VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

HARDWAROVÁ AKCELARACE ŠIFROVÁNÍ

HARDWARE ACCELERATING OF ENCRYPTION ALGORITHM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. David HRADIL
AUTHOR

VEDOUCÍ PRÁCE Ing. Jan KOŘENEK
SUPERVISOR

BRNO 2007

Zadání diplomové práce/4729/2006/xhradi06

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Hradil David, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Hardwarová akcelerace šifrování**
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s jazykem VHDL a algoritmem AES pro šifrování a dešifrování dat.
2. Navrhněte obvodovou realizaci této šifry.
3. Proveďte implementaci navrženého obvodu v jazyku VHDL.
4. Nad vytvořenou implementací proveďte simulaci a syntézu.
5. V závěru diskutujte vlastnosti navrženého řešení a urychlení proti softwarovému zpracování.

Literatura:

- Dle konzultace s vedoucím

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních 2 bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kořenek Jan, Ing.**, UPSY FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

   Jméno a příjmení: **Bc. David Hradil**
   Id studenta: 49422
   Bytem: Tyršova 75, 266 01 Beroun
   Narozen: 29. 03. 1983, Hranice
   (dále jen "autor")

a

2. **Vysoké učení technické v Brně**

   Fakulta informačních technologií
   se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
   jejímž jménem jedná na základě písemného pověření děkanem fakulty:

   .....
   (dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   diplomová práce

Název VŠKP: Hardwarová akcelerace šifrování
Vedoucí/školitel VŠKP: Kořenek Jan, Ing.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě	počet exemplářů: 1
elektronické formě	počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☒ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	Hradil D. .....
Nabyvatel	Autor

Abstrakt

Cílem této diplomové práce bylo vytvořit hardwarovou realizaci obvodu, která by představovala algoritmus AES. Motivací k vytvoření této realizace bylo urychlení vůči softwarovému šifrování. Urychlení je dosaženo pomocí speciálně navržených částí obvodu, které odpovídají jednotlivým operacím AES algoritmu. V prvním kroku bylo potřeba obvod navrhnout. V dalším kroku pak byl tento obvod popsán v jazyce VHDL. Dále pak následovala simulace a syntéza obvodu. Aby bylo možné dané hardwarové řešení porovnat se softwarovým zpracováním, byla za tímto účelem vytvořena SW implementace. Obě implementace byly vytvořeny pro platformu FITKit. HW implementace je vytvořena pomocí technologie FPGA a SW implementace pak pomocí mikrokontroléru. Výsledkem této práce pak bylo téměř tisícinásobné urychlení šifrování oproti klasickému softwarovému zpracování.

Klíčová slova

AES šifra, Rijndael, VHDL, FITKit, FPGA, mikrokontrolér.

Abstract

The goal of this thesis is to design a hardware realization of circuit which will implement the AES algorithm. A motivation was to make an acceleration against the classic software encryption. The acceleration is achieved by special designed parts of the circuit, which correspond to particular operations of the AES algorithm. First, there was necessary to design the circuit. In the next step there was a need to describe the designed circuit by the VHDL language. Then the circuit was simulated and synthesized. Due to comparing the circuit with software processing a software implementation was created. Both implementations were created for the FITKit platform. The hardware implementation is made by the FPGA technology and the software implementation is realized in a microcontroller. The result of the thesis is almost one thousandfold acceleration against the classic software encryption.

Keywords

AES cipher, Rijndael, VHDL, FITKit, FPGA, microcontroller.

Hardwarová akcelerace šifrování

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval zcela samostatně pod vedením Ing. Jana Kořenka. Všechny zdroje, prameny a literaturu, které jsem při tvorbě použil nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Brno, květen 2007 …..…....……

Bc. David Hradil

Poděkování

Tímto bych chtěl poděkovat panu Ing. Janu Kořenkovi za odborné vedení této diplomové práce a poskytnutí cenných rad a připomínek, které jsem využil při její tvorbě.

© David Hradil, 2007.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

OBSAH

1 Úvod…. 10

2 Základy kryptografie a AES algoritmus…. 12

2.1 Historie kryptografie.... 12

2.2 Základní pojmy a terminologie.... 14

2.3 Módy blokových šifer.... 16

2.3.1 Electronic Code Book (ECB).... 16

2.3.2 Cipher Block Chaining (CBC).... 17

2.3.3 Cipher Feedback (CFB).... 17

2.3.4 Output Feedback (OFB).... 18

2.4 Historické kryptografické metody.... 19

2.4.1 Steganografie.... 19

2.4.2 Substituční šifry.... 20

2.4.3 Transpoziční šifry.... 22

2.4.4 Kombinované šifry.... 23

2.5 Moderní symetrické šifry.... 24

2.5.1 Feistelova šifra.... 25

2.5.2 Šifra DES.... 26

2.6 Kryptografie s veřejným klíčem.... 27

2.6.1 Šifra RSA.... 29

2.7 Historie AES.... 30

2.8 Matematická příprava.... 31

2.9 Vlastní specifikace AES.... 32

2.10 Algoritmus šifrování.... 33

2.11 Algoritmus dešifrování.... 34

2.12 Popis jednotlivých operací.... 35

2.13 Expanze klíče.... 37

3 FITKit.... 39

3.1 Popis platformy FITKit.... 39

3.2 *Nahrávání programu a konfigurace FPGA* 40
3.3 *Propojovací systém FITKitu* 41
3.3.1 *Systém SPI* 41
3.3.2 *SPI na FITKitu* 43
4 Návrh hardwarového řešení 46
4.1 *Základní schéma a jeho popis* 46
4.1.1 *Rozhraní obvodu* 46
4.1.2 *Schéma obvodu a popis* 48
4.2 *Jednotka FIFO* 53
4.3 *Jednotka správy dat - stav* 55
4.4 *Jednotka SubBytes* 57
4.5 *Jednotka ShiftRows* 59
4.6 *Jednotka MixColumns* 61
4.7 *Jednotka AddRoundKey* 64
4.8 *Jednotka KeyExpansion* 65
4.9 *Jednotka ControlUnit* 71
5 Implementace softwarového řešení 74
5.1 *Popis základního programu* 75
5.2 *Popis jednotlivých operací* 76
6 Porovnání HW a SW implementace 78
6.1 *Naměřené výsledky u HW implementace* 77
6.2 *Naměřené výsledky u SW implementace* 80
6.3 *Srovnání HW a SW implementace* 81
7 Závěr 84
8 Literatura 85

Kapitola 1

1 Úvod

V dnešní době jsme svědky obrovského rozmachu informačních technologií. Informační technologie nás obklopují téměř všude, kam se jen podíváme. Během dne s nimi přijdeme nespočetněkrát do kontaktu a ani si to nemusíme uvědomovat. Stačí se jen zamyslet, kde všude můžeme kupříkladu nalézt mikrokontrolér. Může se jednat například o zařízení v domácnostech, jako jsou např. pračka, mikrovlnná trouba, televize. Elektronická zařízení ovšem nejsou jen součástí našich domácností, uvažme např. auta, letadla, lodě. Nová auta dnes obsahují neuvěřitelný počet mikrokontrolérů, něco okolo 200 až 300. Informační technologie začaly výrazným způsobem prostupovat a měnit náš život a nebude nejspíš ani dlouho trvat, až člověk začne vylepšovat své tělo pomocí různých technologických objevů.

Veškeré informační technologie jsou ve většině případů nějakým způsobem propojeny a tak jsou vytvářeny komunikační kanály a sítě koncových zařízení, které spolu navzájem komunikují. Snad největší a nejznámější takovou sítí je Internet. Jedná se o milióny navzájem propojených počítačů, které mezi sebou sdílí určité informace. Vezměme do úvahy třeba komplexní bankovní systém. Na jedné straně je uživatel a na druhé banka. Banka se stará o účty svých zákazníků. Zákazník může přistupovat ke svému kontu mnoha způsoby. Jedním ze základních přístupů je platební karta. Zákazník může platit kartou v obchodech, může pomocí karty vybírat peníze z bankomatů a některé typy karet umožňují dokonce i platbu na internetu. K obsahu konta samozřejmě lze přistupovat více způsoby: na přepážce v bance, pomocí internetového a telefonního bankovnictví, pomocí bankomatů v zahraničí atd. Všechny koncové body samozřejmě musí být navzájem určitým způsobem propojeny s bankou.

Asi si lze představit o jak složitý komunikační systém se musí jednat. Momentálně žijeme v době, kdy mezi námi existují lidé se spoustou negativních vlastností. Rádi získávají nelegálním způsobem peníze, které jim nepatří. Mezi těmito lidmi se objevují i inteligentní a nadaní jedinci, kteří hledají v různých informačních technologiích slabiny nebo využívají sociálního inženýrství ku svému finančnímu prospěchu. Vraťme se zpátky k příkladu s bankovním systémem. Je opravdu skvělé, pokud máme vždy a za každých okolností peníze na dosah ruky, ale na druhou stranu se tímto zvyšuje komplexnost systému a s tím i počet komunikačních cest. Zvětšující se počet cest znamená i větší možnosti pro odposlech dat mezi bankou a daným koncovým zařízením. Zajištění bezpečného přenosu je jedním ze způsobů, kterým lze útočníkovi zabránit v odposlechu dat. Samozřejmě bankovní systém byl jen ukázkový systém, zajistit bezpečný přenos dat je vyžadován stále ve více oblastech, nejen ve finanční sféře.

Jedním ze způsobů, kterým můžeme zajistit bezpečný přenos dat je šifrování těchto dat. Šifrování dat se nemusí nutně týkat jen přenosu dat, ale lze jej použít i k šifrování třeba jen souborů na osobním počítači. V dnešní době dochází ke zrychlování zpracování informace ve všech možných oblastech a také ke zrychlování i přenosů těchto informací. S tím právě souvisí i nutnost dostatečným způsobem zrychlit zabezpečení těchto informací, tedy urychlit šifrování. Ve většině případů se šifrují data softwarově, tedy klasickým procesorem. Ovšem tento přístup začíná být pomalý a nedostačující pro požadované rychlosti. Proto se začínají navrhovat hardwarové realizace různých typů šifer.

V této diplomové práci se pokusíme navrhnout obvod, který bude implementovat algoritmus AES. Tento obvod bude připraven pro případné použití v hradlovém poli FPGA, které je součástí platformy FITKit. Obvod bude navržen s ohledem na dostupné zdroje, které toto hradlové pole nabízí. Prvním krokem je samotný návrh obvodu. Tento návrh je potřeba v dalším kroku popsat v jazyce VHDL. Na základě tohoto popisu pak můžeme pomocí simulace verifikovat, zda byl obvod navržen správným způsobem a zda obvod zajišťuje požadovanou funkcionalitu. Dále pak následuje syntéza verifikovaného obvodu a zjištění počtu zdrojů, které bude náš navržený obvod potřebovat pro realizaci v hradlovém poli FPGA. Abychom mohli dále provést srovnání našeho hardwarového řešení vůči klasickému softwarovému šifrování, bude za tímto účelem vytvořena softwarová implementace. Ta bude opět vytvořena pro platformu FITKit, tentokrát ovšem bude realizována pomocí mikrokontroléru, který je také součástí této platformy. Na konci diplomové práce bude provedena analýza naměřených hodnot a rozbor vlastností jednotlivých implementací.

Diplomová práce je členěna do několika kapitol. Ve druhé kapitole je provedeno seznámení se základy kryptografie. Jedná se teoretickou část, ve které jsou popsány kryptografické metody, jež se používaly v minulosti a taky ty, které se používají v současné době. Třetí kapitola popisuje platformu FITKit, především způsob, jakým se nahrává konfigurace do hradlového pole FPGA a jakým způsobem se provádí nahraní programu do mikrokontroléru. Dále je v této kapitole popsán propojovací systém FITKitu. Ve čtvrté kapitole je popsán návrh obvodu spolu s detailním popisem jeho jednotlivých částí. Pátá kapitola je věnována softwarové implementaci. V poslední šesté kapitole je provedeno celkové srovnání vlastností HW a SW realizace. Zkoumáno je především zrychlení a vlastnosti obou implementací.

Kapitola 2

2 Základy kryptografie a AES algoritmus

2.1 Historie kryptografie

Na úvod této kapitoly musíme říci, že kryptografie není vynález poslední doby a ani doby nedávné. První známé formy šifrování se objevily již v antickém Řecku, kdy bylo poprvé použito skutečného zakódování textu za pomoci konkrétního algoritmu. Jedna z nejdůležitějších historických zpráv, která měla zajistit existenci západní civilizace byla, dle historických pramenů, předána také tajným způsobem. Mělo se jednat o zprávu, která nesla důležitou informaci a měla pomoci Řekům v boji proti Peršanům. Jednalo se o termín, kdy král Xerxes vytáhne s armádou proti Řekům. Tuto informaci zjistil Demaratus, syn Aristona a rozhodl se že o tomto faktu informuje své krajany. V této době se používaly voskové destičky, kdy byl vosk nanesen na dřevěnou desku. Demaratus vosk z desky seškrábal a zprávu vyryl přímo na dřevěnou desku a nakonec ji zalil opět voskem, aby při náhodné kontrole vypadaly desky jako zatím nepoužité. Když se zpráva dostala ke svému příjemci, manželce krále Leonidase, bylo tajemství odhaleno a následovaly slavné bitvy u Thermopyl, Salaminy a Platea. Postup Peršanů do Evropy byl nakonec zastaven a v důsledku toho se mohla rozvíjet západní civilizace.

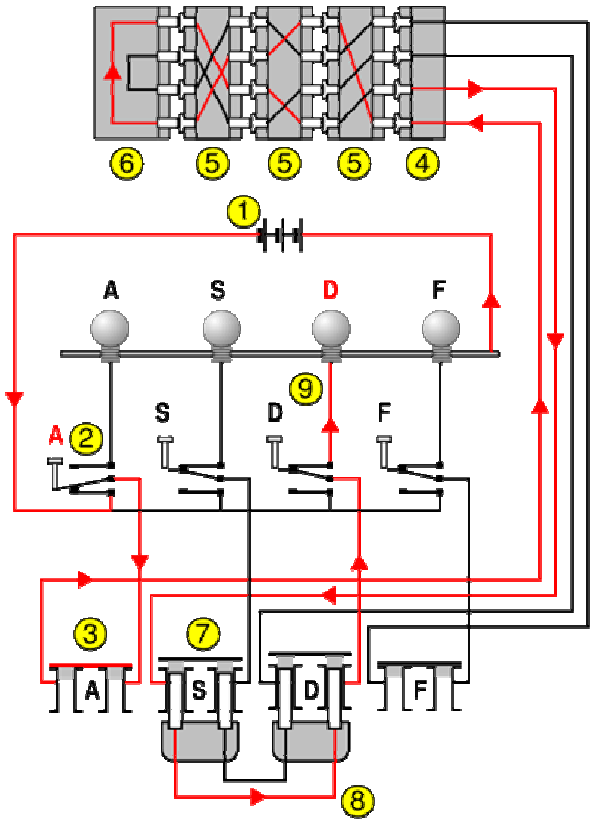
Dalším historickým příkladem kryptografie může být Caesarova šifra, která vlastně představuje první známé algoritmické šifrování. Julius Caeasar šifroval své zprávy tím způsobem, že nahradil každé písmeno třetím následujícím písmenem v abecedě. Slovo „*caesar*“ tedy zašifrujeme jako „*fdhvdu*“. Slabinou tohoto algoritmu je fakt, že každý, kdo zná algoritmus, může danou zprávu dešifrovat. Jedním z menších vylepšení této šifry je, že odesílatel a přijímatel si domluví tajný klíč, kterým je číslo od 1 do 25 a toto číslo znamená o kolik písmen se bude daný znak v abecedě posouvat. I přes toto malé vylepšení je šifra stále slabá a zašifrovaný text můžeme rozluštit buď hrubou silou (máme 26 možností) anebo frekvenční analýzou jednotlivých znaků v textu. Vylepšením caesarovy šifry je monoalfabetická substituční šifra, kdy se nahrazují jednotlivá písmena textu pomocí klíče, jímž je permutace všech 26 písmen abecedy. To znamená, že každé písmeno abecedy se posouvá o jiný počet znaků. Počet všech možných klíčů u této šifry je $26! = 4 * 10^{26}$. Tato šifra byla nerozluštitelná během celého prvního století našeho letopočtu.

Příklad klíče monoalfabetické šifry: *ACJOUZFDKPVIELQWRGMSXBHNTY*
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Po Řecích převzali šifrování i Římané a kryptografie se stala velmi oblíbenou disciplínou, která byla používaná především ke státním a vojenským účelům. Kryptografie následně procházela dalším

vývojem po celá staletí. Velkým skokem ve vývoji kryptografie byla první světová válka, kdy došlo k rozvoji šifrovací problematiky především díky rozšíření bezdrátové telegrafie.

Druhá světová válka znamenala další výrazný posun v této oblasti. V Německu bylo sestrojeno snad nejznámější šifrovací zařízení všech dob a jeho jméno bylo Enigma. Enigma byla vytvořena v roce 1923 pro komerční účely. Později začala být používána státními a vojenskými složkami a později byla speciálně upravena pro Wehrmacht. Enigma je kombinací mechanických a elektrických systémů. Mechanický systém je tvořen klávesnicí, sadou rotujících disků, které se nazývali rotory a byly seřazeny za sebou na jedné hřídeli. Dále pak krokujícím systémem, který slouží k otáčení jednoho nebo více rotorů s každým stiskem klávesy. Tyto mechanické součásti tvoří součást elektrického obvodu. Při stisku klávesy je obvodu sepnut a je vysvícen jeden znak na displeji. Způsob, jakým enigma funguje je zobrazeno na obr. 2.1.



Obr. 2.1 – Schéma funkce Enigmy

Za účelem vysvětlení funkčnosti enigmy jsou zobrazeny na obr. 2.1 pouze 4 prvky. Ve skutečnosti obsahuje enigma 26 lamp (znaky jsou podsvíceny právě těmito lampami), kláves, kolíků a zástrček. Elektrický proud z baterie (1) protéká spínačem (2) do propojovací desky (3 – angl. plugboard). Propojovací deska dovoluje změnu propojení mezi klávesnicí (2) a fixním vstupem do rotorů (4).

Dále signál postupuje skrze 3 rotory (5) a vstupuje na reflektor (6). Reflektor pak vrací proud jinou cestou opět skrze rotory (5) a fixní vstup rotorů (4) a pokračuje na výstup ‚S‘ (7), který je propojen s výstupem ‚D‘ (9), který již toto písmeno vysvítí na displeji.

V roce 1948 a 1949 byly otištěny dva články (teorie informace) Claudie Elwood Shannona, jež započaly novou éru moderní kryptografie a kryptoanalýzy. Do poloviny sedmdesátých let se používali pro šifrování pouze symetrické algoritmy. Symetrické algoritmy používají pro šifrování a dešifrování stejného klíče. V roce 1976 publikovali Whitfield Diffie, Martin Hellman a Ralph Merkle článek o nových možnostech kryptografie. Tím byly položeny základy asymetrických šifrovacích algoritmů, neboli šifrování veřejným klíčem. Dva roky na to (1978) vznikl první asymetrický šifrovací systém a to systém RSA, jež se používá dodnes. Ve stejné době (1977) byl v USA vytvořen symetrický algoritmus DES (Data Encryption Standard) a následně přijat v USA za veřejný standard pro zabezpečení státních informací.

Dalším důležitým bodem v historii je rok 1980, kdy se v Santa Barbaře uskutečnila první velká kryptografická konference s názvem „Crypto“ a tato konference se koná dodnes. Postupem času se začaly objevovat nové algoritmy, jako např. BlowFish, FEAL, IDEA atd. Některé z těchto algoritmů byly prolomeny. V roce 1993 navrhl a popsal Wiener zařízení v ceně 1 milionů dolarů, které dokáže najít klíč algoritmu DES během 7 hodin. Proto byl algoritmus DES nahrazen algoritmem 3-DES, který je praktický trojnásobným algoritmem DES. V roce 1999 došlo k rozbití šifry RSA s délkou klíče 512 bitů.

2.2 Základní pojmy a terminologie

Nejdříve je potřeba objasnit základní pojmy z kryptografie a zavést označení, které budeme nadále používat. Jelikož je kryptografie rychle se rozvíjející oblastí počítačové vědy, neexistuje zde žádná jednotná terminologie a dokonce i definice základních pojmů se často velice liší. Z tohoto důvodu v dalším textu uvedeme spíše objasnění pojmu namísto jeho formální definice.

Slovo *kryptografie* pochází z řečtiny a vzniklo složením dvou slov *kryptós* a *gráphein*, což znamená „skrytý“ a „psát“. Kryptografie neboli šifrování je tedy nauka o metodách utajování smyslu zpráv převodem do podoby, která je čitelná jen se speciální znalostí. K tomu používá určité transformace dat, které jsou založeny na matematických metodách. Použitou transformací bývá nejčastěji šifrovací algoritmus. Opakem kryptografie je *kryptoanalýza*, která se naopak snaží odhalit obsah utajené zprávy, většinou bez znalostí klíče. Zavedeme následující označení:

- Nechť $\boldsymbol{A}$ je konečná množina prvků. Tyto prvky nazveme symboly a množinu abecedou. Pro naše účely budeme volit pouze binární abecedu, tj. $\boldsymbol{A} = \{0, 1\}$, neboť libovolná zpráva může být překódována do této abecedy.

