VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE ŠIFROVACÍCH ALGORITMŮ V JAZYCE VHDL

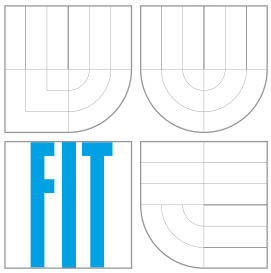
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR KOŽENÝ

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE ŠIFROVACÍCH ALGORITMŮ V JAZYCE VHDL

IMPLEMENTATION OF ENCRYPTION ALGORITHMS IN VHDL LANGUAGE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR KOŽENÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KOŘENEK

BRNO 2008

Abstrakt

Tato práce se zabývá návrhem a implementací šifrovacích algoritmů AES a DES pro vestavěné systémy. Architektury jsou implementovány v jazyce VHDL a navrženy pro umístění do programovatelných logických polí FPGA. Výsledná řešení jsou určena pro obvody Xilinx Spartan 3. Obě architektury pracují v režimu ECB (Electronic Codebook) s použitím vyrovnávacích pamětí. Maximální propustnost navrženého architektury DES je 370 Mb/s při pracovní frekvenci 104 MHz. Pro šifrování algoritmem AES je propustnost na maximální frekvenci 118 MHz až 228 Mb/s. Při porovnání se softwarovými implementacemi, určenými pro vestavěné systémy, dosáhly obě architektury podstatně vyšších propustností.

Klíčová slova

DES, AES, šifrovací algoritmy, kryptografie, FPGA, ECB, vestavěné systémy

Abstract

This thesis deals with design and implementation of AES and DES encryption architectures for embedded systems. Architectures are implemented in VHDL language and design for FPGA technology. The proposed implementations are mapped on the Xilinx Spartan 3 technology. Both architectures are applied in simple ECB (Electronic Codebook) scheme with cache memories. A maximum throughput of design DES architecture 370 Mbps is achived with clock frequency of 104 MHz. The throughput of AES architecture at the maximum clock frequency of 118 MHz is 228 Mbps. Compared to software implementations for embedded systems, we achieve significantly higher throughput for both architectures.

Keywords

DES, AES, encryption algorithms, cryptography, FPGA, ECB, embedded systems

Citace

Petr Kožený: Implementace šifrovacích algoritmů
v jazyce VHDL, diplomová práce, Brno, FIT VUT v Brně, 2008

Implementace šifrovacích algoritmů v jazyce VHDL

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Kořenka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Kožený
15. května 2008

Poděkování

Děkuji Ing. Janu Kořenkovi za poskytnutí literatury, odborné pomoci, cených připomínek a rad, které mi pomáhaly při tvorbě této práce

Obsah

Kapitola 1

Úvod

Počítačové systémy a sítě stále přichází s novými požadavky na vyšší rychlost a bezpečnost dat. Na bezpečnost je kladen velký důraz z důvodu stále častějších útoků na počítačové systémy a stává se jedním z nejdůležitějších faktorů při vývoji celého spektra aplikací. Pro ochranu přenosu a uložení citlivých dat jsou velmi často používané symetrické šifrovací algoritmy. Od sedmdesátých let byl jednoznačně nejdůležitějším algoritmem v oblasti symetrické kryptografie algoritmus DES, který se stal standardem pro šifrování dat. I přesto, že se o jeho bezpečnosti spekulovalo již při jeho vzniku, byl další standart přijat až v roce 2001 a stal jím algoritmus AES, který postupně nahrazuje již zastaralý algoritmus DES.

Šifrovací algoritmy lze realizovat jak softwarově, tak hardwarově. Při softwarové implementaci běží aplikace na univerzálním procesoru, kde se jednotlivé operace vykonávají sekvenčně, neuvažujeme-li paralelní zpracování na úrovni více procesorů nebo procesorových jader. V případě hardwarové implementace je možné navrhnout specializovaný obvod určený pro konkrétní úlohu, jakou může být třeba šifrování vstupního bloku dat. Při návrhu lze uvažovat různé aspekty popisovaných algoritmů a ty využívat pro zařazení různých optimalizací v podobě např. zřetězení nebo paralelismu, což lze při vytváření softwarového řešení pouze v omezené míře. Výsledkem kvalitní hardwarové implementace může být obvod s daleko vyšší propustností, nižší cenou a spotřebou, než u klasického univerzálního výpočetního systému. Nevýhodou tohoto řešení je naopak nákladnější vývoj, proto se toto řešení nejlépe uplatňuje při sériové výrobě, při které se vyrobí velké množství těchto obvodů, nebo při výrobě zařízení, u něhož je kladen velký důraz na rychlost a spotřebu. V případě hardwarové implementace algoritmů DES a AES se dosahuje několiksetkrát až tisíckrát vyšší propustnosti než u řešení softwarového.

Cílem této práce je navrhnout vhodnou hardwarovou realizaci algoritmů AES a DES, s požadavkem na možnost jejich umístění do programovatelného logického pole FPGA, provést jejich implementaci v jazyce VHDL a následnou syntézu. Výsledky hardwarových realizací budou porovnány se softwarovými implementacemi a poskytují možnost srovnat algoritmy mezi sebou. Požadovanou vlastností obou realizací je minimální propustnost 100 Mb/s pro jejich možné využití v síťových aplikacích, a to při snaze o minimální řešení z hlediska obsazenosti zdrojů v obvodu FPGA.

Práce je členěna do několika logických celků, organizovaných následovně. Kapitola 2 je věnována teoretickému rozboru problematiky. Po krátkém shrnutí historie kryptografie a úvodu do symetrického šifrování jsou zde detailně popsány principy a vlastnosti jednotlivých šifrovacích algoritmů DES a AES. V Kapitole 3 je prezentován návrh hardwarové realizace algoritmu DES. Jsou zde popsána zapojení jednotlivých operací, které se uplatňují při šifrování, dále jejich úloha a umístění v rámci celé architektury. V kapitole jsou dále uvedeny

vlastnosti architektury a srovnání implementace se softwarovým řešením. Kapitola 4 se zabývá popisem vnitřního zapojení komponent v šifrování algoritmem AES a jejich zařazení do výsledné architektury. Kapitola také popisuje metody použité pro minimalizaci výsledné architektury. Na závěr shrnuje vlastnosti výsledné implementace a porovnává se softwarovou realizací. V závěrečné kapitole 5 jsou uvedeny dosažené výsledky a rozvedena možnost dalšího vývoje.

Kapitola 2

Teoretický rozbor

Kořeny šifrování sahají do dávné minulosti a šifrování hrálo důležitou roli v mnoha důležitých a zlomových okamžicích naší historie. Například se stalo jedním z rozhodujících faktorů ve výsledku obou světových válek, kdy třeba prolomení šifrovacího stroje Enigma umožňovalo Spojencům dekódovat německé zprávy, což se ukázalo obrovskou výhodou, která přispěla k celkovému vítězství Spojenců. První vojenské šifrovací zařízení využívající transpoziční šifry pochází již ze Sparty z 5. století, substituční šifrování používal již Julius Caesar. V Evropě došlo k rozmachu kryptografie v 15. století, kdy evropští panovníci i Vatikán nechávali šifrovat své diplomatické zprávy. V 18. století se začalo šifrování používat na průmyslové úrovni, což vedlo k velkému rozvoji oboru kryptografie. Obrovský pokrok znamenaly pro šifrování války, zvláště první a druhá světová. Asi nejznámější se stala šifra Enigma, kterou používala za druhé světové války německá armáda. [3, 4, 27]

Začátky moderní kryptografie sahají do 70. let 20. století, kdy byl v roce 1976 přijat jako první šifrovací standard DES. Velký průlom znamenal rok 1977, kdy vědci Rivest, Shamá a Adleman vymysleli jednosměrnou funkci splňující podmínku asymetrického šifrování. Jejich algoritmus byl nazván RSA a dosud nebyl prolomen. Pro snadnější použití šifrovacích algoritmů ve veřejném sektoru byla vyvinuta spousta softwaru umožňujícího ochranu dat a datové komunikace. Tyto softwary využívají všechny dnes dostupné algoritmy, jako je např. IDEA, BlowFish, SkipJack nebo RC5. Platí všeobecné pravidlo, že se nesmí vyvážet do teroristických zemí. V roce 1997 se ukázalo, že není problémem napsat software, který nechá paralelně pracovat tisíce počítačů po celém světě a šifru DES prolomit, proto byla v tomto roce vyhlášena soutěž na symetrický algoritmus, který by DES nahradil. Dne 26. 5. 2002 byla schválena šifra AES jako standard. [27]

2.1 Symetrická kryptografie

Symetrická kryptografie využívá k šifrování i dešifrování pouze jediný klíč. Ten musejí znát obě strany, jak strana vysílající zprávu, tak strana zprávu přijímající. Podstatnou výhodou symetrických šifer je jejich nízká výpočetní náročnost. Algoritmy pro šifrování s veřejným klíčem mohou být i několikanásobně pomalejší. Na druhou stranu velkou nevýhodou symetrického šifrování je distribuce klíčů. Aby mohli dvě strany komunikovat, musí si klíč předat (často elektronicky) a zde vzniká nebezpečí, že se k němu dostane nepovolaná osoba. Symetrické šifry se z hlediska zpracování otevřeného textu dělí na dva základní druhy a to proudové a blokové. Proudové zpracovávají otevřený text po znacích abecedy. Jsou jimi například RC4 a FISH. Blokové šifry zpracovávají informace po blocích dané délky, přičemž

poslední blok přizpůsobí vhodným způsobem délce jednoho bloku. Podstatné u blokových šifer je, že všechny bloky jsou šifrovány a dešifrovány stejnou transformací. Příklady blokových šifer jsou AES, DES, 3DES, IDEA, Blowfish. Většina těchto šifer používá blok o délce 64 bitů, avšak již se přechází na blok 128 bitů, který používá algoritmus AES. [12, 1]

2.2 Algoritmus DES

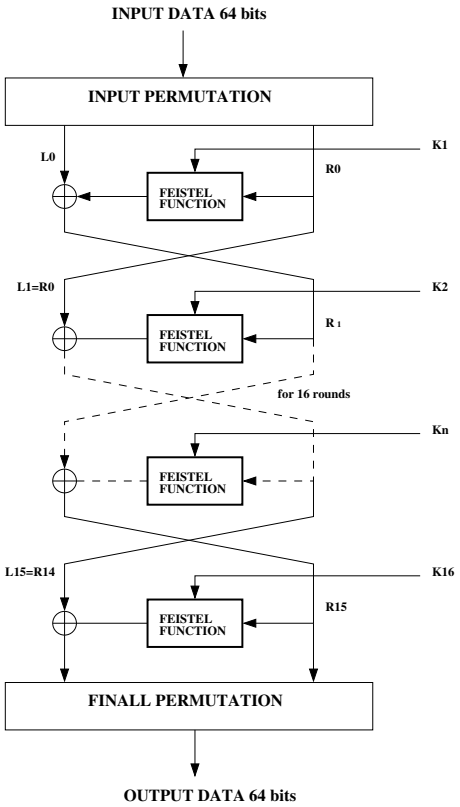
Algoritmus DES [20] je prvním z veřejných kryptografických algoritmů. I přes řadu v současné době známých chyb je velice populární a často používán. V roce 1973 vyhlásilo Ministerstvo obchodu USA soutěž na vytvoření šifrovacího standardu, který by dostatečně zabezpečil ochranu důvěrných dat v informačních systémech. Vysokým nárokům však nevyhověl žádný z navrhovaných algoritmů, a proto se soutěž konala v roce 1974 znovu. Vítězem se stala firma IBM. Ta nevyvinula zcela nový algoritmus, ale 'pouze' zdokonalila svůj stávající šifrovací algoritmus Lucifer, jenž byl vyvinut výzkumným týmem pod vedením doktora Tuchmana. Šifrovací algoritmus DES prošel i bezpečnostním hodnocením National Security Agency (též známé pod zkratkou NSA). V roce 1975 si jej firma IBM nechala patentovat. Zároveň umožnila její bezplatné používání na území USA. V březnu téhož roku byl zveřejněn šifrovací algoritmus DES. V listopadu 1976 byl DES přijat jako šifrovací standard pro zabezpečení neutajovaných dat v civilním a vládním sektoru s předpokládanou délkou používání deset až patnáct let, s podmínkou, že jeho bezpečnost bude každých pět let kontrolována. Díky masovému rozšíření šifrovacího standardu DES však vznikl nový problém, jak ve velkém měřítku nahradit tento šifrovací standard novějším a lepším. DES se totiž stal neoficiálním mezinárodním standardem ve veřejném i soukromém sektoru.

Již v roce 1975 se začalo spekulovat o jeho bezpečnosti. K prvním kritikům patřili Diffie a Hellman. Ti kritizovali zejména nedostatečnou délku šifrovacího klíče. Argumentovali tím, že k rekonstrukci zašifrovaného textu stačí pouze vyzkoušet všechny možné kombinace šifrovacích klíčů. Na základě jejich výhrad byla svolána konference, která dospěla k závěru, že tento druh útoku v reálném čase bude možné praktikovat až technologiemi vyvinutými po roce 1990. V roce 1997 vypsala agentura RSA kryptoanalytickou soutěž, v níž bylo cílem rozluštit text se známým začátkem a délkou šifrovacího klíče 56 bitů. Po necelých pěti měsících byla šifra prolomena. Tímto činem si Rocke Versen, vedoucí týmu DES Challenge, vydělal 10 000 dolarů a zároveň dokázal reálnou prolomitelnost šifrovacího standardu DES. K dešifrování zprávy využil Internet - s pomocí týmu luštitelů zkoušeli kombinace klíčů, dokud nebyla zpráva čitelná. Kvůli relativně nízké délce klíče byl přijat standard známý pod názvem Triple DES (TDES, 3DES). Ten se od klasického DES liší tím, že stejná data projdou algoritmem třikrát, čímž se zvýší efektivní délka klíče.

2.2.1 Popis algoritmu

Algoritmus DES [20] je určen pro šifrování a dešifrování bloků o velikosti 64 bitů pomocí 64-bitového klíče, každý osmý bit je ovšem kontrolní, efektivní délka klíče je tedy pouze 56 bitů. Dešifrování se provádí pomocí stejného klíče jako šifrování a proces dešifrování je inverzní k procesu šifrování. Blok určený k šifrování je podroben úvodní permutaci IP, následují šifrovací výpočty pomocí šifrovacího klíče a nakonec je vykonána inverzní permutace IP^{-1}.

Výpočty prováděné pomocí šifrovacího klíče lze jednoduše definovat funkcí F - šifrovací funkce a funkcí KS (Key Sheduling) - plánování klíče. Princip algoritmu DES je znázorněn na obrázku 2.1.



Obrázek 2.1: Princip algoritmu DES

2.2.2 Popis operací

Permutace IP

Vstupní blok je v prvním kroku podroben permutaci. Tato permutace je označena jako IP a je znázorněna v tabulce 2.1.

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

Výstup permutace IP bude mít jako první bit ten, který byl původně na pozici 58, druhý bude bit původně z pozice 50 a poslední bit bude z pozice 7. Výstup permutace je vstupem do bloku pro výpočet pomocí šifrovacího klíče.

Inverzní permutace

Po provedení výpočtů pomocí šifrovacího klíče se na závěr aplikuje permutace, která je inverzní k úvodní permutaci. Permutace se provádí podle následující tabulky 2.2, stejným způsobem na vstupní permutace.

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

Tabulka 2.2: Inverzní permutace

Výpočet pomocí klíče

Jedná se o výpočet, jenž používá permutovaný vstup do procedury, která vytvoří předvýstup celého šifrování, na který je aplikována inverzní permutace. Tato procedura zahrnuje 16 iterací výpočtu, který bude později popsán funkcí F, jež operuje se dvěma bloky, jeden 32-bitový a druhý 48-bitový. Výstupem je blok o velikost 32 bitů. Délka vstupu do jednotlivých iterací je 64 bitů, obsahujících 32 bitů označených jako L a 32 bitů označených jako R. Celý vstup do iterace můžeme tedy označit LR. Nechť K je blok o velikosti 48 bitů výpočítaný z 64-bitového klíče. Poté $L'R'$ je výstup iterace při vstupu LR a je definován následujícími vztahy 2.1 a 2.2.

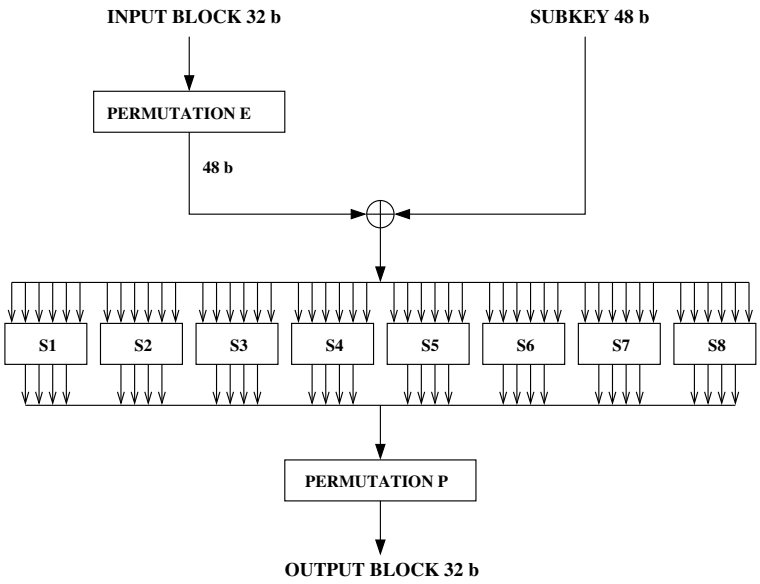
$$L' = R \tag{2.1}$$

$$R' = L \oplus f(R, K) \tag{2.2}$$

Jak už bylo naznačeno, vstup do první iterace výpočtu je permutovaný vstupní blok. Pokud $L'R'$ je výstup 16-té iterace, pak $R'L'$ je předvýstup šifrování. V každé iteraci je použit rozdílný blok K_n, složený ze 48 bitů odvozených z 64-bitového šifrovacího klíče.

Feistelova funkce f

Vstupem této funkce je blok o velikosti 32 bitů, výstupní blok má stejnou. Princip funkce je znázorněno na obrázku 2.2.



Obrázek 2.2: Šifrovací funkce f

Nechť E je permutační funkce, která má na vstupu blok o velikosti 32 bitů a na výstupu blok velikosti 48 bitů. Výstup lze zapsat jako 8 bloků, každý po 6-ti bitech. Tyto bloky jsou vytvořeny výběrem bitů ze vstupní sekvence v pořadí, které znázorňuje tabulka 2.3. První tři bity na výstupu funkce $E(R)$ jsou vstupní bity na pozici 32, 1 a 2 ze vstupního bloku R a poslední dva bity funkce $E(R)$ jsou bity z pozice 32 a 1.

