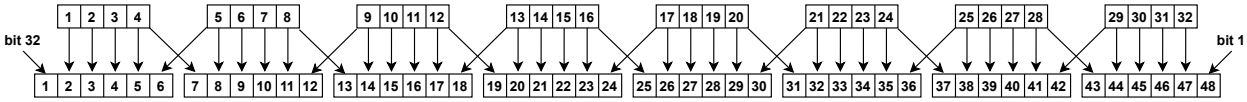
Pak se provede operace XOR s výstupem permutace E a podklíčem K pro tuto iteraci. Nad 48-bitovým výstupem operace XOR je následně provedena substituce, jejímž výsledkem je 32 bitová hodnota. Substituce sestává z osmi substitučních funkcí (zvané S-boxy) S_1,S_2,S_3,S_4,S_5,S_6,S_7,S_8 a každá z nich je definována tabulkou o 4 řádcích a 16 sloupcích. Vstupních 48 bitů se rozdělí na 8 částí po 6 bitech, které tvoří vstupy příslušných substitučních funkcí (např. bity 1 až 6 jsou vstupem funkce S_1, bity 7 až 12 jsou vstupem funkce S_2, atd.). Každá substituční funkce pak transformuje 6-bitový vstup na 4-bitový výstup následovně: Bity 1 a 6 ze vstupu tvoří dohromady 2-bitové číslo, které nabývá hodnoty 0 až 3. Tato hodnota je použita jako index řádku do tabulky dané substituční funkce. Vstupní bity

2,3,4,5 pak dohromady tvoří 4-bitové číslo, které nabývá hodnoty 0 až 15 – tato hodnota je použita jako sloupec do tabulky substituční funkce. Na takto získaném řádku a sloupci se nachází 4-bitová hodnota, která je výstupem daného S-boxu. Pro příklad je uvedena tabulka substituční funkce S_1, ostatní tabulky pro S_2 až S_8 jsou uvedeny v příloze standardu DES [2]. Například vstupní 6-bitová hodnota 100101 je nahrazena hodnotou 8 (binárně 1000) z řádku 11 (3) a sloupce 0010 (2).

Výstupem substitučních funkcí je 8 čtyřbajtových hodnot, které dohromady dávají 32-bitový výstup substituce. Nad touto hodnotou je nakonec provedena přímá permutace P prostým přeuspořádáním hodnot dle tabulky 2.3. Tedy 1. bit výstupu z permutace je 16. bit ze vstupu, 2. bit výstupu je 7. bit vstupu atd. Výstup permutace P je výstupem šifrovací funkce F.



Obrázek 2.2: Schéma šifrovací funkce F



Obrázek 2.3: Schéma rozšiřovací permutace E

S_1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Tabulka 2.2: Tabulka pro substituční funkci S_1

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

Tabulka 2.3: Permutace P

2.2.2 Plánování klíčů

Algoritmus plánování klíčů (KS) vygeneruje ze vstupního 64-bitového klíče 16 48-bitových podklíčů pro jednotlivé iterace Feistelovy funkce. Ve vstupním klíči je každý 8. bit paritní – při vygenerování klíče je dopočítán tak, aby každý bajt klíče měl sudou paritu. Tyto bity mohou sloužit ke kontrole chyb při ukládání a distribuci klíčů. Efektivní délka šifrovacího klíče pro DES je tedy 56 bitů.

Algoritmus plánování klíčů sestává ze vstupního permutačního výběru PC-1 (permuted choice) a 16 iterací, ve kterých se provádějí levé bitové rotace a permutační výběr PC-2. Schéma plánování klíčů je zakresleno na obrázku 2.4. Prvním krokem je permutační výběr PC-1, který je dán tabulkou 2.4 a provádí se stejně jako permutace IP a IP^{-1} uvedené v kapitole 2.2.1. Kromě záměny pozic bitů permutační výběr PC-1 také vynechá paritní bity (bity 8,16,24,32,40,48,56 a 64), výsledkem PC-1 je tak 56-bitová hodnota. Následně se tato 56-bitová hodnota rozdělí na dvě 28-bitové poloviny, ve schématu označené jako C_r a D_r, kde $r \in \{1, 2, ..., 16\}$. Poté následuje 16 iterací, kdy v každé se nejprve provede bitová rotace hodnot C_r a D_r (ve standardu [2] se označují jako *shifts*, ale významově se jedná o rotace). Rotace v každé iteraci se provede o 1 nebo 2 bitové pozice doleva, dle tabulky 2.6. Výsledky rotace C_r a D_r jsou pak vstupy do následující iterace. Na závěr každé iterace se výsledky rotace C_r a D_r spojí a provede se nad nimi permutační výběr PC-2. Ten je dán tabulkou 2.5 a provádí se stejně jako PC-1. Kromě prohození pořadí navíc vynechá 8 bitů ze vstupu (bity 9,18,22,25,35,38,43 a 54), výsledkem PC-2 je tak 48-bitová hodnota. Tato hodnota, ve schématu označená jako K_r, je podklíčem použitým v odpovídající iteraci r Feistelovy funkce při šifrování.

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Tabulka 2.4: Permutační výběr PC-1

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

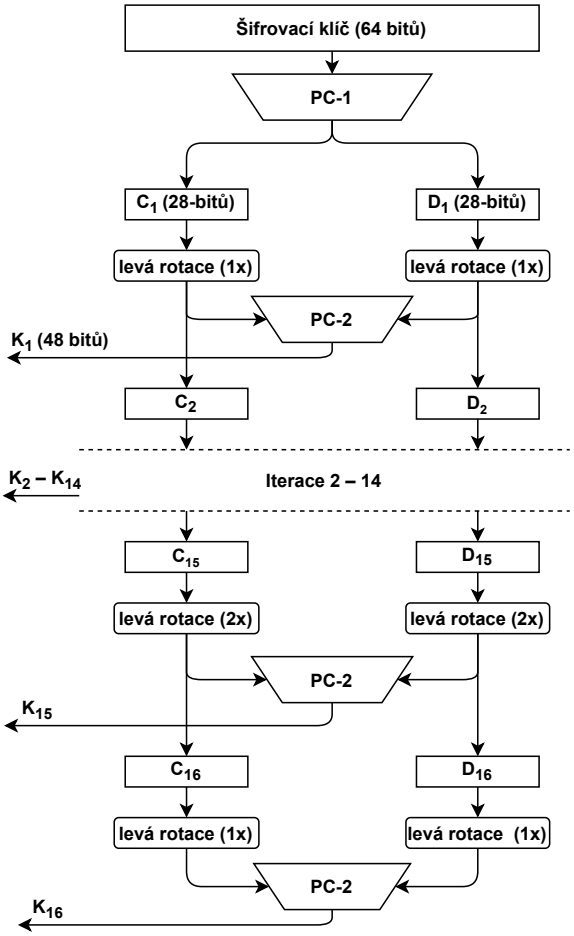
Tabulka 2.5: Permutační výběr PC-2

Číslo iterace	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Počet levých rotací	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabulka 2.6: Rotace pro iterace plánování klíčů

2.2.3 Proces dešifrování

Pro dešifrování se použije stejný algoritmus jako pro šifrování, pouze podklíče K_1 až K_{16} jsou aplikovány v opačném pořadí, tedy pro 1. iteraci Feistelovy funkce se použije podklíč K_{16}, pro 2. iteraci podklíč K_{15} až pro 16. iteraci se použije první podklíč K_1. Algoritmus pro dešifrování nebude v této práci použitý, protože vybraný režim šifrování CTR pracuje pouze s šifrovací částí vybraného algoritmu, jak je blíže popsáno v podkapitole 2.4.



Obrázek 2.4: Schéma plánování klíčů

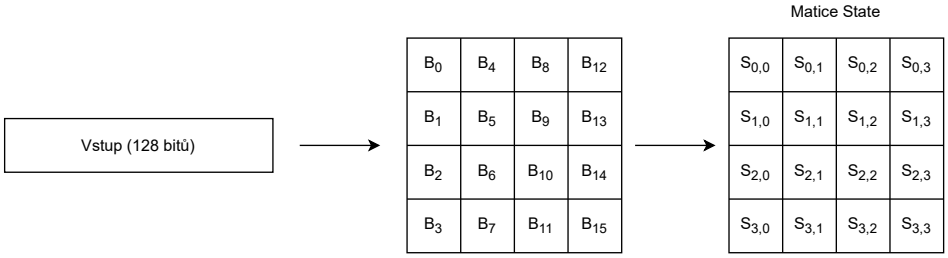
2.3 Algoritmus AES

Algoritmus AES (nebo Rijndael), specifikován ve standardu AES [3], je stejně jako DES symetrický blokový šifrovací algoritmus. Pracuje s bloky o délce 128 bitů a mohou být použity klíče délky 128, 192 nebo 256 bitů. Tyto varianty AES se pak označují jako AES-128, AES-192 a AES-256. Liší se od sebe počtem prováděných iterací a generováním podklíčů. Algoritmus byl navržen tak, aby dokázal operovat s různými délkami datového bloku a klíče. Tato práce se zabývá implementací AES-128, proto bude v této kapitole popsána pouze varianta používající 128-bitový klíč.

2.3.1 Proces šifrování

Vstupní 128-bitový datový blok je reprezentován jako 4x4 matice bajtů, označená jako *State*. Reprezentace je znázorněná na následujícím obrázku 2.5, kde B_0 až B_{15} odpovídají 1. až 16. bajtu vstupního datového bloku, které pak odpovídají položkám $S_{r,c}$ matice *State*, kde r je řádek a c je sloupec matice (indexováno od 0 do 3).

Matice *State* je v průběhu šifrování postupně transformována operacemi *SubBytes(State)*, *ShiftRows(State)*, *MixColumns(State)* a *AddRoundKey(State, Key)*. Operace *AddRoundKey*

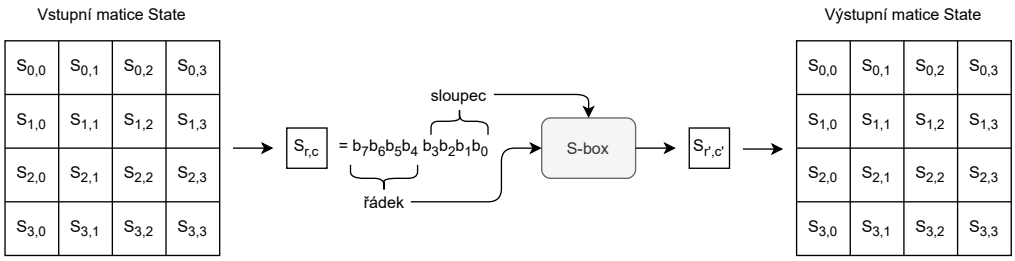


Obrázek 2.5: Reprezentace datového bloku maticí *State*

navíc při transformaci využívá 128-bitový klíč (*Key*), buď přímo vstupní šifrovací klíč, anebo získaný algoritmem Rozšíření klíče, který je popsaný v kapitole 2.3.2. Následuje popis těchto 4 operací, a posléze je detailně popsán proces šifrování.

Operace substituce bajtů (SubBytes)

Operace *SubBytes* nahradí hodnotu každého bajtu vstupní matice *State* jinou hodnotou následovně: Vstupní bajt je rozdělen na dvě poloviny, hodnota vrchních 4 bitů pak tvoří index řádku a spodní 4 bity tvoří index sloupce do substituční tabulky (S-boxu), která je uvedena na následující straně 2.7. Hodnota nacházející se na daném řádku a sloupci je hodnotou nahrazeného bajtu ve výstupní matici *State*. Například bajt s hodnotou `0x4d` je nahrazen hodnotou v řádku `4` a sloupci `d`, tedy hodnotou e3. Postup operace *SubBytes* je znázorněn v následujícím obrázku 2.6.



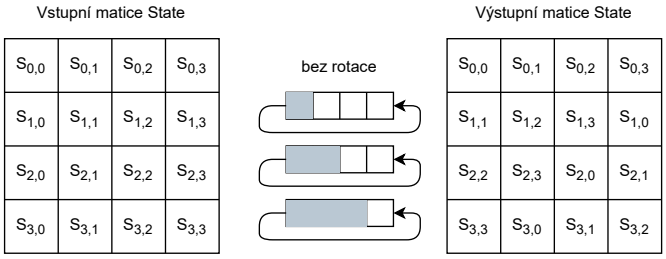
Obrázek 2.6: Operace SubBytes

S-box	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	c1	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

Tabulka 2.7: Substituční tabulka (S-box) operace SubBytes (hexadecimální hodnoty)

Operace rotace řádků (ShiftRows)

Operace ShiftRows je schematicky znázorněna na obrázku 2.7. Operace provede v každém řádku matice State cyklický posun (rotaci) jednotlivých bajtů směrem doleva. Počet posunutí odpovídá indexu řádku, tedy 1. řádek (s indexem 0) není posunut vůbec, 2. řádek se posune o 1 pozici, 3. řádek o 2 a 4. řádek o 3 pozice.