- Nechť ***M*** je množina nazývaná *prostor zpráv* (message space). Množinu ***M*** tvoří řetězce symbolů z abecedy ***A***. Prvek množiny ***M*** nazveme *otevřený text* a označíme jej PT (angl. Plain Text).
- Nechť ***C*** je množina nazývaná prostor šifer (ciphertext space). Množina ***C*** sestává z řetězců symbolů abecedy ***A***, které se obecně mohou lišit od abecedy použité v ***M***. Prvek abecedy ***C*** nazveme *šifrovaný text* a označíme jej CT (angl. Cipher Text).
- Nechť ***K*** je množina nazývaná prostor klíčů (key space). Prvek této množiny nazveme jednoduše *klíč* a označíme k.
- Každý prvek $e \in \boldsymbol{K}$ jednoznačným způsobem definuje bijekci $\mathbf{E}_e : \boldsymbol{M} \rightarrow \boldsymbol{C}$, kterou nazveme *šifrovací funkce* a proces zobrazení $\mathbf{E}_e(m)$, kde $m \in \boldsymbol{M}$, nazveme *zašifrování.*
- Každý prvek $d \in \boldsymbol{K}$ jednoznačným způsobem definuje bijekci $\mathbf{D}_d : \boldsymbol{C} \rightarrow \boldsymbol{M}$, kterou nazveme *dešifrovací funkce* proces zobrazení $\mathbf{D}_d(c)$, kde $c \in \boldsymbol{C}$, nazveme *dešifrování.*
- *Kryptografický systém*, někdy taky označovaný jako *šifra*, je tvořen množinou $\{\mathbf{E}_e : e \in \boldsymbol{K}\}$ šifrovacích funkcí a k ní odpovídající množinou $\{\mathbf{D}_d : d \in \boldsymbol{K}\}$ dešifrovacích funkcí spolu s vlastností , že pro každý prvek $e \in \boldsymbol{K}$ existuje odpovídající prvek $d \in \boldsymbol{K}$ takový, že pro libovolné $m \in \boldsymbol{M}$ platí $\mathbf{D}_d(\mathbf{E}_e(m)) = m$.

U kryptografie rozlišujeme základní rozdělení šifer na symetrické šifry a asymetrické šifry. V případě, že pro danou dvojici (*e, d*) jsou *e* a *d* navzájem odvoditelné (přesněji řečeno je to výpočetně jednoduché), mluvíme o symetrickém šifrování (někdy také označováno jako šifrování s tajným klíčem, neboli tradiční šifrování). V případě, že je velice obtížné (prakticky časově nemožné) odvodit z jednoho klíče druhý, mluvíme v takovém případě o asymetrickém šifrování (šifrování s veřejným klíčem). Asymetrické šifrování se ovšem pro účely této diplomové práce nehodí. Zde se budeme zabývat pouze symetrickými šiframi pro něž platí e = d, tedy že pro šifrování a dešifrování použijeme stejného klíče.

Rozlišujeme několik druhů symetrických šifer, z nich nejdůležitější jsou proudové a blokové šifry:

- **Proudové šifry –** zpracovávají nešifrovaný text společně s proudem hesla (keystream) bit po bitu nebo popř. byte po byte pomocí nějaké jednoduché operace, např. XOR operací. Proud hesla je v tomto případě buď přímo šifrovací klíč (tzv. Vernamova šifra), nebo je z něj nějakým způsobem odvozován.
- **Blokové šifry** – na rozdíl od proudových šifer zpracovávají nešifrovaný text po řetězcích pevné délky n, které se nazývají bloky. Příslušná šifrovací nebo dešifrovací operace se potom provádí vždy nad celým blokem dané délky n.

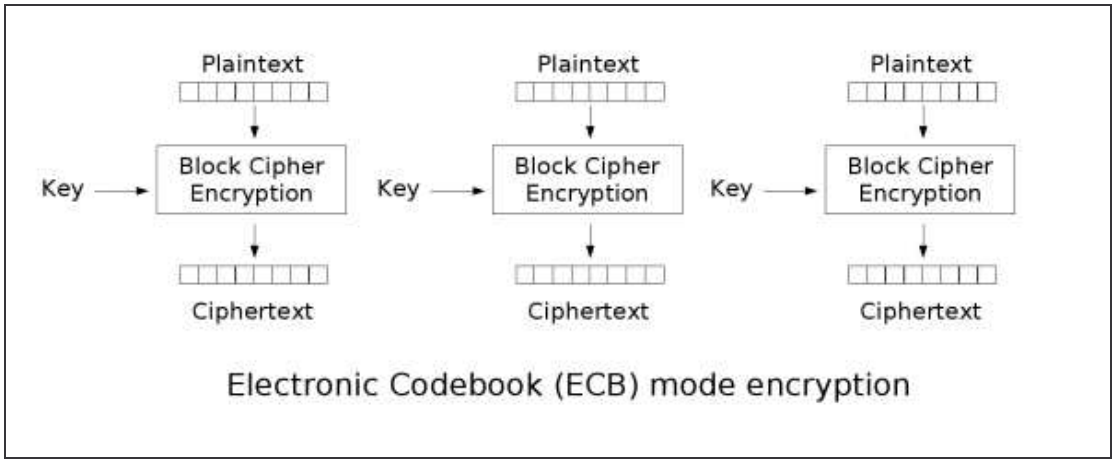
V podstatě se dá říci, že proudové šifry jsou blokové šifry s délkou bloku n = 1. Blokové šifry jsou nejrozšířenější symetrické techniky a právě AES je jednou z těchto symetrických šifer. Popisu AES jsou věnovány následující kapitoly.

2.3 Módy blokových šifer

Způsob, jakým se postupuje při šifrování zprávy (označme ji M), jejíž délka je větší než velikost bloku je následující: M rozdělíme na bloky o délce n bitů, tedy $M = M_1M_2M_3\ldots$, a každý z bloků M_i šifrujeme podle daného módu na blok C_i. Zřetězením bloků C_i pak dostaneme zašifrovaný text C, tedy $C = C_1C_2C_3$. Pokud délka zprávy M není dělitelná délkou n beze zbytku, je potřeba provést před šifrováním roztáhnutí posledního bloku na potřebnou délku – tzv. *padding*. Při dešifrování takovéto zprávy je pak samozřejmě potřeba tento blok odpovídajícím způsobem „zkrátit“. Blokové šifry se používají v těchto základních módech:

2.3.1 Electronic Code Book (ECB)

V tomto módu se každý blok M_i šifruje stejným klíčem bez závislosti na ostatních blocích, tedy $C_i = E_k(M_i)$. Dešifrování se provádí stejným způsobem, tedy samostatně pro každý blok $M_i = D_k(C_i)$. Je zřejmé, že při šifrování dvou stejných bloků $M_i = M_j$ dostaneme také stejné zašifrované bloky $C_i = C_j$. Toho lze využít při budování tzv. *kódové knihy*, což je vlastně obrovský slovník bloků nezašifrovaného textu a jim odpovídajících bloků zašifrovaného textu pro daný klíč k (avšak bez jeho znalosti!). Z tohoto důvodu se režim ECB používá jen velmi omezeně, nejčastěji pro krátké zprávy, a s maximální možnou délkou bloku, neboť na té závisí i celková velikost kódové knihy. Ilustrace viz obr. 2.2:



Obr. 2.2 – Bloková šifra v režimu ECB

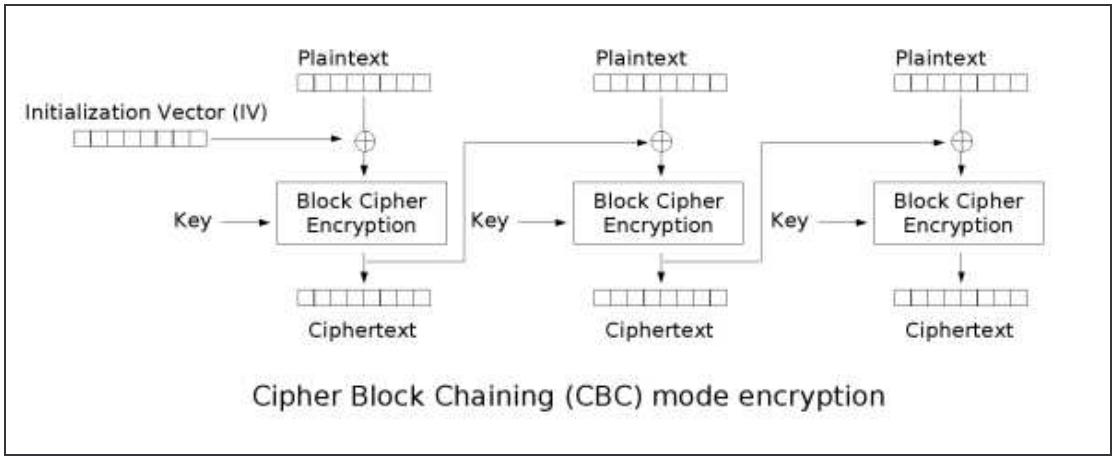
2.3.2 Cipher Block Chaining (CBC)

Tento mód odstraňuje nedostatky ECB zavedením kontextové závislosti do šifry. Pro tyto potřeby je zaveden *inicializační vektor* (IV), který je veřejný, nejčastěji se generuje náhodně a předává se ještě před šifrovanými daty v otevřené podobě. Pro i > 1 a operaci XOR značenou jako ⊕ je šifrování v CBC módu dáno vztahy:

$$C_1 = E_k(M_1 \oplus IV),\ C_i = E_k(M_i \oplus C_{i-1})$$

a dešifrování podobně:

$$M_1 = D_k(C_1) \oplus IV,\ M_i = D_k(C_i) \oplus C_{i-1}$$



Obr. 2.3 – Šifrování a dešifrování v módu CBC

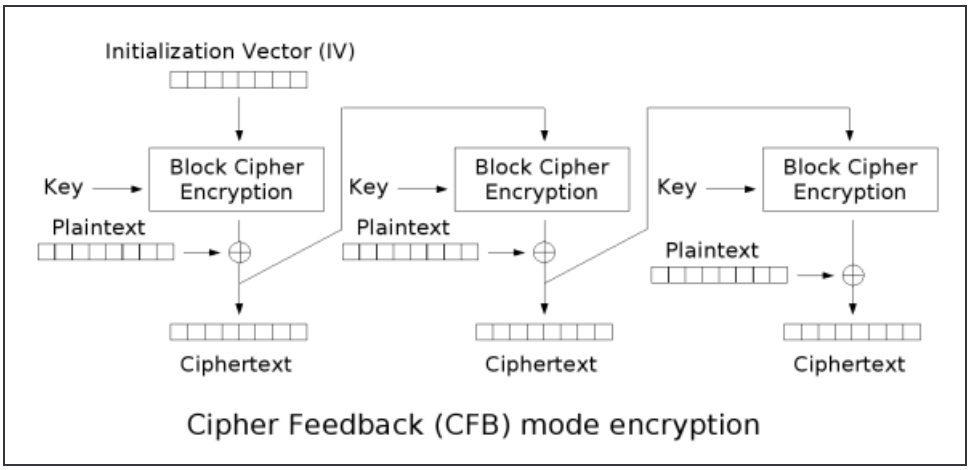
U šifry v módu CBC je tedy zajištěno, že dva stejné bloky vstupních dat budou zašifrovány odlišně. Navíc náhodný výběr inicializačního vektoru nám zajišťuje, že i dvě stejné zprávy budou zašifrovány zcela odlišně. Mód CBC je také málo náchylný k chybám, neboť pokud je na vstupu šifry chybný blok C_i, pak tato chyba ovlivní při dešifrování nejvýše dva následující bloky, tedy M_i a M_{i+1}. Z těchto důvodů je CBC je nejpoužívanější mód u blokových šifer. Ilustrace viz obrázek 2.3.

2.3.3 Cipher Feedback (CFB)

Šifra v módu CFB je podobná CBC módu. Blok šifrovaného textu je získán zašifrováním minulého bloku šifrovaného textu (typicky posledních 64b) a přičtením části vzniklého zašifrovaného textu (obvykle v délce 1 byte) k stejně dlouhému bloku otevřeného textu. Ilustrace viz obr. 2.4. Zašifrování a dešifrování v tomto módu je dáno následujícími vztahy:

Šifrování: $C_0 = IV,\ C_i = E_k(C_{i-1}) \oplus M_i$

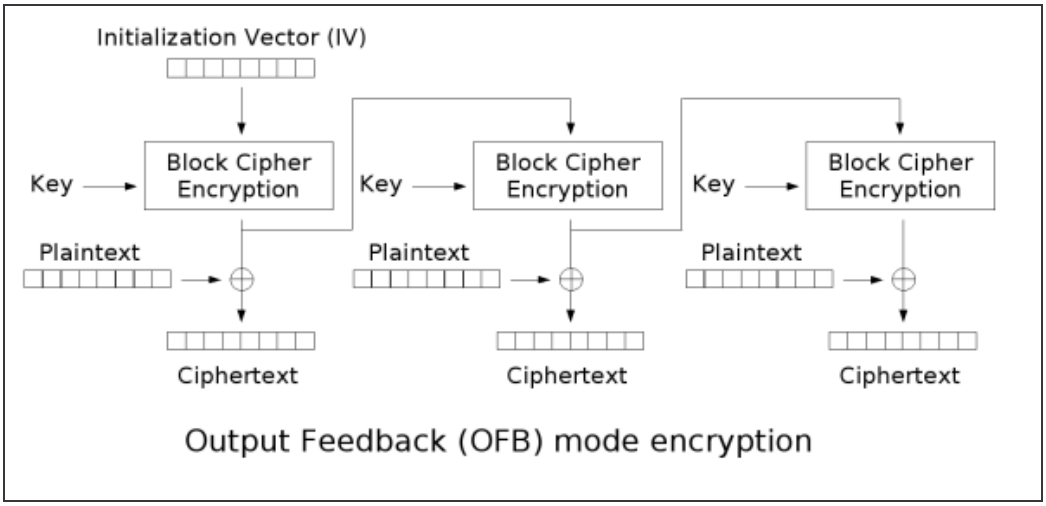
Dešifrování: $M_i = E_k(C_{i-1}) \oplus C_i$



Obr. 2.4 – Bloková šifra v režimu CFB

2.3.4 Output FeedBack (OFB)

Tento režim převádí blokovou šifru na streamovou synchronní šifru: šifra generuje binární posloupnosti, které se nazývají keystreams (neexistuje český překlad). Tyto posloupnosti jsou pak „xorovány“ (provedena operace XOR) s datovými bloky otevřeného textu a tím jsou vytvářeny bloky zašifrovaného textu. Keystream je dále pak použit jako vstupní blok dat v další iteraci šifry. Vše nejlépe ilustruje obr. 2.5 :



Obr. 2.5 - Bloková šifra v režimu OFB

2.4 Historické kryptografické metody

2.4.1 Steganografie

Slovo steganografie pochází z řečtiny a vzniklo složením slov *steganós* a *graphéin*, jež v překladu znamenají „schovaný“ a „psát“. Steganografie je věda zabývající se ukrytím komunikace prostřednictvím ukrytí zprávy. Zpráva je ukryta takovým způsobem, aby si pozorovatel neuvědomil, že komunikace vůbec probíhá. Zachycení zprávy se rovná prolomení steganografie. V antickém Řecku se používaly např. k přenosu zpráv dřevěné destičky zalité voskem, do kterého bylo vyryto písmo. Jeden z prvních zaznamenaných případů steganografie popisuje vyrytí zprávy přímo do dřevěné destičky, které byla následně zalita voskem a vypadala tedy jako nepopsaná. Jindy byla třeba zpráva vytetována na oholenou hlavu otroka a následně se nechala zarůst vlasy. Přečíst ji bylo možné až po dalším oholení. Za druhé světové války se používala například technika mikroteček. Šlo o malé nezřetelné tečky na pásku filmu, které teprve při prohlédnutí pod mikroskopem obsahovaly drobné písmo. Dalšími příklady steganografie mohou být neviditelné inkousty, vodoznaky (angl. watermarks) uvnitř obrazu (např. bankovky), nebo ukrytí textu uvnitř jiného textu – viz obrázek 2.6.

U moderní steganografie se zpráva většinou skládá z posloupnosti bitů, z nichž některé bity jsou informační a ostatní nemají žádnou informační hodnotu, jedná se o tzv. „vycpávku“. Pomocí klíče oprávněný příjemce zjistí, které bity se mají vypustit a které představují skrytou informaci. Příkladem může být např. ukrytí informace do digitálního obrazu, kdy informační bit představuje nejméně významný bit každého pixelu. Dalším příkladem může být zakódování zprávy na místo nepodstatného šumu v souborech se zvuky, videem a podobně.

Dear George, 3rd March
Greetings to all at Oxford. Many thanks for your
letter and for the Summer examination package.
All Entry Forms and Fees Forms should be ready
for final dispatch to the Syndicate by Friday
20th or at the very least, I'm told, by the 21st.
Admin has improved here, though there's room
for improvement still; just give us all two or three
more years and we'll really show you! Please
don't let these wretched 16+ proposals destroy
your basic O and A pattern. Certainly this
sort of change, if implemented immediately,
would bring chaos.

Obr. 2.6 – Příklad steganografické šifry

2.4.2 Substituční šifry

Substituční šifra obecně spočívá v nahrazení každého znaku zprávy jiným znakem podle určitého pravidla. Aby příjemce získal otevřený text, musí na zašifrovaný text použít inverzní substituci. V klasické kryptografii existují čtyři základní typy substitučních šifer:

1) **Monoalfabetická substituční šifra -** jedná se o nejjednodušší druh substitučních šifer, kde se podle konkrétní substituční tabulky (obměněné abecedy) nahrazuje písmeno v otevřeném textu za písmeno v zašifrované podobě. Z toho je patrné, že šifra zachovává frekvenci výskytů písmen i v zašifrované podobě. To znamená, že pokud v českých textech převažuje písmeno „A“ a toto písmeno se v zašifrovaném textu změní na „F“, bude potom v zašifrovaném textu písmeno „F“ také převažovat. Tento poznatek velice zjednodušuje kryptoanalýzu, která v tomto případě využívá frekvenční analýzu písmen zašifrovaného textu. U této šifry lze použít klíče následujícím způsobem (viz tabulka 2.1):

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
V	**E**	**S**	**L**	**O**	A	B	C	D	F	G	H	I	J	K	M	N	P	Q	R	T	U	W	X	Y	Z

Tab. 2.1 – příklad monoalfabetické šifry s klíčem „VESLO“

2) **Homofonní substituční šifra** – podobá se monoalfabetické substituční šifře, avšak jeden znak otevřeného textu může být nahrazen jedním z několika možných znaků šifrovaného textu. Znak „A“ by mohl být nahrazen např. 5, 10, 13 nebo 25, „B“ např. 6 nebo 15 atd. Počet znaků zašifrovaného textu pro jeden znak otevřeného textu se může lišit. Oproti předchozímu typu šifry nelze použít stejný postup kryptoanalýzy, neboť dochází tomto případě ke zploštění frekvenční analýzy. Příklad homofonní šifry je na obr. 2.7.

Příklad homofonní šifry

- **Zobrazuje písmena na číslice 00, ..., 99**

Plaintext	Ciphertext
A	17 19 34 41 56 60 67 83
I	08 22 53 65 88 90
L	03 44 76
N	02 09 15 27 32 40 59
O	01 11 23 28 42 54 70 80
P	33 91
T	05 10 20 29 45 58 64 78 99

- **Zpráva:** P L A I N P I L O T
- **Šifrovaný text:** 91 44 56 65 59 33 08 76 28 78

Obr. 2.7 – Příklad homofonní šifry

3) **Polygramová substituční šifra -** jedná se o takovou šifru, kde šifrování probíhá mezi skupinami znaků. Skupina „AA“ může být nahrazena skupinou „JH“, skupina „AB“ skupinou „DK“ atd.

4) **Polyalfabetická substituční šifra** – skládá se z několika jednoduchých šifer, které se postupně pro jednotlivé znaky otevřeného textu střídají. Dvě nejznámější polyalfabetické substituční šifry jsou Vigenerova šifra a Vernamova šifra:

Vigenerova šifra: vytvořena Blaise de Vigenerem v roce 1550 a byla nerozlomitelná po dobu přibližně 300 let. Tato šifra využívá caesorova principu s rozdílnými posuvy pro jednotlivé znaky, aby se zakryla frekvence znaků. Znaky klíče definují posuv pro jednotlivá písmena otevřeného textu. Klíč je periodicky opakován, aby obsáhl celou délku šifrovaného textu. Základem šifrování je Vigenerova tabulka (tabulka 2.2) se dvěma abecedami. Ve vodorovné abecedě nalezneme písmeno zprávy a ve svislé pak odpovídající písmeno hesla. V průsečíku těchto písmen pak nalezneme daný zašifrovaný znak.