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

Tabulka 2.3: Selekční funkce E

V dalším kroku šifrovací funkce F se aplikuje logická funkce XOR mezi jednotlivými bity výstupu selekční funkce E a vstupním 48-bitovým klíčem. Výstupem této oprace je 48 bitů, které vstupují po 6 bitech do substitučních funkcí S_n. Každá z jedinečných funkcí $S_1, S_2, S_3, ..., S_8$ má na vstupu 6 bitů, které převede na výstupní blok o velikosti 4 bity, za

použití příslušných tabulek. Tabulka 2.4 je doporučena pro funkci S_1. Tabulky pro funkce $S_2 - S_8$ lze nalézt v [20].

row/column	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 2.4: Tabulka pro substituční funkci S1

Pokud je S_1 funkce definovaná předchozí tabulkou a B je blok, který má velikost 6 bitů, potom výsledek $S_1(B)$ se vytváří následujícím způsobem: první a poslední bit vstupu B reprezentují číslo v intervalu $< 0; 3 >$. Tuto hodnotu označíme písmenem i. Prostřední bity bloku B reprezentují číslo, které je v intervalu $< 0; 15 >$. Tuto hodnotu označíme písmenem j. V tabulce pro funkci S_1 nalezneme číslo na i-tém řádku a v j-tém sloupci. Toto číslo je v intervalu $< 0; 15 >$ a určuje 4bitový výstupní blok.

Například pro vstup 011011 je příslušný řádek 01 (řádek číslo 1) a číslo sloupce má binární hodnotu 1101 (sloupec číslo 13). Hodnota v tabulce určená těmito indexy je 5. Binární výstup této funkce je tedy 0101. Čtyřbitové výstupy funkcí dávají dohromady 32-bitovou hodnotu, na kterou se aplikuje permutační funkce P. Tato funkce vytváří z 32-bitového vstupu 32-bitový výstup pomocí bitové permutace vstupního bloku. Funkce je definována tabulkou 2.5.

Výstup funkce $P(L)$ je realizován pomocí tabulky na základě vstupního bloku L. Prvním bitem výstupního bloku je 16-tý bit bloku vstupního, druhý bit na výstupu je sedmý bit ze vstupu atd. Nechť $S_1, ..., S_8$ jsou odlišné selekční funkce, P je permutační funkce a E je definovaná selekční funkce. Pro definici výstupu funkce si nejprve nadefinujeme mezivýsledek v podobě 6-bitových bloků $B_1, ..., B_8$:

$$B_1B_2B_3B_4B_5B_6B_7B_8 = K \oplus E(R) \tag{2.3}$$

Výstup funkce $f(R, K)$ lze poté definovat takto:

$$f(R, K) = P(S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)) \tag{2.4}$$

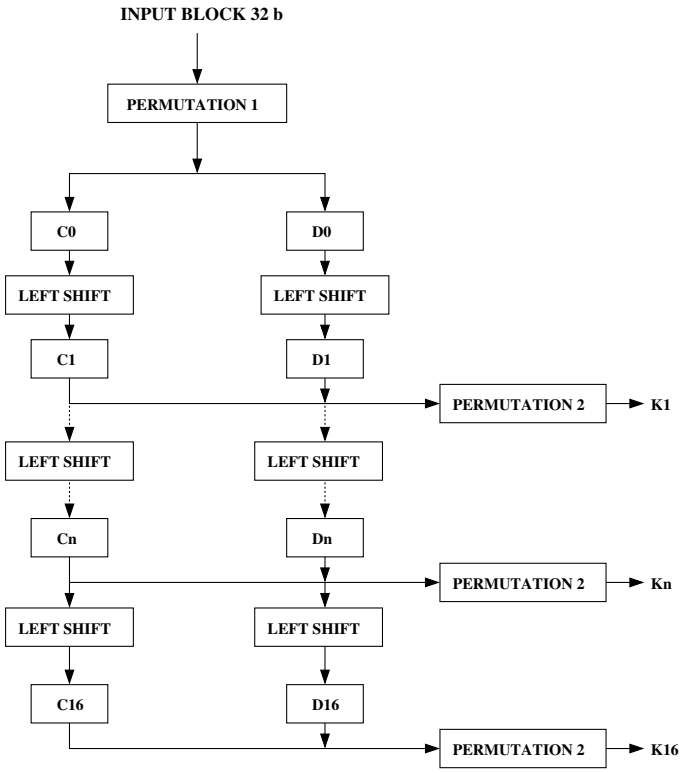
Výsledek funkce se nejprve rozdělí na osm 6-bitových bloků B_n. Jednotlivé bloky se stávají vstupem funkcí S_n. Výstupy těchto osmi funkcí, každý o délce 4 bity, se spojí do jednoho bloku o velikosti 32 bitů, který se stává vstupem permutační funkce P. Výstup této funkce se pak stane výstupem funkce F.

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 2.5: Permutační funkce P

Rozšíření klíče (Key expansion)

Při tomto procesu se vytváří klíče pro jednotlivé kroky šifrování $K_1, K_2, ..., K_{16}$. Jednotlivé bloky K_n mají velikost 48 bitů a jsou odvozeny ze šifrovacího klíč KEY. Výpočet je znázorněn na obrázku 2.3.



Obrázek 2.3: Výpočet Key expansion

V první kroku se na šifrovací klíč KEY aplikuje permutační funkce PC-1 (tabulka 2.6). Vstupní šifrovací klíč je blok o velikosti 64 bitů. Každý osmý bit je paritní. Délka použitelného klíče je 56 bitů. Funkce PC-1 provede permutaci nad těmito 56 bity a rozdělí je do dvou bloků, kde každý má velikost 28 bitů.

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 2.6: Permutační funkce PC-1

První blok se označuje C_0 a skládá se z bitů 57, 49, 41, ... , 36 vstupního klíče KEY. Druhý blok se označuje D_0 obsahuje bity 63, 55, 47,..., 4 vstupního šifrovacího klíče. Při tomto procesu se postupně počítají bloky C_n a D_n pro $n \in < 1, 16 >$. Bloky C_n a D_n se vytvářejí z hodnot bloků C_{n-1} a D_{n-1} aplikací levého bitového posuvu. Počet posouvaných bitů se v jednotlivých iteracích liší podle následující tabulky 2.7.

Iterace	Počet posouvaných bitů
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Tabulka 2.7: Posun jednotlivých bitů v jednotlivých iteracích Key expansion

Hodnota klíče K_n pro jednotlivé kroky šifrování jsou výstupem permutační funkcí PC-2. Vstupem této operace je blok o velikosti 56 bitů, který se vytvoří bitovou konkatenací bloků C_nD_n. Výstupem je blok o velikosti 48 bitů. Permutační funkce PC-2 je definována následující tabulkou 2.8:

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	25	8
16	7	27	20	13	2
41	52	31	45	33	53
16	7	20	21	2	4
44	49	39	56	34	53
46	42	50	36	29	32

Tabulka 2.8: Permutační funkce PC-2

První bit bloku K_n je 14tý bit z bloku C_nD_n, druhým bitem je 17tý atd.

2.3 Algoritmus AES

V září roku 1997 vyhlásil NIST (National Institute of Standards and Technology) veřejnou soutěž na symetrický šifrovací algoritmus, který by nahradil stávající algoritmus DES. Měl umožňovat šifrování bloků dat o velikosti 128 bitů s použitím klíčů o velikosti 128, 192 nebo 256 bitů. Požadavky na nový algoritmus byly tři: bezpečnost, cena a lepší možnost softwarové implementace. O devět měsíců později bylo do soutěže přihlášeno patnáct různých algoritmů (CAST-256, CRYPTON, DEAL, DFC, E2, FROG, HPC, LOKI97, MAGENTA, MARS, RC6, Rijndael, SAFER+, Serpent a Twofish), které byly podrobeny náročnému zkoumání a posuzování. V letech 1998 a 1999 uspořádal NIST dvě konference, na kterých se probíraly průběžné výsledky zkoumání.

V srpnu roku 1999 bylo vybráno pět finalistů (MARS, RC6, Rijndael, Serpent a Twofish). Všech těchto pět finalistů obstálo při kryptoanalýze i svým výkonem v různých prostředích. Následovalo další kolo intenzivního analyzování, které v dubnu 2000 vyústilo v další konferenci. Na ní představitelé všech pěti finalistů prezentovali své argumenty, proč by měl být jejich standard zvolen jako AES. Již 2. října 2000 NIST vyhlásil jako vítěze algoritmus Rijndael autorů Joana Daemena a Vincenta Rijmena. O rok později, 26. 11. 2001, NIST publikoval algoritmus AES jako standard U. S. FIPS PUB 197 (FIPS 197) s účinností od 26. 5. 2002. V současnosti je AES jedním z nejpopulárnějších algoritmů používaných při symetrickém šifrování.

2.3.1 Popis algoritmu AES

I v případě AES [21] se jedná o symetrickou blokovou šifru, kde délka šifrovaného a dešifrovaného bloku je 128 bitů a délka klíče může mít velikost 128, 192 nebo 256 bitů. Očekává se, že algoritmus AES bude mít život minimálně 20 až 30 let, protože mu v tomto časovém horizontu s největší pravděpodobností nehrozí útok hrubou silou, který již dnes dokáže prolomit šifrování pomocí DES. Není ale třeba se obávat, že by platnost AES musela být po uplynutí 30 let z bezpečnostních důvodů ukončena kvůli krátkým klíčům. Důležitou vlastností je, že AES je veřejně dostupný standard, za jehož použití se neplatí žádné licenční poplatky. Nese razítko amerického standardizačního úřadu NIST a od 26. 5. 2002 je možné jej v americké státní správě používat k ochraně citlivých neutajovaných informací. Očekává se, že se i ve světě stane převládajícím symetrickým algoritmem, jako tomu bylo u algoritmu DES před 25 lety.

Pro algoritmus AES je délka vstupního bloku, výstupního bloku a mezivýsledku State 128 bitů. Tuto hodnotu reprezentuje hodnota $N_b = 4$, což je počet 32-bitových slov (počet sloupců) v mezivýsledku. Pro AES může být délka šifrovacího klíče rovna 128, 192 nebo 256 bitů. Délka klíče je vyjádřena hodnotou $N_k = 4, 6$ nebo 8 . Vstupní blok při šifrování i dešifrování prochází určitým počtem iterací, který závisí na délce šifrovacího klíče. Závislost počtu iterací na délce šifrovacího klíče je uvedena v tabulce 2.9.

	Délka klíče N_k	Velikost bloku N_b	Počet iterací N_r
AES-128	4	4	10
AES-196	6	4	12
AES-256	8	4	14

Tabulka 2.9: Závislost počtu iterací na délce šifrovacího klíče

Šifrovací klíč

Šifrovací klíč je možné vyjádřit ve formě matice o velikosti $N_b \times 4$ bytů, přičemž N_b je rovno 4, 6 nebo 8 pro klíč dlouhý 128, 196, nebo 256 bitů. První 4 bajty klíče tvoří 32-bitové slovo w_0 rozšířeného klíče, další 4 byby slovo w_1 atd.

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

a)

k[0]	k[4]	k[8]	k[12]	k[16]	k[20]
k[1]	k[5]	k[9]	k[13]	k[17]	k[21]
k[2]	k[6]	k[10]	k[14]	k[18]	k[22]
k[3]	k[7]	k[11]	k[15]	k[19]	k[23]

b)

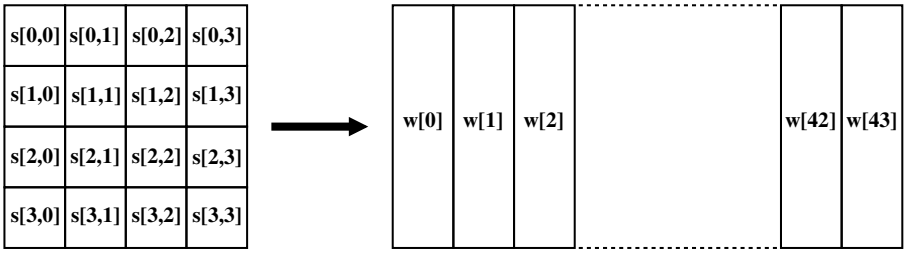
k[0]	k[4]	k[8]	k[12]	k[16]	k[20]	k[24]	k[28]
k[1]	k[5]	k[9]	k[13]	k[17]	k[21]	k[25]	k[29]
k[2]	k[6]	k[10]	k[14]	k[18]	k[22]	k[26]	k[30]
k[3]	k[7]	k[11]	k[15]	k[19]	k[23]	k[27]	k[31]

c)

Obrázek 2.4: Maticové vyjádření šifrovacího klíče a)AES-128 b)AES-192 c)AES-256

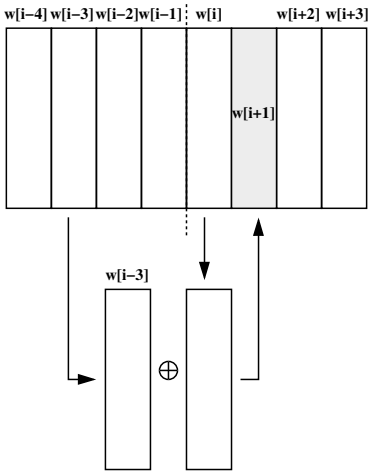
Rozšíření klíče

AES algoritmus bere šifrovací klíč K, na kterém provede Rozšíření klíče (Key Expansion). Tento proces generuje celkový počet $N_b(N_r+1)$ slov, protože při úvodní aplikaci klíče je požadováno N_b slov klíče a v každé z N_r iterací je také potřeba N_b slov z rozšířeného šifrovacího klíče. Výstupem procesu rozšiřování klíče je lineární pole W, které obsahuje $N_b(N_r + 1)$ 4-bytových položek. Jednu položku tohoto pole, která obsahuje 4 byty, lze zapsat jako $w[i]$, kde $i \in (0; N_b(N_r + 1))$.



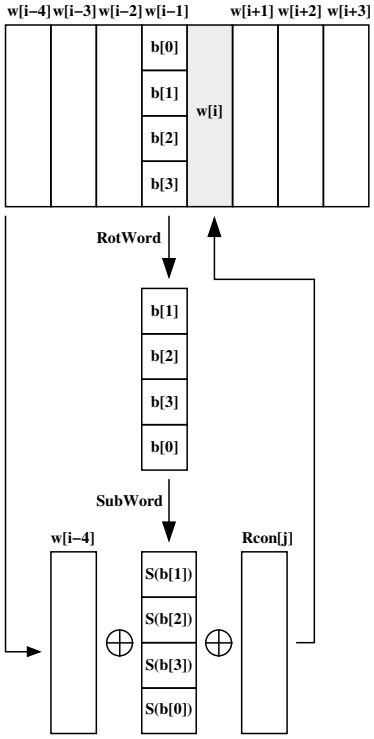
Obrázek 2.5: Rozšíření klíče AES

Operace rozšiřování klíče probíhá v krocích, kde v každém kroku je vytvořeno N_k 4-bytových slov. Prvních N_k slov pole rozšířeného klíče vyplněno hodnotami šifrovacího klíče. Každá další položka $w[i]$ tohoto pole je výsledkem operace XOR provedené na již existujících položkách rozšířeného klíče $w[i-1]$ a $w[i-N_k]$ (obrázek 2.6).



Obrázek 2.6: Key expansion - varianta pro položky, které nejsou násobkem N_k

Pro položky na pozicích, které jsou násobkem hodnoty N_k je tvorba o něco složitější. V první fázi se načte položka $w[i-1]$, na které se postupně provedou operace RotWord, což je operace cyklického posuvu bytů naznačená na obrázku 2.7, a dále operace SubWord (podrobnosti v kapitole o substituci SubBytes), která postupně aplikuje substituci na všechny byty výsledku operace RotWord. Po vykonáních těchto dvou funkcí se na jejich výsledku provede operace XOR s položkou pole konstant Rcon[j] (tabulka 2.10), kde index j odpovídá iteraci, pro kterou je počítaný klíč určen.



Obrázek 2.7: Key expansion - varianta pro položky, které jsou násobkem N_k

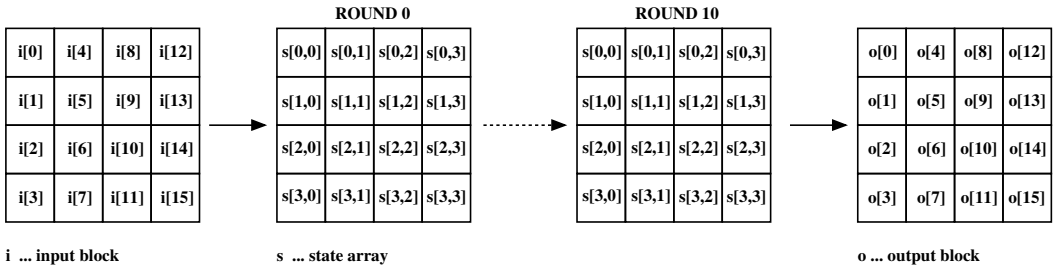
Na závěr se, stejně jako u všech ostatních položek, aplikuje operace XOR s položkou $w[i - N_k]$. Pokud má šifrovací klíč délku 256 bitů a pro položku na pozici i platí, že $i - 4$ je násobkem hodnoty N_k, pak se vytváří nový prvek pole aplikací operace Subword, po které následuje součet s položkou $w[i - N_k]$.

j	1	2	3	4	5	6	7	8	9	10
rcon[0]	01	02	04	08	10	20	40	80	1b	36
rcon[1]	00	00	00	00	00	00	00	00	00	00
rcon[2]	00	00	00	00	00	00	00	00	00	00
rcon[3]	00	00	00	00	00	00	00	00	00	00

Tabulka 2.10: Tabulka konstant Rcon

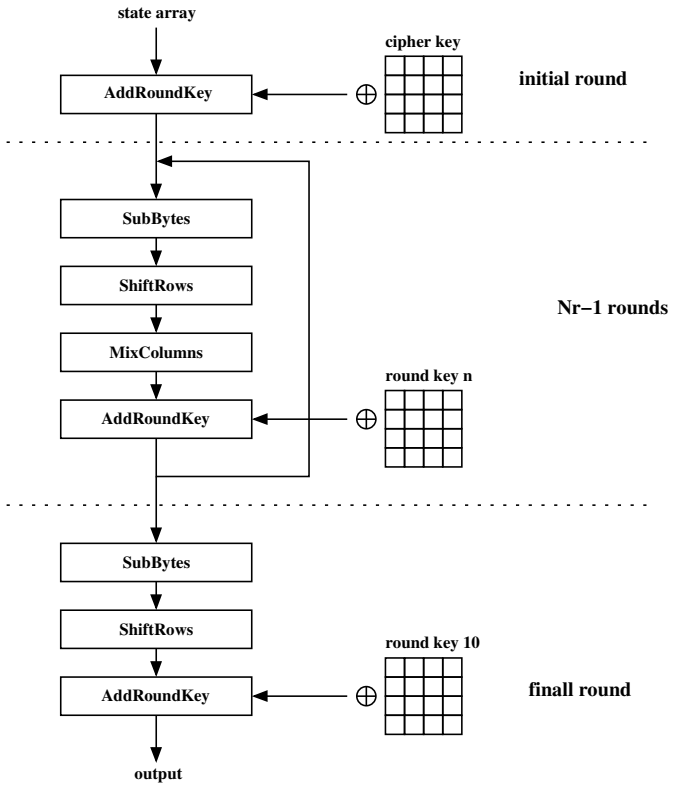
2.3.2 Šifrování

Vstupní i výstupní data jsou organizována do matice 4×4 byty. Výstupem každé iterace je mezivýsledek State, který lze také zapsat jako matici 4×4 byty. Na úvod šifrování se zkopíruje vstupní blok do matice mezivýsledku State (podle obrázku 2.8), který poté prochází N_r iteracemi.