Obrázek 2.7: Operace ShiftRows

Operace míchání sloupců (MixColumns)

Operace MixColumns se dívá na matici *State* jako na čtveřici sloupců o velikosti 4 bajty, a v každém sloupci $c \in \{0, 1, 2, 3\}$ je každý bajt nahrazen novou hodnotou vypočítanou z hodnot všech 4 bajtů tohoto sloupce c. Tuto operaci lze zapsat jako maticové násobení (rovnice 2.3), avšak namísto operace sčítání se použije operace XOR, a operace násobení je popsána dále. Bajty každého sloupce $c \in \{0, 1, 2, 3\}$ jsou tedy nahrazeny novými hodnotami dle následujících vztahů:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{2.3}$$

$$S'_{0,c} = (2 \cdot S_{0,c}) \oplus (3 \cdot S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \tag{2.4}$$
$$S'_{1,c} = S_{0,c} \oplus (2 \cdot S_{1,c}) \oplus (3 \cdot S_{2,c}) \oplus S_{3,c} \tag{2.5}$$
$$S'_{2,c} = S_{0,c} \oplus S_{1,c} \oplus (2 \cdot S_{2,c}) \oplus (3 \cdot S_{3,c}) \tag{2.6}$$
$$S'_{3,c} = (3 \cdot S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (2 \cdot S_{3,c}) \tag{2.7}$$

Násobení (operace $\cdot$) v *MixColumns* je definováno jinak než běžné bitové násobení. Násobí se mezi sebou vždy 8-bitová hodnota (bajt v matici State), a 8-bitová konstanta (při šifrování jsou to konstanty `0x01`, `0x02` a `0x03`). Násobení jakoukoliv konstantou se implementuje pomocí násobení konstantou `0x02` a jejích mocnin (mezi než počítáme i konstantu $2^0 = 1$), a sečtením mezivýsledků násobení pomocí XOR, následovně: Při násobení konstantou `0x01` je stejně jako při běžném násobení výsledkem původní hodnota. Operace násobení konstantou `0x02` se ve standardu nazývá *xtimes*, a je implementována jako bitový posun doleva s podmíněným následným XOR s hodnotou 0x11b, pokud výsledek přeteče nad 8 bitů (takže ve vstupní hodnotě má bit na indexu 7 hodnotu 1), tedy následovně dle vztahů 2.8 a 2.9:

$$B[7] = 0 \implies xtimes(B) = B << 1 \tag{2.8}$$
$$B[7] = 1 \implies xtimes(B) = (B << 1) \oplus 11b \tag{2.9}$$

Tímto se zajistí oříznutí výsledku na 8 bitů v případě přetečení. Násobení mocninami dvojky se provede opakovanou aplikací *xtimes*, příklad pro násobení hodnoty `0x5a` konstantou 8 je uvedený v 2.10. Jiné konstanty se rozloží na součet mocnin dvojky, a provede se součet (XOR) mezivýsledků násobení. Příklady pro konstanty `3` a `7` jsou uvedeny v rovnicích 2.11 a 2.11.

$$5a \cdot 8 = xtimes(xtimes(xtimes(5a))) = e6 \tag{2.10}$$
$$5a \cdot 3 = 5a \cdot (2 \oplus 1) = xtimes(5a) \oplus 5a = ee \tag{2.11}$$
$$5a \cdot 7 = 5a \cdot (4 \oplus 2 \oplus 1) = (5a \cdot 4) \oplus (5a \cdot 2) \oplus 5a = 9d \tag{2.12}$$

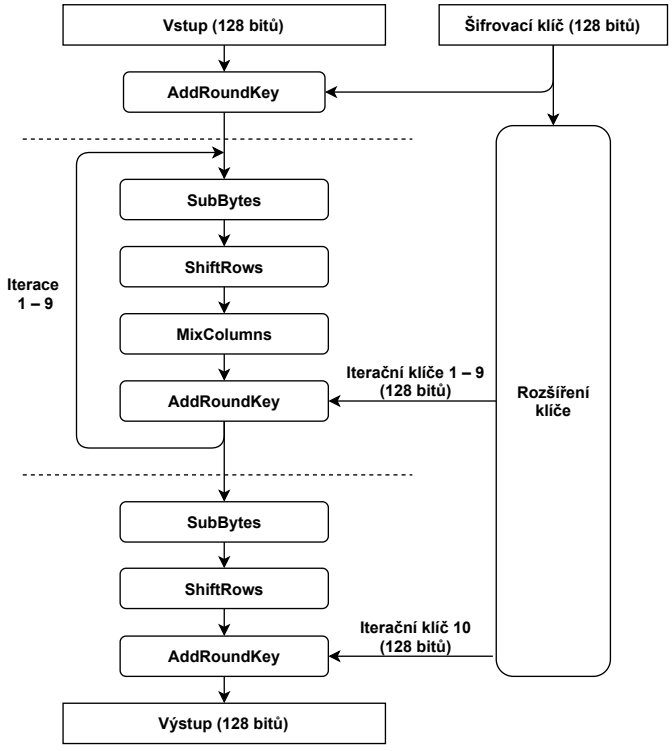
Operace přičtení podklíče (AddRoundKey)

Vstupem operace *AddRoundKey* je matice *State* a 128-bitová hodnota podklíče. Ten je reprezentován pomocí čtyř 32-bitových slov nazvaných w_0, w_1, w_2 a w_3 (tato reprezentace je ukázána v kapitole 2.3.2 na obrázku **??**). Mezi těmito slovy klíče a sloupci ve vstupní matici, které se označí jako $[S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}]$, kde $c \in \{0, 1, 2, 3\}$, se provede operace XOR, jejíž výsledky tvoří sloupce ve výsledné matici *State*, které se označí jako $[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}]$. Operaci *AddRoundKey* vyjadřuje následující vztah:

$$[S'_{0,c}, S'_{1,c}, S'_{2,c}, S'_{3,c}] = [S_{0,c}, S_{1,c}, S_{2,c}, S_{3,c}] \oplus w_c \quad c \in \{0, 1, 2, 3\} \tag{2.13}$$

Šifrování

Proces šifrování je znázorněn ve schématu 2.8 a probíhá následovně: Na začátku se provede operace *AddRoundKey* přímo nad vstupním datovým blokem a nad vstupním šifrovacím klíčem. Výsledek pak projde postupně 10 iteracemi. Prvních 9 iterací (ve schématu znázorněné jako cyklus mezi dvěma přerušovanými čarami), provedou postupně transformace *SubBytes*, *ShiftRows*, *MixColumns* a *AddRoundKey*, která použije podklíč pro příslušnou iteraci vygenerovaný algoritmem plánování klíčů, který je popsán v následující kapitole. Poslední 10. iterace provede pouze transformace *SubBytes*, *ShiftRows* a *AddRoundKey* s použitím posledního 10. podklíče, vynechá tedy transformaci *MixColumns*. Výstupem poslední iterace je 128-bitový zašifrovaný datový blok.

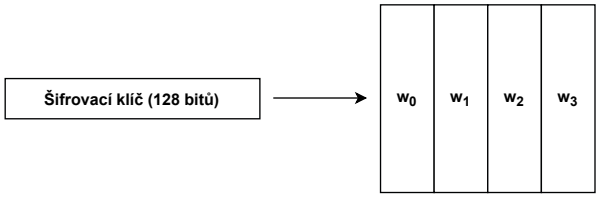


Obrázek 2.8: Schéma šifrování AES

2.3.2 Rozšíření klíče

V algoritmu rozšíření klíče (Key Expansion) se vstupní 128-bitový šifrovací klíč rozloží na 4 sloupce 32-bitových slov označené jako w_0, w_1, w_2 a w_3, kde w_0 odpovídá bitům 0 až 31 šifrovacího klíče, w_1 bitům 32 až 63, w_2 bitům 64 až 95 a w_3 bitům 96 až 127. Tuto reprezentaci ukazuje obrázek 2.9.

Tyto 4 slova jsou použita v operaci *AddRoundKey* v prediterační fázi šifrování (viz schéma 2.8). Algoritmus rozšíření klíče pak z těchto 4 slov postupně vygeneruje dalších 40 32-bitových slov, které odpovídají 10 podklíčům pro jednotlivé iterace šifrování. Vygenerovaná slova w_4, w_5, w_6, w_7 odpovídají prvnímu podklíči, slova w_8, w_9, w_{10}, w_{11} druhému podklíči, až poslední 4 slova w_{40}, w_{41}, w_{42}, w_{43} odpovídají poslednímu 10. podklíči. Generování slov se provede následovně: Každé slovo w_i, jehož index je násobkem 4 (tedy slova



Obrázek 2.9: Rozložení klíče na 32-bitová slova

w_4, w_8, w_{12}, ..., až w_{40}) se vypočítá ze slova předchozího (w_{i-1}) a ze slova o 4 pozice nazpět (w_{i-4}) dle vztahu 2.14. Při tomto výpočtu je použita operace XOR, konstanta z tabulky *Rcon*, operace *RotWord* a operace *SubWord*.

$$w_i = SubWord(RotWord(w_{i-1})) \oplus Rcon[i/4] \oplus w_{i-4} \quad i \in \{4, 8, ..., 40\} \tag{2.14}$$

Tabulka *Rcon* je uvedena v 2.8 a obsahuje 10 32-bitových konstant pro každý generovaný podklíč. Výraz $Rcon[i/4]$ ve vztahu 2.14 značí hodnotu z *Rcon* na indexu $i/4$, což zde znamená celočíselné dělení. Tedy například pro slova w_4, w_5, w_6 a w_7, která tvoří první podklíč, je hodnota indexu $i/4 = 1$, tedy všechna použijí při výpočtu konstantu $Rcon[1]$. Operace *RotWord* rozdělí vstupní 32-bitové slovo w na jednotlivé bajty, a provede rotaci pozic bajtů slova o 1 doleva. Tyto bajty tvoří výstupní slovo w' operace *RotWord*, zapsaná v následujícím vztahu 2.15:

$$RotWord(w = [B_0, B_1, B_2, B_3]) = [B_1, B_2, B_3, B_0] = w' \tag{2.15}$$

Index	32-bitová hodnota
1	01 00 00 00
2	02 00 00 00
3	04 00 00 00
4	08 00 00 00
5	10 00 00 00
6	20 00 00 00
7	40 00 00 00
8	80 00 00 00
9	1b 00 00 00
10	36 00 00 00

Tabulka 2.8: Hodnoty tabulky Rcon (hexadecimálně)

Operace *SubWord* rozdělí vstupní 32-bitové slovo na jednotlivé bajty a nahradí je novými hodnotami stejně jako operace *SubBytes*, která je popsaná v předchozí podkapitole. Operaci *SubWord* můžeme vyjádřit následujícími vztahy, kde w je vstupní slovo, a pro každý bajt B_i odpovídá hodnota r_{Bi} vrchním 4 bitům, tedy $r_{Bi} = b_7b_6b_5b_4$, a hodnota c_{Bi} odpovídá spodním 4 bitům, tedy $c_{Bi} = b_3b_2b_1b_0$. Výraz $Sbox[r_{Bi}, c_{Bi}]$ pak označuje hodnotu z tabulky S-box 2.7 v řádku r_{Bi} a sloupci c_{Bi}, a w' značí výstupní 32-bitové slovo.

$$SubWord(w = [B_1, B_2, B_3, B_4]) = [S(B_1), S(B_2), S(B_3), S(B_4)] = w' \tag{2.16}$$
$$S(B_1) = Sbox[r_{B1}, c_{B1}] \tag{2.17}$$
$$S(B_2) = Sbox[r_{B2}, c_{B2}] \tag{2.18}$$
$$S(B_3) = Sbox[r_{B3}, c_{B3}] \tag{2.19}$$
$$S(B_4) = Sbox[r_{B4}, c_{B4}] \tag{2.20}$$

Výpočet ostatních slov w_i, tedy slov, jejichž index není násobek 4, se provede pouze jako XOR předchozího slova (w_{i-1}) a slova o 4 pozice nazpět (w_{i-4}), což je vyjádřeno následujícím vztahem 2.21:

$$w_i = w_{i-1} \oplus w_{i-4} \quad i \in \{5, 6, 7, 9, 10, 11, ..., 41, 42, 43\} \tag{2.21}$$

Každá čtveřice 32-bitových slov vygenerovaných algoritmem rozšíření klíče pak odpovídá jednomu iteračnímu podklíči. Pokud podklíč pro iteraci r označíme K_r, pak můžeme výsledné 128-bitové podklíče vyjádřit pomocí vygenerovaných slov w_i ($i \in \{4, 5, 6, ..., 43\}$) následujícím vztahem 2.22:

$$K_r = [w_r, w_{r+1}, w_{r+2}, w_{r+3}] \quad r \in \{1, 2, 3, ..., 10\} \tag{2.22}$$

2.3.3 Proces dešifrování

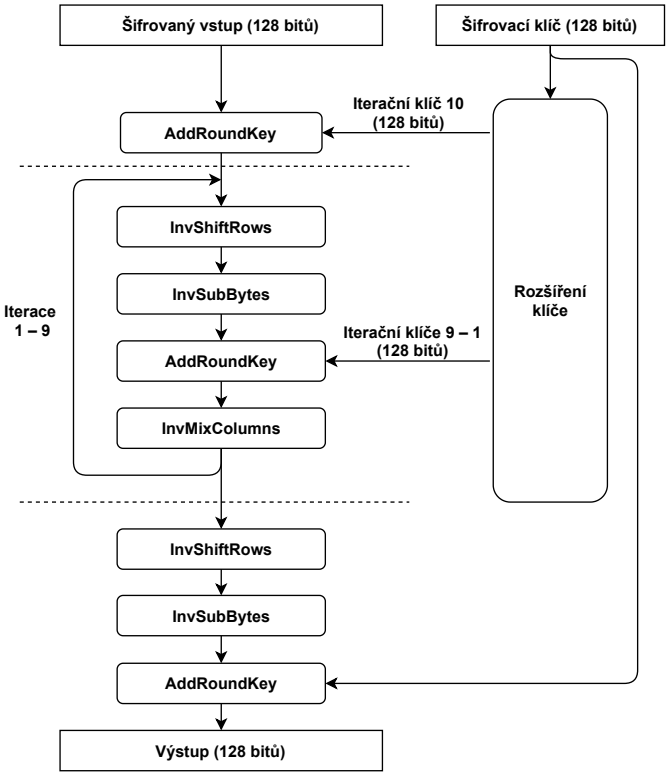
Dešifrování v algoritmu AES je inverzní k procesu šifrování. Algoritmus plánování klíčů zůstává stejný, ale zašifrovaný datový blok prochází transformacemi v opačném pořadí než při šifrování (tedy od konce na začátek) s tím, že se použijí transformace inverzní k těm původním. Ty se nazývají *InvSubBytes*, *InvShiftRows* a *InvMixColumns*. Transformace *AddRoundKey* zůstává stejná, protože operace XOR, kterou provádí, je inverzní sama k sobě.

Transformace *InvSubBytes* stejně jako *SubBytes* každý bajt matice *State* nahradí novou hodnotou. Tuto hodnotu získá ze substituční tabulky, která je inverzní k substituční tabulce S-box 2.7 pro *SubBytes*. Tento inverzní S-box je z původního vypočten tak, že hodnota z původního S-boxu udává řádek a sloupec v inverzním S-boxu, a řádek a sloupec v původním S-boxu dává dohromady hodnoty v inverzním S-boxu. Například v S-boxu je v řádku 3 a sloupci 1 hodnota $c7$, pak v inverzním S-boxu je v řádku c a sloupci 7 hodnota 31. Transformace *InvShiftRows* provede rotace řádků matice *State* stejně jako *ShiftRows*, ale opačným směrem (doprava). První řádek je tedy bez rotace, druhý o 1 pozici, třetí o 2 pozice a čtvrtý o 3 pozice doprava. Transformace *InvMixColumns* provádí transformaci matice *State* stejným způsobem jako *MixColumns*, pouze s použitím jiných konstant $0e$, $0b$, $0d$ a 09. Operace je vyjádřena následujícím vztahem násobení matic 2.23, kde $S_{r,c}$ jsou bajty vstupní matice *State* a $S'_{r,c}$ bajty výstupní matice *State*, kde $r, c \in \{0, 1, 2, 3\}$:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \times \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{2.23}$$

Schéma procesu dešifrování je zakresleno na obrázku 2.10. Jako proces šifrování je rozčleněn na přediterační fázi a 10 iterací. Při dešifrování se však v přediterační fázi jako vstup transformace *AddRoundKey* použije poslední 10. podklíč vygenerovaný algoritmem plánování klíčů. Během 1. až 9. iterace, ve schématu znázorněné jako cyklus mezi přerušovanými

čarami, použijí podklíče v opačném pořadí, tedy pro 1. iteraci je použit 9. podklíč, pro 2. iteraci 8. podklíč, až pro 9. iteraci 1. podklíč. V poslední 10. iteraci se jako vstup transformace *AddRoundKey* použije samotný šifrovací klíč. Výstupem algoritmu je 128-bitový dešifrovaný datový blok.