Příklad: Otevřený text: „vigenerescipher“
Klíč: „keykeykeykeykey“
Šifrovaný text: „fmeorcbiqmmnrip“

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Tab. 2.2 – Vigenerova tabulka pro šifrování

Vernamova šifra: nebo také jednorázová tabulková šifra (angl. *one-time pad*) je jednoduchý šifrovací postup patentovaný v roce 1917 Gilbertem Vernamem. Spočívá v posunu každého znaku zprávy o náhodně zvolený počet míst v abecedě. To se prakticky rovná náhradě zcela náhodným písmenem a na tomto faktu je založen důkaz, že Vernamova šifra je v principu nerozluštitelná. Aby byla šifra nerozluštitelná musí být splněny následující tři podmínky:

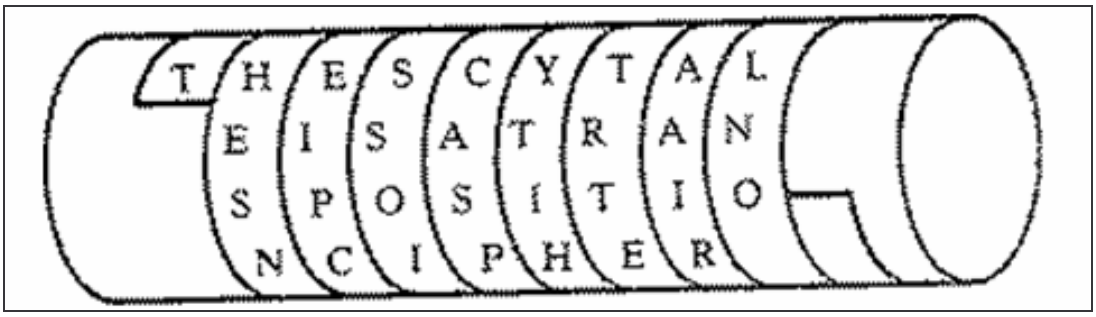
- **Klíč musí mít stejnou délku jako všechny šifrované zprávy** – kratší klíč v principu umožňuje útok hrubou silou
- **Klíč musí být absolutně náhodný**. Nemohou být použity počítačové generátory pseudonáhodných čísel, neboť jejich činnost lze poměrně snadno předvídat. Nejvhodnější je použití fyzikálních metod, např. použití tepelného šumu nebo kvantových procesů, neboť jejich základní vlastností je náhodnost.
- **Klíč se nesmí nikdy použít znova**. Tato podmínka je důsledkem předchozí, neboť opakovaný klíč není náhodný.

Zde popsané zacházení s klíčem je v praxi velmi obtížné. Dlouhý náhodný klíč si člověk nemůže zapamatovat, a proto musí být nějakým způsobem zaznamenán. Jeho generování taky není jednoduché, neboť musí být zajištěno, že klíč zná pouze odesílatel a příjemce zprávy a nikdo jiný. Komunikující strany se tedy musí předem dohodnout na klíči nějakým bezpečným způsobem a ihned po odeslání první zprávy klíč zničit. Ovšem zde vzniká známý filozofický problém slepice a vejce. Jestli bychom chtěli odeslat 2 MB tajných dat, musíme předem bezpečně odeslat 2 MB dat (klíč). I přes svou sílu se Vernamova šifra používala jen výjimečně, např. pro krytí horké linky mezi Moskvou a Washingtonem během studené války. Praktický problém distribuce klíče ovšem není v dnešní době neřešitelný. S využitím poznatků kvantové fyziky můžeme v dnešní době provést absolutně bezpečnou výměnu klíče na dálku. Díky těmto poznatkům taky vznikla kvantová kryptografie, která poskytuje zprávám nejvyšší možnou míru zabezpečení, což je dávný sen všech kryptografů všech dob.

2.4.3 Transpoziční šifry

Transpozice neboli přesmyčka spočívá ve změně pořadí znaků otevřeného textu podle určitého pravidla. Uvedeme si 3 základní typy transpozičních šifer:

1) První takováto používaná šifra je **Scytale**, která se používala ve starém Řecku. Na tyči je namotán proužek papíru. Zpráva se pak zapisuje na tento omotaný proužek po řádcích. Po odmotání proužku jsou na něm zdánlivě náhodné znaky. Tato šifra není příliš bezpečná – klíčem je totiž pouze průměr tyče. Pro ilustraci viz obr. 2.8.



Obr. 2.8 – Scytale – antická transpoziční šifra

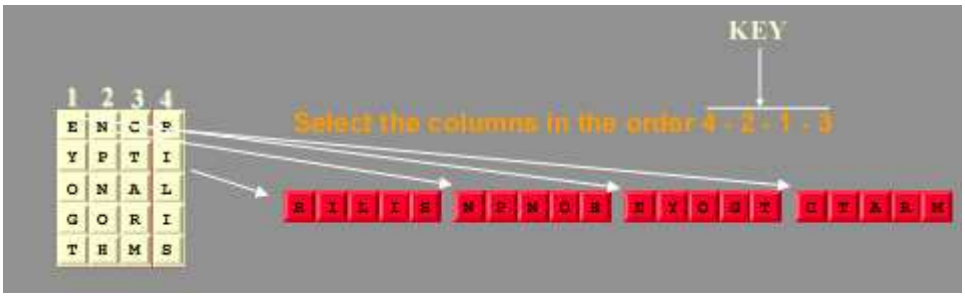
2) Šifra **Rail Fence** – jedná se o transpoziční šifru, která získala své jméno podle způsobu, jakým se šifrují data. Otevřený text se vepisuje do mřížky v diagonále seshora dolů a od posledního řádku mřížky se data začnou vepisovat opět v diagonále, ale tentokrát odzdola nahoru. Vše ilustruje následující příklad. Zašifrované data se pak získají čtením vyplněných buněk po řádcích. Rail Fence šifra není příliš silná, neboť počet praktických klíčů je velmi malý a člověk ji může rozluštit přímo ručním rozepsáním na papíře.

Příklad: Chceme zašifrovat text „WE ARE DISCOVERED FLEE AT ONCE“

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Zašifrovaný text pak přečteme po řádcích: „WECRL TEERD SOEEF EAOCA IVDEN“.

3) **Sloupcová transpozice –** otevřený text se zapíše po řádcích a šifrovaný text se vytvoří tak, že se čte po sloupcích v daném pořadí (klíč je pořadím). Příklad viz následující obrázek 2.9 - otevřený text „ENCRYPTIONALGORITHMS“ je vepsán do matice 5x4. Klíčem je pořadí 4 - 2 -1 -3.



Obr. 2.9 – Sloupcová transpozice

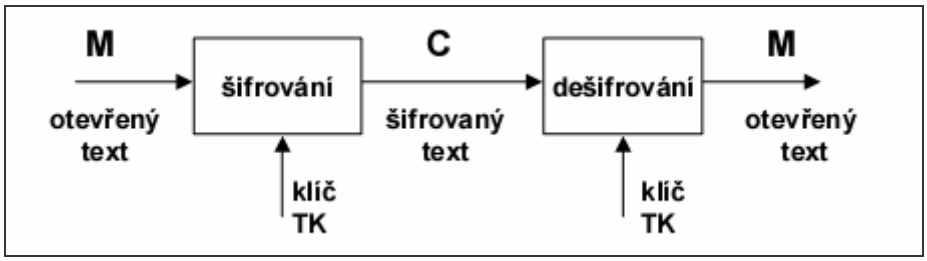
2.4.4 Kombinované šifry

Někdy také označovány jako složené šifry (angl. product ciphers). Jedna šifra, ať substituční nebo transpoziční se nejevily dostatečně bezpečné. Tuto nevýhodu, popř. i jiné slabiny těchto šifer je možné alespoň částečně eliminovat kombinací těchto šifer. Ovšem je potřeba vzít v úvahu, že kombinací stejných šifer zesložitění luštění nedocílíme. Např. následné použití dvou

monoalfabetických substitučních šifer je pro potenciálního útočníka to samé jako použití jedné této šifry ovšem s jiným klíčem. K vytvoření nové a bezpečnější šifry je potřeba zkombinovat transpoziční a substituční šifru. Pro ruční realizaci je toto dosti komplikované, ale v minulosti se tyto kombinace používaly. Pro mechanický stroj je realizace taky dosti obtížná. Princip skládaných šifer se většinou používá v moderní kryptografii.

2.5 Moderní symetrické šifry

Symetrická kryptografie (též taky nazývaná „konvenční kryptografie“ či angl. conventional cryptography) používá stejný klíč, jak pro zakódování, tak pro dekódování zprávy. Ilustrace viz obrázek 2.10.



Obr. 2.10 – Průběh symetrického šifrování a dešifrování

Vstupem je tedy nějaký tajný text ze stanovené abecedy a klíč. Šifrovací funkcí se za pomocí klíče tajný text převede na kód, který může být odeslán příjemci zprávy. Příjemce pak použije dešifrovací funkci se stejným klíčem a tím získá původní tajný text. Důležité je, že pro šifrování musí mít příjemce k dispozici stejný klíč, jakým byl text zakódován. Je tedy třeba zajistit bezpečný způsob doručení klíče, aby se tento klíč nedostal do nepovolaných rukou. Pro šifrování se používají funkce, u kterých platí, že při znalosti vstupního a zakódovaného textu je velmi obtížné vygenerovat klíč, ač vlastní kódovaní a dekódovaní pomocí tohoto klíče je rychlá záležitost. Obtížnost případného zjištění klíče záleží zejména na vlastní délce klíče. Šifrovaná zpráva musí odolat útoku hrubou silou, který přepokládá vyzkoušení všech možných klíčů. Pokud je klíč délky 8 bitů, pak existuje 2^8 (256) možných klíčů. Před několika roky se podařilo rozbít šifru DES s 56 bitovým klíčem. V dnešní době se používají běžně klíče o velikosti 128 bitů, kdy zjištění takového klíče by teoreticky trvalo asi 10^{39} let. To znamená, že i při současném vývoji informačních technologií budou klíče s touto délkou ještě nějakou chvíli použitelné. Mezi symetrické metody patří např. již zmíněný algoritmus DES (délka klíče 56 bitů), který si v této kapitole ještě detailně popíšeme, či jeho nástupce 3DES (násobný DES – klíč 3x delší než u obyčejného DES, tj. 168 bitů) anebo další, jako jsou BlowFish (proměnná délka klíče až 256 bitů) a IDEA (délka klíče 128 bitů), CIFER (délka klíče 16 bitů) atd.

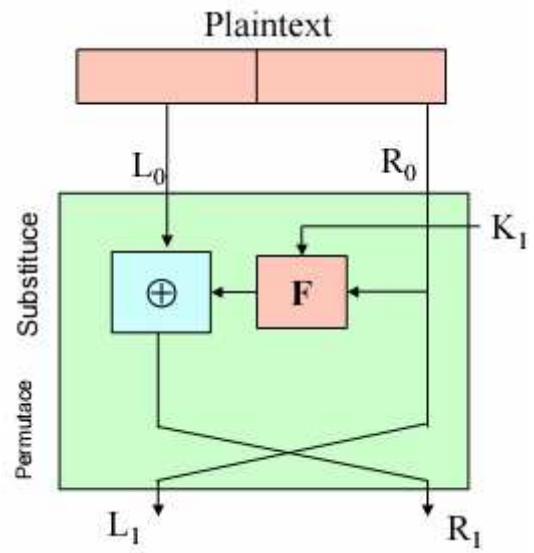
Výhody a nevýhody symetrické kryptografie:

- Výhodou symetrických metod je hlavně jejich rychlost.
- Největší nevýhodou je, že pokud chceme s někým komunikovat, musíme si předem bezpečným kanálem předat klíč. To někdy může být obrovský problém.
- Druhá nevýhoda je počet klíčů. Chceme-li zajistit, aby mohli spolu tajně komunikovat dvě osoby, je zapotřebí jednoho klíče. Pro tři osoby jsou to již tři klíče, pro čtyři osoby šest klíčů, obecně je počet klíčů dán vztahem: *počet klíčů = n * (n - 1) / 2*, kde *n* je počet osob. Při vyšším počtu osob tak začíná být správa klíčů problémem.

V dalším textu si popíšeme základní Feistelovu šifru, která představuje základ některých symetrických šifer a taky šifru DES, která zašifrovala zatím nejvíce dat ze všech šifer.

2.5.1 Feistelova šifra

Tuto šifru navrhl v roce 1973 Horst Feistel, který v té době pracoval pro firmu IBM. Na principu Feistelovy šifry byl později založen IBM Lucifer algoritmus, který se stal základem pro algoritmus DES v roce 1977. Mnoho symetrických algoritmů využívá právě Feistelova principu – jedná se v principu o substitučně permutační síť. Znázornění jedné iterace algoritmu je na obrázku 2.11.



L_0 = levá polovina bloku

R_0 = pravá polovina bloku

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$

Obr. 2.11 – Feistelova šifra

Proces u Feistelovy šifry je reverzibilní - to znamená, že se při dešifrování použije stejný algoritmus, ale pořadí sub-klíčů je opačné:

$R_{i-1} = L_i$

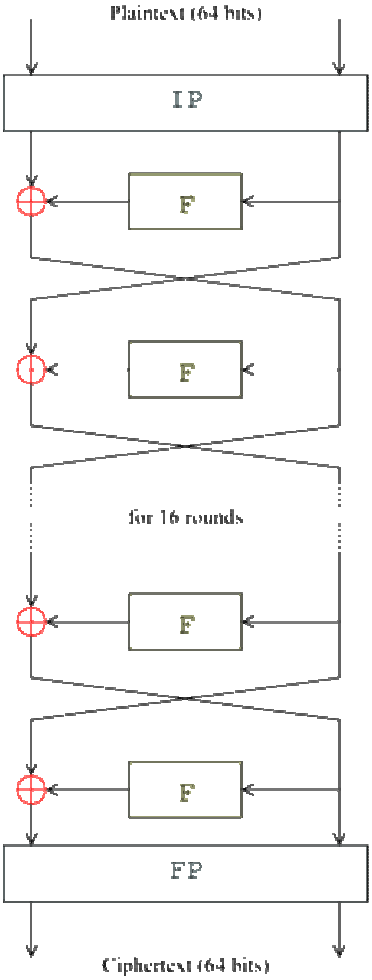
$L_{i-1} = R_i \oplus F(R_{i-1}, K_{i-1})$

Sub-klíče se u Feistelovy šifry vytvářejí v bloku „*Key Schedule*“. Tvorba sub-klíčů má velmi výrazný vliv na bezpečnost algoritmu.

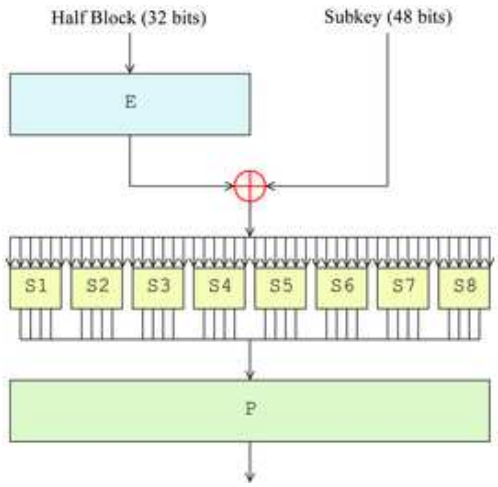
2.5.2 Šifra DES

Šifra DES byla vyvinuta firmou IBM v roce 1976 na zakázku NIST (National Institute of Standards and Technology) . NSA (National Security Agency) u algoritmu modifikovala S-boxy a taky upravila délku klíče ze 128 na 56 bitů. Tento algoritmus šifruje bloky dat o šířce 64 bitů klíčem o velikosti 56 bitů. Šifra má 16 rund (iterací). Počet všech možných klíčů je $2^{56} = 7.2 \times 10^{16}$. Na obrázku 2.12 je zobrazen hlavní cyklus algoritmu. Tento algoritmus začíná počáteční permutací, která je označena IP a končí koncovou permutací, jež je označena FP. Tyto permutace jsou navzájem inverzní. Před započetím hlavních iterací šifry, je 64-bitový blok rozdělen na dvě poloviny stejně jako u Feistelovy šifry. Feistelův princip, kterého tato šifra využívá, zajišťuje, že šifrování a dešifrování jsou velmi podobné procesy. Jediný rozdíl spočívá v tom, že sub-klíče jsou u dešifrování aplikovány v opačném pořadí než u šifrování. Červený symbol ⊕ v obrázku označuje operaci XOR.

Na obrázku 2.13 je zobrazena funkce F. Tato funkce pracuje s blokem dat o velikosti 32 bitů a skládá se ze 4 kroků:



Obr. 2.12 – Hlavní cyklus DES



Obr. 2.13 – Znázornění funkce F

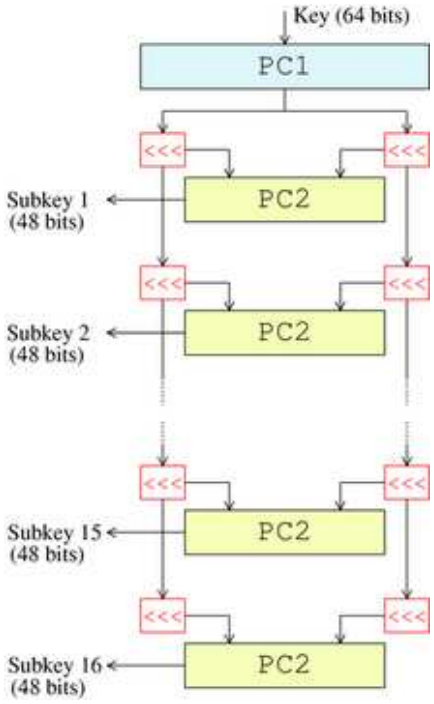
1. krok - představuje expanzi (jednotka E) 32-bitového vstupního bloku na 48 bitů pomocí expanzní permutace, jež duplikuje některé bity vstupní posloupnosti.

2. krok – v tomto kroku se pomocí operace XOR sečte expandovaný blok o velikosti 48 bitů spolu se sub-klíčem, jež má taky velikost 48 bitů.

3. krok – představuje jediný nelineární krok DES algoritmu. Výsledek z předchozího kroku je rozdělen do 8 6-bitových částí a každá tato část je substituována jedním S-boxem. Výstupem každého S-boxu jsou 4 bity. S-box představuje vyhledávací tabulku, která podle zadané vstupní adresy (6 bitů) vypíše na výstup obsah jedné ze 64 položek této tabulky.

4. krok – na závěr je na 32-bitový výstup z S-boxů aplikována permutace P (tzv. P-box).

Na obrázku 2.14 je zobrazen rozvoj klíče pro šifrování, neboli algoritmu, který vytváří sub-klíče. Na začátku algoritmu je pomocí permutačního výběru (PC1 – angl. permutation choice) vybráno 56 bitů klíče z počátečních 64 bitů – zbývajících 8 bitů je nevyužito. Těchto 56 bitů je poté rozděleno na dvě 28-bitové poloviny a každá je pak zpracovávaná samostatně. Dále pak jsou obě poloviny rotovány doleva o jeden nebo dva bity, v závislosti na konkrétní iteraci algoritmu. 48-bitový sub-klíč je pak následně vybrán pomocí druhého permutačního výběru (PC2) – 24 bitů z levé poloviny a 24 bitů z pravé poloviny.

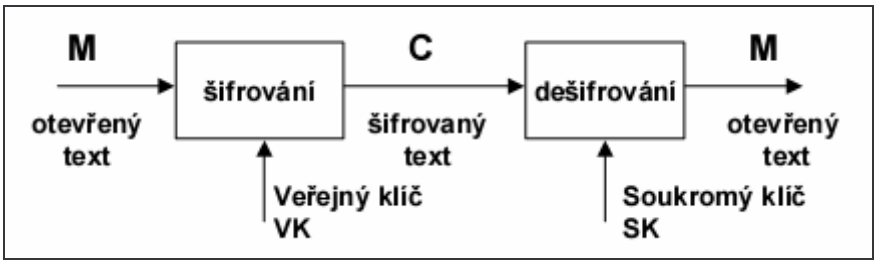


Obr. 2.14 – Rozvoj klíče

2.6 Kryptografie s veřejným klíčem

Jinak taky nazývaná asymetrická kryptografie (angl. public key cryptography). Asymetrická proto, že využívá jiného klíče pro zakódování zprávy a jiného pro dekódování. Dohromady se oba klíče

nazývají párem klíčů (angl. keypair). Šifruje se pomocí tzv. veřejného klíče (angl. public key) a dešifruje se pomocí soukromého klíče (angl. private key). Viz obrázek 2.15.



Obr. 2.15 – Průběh asymetrického šifrování a dešifrování

Veřejný klíč je skutečně veřejný, tj. pokud uživatel chce, aby mu někdo mohl poslat zakódovanou zprávu, musí nejprve dát k dispozici tento svůj veřejný klíč. Ten použije odesílatel pro zakódování tajné zprávy a kód odešle. Pro dekódování pak potřebuje příjemce mít druhý klíč z páru, tzv. soukromý klíč, který jako jediný lze použit pro dekódování. Klíčový pár se tvoří většinou zároveň. Algoritmus uživateli vygeneruje oba klíče, veřejný klíč uživatel dá k dispozici kolegům a soukromý klíč si dobře uschová. S délkou klíče je to u asymetrických šifer oproti symetrickým jinak. Asymetrické šifry totiž pracují se specifickým druhem čísel, např. s prvočísly. Při kryptoanalýze se pak tedy stačí zabývat jen tímto oborem čísel. Z tohoto důvodu je potřeba počet bitů klíče oproti symetrickým metodám patřičně navýšit, aby byla zachována patřičná míra bezpečnosti. V dnešní době se tak běžně pracuje s 1024-bitovými či 2048-bitovými klíči. Mezi nejznámější asymetrické algoritmy patří algoritmy DH (Diffie – Hellman, 1976), RSA (Rivest – Shamir – Adleman, 1977) a DSA (Digital Signature Algorithm, 1991). V této kapitole si popíšeme šifru RSA.

Výhody a nevýhody asymetrické kryptografie:

- Hlavní výhodou je, že nemusíme nikam posílat náš soukromý klíč a proto tedy nemůže dojít k prozrazení klíče při přenosu, jako je tomu u symetrických šifer. Naproti tomu veřejný klíč je možné dát všem k dispozici.
- Je třeba méně klíčů než u symetrických metod – při komunikaci více lidí postačí pro každou osobu jen jeden pár klíčů.
- Nevýhodou asymetrických šifer je jejich rychlost. Tyto metody jsou řádově až 1000-krát pomalejší než symetrické metody.
- Další nevýhodou asymetrické kryptografie je nutnost ověřené pravosti veřejného klíče (tedy identifikace majitele klíče). Pro tyto účely existují např. certifikační úřady, které zjednodušeně řečeno udržují databázi osob s ověřenou totožností a jejich veřejných klíčů.