Obrázek 2.8: Proces šifrování AES

Prvních $N_r - 1$ má stejný průběh, v poslední iteraci se provádí méně operací. Mezivýsledek poslední iterace se zkopíruje na výstup. Proces šifrování je znázorněn na obrázku 2.9.



Obrázek 2.9: Princip šifrování algoritmem AES

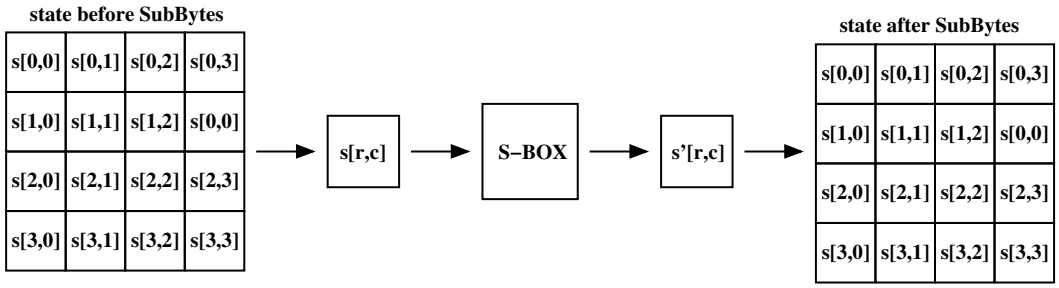
Jak lze vidět na obrázku 2.9, při šifrování pomocí aloritmu AES se na matici State aplikují následující transformace:

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

Po úvodním zkopírování vstupního bloku do matice State se provede zašumění dat šifrovacím klíčem transformací AddRoundKey, která je popsána později. Z obrázku 2.9 je patrné, že v prvních $N_r - 1$ iteracích jsou aplikovány všechny transformace a v poslední iteraci se již neuplatňuje transformace MixColumns. Dále budou popsány jednotlivé transformace šifrování.

Transformace SubBytes

Transformace SubBytes je nelineární operace substituce, použita nezávisle na všechny byty v aktuálním bloku mezivýsledku použitím substituční tabulky S-BOX. Na následujícím obrázku je znázorněno jakým způsobem se nahrazují postupně všechny byty v matici 4×4, kde hodnota bytu uložená v matici dává pozici v tabulce, na které je uložena nová hodnota příslušného bytu.



Obrázek 2.10: Transformace SubBytes

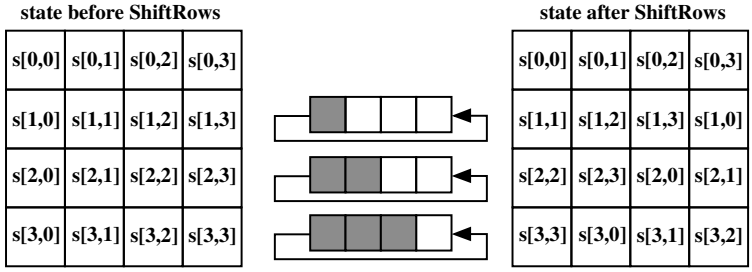
Nová hodnota příslušného bytu se získá z tabulky S-BOX tak, že první čtyři bity určí pozici řádku, na kterém se nachází substituční hodnota, a druhé čtyři bity určí sloupec této hodnoty. Tyto tabulky jsou dostupné v [21].

Transformace ShiftRows

Tato transformace pracuje s jednotlivými řádky mezivýsledku State. Poslední tři řádky State Array jsou cyklicky posouvány o různý počet bytů. První řádek se neposouvá. Transformaci lze vyjádřit následujícím vztahem:

$$S'_{r,c} = S_{r,[c+shift(r,N_b)]modN_b} \qquad pro \quad 0 < r < 4 \quad a \quad 0 \leq c < N_b \tag{2.5}$$

kde hodnota funkce $shift(r, N_b)$ závisí na posouvaném řádku r, například pro $N_b = 4$ jsou hodnoty této funkce následující: $shift(1, 4) = 1$, $shift(2, 4) = 1$ a $shift(3, 4) = 3$. Nejlépe tuto transfromaci ilustruje obrázek 2.14.



Obrázek 2.11: Transformace ShiftRows

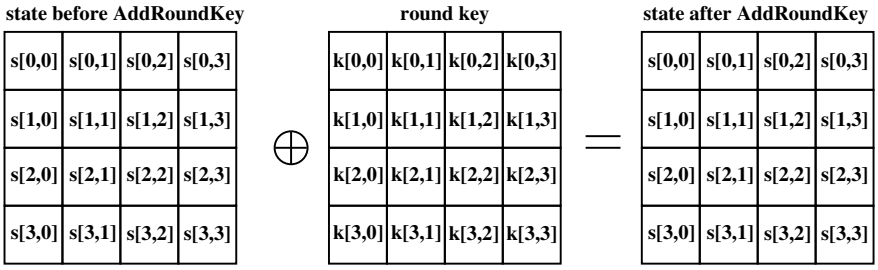
Transformace MixColumns

V této operaci se provádí úprava po sloupcích mezivýsledku. Úprava jednoho sloupce lze popsat jako maticové násobení:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{2.6}$$

Transformace AddRoundKey

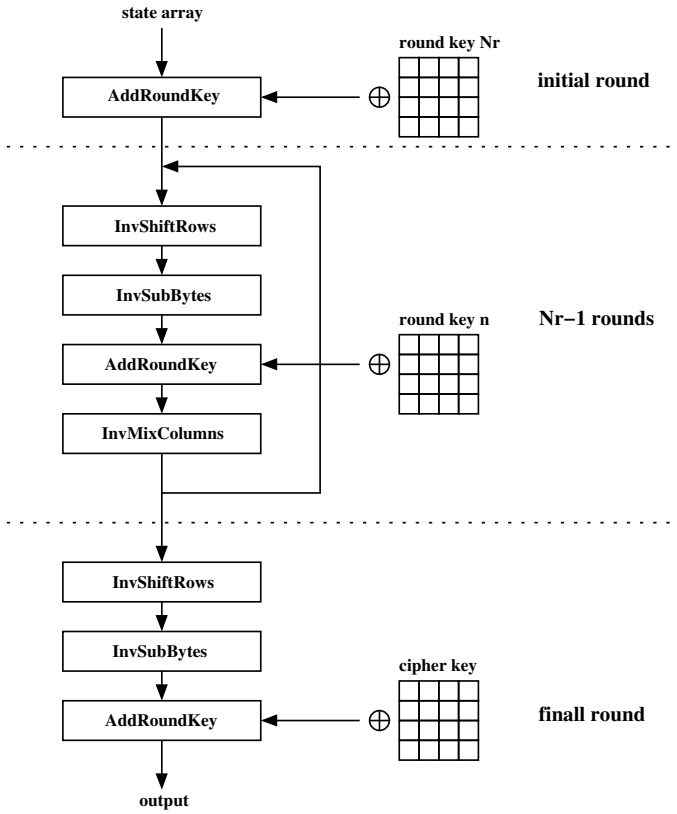
Při této operaci se přičítá iterační klíč k mezivýsledku za použití bitové operace XOR mezi jednostlivými bity vstupních bloků. Operace je znázorněna na následujícím obrázku.



Obrázek 2.12: Tranformace AddRoundKey

2.3.3 Dešifrování

V případě AES se jedná o symetrickou šifru, proto se při dešifrování používá stejný klíč jako při šifrování. Při dešifrování se používají inverzní transformace k transformacím použitým při šifrování. Proces dešifrování je znázorněn v následujícím obrázku:



Obrázek 2.13: Princip dešifrování algoritmem AES

Dešifrování se stejně jako šifrování provádí v N_r iteracích. Počet iterací závisí na velikosti šifrovacího klíče N_k. Prvních $N_r - 1$ iterací se provádí stejným způsobem, kde se aplikují všechny následně popsané transformace. Při poslední iteraci se neprovádí operace InvMixColumns. Při transformaci AddRoundKey se používá klíč v opačném pořadí než při šifrování. Dešifrování obsahuje následující inverzní transformace, přičemž transformace AddRoundKey je inverzní sama sobě:

- InvSubBytes
- InvShiftRows
- InvMixColumns
- AddRoundKey

Transformace InvSubBytes

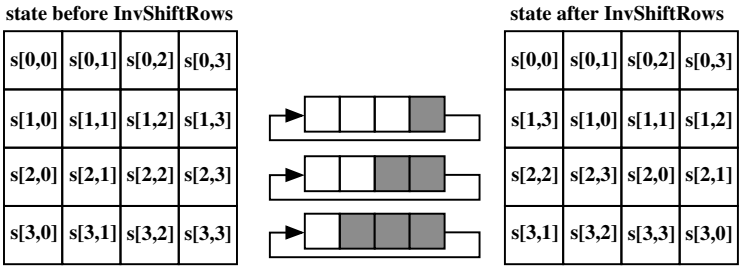
Pracuje stejným způsobem jako transformace SubBytes s tím rozdílem, že používá inverzní substituční tabulku S-BOX.

Transformace InvShiftRows

Byty v posledních třech řádcích mezivýsledku jsou cyklicky posouvány o rozdílný počet bytů. První řádek se neposouvá. Další tři řádky se cyklicky posouvají o $N_b - shift(r, N_b)$, kde hodnota funkce shift závisí na čísle řádku. Tuto transformaci lze popsat následujícím způsobem:

$$S'_{r,[c+shift(r,N_b)]modN_b} = S_{r,c} \qquad pro \quad 0 < r < 4 \quad a \quad 0 \leq c < N_b \tag{2.7}$$

a lze ji ilustrovat následujícím obrázkem:



Obrázek 2.14: Transformace InvShiftRows

Transformace InvMixColumns

Je to inverzní transformace k MixColumns. Mezivýsledek se upravuje po sloupcích a úpravu lze popsat jako násobení maticí.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \cdot \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \tag{2.8}$$

2.4 Existující implementace

2.4.1 Hadwarové realizace DES

Ačkoliv vlastnosti algoritmu DES jsou přímo určeny pro efektivní realizaci pomocí hardwaru, například programovatelných logických polí FPGA, lze nalézt i řešení pro jiné platformy. Existují softwarové implementace [16, 7, 17] a řešení určená pro VLSI [11, 30]. Nejvíce realizací je ale určených pro obvody FPGA [13, 31, 14, 18, 5].

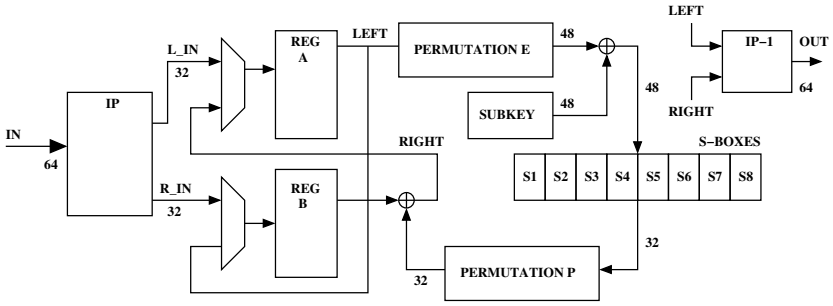
V tabulce 2.11 jsou porovnány vlastnosti některých realizovaných hardwarových implementací algoritmu DES, které byly popsány zpravidla v odborných článcích, založených na výzkumu univerzit po celém světě. Architektury popsáné v těchto publikacích poskytují propustnost v rozmezí od 26 Mb/s do 10752 Mb/s za použití různých metod návrhu.

Autoři	Použité zařízení	CLB slices	Maximální frekv.(MHz)	Propustnost (Mb/s)	Propusnost (Mb/s)/ CLB slices
Wong [31]	XC4020E	438	10	26.7	0.06
Kaps [14]	XC4020EX	741	25.18	402.7	0.54
Free-DES	XCV400	5263	47.7	3052	0.57
McLoony [18]	XCV1000	6446	59.5	3808	0.59
Pierson [30]	ASIC	-	-	9280	-
Patterson[5]	XCV150	1584	168	10752	6.78
Saqib [22]	XCV400e	117	68.05	274	2.34

Tabulka 2.11: Porovnání realizovaných hardwarových řešení algoritmu DES

Implementace šifrovacího jádra Free-DES používá metodu zřetězeného zpracování v ECB režimu a její maximální propustnost je 3052 Mb/s. Implementace určená navržená na platformě Jbits [5] poskytuje nejvyšší rychlost šifrování a to až 10752 Mb/s. V této realizaci je použito plně zřetězené zpracování iterací. Architektura navržená v [14] implementuje dvě zřetězená řešení - první zřetězené řešení obsahuje 2 stupně zpracování, ve druhém řešení jsou použity stupně 4. První řešení dosahuje propustnosti až 183.3 Mb/s, druhé 402.7.

Většina implementací DES určených pro FPGA používá částečné nebo plné zřetězení. Výjimkou je návrh v [31], která realizuje DES pomocí iterační metody bez zřetězení. Tato implemetace zabírá v FPGA 438 slice a jeden blok zpracuje za 24 taktů, maximální propustnost tohoto systému je 26 Mb/s. Další architekturou, ve které se neuplatňuje zřetězení [22] využívá iterační metodu zpracování celého vstupního bloku o velikosti 64 bitů. Toto řešení poskytuje při spotřebě 165 slice propustnost 274 Mb/s. Blokové schéma tohoto řešení je zobrazno na obr. 2.15.



Obrázek 2.15: Realizace DES [22]

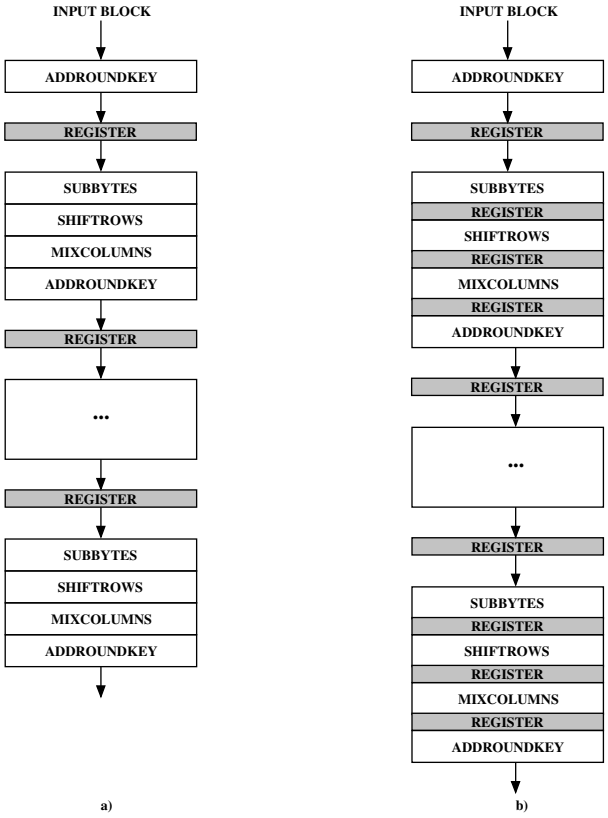
2.4.2 Hardwarové realizace AES

I když je AES jeden z výpočetně nejefektivnějších šifrovacích algoritmů, je stále poměrné naročný na výpočet a není schopný zajistit propustnost požadovanou některými aplikacemi v případě softwarové implementace. Motivací pro návrh rozličných hardwarových akcelerací pro algortimus AES je buď velmi vyskoká propustnost [26, 2, 19] nebo snaha o návrh pro zařízení vestavěných systému, které disponují omezeným množstvím hardwarových zdrojů [9, 24, 23].

Tabulka 2.12 porovnává vlasnosti existujících hardwarových implementací s propustností vyšší než 2Gb/s, jejichž propusnost se nachází v rozmezí od 2,1 Gb/s do 34,76 Gb/s. Návrhy architektur s velmi vysokou propustností (nad 20 Gb/s) [2] jsou zpravidla založeny na technice rozvinutí smyčky s vnějším zřetězením iterací (obrázek 2.16a). Nejvyšších propustností (až 32 Gb/s) lze dosáhnout kombinací vnějšího a vnitřního zřetězení iterací (obrázek 2.16b) [26]. Takové realizace jsou ovšem velmi náročné na velikost plochy na čipu (viz tabulka 2.12).

V implementaci [26] je realizace nejdůležitějších operací založena na mapování jejich předpočítaných výsledků do blokových pamětí RAM. Těmito operacemi jsou SubBytes a Inv/MixColumns. To ovšem vede při technice rozbalení smyček k potřebě velkého množství BlockRAM. Jak lze vidět v tabulce 2.12 je potřeba 80 blokových pamětí pro tuto implementaci. Dále byly pro tuto implementaci vytvořeny dvě varianty. V jedné je použito pouze rozbalení smyček s vnějším zřetězením iterací (obr. 2.16a) a tato realizace dosahuje propustnosti 19.95 Gb/s, při taktování obvodu na frekvenci 156 MHz. Po přidání vnitřního zřetězení iterací (obr. 2.16b) lze zvýšit frekvenci na 272 MHz a docílit propustnosti až 34,76 Gb/s. Přidání vnitřního zřetězení iterací s sebou nese režii v podobě potřebných oddělovacích registrů, nárust potřebných hardwarových zdrojů pro tyto registry není zdaleka tak vysoký, jako zvýšení propustnosti, kterou tímto způsobem dosáhneme.

Další realizací obvodu s vysokou propustností je [2]. V této implementaci se opět dosahuje vysoké propustnosti pomocí techniky rozbalení smyček s kobminací vnějšího a vnitřního zřetězením (obr. 2.16a). Jednotlivé operace jsou ale realizovány pomocí kombinační logiky, přičemž zřetězení se zde uplatňuje i v rámci operací (především SubBytes).