Obrázek 2.10: Schéma dešifrování algoritmu AES

2.4 Režimy operace blokových šifer

Režim operace je způsob, jak zašifrovat zprávu libovolné délky za pomocí blokové šifry [9]. První je třeba zarovnat zprávu na délku bloku vybrané šifry. To lze jednoznačně provést tak, že na konec zprávy se připojí bit 1 následovaný potřebným počtem bitů 0 tak, aby výsledná délka zprávy byla násobkem délky bloku šifry. Pokud je délka zprávy již násobkem délky bloku, připojí se k ní jeden celý blok, který začíná bitem 1 a zbytek tohoto bloku obsahuje 0. Takto může příjemce poté, co zprávu dešifruje, jednoznačně odříznout poslední posloupnost bitů začínající 1 a jinak obsahující samé 0, a získat zprávu v její původní délce, aniž by bylo třeba se zprávou odesílat i její skutečnou délku.

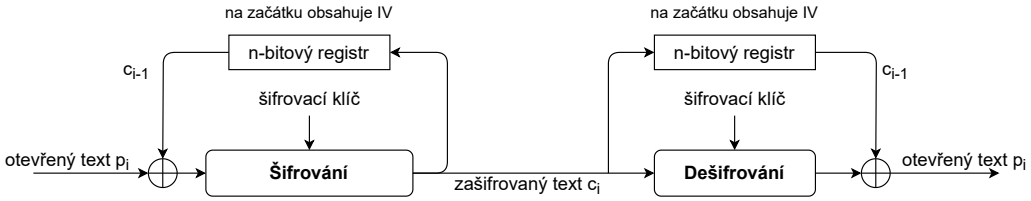
Dále představené režimy operace již předpokládají zprávu zarovnanou na délku bloku, kterou označím n. Každý režim zpracovává bloky zprávy jiným způsobem, z čehož plynou výkonnostní a bezpečnostní výhody a nevýhody, které jsou u každého uvedeny. Na závěr je zvolen režim použitý pro tuto práci.

2.4.1 Režim ECB (Electronic codebook)

Režim ECB je nejednodušší a odpovídá přímočarému použití šifrovacího algoritmu – šifrování se provádí na každém bloku samostatně, a výstupní zpráva je spojením zašifrovaných bloků. Dešifrování je stejný proces za použití dešifrovacího algoritmu. Tento režim je deterministický a není tedy bezpečný vůči útoku s výběrem otevřeného textu (chosen-plaintext attack, CPA) [9]. Navíc, pokud zpráva obsahuje dva stejné bloky, zašifrovaná zpráva bude obsahovat odpovídající stejné zašifrované bloky, z čehož může útočník při odposlouchávání zjistit další informace [9]. Režim ECB by tudíž neměl být používán.

2.4.2 Režim CBC (Cipher block chaining)

Při šifrování v režimu CBC je prvně vygenerována náhodná n-bitová hodnota zvaná inicializační vektor (initial vector, IV). Proces šifrování zprávy na straně příjemce a dešifrování na straně odesílatele je znázorněno na schématu 2.11. Otevřená (nešifrovaná) zpráva je tvořena libovolně dlouhou posloupností n-bitových bloků p_1, p_2, ..., až p_i, zašifrovaná zpráva je pak tvořena stejně dlouhou posloupností n-bitových bloků c_1, c_2, ..., až c_i. Při šifrování je používán také n-bitový registr, do kterého je na začátku uložena hodnota inicializačního vektoru IV. Hodnotu v n-bitovém registru označíme jako reg, šifrovací funkci zvoleného algoritmu (např. DES nebo AES) za použití šifrovacího klíče K označíme jako $Enc_K(p)$, kde vstup p otevřený datový blok. V každém kroku šifrování (zakresleno v levé části schématu) je pak vypočítán blok šifrované zprávy aplikací šifrovací funkce na výsledek operace XOR mezi obsahem n-bitového registru a blokem otevřeného textu, tedy $c_i = Enc_K(p_i \oplus reg)$. Tato hodnota c_i je pak uložena do registru. Protože na začátku obsahuje registr hodnotu IV, první šifrovaný blok je získán jako $c_1 = Enc_K(p_1 \oplus IV)$. Každý další šifrovaný blok je získán jako $c_i = Enc_K(p_i \oplus c_{i-1})$. Posloupnost zašifrovaných bloků je pak uložena nebo odeslána příjemci. Inicializační vektor IV není utajovaný a je potřeba při dešifrování. Proto je uložen nebo odeslán společně se zašifrovaným textem [9].



Obrázek 2.11: Schéma režimu Cipher block chaining

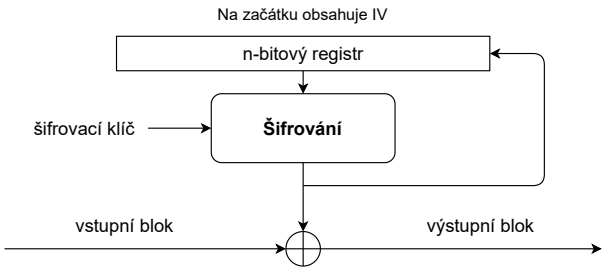
Dešifrování (zaznačené na pravé straně schématu 2.11) opět začíná načtením IV do n-bitového registru reg. Dešifrovací funkci vybraného algoritmu za použití klíče K označíme jako $Dec_K(c)$, kde c je zašifrovaný datový blok. V každém kroku je pak aplikována šifrovací funkce Dec na zašifrovaný blok c_i a mezi výsledkem a současnou hodnotou registru reg je provedena operace XOR. Výsledkem toho je blok otevřeného textu, tedy $p_i = Dec_K(c_i) \oplus reg$. První otevřený blok je získán jako $p_1 = Dec_K(c_1) \oplus IV$, následující jako $p_i = Dec_K(c_i) \oplus c_{i-1}$.

V režimu CBC je obsah zašifrovaného bloku díky zřetězení závislý na všech předchozích blocích zprávy. Odstraňuje se tedy problém ECB, kdy 2 stejné bloky ve zprávě vyústily ve 2 stejné bloky v šifrované zprávě. Navíc náhodné generování incializačního vektoru činí režim CBC nedeterministickým [9], tedy bezpečným vůči útoku s výběrem otevřeného textu.

Nevýhodou je nemožnost paralelizace výpočtu šifrování (dešifrování je možné paralelizovat), protože šifrování následujícího bloku může začít, až když je zašifrován blok předcházející.

2.4.3 Režim OFB (Output feedback)

Režím output feedback opět používá náhodně vygenerovaný inicializační vektor (IV). Jak je znázorněno ve schématu 2.12, hodnota IV se na začátku načte do n-bitového registru. V každém kroku je hodnota v registru reg zašifrována šifrovací funkcí vybraného algoritmu (označená jako $Enc_K(reg)$). Výstupní blok z šifrování r_i je zpět uložen do registru, a také je použit k vymaskování vstupního bloku zprávy m_i pomocí operace XOR. To je zapsáno jako $c_i = m_i \oplus Enc_K(r_i)$, kde c_i je výstupní blok šifrování v režimu OFB. První výstupní blok je tedy získán jako $c_1 = m_1 \oplus Enc_K(IV)$, další jako $c_2 = m_1 \oplus Enc_K(Enc_K(IV))$, atd. Při dešifrování v režimu OFB se bloky šifrované zprávy c_i posílají na vstup identického algoritmu jako při šifrování. Schéma 2.12 tedy znázorňuje jak šifrovací, tak dešifrovací algoritmus. Z toho vyplývá, že režim OFB nepoužívá dešifrovací funkci vybraného algoritmu. Jako režim CBC je i OFB režim odolný vůči útoku s výběrem otevřeného textu. Jeho nevýhodou je, že ani šifrování ani dešifrování není možné paralelizovat.



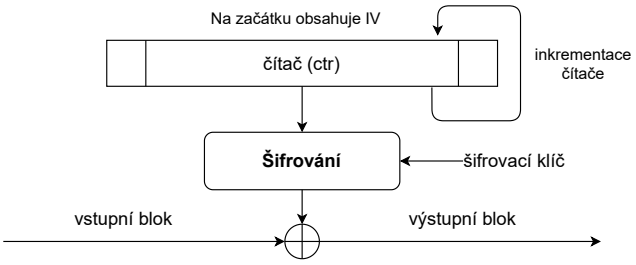
Obrázek 2.12: Schéma režimu Output feedback

2.4.4 Režim CTR (Counter)

Režim CTR je obdobou režimu OFB, ale namísto zpětné vazby, tedy opakovaného šifrování inicializačního vektoru IV, je zde použit čítač ctr, který inkrementuje svou hodnotu pro každý další blok, jak je znázorněno ve schématu 2.13. V každém kroku je hodnota čítače ctr zašifrována šifrovací funkcí vybraného algoritmu Enc_K a výsledný blok je použit k vymaskování vstupního bloku m_i pomocí operace XOR. Výpočet výsledného šifrovaného bloku c_i lze vyjádřit jako $c_i = m_i \oplus Enc_K(ctr + i)$, kde $i \in \{0, 1, ..., (L-1)\}$, kde L je počet bloků zprávy. První výstupní blok (0) je tedy získán jako $c_0 = m_0 \oplus Enc_K(ctr)$, druhý blok (1) jako $c_1 = m_1 \oplus Enc_K(ctr + 1)$, až poslední blok (L-1) jako $c_{L-1} = m_{L-1} \oplus Enc(ctr + (L-1))$. Jako režim OFB i režim CTR používá při šifrování i dešifrování stejný algoritmus, a dešifrovací funkce není vůbec použita.

Není přesně specifikováno, jak má být vygenerována počáteční hodnota čítače ctr, a jak má být inkrementována – to by měla rozhodnout konkrétní implementace [10]. Může např. vždy používat čítač s počáteční hodnotou 0, nebo pro každou zprávu generovat náhodnou n-bitovou počáteční hodnotu čítače, anebo odvodit hodnotu čítače z nějakých hodnot přenášených komunikačním protokolem, např. délka zprávy nebo sekvenční čísla použitá protokolem IPsec. Pokud má být režim CTR bezpečný vůči útokům se známým otevřeným textem, musí implementace zajistit, aby stejná hodnota ctr nebyla použita vícekrát se

stejným šifrovacím klíčem (anebo to bylo velice nepravděpodobné). V této práci je použita implementace, kdy počáteční hodnota čítače *ctr* je náhodné n-bitové číslo, je inkrementován jako n-bitové číslo bez znaménka s přetečením zpět do hodnoty 0, a hodnota *ctr* je odesílána jako první blok zašifrované zprávy.



Obrázek 2.13: Schéma režimu Counter

Takto implementovaný režim CTR je tedy bezpečný vůči útoku se známým otevřeným textem. Jeho další výhodou je možnost paralelizace při implementaci v FPGA díky tomu, že každý blok zprávy může být šifrován nezávisle na ostatních, a je tedy možné vytvoření zřetězené linky. Navíc implementace režimu CTR je obousměrná, tzn. stejný modul slouží pro šifrování i dešifrování. Z těchto důvodů jsem v této práci zvolil pro implementaci šifrovacích algoritmů DES a AES režim CTR.

Kapitola 3

Návrh komponent DES a AES

Tato kapitola popisuje architekturu komponent implementující algoritmy DES a AES. Komponenty jsou určeny k začlenění do modulu CTR, který je popsán v kapitole 5, ale mohou být použity i samostatně. Cílem bylo umístění výsledného modulu do FPGA obvodu Intel Arria 10 SX 480 a optimalizovat moduly z pohledu maximální propustnosti s požadavkem dosáhnout propustnosti alespoň 10 Gbit/s.

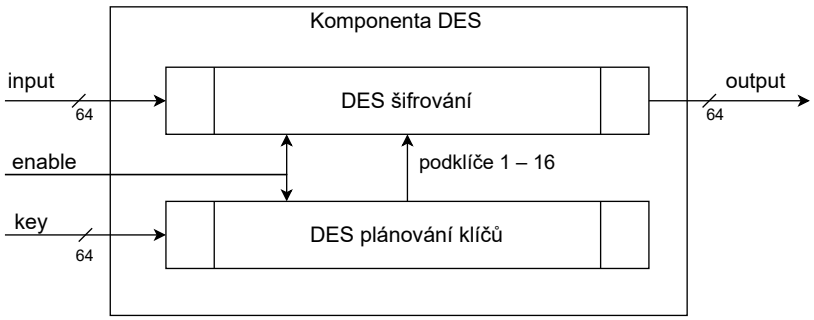
U obou komponent je použita optimalizace rozvinutí smyčky (zřetězení iterací), kdy iterace šifrování nebo generování podklíčů nejsou prováděny v cyklu za sebou, ale každá iterace je samostatně umístěna do obvodu. Dále je použita optimalizace vnitřního zřetězení iterací, kdy se mezi výpočetně náročné operace v každé iteraci vloží další registry. Takto implementovaný obvod zabírá sice několikanásobně více zdrojů v FPGA, ale také umožňuje několikanásobně zvýšit maximální možnou operační frekvenci obvodu díky výrazně zkráceným cestám signálů. Po syntéze zabírá komponenta DES méně než 1 % cílových zdrojů a její maximální operační frekvence je 489 MHz, a propustnost činí 31,2 Gbit/s. Komponenta AES pak zabírá 6 % cílových zdrojů s maximální operační frekvencí 437 MHz a propustností 55,9 Gbit/s. Obě komponenty splňují požadavek na minimální propustnost 10 Gbit/s. Pro výsledek práce jsou však podstatné až výsledky syntézy celého modulu implementující režim šifrování CTR, které jsou shrnuty v kapitole 6.