Vzhledem k pomalé rychlosti, se většinou využívá kombinace obou metod. Např. se tajný klíč symetrického algoritmu zakóduje veřejným klíčem asymetrické metody a tak je zaručen jeho bezpečný přenos k příjemci. Zprávu můžeme rovnou zakódovat tajným klíčem.

2.6.1 Šifra RSA

RSA (podle iniciál autorů Rivest, Shamir, Adleman) je šifra s veřejným klíčem. Jedná se o první algoritmus, který je vhodný, jak pro podepisování, tak i šifrování. Používá se i dnes, přičemž při dostatečné délce klíče, jej lze považovat za bezpečný. Bezpečnost RSA je postavena na předpokladu, že rozložit číslo na součin prvočísel (faktorizace) je velmi obtížná úloha. Z čísla n = p * q je tedy, v rozumném čase, prakticky nemožné zjistit činitele p a q.

Popis činnosti algoritmu:

Adam a Pavel by spolu rádi komunikovali prostřednictvím otevřeného (nezabezpečeného) kanálu a Pavel by chtěl poslat Adamovi soukromou zprávu. Nejprve si musí Adam vytvořit klíčový pár:

1. Zvolí se dvě velká a navzájem různá náhodná prvočísla p a q.
2. Spočítá se jejich součin $n = p \times q$.
3. Spočítá se hodnota Eulerovy funkce $\varphi(n) = (p - 1)(q - 1)$.
4. Zvolí se celé číslo e menší než $\varphi(n)$, které je s $\varphi(n)$ nesoudělné.
5. Nalezne se číslo d tak, aby platilo $d \times e \equiv 1 \pmod{\varphi(n)}$.

Veřejným klíčem je pak dvojice (n, e), přičemž n se označuje jako modul a e jako veřejný (šifrovací) exponent. Soukromým klíčem je dvojice (n, d), kde d se označuje jako soukromý (dešifrovací) exponent. Veřejný klíč poté Adam zveřejní a zašle Pavlovi. Soukromý klíč si ponechá v tajnosti. Zašifrování zprávy pak probíhá následovně: Pavel chce Adamovi poslat zprávu M. Tuto zprávu převede určitým domluveným způsobem na číslo m (m < n). Zašifrovaným textem je pak číslo $c = m^e \bmod n$. Tento zašifrovaný text poté zašle nezabezpečeným kanálem Adamovi. Adam od Pavla příjme zašifrovaný text c. Původní zprávu získá následujícím výpočtem: $m = c^d \bmod n$. Fakt, že tímto výpočtem získáme původní zprávu, je důsledkem následující rovnosti: $c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}$.
Příklad zašifrování a dešifrování:

- p = 61 (první prvočíslo)
- q = 53 (druhé prvočíslo)
- n = p x q = 3233 (modul, veřejný)
- e = 17 (veřejný, šifrovací exponent)
- d = 2753 (soukromý, dešifrovací exponent)

Pro zašifrování zprávy m = 123 proběhne výpočet: $c = 123^{17} \bmod 3233 = 855$
Pro dešifrování zprávy c = 855 proběhne výpočet: $m = 855^{2753} \bmod 3233 = 123$

Algoritmus RSA lze snadno použít pro *digitální podpis*. Základním principem digitální podpisu je následující použití šifry: pokud chce Adam poslat Pavlovi podepsanou zprávu, připojí k ní číslo získané zašifrováním (je použit soukromý klíč) hash kódu své zprávy. Pavel poté zpětně dešifruje (je použit veřejný klíč) tento podpis a porovná výsledek s hash kódem zprávy. Pokud zpráva nebyla během přenosu změněna, vyjde stejná hodnota (jelikož algoritmus je z hlediska šifrování a dešifrování symetrický - lze zaměnit *e* a *d*). Jelikož jediný, kdo zná tajný klíč Adama, je sám Adam, je tím zaručeno, že hash kód zašifroval Adam.

Co se týče *šifrování* zpráv, je RSA o hodně pomalejší než dříve popsaný algoritmus DES. V praxi se typicky tajná zpráva zašifruje symetrickým algoritmem, symetrický klíč se zašifruje RSA algoritmem a nakonec se příjemci pošle asymetricky zašifrovaný tajný klíč a symetricky zašifrovaná zpráva.

2.7 Historie AES

V roce 1977 byl jako americký šifrovací standard vybrán algoritmus DES (Data Encryption Standard). DES byl určen pro šifrování citlivých (nikoliv však utajovaných) dat. Ve velice krátké době se stal nejrozšířenější šifrou na světě a de facto i světovým komerčním šifrovacím standardem. Do dnešní doby, i přes veškeré úsilí kryptologů na celém světě, se nepodařilo najít analytický útok, který by umožnil „prolomení“ této šifry.

Na druhou stranu, co nedokázali kryptologové analytickými útoky, docílil rozvoj výkonnosti výpočetní techniky. Velice výraznou slabinou u šifry DES byl totiž poměrně krátký klíč – jen 56 bitů. V roce 1993 byl publikován článek s informací o tom, že je možné sestrojit zařízení v ceně okolo jednoho miliónu dolarů, které dokáže vyluštit klíč DES řádově během několika hodin. V roce 1997 se tak tomu skutečně stalo, kdy díky spojení několika desítek tisíc počítačů na internetu byl konečně vyluštěn konkrétní klíč DES. Od té chvíle bylo zřejmé, že je potřeba nalézt jeho nástupce.

A z tohoto důvodu vypsal americký standardizační úřad NIST (National Institute of Standards and Technology) výběrové řízení na AES (Advanced Encryption Standard). Součástí výběrového řízení byla definice požadavků, které musí nový standard splňovat. Z 15 kandidátů kteří splnili podmínky, byl po pečlivém výběru, který sledovala celá kryptografická veřejnost a který trval celé 4 roky, vyhlášen vítěz. Za AES byl 26. listopadu 2001 schválen algoritmus Rijndael (čti „rijndél“) belgických kryptologů Vincenta Rijmena a Joana Daemena. Rijndael je ještě všestrannější šifra než specifikují podmínky AES, přesto budeme-li v následujícím textu mluvit o šifře Rijndael, budeme mít na mysli její specifikaci dle AES. Očekává se, že algoritmus bude sloužit nejméně 20 let a bude masově používán po celém světě.

2.8 Matematická příprava

V následující části si představíme základní matematické koncepty potřebné pro AES šifru. Všechny byty v AES algoritmu jsou interpretovány jako prvky Galoisova tělesa GF(2^8). Tyto prvky mohou být mezi sebou sečítány a násobeny, ovšem tyto operace jsou odlišné od těch, které se používají pro běžná čísla. Prvky Galoisova tělesa jsou reprezentovány polynomy s koeficienty odpovídajícími bitům b_7, b_6, …, b_0 daného bytu, tedy:

$$\mathbf{b_7x^7 + b_6x^6 + \ldots + b_1x + b_0}$$

- **Sečítání**

Sečtení dvou prvků v Galoisově tělesu je dosaženo sečtením koeficientů odpovídajících mocnin polynomů. Sečtení je jednoduché a odpovídá operaci XOR na bytové úrovni. Ilustrační příklad viz obrázek 2.16.

$(x^6+x^4+x^2+x+1) + (x^7+x+1) = x^7+x^6+x^4+x^2$	(polynomial notation);
$\{01010111\} \oplus \{10000011\} = \{11010100\}$	(binary notation);
$\{57\} \oplus \{83\} = \{d4\}$	(hexadecimal notation).

Obr. 2.16 – Příklad sečtení dvou prvků v GF(2^8)

- **Násobení**

Operace násobení • je definována jako násobení polynomů modulo ireducibilním polynomem *m(x)*. Pro AES je tento polynom $m(x) = x^8 + x^4 + x^3 + x + 1$.

Příklad {57} • {83} = {c1}:

$$\begin{aligned}
(x^6+x^4+x^2+x+1)\,(x^7+x+1) &= x^{13}+x^{11}+x^9+x^8+x^7+ \\
&\quad x^7+x^5+x^3+x^2+x+ \\
&\quad x^6+x^4+x^2+x+1 \\
&= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1
\end{aligned}$$

$$x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \ \text{modulo}\ (x^8+x^4+x^3+x+1)$$
$$= x^7+x^6+1.$$

Obr. 2.17 – Příklad násobení dvou prvků v GF(2^8)

- **Polynomy s koeficienty v GF(2^8)**

Můžeme také definovat polynomy, jejichž koeficienty jsou prvky tělesa GF(2^8). AES používá polynomy nejvýše 3. stupně, které reprezentují 4-bytové vektory. Sčítání těchto polynomů je opět XOR jednotlivých koeficientů. Výsledek násobení takovýchto polynomů $a(x) \otimes b(x)$ je polynom $d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$, pro který platí:

$$d_0 = (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3)$$
$$d_1 = (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3)$$
$$d_2 = (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3)$$
$$d_3 = (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)$$

Jedná se o násobení polynomů modulo polynomem $x^4 + 1$. Toto násobení můžeme také zapsat ve tvaru matice jako:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Operace nad polynomy popsané v této kapitole jsou v praxi optimalizovány pro hardware, a některé z nich se pro zvýšení rychlosti, v případě softwarové implementace, provádějí tabulkově.

2.9 Vlastní specifikace AES

AES šifra je bloková šifra, a ačkoli podporuje i větší bloky, je délka vstupního a výstupního bloku dat definována podle normy jako 128 bitů. Tato délka je reprezentována konstantou ***Nb = 4***, což znamená počet 32-bitových slov (4 * 32 bitů = 128 bitů). U AES algoritmu je délka klíče volitelně buď 128, 192 nebo 256 bitů. Označuje se ***Nk*** a opět se udává v počtech slov, tedy ***Nk = 4, 6*** nebo ***8***. AES pracuje v cyklu a počet iterací se označuje proměnnou ***Nr***, jejíž hodnota je závislá na délce klíče, jak ukazuje následující tabulka 2.3:

	Délka klíče Nk slov	Délka bloku Nb slov	Počet iterací cyklu Nr
AES – 128	4	4	10
AES – 192	6	4	12
AES – 256	8	4	14

Tab. 2.3 – specifikace konstant u AES

2.10 Algoritmus šifrování

Před samotnou operací šifrování se provádí takzvaná rutina expanze klíče (*Key Expansion Routine*), která slouží k naplnění pomocného pole ***w*** o velikosti 4 x (Nr + 1) 32-bitových slov. Těmto slovům říkáme *rundovní klíče* (angl. *Round Keys*). V prvním kroku šifry se okopíruje vstupní blok dat do tzv. *stavu* (angl. *State*). Stav je dvourozměrné pole o velikosti 4 x Nb, nad nímž probíhají veškeré další operace šifry. Celý algoritmus AES šifry je následující:

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

        state = in

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

        for round = 1 step 1 to Nr–1
                SubBytes(state)
                ShiftRows(state)
                MixColumns(state)
                AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        end for

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


        out = state
end
```

Pseudokód šifrovacího algoritmu AES

V prvním kroku, jak již bylo řečeno, se okopíruje vstupní blok dat do stavu a v dalším kroku se přidají ke stavu počáteční 4 klíče cyklu pomocí operace AddRoundKey. Dále pak následuje hlavní cyklus šifry, jehož počet iterací je dán hodnotou Nr sníženou o jedničku (viz předchozí kapitola 2.9). V hlavním cyklu jsou vykonávány celkem 4 základní procedury a všechny z nich mění hodnoty stavu. Tyto procedury budou popsány v následujících kapitolách. Na závěr se provede ještě jedna iterace cyklu, ovšem s vynechanou procedurou MixColumns. Poslední instrukcí šifry je okopírování stavu na výstup.

2.11 Algoritmus dešifrování

Algoritmus dešifrování využívá stejného schématu jako šifrování. Dochází pouze k modifikacím jednotlivých operací a ke změně pořadí při jejich provádění. Celý algoritmus AES dešifrování je následující:

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

        state = in

        AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

        for round = Nr-1 step -1 downto 1
                InvSubBytes(state)
                InvShiftRows(state)
                InvMixColumns(state)
                AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
        end for

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

        out = state
end
```

Pseudokód dešifrovacího algoritmu AES

2.12 Popis jednotlivých operací

SubBytes a InvSubBytes

SubBytes je nelineární bytová substituce, která je aplikována na každý byte $s_{i,j}$ v matici state. Tato substituce se skládá ze dvou kroků:

1. Výpočet multiplikativní inverze prvku ***s*** v Galoisově tělese GF(2^8), tedy $t = s^{-1} \bmod m(x)$.
2. Poté na byte *t* aplikujeme afinní transformaci nad GF(2^8) definovanou takto:

$$\begin{bmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \\ s'_4 \\ s'_5 \\ s'_6 \\ s'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Operace InvSubBytes se provede jako inverze afinní transformace (většinou tabulkově) následovaná výpočtem multiplikativní inverze prvku. Operaci SubBytes nemusíme v praxi počítat, ale můžeme si ji uložit jako tabulku, nazývanou ***S-box*** (tento postup je využit v této diplomové práci). Viz obrázek 2.18.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Obr. 2.18 - S-Box: substituční hodnoty pro byte *xy* (v hexadecimálním tvaru)

Příklad použití S-boxu: jestliže $s_{1,1}$ = {53}, pak substituční hodnota bude určena průsečíkem řádku s indexem '5' a sloupce s indexem '3'. Výsledkem je pak hodnota {ed}, jak si můžeme ověřit v obrázku 2.18.

MixColumns a InvMixColumns

Operace MixColumns se provádí nad každým sloupcem matice state. Na sloupec se nahlíží jako na polynom s koeficienty z GF(2^8) a násobí se modulo $x^4 + 1$ fixním polynomem $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$, kde ve složených závorkách jsou hexadecimálně zapsány polynomy nad GF(2^8). Celou tuto operaci můžeme také zapsat jako násobení matice a vektoru:

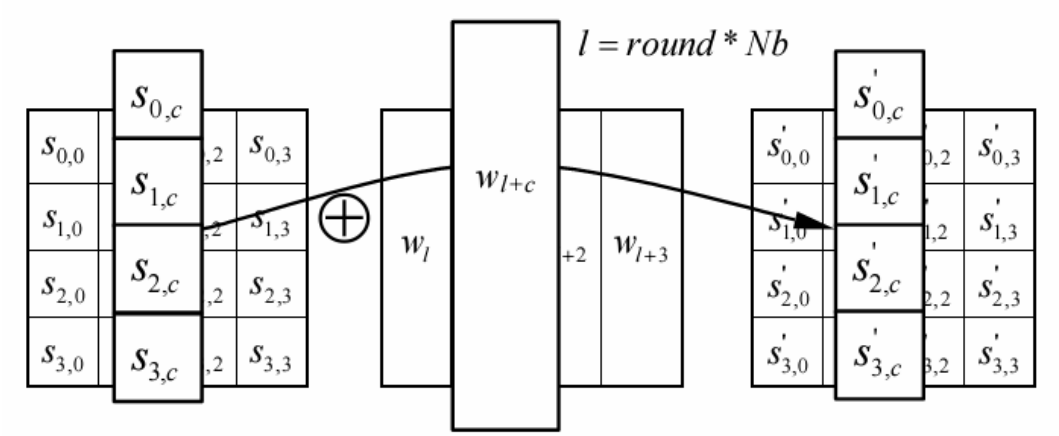
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

kde c je index sloupce v matici state a platí pro něj 0 <= c < 4.

Inverzní operace je podobná, násobí se (opět modulo $x^4 + 1$) inverzí polynomu a(x), tedy polynomem $a^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$.

AddRoundKey

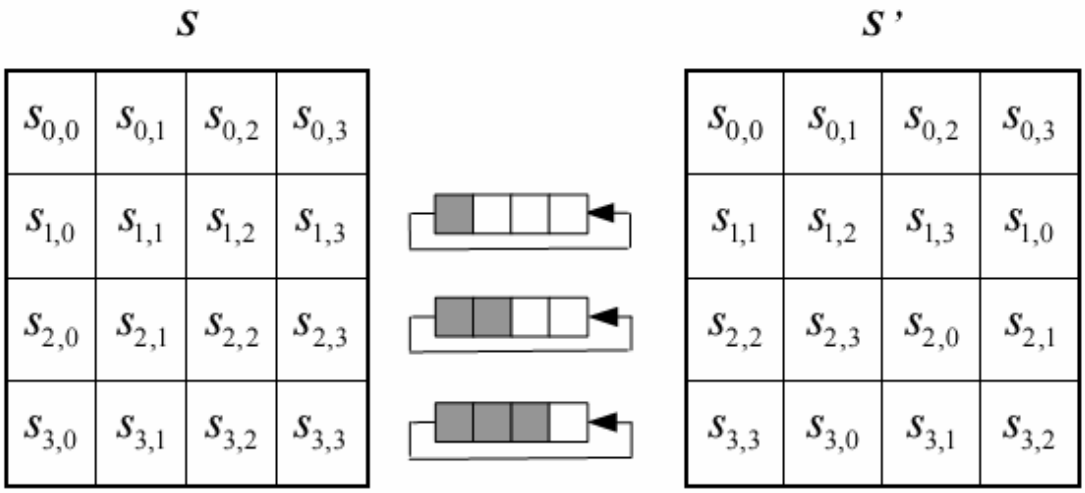
Tato operace představuje jednoduchou XOR operaci jednotlivých sloupců matice state s rundovními klíči. Tyto klíče se v každé iteraci cyklu mění. Vše nejlépe objasní obrázek 2.19.



Obr. 2.19 – Operace AddRoundKey

ShiftRows a InvShiftRows

Operace ShiftRows provádí v matici state rotaci jednotlivých řádků doleva. První řádek matice se nerotuje, druhý řádek se rotuje doleva o jeden byte, třetí řádek o dva byte a čtvrtý řádek se rotuje o tři byte. Operaci ShiftRows znázoňuje obrázek 2.20.



Obr. 2.20 – Operace ShiftRow

2.13 Expanze klíče

Celá expanze šifrovacího klíče na pole rundovních klíčů je znázorněna na konci této kapitoly. Při této expanzi se využívá operace RotWord, což je cyklický posun jednotlivých bytů 32-bitového slova o jeden byte doleva a SubWord, což je již známá operace SubByte aplikovaná opět na jednotlivé byty 32-bitového slova. Cyklická konstanta Rcon[i] je 32bitové slovo tvaru [x^{i-1}, {00}, {00}, {00}], kde x^{i-1} reprezentuje prvek GF(2^8). Doplňme ještě, že expanze klíče se nemusí vždy provádět před spuštěním vlastního šifrování, ale dá se také počítat v jeho průběhu. Toto je obzvláště výhodné pro zařízení jako jsou čipové karty apod., která nedisponují velkým množstvím paměti.

Závěrem: Tolik tedy k šifře Rijndael. Její návrh je velmi jednoduchý, základem jsou operace nad různými algebraickými strukturami. Je velmi flexibilní při realizaci na různých typech hardware. Přestože to tak na první pohled nevypadá, vychází AES z principů, které byly použity již při tvorbě DES a dodnes nebyly zpochybněny. Použité délky klíčů však u AES znemožňují útok hrubou silou. Nejsou známy žádné slabé klíče[1], a navíc šifra vykazuje skvělou odolnost vůči lineární a diferenciální[2] kryptoanalýze.