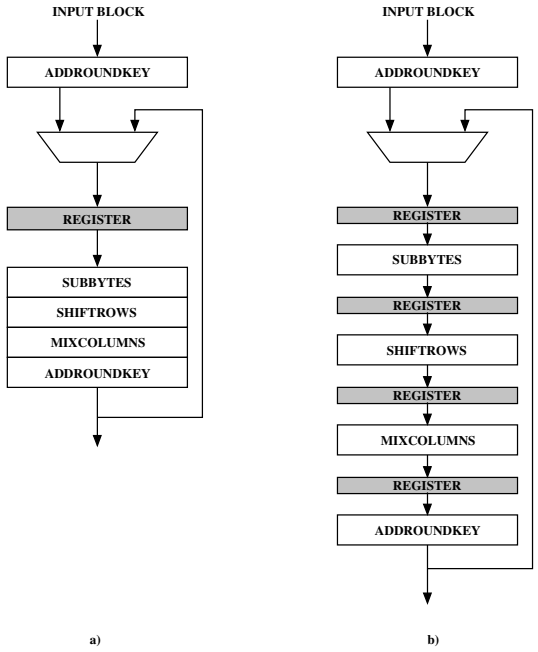
Obrázek 2.16: Způsoby realizace: a) vnější zřetězení iterací b) kombinace vnějšího a vnitřního zřetězení

Propustností okolo 2 Gb/s lze dosáhnout pomocí iteračních metod v kombinaci s vnitřním zřetězením iterací (obr. 2.17b) [19]. Tímto způsobem lze dosáhnout velmi výrazné úspory potřebných hardwarových zdrojů, jako tomu je u třetí implementace pomocí iterační metody v [26]. Tento fakt je patrný z tabulky 2.12. Jednotlivé operace jsou zde zpravidla realizovány stejnými technikami, jako je tomu u architektur s velmi vysokou propustností, kde jsou jednotlivé transformace aplikovány na celý vstupní blok o velikosti 128 bitů.

Autoři	Chaves1 [26]	Chaves2 [26]	Chaves3 [26]	Hodjat [2]	Nedjah [19]
Zařízení	XC2VP20	XC2VP20	XC2VP20	VirtexII-Pro	VIRTEX-II
CLB slices	3513	3168	515	5177	1937
BRAM	80	80	12	84	-
Max. freq (MHz)	272	156	156	168.35	190
Latence (takty)	30	10	10	31	33
Propustnost(Gb/s)	34.76	19.95	2.33	21.54	2.1
Propustnost(Mb/s)/CLB	9.9	6.3	4.5	4.1	10.8

Tabulka 2.12: Porovnání realizovaných hardwarových řešení algoritmu AES s propustností vyšší než 1 Gb/s

V praxi lze nalézt jenom několik aplikací, které potřebují propustnost 20 Gb/s, zatímco flexibilní, hardwarově méně náročné řešení s nízkým příkonem je výhodné pro velké množství aplikací například ve vestavěných systémech. Mnoho aplikací (wireless komunikace, digital cinema, pay TV) požaduje malou propustnost do 100 Mb/s a často i nižší. Při snaze o návrh kompaktních a efektivních řešení se používá výhradně iterační metoda [9, 24], často v kombinaci s vnitřním zřetězením iterace (obr. 2.17b) [24]. Výpočet iteračních klíčů pro jednotlivé iterace se provádí před procesem šifrování nebo dešifrování.



Obrázek 2.17: Způsoby realizace: a) obecná architektura b) vnitřní zřetězení iterací

V tabulce 4.2 je zobrazeno srovnání nejdůležitějších vlastností implementací s propustností do 2 Gb/s. První studovaná implementace [9] využívá pouze iterační metodu bez zřetězeného zpracování, nejdůležitější transformace SubBytes a MixColumns jsou obě realizovány pomocí dualportových blokových pamětí RAM a architektura zpracovává najed-

nou bloky o velikosti 32 bitů z celkových 128 bitů velikosti šifrovaného bloku, což vede k další velmi výraznému úspoře zdrojů. Tato realizace dosahuje propustnosti 208 Mb/s v technologii XC3S50-4 a 358 Mb/s v technologii XC2V40-6.

Druhé řešení [24] používá iterační metodu s vnitřním zřetězením iterací. Tato implementace realizuje transfromaci SubBytes pomocí BlockRAM a transformaci MixColumns pomocí kombinační logiky. I zde je vytvořena logika pro zpracování bloků o velikosti 32 bitů. Tímto řešením lze dosáhnout propustnosti 139 Mb/s.

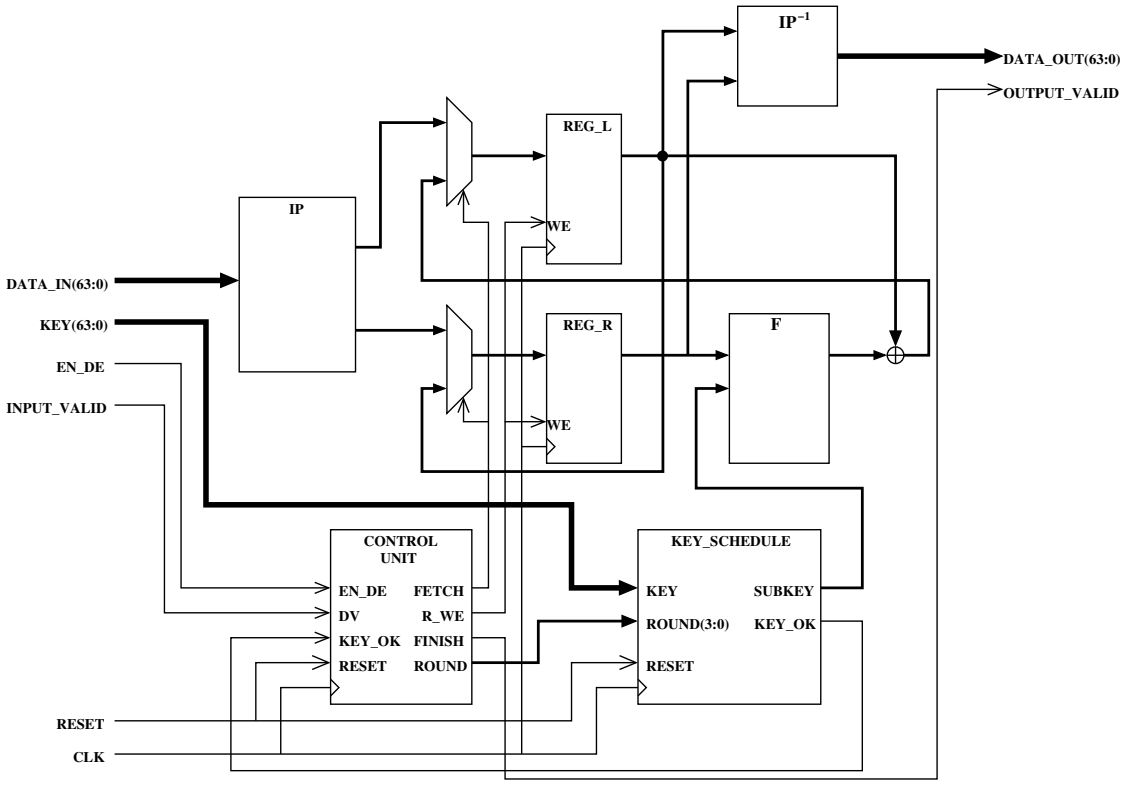
Autoři	Rouvroy [9]	Rouvroy [9]	Chodowiec [24]	Alho [23]
Zařízení	XC3S50-4	XC2V40-6	XC2S30-5	-
CLB slices	163	146	222	n/a
BRAM	3	3	3	
Max. freq (MHz)	71.5	123	50	130
Doba výpočtu (takty)	46	46	160	160
Propustnost (Mb/s)	208	358	139	104
Propustnost (Mb/s)/CLB	9.9	6.3	4.5	n/a

Tabulka 2.13: Porovnání realizovaných hardwarových řešení algoritmu AES s propustností nižší než 1 Gb/s

Kapitola 3

Architektura DES

Při návrhu této architektury jsme se zaměřili na možnost jejího umístění do aplikací vestavěných systémů. Ty často obsahují obvody FPGA s poměrně malým množstvím dostupných hardwarových zdrojů, proto byla architektura navrhována se snahou o minimalizaci velikosti výsledného řešení. Naším cílem byla možnost umístit výslednou architekturu do obvodu FPGA typu Spartan 3 XC3S50-4PQ208C (umístěném na platformě FITKit) a dosáhnout minimální propustnosti 100 Mb/s.



Obrázek 3.1: Blokové schéma architektury DES

Návrh byl proveden na základě poznatků z exitujících realizací (kapitola 2.4.1), které svojí velikostí teoreticky umožňují umístění do námi zvoleného obvodu Spartan [5, 18, 22]. Z těchto poznatků vyplývá, že nejvhodnějším řešením s ohledem na velikost obvodu a požadovanou rychlost je iterační metoda [22], která vede k minimalizaci hardwarových zdrojů výsledné implemetace. Tato metoda obsahuje logiku, pro zpracovaní jedné iterace šifrování, přičemž výpočet se musí opakovat podle definovaného počtu opakování. V případě algoritmu DES se jedná o 16 iterací.

Vstupy a výstupy architektury DES

data_in(63:0) – 64-bitový vstup reprezentující data, který mají být šifrovaná nebo dešifrovaná. Data jsou načítána do vnitřního registru při náběžné hraně hodinového signálu CLK a řídícího signálu INPUT_VALID nasteveného do úrovně logická 1.

key(63:0) – vstup o šířce 64 bitů reprezentuje šifrovací klíč pro šifrování a dešifrování algoritmem DES. Klíč na vstupu musí být stabilní po dobu běhu procesu Key expansion.

data_out(63:0) – 64-bitový výstup reprezentující výsledek procesu šifrování nebo dešifrování. Data mohou být z výstupu čtena při nástupné hraně hodinového signálu CLK a platnost těchto dat na výstupu je indikována kontrolním signálem OUTPUT_VALID, který je po dobu platnosti dat na výstupu nastaven do aktivní úrovně.

input_valid – tento signál synchronizovaný hodinovým signálem CLK spouští načítání vstupních dat do vnitřního registru a také šifrování nebo dešifrování samotné.

output_valid – tento výstupní signál je v aktivní úrovni, pokud je výsledek šifrování vystaven na výstupu data_out.

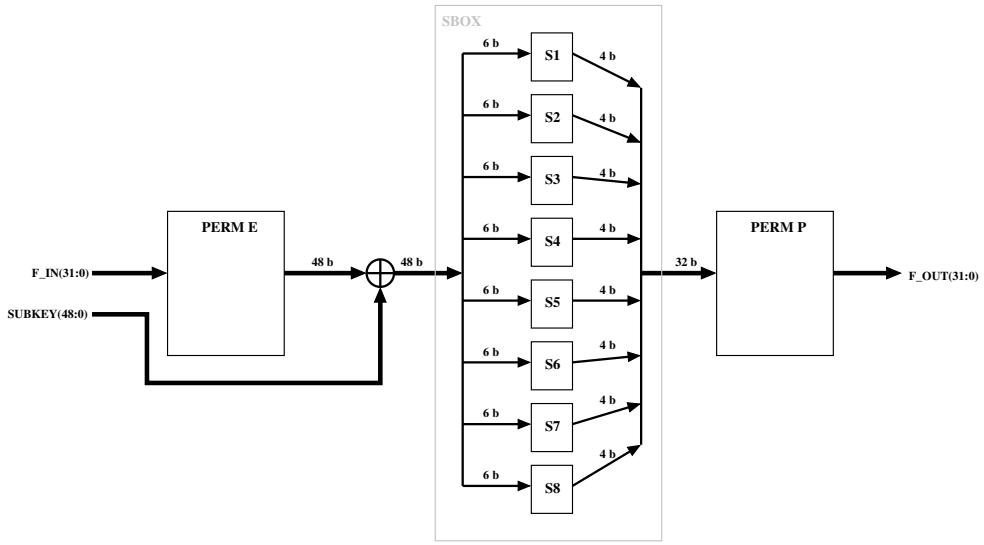
Algoritmus DES je založen především na dvou operacích: fixní permutace a substituce [22]. Obě tyto operace lze velmi efektivně implementovat v obvodech FPGA. Fixní permutace ve skutečnosti nezabírá téměř řádné zdroje v obvodu FPGA, lze ji implementovat pouze záměnou vodičů. Substituce je v algoritmu DES reprezentována osmi S-Box entitami (každá o velikosti 64 x 4 bity), celková velikost paměti pro substituci je 2Kb, což je relativně malé množství paměti, které lze implementovat za použití distribuované paměti v FPGA. Tyto vlastnosti lze využít při efektivní HW realizaci algoritmu DES. Fixní permutace je základem operací IP, IP^{-1}, permutace E, permutace P a operací uplatňujících se v procesu Key expansion, permutace PC a permutace PC^{-1}. Substituce je základem šifrovací funkce F, ve které se nachází osm substitučních tabulek.

Na obrázku 3.1 je zobrazeno blokové schéma navrženého zapojení architektury pro šifrování a dešifrování pomocí algoritmu DES. Na vstupní 64-bitový blok dat je aplikována permutační operace IP. Výsledek této operace je přes multiplexory přiveden na vstupy vnitřních 32-bitových registrů REG_L a REG_R, ve kterých se ukládají výsledky jednotlivých iterací. Vstup do těchto registrů je potřeba multiplexovat, protože v prvním taktu po spuštění šifrování se načítají data z výstupu permutace IP a v dalších taktech je potřeba na vstup přivést výsledky výpočtu jednotlivých kol. Obvod je navržen tak, aby v každém hodinovém taktu bylo zpracována jedna iterace z celkových 16, potřebných pro celkové zašifrování dat.

Výstup registru REG_L, ve kterém je uloženo spodních 32 bitů mezivýsledků, je přes multiplexor přímo přiveden na vstup registru REG_R, ve kterém se ukládá 32-bitová horní polovina mezivýsledků. Výstup registru REG_R je přiveden na vstup kombinačního obvodu, jehož výstupem je hodnota šifrovací funkce F, pro vstupní blok o velikost 32 bitů. Nad výstupem šifrovací funkce a výstupem registru REG_L je provedena logická operace XOR, pro jednotlivé bity těchto 32-bitových bloků. Výsledek této operace je přes multiplexor přiveden na vstup registru REG_L.

Návrh šifrovací funkce F

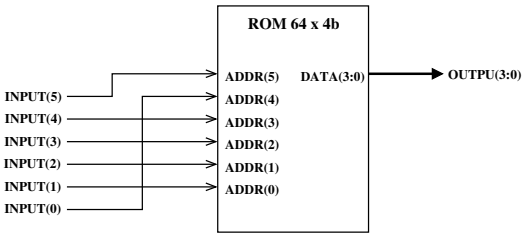
Tato funkce je základem algoritmu DES a její realizace je také nejnáročnější. Kromě použití permutačních sítí jsou zde umístěny substituce, a to ve formě pevných substitučních tabulek nazývaných S-BOXy [8]. V architektuře je funkce popsána komponentou, která obsahuje kombinační logiku realizující feistelovu funkci F. Na obrázku 3.2 je zobrazeno její obvodové zapojení.



Obrázek 3.2: Šifrovací funkce F

V této variantě feistelovy funkce je vstupní blok o velikosti 32 bitů přiveden na vstup permutace E (pracuje podle tabulky 2.3). Tato operace rozšíří vstup široký 32 bitů na výstup o velikost 48 bitů. Toto rozšíření je dáno dvojnásobným použitím některých vstupních bitů v permutaci. Tento 48bitový blok je rozdělen na 8 částí po 6 bitech, kde každá část je přivedena na vstup jednoho z 8 obvodů realizujících substituci. Tyto obvody jsou realizovány asynchronními pamětmi ROM o velikosti 64x4 b (obr. 3.3). Vstupní bity slouží jako adresa této paměti, výstupem jsou data uložená na této adrese.

Výstupem každé této substituce jsou 4 bity, které dají dohromady 32-bitový vstupní blok do jednotky, která provádí permutaci P (tabulka 2.5). Výstup této operace je výsledek funkce F.

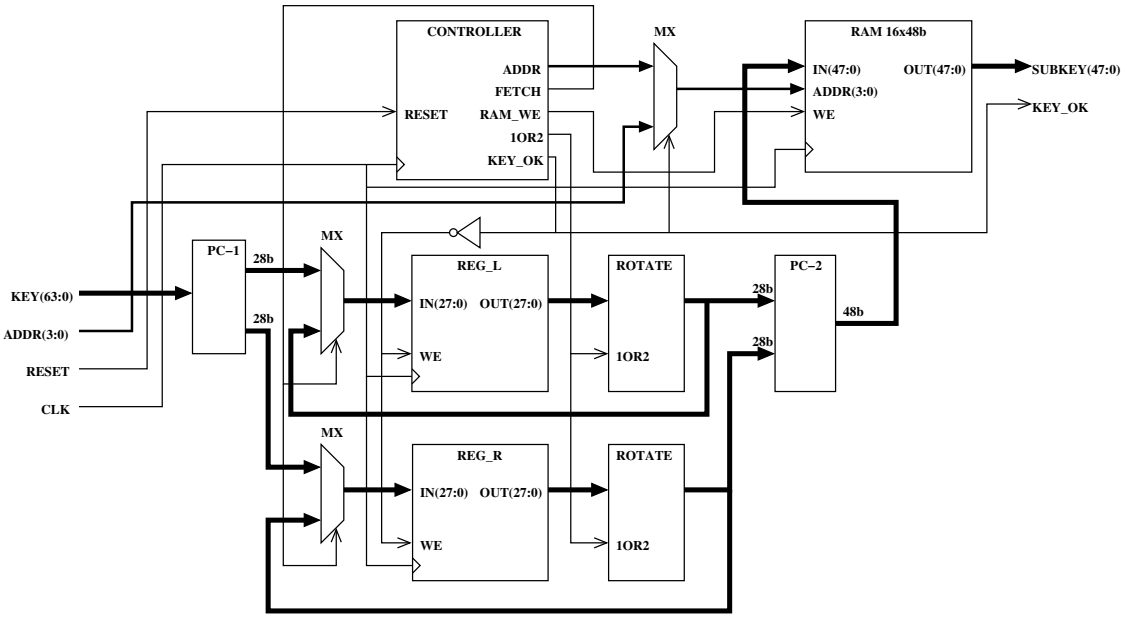


Obrázek 3.3: Substituce Sx, realizovaná pamětí ROM

Návrh obvodu pro výpočet iteračních klíčů (Key schedule)

Pro proces šifrování je nutné zajistit výpočet iteračních klíčů na základě vstupního 64-bitového šifrovacího klíče. Jsou dvě možnosti jak tento výpočet realizovat. První možností je výpočet iteračních klíčů v rámci každé jedné iterace. Druhou možností je provést výpočet před samotným šifrováním a výsledky uložit do paměti RAM.

Pro tento návrh byla použitá druhá varianta, protože navrhovaný obvod má umožňovat i dešifrování, kde jsou použity iterační klíče v opačném pořadí a provést výpočet klíčů zároveň s dešifrováním nelze. První varianta je lepším řešením pro systémy, které jsou určeny pouze pro šifrování.