3.1 Návrh komponenty DES

Komponenta DES implementuje šifrování 64-bitového datového bloku s použitím 64-bitového šifrovacího klíče. Architektura komponenty byla navržena vzhledem k požadavku na maximální propustnost pomocí rozvinutí iterací algoritmu DES, vycházející z existující implementace [6]. Obvod se tedy skládá ze 16 stejných logických obvodů realizujících jednotlivé iterace šifrování. Výstupy jednotlivých iterací jsou ukládány do registrů, uspořádaných do zřetězené linky. Obvod se skládá ze dvou takových zřetězených linek - jedna implementující DES šifrování (odpovídající schématu 2.1) a druhá implementující algoritmus plánování klíčů (odpovídající schématu 2.4). Linka plánování klíčů v každé své fázi odesílá do zřetězené linky šifrování podklíč pro odpovídající iteraci šifrování. Architektura komponenty DES je znázorněná ve schématu 3.1 a detailněji zakreslena v obrázku 3.2. Zde bloky R_i a L_i značí registry zřetězené linky pro šifrování, bloky F, IP a IP^{-1} značí šifrovací funkci F, vstupní permutaci IP a výstupní permutaci IP^{-1}, popsané v kapitole 2.2.1 a symbol hradla značí operaci XOR. Bloky C_i a D_i značí registry zřetězené linky pro plánování klíčů a bloky PC-1, PC-2 a rol(1) značí permutační výběry PC-1 a PC-2 a operaci bitové rotace

doleva, které jsou popsané v kapitole 2.2.2. Zakresleny jsou pouze vstupy, výstupy a 1. iterace, iterace 2 až 16 jsou stejné jako 1. iterace. Následuje bližší popis vstupů a výstupů komponenty:

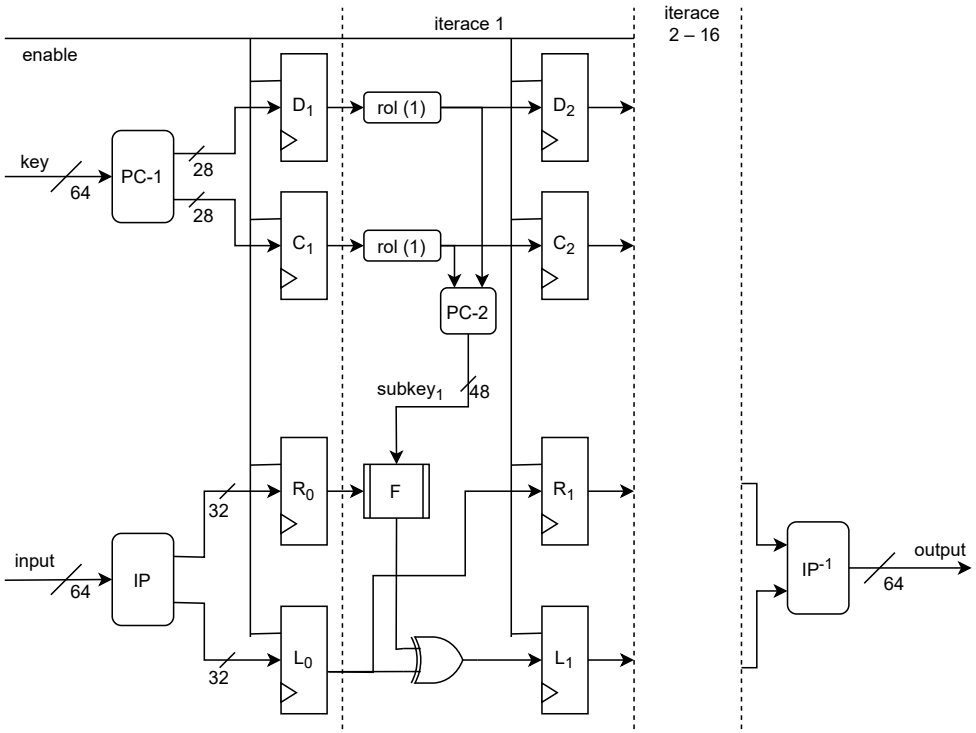
- **CLK** - hodinový signál synchronizující zápis do registrů. Na schématech není znázorněn jako vodič, ale u bloků značících registry jako trojúhelník.
- **input(63:0)** - vstupní 64-bitový blok. Při náběžné hraně je signál **input** po provedení vstupní permutace IP rozdělen na 2 poloviny a ty jsou načteny do registrů L_0 a R_0.
- **key(63:0)** - vstupní šifrovací klíč, kterým má být zašifrován vstupní blok **input**. Klíč musí být platný ve stejném hodinovém cyklu jako **input**. Při náběžné hraně CLK je po provedení permutace $PC-1$ načten do registrů C_1 a D_1.
- **output(63:0)** - výstupní blok šifrování, dostupný po 17 hodinových taktech od zavedení příslušného vstupního bloku **input** a klíče **key**.
- **enable** - signál, který je přivedený na povolovací hradlo všech registrů v celé komponentě. Pokud je **enable** v log. 1, zápis do registrů je povolen a uskuteční se v každém hodinovém cyklu. Pokud je v log. 0, zápis do registrů není povolen a všechny registry tak zachovají svou současnou hodnotu. Tento signál slouží efektivně ke zmrazení celé zřetězené linky v případě, že jiná komponenta čtoucí výstup šifrování (**output**) je zaneprázdněná a nedokáže přijmout další data.



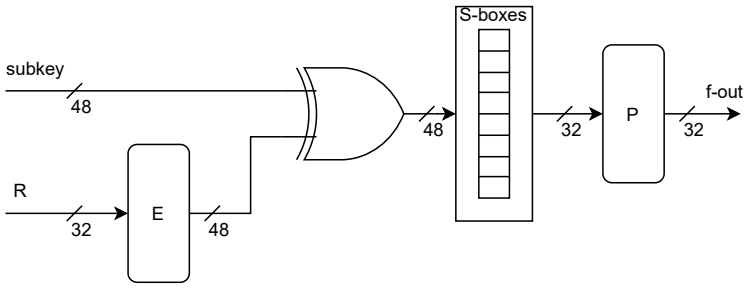
Obrázek 3.1: Schéma architektury DES komponenty

Komponenta DES neobsahuje signály pro určení platnosti vstupů nebo výstupů – každý hodinový cyklus jsou vstupy **input** a **key** načteny jako platné a za 17 hodinových cyklů je na výstup **output** přiveden odpovídající zašifrovaný blok. Řízení platných vstupů a výstupů a synchronizaci se zřetězenou linkou (zpoždění 17 taktů) musí zajistit nadřazená komponenta, která je popsána v kapitole 5.

Kromě registrů musejí být v obvodu komponenty DES realizovány operace fixní permutace, rotace, substituce a XOR. Fixní permutace i rotace nespotřebovávají v FPGA žádné výpočetní zdroje, protože se jedná o propojení vodičů. Zřetězená linka pro plánování klíčů obsahuje pouze permutace (PC-1 a PC-2) a rotace (rol), její obvodová realizace se tudíž skládá pouze s registrů a vodičů. Stejně tak v šifrovací lince nezabírají permutace IP a IP^{-1} žádné logické zdroje. V obvodu pro šifrovací funkci F (znázorněný na schématu 3.3) z hlediska zdrojů odpadají expanze E a permutace P. Substituce a operace XOR musejí být implementovány v obvodech LUT a přispívají tak ke zpoždění signálů mezi dvěma registry a mají vliv na maximální frekvenci obvodu.



Obrázek 3.2: Schéma architektury DES komponenty (podrobně)

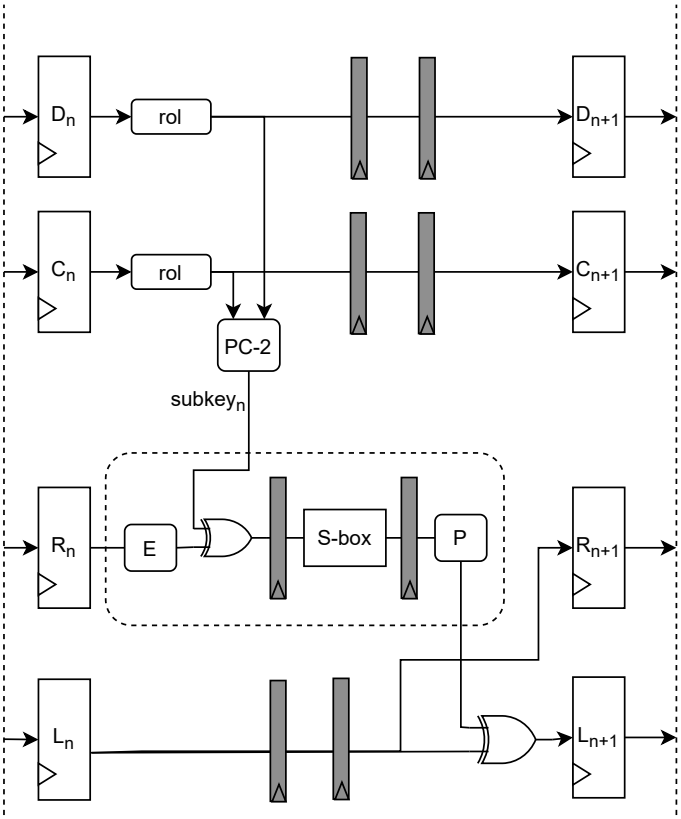


Obrázek 3.3: Schéma obvodu šifrovací funkce F(R, subkey)

3.1.1 Vnitřní zřetězení iterací

V každé iteraci obvodu pro šifrování se nachází kritická cesta, tj. cesta mezi dvěma registry s největším zpožděním, tedy s největším počtem operací prováděných ve funkčních obvodech FPGA mezi registrem R_n a L_{n+1}. Hodnota prochází přes funkci F a operaci XOR, tedy (po vynechání permutací) operací XOR, S-boxy a XOR. Pokud se mezi tyto operace vloží další registry, může to přispět ke zvýšení maximální frekvence obvodu tím, že se kritická cesta zkrátí. Na cesty, které běží paralelně s kritickou cestou, se musí vložit odpovídající počet zpožďováních registrů pro synchronizaci výpočtu. Tato optimalizace se nazývá vnitřní zřetězení kroku algoritmu. V každé iteraci se tedy mezi první operaci XOR a S-box, a mezi S-box a druhou operaci XOR se vloží 2 optimalizační registry, a po dvou zpožďováních registrech v každé iteraci na cestu mezi L_n a R_{n+1}, což je znázorněno na obrázku 3.4.

Přidané registry jsou zakresleny šedou barvou, pro přehlednost není zakreslen signál *enable*, vedoucí ke všem registrům. Taková optimalizace sice způsobí zvětšenou latenci každé iterace o 2 hodinové takty, celé linka tedy zvýší latenci na 17 + 2*16 = 49 taktů, ale umožní dosáhnout obvodu vyšší maximální pracovní frekvence.



Obrázek 3.4: Schéma jedné iterace DES s vnitřním zřetězením

Komponenta DES implementovaná pouze s rozvinutím iterací (bez vnitřního zřetězení) po syntéze pro cílovou platformu Intel Arria 10 SX 480 dosahuje maximální možné pracovní frekvence 412 MHz. Při použití vnitřního zřetězení dosahuje komponenta maximální pracovní frekvence 489 MHz, dosahuje tedy o 18,6 % většího zrychlení oproti implementaci bez vnitřního zřetězení. Komponenta je schopná každý hodinový takt přijmout nová data, její maximální propustnost lze tedy vypočítat jako šířku datového bloku vynásobenou maximální frekvencí, což je vyjádřeno následujícím vztahem 3.2:

$$T_{des} = blocksize_{des} * f_{max} = 64 * 489 = 31296\ Mbit/s = 31,29\ Gbit/s \tag{3.1}$$

Byla implementována a syntetizována ještě jedna verze komponenty DES s vnitřním zřetězením, která využívá pouze jeden optimalizační registr vložený za S-box. Tato verze dosahuje po syntéze maximální frekvence 437 MHz, tedy o 10 % horší než verze s dvěma optimalizačními registry.

3.2 Návrh komponenty AES

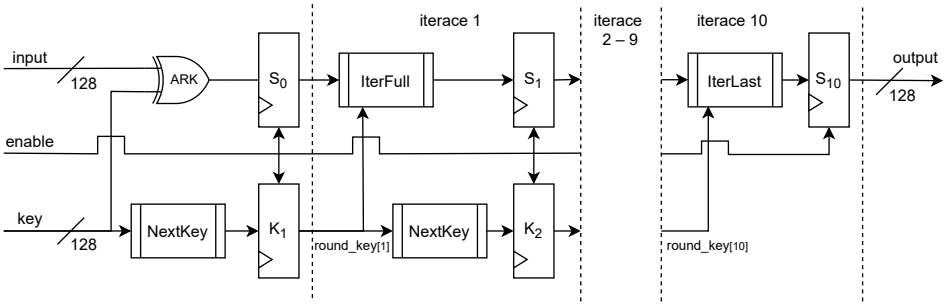
Komponenta AES implementuje šifrování 128-bitového datového bloku s použitím 128-bitového šifrovacího klíče. Tak jako v komponentě DES je použito rozvinutí iterací šifrovacího algoritmu, vycházející z existující implementace [8]. Schéma komponenty je zakresleno na obrázku 3.5. Obvod se skládá ze dvou zřetězených linek: první implementující algoritmus AES šifrování, popsaný v kapitole 2.3.1, a druhá implementující algoritmus plánování klíčů, popsaný v kapitole 2.3.2.

Zřetězená linka, implementující plánování klíčů, se skládá z 10 shodných iterací. V první iteraci je ze vstupního 128-bitového klíče (odpovídající vstupnímu signálu *key*) vypočítán první 128-bitový podklíč a jeho hodnota je uložena do registru K_1. Tento výpočet je ve schématu vyznačen blokem *NextKey*. V každé další iteraci je z podklíče uloženého v registru K_r vypočítána hodnota dalšího podklíče a uložena do následujícího registru K_{r+1}, kde $r \in \{1, 2, 3, ...9\}$. Hodnoty z registrů K_1 až K_{10}, které jsou ve schématu označeny jako signály $round\_key[1]$ až $round\_key[10]$, vedou k odpovídajícím iteracím zřetězené linky šifrování jako vstupy (podklíče) do bloků ve schématu označených jako *IterFull*.