---

[1] Klíče, při jejichž použití šifra vykáže určité pravidelnosti ve výstupu, či dokonce předvídatelné chování.
[2] Typy kryptoanalytických metod.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
        word temp
        i = 0
        while (i < Nk)
                w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
                i = i+ 1
        end while
        I = Nk
        while (i < Nb * (Nr+1))
                temp = w[i-1]
                if (i mod Nk = 0)
                        temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
                else if (Nk > 6 and i mod Nk = 4)
                        temp = SubWord(temp)
                end if
                w[i] = w[i-Nk] xor temp
                i = i + 1
        end while
end
```

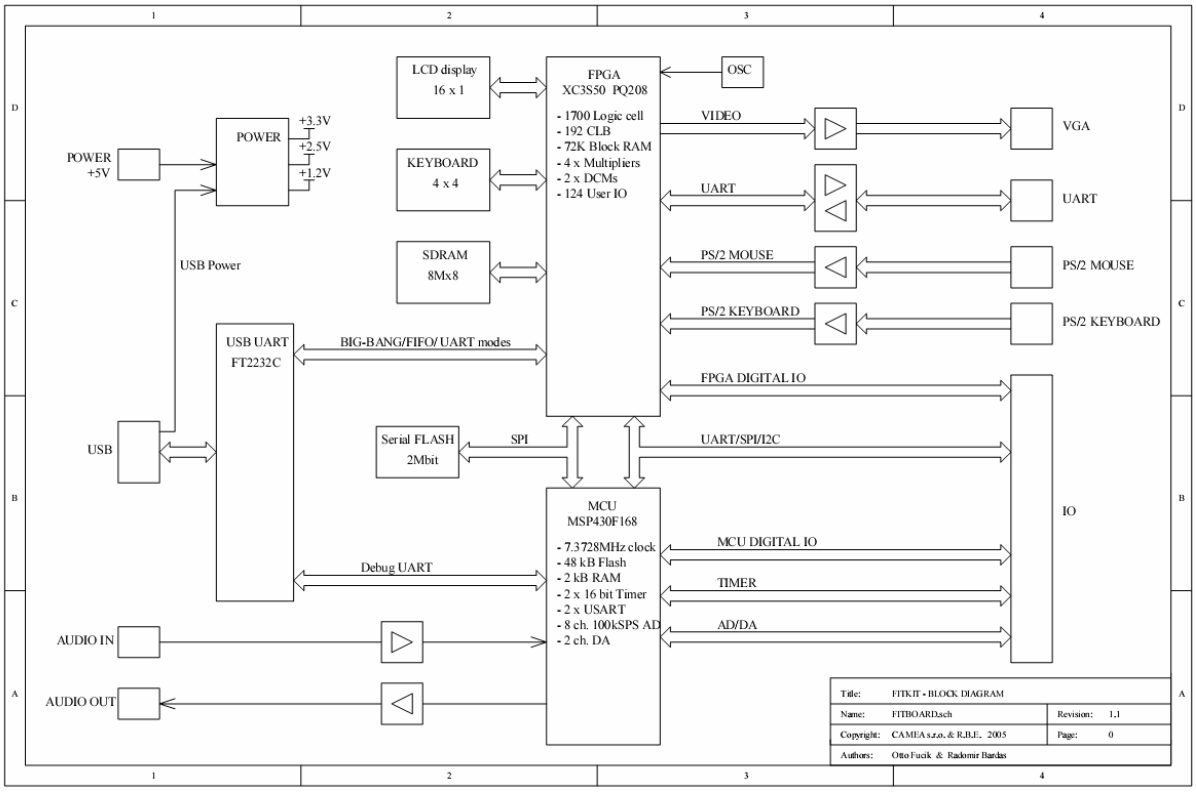
Obr. 2.10 – KeyExpansion procedura

Kapitola 3

3 FITKit

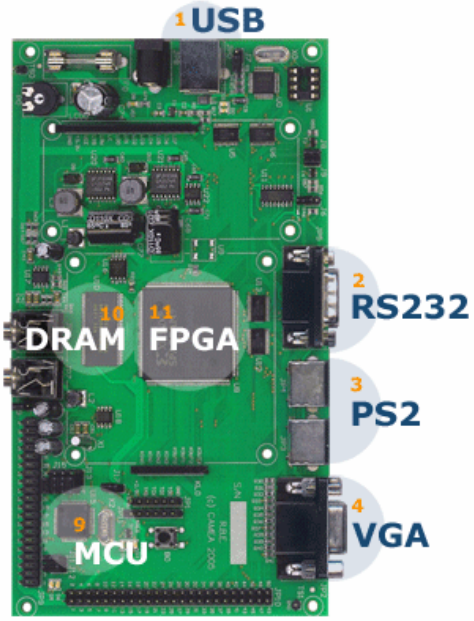
3.1 Popis platformy FITKit

Platforma FITKit vznikla pod záštitou Fakulty informačních technologií (FIT) Vysokého učení technického v Brně. Cílem vzniku této platformy byl fakt, že výuka technických kurzů musí být směrována nejen teoreticky, ale hlavně i prakticky. Cílem nasazení platformy do výuky na FIT bylo umožnit studentům, aby mohli navrhovat a prakticky realizovat nejen softwarové, ale také hardwarové projekty či dokonce celé aplikace. Obvod šifry AES, jež je popisován v této diplomové práci, byl navržen s ohledem na velikost FPGA, jež je součástí platformy FITKit. To z toho důvodu, aby bylo možné navržený obvod případně umístit do FPGA na FITkitu. Na obrázku 3.1 vidíme základní blokové schéma FITKitu.



Obr. 3.1 – Základní blokové schéma FITKitu

Na obrázcích 3.2 a 3.3 dále pak vidíme, jak FITKit vypadá ve skutečnosti.



Obr. 3.2 – pohled na PCB

Obr. 3.3 – pohled shora

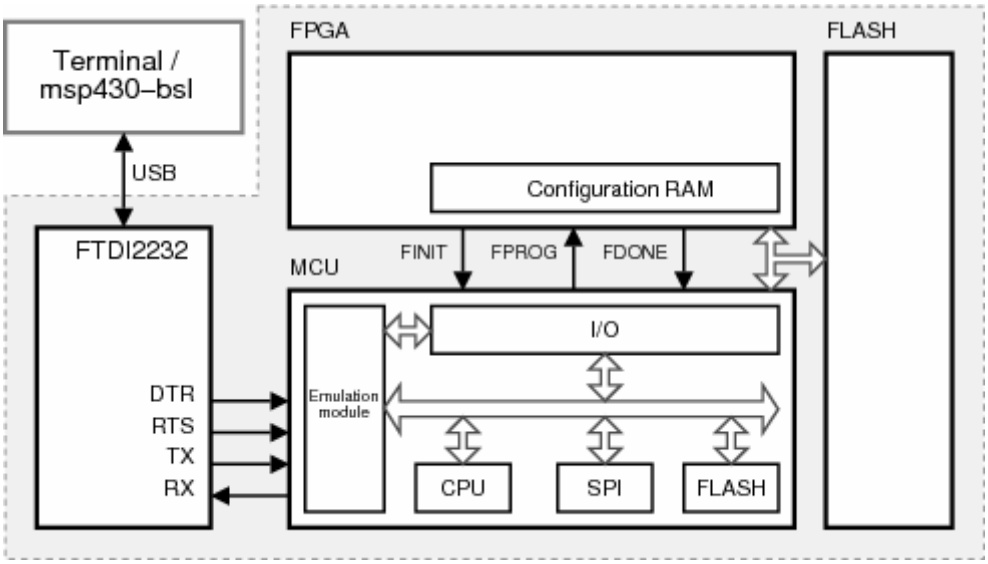
Platforma FITKit obsahuje dvě hlavní programovatelné jednotky a to sice mikrokontrolér MSP430F168 od firmy Texas Instrument a hradlové pole FPGA Spartan 3 XC3S50 – 4PQ208C od firmy Xilinx. Na piny mikrokontroléru je připojen audio vstup a výstup. Dále pak je k němu připojen konektor RS232 pro sériovou komunikaci s okolím. Na piny hradlového pole FPGA je připojeno více zařízení:

- LCD displej – jedná se o jednořádkový displej, na kterém lze zobrazit až 16 znaků.
- Klávesnice o velikosti 4x4.
- 2 porty PS2 – tyto porty se používají v dnešní době pro připojení myši a klávesnice k PC.
- VGA konektor – pro připojení zobrazovacího zařízení, např. LCD monitoru.
- 64 Mb synchronní paměť v konfiguraci 8M8.

V následující části popíšeme, jak je možné nahrát do mikrokontroléru program a do FPGA jeho konfiguraci.

3.2 Nahrávání programu a konfigurace FPGA

Na obrázku 3.4 si můžeme prohlédnout základní blokové schéma systému pro bootování, nahrávání programu pro MCU (mikrokontrolér) a konfigurace FPGA do hradlového pole FPGA. Chceme-li nahrát kód programu do MCU, poslouží nám k tomu program *msp430-bsl* a *boot leader*, který je součástí mikroprocesoru a umožňuje načtení programu do FLASH paměti MCU. Chceme-li nahrát konfiguraci do FPGA, můžeme to provést dvěma následujícími způsoby:



Obr. 3.4 – Základní schéma pro načítání programu a konfigurace

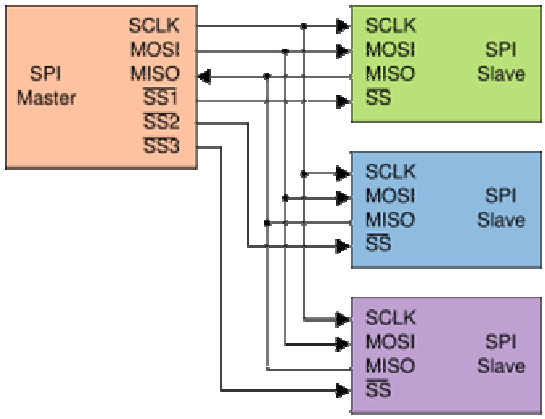
1) První způsob spočívá v nahrání konfigurace FPGA do FLASH paměti a odtud pak do FPGA. První krok spočívá v použití programu, který musí být přítomen v MCU. Tento program nám umožní za pomocí standardního terminálu (musí podporovat protokol xmodem) přenést soubor s konfigurací do paměti FLASH. Spuštění přenosu vykonáme v terminálu zadáním příkazu „*flash w fpga*“. MCU v té chvíli začne čekat na příjem konfiguračního souboru. V terminálu vybereme *File Transfer*, zvolíme typ protokolu *xmodem* a zahájíme přenos konfigurace do FLASH paměti. Druhý krok, tedy nahrání konfigurace z FLASH do FPGA se provádí automaticky při startu FITKitu pomocí speciální funkce, která je součástí knihovny *libfitkit*. Libfitkit knihovna obsahuje základní příkazy pro práci s MCU.

2) Druhý způsob spočívá v přímém nahrání konfigurace do FPGA. Za tímto účelem musíme mít opět v MCU přítomen program, který nám toto umožní. Postup nahrání je podobný jako v minulém případě. Spustíme v terminálu příkaz „*prog fpga*“ a vybereme příslušný soubor s konfigurací. Po skončení přenosu máme konfiguraci FPGA nahranou v hradlovém poli FPGA.

3.3 Propojovací systém FITKitu

3.3.1 Systém SPI

Abychom mohli blíže vysvětlit propojovací systém FITKitu, musíme si napřed popsat, co je to systém SPI, pomocí něhož jsou komponenty propojeny. SPI (angl. Serial Peripheral Interface) je vysokorychlostní sériové rozhraní typu master - slave. SPI sběrnice může pracovat s jedním hlavním zařízením (angl. master device) a s jedním nebo více podřízenými zařízeními (angl. slave devices). Vše ilustruje obrázek 3.5. Rozhraní SPI je tvořeno následujícími signály:

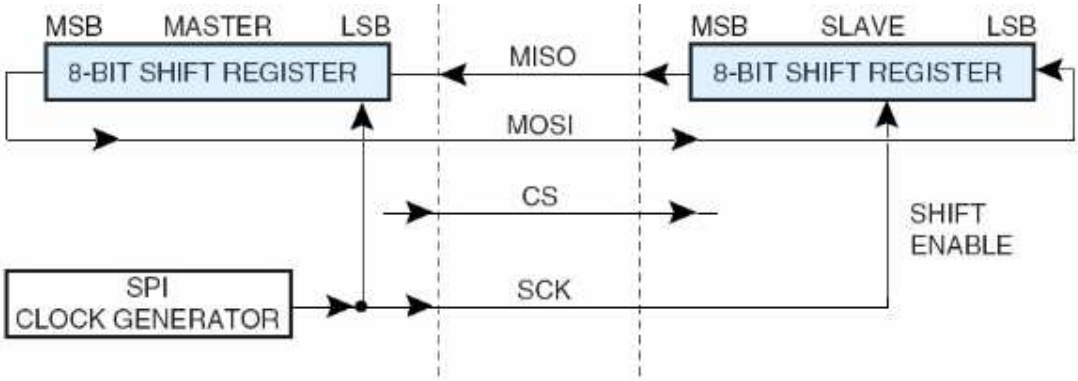


Obr. 3.5 – Schéma SPI

1. **SCLK** – seriál clock – jedná se o signál hodin, je generován master zařízením a slouží pro vzorkování dat.
2. **MOSI** – master output slave input – jedná se o data vystupující z master zařízení (generuje master).
3. **MISO** – master in slave out – jedná se o data vystupující ze slave zařízení (generuje slave).
4. **SC** – chip select – signál pro výběr konkrétního zařízení, aktivní v logické 0.

Každé zařízení obsahuje jeden 8-bitový posuvný registr. Komunikace mezi master a slave zařízením probíhá následujícím způsobem (proces komunikace je znázorněn na obrázku 3.6):

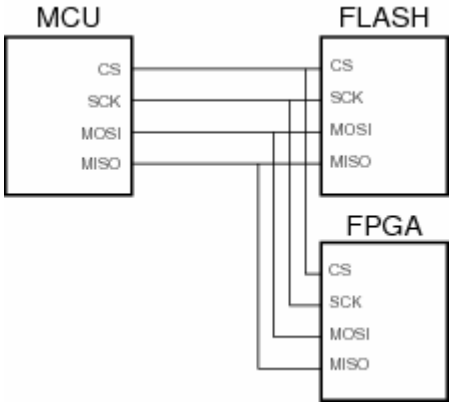
1. Do registru se zapíšou požadovaná data k přenosu.
2. Signál CS se přepne do hodnoty logické 0, čímž se nastartuje přenos dat.
3. Master zařízení generuje signál SCLK, kdy s nástupnou, popř. sestupnou hranou dojde k přenosu dvou bitů pomocí signálů MOSI a MISO a zároveň i k posunu dat v registru.
4. Po 8 hodinových taktech jsou data přenesena



Obr. 3.6 – Proces komunikace mezi master a slave zařízením

3.3.2 SPI na FITKitu

Na obr. 3.7 si můžeme prohlédnout základní schéma propojení hlavních programovatelných jednotek na FITKitu. Jak je z obrázku patrné, je mikrokontrolér nakonfigurován v master režimu a paměť FLASH spolu s hradlovým polem FPGA v režimu slave. Uvnitř FPGA mohou být realizovány řadiče periferií, např. klávesnice, LCD displeje apod., jež jsou připojeny k pinům hradlového pole FPGA.



Obr. 3.7 – Propojení základních jednotek FITKit pomocí SPI

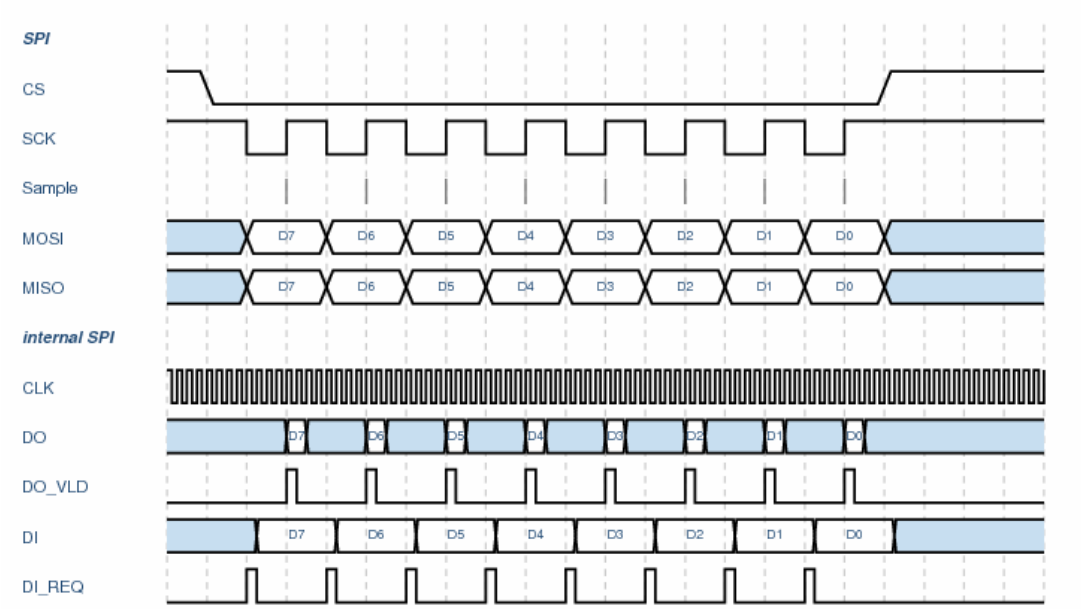
V mikrokontroléru jsou pak naprogramovány ovladače těchto periferií, které vlastně představují příkazy, jež má periférie provést. Může se např. jednat o zobrazení určitého textu na LCD displeji. Tyto instrukce a data se přenášejí sériově pomocí sběrnice SPI z mikrokontroléru do FPGA a naopak v bitové podobě. Jelikož slave zařízení FLASH a FPGA sdílejí veškeré signály, dochází k rozlišení adresovaného zařízení na úrovni přenášených bitů. Za tímto účelem musí jednotky používat disjunktní množinu operačních kódů.

Abychom mohli uvnitř FPGA vytvořit libovolné množství zařízení, byla vytvořena vnitřní SPI sběrnice a protokol, který umožňuje komunikaci s konkrétním zařízením. Datový rámec protokolu můžeme rozdělit na tři části (jak je ukázáno na obrázku 3.8). Komunikace pak probíhá následovně:

Obr. 3.8 – Datový rámec komunikačního protokolu

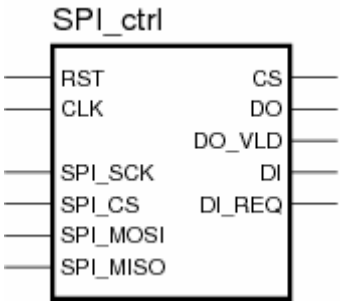
1. Prvně musíme odeslat operační kód. Operační kód je 8 bitů, pomocí nichž rozlišíme, zda adresujeme FPGA nebo FLASH paměť. Dále obsahuje příznaky na 0. a 1. bitu, které říkají, zda se bude jednat o čtení nebo zápis dat.
2. Dále se posílá adresa cílového zařízení. Počet bitů adresy může být libovolný.
3. Nakonec se odesílají data, opět je bitová délka dat neomezená.

Základ vnitřní SPI sběrnice je tvořen SPI řadičem, který provádí jednoduchou konverzi SPI protokolu na vnitřní sériový protokol. Na vnitřní sběrnici pak může být umístěn libovolný počet SPI dekodérů, jejichž úkolem je jednak dekódovat adresu cílového zařízení, ale hlavně poskytnout jednoduché paralelní rozhraní umožňující obousměrnou komunikaci. Na obrázku 3.8 si můžeme prohlédnout úrovně signálů při přenosu jednoho byte.



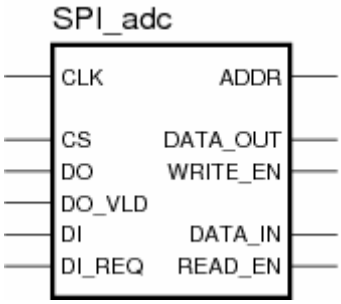
Obr. 3.8 – úrovně signálů při přenosu jednoho byte

SPI Řadič – tato komponenta kromě převodu SPI protokolu, taky dekóduje, zda se jedná o transakci týkající se FPGA či nikoliv. Při vzestupné hraně signálu SCK dochází ke čtení bitu signálu SPI_MOSI a při sestupné hraně dochází ke změně hodnoty signálu SPI_MISO. Všechny signály vnitřní sběrnice jsou synchronizovány hodinovým signálem CLK. Po aktivaci signálu SPI_CS načte řadič 4 bity a pokud odpovídají kódu FPGA, je aktivován signál CS a povoleno generování signálů DO_VLD a DI_REQ. Pokud se identifikace neshoduje, je signál CS neaktivní a řadič čeká na konec transakce indikované deaktivací signálu SPI_CS. Na obrázku 3.9 je vidět základní rozhraní SPI řadiče.



Obr. 3.9 – Rozhraní SPI řadiče

SPI Dekodér – jedná se o synchronní komponentu, jejíž jádro tvoří 4-stavový automat, čítač přenesených bitů a posuvné registry k uchování adresy a dat. Při aktivaci signálu CS, dojde k načtení 4 bitů, jež tvoří spodní nibble operačního kódu. V těchto 4 bitech je zakódováno, zda se bude jednat o čtecí či zapisovací operaci. Dále se načte určitý počet adresových bitů a pokud se shodují s bázovou adresou daného zařízení, dekodér začne načítat datové bity. Při načtení určitého počtu datových bitů, jsou následně vystaveny na sběrnici (signál DATA_OUT) a zároveň vygenerován signál WRITE_EN. Pokud je zvolen režim čtení je vystaven signál READ_EN, který informuje o tom, že je nutné v dalším taktu vystavit na sběrnici DATA_IN platná data. Tyto data jsou pak následně přenesena do MCU. Na obrázku 3.10 je vidět rozhraní SPI dekodéru.



Obr. 3.10 – Rozhraní SPI dekodéru

V této kapitole jsme si krátce popsali platfotmu FITKit a způsob propojení hlavních programovatelných jednotek. Bližší detaily můžeme nalézt na webových stránkách věnovaných platformě FitKit, viz [3].

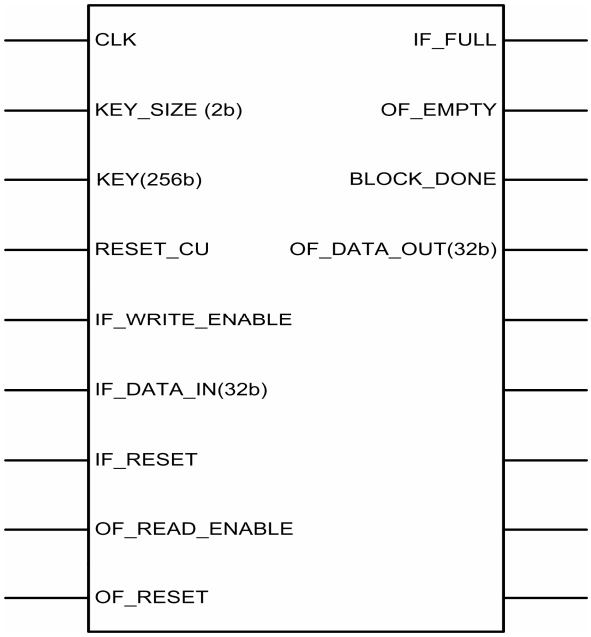
Kapitola 4

4 Návrh hardwarového řešení

4.1 Základní schéma a jeho popis

4.1.1 Rozhraní obvodu

Předpokladem pro návrh obvodu AES šifry byl fakt, že tento obvod nesměl využívat více zdrojů než má k dispozici hradlové pole FPGA na platformě FITKit, aby bylo možné případně tento obvod fyzicky realizovat. Na obrázku 4.1 vidíme základní rozhraní navrženého obvodu, jež realizuje AES algoritmus šifrování.



Obr. 4.1 – základní rozhraní obvodu

Popis jednotlivých signálů:

- Signál ***CLK*** je, jak název napovídá, hodinový signál (angl. Clock) a veškerá činnost spojena s přenosem dat a řízením jednotlivých částí obvodu je synchronizována tímto signálem. Reaguje se vždy na vzestupnou hranu signálu.
- Signál ***KEY_SIZE*** o velikosti 2 bity určuje velikost klíče, který se vystavuje na vstupní sběrnici *KEY*. Signál nabývá následujících hodnot:
  - *KEY_SIZE* = 00 - vstupní klíč má velikost 128 bitů a musí být vystaven na sběrnici *KEY* na bitové pozice 255 až 128 (pozici bitu určujeme od nulté pozice)

- *KEY_SIZE* = 01 - vstupní klíč má velikost 192 bitů a musí být vystaven v tomto případě na bitové pozice 255 až 64.
- *KEY_SIZE* = 10 – vstupní klíč má velikost 256 bitů, tedy nejvyšší možnou délku klíče. V tomto případě jsou obsazeny všechny bitové pozice sběrnice KEY, tedy 255 až 0.
- *KEY_SIZE* = 11 – není definováno

- Sběrnice ***KEY*** – jak již bylo řečeno, jedná se o vstup, na který se přivádí klíč
- Signál ***RESET_CU*** je signál, který provádí celkový reset obvodu. V principu to znamená, že vnitřní řídící jednotka přejde do počátečního stavu. Nejedná se o reset použitých pamětí ani o reset registrů. Tento signál využijeme v případě, že chceme za běhu algoritmu změnit klíč. Postup, jak toto provedeme, je následující:

  1. Na sběrnici *KEY* vystavíme klíč.