Obrázek 3.4: Blokové schéma Key schedule

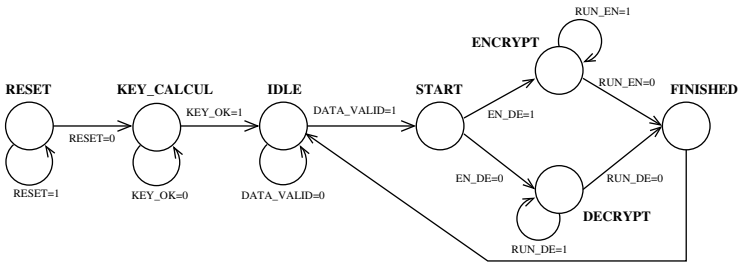
Blok Key schedule (obr. 3.4) pracuje se vstupním 64-bitovým klíčem. Na základě tohoto klíče vypočítá 16 podklíčů, kde každý je určený právě pro jednu iteraci. Podklíč n se počítá na základě podlíče $n-1$. I v případě návrhu Key schedule byla použita iterační metoda.

Vstupní data o velikosti 64 bitů vstupují do permutačního bloku PC-1. V tomto bloku je provedena permutace s 56 bity vstupu, které jsou rozděleny na poloviny po 28 bitech. Zbývajících osm bitů je paritních a slouží pro ověření integrity vstupního šifrovacího klíče. Pro ukládání jednotlivých mezivýsledků jsou použity dva 28-bitové registry. Nad výstupy obou registrů jsou provedeny levé rotace o dva nebo jeden bit, počet posouvaných bitů je určen signálem $1OR2$, který je výstupem řídící jednotky. Výsledky obou rotací jsou vstupem do permutačního bloku $PC-2$ a zároveň vstupem do registrů, ve kterých jsou ukládány jednotlivé výsledky iterací. Výstup permutace $PC-2$ je ukládám do paměti RAM, určené pro uložení všech iteračních klíčů. Klíče jsou uložené do paměti RAM stejným způsobem při šifrování i dešifrování.

3.1 Šifrování a dešifrování

Šifrování i dešifrování pomocí algoritmu DES se provádí stejným způsobem, jediný rozdíl je v pořadí použití iteračních klíčů. V případě dešifrování se uplatňují v opačném případě, než tomu je při šifrování. S touto skutečností pracuje řídící jednotka, která přivádí na vstup bloku Key schedule adresu klíče jednotlivé iterace.

Procesy šifrování a dešifrování jsou prováděny podle stavového diagramu na obrázku 3.5. Z něho je patrné, že po resetu systému se automaticky spustí výpočet rundovních klíčů. Po výpočtu se čeká na vstupní data, při platnosti vstupních dat se přechází do stavu šifrování nebo dešifrování, podle vstupního signálu EN_DE, který tuto akci definuje. Po dokončení procesu šifrování nebo dešifrování se přechází do stavu, ve kterém je výsledek šifrování po dobu jednoho taktu na výstupu obvodu. Poté se opět přechází do stavu, ve kterém se čeká na vstupní data.



Obrázek 3.5: Stavový diagram šifrování a dešifrování DES

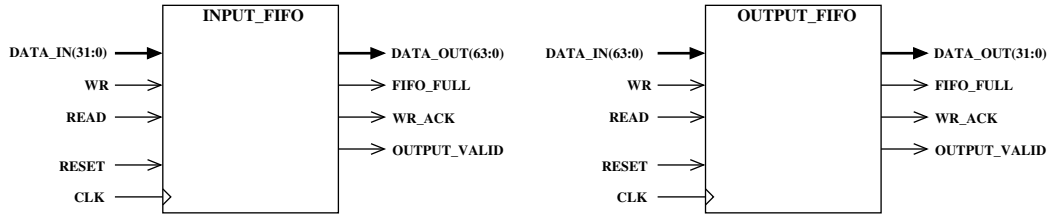
Výsledná rychlost šifrování je odvozena od délky zpracování jednoho bloku. Počáteční zpoždění obvodu je způsobeno výpočtem iteračních klíčů. Jednotka pro jejich výpočet je navržena takovým způsobem, aby byla schopna za jeden takt hodinového cyklu vypočítat a uložit klíč právě pro jednu iteraci. Výpočet všech 16 klíčů zabere 17 hodinových cyklů, v jednom taktu se načítá klíč. V případě délky zpracování vstupního bloku určeného pro šifrování a dešifrování, je obvod také navržen pro zpracování jedné iterace za jeden hodinový takt. Pokud přičteme jeden takt potřebný pro nastartování celého výpočtu a jeden takt pro zápis výsledku na výstup, dostaneme, že obvod zpracuje 64-bitový vstupní blok za 18 taktů. Předpoklady návrhu budou ověřeny při simulaci obvodu.

3.2 Režim blokového šifrování DES

Režimem blokového šifrování se rozumí způsob, jakým budeme postupovat při šifrování zprávy, která je delší než velikost bloku šifrovaného algoritmem. Nejpoužívanějšími režimy jsou [12, 1]:

- **ECB (Electronic Code Book)** – režim ECB pracuje tak, že srozumitelný otevřený text P (Plaintext) je rozdělen na bloky Pi o délce odpovídající délce bloku blokového šifrovače a každý blok je samostatně zašifrován blokovým šifrovačem E aplikací stále stejného klíče K. Výsledné bloky se opět spojí do jedné zprávy – šifry C (Ciphertext). Dešifrování se provádí opačným způsobem.
- **CBC (Cipher Block Chaining)** – podobně jako u režimu ECB i režim CBC pracuje tak, že otevřený text P je rozdělen na bloky o délce odpovídající délce bloku blokového šifrovače a každý blok je blokovým šifrovačem zašifrován samostatně. Na rozdíl od režimu ECB je však pro každý blok před jeho zašifrováním provedena operace XOR (non-ekvivalence) s předchozím zašifrovaným blokem.
- **CFB (Ciphertext FeedBack)** – v tomto režimu se šifrovaný text vůbec nestává blokovou šifrou, bloková šifra slouží jako generátor pseudonáhodné posloupnosti, která je pak použita pro zašifrování otevřeného textu (zprávy) operací XOR.

Pro realizaci architektury DES byl vybrán režim ECB, kvůli jeho snadné implementaci. Pro možnost připojení systémů s rozdílnou rychlostí byly připojeny na vstup a výstup architektury vyrovnávací paměti cache, realizované pomocí struktur FIFO (obr. 3.6). Na vstup je připojena komponenta s názvem INPUT_FIFO a na výstup komponenta OUTPUT_FIFO.

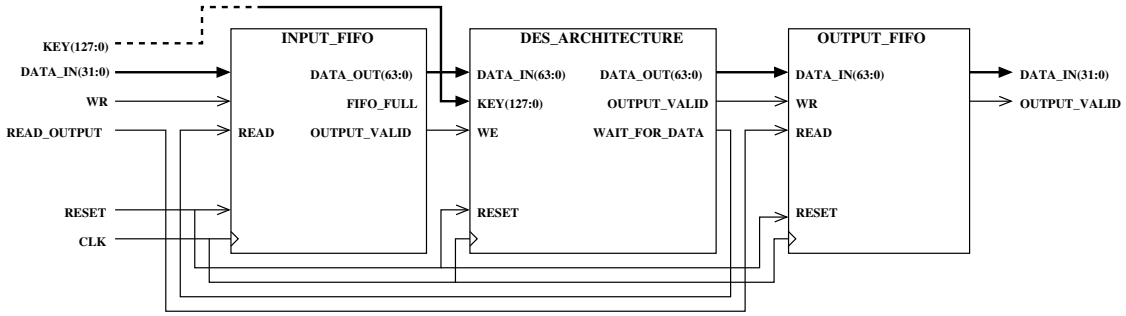


Obrázek 3.6: Jednotky FIFO, určené pro DES [32]

Vstupní data jednotky INPUT_FIFO mají velikost 32 bitů, výstupní datový blok je 64-bitový. Pro načtení jednoho výstupního bloku INPUT_FIFO je tedy nutné 2x zápis vstupních dat. Vstupem jednotky OUTPUT_FIFO je blok o velikosti 64 bitů, výstupem je blok velký 32 bitů. Pro přečtení jednoho výstupního bloku jsou potřeba dvě čtení. Všechny činnosti paměti FIFO jsou synchronizovány hodinovým signálem CLK. Jednotka je asynchronně resetována signálem RESET. Při resetu se vyprázdní obsah paměti FIFO. Je-li aktivován signál WR, zapíše se datový signál DATA_IN do paměti na adresu určenou vnitřním ukazatelem pro zápis, pokud paměť není plná. Signál WR_ACK pak svou aktivní úrovní signalizuje, že byl zápis uskutečněn. Je-li aktivní signál READ, bude v následujícím taktu hodin na výstupu (signál DATA_OUT) obsah paměťového místa určeného vnitřním ukazatelem čtení, který je platný pokud paměť není prázdná, a signál OUTPUT_VALID indikuje platnost dat na výstupu. Při čtení se inkrementuje obsah ukazatele vnitřního ukazatel čtení,

při zápisu se inkrementuje obsah ukazatele vnitřního ukazatele zápisu. Současně se aktualizuje obsah registru představujícího počet slov zapsaných v paměti. Stav plné paměti se indikuje signálem FIFO_FULL.

Celá architektura bude mít po přidání vstupní a výstupní jednotky FIFO strukturu zobrazenou na obrázku 3.7. Šifrovací klíč je možné je realizovat jako vstup celého obvodu, nebo jako pevně zadanou hodnotu uvnitř architektury.



Obrázek 3.7: Struktura architektury DES s přidanými jednotkami FIFO

3.3 Simulace a syntéza architektury DES

Po fázi návrhu byla provedena implementace architektury v jazyce VHDL (za použití literatury [28, 6]). Funkčnost implementace byla otestována pomocí dostupných simulačních nástrojů. Pomocí simulace byl ověřen navržený počet taktů potřebných pro zašifrování nebo dešifrování jednoho bloku dat o velikosti 64 bitů. Jeden blok dat je zpracován za 18 cyklů a z této informace lze vyjádřit maximální teoretickou propustnost architektury (rovnice 4.14), kterou lze vypočítat při znalosti délky zpracování vstupního bloku v taktech a maximální možné frekvence obvodu.

$$P_{des} = \frac{block\_size}{cycles} \cdot f_{des} = \frac{64}{18} \cdot f_{des} \tag{3.1}$$

Proces rozšíření klíče je spuštěn hned po resetu systému a trvá 17 taktů, což je doba po které lze (de)šifrovat první data.

3.3.1 Dosažené výsledky

Po simulaci obvodu byla provedena jeho syntéza. Ta byla vykonána pro FPGA typu Spartan 3 a jejím výstupem jsou informace o velikosti plochy zabrané navrženou architekturou na tomto čipu (tabulka 3.1).

	Rozdělení	Počet zdrojů	Dostupné zdroje	Obsazeno (%)
Slices	-	251	768	32
Slices flip flops	-	138	1536	8
4 LUTS		482	1536	31
	logic	434	-	-
	RAM	48	-	-

Tabulka 3.1: Obsazené zdroje architektury DES

Velikost jednotlivých částí architektury DES je uvedena v tabulce 3.2. Z této tabulky je patrné, že nejméně náročné operace jsou bitové permutace, které lze velmi efektivně realizovat pomocí hardware pouhou záměnou vodičů. Registry pro uložení 32-bitových výsledků jednotlivých iterací jsou realizovány 32 klopnými obvody, realizace dvouvstupových multiplexorů pro řízení vstupních dat do registrů je také v hardwarovém řešení velmi nenáročné. Nejvíce zdrojů je obsazeno blokem pro výpočet iteračních klíčů ze vstupního 56-bitového šifrovacího klíče. Při použití varianty s předpočítanými iteračními klíči v paměti ROM by úspora plochy obsazené architekturou DES byla téměř 45 %.

	Slices	Slices flip flops	4 LUTS
Init perm	-	-	-
2 x Register 32	-	64	-
2 x Mux 32	36	-	8
Function F	76	-	176
Key schedule	118	64	227
Controller	21	10	41
Finall perm	-	-	-
Ostatní	-	-	30

Tabulka 3.2: Obsazené zdroje jednotlivých částí architektury DES

Výstupem procesu syntézy je dále informace o teoretické maximální frekvenci, na které může daný obvod pracovat. Tato maximální frekvence obvodu pro šifrování DES je 104 MHz. S touto informací již lze dopočítat maximální propustnost P_{des} architektury:

$$P_{des} = \frac{block\_size}{cycles} \cdot f_{des} = \frac{64}{18} \cdot 104 = 369,78 Mb/s \tag{3.2}$$

3.3.2 Srovnání se softwarovým řešením

Srovnání bylo provedeno se softwarovou implementací [15], která byla spuštěn na mikroprocesoru MSP430F1611, umístěném na FITKitu. Při testu bylo zašifrováno 1500 bloků o velikosti 64 bitů. Tento výpočet trval 49,8 s. Z těchto hodnot lze vypočítat rychlost šifrování:

$$P_{des\_sw} = \frac{1500 \cdot 64}{49,8} = 1927b/s \tag{3.3}$$

Hardwarová realizace je podle teoretické rychlosti $201 \cdot 10^3$krát rychlejší než softwarová implementace, spuštěná na mikroprocesoru určeného pro vestavěné systémy.

Pro srovnání byla tato implementace [15] spuštěna na stolním počítači s dvoujádrovým procesorem Intel Core Duo, pracujícím na frekvenci 1,6 GHz. Na tomto procesoru bylo $5 \cdot 10^6$ bloků zašifrováno za 29,3 s, rychlost šifrování tedy byla $1,1\ Mb/s$. Na tom samém procesoru byla spuštěna optimalizovaná implementace [10], kterou na mikroprocesoru MSP430 nebylo možné spustit. Tato implementace dokázala zašifrovat $30 \cdot 10^6$ bloků za 14 s, čímž dosáhla rychlost šifrování 130 Mb/s. Optimalizovaná implementace byla více než 100krát rychlejší. Je ovšem jasné, že pokud by se nám podobným způsobem podařilo optimalizovat softwarové řešení, určené pro vestavěné systémy, bylo by zrychlení hardwarové akcelerace stále v řádu tisíců.

Z těchto výsledku je evidentní, že použití hardwarových implementací pro realizaci šifrování ve vestavěných systémech je velmi výhodné a v některých případech přímo nezbytné, protože rychlost softwarových řešení v řádech kb/s je v mnoha případech nedostačující.

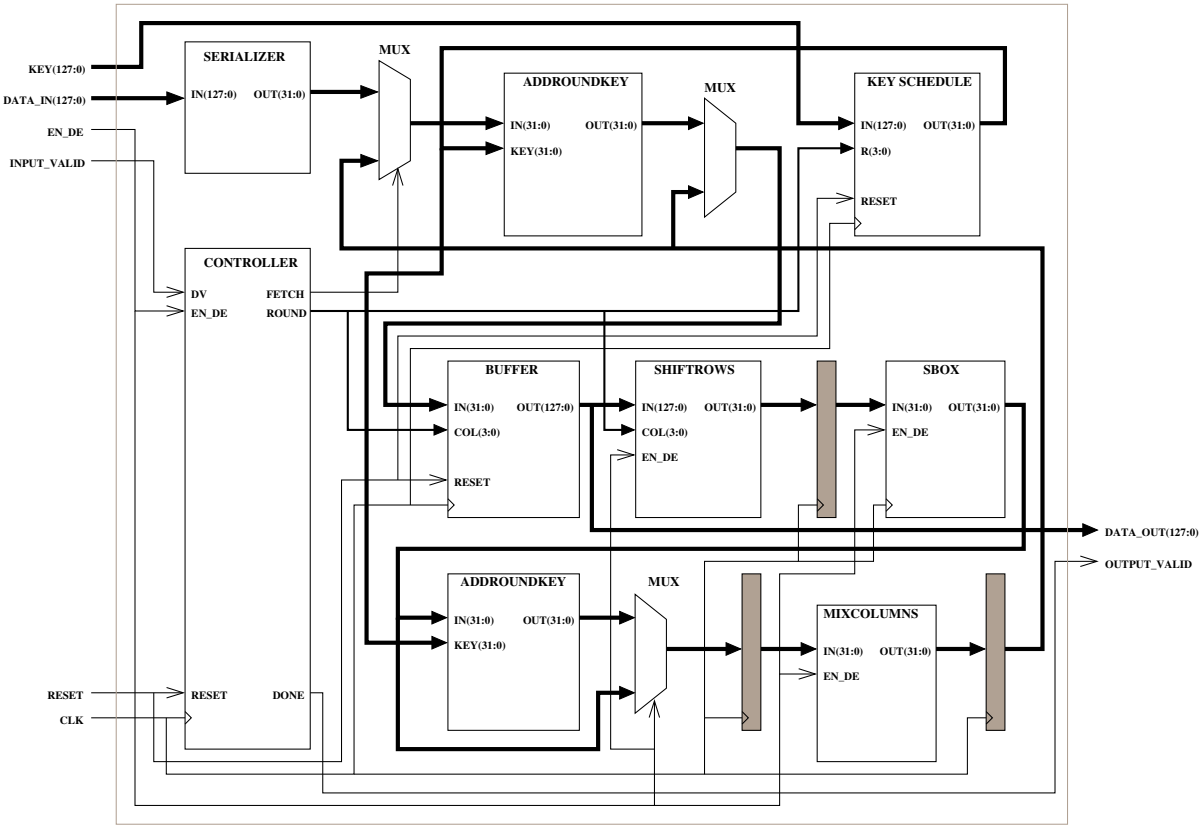
3.3.3 Použité nástroje

Simulace byla provedena v programu Xilinx ModelSim XE III 6.3c, která slouží pro simulaci komponent napsaných v jazyce VHDL. Pro synstézu byl použit nástroj ISE WebPACK 9.1i. Oba programy jsou k dispozici zdarma pro nekomerční využití a jsou dostupné na stránce http://www.xilinx.com/webpack/index.html.

Kapitola 4

Architektura AES

I v případě architektury AES bylo našim cílem navrhnout architekturu vhodnou pro aplikaci ve vestavěných systémech. Z toho vyplývá požadavek na nutnost minimalizace plochy zabrané v obvodu FPGA výslednou implementací. Architektura byla navržena pro možnost jejího umístění na FPGA typu Spartan 3 se snahou dosáhnout minimální propustnosti 100 Mb/s. Při návrhu se opět vycházelo z informací, obsažených v popisech existujících architektur, se zaměřením na implementace s propustností nižší než 1 Gb/s.



Obrázek 4.1: Blokové schéma architektury AES

Algoritmus AES podporuje délku klíče 128, 192 nebo 256 bitů. Pro návrh a implementaci byla vybrána nejvíce používaná varianta s délkou šifrovacího klíče 128 bitů, která se

označuje AES-128. Pro blokové šifrování je nejefektivnější technikou vnější zřetězení iterací (obr. 2.16), především v případě, že šifrování neobsahuje velké množství iterací. Při této technice ovšem architektura musí obsahovat logiku pro zpracování všech iterací a z tohoto důvodu není toto řešení příliš úsporné (viz tabulka 2.17).