Zřetězená linka, implementující šifrování, se skládá z přediterační fáze, 9 shodných iterací a poslední 10. iterace, která je mírně odlišná od předchozích. Přediterační fáze je ve schématu znázorněna jako hradlo XOR s popiskem *ARK*, a implementuje operaci *AddRoundKey* nad vstupním blokem *input* a šifrovacím klíčem *key*. Výsledek je uložen do registru S_0. V následujících iteracích 1 až 9 je hodnota v registru S_{r-1}, která odpovídá matici *State*, popsané v kapitole 2.3.1, přivedena spolu s podklíčem $round\_key[r]$ z linky plánování klíčů na vstup bloku *IterFull*, který implementuje jednu iteraci šifrování AES a je popsán detailněji dále. Výstup z tohoto bloku je uložen do následujícího registru zřetězené linky S_r, kde $r \in \{1, 2, 3, ...9\}$. Ve schématu je zakreslena pouze první z těchto devíti iterací. V poslední iteraci je hodnota v registru S_9 přivedena spolu s podklíčem $round\_key[10]$ na vstup bloku nazvaného *IterLast*, který implementuje poslední iteraci AES šifrování a je popsán dále. Výstup bloku *IterLast* je uložen do registru S_{10}, jehož hodnota je přivedena na výstup *output* komponenty AES. Následuje bližší popis vstupních a výstupních signálů komponenty:

- **CLK** - hodinový signál synchronizující zápis do registrů. Na schématech není znázorněn jako vodič, ale u bloků značících registry jako trojúhelník.
- **input(127:0)** - vstupní 128-bitový datový blok.
- **key(127:0)** - vstupní šifrovací klíč, kterým má být zašifrován vstupní blok **input**. Klíč musí být platný ve stejném hodinovém cyklu jako **input**.
- **output(127:0)** - výstupní blok šifrování, dostupný po 11 hodinových taktech od zavedení příslušného vstupního bloku **input** a klíče **key**.
- **enable** - signál, který je přivedený na povolovací hradlo všech registrů v celé komponentě. Pokud je **enable** v log. 1, zápis do registrů je povolen a uskuteční se v každém hodinovém cyklu. Pokud je v log. 0, zápis do registrů není povolen a všechny registry tak zachovají svou současnou hodnotu. Tento signál slouží efektivně ke zmrazení celé zřetězené linky v případě, že jiná komponenta čtoucí výstup šifrování (**output**) je zaneprázdněná a nedokáže přijmout další data.

Blok *IterFull* ze schématu 3.5 implementuje transformace *SubBytes*, *ShiftRows*, *MixColumns* a *AddRoundKey*, které jsou popsané v kapitole 2.2.1. Architektura obvodu je zakres-

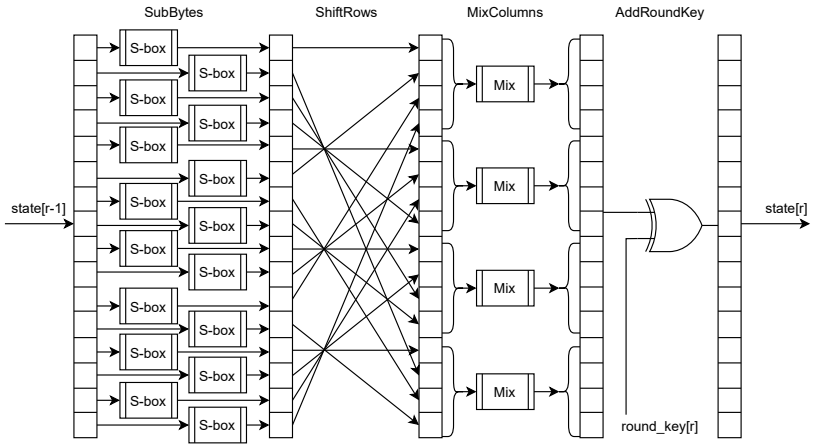


Obrázek 3.5: Schéma architektury AES komponenty

lena na obrázku 3.6. Zde je transformovaná 128-bitová hodnota znázorněna jako sloupec o 16 polích, které odpovídají jednotlivým bajtům této hodnoty, kdy pole nejvíce nahoře odpovídá prvnímu bajtu až pole úplně dole odpovídá 16. bajtu hodnoty. Vstupem do bloku *IterFull* je 16-bajtová hodnota z registru S_{r-1}, kde $r \in \{1, 2, ...10\}$, která je zaznačena jako signál $state[r-1]$. Každý ze vstupních 16 bajtů je vstupem do substituční funkce *S-box*, která implementuje operaci nahrazení bajtů (*SubBytes*) dle tabulky 2.7 a je realizována v kombinačních obvodech LUT. Výstupy substitučních funkcí jsou přesměrovány na jiné bajtové pozice. Toto přesměrování vodičů odpovídá operaci rotace řádků (*ShiftRows*) dle obrázku 2.7, a nespotřebovává žádné logické zdroje. Přesměrované bajty jsou nyní rozděleny na čtveřice, jak je to znázorněno ve schématu. Tyto čtveřice bajtů, které odpovídají 4 sloupcům matice *State*, jsou vstupem do funkce *Mix*, která implementuje operaci míchání sloupců (*MixColumns*) dle vztahu 2.3. Výstupem funkcí *Mix* je dohromady 16 bajtů, které jsou vymaskovány podklíčem pro příslušnou iteraci r za použití funkce XOR. Podklíč pro iteraci r je ve schématu zaznačen signálem $round\_key[r]$, a funkce XOR, která realizuje operaci *AddRoundKey*, je zaznačena hradlem XOR. Funkce *Mix* a operace *AddRoundKey* se skládají s funkcí XOR a jsou realizovány v obvodech LUT. Výstupní 16-bajtová hodnota, zaznačena jako $state[r]$, je uložena do následujícího registru zřetězené linky S_r.

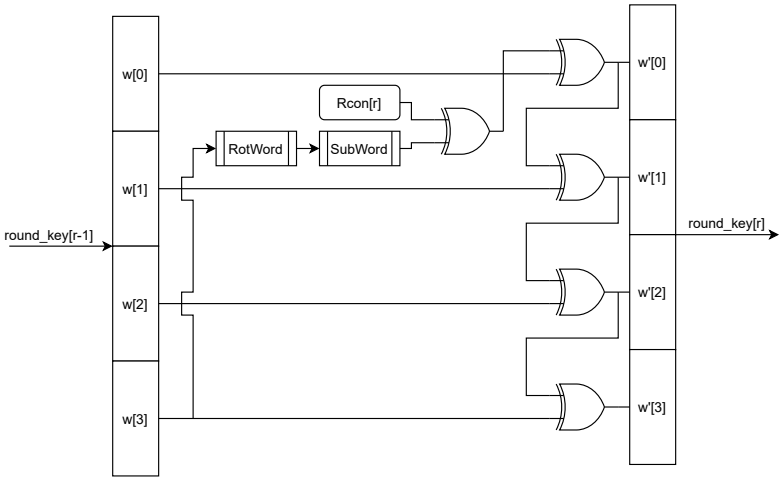
Blok *IterLast*, realizující poslední iteraci šifrování, je stejný jako blok *IterFull*, pouze neobsahuje funkce *Mix*, takže hodnota z fáze rotace řádků *ShiftRows* je přímo vstupem do funkce XOR ve fázi přičtení iteračního klíče *AddRoundKey*.

Blok *NextKey* realizuje jednu iteraci algoritmu plánování klíčů, který je popsaný v kapitole 2.3.2. Architektura bloku je zakreslena na následujícím obrázku 3.7. Jeho vstupem je 128-bitový podklíč, znázorněný jako signál $round\_key[r-1]$, kde $r \in \{1, 2, ..., 10\}$. Pro první iteraci *NextKey* odpovídá tento vstup samotnému šifrovacímu klíči (tedy vstupnímu signálu *key*). Pro ostatní iterace je signál $round\_key[r-1]$ hodnotou z registru K_{r-1}. Vstupní signál se skládá ze čtyř 32-bitových slov, zaznačených jako $w[0]$ až $w[3]$. Z nich jsou vygenerovány 4 nové 32-bitové slova $w'[0]$ až $w'[3]$, které tvoří výstupní podklíč $round\_key[r]$. První výstupní slovo $w'[0]$ je vypočítáno dle vztahu 2.14 s použitím funkcí označených ve schématu jako *RotWord*, *SubWord*, a XOR. Funkce *RotWord* provede rotaci bajtů slova $w[3]$ dle vztahu 2.15, což je realizováno pouze propojením vodičů. Na výstup funkce *RotWord* je aplikována funkce *SubWord*, která nahradí každý ze 4 bajtů slova novou hodnotou dle vztahů 2.16. Mezi výstupem *SubWord* a 32-bitovou konstantou z tabulky *Rcon* 2.8 na indexu r je pak provedena operace XOR, a mezi výsledkem a vstupním slovem $w[0]$ je opět proveden XOR. Výsledkem je slovo $w'[0]$ výstupního podklíče $round\_key[r]$. Zbývající výstupní slova $w'[1]$, $w'[2]$, $w'[3]$ jsou pak výsledkem operace XOR mezi předcházejícím výstupním slovem, a



Obrázek 3.6: Schéma architektury bloku iterace AES (IterFull)

slovem vstupního podklíče na stejné pozici, tedy $w'[1] = w'[0] \oplus w[1]$, $w'[2] = w'[1] \oplus w[2]$, a $w'[3] = w'[2] \oplus w[3]$. Funkce *SubWord* i funkce XOR jsou realizovány v obvodech LUT. Výstupní klíč $round\_key[r]$ je uložen do následujícího registru zřetězené linky plánování klíčů K_r.



Obrázek 3.7: Schéma architektury bloku generování podklíče (NextKey)

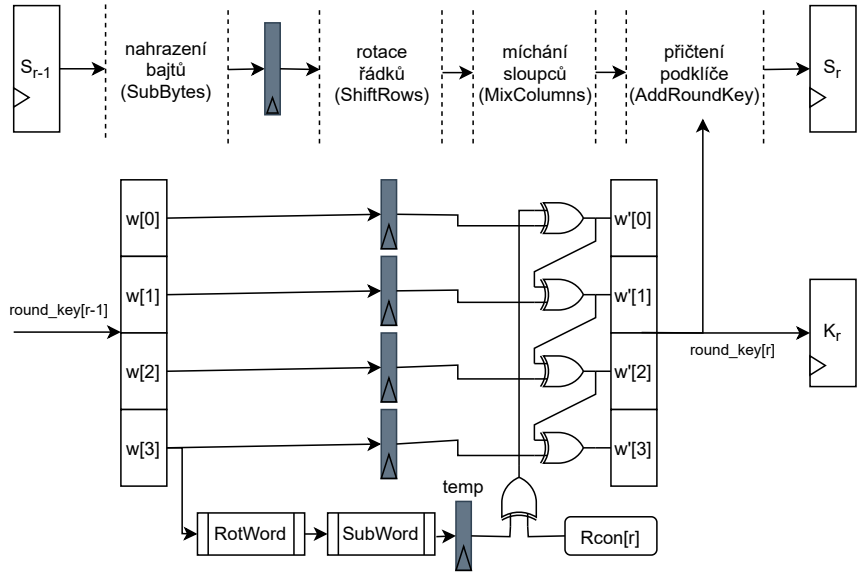
3.2.1 Vnitřní zřetězení iterací

Každou iteraci v obvodu šifrování AES lze stejně jako u komponenty DES dále optimalizovat vložením registrů mezi operace na kritických cestách. Ve zřetězené lince pro šifrování je nejvíce zdrojů spotřebováno na fázi nahrazení bajtů (*SubBytes*) a na fázi míchání sloupců (*MixColumns*) [8]. Ve zřetězené lince plánování klíčů je nejvíce zdrojů spotřebováno na fázi substituci slova (*SubWord*). Mezi tyto operace jsou tedy vloženy optimalizační registry. Byly implementovány dvě verze komponenty AES využívající optimalizaci vnitřního zřetězení iterací. První verze je zakreslena ve schématu 3.8 a využívá jeden optimalizační

registr. Ve schématu je zakreslena jedna iterace ve zřetězené lince šifrování jako cesta mezi registry S_{r-1} a S_r, kde $r \in \{1, 2, ...10\}$, a jedna iterace zřetězené linky plánování klíčů jako cesta začínající signálem $round\_key[r-1]$. V první iteraci je tento signál samotným vstupním šifrovacím klíčem komponenty *key*, v dalších iteracích je to hodnota z registru K_{r-1}. Z hodnoty $round\_key[r-1]$ je vypočítána hodnota podklíče pro iteraci šifrování r. Tato hodnota je označena jako signál $round\_key[r]$, který je vstupem do fáze přičtení podklíče (*AddRoundKey*) a je uložena do registru K_r.

Vložené registry jsou zde zakresleny šedou barvou. Do iterace šifrování je vložen jeden optimalizační registr, který uchovává výstup z fáze nahrazení bajtů. Do iterace plánování klíčů je vložen optimalizační registr, označený jako *temp*, uchovávající výsledek funkce *SubWord*, a 4 zpožďovací registry, které synchronizují hodnoty jednotlivých slov podklíče $W[0]$ až $w[3]$ s optimalizačními registry. Mezi obsahem registru *temp* a hodnotou $Rcon[r]$ je v dalším taktu provedena operace XOR a jsou vypočítány hodnoty slov $w'[0]$ až $w'[3]$ iteračního podklíče $round\_key[r]$, jak bylo dříve popsáno v architektuře bloku *NextKey*.

V poslední iteraci šifrování je umístění optimalizačních ani zpožďovacích registrů nemění, je pouze vynechána fáze míchání sloupců (*MixColumns*). Tato optimalizace vnáší do každé z deseti iterací prodlevu jednoho taktu, prodleva celé zřetězené se tedy zvýší z 11 na 21 taktů.



Obrázek 3.8: Schéma jedné iterace AES využívající jeden optimalizační registr

Druhá verze komponenty AES s vnitřním zřetězením využívá navíc další optimalizační registr, vložený za fázi míchání sloupců. Ve výpočtu podklíče je pak vložen další optimalizační registr, který uchovává výsledek operace XOR mezi hodnotou z registru *temp* a konstantou $Rcon[r]$, a další 4 zpožďovací registry pro slova podklíče. Tyto 4 registry berou hodnotu z předchozích zpožďovacích registrů a synchronizují hodnoty slov $w[0]$ až $w[4]$ s optimalizačními registry. Tato optimalizace vnáší do každé iterace prodlevu další 2 hodinové takty, prodleva celé zřetězené linky tak činí 31 taktů.