  2. A zároveň uvedeme signál *RESET_CU* do logické úrovně 1.
  3. S dalším hodinovým taktem se provede přechod vnitřní řídící jednotky do počátečního stavu.
  4. V následujících taktech se provede operace expanze klíče a nově příchozí data do obvodu jsou již šifrována pomocí tohoto klíče.

  Použití tohoto signálu, za účelem změny šifrovacího klíče, přepokládá, že se momentálně nešifrují žádná data. Pokud bychom tenhle fakt nevzali v úvahu, došlo by k přerušení šifrování aktuálních dat a přišli bychom tak o 128 bitů dat (jeden datový blok).
- Signál ***IF_WRITE_ENABLE*** **(**IF - Input FIFO**)** slouží pro aktivaci zápisu dat do vstupní fronty, jež je součástí obvodu. Nejprve musíme na vstupní sběrnici *IF_DATA_IN* vystavit potřebná data k zápisu. Následně uvedeme signál *IF_WRITE_ENABLE* do logické úrovně 1 a s dalším hodinovým taktem dojde k zápisu dat do vstupní fronty. Vnější zapisovací jednotka musí ovšem hlídat, zda není vstupní fronta plná. Za tímto účelem je potřeba sledovat výstupní signál *IF_FULL*.
- Sběrnice ***IF_DATA_IN*** slouží pro vystavení dat, které chceme zapsat do vstupní fronty.
- Signál ***IF_RESET*** slouží pro vymazání obsahu paměti vstupní fronty. Tento reset ve skutečnosti nemaže obsah paměti, ale mění hodnoty adresových čítačů, jež jsou součástí vstupní fronty. Je to z toho důvodu, že pokud bychom návrh udělali tím způsobem, že by se mazal i obsah paměti, nebylo by možné použít při syntéze prvky BlockRAM v FPGA, neboť by paměť byla vytvořena pomocí distribuované paměti, tedy registrů typu Flip-Flop.
- Signál ***OF_READ_ENABLE*** **–** signál pro aktivací čtení dat z výstupní fronty. Data se po zašifrování ukládají do vnitřní výstupní fronty. S dalším hodinovým taktem se pak data vystaví na sběrnici *OF_DATA_OUT*. Tento signál nemá žádný smysl aktivovat v případě,

že je výstupní fronta prázdná. K vystavení dat v takovém případě totiž nedojde. Zda je fronta prázdná, můžeme zjistit pomocí signálu *OF_EMPTY*.

- ***OF_RESET*** – stejně jako signál *IF_RESET* slouží k vymazání obsahu paměti, v tomto případě ke smazání paměti výstupní fronty.
- Signály ***IF_FULL*** a ***OF_EMPTY*** již byly popsány a mají informační charakter. Signál *IF_FULL* informuje o tom, že vstupní fronta je plná a nelze provést další nahrání dat. Signál *OF_EMPTY* na druhou stranu informuje o tom, že výstupní fronta je prázdná.
- ***BLOCK_DONE*** je signál, který nabývá hodnoty logické úrovně 1 po dobu periody jednoho hodinového taktu v případě, že obvod skončí zašifrování jednoho bloku dat. Jedná se v principu opět o informační signál, který může vnější jednotka sledovat a na který může reagovat.
- ***OF_DATA_OUT*** je sběrnice, na kterou AES obvod vystavuje data z výstupní fronty po aktivaci signálu *OF_READ_ENABLE*.

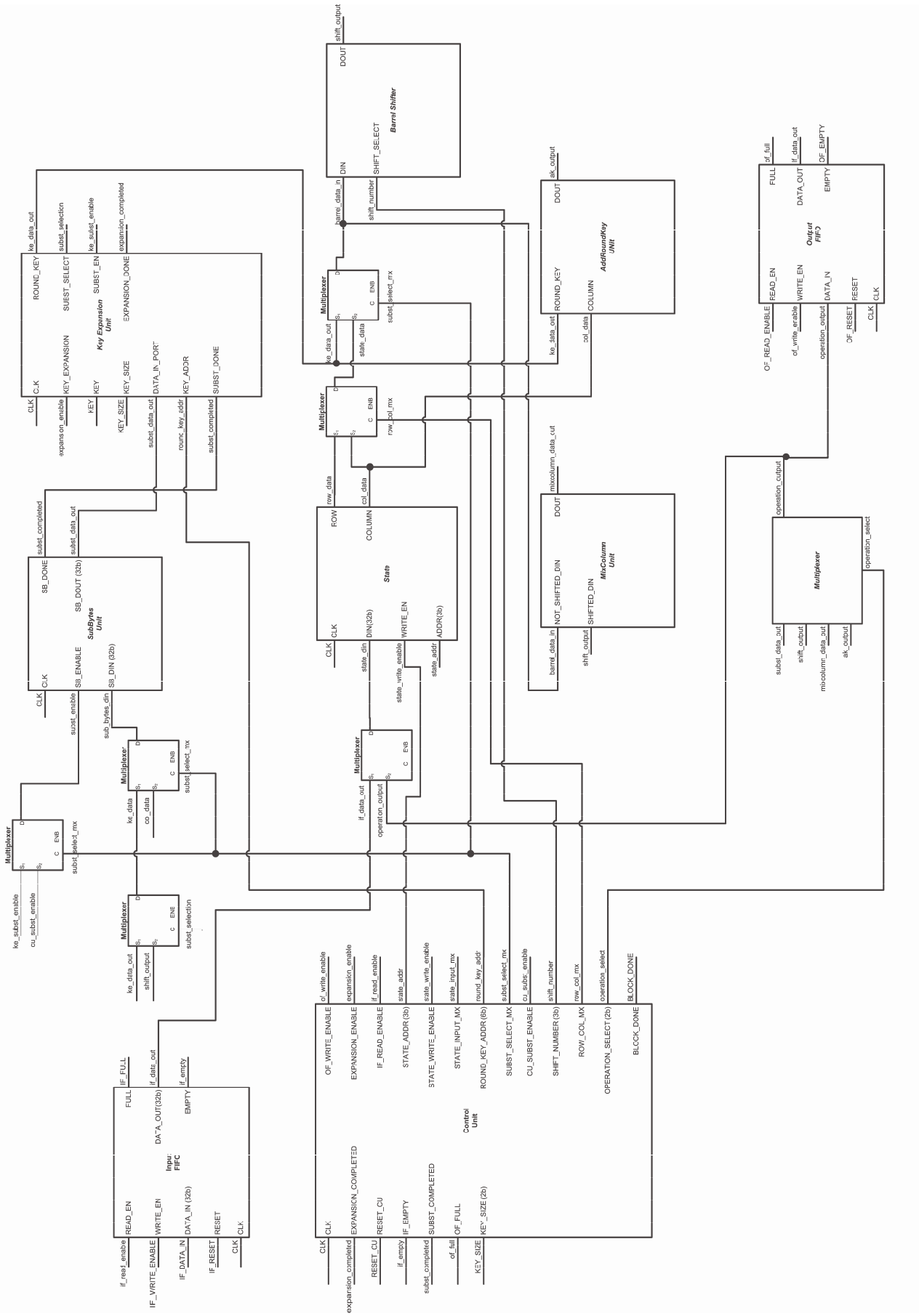
Datové bloky u AES šifry mají 128 bitů, ovšem u rozhraní obvodu vidíme, že datový vstup má pouze 32 bitů. Je to z toho důvodu, že při návrhu bylo nutné vzít v potaz omezující podmínku ve formě určitého počtu dostupných zdrojů, které poskytuje hradlové pole FPGA na FITKitu. U hardwarových realizací AES šifry se obvykle pracuje se vstupních blokem dat buď po bytech anebo po slovech (32bitů). Z toho důvodu má datový vstup právě 32 bitů.

4.1.2 Schéma obvodu a popis

Na obrázku 4.2 je zobrazeno blokové schéma obvodu na nejvyšší úrovni abstrakce. Na obrázku nejsou propojeny všechny signály z důvodu přehlednosti, ale výstupy a vstupy jednotlivých bloků jsou podrobně popsány a neměl by být problém pochopit zapojení tohoto schématu. Signály rozhraní obvodu, které byly popsány v minulé části kapitoly, jsou popsány velkými písmeny. Ostatní signály (vnitřní) jsou malými písmeny. V této části si popíšeme základní datový a řídící tok obvodu spolu s krátkým popisem jednotlivých jednotek. Bližší popis těchto jednotek lze najít v dalších částech této kapitoly.

Pokud bude v dalším textu uveden pojem „aktivace signálu“ nebo „povolení činnosti“, bude tím myšlen přechod signálu do logické úrovně 1 a pojem „deaktivace signálu“ nebo „zakázání činnosti“ bude znamenat přechod signálu do logické úrovně 0. Jádrem celého obvodu je řídící jednotka (angl. Control Unit), jež provádí generování většiny řídících signálů pro ostatní jednotky.

Při startu obvodu se aktivuje signál *expansion_enable*. Tím se provede spuštění jednotky KeyExpansion, která slouží k rozvoji rundovních klíčů. Po skončení rozvoje je aktivován signál *expansion_completed*, který řídící jednotce oznamí, že došlo ke skončení rozvoje. Také dojde k deaktivaci signálu *expansion_enable*.



Obr. 4.2 – Schéma obvodu na nejvyšší úrovni abstrakce

KeyExpansion jednotka používá sdílenou jednotku SubBytes a proto se nastavuje signál *subst_select_mx*, kterým předáme řízení sdílené jednotky SubBytes jednotce KeyExpansion. KeyExpansion jednotka využívá také sdílené jednotky BarrelShifter. Tato jednotka ovšem nemá žádné vstupní řídící signály. Jedná se totiž o kombinační logiku.

Po rozvoji klíče následuje okopírování 4 slov (1 datového bloku) ze vstupní fronty FIFO do stavu. Stav je paměť, nad kterou se dále pak provádí operace algoritmu AES. Okopírování vstupního bloku dat do stavu se provede následovně:

1. Na adresovou sběrnici *state_addr* řídící jednotka vystaví požadovanou adresu sloupce ve stavu a aktivuje na dobu jedné periody hodin signál *if_read_enable*.
2. S následujícím taktem hodinového signálu dojde k vystavení datového slova ze vstupní fronty na vstup stavu. V tomto okamžiku se vystaví na adresovou sběrnici *state_addr* adresa dalšího sloupce a povolení k zápisu do stavu pomocí signálu *state_write_enable*.
3. V dalším taktu pak dojde k zapsání slova do stavu a k načtení dalšího sloupce na vstup stavu.

Tento postup se opakuje pro všechny 4 sloupce. Po načtení těchto hodnot se deaktivuje signál *state_write_enable* a přikročí se k vykonání dalších operací AES algoritmu.

Následuje počáteční operace AddRoundKey. Operace je prováděna jednotkou se stejným názvem v obvodu a úkolem této jednotky je provést XOR operaci mezi stavem a počáteční čtveřicí rundovních klíčů. Postup operace je následující:

1. Na adresovou sběrnici *state_addr* se vystaví adresa prvního sloupce a na adresovou sběrnici *round_key_addr* pak adresa rundovního klíče.
2. V následujícím hodinovém taktu jsou data vystavena na výstupních sběrnicích. V případě stavu, jsou data vystavena na sběrnici *col_data* a v případě rundovního klíče, je hodnota na výstupní sběrnici *ke_data_out* u jednotky KeyExpansion.
3. Tyto výstupy jsou přivedeny na vstupy operace AddRoundKey. V tomto hodinovém cyklu je zároveň zjištěn výsledek, neboť operace je tvořena pouze kombinační logikou. Z tohoho důvodu můžeme aktivovat zápis do stavu (*state_write_enable*) a výsledek uložit.
4. S následujícím taktem hodin se výsledek uloží do stavu a pokračujeme od bodu 1 pro ostatní sloupce.

V počátečním stavu této operace se musí ještě provést nastavení multiplexoru, který vybírá jeden ze čtyř výstupů (výsledků 4 základních operací AES) pomocí signálu *operation_select*. Vybraný výstup se přivádí na sběrnici *operation_output*. Také se musí nastavit signál *state_input_mx*, který vybírá u dalšího multiplexoru, zda se na vstup stavu přivedou data ze vstupní fronty (*if_data_in*), anebo výsledek některé ze 4 operací (*operation_output*). Významy hodnot těchto signálů budou uvedeny v 9. části této kapitoly.

Další činnost obvodu představuje hlavní cyklus algoritmu. Jedná se o posloupnost 4 základních operací, které se na základě velikosti klíče, opakují buď 9-krát pro velikost 128 bitů, 11-krát pro velikost 192 bitů a 13-krát pro velikost 256 bitů.

Jako první operace cyklu se provádí operace SubBytes. V počátečním stavu této operace se musí opět nastavit určitým způsobem některé z multiplexorů, které přepínají potřebné řídící a datové signály. Jedná se opět o výběr jednoho z výstupů operací pomocí signálu *operation_select*. A také se nastavuje signál *state_input_mx*. Nově v této operaci přibývá nastavení signálu *subst_select_mx*. Tento signál je poměrně důležitý, neboť slouží k určení, zda jednotku SubBytes bude aktuálně používat řídící jednotka (Control Unit) nebo jednotka pro rozvoj klíče (KeyExpansion Unit). Postup operace je následující:

1. V počátečním stavu se vystaví na adresovou sběrnici *state_addr* adresa prvního sloupce.
2. V dalším cyklu hodin máme vystaveny data na sběrnici *sub_bytes_din***.** V tomto cyklu povolujeme činnost jednotky SubBytes pomocí signálu *cu_subst_enable.*
3. Po skončení substituce nastaví jednotka SubBytes signál *subst_completed*, kterým oznámí řídící jednotce konec své činnosti. Substituovaná data jsou vystavena na sběrnici *subst_data_out*.
4. Řídící jednotka povolí zápis do stavu *(state_write_enable*) a s dalším hodinovým taktem se výsledná data zapíší. Také deaktivuje signál *cu_subst_enable*, aby se mohla jednotka SubBytes dostat do počátečního stavu.

Tento postup se opakuje stejně pro zbývající 3 sloupce.

Jako další se provádí operace ShiftRows. Jedná se o posun řádků stavu. Každý řádek se posouvá o jiný počet bytů doleva. První řádek se neposouvá vůbec, druhý řádek o jeden byte doleva, druhý řádek pak o dva byty a poslední řádek o tři byty. Ve skutečnosti se nejedná o posuny, ale o rotace. V této operaci se nepracuje s daty po sloupcích, ale po řádcích. K tomuto účelu je přizpůsoben i stav. 3-bitová adresa *state_addr* určuje nejen adresu sloupce, ale i adresu řádku. Jak je zřejmé ze schématu, jsou řádková data vystavována na sběrnici *row_data* a sloupcová na *col_data*. Na začátku této operace nastavíme pomocí 2 multiplexorů vstup jednotky ShiftRows, která je implementována pomocí válcového posouvače (angl. Barrel Shifter). První multiplexor nastavíme pomocí signálu *row_col_mx*, který vybírá, zda vstupem do jednotky bude sloupec nebo řádek. Válcový posouvač je totiž využíván i v operaci MixColumn, kde je potřeba posunout sloupec o jeden bit. Další multiplexor je nastaven pomocí signálu *subst_select_mx*, který vybírá mezi daty ze stavu (tedy buď sloupec nebo řádek) nebo daty z jednotky KeyExpansion. Operace pracuje následovně:

1. Na adresovou sběrnici *state_addr* vystavíme adresu požadovaného řádku a signálem *shift_number* určíme o kolik bytů se budou data posouvat.
2. V dalším hodinovém taktu jsou data vystavena na sběrnici *barrel_data_in* a jelikož je válcový posouvač vytvořen pomocí kombinační logiky, je výsledek připraven v tomto hodinovém cyklu na sběrnici *shift_output*. Proto rovnou povolíme zápis do stavu

(*state_write_enable*). Taky musíme vybrat signálem *operation_select* požadovaný výstup multiplexoru pro zápis do stavu. Stejně tak signál *state_input_mx*.

3. V dalším hodinovém taktu dojde k zapsání dat do stavu.

Pro zbývající řádky pak pokračujeme od kroku jedna, kdy jen měníme hodnotu posuvu, tedy hodnotu signálu *shift_number*.

Dále pak následuje předposlední operace cyklu AES a tou je operace MixColumn. MixColumn operace spočívá v provedení několika matematických operací nad jedním sloupcem stavu. Ke své činnosti využívá jednotku BarrelShifter. Tato jednotka je opět tvořena čistě kombinační logikou a zpracování dat netrvá déle než jeden takt hodin. Na začátku zpracování opět musíme nastavit datové a řídící cesty pomocí multiplexorů. Signálem *row_col_mx* nastavíme zpracování sloupce, signálem *subst_select_mx* pak zpracování stavových dat. Vybereme výstup *mixcolumn_data_out* pro zápis do stavu pomocí signálu *operation_select* a nakonec pomocí signálu *state_input_mx* vybereme, že se do stavu bude zapisovat výsledek operace. Tím máme nastavenou konfiguraci obvodu pro jednotku MixColumns. Postup zpracování dat je následující:

1. Vystavíme na sběrnici *state_addr* adresu sloupce ve stavu.
2. V dalším taktu jsou data vystavena a zároveň zpracována jednotkou MixColumns. Výsledek zpracování je dostupný na sběrnici *mixcolumn_data_out*. V tomto cyklu hodin také nastavíme povolení zápisu do stavu (*state_write_enable*).