Jelikož je naším cílem návrh systému pro vestavěné systémy, ve kterých není k dipozici velké množství hardwarových zdrojů je nutné zvolit řešení, které sice dosahuje nižší propustnosti, ale je zde ve velké míře šetřeno s hardwarovými zdroji obvodů FPGA. Pro návrh tedy bylo použito iterační metody s vnitřním zřetězením iterací, základní myšlenka této techniky je zobrazena na obrázku 2.17.

Vstupy a výstupy architektury AES

data_in(127:0) – 128-bitový vstup reprezentující data, který mají být šifrovaná nebo dešifrovaná. Data jsou načítána do vnitřního registru při náběžné hraně hodinového signálu CLK a řídícího signálu INPUT_VALID nastaveného do úrovně logická 1.

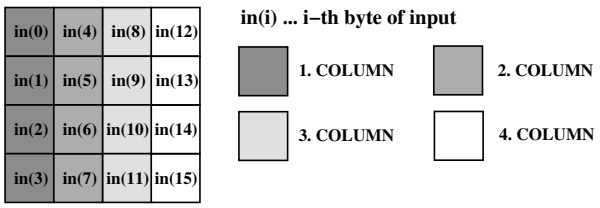
key(127:0) – vstup o šířce 128 bitů reprezentuje šifrovací klíč pro šifrování a dešifrování algoritmem AES. Klíč na vstupu musí být stabilní po dobu běhu procesu Key expansion.

data_out(127:0) – 128-bitový výstup reprezentující výsledek procesu šifrování nebo dešifrování. Data mouhou být z výstupu čtena při nástupné hraně hodinového signálu CLK a platnost těchto dat na výstupu je indikována kontrolním signálem OUTPUT_VALID, který je po dobu platnosti dat na výstupu nastaven do aktivní úrovně.

input_valid – tento vstup, synchronizovaný hodinovým signálem CLK, spouští načítání vstupních dat do vnitřního registru a také šifrování nebo dešifrování samotné při jeho aktivní úrovni.

output_valid – tento výstupní signál je v aktivní úrovni pokud je výsledek šifrování vystaven na výstupu data_out

Algoritmus AES je náročnější na hardwarovou implementaci, než je algoritmus DES. Použití stejného iteračního zpracování jako tomu bylo u návrhu a realizace architektury DES bylo s ohledem na požadavky dané na systém, především na velikost řešení, nemožné. Zpracování celého bloku vstupním dat jako celku, jako tomu bylo u DES, není v případě minimalizace velikosti výsledného obvodu příliš vhodné. Lze zde ovšem využít zřetězeného zpracování vstupního bloku po částech [24]. Těmito částmi jsou jednotlivé sloupce, pokud si představíme vstupní blok dat jako matici o velikosti 4x4 B (obrázek 4.2).



Obrázek 4.2: Mezivýsledek při výpočtu iterací architektury AES

Zřetězení lze ovšem zařadit pouze v rámci jedné iteraci kvůli datovým závislostem mezi jednotlivými koly výpočtu. V jedné iteraci jsou tedy fáze načítání dat do linky, vykonání

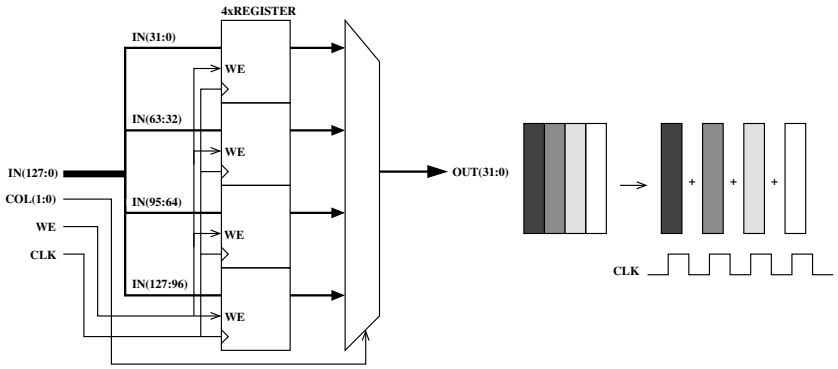
jednotlivých operací nad těmito daty a fáze doběhnutí (uložení dat do bufferu). Zřetězení jedné iterace je popsáno v kapitole 4.2.1.

Na obrázku 4.1 je znázorněno blokové schéma architektury AES. Vstupní blok je v prvním kroku převeden na posloupnost čtyř 32-bitových bloků (sloupců matice). Tvorba této posloupnosti je realizována v jednotce nazvané Serializer. Výstup této komponenty je po zašumění dat klíčem zapisován do vnitřního bufferu (obr. 4.4). Při fázi načítání dat je vstupem bufferu blok, který připočítává k výstup serializéru šifrovací klíč. Po načtení celého bloku dat o velikosti 128 bitů je započata fáze šifrování. Ta se provádí pro 128-bitový šifrovací klíč v 10 iteracích. V každé iteraci se data zpracovávají po 32-bitových blocích, přičemž výpočet je zřetězen v uzavřené lince, která obsahuje 4 stupně (obr. 4.15).

4.1 Popis jednotlivých částí architektury

4.1.1 Komponenta Serializer

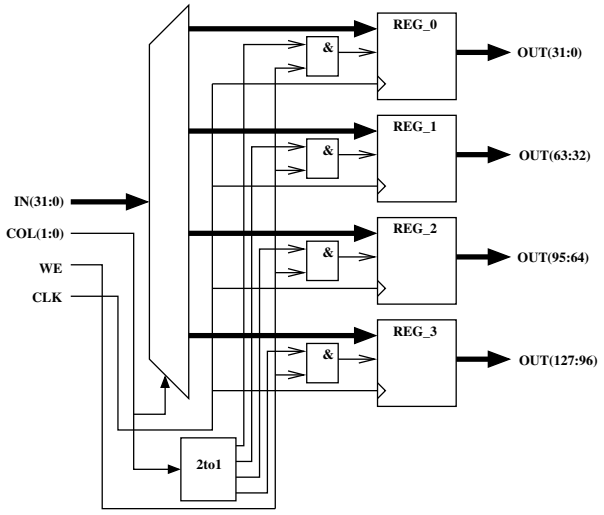
Jelikož jsou data při šifrování zpracovávána po blocích o velikosti 32 bitů je nutné zajistit, aby byl vstupní blok o velikosti 128 bitů převeden na sekvenci 32-bitových bloků. Jednotka Serializer zajišťuje převod vstupního bloku o velikosti 128 bitů na matici o velikosti 4x4 B, kterou uloží po sloupcích do čtyř vnitřních registrů o velikosti 4x8 bitů. Dalším úkolem tohoto obvodu je vystavení příslušného sloupce na výstup podle 2-bitového vstupu COL, při správné posloupnosti na tomto vstupu lze zajisti na výstupu příslušnou sekvenci sloupců matice, kdy 4 po sobě následující takty jsou postupně poslány na zpracování všechny 4 sloupce matice (obr. 4.3).



Obrázek 4.3: Zapojení obvodu Serializer

4.1.2 Komponenta Buffer

Na obrázku 4.4 je zobrazen Buffer, ve kterém jsou ukládány výsledky jednotlivých iterací. Jelikož jsou iterace zřetězené, zapisuje se výsledek iterace do bufferu po dobu čtyř taktů. Buffer obsahuje 4 registry, pro každý sloupec 128-bitového bloku je určen jeden. Registr pro zápis se vybírá vstupem COL. Vstupní signal WE povoluje zápis sloupce do buffer. Aby byla do registru zapsána vstupní data, musí být vstupní signál při nástupné hraně hodinové signálu v úrovni logická 1 a vstup COL musí adresovat příslušný registr. Výstupem tohoto bloku je matice 4x4 B o celkové velikosti 128 bitů. Tento výstup je veden na vstup obvodu realizující transformaci ShiftRows a také na výstup celé architektury AES.



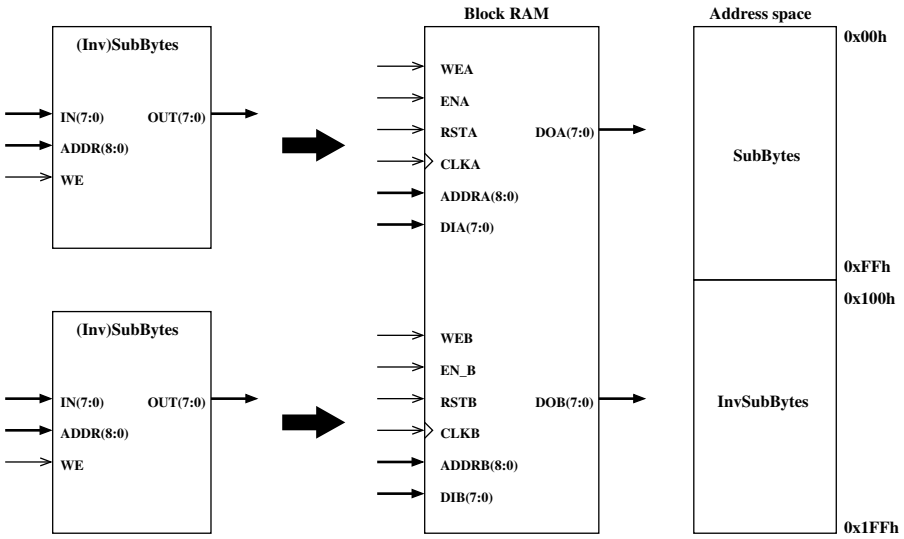
Obrázek 4.4: Obvodové zapojení komponenty Buffer

4.1.3 Návrh transformace SubBytes a InvSubBytes

Rychlost této transformace má velký vliv na celkovou rychlost šifrování celé architektury. Tuto operaci lze realizovat dvěma způsoby. Prvním způsobem je použití kombinačního obvodu, který transformaci realizuje jako výpočet nad galoisovým polem $GF(2^8)$. Tento výpočet je ovšem poměrně náročný a vnaší do systému značné zpoždění, které výrazně zpomaluje výkonnost navrhovaného zřetězeného zpracování. Takto navržený obvod pro výpočet transformace SubBytes měl podle syntetizátoru zpoždění přes 25 ns, což by znamenalo nutnost zařazení dalších stupňů do obvodu pro tuto transformaci, čímž by počet stupňů výrazně stoupl. Z těchto důvodů byl do systému zasazen obvod pro operace SubBytes, realizovaný pomocí paměti RAM. Dalším důvodem pro takovou realizaci SubBytes byla možnost umístění do vestavěných dualportových blokových pamětí BlockRAM v obvodu Spartan 3. Tyto blokové paměti jsou v této architektuře k dispozici 4, přičemž pro navrhovanou architekturu AES se využíjí pouze 2.

V každé iteraci šifrování resp. dešifrování AES se transformace SubBytes resp. InvSubBytes aplikuje na 16 B, při procesu Key expansion se v každé iteraci výpočtu klíčů aplikuje na 4 B. Což vychází na 36 paměťovch prvků pro tuto architekturu, což by zabralo 18 dualportových blokových pamětí RAM. Pro navrhovanou architekturu s vnitřním zřetězením iterací, použití stejné paměti pro SubBytes a InvSubBytes a sdílení operace SubBytes pro šifrování a proces výpočtu iteračních klíčů Key expansion dostáváme potřebu 4 paměťových prvků, které je možné umístit do dvou dualportových blokových pamětí RAM.

Velikost paměti pro operaci SubBytes i InvSubBytes je 256x8 b. Předpočítaná data pro tyto transformace lze umístit do jedné paměti o velikost 512x8 b způsobem zobrazeným na obrázku 4.5, prvních 256 bytů je využito pro operaci SubBytes a druhá polovina je využita pro InvSubBytes [24].



Obrázek 4.5: Transformace SubBytes a InvSubBytes

4.1.4 Návrh transformace MixColumns a InvMixColumns

Tato transformace je popsána v kapitole 2.3.2. Transformace InvMixColumns je inverzní k transformaci MixColumns a používá se při dešifrování. Transformace jsou realizovány pomocí kombinační logiky. Z následujících rovnic je patrné, že lze paralelně zpracovat výsledek transfromace MixColumns a zároveň data, která jsou základem pro výpočet operace InvMixColumns z výsledku transformace MixColumns. Následující rovnice vyjadřují transformaci MixColumns jednoho sloupce mezivýsledku:

$$mx'_{0,c} = (02) * s_{0,c} \oplus (03) * s_{1,c} \oplus s_{2,c} \oplus s_{3,c} \tag{4.1}$$
$$mx'_{1,c} = s_{0,c} \oplus (02) * s_{1,c} \oplus (03) * s_{2,c} \oplus s_{3,c} \tag{4.2}$$
$$mx'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (02) * s_{2,c} \oplus (03) * s_{3,c} \tag{4.3}$$
$$mx'_{3,c} = (03) * s_{0,c} \oplus s_{1,c} \oplus s_{2,c} \oplus (03) * s_{3,c} \tag{4.4}$$

Následující rovnice vyjadřují výpočet operace InvMixColumns pro jeden sloupec mezivýsledku:

$$imx'_{0,c} = (0e) * s_{0,c} \oplus (0b) * s_{1,c} \oplus (0d) * s_{2,c} \oplus (09) * s_{3,c} \tag{4.5}$$
$$imx'_{1,c} = (09) * s_{0,c} \oplus (0e) * s_{1,c} \oplus (0b) * s_{2,c} \oplus (0d) * s_{3,c} \tag{4.6}$$
$$imx'_{2,c} = (0d) * s_{0,c} \oplus (09) * s_{1,c} \oplus (0e) * s_{2,c} \oplus (0b) * s_{3,c} \tag{4.7}$$
$$imx'_{3,c} = (0b) * s_{0,c} \oplus (0d) * s_{1,c} \oplus (09) * s_{2,c} \oplus (0e) * s_{3,c} \tag{4.8}$$

Na prvním bytu sloupce je znázorněno jak lze výpočet rozložit:

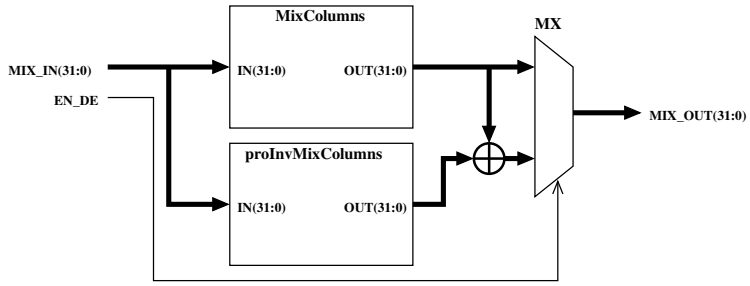
$$imx'_{0,c} = (02)*s_{0,c} \oplus (0c)*s_{0,c} \oplus (03)*s_{1,c} \oplus (08)*s_{1,c} \oplus s_{2,c} \oplus (0c)*s_{2,c} \oplus s_{3,c} \oplus (08)*s_{3,c} \quad (4.9)$$

Výpočet transformace InvMixColumns lze tedy zapsat jako kombinaci výsledku operace MixColumns a výpočtu nazvaného proInvMixColumns:

$$imx'_{0,c} = mx'_{0,c} \oplus \underbrace{((0c) * s_{0,c} \oplus (08) * s_{1,c} \oplus (0c) * s_{2,c} \oplus (08) * s_{3,c})}_{proInvMixColumns_{0,c}} \quad (4.10)$$

Operace MixColumns a proInvMixColumns lze vykonávat paralelně, obě operace požadují jako vstup stejná data.

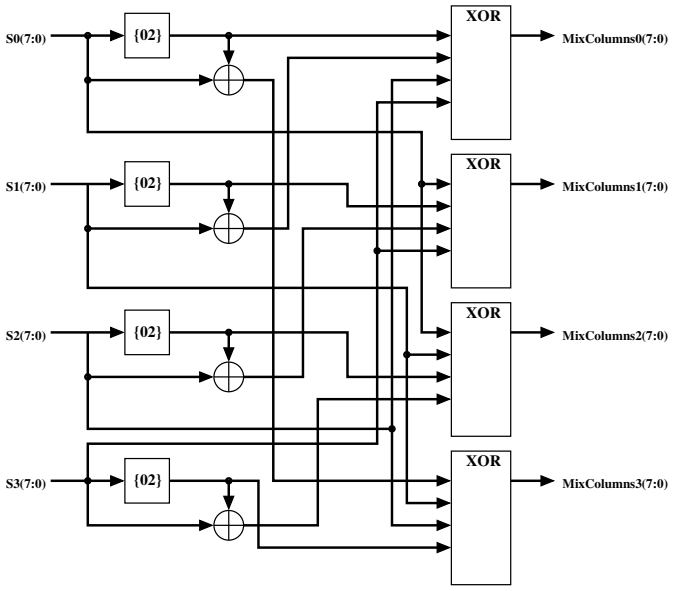
Návrh zapojení tohoto obvodu je na obrázku 4.6. Vstupní data jsou přivedena na oba bloky vykonávající transformace MixColumns a proInvMixColumns, výstupy těchto bloků jsou přivedeny na multiplexor, přičemž nad výsledkem transformace MixColumns a proInvMixColumns je provedena logická operace XOR mezi jednotlivými bity výstupních bloků. Výstup celého bloku je řízen signálem EN_DE. V případě, že se data šifrují, je výstupem výsledek operace MixColumns, v případě dešifrování je to výsledek operace InvMixColumns.