Po syntéze pro cílovou platformu Intel Arria 10 SX 480 dosahuje komponenta AES bez vnitřního zřetězení maximální pracovní frekvence 291 MHz. První verze komponenty AES s vnitřním zřetězením iterací (používající 1 optimalizační registr) dosahuje maximální frek-

vence 423 MHz, a druhá verze (používající 2 optimalizační registry) dosahuje maximální frekvence 437 MHz. Jeden optimalizační registr tedy přinesl zlepšení maximální frekvence, tedy i propustnosti o 45 % oproti verzi bez vnitřního zřetězení iterací. Verze se dvěma optimalizačními registry dosáhla nepatrného zlepšení o 3 % oproti verzi s jedním optimalizačním registrem. Při použití ve výsledné komponentě, popsané v kapitole 5, však dosáhla větší maximální frekvence verze komponenty AES s jedním optimalizačním registrem. Verze se dvěma registry byla pomalejší kvůli zvětšené prodlevě zřetězené linky, což mělo za následek větší zpoždění některých řídicích signálů výsledné komponenty, které jsem již nedokázal lépe optimalizovat. Proto je v této práci použita verze komponenty AES s vnitřním zřetězením s jedním optimalizačním registrem.
Komponenta AES, stejně jako komponenta DES, dokáže každý hodinový cyklus přijmout nová data k šifrování. Propustnost této verze komponenty AES, vypočítaná jako násobek bitové šířky datového bloku (128 bitů) a maximální pracovní frekvence 423 MHz, činí tedy:

$$T_{aes} = blocksize_{aes} * f_{max} = 128 * 423 = 54144\, Mbit/s = 54,14\, Gbit/s \tag{3.2}$$

Kapitola 4

Rozhraní výsledného modulu

Cílem této práce je implementace šifrovacího modulu pro obvod Intel Arria 10 SX 480. Tento modul implementuje blokové šifrování v režimu CTR, který je popsaný v kapitole 2.4.4, za použití šifer DES nebo AES. Režim CTR používá zcela stejný algoritmus jak pro šifrování, tak pro dešifrování, tzn. pokud je vstupem šifrovaná zpráva, výstupní zpráva je dešifrovaná, a naopak, pokud je vstup otevřený (nešifrovaný), výstupem je zašifrovaná zpráva. Proto je dále proces, který modul implementuje, nazýván pouze jako šifrování.

4.1 Protokol AXI4-Stream

Modul komunikuje pomocí dvou samostatných rozhraní: jedno pro příjem dat určených k šifrování, a druhé pro odesílání zašifrovaných dat. Obě tyto rozhraní využívají pro komunikaci protokol AMBA 4 AXI4-Stream [4]. Protokol AXI4-Stream je používán jako standardní rozhraní k propojení komponent, které si potřebují vyměňovat data. Výměna dat probíhá mezi odesílatelem (master), který data vytváří nebo přeposílá a komunikaci zahajuje, a příjemcem (slave), který data přijímá a zpracovává. Rozhraní přenáší data na úrovni bajtů a využívá datové a řídicí signály, a další signály pro přenos informací o jednotlivých datových bajtech nebo informací pro směrování. Větší část těchto signálů je nepovinných, a modul v této práci využívá ve svém rozhraní podmnožinu signálů AXI4-Stream, které jsou popsány dále. V popisu jsou použity pojmy přenos, paket a datový proud. Přenos (transfer) označuje jeden přenos dat probíhající v jednom hodinovém taktu a je vymezen signalizací dle postupu, který se označuje jako *handshake*, který je popsán níže. Paket (packet) označuje datovou jednotku skládající se z jednoho či více přenosů (transfers) a má jednoznačně definovaný první a poslední přenos (začátek a konec). Datový proud (data stream) označuje přenos dat přes AXI4-Stream rozhraní jakožto posloupnost paketů. Jednotlivé pakety v datovém proudu se nesmějí prolínat, tzn. nemůže být proveden přenos, náležící k jednomu paketu, následně přenos náležící druhému paketu, a pak přenos náležící opět k prvnímu. Odesílání dalšího paketu může započít až po odeslání všech přenosů předchozího paketu.

Modul očekává na svém vstupním rozhraní datový proud, kde jednotlivé pakety odpovídají zprávám určeným k šifrování. Na výstupním rozhraní modulu je odesílán datový proud zašifrovaných paketů ze vstupního datového proudu. Následuje popis signálů použitých v rozhraních modulu: signály `TVALID`, `TDATA`, `TID` a `TLAST` vedou od odesílatele (master) k příjemci (slave), a signál `TREADY` vede od příjemce k odesílateli.

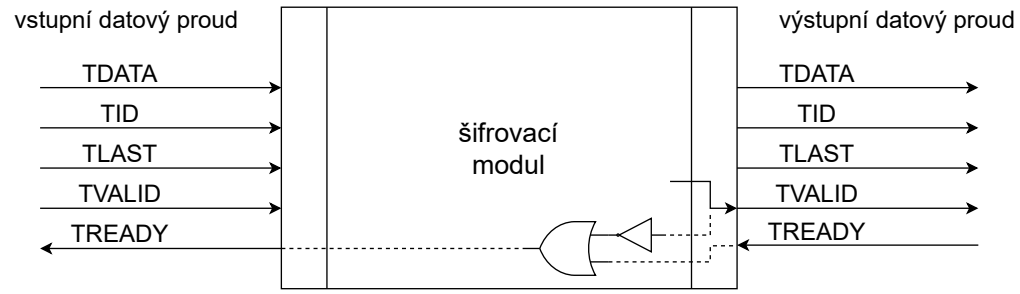
- **ACLK** - hodinový signál rozhraní, který je stejný jako globální hodinový signál celého obvodu `CLK`. Všechny signály rozhraní jsou čteny vždy na náběžné hraně signálu *ACLK*.
- **TDATA[(n-1):0]** - signál pro přenos datového bloku, jehož šířka odpovídá šířce bloku vybraného algoritmu (pro DES je $n = 64$, pro AES je $n = 128$). Platnost signálu je určována signálem *TVALID*.
- **TVALID** - signál značící příjemci, že odesílatel chce provést přenos (transfer) a hodnoty signálů `TDATA`, `TLAST` a `TID` jsou platné. Hodnota signálu **TREADY** poté určí, jestli se přenos provede. Proces přenosu (*handshake*) je popsán níže.
- **TLAST** - signál označující poslední datový přenos v tomto paketu, tedy konec paketu. Začátek paketu, tedy první datový přenos, není v rozhraní signalizován. Za první přenos je příjemcem považován první přenos (označený signálem `TVALID` v log. 1) po resetování obvodu, nebo první přenos následující za posledním přenosem, který byl označen signálem *TLAST* v log. 1.
- **TID[7:0]** - signál nesoucí identifikátor paketu, ke kterému právě probíhající datový přenos náleží. Standard nespecifikuje, jakou bitovou šířku má tento signál. V této práci byla zvolena největší doporučená šířka 8 bitů. Hodnota signálu `TID` také slouží pro výběr šifrovacího klíče, který se použije pro šifrování paketu s tímto identifikátorem. Toto je podrobně popsáno v následující kapitole 5.
- **TREADY** - signál, kterým příjemce dává najevo odesílateli, že v současném hodinovém taktu je připraven přijmout datový přenos, a že odesílatel může v následujícím taktu zahájit další datový přenos.

Proces datového přenosu (handshake)

Proces datového přenosu, nazvaný jako *TVALID-TREADY handshake*, je postup, jakým si odesílatel úspěšně předá příjemci jeden blok dat. Přenos zahajuje odesílatel tím, že signály `TDATA`, `TLAST` a `TID` nastaví na platné hodnoty a signál `TVALID` nastaví do logické 1. Jakmile je příjemce připraven tento přenos uskutečnit, tedy přečíst a zpracovat tyto data, nastaví signál `TREADY` do logické 1. Pokud je příjemce vždy připraven data přijmout, může mít signál `TREADY` nastavený stále v log. 1, ale na to se odesílatel nesmí spoléhat. Přenos je uskutečněn, když mají oba signály `TREADY` a `TVALID` při náběžné hraně hodinového signálu `ACLK` hodnotu log. 1. Poté může odesílatel v dalším taktu zahájit přenos jiných dat, anebo nastavit signál `TVALID` zpět do log. 0. Pokud příjemce nemůže data přijmout, nastaví signál `TREADY` do log. 0. Odesílatel pak musí hodnoty všech signálů `TDATA`, `TLAST`, `TID` a `TVALID` ponechat stejné, dokud příjemce nenastaví signál `TREADY` do log. 1. Přenos se pak uskuteční na nejbližší náběžné hraně hodinového signálu `ACLK`.

Na obrázku 4.1 je zakresleno schéma vstupního a výstupního AXI4-Stream rozhraní šifrovacího modulu. Architektura šifrovacího modulu, podrobně popsána v následující kapitole 5, umožňuje zpracovat v každém hodinovém taktu nová vstupní data, a po zpracování každý takt odesílat data na výstupní rozhraní. Pokud je tedy komponenta, odebírající zpracovaná data z výstupu šifrovacího modulu, také schopná každý takt data přijmout, je toho schopen i šifrovací modul. V situaci, kdy šifrovací modul provádí přenos, nastaví tedy signál `TVALID` na výstupu do log. 1, ale odebírající komponenta není schopná data přijmout, takže má nastavený signál `TREADY` v log. 0, dojde k pozastavení celého šifrovacího modulu. Na

vstupní rozhraní je ve stejném taktu pak nastaven signál `TREADY` do log. 0, který je jinak stále nastaven do log. 1. Vstupní rozhraní je tak pozastaveno, dokud odebírající komponenta neobnoví odběr dat. Tato řídicí logika je ve schématu zakreslena jako hradlo NOT a hradlo OR tak, že vstupní `TREADY` = ¬`TVALID` ∨ `TREADY`, takže vstupní `TREADY` je vždy v log. 1 kromě stavu, kdy `TVALID` = log. 1 a zároveň `TREADY` = log. 0.

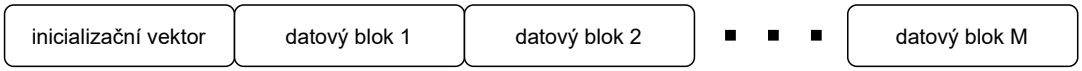


Obrázek 4.1: Schéma signálů vstupního a výstupního AXI4-Stream rozhraní modulu

4.2 Formát paketů

Pakety v datovém proudu vstupního i výstupního rozhraní mají stejný formát, znázorněný na obrázku 4.2. Jeden paket odpovídá jedné zprávě určené k šifrování s použitím jednoho šifrovacího klíče, a zpráva je zarovnaná na násobky bitové délky bloku, kdy jeden blok je dlouhý 64 bitů, pokud je šifrovací modul implementován s užitím komponenty DES, a 128 bitů, pokud je modul implementován s užitím komponenty AES. Každý paket se skládá alespoň ze 2 datových bloků, kdy jeden datový blok odpovídá jednomu přenosu přes AXI4-Stream rozhraní. První blok v každém paketu vždy obsahuje počáteční hodnotu čítače pro šifrování v režimu CTR, v obrázku označen jako inicializační vektor. Za ním následuje M datových bloků, určených k šifrování, kde $M \geq 1$. Celý paket má tedy délku $M + 1$ bloků. Každý paket má také vlastní hodnotu `TID`, která ale není nutně unikátní pro různé pakety. Dle hodnoty `TID` se v šifrovacím modulu z jeho vnitřní paměti vybere šifrovací klíč pro daný paket. Toto je popsáno v následující kapitole 5.

Z každého vstupního paketu šifrovací modul oddělí incializační vektor, datové bloky 1 až M zašifruje, a k zašifrovaným datovým blokům připojí tento stejný inicializační vektor jako první blok paketu, který je odeslán na výstup.

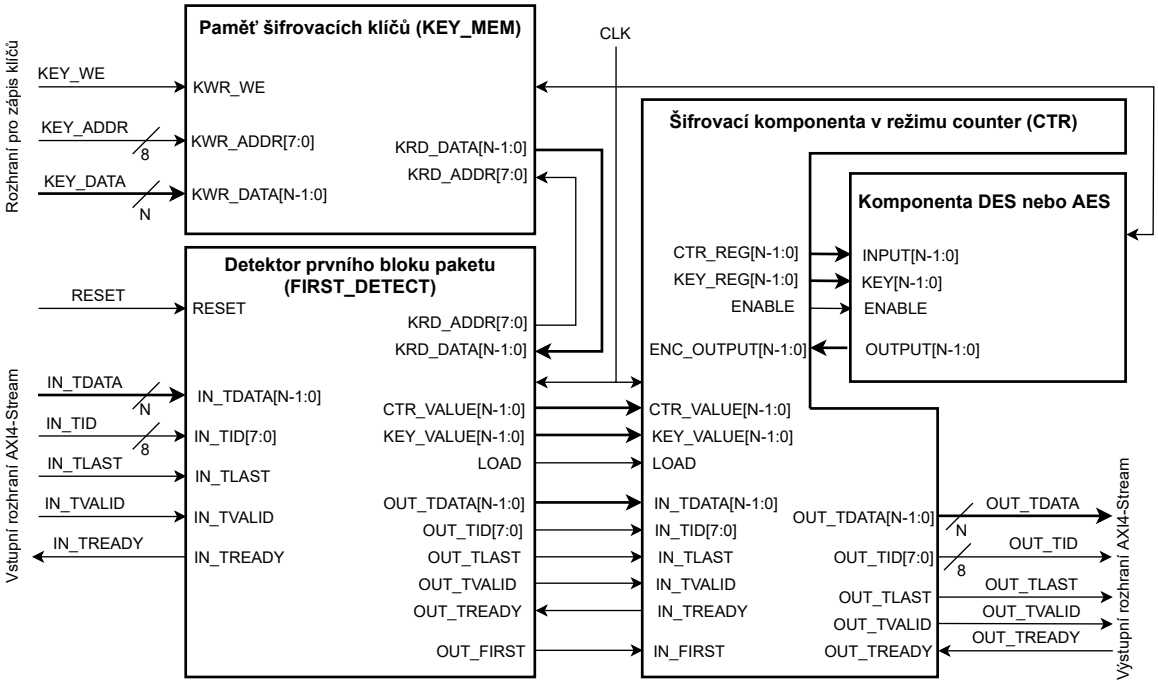


Obrázek 4.2: Formát paketu vstupního i výstupního datového proudu modulu

Kapitola 5

Architektura výsledného modulu

Architektura výsledného šifrovacího modulu byla navržena pro umístění do obvodu Intel Arria 10 SX 480 s cílem dosáhnout co nejvyšší propustnosti. Schéma výsledného modulu je zakresleno v obrázku 5.1 a vztahuje se k němu následující popis.