3. Výsledná data jsou zapsána do stavu.

Stejně jako u předchozích operací se tento postup opakuje i pro ostatní sloupce.

Zbývá poslední operace a tou je operace AddRoundKey. Jedná se o provedení operace XOR mezi sloupcem stavu a mezi odpovídajícím rundovních klíčem. Jednotka AddRoundKey je tedy opět tvořena kombinační logikou. Nastavení multiplexoru je v tomto případě poměrně jednoduché, stačí nastavit pomocí signálu *operation_select* výstup operace (*ak_output*) na vstup stavu. Také je potřeba nastavit signál *state_input_mx* pro výběr mezi mezi výsledkem operace a daty ze vstupní fronty. Postup zpracování je pak následující:

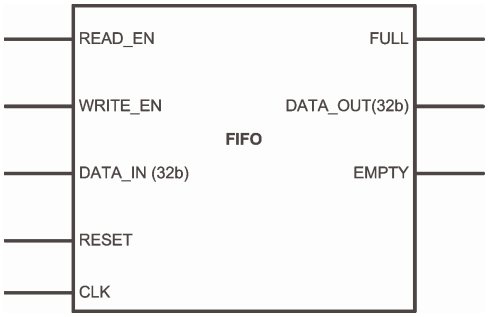
1. Na adresovou sběrnici *state_addr* vystavíme adresu sloupce ve stavu.
2. V dalším taktu jsou data vystavena a zároveň zpracována jednotkou. Výsledek je vystaven na sběrnici *ak_output*. V dalším taktu dojde k zápisu výsledku a proto musíme povolit signál zápisu do stavu (*state_write_enable*).
3. Data jsou zapsána do stavu. Proceduru opakujeme od kroku 1 pro ostatní sloupce.

Po skončení základního cyklu algoritmu je provedena ještě jedna iterace, ovšem s vynecháním operace MixColumns. Po skončení posledního cyklu je aktivován signál *block_done* na dobu jedné periody hodinového signálu. Nakonec přechází řídící jednotka do stavu, který odpovídá skončení expanze klíče a dochází ke kopírování nových dat do stavu a celý cyklus se opět opakuje.

V této části byla popsána základní činnost obvodu. Bližší detaily ohledně fungování jednotlivých jednotek lze najít v dalších částech této kapitoly.

4.2 Jednotka FIFO

Na obrázku 4.3 si můžeme prohlédnout základní rozhraní jednotky FIFO (angl. First In First Out). Výraz FIFO popisuje princip chování paměti, kdy data, která budou zapsána jako první na řadě, opustí tuto paměť při čtení taky jako první.

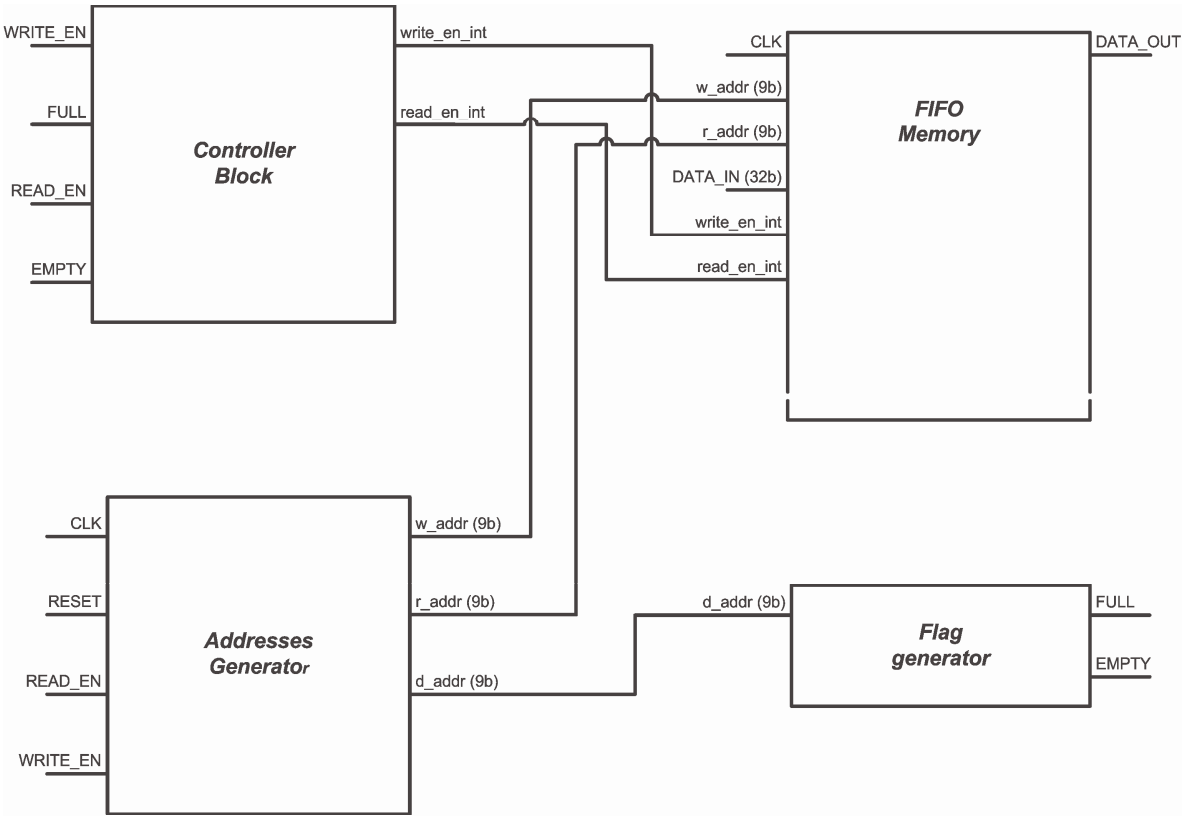


Obr. 4.3 – Základní rozhraní jednotky FIFO

V programování se jedná o abstraktní datový typ (typ fronty) pro ukládání dat. V navrženém obvodu se paměť FIFO vyskytuje hned na dvou místech. V prvním případě se jedná o frontu vstupních dat, která jsou určena ke zpracování (zašifrování). V druhém případě se jedná o paměť pro uložení zašifrovaných dat. Vnější jednotka, která pracuje s obvodem AES vlastně používá při zápisu dat do obvodu právě první výskyt FIFO paměti. Při čtení dat pak druhý výskyt paměti. Význam signálů rozhraní FIFO (na obr. 4.3) je následující:

- Signál *READ_EN* slouží pro aktivaci čtení dat, jež budou s dalším hodinovým taktem vystavena na sběrnici *DATA_OUT*.
- Signál W*RITE_EN* slouží pro povolení zápisu dat do obvodu. V dalším hodinovém taktu pak budou zapsána data, jež jsou vystavena na sběrnici *DATA_IN*.
- Signál *RESET* slouží k vymazání obsahu vnitřní paměti FIFO. Nejedná se přímo o smazání obsahu paměti, ale o úpravu vnitřních adresových čítačů. Kdybychom chtěli mazat i obsah paměti, nebylo by pak možné využít při syntéze obvodu komponentu BlockRam a paměť by byla vytvořena pomocí distribuované paměti v technologii FPGA. To by ovšem znamenalo velkou spotřebu prvků cílové technologie a obvod by se nevešel do typu FPGA, které je součástí platformy FITKit.
- *CLK* – hodinový signál k synchronizaci paměti FIFO.
- *FULL* – jedná se o informační signál, který udává, zda je paměť plná (log. úroveň 1) či nikoli (log. úroveň 0).
- *EMPTY* – jedná se o komplement signálu *FULL*, který říká, zda je paměť prázdná (log. úroveň 1) či nikoliv (log. úroveň 0).

Na obrázku 4.4 si můžeme prohlédnout vnitřní schéma této jednotky. Poté následuje popis.



Obr. 4.4 – vnitřní schéma jednotky FIFO

Jak je patrno z obrázku, je blok FIFO složen celkem ze čtyř částí:

1. První část tvoří řídící blok (Controller block), který generuje vnitřní povolovací signály pro čtení a zápis do paměti. Tato jednotka slouží jako ochrana před nechtěným zápisem do plné paměti a taky před čtením prázdné paměti. Na základě vstupního signálu *WRITE_EN* a příznaku *FULL* aktivuje vnitřní signál *write_en_int*. To znamená, že pokud je signál *WRITE_EN* v hodnotě logické 1 a příznak *FULL* v hodnotě logické 0, pak se vnitřní signál aktivuje a povolí se zápis do paměti (na obrázku 4.4 - FIFO Memory). Podobně u čtecího signálu – pokud je signál *READ_EN* v logické hodnotě 1 a příznak *EMPTY* v logické 0, pak se vnitřní signál *read_en_in* aktivuje a povolí čtení z paměti.
2. Další blok představuje generátor čtecích a zápisových adres pro paměť (Addresses generator). Blok obsahuje 3 vnitřní čítače , první pro uchování čtecí adresy (*r_addr*), druhý pro úschovu zapisovací adresy (*w_addr*) a poslední čítač uchovává aktuální počet položek, které se nacházejí v paměti (*d_addr*). Při aktivním vstupním signálu *WRITE_EN* a za podmínky, že paměť není plná, se s dalším hodinovým taktem zvýší hodnota čítačů w_addr a d_addr o hodnotu 1. Podobně při aktivním vstupním signálu *READ_EN* a za podmínky, že paměť není prázdná, dojde s dalším hodinovým taktem ke zvýšení hodnoty čítače r_addr a ke snížení čítače d_addr o hodnotu 1. Je možná překvapující, že dojde ke zvýšení hodnoty

r_addr. Je to z toho důvodu, že je paměť implementována jako kruhový buffer. To znamená, že při dosažení konce paměti se začíná zapisovat od jejího začátku, pokud není paměť plná.

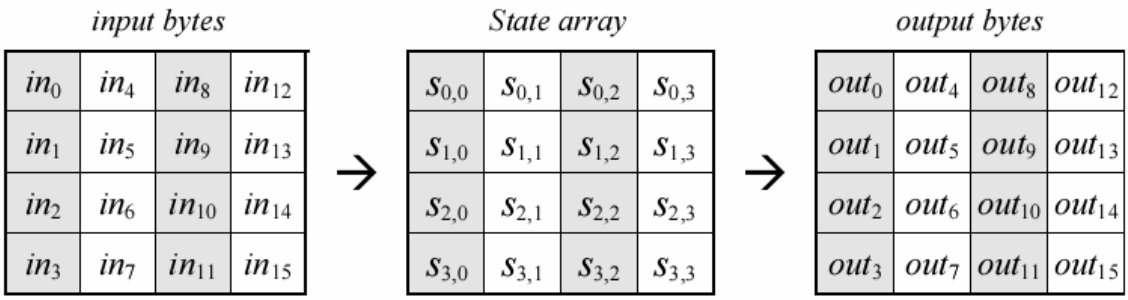
3. Blok FIFO Memory - jedná se o samotnou paměť pro úschovu hodnot. Paměť je implementována jako dvou-portová. To znamená, že obsahuje samostatný port pro zápis dat a další samostatný port pro čtení dat. Paměť má velikost 512 položek po 32 bitech. V jazyce VHDL byla paměť označena, aby při syntéze paměti bylo použito komponenty BlockRam. Popis rozhraní paměti je samo vysvětlující, ale pro úplnost jej uvedeme. Při aktivním signálu *write_en_int* dojde s dalším hodinovým taktem k zápisu dat, které jsou vystaveny na sběrnici *DATA_IN,* na adresu paměti, která je dána čítačem *w_addr*. Podobně u čtení – při aktivním signálu *read_en_int* dojde s dalším hodinovým taktem ke čtení dat z adresy, která je dána čítačem r_addr, a jejich vystavení na sběrnici *DATA_OUT*.
4. Poslední blok (Flag Generator) slouží ke generování příznaků, zda je paměť plná, popř. prázdná. Příznaky se generují na základě čítače d_addr, který obsahuje aktuální počet položek v paměti. Generátor příznaků je vlastně komparátor, který zjišťuje, zda čítač d_addr obsahuje pouze samé nuly, popř. samé jedničky. V případě pouze nul se aktivuje signál *EMPTY* a případě samých jedniček signál *FULL*.

4.3 Jednotka správy dat - stav

Jedná se o paměť, nad kterou probíhají operace AES algoritmu. Paměť je uspořádaná do dvourozměrné mřížky o velikosti 4 x 4 bytů. Na začátku zpracování datového bloku jsou tyto data okopírována do stavu. Dále pak probíhá postupně načítání a ukládání jednotlivých sloupců (popř. řádků) podle potřeby jednotlivých operací. Na závěr jsou pak zašifrovaná data okopírována ze stavu do výstupní fronty FIFO. Na obrázku 4.5 si můžeme prohlédnout jakým způsobem probíhá okopírování vstupních dat do stavu a ze stavu do výstupní fronty. Pokud číslo řádku označíme proměnnou r v rozsahu $0 \leq r < 4$ a číslo sloupce proměnnou c v rozsahu $0 \leq c < 4$, pak můžeme okopírování dat do stavu a ze stavu popsat vzorci:

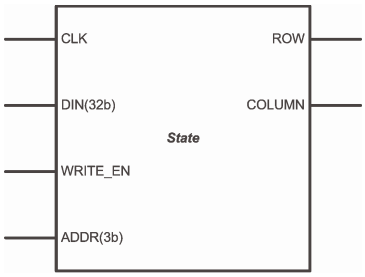
$$state[r, c] = input[r + 4*c]$$

$$output[r + 4*c] = state[r, c]$$



Obr. 4.5 – Znázornění kopírování dat do a ze stavu

Na obrázku 4.6 je zobrazeno základní rozhraní stavu:



Obr. 4.6 – Rozhraní stavu

Význam signálů u stavu je v podstatě podobný jako u jiných typů pamětí:

- *CLK* – s nástupnou hranou hodinového signálu dojde k uložení hodnoty ze vstupní sběrnice *DIN* do paměti, popř. k načtení dat na jednu z výstupních sběrnic.
- *WRITE_EN* – signál povolující zápis do stavu
- *ADDR* – u tohoto signálu je dobré se chvíli pozastavit a popsat si jednotlivé hodnoty, kterých může adresa nabývat: hodnoty 000 až 011 určují adresy prvního až čtvrtého sloupce, hodnoty 101 až 111 pak adresy druhého až čtvrtého řádku. Hodnota 100 není využita, neboť není potřeba adresovat první řádek. Jediná operace, která využívá řádkového přístupu je operace ShiftRows a ta s prvním řádkem nepracuje.
- Na sběrnici *ROW* se vystavují adresované řádky a na sběrnici *COLUMN* pak sloupce.

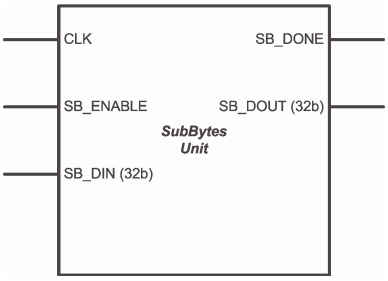
Obr. 4.7 – Zápis 4 vstupních slov do stavu

Na obrázku 4.7 je ukázka průběhu signálů při zápisu vstupních dat, které se nacházejí na sběrnici *DIN,* do stavu a jejich opětovném načtení. Prvně se data načtou po sloupcích a v druhém případě po řádcích. První řádek se nečte, neboť k tomu stav není uzpůsoben a ani v algoritmu AES není potřeba tento řádek číst.

Jednotka správy dat (stav) je v FPGA implementován pomocí distribuované paměti.

4.4 Jednotka SubBytes

Jednotka SubBytes realizuje nelineární bytovou substituci, která operuje nezávisle nad každým bytem stavu pomocí substituční tabulky S-Box. Data do této jednotky nahráváme po sloupcích, stejně tak je po sloupcích z této jednotky čteme. Na obrázku 4.7 si můžeme prohlédnout rozhraní této jednotky.



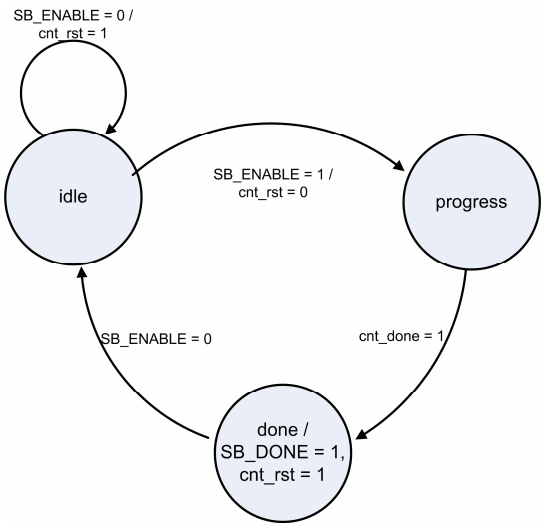
Obr. 4.7 – Rozhraní jednotky SubBytes

Význam jednotlivých signálů je opět samo vysvětlující, ale pro úplnost uvedeme jejich význam:

- *CLK* – hodinový signál pro synchronizaci obvodu
- *SB_ENABLE* – slouží pro aktivaci jednotky. Po aktivaci tohoto signálu dojde v dalším hodinovém taktu k započetí substituce vstupních dat ze sběrnici *SB_DIN*.
- *SB_DONE* je signál, který informuje o skončení činnosti jednotky. Při aktivaci tohoto signálu jsou na sběrnici SB_DOUT vystavena substituovaná data.

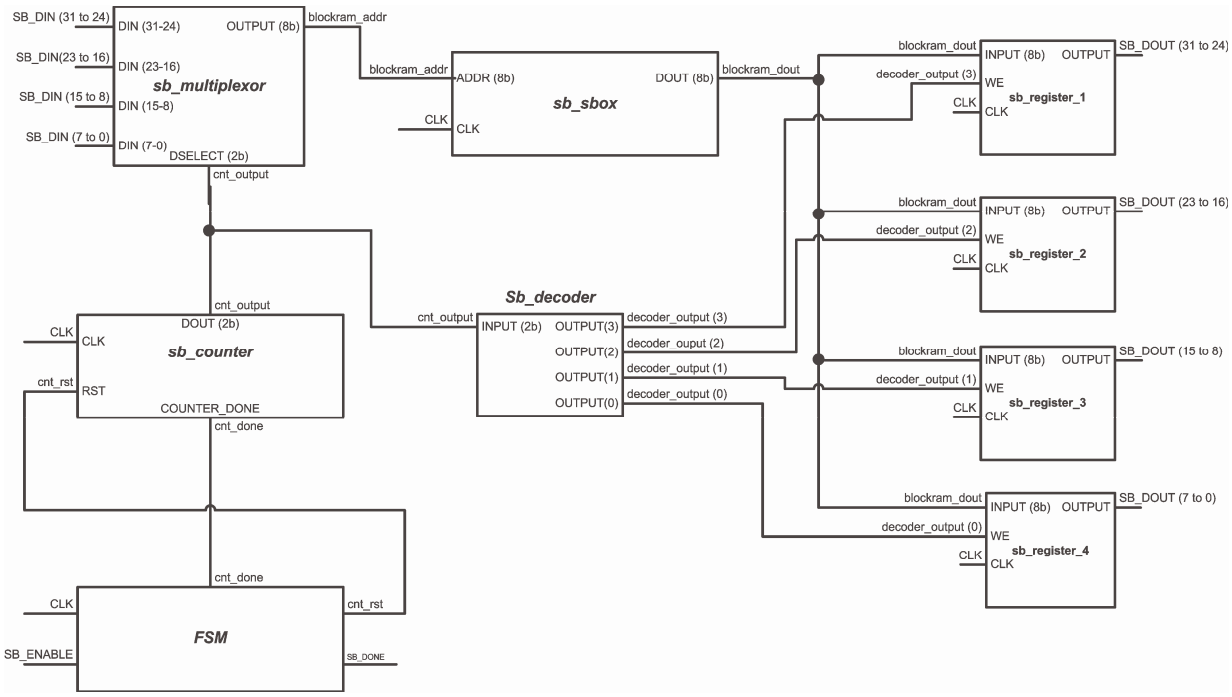
Na obrázku 4.9 si můžeme prohlédnout základní schéma jednotky SubBytes. Celá jednotka je řízena konečným stavovým automatem (na obrázku blok označený FSM). Chování každého konečného stavového automatu lze vyjádřit pomocí přechodového diagramu, popř. přechodové tabulky. Následující diagram na obrázku 4.8 popisuje chování řídícího automatu jednotky SubBytes. Jedná se o kombinaci Mealyho a Moorova automatu.

Obvod se za nečinného stavu nachází ve stavu *IDLE*, kdy dochází s každým hodinovým taktem k nastavení čítače (*cb_counter*) na počáteční hodnotu. Při aktivaci jednotky pomocí signálu *SB_ENABLE* dojde k přechodu automatu do stavu *PROGRESS*, kdy dojde k přerušení resetu čítače a v následujících taktech bude čítač pracovat standardním způsobem. Ve stavu *PROGRESS* je s každým hodinovým taktem zvýšena hodnota čítače o 1. Výstup čítače tvoří selekční signál (*cnt_output*) pro multiplexor, který vybírá jeden ze 4 bytů vstupního slova. Tento byte se pak pomocí sběrnice *blockram_addr* dostává na adresový vstup s-boxu. S dalším hodinovým taktem je na výstupu s-boxu (*blockram_dout)* k dispozici substituovaný byte. Tento byte je pak přiveden na vstupní datové sběrnice 8-bitových záchytných registrů. Do jednoho z těchto registrů se v dalším hodinovém taktu zapíše aktuální substituovaný byte.



Obr. 4.8 – Stavový přechodový diagram FSM

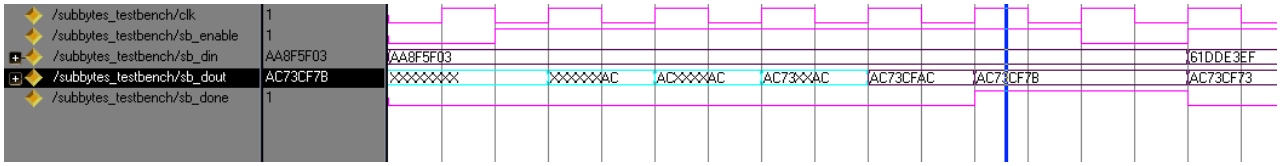
Do kterého registru bude proveden zápis, je určeno dekodérem (sb_decoder), jež má na vstupu výstup čítače (2 bity). Výstup dekodéru tvoří 4 bity, jeden bit pro každý registr. Bity jsou přivedeny k registrům na jejich povolovací vstupy pro zápis.



Obr. 4.9 – Schéma jednotky SubBytes

S dalšími 3 hodinovými takty dojde stejným způsobem k substituci dalších zbývajících 3 bytů. Po skončení substituce, jež je indikováno přetečením čítače *cnt_done*, dojde k přechodu řídícího automatu do stavu *DONE* a aktivaci informačního signálu *SB_DONE*. Tento signál upozorní hlavní řídící jednotku na fakt, že na sběrnici je již vystaveno kompletní substituované slovo. Řídící jednotka