Obrázek 4.6: Obvodové zapojení operací MixColumns a InvMixColumns

Na následujícím obrázku je zobrazeno obvodové zapojení operace MixColumns. Tento obvod transformuje výpočtem 32-bitový vstup na výstup o stejné délce. Operace {02} se realizují v tělese $GF(2^8)$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ [21]. V transformaci je využita skutečnost, že násobení {03} v tělese $GF(2^8)$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$ lze zapsat následujícím způsobem:

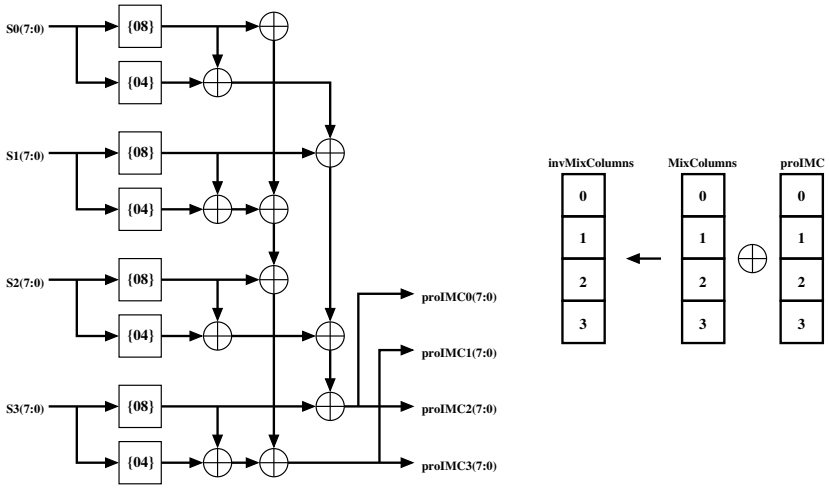
$$(03) \cdot x = (02) \cdot x \oplus (01) \cdot x = (02) \cdot x \oplus x \quad (4.11)$$



Obrázek 4.7: Operace MixColumns

Při operaci proInvMixColumns se počítají data pro výpočet transformace InvMixColumns na základě rovnice 4.10. Obvodové zapojení komponenty proInvMixColumns je zobrazeno na obr. 4.8. Podle rovnice 4.10 se zde aplikují operace {08} a {0c}. Operaci {0c} lze zapsat takto:

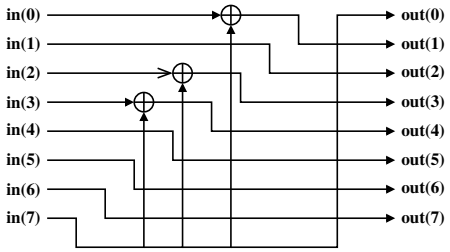
$$(0c) \cdot x = (08) \cdot x \oplus (04) \cdot x \tag{4.12}$$



Obrázek 4.8: Operace proInvMixColumns

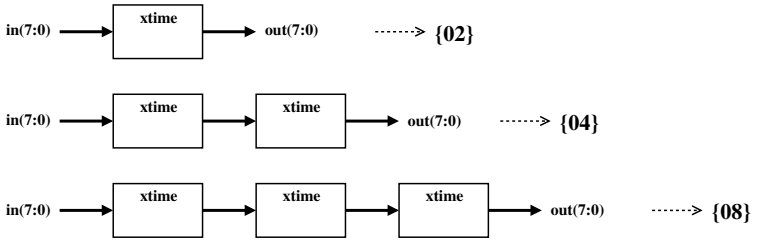
Všechny operace {02}, {04} a {08} použité v komponentách MixColumns a proInvMixColumns lze realizovat pomocí operace *xtime*, jejíž zapojení je uvedeno na obrázku 4.9. Jedná se o jednoduchý obvodu s 8-bitovým vstupem a výstupem. Mezi vybranými vstupy

je aplikována logická operace XOR, jejíž výsledek je přiveden na výstup. Některé vstupy jsou přímo propojeny na výstup.



Obrázek 4.9: Operace xtime

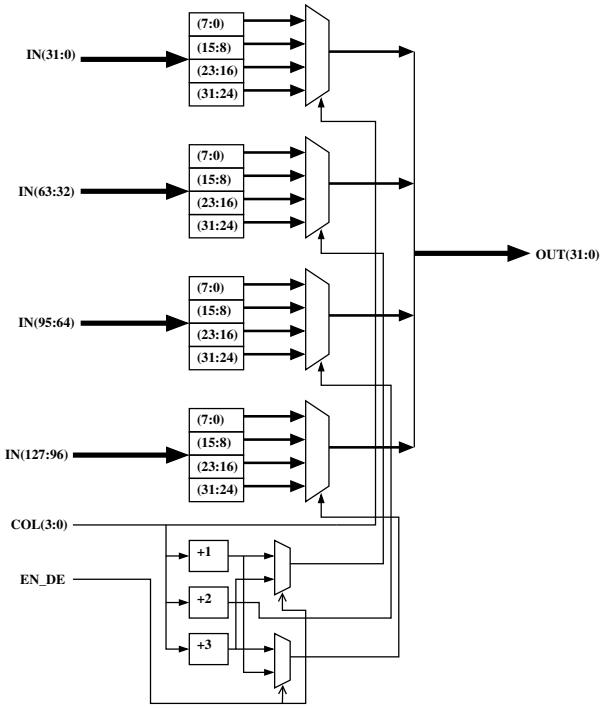
Jakým způsobem se pomocí xtime realizují operace {02}, {04} a {08} je znázorněno na následujícím obrázku 4.10.



Obrázek 4.10: Operace {02}, {04} a {08}

4.1.5 Návrh transformace ShiftRows a InvShiftRows

Na následujícím obrázku je zobrazeno navržené obvodové zapojení transformace ShiftRows a InvShiftRows. ShiftRows se uplatňuje při procesu šifrování, InvShiftRows je operace inverzní a je použita při dešifrování. Výběr operace je prováděn signálem EN_DE, sloupec mezivýsledku je vybírán pomocí vstupu COL.



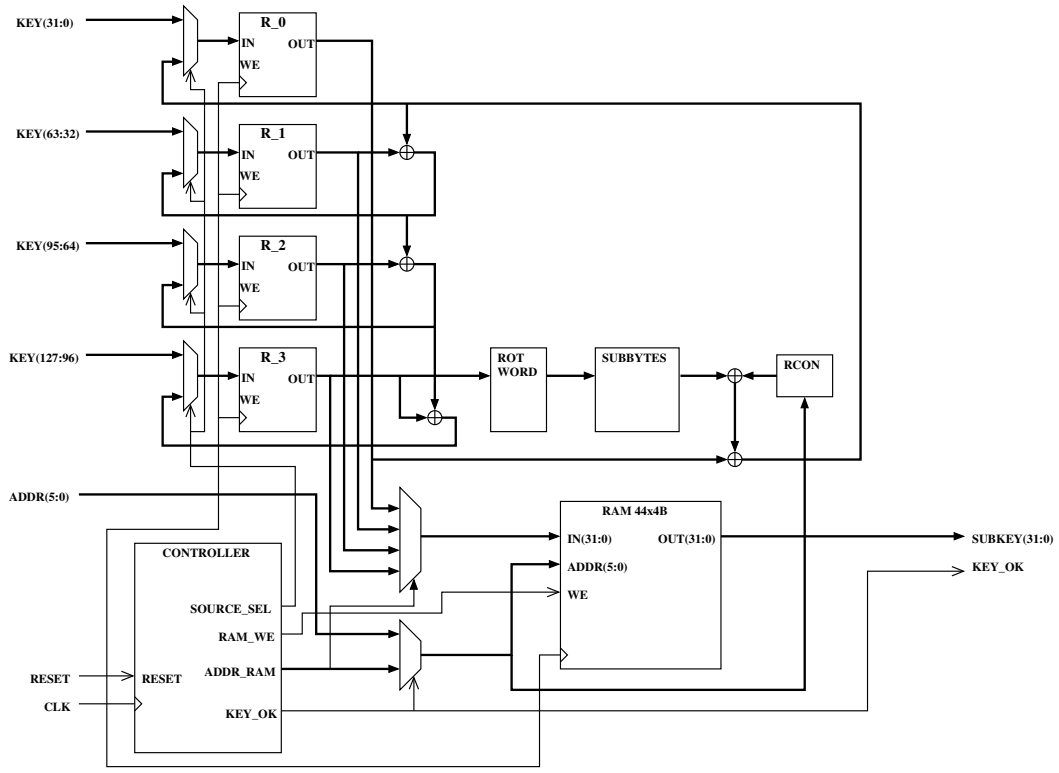
Obrázek 4.11: Obvodové zapojení operací ShiftRows a InvShiftRows

Další funkcí tohoto obvodu je díky možnosti vybrat výstupní blok, je stejně jako u serializéru převod vstupní matice obsahující 128-bitový blok mezivýsledku na sekvenci čtyř 32-bitových bloků, které představují sloupce vstupní matice.

4.1.6 Návrh komponenty Key schedule

Tento blok zajišťuje výpočet iteračních klíčů, na základě vstupního šifrovacího klíče. Navrhovaná architektura je určena pro šifrování pomocí klíče s délkou 128 bitů. Pro tuto délku klíče se šifrování a dešifrování vykonává v 10 iteracích. Pro každou tuto iteraci je nutné vypočítat subklíč o velikosti 128 bitů. Výpočet těchto podklíčů zajišťuje obvod zobrazený na obrázku 4.12.

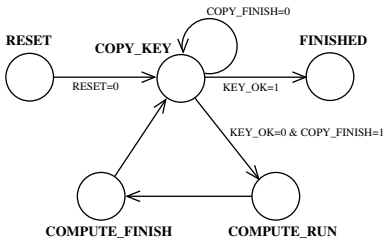
Dále tento obvod zajišťuje uložení těchto klíčů a jejich distribuci pro jednotlivé iterace na základě 6-bitové vstupní adresy ADDR. Obvod je synchonizován vstupním globálním hodinovým signálem CLK a asynchronně resetován globálním signálem RESET. Vstupem obvodu je šifrovací klíč o délce 128 bitů, ze kterého se počítají iterační klíče. Dalším vstupem je adresa sloupce iteračního klíče, určeného pro transfromaci AddRoundKey. Na výstup obvodu Key schedule, o velikost 32 bitů, je přiveden výstup interní paměti RAM a také signál KEY_OK, který indikuje dokončení výpočtu iteračních klíčů.



Obrázek 4.12: Obvodové zapojení Key schedule

Systém pracuje ve dvou režimech, v prvním počítá podklíče pro jednotlivé iterace, které ukládá do paměti RAM, ve druhém předává podklíče na výstup na základě vstupní adresy. Výpočet iteračních klíčů je automaticky spuštěn po resetu systému a pracuje podle stavového diagramu zobrazeného na obrázku 4.13. V prvním kroku se vstupní klíč KEY zapisuje do paměti RAM v podobě matice 4×4 byty, během 4 taktů hodinového signálu. Po úvodním zapsání šifrovacího klíče se provádí výpočet podklíčů, další klíč je vždy počítán z klíče předchozího. Výpočet jednoho klíče je proveden během jednoho taktu a poté je opět spuštěno kopírování podklíče do paměti RAM. Délka tohoto kopírování je stejně jako u vstupního klíče 4 cykly. Výpočet a zápis se opakuje tak dlouho, dokud není řídící jednotkou nastaven signál KEY_OK do úrovně logická 1. Celý výpočet pole iteračních klíčů trvá 65 taktů. První sloupec subklíče (z celkových 4) je uložen na adrese, která odpovídá příslušné iteraci při šifrování. Subklíče se počítají stejným způsobem při procesu šifrování i dešifrovaní, i když by bylo výhodné je pro dešifrování ukládat do paměti v opačném pořadí, ve kterém jsou používány. Stejné ukládání bylo zvoleno z důvodu možné rozšiřitelnosti architektury, kde by bylo možné mít šifrovací klíč a podklíče předpočítané a uložené v paměti ROM.

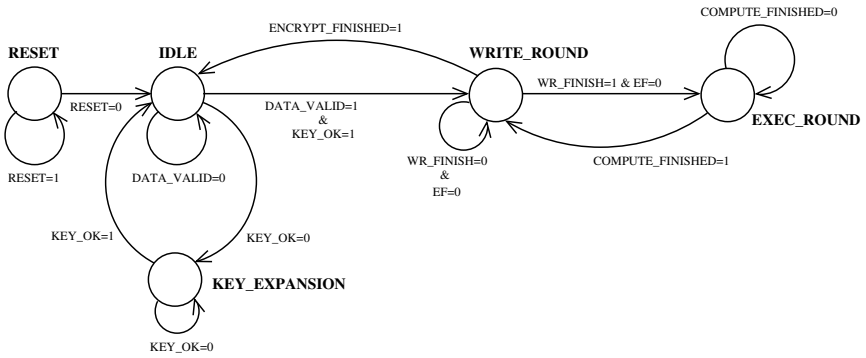
Ve druhém režimu si lze představit celý obvod jako asynchronní pamět RAM, ze které se pouze čtou klíče pro jednotlivé iterace na základě vstupní adresy.



Obrázek 4.13: Stavový diagram procesu Key expansion

4.2 Šifrování a dešifrování

V předchozí kapitole byly popsány jednotlivé komponenty architektury AES. Již bylo řečeno, že při návrhu byla snaha o optimální velikost výsledného obvodu při zachování rychlosti minimálně 100 Mb/s. V případě algoritmu AES nejsou procesy šifrování a dešifrování co do délky jednotlivých operací a jejich pořadí stejné, jako tomu je u algoritmu DES, proto bylo dalším cílem navrhnout obvod takovým způsobem, aby rychlost šifrování a dešifrování byla shodná.



Obrázek 4.14: Stavový diagram procesu šifrování a dešifrování AES

Šifrování a dešifrování pracuje podle předchozího stavového diagramu 4.14. Tento stavový diagram je realizován v řídící jednotce Controller. Po resetu systém přejde do stavu výpočtu šifrovacího klíče KEY_EXPANSION, ve kterém setrvává pokud je vstupní signál řídící jednotky KEY_OK = 0. Pokud je výpočet klíče dokončen obvod čeká na vstupní data. Pokud jsou vstupní data platná, zkopírujeme je do vnitřního bufferu architektury (obr. 4.4), tato akce trvá 4 takty. Po načtení dat je spuštěn výpočet, ve kterém jsou na jednotlivé sloupce mezivýsledku, uloženého v bufferu, aplikovány jednotlivé transformace, jejichž návrh a obvodové zapojení byly popsány výše. Pokud je spuštěn výpočet a do bufferu se nezapisují žádná data, systém se nachází ve stavu EXEC_ROUND. Po dokončení operací nad sloupcem mezivýsledku (1×4 B), jsou data uložena na svoji pozici do vnitřního bufferu, obvod tedy přejde do stavu WRITE_ROUND. Stavy ukládání a výpočtu se střídají do

okamžiku dokončení všech iterací šifrovacího nebo dešifrovacího procesu. Ukončení indikuje vnitřní čítač iterací, umístěný v řídící jednotce.

Jak už bylo řečeno z důvodu minimalizace velikosti obvodu byla pro návrh použita metoda vnitřního zřetězení cyklů. Obvod poté obsahuje logiku pro výpočet pouze 1/4 jedné iterace šifrování nebo dešifrování.

4.2.1 Zřetězení výpočtu iterace

Požadavkem na výslednou architekturu byla minimální propustnost 100 Mb/s. Jelikož sekvenční zpracovaní jednotlivých iterací, jaké bylo zařazeno do architektury DES, by vnášelo příliš velké zpoždění, které by nedovolovalo splnit požadavek na rychlost šifrování, bylo nutné zvýšit výpočetní výkon systému, aby mohlo být dosaženo požadované rychlosti. Zvýšení výpočetního výkonu sekvenčního přístupu je možno dosáhnout jednak zřetězením, a jednak replikací (opakováním) funkčních jednotek. Jde o dva nezávislé přístupy k paralelismu, časový a prostorový, přičemž oba mohou být použity současně s kumulací přínosů.

Prostorový paralelismus vede ke značnému zvýšení potřebných hardwarovových zdrojů a jelikož navrhujeme architekturu se snahou o minimalizaci velikosti výsledného řešení, je pro nás vhodnější zařazení časového paralelismu (zřetězení), kde nárust potřebných zdrojů není zdaleka tak velký. Zřetězené zpracování úloh je založeno na tom, že vykonávané úlohy se rozdělí na sekvenci kroků, nejlépe stejného trvání, přičemž v každém kroku se používají samostatné technické prostředky. Tak je možné, že v jednom okamžiku se provádí několik úloh, každá je ovšem v jiném stavu rozpracování a využívá jiný stupeň linky.

Požadavky a režie zřetězeného zpracování [29]:

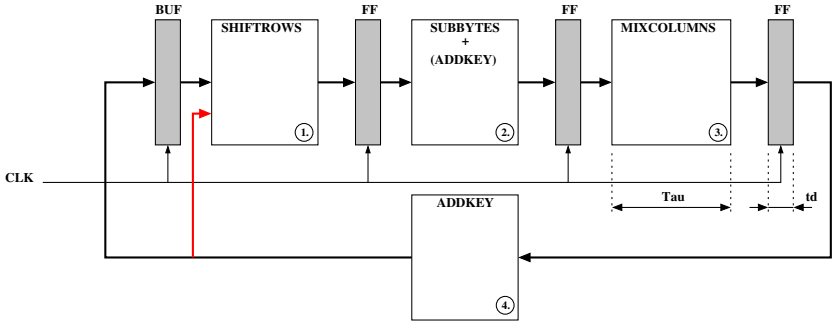
- nepřetržitý přísun dat nebo kódů, nad nimiž je třeba provádět stejnou základní operaci nebo je zpracovávat podobným způsobem
- zpracování musí být možno rozdělit na sekvenci (nezávislých) kroků, realizovaných jednotlivými stupni řetězu,
- trvání jednotlivých kroků by mělo být přibližně stejné.

Na celkové dosažitelné zrychlení má vliv:

- nezbytné zastavování linky,
- náběh a doběh řetězeného zpracování při konečném počtu N zpracovaných položek a
- zpoždění oddělovacích registrů.

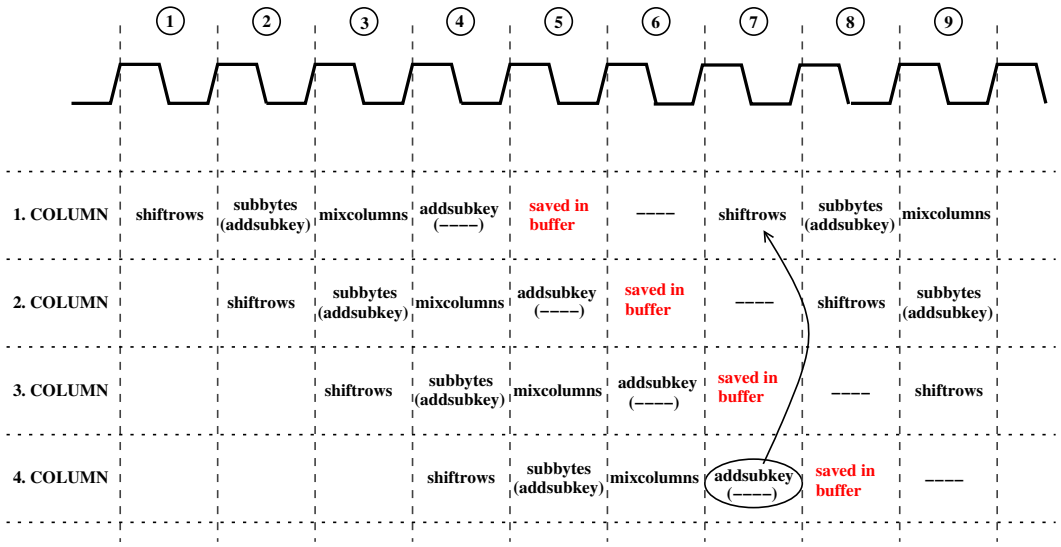
Realizace zřetězení v architektuře AES

Při zařazení zřetězení do systému je nejdůležitějším krokem vhodný návrh jednotlivých stupňů a jejich počet. Mezi jednotlivé stupně jsou pak zařazeny oddělovací registry. Pro zřetězení iterace v architektuře AES byly vybrány 4 stupně, přičemž v každém stupni je realizována jedna nebo dvě transformace. Jakým způsobem byly jednotlivé stupně rozděleny je patrné z obrázku 4.15, stupně jsou číslovány v pořadí jakým přes ně data procházejí. Data se načítají do zřetězené linky z Bufferu, který má také úlohu oddělovacího registru, ve kterém se ukládá výsledek posledního stupně.