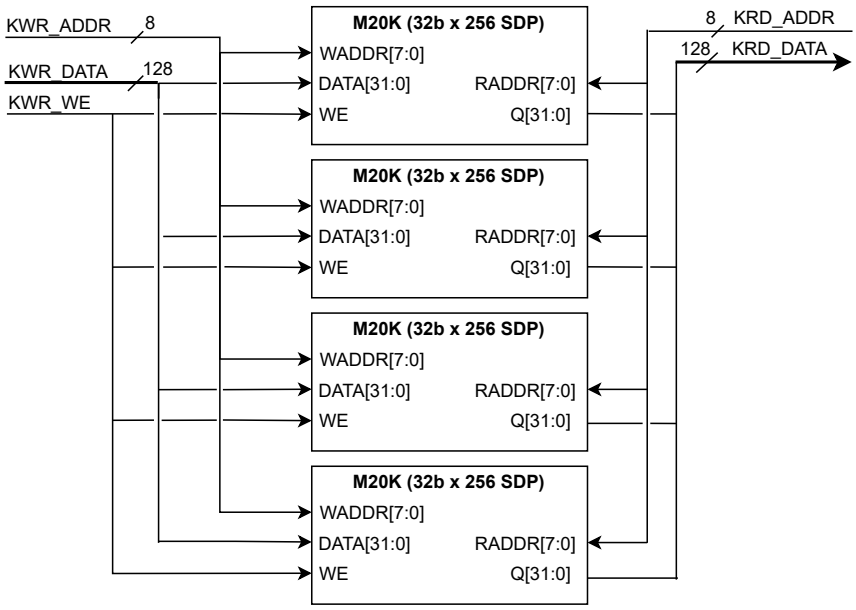
Obrázek 5.1: Schéma architektury výsledného šifrovacího modulu

Modul je složen ze tří částí: 1) paměti pro ukládání šifrovacích klíčů (KEY_MEM), 2) řídicí jednotky pro detekci prvního bloku (tedy inicializačního vektoru) příchozích paketů (FIRST_DETECT), a 3) jednotky implementující šifrování v režimu CTR (CTR), která používá komponenty DES nebo komponentu AES z kapitoly 3. Tyto tři části jsou popsány v následujících podkapitolách. Modul je navržen pro použití jak s komponentou DES, tak s komponentou AES. Od použité komponenty (DES nebo AES) se odvodí dva parametry pro ostatní prvky modulu: 1) Šířka datových vodičů a registrů, označená jako N. Pro DES je $N = 64$, pro AES je $N = 128$. 2) Prodleva zřetězené linky pro šifrování v hodinových

taktech, označená jako D (delay) a použitá v podkapitole 5.3. Pro použitou verzi DES se dvěma optimalizačními registry je $D = 49$, pro použitou verzi AES s jedním optimalizačním registrem je $D = 21$.
Rozhraní modulu tvoří globální hodinový signál `CLK` a globální `RESET`, vstupní a výstupní AXI4-Stream rozhraní, popsané v předcházející kapitole 4, a rozhraní pro zápis šifrovacích klíčů. To se skládá ze tří signálů, označených jako `KEY_WE` – povolení zápisu klíče (write enable), `KEY_ADDR` – 8-bitová adresa klíče a `KEY_DATA` – N-bitová hodnota klíče. Tyto vstupní signály jsou přivedeny přímo na odpovídající porty paměti šifrovacích klíčů.

5.1 Paměť šifrovacích klíčů

Paměť šifrovacích klíčů umožňuje uložit 256 N-bitových šifrovacích klíčů, které jsou následně používány komponentou DES nebo AES. Adresa klíče (široká 8 bitů) odpovídá hodnotě identifikátoru paketu `TID`. Klíče mohou být ukládány do paměti před nebo během šifrování, alespoň jeden hodinový takt před odesláním paketu na vstupní rozhraní šifrovacího modulu. Schéma paměti klíčů pro variantu používající AES je zakresleno na obrázku 5.2. Paměť poskytuje dvě rozhraní: 1) Pro zápis klíčů (tvořené signály `KWR_ADDR`, `KWR_DATA` a `KWR_WE`, a 2) Pro čtení klíčů (tvořeno signály `KRD_ADDR` a `KRD_DATA`), které používá jednotka CTR 5.3.



Obrázek 5.2: Schéma paměti šifrovacích klíčů (verze pro AES)

Pro implementaci paměti klíčů jsou využity vestavěné bloky paměti, které se na zařízeních Intel Arria 10 označují jako **M20K**. Bloky **M20K** dokáží ukládat až 20 kilobitů dat dle konfigurace datové šířky a hloubky [5]. Paměť klíčů bylo nejvýhodnější sestavit z **M20K** bloků s datovou šířkou 32 bitů a hloubkou 256 (tedy 8-bitovou adresou), pracující v režimu jednoduché dvouportové paměti (simple dual-port – SDP), tzn. poskytuje jeden port pouze pro zápis a jeden port pouze pro čtení. Paměť 128-bitových klíčů pro AES, která je znázorněna na schématu, je sestavena ze čtyř takových bloků, které jsou označené jako **M20K (32b x 256 SDP)**. Paměť 64-bitových klíčů pro DES je složena ze dvou bloků **M20K**

(32b x 256 SDP). Na každý port pro zápis je přivedena stejná vstupní adresa `KWR_ADDR`, a vstupní 128-bitová hodnota klíče `KWR_DATA` je rozdělena na čtyři 32-bitové části, každá z nich je pak přivedena na port pro zápis jednoho bloku **M20K**. Každý ze 4 bloků **M20K** tak uloží jednu ze čtyř částí klíče na stejné adrese. Stejným způsobem jsou zavedeny signály pro čtení klíčů `KRD_ADDR` a `KRD_DATA`.

Hodnota klíče `KWR_DATA` je zapsána na danou adresu `KWR_ADDR` při následující náběžné hraně hodinového signálu `CLK`, pouze pokud je signál `KWR_WE` v log. 1. Čtená hodnota klíče z adresy `KRD_ADDR` je přivedena na výstup `KRD_DATA` v následujícím taktu (po náběžné hraně `CLK`) od vystavení adresy. Pokud je ve stejném taktu proveden zápis i čtení na stejnou adresu, je přečtena stará hodnota před zápisem.

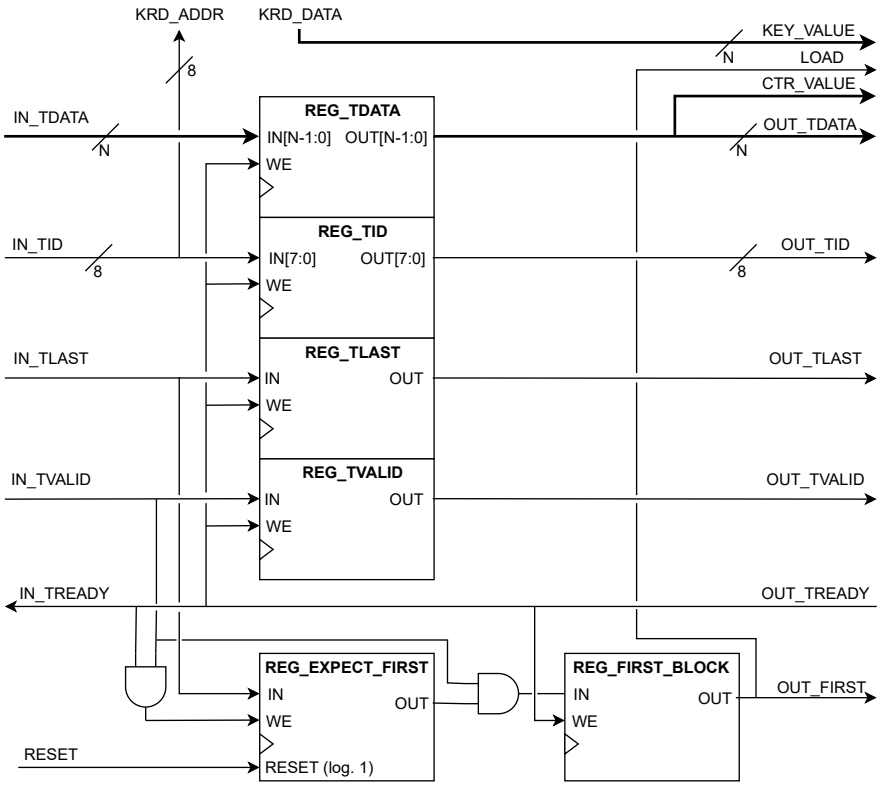
5.2 Detektor prvního bloku paketu

První přenos (blok) každého vstupního paketu nese inicializační vektor (IV), jak je popsáno v kapitole 4. Hodnota IV slouží při šifrování jako počáteční hodnota čítače režimu CTR. Přitom rozhraní AXI4-Stream neobsahuje signál, který označuje první blok v paketu – přijímač musí za první blok považovat první platný přenos po resetu zařízení, nebo pokud předchozí platný přenos měl nastavenou hodnotu `TLAST` v log. 1. Jednotka detektoru implementuje tuto detekci prvního bloku paketu, a také načtení šifrovacího klíče pro nový paket z paměti klíčů 5.1. Šifrovací klíč a hodnotu IV odesílá pak do jednotky CTR 5.3.

Schéma jednotky je zakresleno na obrázku 5.3. Rozhraní tvoří vstupní a výstupní signály AXI4-Stream, signály pro čtení z paměti klíčů, signál `OUT_FIRST`, který informuje jednotku CTR, že tento přenos je prvním v paketu, a signály s hodnotou klíče `KEY_VALUE`, s hodnotou IV `CTR_VALUE` a signál `LOAD` informující jednotku CTR, že je třeba načíst nový klíč a novou hodnotu čítače (IV).

Pro detekci prvního bloku paketu slouží registr **REG_EXPECT_FIRST**, který obsahuje log. 1, pokud následující platný přenos na rozhraní AXI4-Stream bude prvním blokem. Registr je nastaven do log. 1 po globálním resetu, a pak vždy, když proběhne platný přenos posledního bloku v paketu. Toto je implementováno přivedením vstupního signálu `TLAST` na vstupní port registru (`IN`) a výsledek `OUT_TREADY` $\wedge$ `IN_TVALID` na port `WE` povolující zápis do registru.

Data ze vstupního rozhraní AXI4-Stream jsou uložena v každém taktu do registrů s odpovídajícím označením, pokud je signál `OUT_TREADY` v log. 1 (tzn. modul může přijímat nová data). Zároveň je signál `IN_TID` přiveden na adresu pro čtení klíče `KRD_ADDR`. V následujícím taktu bude tedy hodnota klíče dostupná na vstupu `KRD_DATA`, který je přímo zapojen do portu jednotky CTR jako `KEY_VALUE`. Hodnota je využita pouze pokud je signál `LOAD` v log. 1 (tj. tehdy, když je do jednotky CTR odesílán první blok paketu). Zároveň, pokud je přenos platný (tj. `IN_TVALID` je v log. 1) a je očekáván první blok paketu (tj. **REG_EXPECT_FIRST** obsahuje log. 1), je do registru **REG_FIRST_BLOCK** uložena log. 1. Jestliže je přenos platný, ale není očekáván první blok paketu, je do tohoto registru uložena log. 0. V následujícím taktu jsou hodnoty AXI4-Stream registrů odeslány do jednotky CTR společně s příznakem `OUT_FIRST` (toho je potřeba na konci šifrování). Pokud je odesílán první blok, je také signál `LOAD` nastaven do log. 1 – což pro jednotku CTR znamená načtení nového klíče (`KEY_VALUE`) a čítače (`CTR_VALUE`).

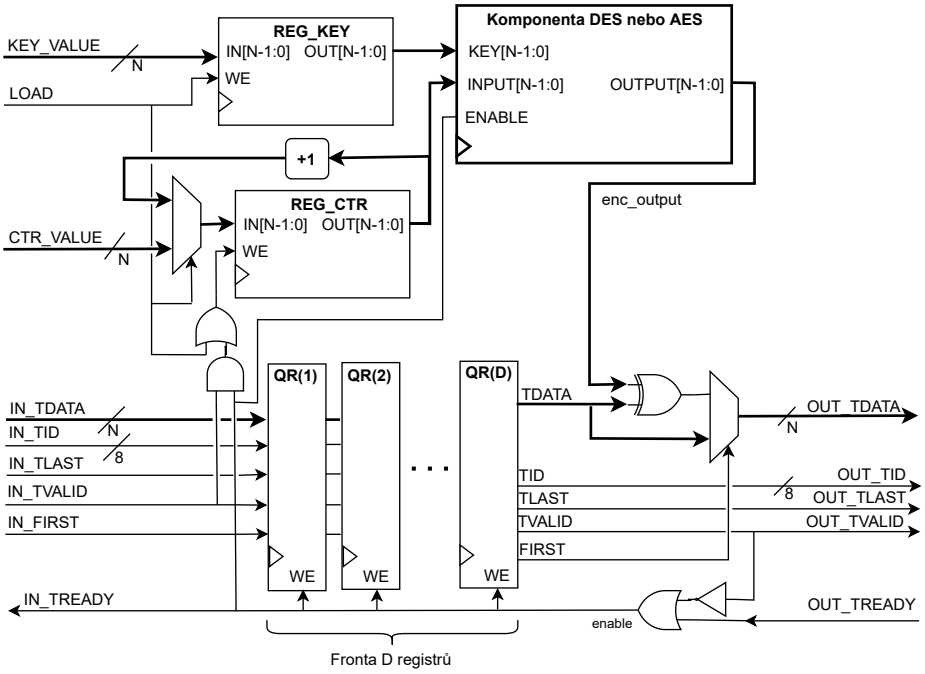


Obrázek 5.3: Schéma detektoru prvního bloku

5.3 Šifrovací jednotka v režimu CTR

Architektura šifrovací jednotky v režimu CTR je zakreslena ve schématu 5.4, a sestává z registru pro šifrovací klíč (**REG_KEY**), čítače (**REG_CTR**), komponenty DES nebo AES a fronty D registrů (pro DES $D = 49$, pro AES $D = 21$). Do registru pro klíč je zapsána hodnota ze vstupu `KEY_VALUE` vždy, když je vstupní signál `LOAD` v log. 1. Hodnota v registru slouží jako vstup na port `KEY` komponenty DES nebo AES. Do registru čítače je zapsána hodnota ze vstupu `CTR_VALUE`, pokud je signál `LOAD` v log. 1. Jinak, pokud na vstupním rozhraní AXI4 může proběhnout platný přenos, se hodnota čítače zvýší o 1. Hodnota čítače slouží jako vstup na port `INPUT` komponenty DES nebo AES. Hodnota čítače zašifrovaná klíčem je po D hodinových taktech dostupná na portu `OUTPUT` komponenty DES nebo AES – tento signál je označen jako `enc_output`. Vstupní signály z rozhraní AXI4 `IN_TDATA`, `IN_TID`, `IN_TLAST`, `IN_TVALID` spolu s příznakem `IN_FIRST` jsou uloženy do prvního registru **QR(1)** (queue register) ve frontě. V každém hodinovém taktu se obsah každého registru ve frontě uloží do následujícího registru. Počet registrů ve frontě odpovídá prodlevě zřetězené linky šifrování v komponentě DES nebo AES, a slouží k synchronizaci datového bloku s jemu příslušející hodnotou čítače, která prochází šifrováním. Z posledního registru ve frontě **QR(D)** jsou data odesílána na výstupní rozhraní AXI4-Stream šifrovacího modulu. Jestliže data nesou první blok paketu, tedy inicializační vektor, je příznak `FIRST` v log. 1 a v tom případě se na výstupní port `OUT_TDATA` odesílá přímo inicializační vektor (`TDATA`). Jinak se jedná o blok zprávy, který je v režimu CTR vymaskován se zašifrovanou hodnotou čítače. Na výstup `OUT_TDATA` je tedy přivedena hodnota `TDATA` $\oplus$ `enc_output`.

Jednotkou vede řídicí signál, označený jako `enable`, a určuje, zda je celý šifrovací modul v provozu. Signál `enable` je vždy aktivní (v log. 1) kromě situace, kdy jsou připravena výstupní data pro přenos (výstup z posledního registru fronty **QR(D)** má `TVALID` v log. 1), ale komponenta odebírající výstup z modulu je nemůže přijmout, takže signál `OUT_TREADY` je nastaven v log. 0. Hodnota `enable` je tedy získána jako `TVALID` $\vee$ `OUT_TREADY`. Pokud je `enable` deaktivován (v log. 0), je zakázán zápis do všech registrů fronty **QR(1)** až **QR(D)** a je pozastaven čítač a zřetězená linka šifrování komponenty DES nebo AES. Signál `enable` je také přiveden do jednotky pro detekci prvního bloku paketu přes port `IN_TREADY`, takže i funkce tohoto modulu je pozastavena.



Obrázek 5.4: Schéma šifrovací jednotky režimu CTR

Jestliže komponenta odebírající zašifrovaná data z modulu je vždy připravena přijímat, pak i šifrovací modul je schopen v každém taktu přijmout a odeslat nová data, a dosahovat tak maximální propustnosti.

Kapitola 6

Výsledky

Výsledný šifrovací modul a komponenty AES a DES byly implementovány v jazyce VHDL. Byly vytvořeny dvě varianty výsledného modulu: Pro šifru DES, využívající komponentu DES se 2 registry ve vnitřním zřetězení iterací, a pro šifru AES, využívající komponentu AES s jedním registrem ve vnitřním zřetězení.

6.1 Funkční verifikace a syntéza

Nad oběma verzemi byla provedena funkční verifikace za použitím nástroje ModelSim (Intel FPGA Starter edition 2020.3). Pro verifikaci byl vygenerovány pakety s náhodným obsahem a náhodnou délkou a k nim příslušné náhodně generované šifrovací klíče a inicializační vektory. Ke každému paketu byl vytvořen referenční šifrovaný výstup, pro DES pomocí vlastního skriptu implementující režim CTR, pro AES za použití nástroje openssl. Během verifikace byly prvně do paměti klíčů zapsány všechny náhodně vygenerované šifrovací klíče. Poté byly vygenerované pakety, s náhodnými časovými rozestupy mezi sebou a také mezi jednotlivými přenosy v paketu, odesílány na vstupní rozhraní šifrovacího modulu. Zašifrované pakety pak byly odebírány z výstupního rozhraní, a porovnány s referenčními šifrovacími výstupy.

Pomocí náhodných časových rozestupů mezi přenosy a náhodných délek paketů byla ověřována správná implementace řídicích signálů, a správná časová synchronizace všech jednotek, zejména načítání nového klíče a čítače, zpoždění fronty nebo maskování jednotlivých odesílaných bloků odpovídajícím zašifrovaným čítačem. Náhodně voleným klíčem, inicializačním vektorem a obsahem paketů se pak ověřovala správná funkčnost komponenty DES nebo AES, tedy korektní implementace šifrovacího algoritmu a také synchronizace kroků zřetězených linek. Pro oba moduly bylo vygenerováno 1000 paketů s délkou 1 až 100 bloků, a všechny verifikací úspěšně prošly. Lze tedy předpokládat, že moduly jsou pravděpodobně implementovány správně.

Po funkční verifikaci byla provedena syntéza samostatných komponenty DES a AES, a obou variant šifrovacího modulu DES a AES pro zařízení Intel Arria 10 SX 480. Syntéza byla provedena v softwaru Intel Quartus Prime Pro Edition nástrojem Precision Synthesis, s nastavením pro maximální optimalizaci rychlosti výsledného obvodu. V tabulce 6.1 jsou uvedeny dostupné zdroje zařízení a spotřeba těchto zdrojů u jednotlivých modulů. Jsou to základní logické bloky ALM (adaptive logic module), registry a bloky vestavěné paměti (RAM). Dále je uvedena maximální pracovní frekvence obvodů (Fmax), a propustnost modulu získaná vynásobením Fmax s šířkou datového bloku (64 resp. 128 bitů).

Zařízení	ALM	Registry	Bloky RAM	Fmax	Propustnost
Intel Arria 10 SX 480	183 590	734 360	1 431	800 MHz	–
Komponenta DES	2 000 (1 %)	7 636	0	489 MHz	31 296 Mb/s
Komponenta AES	10 135 (6 %)	7 683	0	423 MHz	54 144 Mb/s
Šifrovací modul (DES)	2 187 (1 %)	8 339	8 (< 1 %)	410 MHz	26 240 Mb/s
Šifrovací modul (AES)	10 288 (6 %)	8 873	9 (< 1 %)	271 MHz	34 688 Mb/s

Tabulka 6.1: Dostupné zdroje a výsledky syntézy šifrovacích modulů

Výsledky ukazují, že většinu zdrojů ve výsledných šifrovacích modulech (DES nebo AES) zabírají příslušné komponenty DES nebo AES, v nichž všechny zdroje zabírají registry a kombinační logika, realizující příslušný šifrovací algoritmus. U šifrovacích modulů je také větší spotřeba blokových pamětí, než je třeba na paměť klíčů (u DES jsou potřeba 2 blokové RAM, u AES 4 blokové RAM). Tyto další bloky byly automaticky vygenerovány nástrojem pro syntézu namísto některých registrů ve frontě jednotky CTR, pro účely optimálnější rychlosti a úspory zdrojů.

6.2 Srovnání se softwarovým řešením

Pro algoritmus DES není šifrovací režim CTR standardizovaný [1]. Implementace v této práci je tedy experimentální, a žádná softwarové řešení DES v režimu CTR nebyla nalezena. Pro srovnání algoritmu AES byla vybrána vysoce optimalizovaná implementace AES-128-CTR pro vestavěný procesor ARM Cortex-M4 [11]. Zde je výkon měřen jako průměrný počet cyklů procesoru potřebný k zašifrování jednoho datového bloku, což na procesoru ARM Cortex-M4 činí 554,4 cyklů/blok, kdy jeden blok odpovídá 128 bitům. Jádro procesoru umístěné na použité vývojové desce běží na frekvenci 168 MHz, z čehož se vypočítá průměrná propustnost softwarové implementace AES šifrování T_{aes_sw} následovně:

$$T_{aes\_sw} = \frac{frequency}{cycles\_per\_block} * block\_size = \frac{168 * 10^6}{554.4} * 128 = 38.788\ Mbit/s \qquad (6.1)$$

Softwarová implementace algoritmu AES v režimu CTR pro procesor pro vestavěné systémy je tedy oproti realizaci v obvodu FPGA, která dosahuje propustnosti 34,688 Gbit/s, dle teoretické rychlosti asi 894krát pomalejší. Tato softwarová implementace [11] byla přitom speciálně optimalizovaná na výkon a dosahovala asi dvojnásobné rychlosti oproti jiným, existujícím implementacím AES. Z těchto výsledků plyne, že vytvořená implementace přináší možnost vysokorychlostní šifrované komunikace při použití ve vestavěných systémech, a je možné očekávat řádově tisícinásobné zvýšení propustnosti oproti softwarovým implementacím.

Kapitola 7

Závěr

Cílem práce bylo navrhnout hardwarovou realizaci šifrovacích algoritmů DES a AES tak, aby bylo dosaženo požadované propustnosti 10 Gbit/s. Všechny stanovené cíle zadání byly splněny. Nejprve jsem nastudoval problematiku šifrovacích algoritmů DES a AES. Výsledky studia jsem shrnul do kapitoly 2 teoretického rozboru práce. Kapitola obsahuje stručný úvod do kryptografie a poté detailně popisuje proces šifrování a dešifrování algoritmů DES a AES, a režimy operace těchto algoritmů. Pro implementaci je v této práci zvolen režim čítače (režim CTR). Na základě znalosti algoritmů DES a AES byl proveden návrh obvodových realizací šifrovacích procesů těchto algoritmů, který je popsán v kapitole 3. Obvodové realizace byly navrženy s ohledem na maximální propustnost za použití rozvinutí iterací algoritmů, takže výsledný obvod tvoří zřetězenou linku. Ke každé z těchto obvodových realizací byly pak vytvořeny další dvě varianty využívající vnitřní zřetězení iterací, tedy další optimalizační registry mezi částmi každé iterace pro zkrácení kritických cest obvodu, s cílem dosáhnout vyšší propustnosti zvýšením maximální možné hodinové frekvence obvodu. Pro oba algoritmy tak byly vytvořeny 3 varianty: první bez vnitřního zřetězení, druhá využívající jeden optimalizační registr a třetí využívající dva optimalizační registry ve vnitřním zřetězení iterací. Poté byla provedena implementace v jazyce VHDL a syntéza všech těchto variant, a jejich porovnání z hlediska propustnosti. Jak pro DES, tak pro AES dosáhla nejvyšší propustnosti varianta se dvěma optimalizačními registry.

Dále byl proveden návrh modulu realizující šifrovací režim čítače, opět s cílem dosáhnout maximální propustnosti. Architektura modulu je popsána v kapitolách 4 a 5. V kapitole 4 je popsáno datové rozhraní AXI4-Stream, jeho zapojení do výsledné architektury modulu a formát zpráv určených k šifrování a dešifrování. V kapitole 5 je pak popsána architektura celého výsledného modulu sestávající s paměti šifrovacích klíčů, vstupní řídicí jednotky a jednotky implementující šifrování v režimu čítače algoritmu DES nebo AES. Modul byl implementován v jazyce VHDL tak, aby mohl být syntetizován jak s komponentou DES, tak s komponentou AES, představené v kapitole 3.

Nad oběma variantami šifrovacího modulu byla provedena funkční verifikace. Poté byla provedena syntéza obou variant šifrovacího modulu pro cílový obvod Intel Arria 10 SX 480 a bylo provedeno vyhodnocení propustnosti a použitých zdrojů. Varianta modulu DES spotřebuje asi 1 % zdrojů cílového obvodu, a dosahuje propustnosti 31,29 Gbit/s při maximální frekvenci 410 MHz. Varianta modulu AES spotřebuje asi 6 % zdrojů a dosahuje propustnosti 32,68 Gbit/s při maximální frekvenci 271 MHz. V obou případech byl tedy splněn požadavek dosáhnout minimální propustnosti 10 Gbit/s. Nakonec byla vytvořená implementace pro FPGA srovnána se softwarovým řešením pro vestavěný mikroprocesor, přičemž softwarové řešení dosahuje řádově tisíckrát horší propustnosti v řádu desítek Mbit/s. Výsledky

verifikace, syntézy, jejich vyhodnocení, a srovnání s existujícím softwarovým řešením jsou shrnuty v kapitole 6.

V optimalizaci propustnosti navržených architektur by bylo možné dále pokračovat, například implementací některých operací algoritmů s použitím blokových pamětí namísto kombinačních LUT, nebo použitím jiných výpočetních postupů operací na úrovni kombinačních LUT. Také by bylo možné umístit dva nebo více modulů vedle sebe a šifrovat více vstupních bloků naráz, což to režim CTR umožňuje díky tomu, že hodnotu čítače lze předpočítat na více kroků dopředu. Čtyři takto paralelně umístěné moduly by měly šanci na dosažení propustnosti až 100 Gbit/s.

Literatura

[1] *DES modes of operation* [online]. National Institute of Standards and Technology, prosinec 1980 [cit. 2021-05-05]. Dostupné z: https://csrc.nist.gov/csrc/media/publications/fips/81/archive/1980-12-02/documents/fips81.pdf.

[2] *Data encryption standard (DES)* [online]. National Institute of Standards and Technology, říjen 1999 [cit. 2021-05-05]. Dostupné z: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf.

[3] *Advanced encryption standard (AES)* [online]. National Institute of Standards and Technology, listopad 2001 [cit. 2021-05-05]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[4] *AMBA 4 AXI4-Stream Protocol specification* [online]. IHI 0051A. ARM Holdings, březen 2010 [cit. 2021-05-05]. Version 1.0. Dostupné z: https://developer.arm.com/documentation/ihi0051/a.

[5] *Intel Arria 10 Device Overview* [online]. A10-OVERVIEW. Intel Corporation, říjen 2020 [cit. 2021-05-05]. Dostupné z: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf.

[6] Abdellatif, K. M. Hardware Implementation of DES Using Pipelining Concept with Time-Variable Key. In: [online]. 22nd International Conference on Microelectronics (ICM 2010), Prosinec 2010 [cit. 2021-05-05]. Dostupné z: https://www.researchgate.net/publication/234111976_Hardware_Implementation_of_DES_Using_Pipelining_Concept_with_Time-Variable_Key.

[7] Biryukov, A. Feistel Cipher. In: Tilborg, H. C. A. van a Jajodia, S., ed. *Encyclopedia of Cryptography and Security*. Boston, MA: Springer US, 2011, s. 455–455. DOI: 10.1007/978-1-4419-5906-5_577. ISBN 978-1-4419-5906-5. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_577.

[8] Hodjat, A. a Verbauwhede, I. A 21.54 Gbits/s fully pipelined AES processor on FPGA. In: *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. 2004, s. 308–309. DOI: 10.1109/FCCM.2004.1.

[9] Katz, J. a Lindell, Y. *Introduction to modern cryptography*. 1. vyd. Boca Raton: Chapman Hall/CRC, 2008. Chapman Hall/CRC cryptography and network security. ISBN 978-1-58488-551-1.

[10] Lipmaa, H., Rogaway, P. a Wagner, D. *Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption* [online]. Září 2000 [cit. 2021-05-05].

Dostupné z: https://www.researchgate.net/publication/2817314_Comments_to_NIST_concerning_AES-modes_of_operations_CTR-mode_encryption.

[11] Schwabe, P. a Stoffelen, K. *All the AES You Need on Cortex-M3 and M4* [online]. Nijmegen The Netherlands: Radboud University Digital Security Group, 2016 [cit. 2021-05-05]. Dostupné z: https://eprint.iacr.org/2016/714.pdf.