Obrázek 4.15: Návrh zřetězeného zpracování iterace AES

První stupeň realizuje transformaci ShiftRows, v druhém stupni se na 32-bitový blok aplikují transfromace SubBytes v případě šifrování a dvě transformace InvShitRows a AddRoundKey v případě dešifrování. Třetím stupněm jsou náročně operace MixColumns při šifrování a InvMixcolumns při dešifrování. V posledním stupni se prování transformace AddRoundKey v případě, pokud jsou data šifrována. Větší optimalizaci při návrhu zřetězení brání v určité míře fakt, že při šifrovaní dat jsou transformace aplikovány v poněkud jiném pořadí než při dešifrovaní jejich operace inverzní.



Obrázek 4.16: Časový diagram zřetězeného zpracování iterace v architektuře AES

Při zřetězení je nutné počítat s hardwarovou režií v podobě oddělovacích registrů, v tomto případě se jedná o 3 registry, funkci oddělovacího regitru plní totiž i vnitřní buffer pro ukládání mezivýsledků, který by byl součástí i nezřetězeného zpracování.

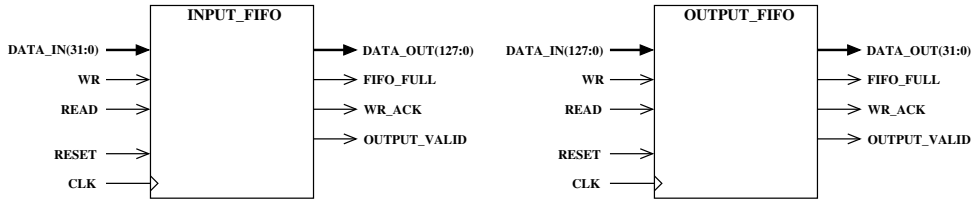
Na obrázku 4.16 je zobrazen časový diagram zřetězeného zpracování iterací. Pro počet stupňů $k = 4$ a počet zpracovávaných položek $N = 4$ by měl být počet taktů pro jejich zpracování roven $(k + N - 1) = 7$.

Jak vidět z časového diagramu na obrázku 4.16 jsou ale data potřebná pro transformaci ShiftRows předána z výstupu transformace AddRoundKey a zpracována v jednom taktu. Tato skutečnost vede ke snížení počtu taktů, potřebných pro zpracování všech 128 bitů mezivýsledku na 6. Zrychlení jedné iterace oproti nezřetězenému zpracování je vyjádřeno v rovnici 4.13, tata hodnota je ovšem vyjádřena pouze pro více iterací, přičemž v poslední iteraci bude výpočet dat se zapsáním do bufferu trvat celých 7 taktů.

$$S_n = \frac{T_1}{T_k} = \frac{Nk\tau}{(k+N-2)(\tau+t_d)} = \frac{4 \cdot 4 \cdot 12ns}{(4+4-2)(12ns+2ns)} = 2,28 \tag{4.13}$$

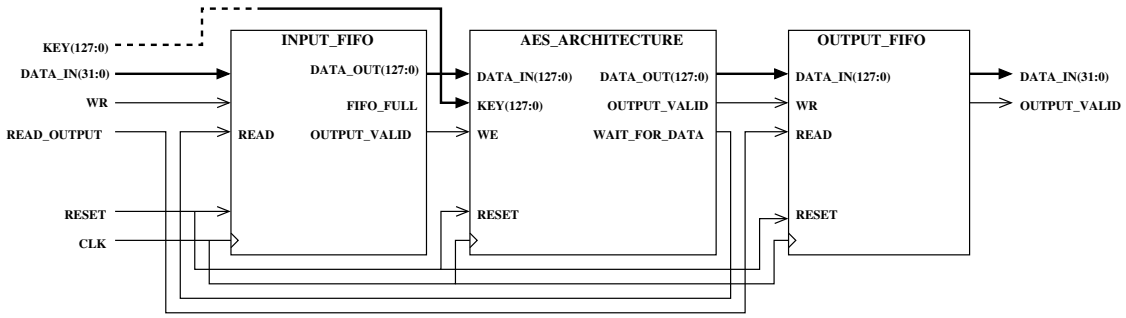
4.3 Režim blokového šifrování AES

Pro šifrování bloků, které jsou delší než velikost bloků šifrovaného alogirtmem AES byl zvolen, stejně jako u architektury DES, režim ECB, ve kterém se data šifrují nezávisle na ostatních šifrovaných blocích. I v případě AES byly přidány na vstup i výstup jednotky vyrovnávací paměti realizované pomocí struktury FIFO. Jelikož AES pracuje s bloky o velikosti 128 bitů, musely být jednotky FIFO použité pro šifrování DES upraveny na tuto šířku (obr. 4.17). Tyto jednotky pracují způsobem popsaným v kapitole 3.2, která se zabývá režimem blokového šifrování DES.



Obrázek 4.17: Jednotka FIFO určená pro AES [32]

Celá architektura bude mít po přidání vstupní a výstupní jednotky FIFO strukturu zobrazenou na obrázku 4.18. Šifrovací klíč je možné je realizovat jako vstup celého obvodu, nebo jako pevně zadanou hodnotu uvnitř architektury.



Obrázek 4.18: Struktura architektury AES s přidanými jednotkami FIFO

4.4 Implementace a syntéza architektury AES

Po fázi návrhu byla architektura implementována v jazyce VHDL a otestována pomocí simulačních nástrojů. Pomocí simulace byl ověřen navržený počet taktů potřebných pro zašifrování resp. dešifrování jednoho bloku dat o velikosti 128 bitů. Jeden blok dat je zpracován za 66 cyklů, z této informace lze vyjádřit maximální teoretickou propustnost architektury (rovnice 4.14), kterou lze vypočítat při znalosti délky výpočtu v taktech a maximální možné frekvence obvodu.

$$P = \frac{block\_size}{cycles} \cdot f_{aes} = \frac{128}{66} \cdot f_{aes} \tag{4.14}$$

Proces rozšíření klíče pracuje po resetu systému po dobu 65 taktů, což je doba po které lze šifrovat/dešifrovat první data.

4.4.1 Dosažené výsledky

Po simulaci architektury byla provedena jeho syntéza pro FPGA typu Spartan 3. Počet potřebných hardwarových zdrojů v tomto obvodu je vypsán v tabulce 4.1. Z těchto výsledku lze říci, že výsledná implementace zabere asi 60 % zdrojů dostupných v obvodu Spartan 3.

	Rozdělení	Počet zdrojů	Dostupné zdroje	Obsazeno (%)
Slices	-	586	768	76
Slices flip flops	-	489	1536	31
4 LUTS		1120	1536	72
	logic	1024	-	-
	RAM	96	-	-
BRAM	-	2	4	50

Tabulka 4.1: Obsazené zdroje architektury AES

Pro zjištění nejkritičtějších částí z hlediska potřeby hardwarových zdrojů byla vytvořena tabulka 4.2. Jak z ní lze zjistit, nejvíce zdrojů zabírá blok pro výpočet iteračních klíčů, pokud by byla použita varianta uložení předpočítaných klíčů do paměti ROM, snížila by se potřeba zdrojů téměř o 30%.

	Slices	Slices flip flops	4 LUTS	BRAM
Serializer	106	128	64	-
AddRoundKey	18	-	32	-
Buffer	2	128	4	-
SubBytes	-	-	-	2
ShiftRows	60	-	112	-
MixColumns	84	-	154	-
KeySchedule	235	144	392	-
Controller	24	14	47	-
Ostatní	57	75	315	-

Tabulka 4.2: Obsazené zdroje architektury AES jednotlivými komponentami

Výstupem procesu syntézy je také informace o teoretické maximální frekvenci, na které může daný obvod pracovat. Tato maximální frekvence obvodu pro šifrování AES je 118 MHz. S touto informací již lze dopočítat maximální propustnost P architektury:

$$P_{aes} = \frac{block\_size}{cycles} \cdot f_{aes} = \frac{128}{66} \cdot 118 = 228\ Mb/s \tag{4.15}$$

4.4.2 Srovnání se softwarovým řešením

Pro srovnání byl opět použit mikroprocesor určený pro vestavěné systémy MSP430F1611, umístěný na platformě FITKit, kde byla implementace [25] testována. Při testu bylo zašifrováno 1500 bloků o velikosti 128 bitů. Tento výpočet proběhl na mikrokontroléru za 159,9 s. Dostáváme tedy rychlost šifrování:

$$P_{aes\_sw} = \frac{1500 \cdot 128}{159,9} = 1200\ b/s \tag{4.16}$$

Z této hodnoty je opět patrné, že šifrování na tomto mikrokontroléru je velmi pomalé a velmi dobře se zde uplatní hardwarová akcelerace šifrování pomocí obvodů FPGA určených pro vestavěné systémy. Šifrování na obvodu Spartan 3 může teoreticky pracovat $190 \cdot 10^3$krát rychleji než na mikrokontroléru MSP430. I při výrazné optimalizaci softwarové implementace určené pro vestavěné systémy by zrychlení při hardwarové akceleraci bylo v řádech tisíců.

Pro simulaci a syntézu byly použity stejné nástroje jako u architektury DES.

Kapitola 5

Závěr

Na základě znalostí principu algoritmů DES a AES byla návržena obvodová realizace šifrování a dešifrování dat pro oba tyto algoritmy. V obou architekturách bylo navrženo zpracování dat pomocí iterační metody, která umožňuje úsporu potřebných zdrojů. V případě návrhu architektury AES bylo dále použito vnitřní zřetězení iterací, opět při snaze o snížení potřebných hardwarových zdrojů. Z hlediska propustnosti byl návrh proveden s ohledem na požadavek minimální rychlosti šifrování 100 Mb/s, metrikou pro množství hardwarových zdrojů potřebných pro implementované architektury byla možnost jejího umístění na FPGA typu Spartan 3. Jednotlivé komponenty navržených architektur byly detailně popsány v kapitole 3 pro architekturu DES a v kapitole 4 pro architekturu AES.

Návržené architektury byly implementovány v jazyce VHDL a jejich funkčnost byla ověřena pomocí simulačních nástrojů. Výsledekem syntézy, která byla vykonána nad implementovanými architekturami, jsou informace o velikosti jednotlivých realizací a také údaj o maximální teoretické frekvenci, na které tyto obvody mohou pracovat. Z výsledků je patrné, že snaha o minimalizaci při návrhu byla úspěšná, obvody lze umístit i na obvody FPGA, které neobsahují velké množství dostupných hardwarových zdrojů, příkladem tedy může být obvod FPGA typu Spartan3 XC3S50-4PQ208C, který je součastí výukové platformy FITKit používané v rámci Fakulty informačních technologií na VUT v Brně. Množství potřebných zdrojů pro realizaci architektury DES je shrnuto v kapitole 3.3.1, pro architekturu AES jsou tyto údaje dostupné v kapitole 4.4.1.

Podle výsledků syntézy může navržená architektura DES pracovat na frekvenci až 104 MHz a s dobou výpočtu jednoho vstupní bloku 18 taktů tak může dosahovat propustnosti až 370 Mb/s. Navržený obvod pro šifrovaní pomocí algoritmu AES lze synchronizovat frekvencí až 118 MHz a při době výpočtu 66 cyklů je možné šifrovat rychlostí až 228 Mb/s. Těmito hodnotami byl splněn požadavek na minimální propustnost obvodů 100 Mb/s.

Srovnání se softwarovým řešením bylo provedeno na mikrokontroléru MSP43OF1611, který je určený pro vestavěné systémy a je také součástí platformy FITKit. Při srovnání architektury DES je možné dosáhnout, podle výsledků syntézy, s hardwarovou akcelerací až $201 \cdot 10^3$krát vyšší rychlosti šifrování oproti softwarovému řešení. V případě architektury AES byla softwarová implementace pro vestavěné systémy $190 \cdot 10^3$krát pomalejší než hardwarová realizace.

Do budoucna lze očekávat vývoj technologií FPGA, který umožní jiný přístup z pohledu optimalizace pomocí zřetězeného zpracování a prostorového paralelismu. Při dostatečném množství dostupných zdrojů lze prostorovým paralelismem v architektuře AES dosáhnout až 4x vyšší propustnosti při zhruba stejném nárustu potřebných hardwarových zdrojů. Součástí dalšího vývoje by mohlo být zařazení hardwarových realizací do některé funkční

aplikace. Z hlediska architektur samotných by bylo možné pokračovat v optimalizaci jednotlivých návrhů a to ve směru zvýšení propustnosti nebo snížení výsledné velikosti obvodu. V případě DES by bylo možné přidat zřetězené zpracování, které by vedlo ke zvýšení frekvence, na které by mohla architektura pracovat. Pro architekturu AES by mohl vývoj pokračovat v optimalizaci zřetězeného zpracování.

Literatura

[1] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone . *Handbook of Applied Cryptography*. CRC Press, 1997. ISBN 80-238-5400-3.

[2] Alireza Hodjat, Ingrid Verbauwhede. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA, 2004.

[3] Ondřej Baar. Historie šifrování. `http://www.owebu.cz/bezpecnost/vypis.php?clanek=1216`.

[4] Ondřej Bitto. Historie kryptologie. `http://www.fi.muni.cz/usr/jkucera/pv109/2003/xbitto.htm`.

[5] Cameron Patterson. High Performance DES Encryption in Virtex(tm) FPGAs Using Jbits(tm). In *FCCM '00: Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, page 113. IEEE Computer Society, 2000.

[6] Enoch O. Hwang. *Logic and microprocessor design with VHDL*. Thomson Canada Limited, 2006. ISBN 0-534-46593-5.

[7] David C. Feldmeier. A high-speed software DES implementation. Technical report, 1989.

[8] Frank Hoornaert, Jo Goubert, Yvo Desmedt. Efficient hardware implementation of the DES. Springer-Verlag, 1998.

[9] Gaël Rouvroy and François-Xavier Standaert and Jean-Jacques Quisquater and Jean-Didier Legat. Compact and Efficient Encryption/Decryption Module for FPGA Implementation of AES Rijndael Very Well Suited for Small Embedd. In *ITCC 2004, special session on embedded cryptographic hardware*, pages 583–587. IEEE Computer Society, 2004.

[10] Dr. B. R. Gladman. Ibm pc implementation of the des cryptographic algorithm. `www.ussrback.com/crypto/aes/des/`.

[11] Hans Eberle. A High-Speed DES Implementation for Network Applications. *Lecture Notes in Computer Science*, 740:521–539, 1993.

[12] Hanáček, P., Staudek, J. *Bezpečnost informačních systémů*. ÚSIS Praha, 2000. ISBN 80-238-5400-3.

[13] J. Leonard, W. H. Mangione-Smith. A Case Study of Partially Evaluated Hardware Circuits: Key-Specific {DES}. In W. Luk and P. Y. K. Cheung and M. Glesner, editor, *Field-Programmable Logic and Applications. 7th International Workshop*, volume 1304, pages 151–160, London, U.K., 1997. Springer-Verlag.

[14] Jens-Peter Kaps,Christof Paar. Fast DES Implementation for FPGAs and Its Application to a Universal Key-Search Machine. In *Selected Areas in Cryptography*, pages 234–247, 1998.

[15] Paul Kocher. Software model of asic des implementation. `http://www.cryptography.com`.

[16] Marc Davio, Yvo Desmedt, Jo Goubert, Frank Hoonaert, Jean-Jacques Quisquater. Efficient hardware and software implementations for the DES. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 144–146. Springer-Verlag New York, Inc., 1985.

[17] Matt Bishop. An Application of a Fast Data Encryption Standard Implementation. Technical report, Hanover, NH, USA, 1988.

[18] McLoone, M. McCanny, J.V. High-performance FPGA implementation of DES using a novel method for implementing the key schedule. In *Circuits, Devices and Systems, IEE Proceedings*, pages 373–378, 2003.

[19] Nadia Nedjah, Luiza de Macedo Mourelle , Marco Paulo Cardoso. A Compact Piplined Hardware Implementation of the AES-128 Cipher. In *Proceedings of the Third International Conference on Information Technology: New Generations*. IEEE Computer Society, 2006.

[20] National Institute of Standards and Technology. FIPS PUB 46-3: Data Encryption Standard (DES), oct 1999. supersedes FIPS 46-2.

[21] National Institute of Standards and Technology. Specification for the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001.

[22] Nazar A. Saqib, Francisco Rodriguez-Henriquez, and Arturo Diaz-Perez. A Compact and Efficient FPGA Implementation of the DES Algorithm. 2004.

[23] Panu Hamalainen, Timo Alho, Marko Hannikainen, Timo D. Hamalainen. Design and Implementation of Low-Area and Low-Power AES Encryption Hardware Core. In *DSD '06: Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 577–583. IEEE Computer Society, 2006.

[24] Pawel Chodowiec and Kris Gaj. Very Compact FPGA Implementation of the AES Algorithm. In *CHES*, pages 319–333. Springer-Verlag, 2003.

[25] Niyaz PK. Advanced encryption standard implementation in c. `http://www.hoozi.com/Downloads/AES_Encrypt.rar`.

[26] Ricardo Chaves, Georgi Kuzmanov, Stamatis Vassiliadis, Leonel Sousa. Reconfigurable Memory Based AES Co-Processor. In *International Parallel and Distributed Processing Symposium, 13th Reconfigurable Architectures Workshop*, pages 192 – . IEEE Computer, April 2006.

[27] RNDr. Pavel Vondruška. Cesta kryptologie do nového tisíciletí. *ComputerWorld 37/2000 - 40/2000*, 2000.

[28] Volnei A. Pedroni. *Circuit Design vith VHDL*. MIT press, 2004. ISBN 0-262-16224-5.

[29] Václav Dvořák, Vladimír Drábek. *Architektura procesorů*. VUTIUM Brno, 1999. ISBN 80-214-1458-8.

[30] Wilcox, D., Pierson, L., Robertson, P., Witzke, E.L., Gass, K. A DES ASIC Suitable for Network Encryption at 10 Gbps and Beyond. In *In: CHESS 99*, volume 1717. Springer Berlin / Heidelberg, 1999.

[31] Wong, K., Wark, M., Dawson, E. A Single-Chip FPGA Implementation of the Data Encryption Standard (des) Algorithm. In *In: IEEE Globecom Communication Conf.*, volume 2, pages 827 – 832, 1998.

[32] WWW stránky: Doc. Ing. Jaromír Kolouch,CSc. Možnosti realizace pamětí ram v obvodech fpga. `http://www.elektrorevue.cz/clanky/03027/index.html`.

Seznam tabulek

Seznam obrázků