pak v dalším hodinovém taktu deaktivuje signál *SB_ENABLE*, čímž informuje jednotku SubBytes o tom, že data byla převzata. Řídící automat jednotky SubBytes následně přechází do počátečního stavu *IDLE*. Na obrázku 4.9 si můžeme prohlédnout průběh signálů při substituci jednoho slova. Jednotka má na vstupu hexadecimální hodnotu *AA8F5F03* a na výstupu substituovanou hodnotu *AC73CF7B*.



Obr. 4.9 – průběh signálů při substituci

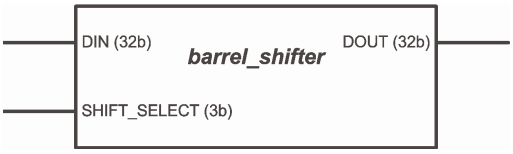
Na závěr je potřeba poznamenat, že paměť, ve které jsou uloženy substituční hodnoty, je realizována pomocí komponenty BlockRAM cílové technologie FPGA. Další důležité poznatky: s-box je realizován pomocí paměti, která je tvořena 256 položkami o 8 bitech. S-box funguje tak, že při vystavení 8-bitového čísla na jeho vstup, dojde k vystavení substituovaného byte v dalším taktu hodinového signálu. Vstupní byte tedy tvoří adresu do paměti s-box.

4.5 Jednotka ShiftRows

Jednotka ShiftRows představuje implementaci stejnojmenné operace. Slouží k cyklickému posuvu posledních tří řádků stavu o různý počet bytů. První řádek není posouván. Posun řádků ve stavu pracuje podle následující rovnice:

$$s'_{r,c} = s_{r,(c + shift(r)) \bmod 4}$$

kde *r* je index řádku v rozsahu $0 < r < 4$ a *c* je index v rozsahu $0 \leq c < 4$. Hodnota operace shift(r) závisí na hodnotě r. Platí, že shift(1) = 1, shift(2) = 2, shift(3) = 3. Definiční obor operace shift je definován pouze těmito třemi čísly. Na obrázku 4.10 si můžeme prohlédnout rozhraní této jednotky.



Obr. 4.10 – Rozhraní jednotky ShiftRows

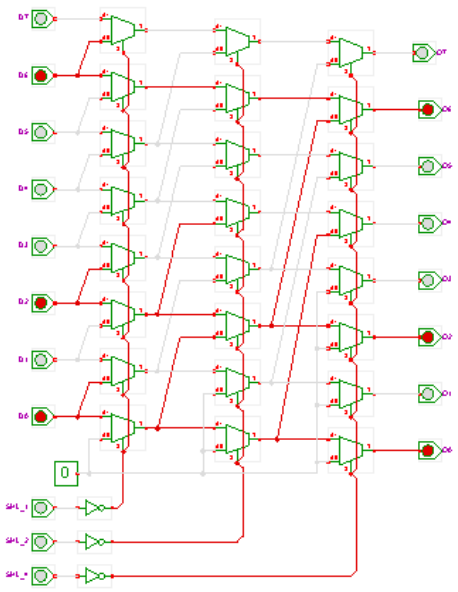
Význam jednotlivých signálů je následující:

- Na sběrnici *DIN* se vystavují vstupní data a na sběrnici *DOUT* jednotka vystavuje výsledná data
- Pomocí signálu *SHIFT_SELECT* vybíráme o kolik bitů se budou vstupní data posouvat. Signál může nabývat následujících hodnot:

- 1XX – vstupní data budou posouvána při této hodnotě o jeden bit doleva. Každý byte se posouvá samostatně a nedochází k posuvu nejvyššího bitu do následujícího bytu. Tento posuv je využíván jednotkou MixColumns, která na svém jednom vstupu potřebuje posunutý sloupec a na druhém vstupu stejný sloupec, ale v neposunutém tvaru.
- X1X – vstupní data budou při aktivní hodnotě prostředního bitu posunuty (přesněji rotovány) o jeden byte doleva, tedy o 8 bitů.
- XX1 – při aktivní hodnotě nejméně významného bitu budou data posunuta (opět rotovány) o 2 byty doleva, tedy o 16 bitů.

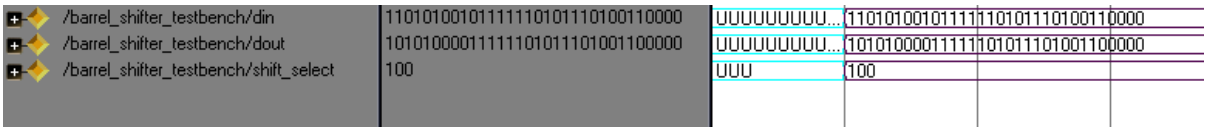
Hodnoty vstupního signálu *SHIFT_SELECT* lze různě kombinovat (sčítat posuvy) a tak určit i jiné hodnoty posuvu než výše uvedené. Například při hodnotě 011 dojde k posuvu o 3 byty (tedy 24 bitů).

Jednotka ShiftRows je realizována pomocí upraveného válcového posouvače (angl. barrel shifter). Válcový posouvač realizuje posuv vstupních dat v jednom hodinovém taktu. Válcový posouvač v jednotce ShiftRows je realizován jako sekvence tří řad multiplexorů, kdy v každé řadě je 32 multiplexorů. To odpovídá počtu bitů v jednom datovém slovu. Výstup jednoho multiplexoru je vyveden na vstup dalšího multiplexoru v následující řadě. První řada upraveného válcového posouvače posouvá vstupní slovo buď o 0 nebo 1 bit. Ovšem jak již bylo popsáno, v této řadě multiplexorů se posouvá každý byte samostatně. To znamená, že na pozici nejméně významného bitu se nasouvá nula a nejvyšší významový bit není přenášen do následujícího bytu. Selekční signál multiplexorů této řady odpovídá nejvyššímu bitu vstupního signálu *SHIFT_SELECT*. Na vstupní datový signál u multiplexorů, který odpovídá selekční hodnotě 0, je přiveden bit vstupu se stejným pořadovým indexem. Na druhý datový signál je přiveden bit vstupu s indexem o 1 nižší, popř. nulová hodnota u nejméně významného bitu. Právě tento fakt zaručuje rotaci jednoho bitu doleva při aktivním selekčním signálu. Zbývající dvě řady fungují analogickým způsobem, ovšem zde se provádí rotace, která se týká celého datového slova, a neplatí samostatné zpracování po bytech jako tomu bylo u první řady. Na jeden datový vstup multiplexoru je přiveden výstup předchozího multiplexoru se stejným pořadovým indexem. Na druhý vstup pak výstup multiplexoru s nižším pořadovým indexem (popř. i vyšším v případě počátečních indexů). Tento pořadový index odpovídá hodnotě posunu. Druhá řada rotuje o 8 bitů doleva a na selekční vstupy multiplexorů je přiveden prostřední bit signálu *SHIFT_SELECT*. Poslední třetí řada rotuje o 16 bitů doleva a na selekční vstupy je tentokrát přiveden nejméně významný bit signálu *SHIFT_SELECT*. Na následujícím obrázku 4.11 si můžeme prohlédnout ilustraci klasického 8-bitového válcového posouvače.



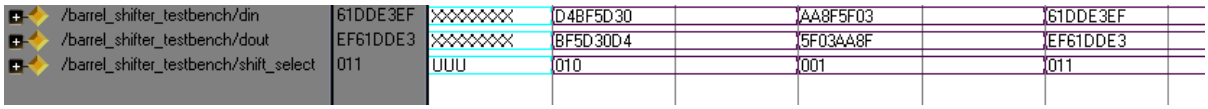
Obr. 4.11 – Klasický 8-bitový válcový posouvač

Na následujícím obrázku 4.12 si můžeme prohlédnout rotaci datového slova o jeden bit při simulaci jednotky. Tuto rotaci využívá jednotka MixColumn.



Obr. 4.12 – Jednotka ShiftRows - posuv o jeden bit doleva

Na následujícím obrázku 4.13 je pak zachycena rotace datového slova o jeden, dva a tři byty při simulaci jednotky. Toho je využíváno k implementaci operace ShiftRows.



Obr. 4.13 – ShiftRows – rotace o jeden, dva a tři byty doleva

4.6 Jednotka MixColumns

Operace MixColumn se provádí po sloupcích. Každý sloupec je v této operaci brán jako čtyřčlenný polynom nad Galoisovým polem (popsáno v části 2.8). Tyto sloupce jsou násobeny fixním polynomem $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}x$. Hodnoty v závorkách jsou v hexadecimálním tvaru. Toto násobení může být popsáno pomocí násobení matic, což nám dá následující rovnice:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

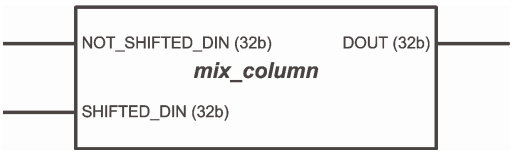
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Násobení hodnotou {02} může být na bytové úrovni provedeno jako posuv o jeden bit doleva následovaný podmíněnou operaci XOR s hodnotou {1B}. Operace XOR bude provedena právě tehdy, pokud neposunutý byte má v nejvíce významném bitu hodnotu logické 1. Násobení hodnotou {03} lze rozepsat následujícím způsobem: {value} • {03} = {value} • ({01} ⊕ {02}) = {value} ⊕ {value}• {02}. To znamená, že operaci MixColumns lze vcelku snadno provést na bytové úrovni jen pomocí posuvů a operací XOR. V následujícím kódu jazyka C je popsána procedura pro výpočet operace MixColumns. Jak si můžeme všimnout, jedná se o docela jednoduchý program. Na základě tohoto kódu pak byla vytvořena optimalizovaná jednotka MixColumns.

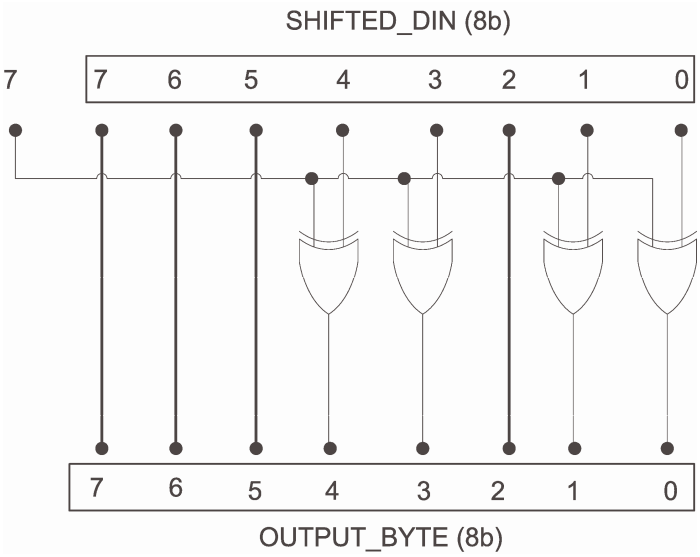
```
void gmix_column(unsigned char *r) {
    unsigned char a[4];
    unsigned char b[4];
    unsigned char c;
    unsigned char h;
    for(c=0; c<4; c++) {
      a[c] = r[c];
      h = r[c] & 0x80; /* high bit */
      b[c] = r[c] << 1;
      if(h == 0x80)
           b[c] ^= 0x1b; /* Rijndael's Galois field */
      }
    r[0] = b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1];
    r[1] = b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2];
    r[2] = b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3];
    r[3] = b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0];
}
```

Tato jednotka má rozhraní, které si můžeme prohlédnout na obrázku 4.14. Na sběrnici *NOT_SHIFTED_DIN* se přivádí sloupec stavu, nad kterým proběhne operace MixColumns. Na sběrnici *SHIFTED_DIN* jsou pak stejná data jako v prvním případě, ale každý byte tohoto datového slova je posunut o jeden bit doleva nezávisle na ostatních bytech slova. Tento posun zajišťuje jednotka ShiftRows, na jejíž vstup je přivedeno stejné datové slovo jako na sběrnici *NOT_SHIFTED_DIN*. Na sběrnici *DOUT* je pak vyveden výstup.



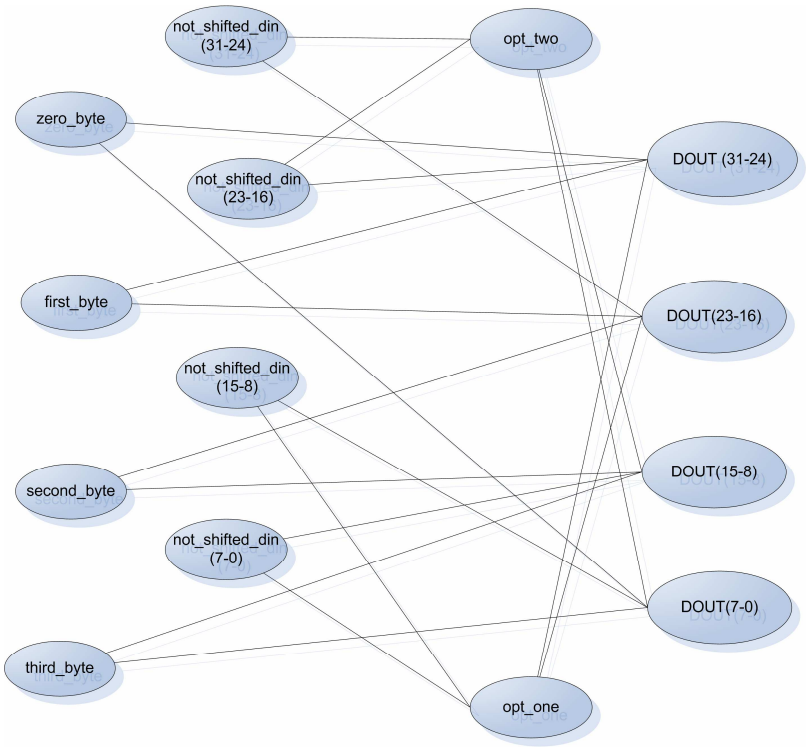
Obr. 4.14 – rozhraní jednotky MixColumns

Jednotka MixColumn je tvořena pouze kombinační logikou a proto je výsledek operace znám s následujícím hodinovým taktem. V prvním kroku operace MixColumn musíme provést operaci posunu o jeden bit. Tuto práci ovšem již vykoná jednotka ShiftRows. My se pak musíme postarat o podmíněnou operaci XOR. Ta je v každém bytu slova prováděna tak, jak je zobrazeno na následujícím obrázku 4.15.



Obr. 4.15 – podmíněná operace XOR

Na obrázku můžeme vidět 8 bitů, které představují jeden z posunutých bytů vstupního slova. Nejlevější bit, označený číslicí 7 představuje nejvyšší bit z neposunutého bytu. Právě hodnota tohoto bitu je určující podmínka, zda se provede podmíněná operace XOR. Operace XOR probíhá mezi vstupní hodnotou bytu a mezi hodnotou 1B (hexadecimálně), která v binárním zápisu má následující tvar: 00011011. Vstupní bity s indexem sedm až pět a s indexem dva jsou vyvedeny přímo na výstup a to z toho důvodu, že výsledek operace XOR při vstupních hodnotách 0 a X bude vždy X (0 nebo 1, podle zadané hodnoty). Dojde vlastně k okopírování vstupu na výstup. Co se týče bitů na indexech čtyři, tři, jedna a nula, zde dochází k provedení operace XOR mezi bitem s daným indexem a nejvyšším bitem neposunutého byte. Ten svou hodnotou určuje, zda dojde k provedení podmíněné operace XOR (hodnota bitu je rovna 1) či nikoliv (hodnota bitu je 0). Dalším krokem je provedení operací XOR mezi jednotlivými byty neposunutého datového slova a mezi byty upravenými pomocí posuvu a podmíněné operace XOR. Vykonává se tedy druhá část kódu v jazyce C, který je vypsán na stránce č. 61. Schéma v jakém pořadí probíhají operace XOR je zobrazeno na následujícím obrázku 4.16.

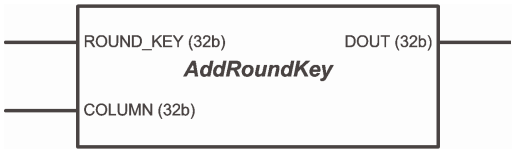


Obrázek 4.16 – Způsob zpracování druhého kroku MixColumn

Každý prvek (ovál) zobrazený v obrázku 4.16 znamená jeden byte. Hodnoty *zero_byte, first_byte, second_byte, third_byte* jsou byty, které vznikly posunutím a podmíněnou operací XOR (viz první kroky operace MixColumn). Dále pak byty s označením *not_shifted_din* jsou byty vstupního datového slova ze sběrnice *NOT_SHIFTED_DIN*. Byty s označením *opt_one* a *opt_two* představují optimalizaci v návrhu této jednotky. Díky ní dojde k úspoře 16 dvou-vstupových multiplexorů. *DOUT* nakonec popisuje byty výstupního datového slova. Každý uzel v tomto schématu, do nějž vstupují jiné prvky, představuje operaci XOR vstupních prvků. Zpracování probíhá zleva doprava, tak jak je zobrazeno na schématu - vstupy jsou nalevo, výstupy napravo.

4.7 Jednotka AddRoundKey

Jednotka AddRoundKey představuje stejnojmennou operaci. V této operaci se provádí XOR stavu s odpovídajícími rundovními klíči. Tato jednotka je velice jednoduchá a jejím jediným úkolem je provádět operaci XOR jejích 2 datových vstupů. Rozhraní této jednotky lze vidět na následujícím obrázku 4.17.

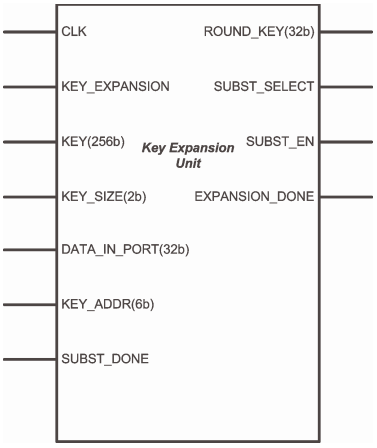


Obr. 4.17 – Rozhraní jednotky AddRoundKey

Na sběrnici *ROUND_KEY* je přiváděn rundovní klíč, který je uložen v jednotce KeyExpansion. Tato jednotka bude popsána v následující části kapitoly. Na sběrnici *COLUMN* je přiváděn sloupec stavu. Jelikož je jednotka tvořena opět kombinační logikou, bude výsledek operace k dispozici v příštím hodinovém taktu na sběrnici *DOUT*. VHDL popis jednotky je uložen v souboru *add_round_key.vhd*.

4.8 Jednotka KeyExpansion

V této části kapitoly si popíšeme jednotku, která slouží k provedení expanze klíče a taky jako paměť pro uchování rundovních klíčů. Algoritmus expanze byl detailně popsán v části 2.13 a proto přímo přistoupíme k popisu jednotky. Na obrázku 4.18 si můžeme prohlédnout rozhraní jednotky.



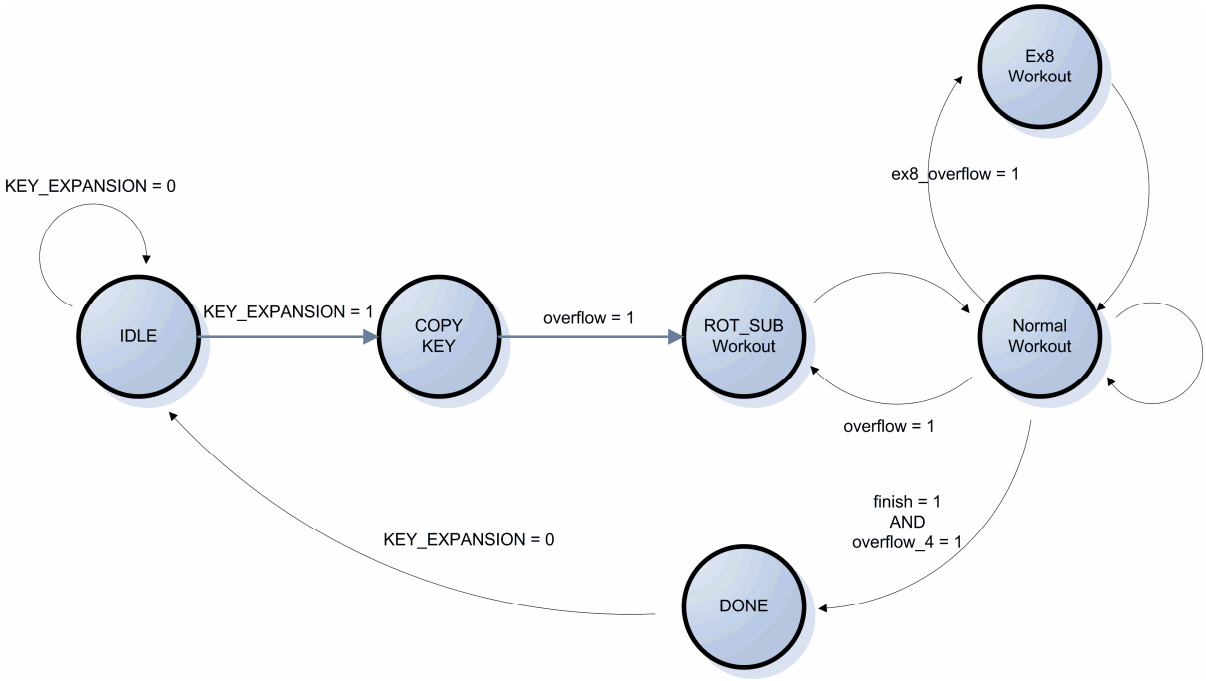
Obr. 4.18 – rozhraní jednotky KeyExpansion

Význam signálů je následující:

- *CLK* – hodinový signál pro synchronizaci jednotky
- *KEY_EXPANSION* je signál, který slouží k aktivaci expanze klíče
- Pomocí signálu *KEY_SIZE* je určeno, jak velký klíč je vystaven na sběrnici *KEY*
- *DATA_IN_PORT* sběrnice slouží pro vstup dat z jednotky SubBytes. KeyExpansion totiž využívá ke své činnosti jednotek ShiftRows a SubBytes.
- *KEY_ADDR* je adresová sběrnice, která slouží pro adresaci vnitřní paměti jednotky. Pokud zrovna jednotka není ve stavu expanze klíče, dojde v dalším hodinovém taktu k vystavení odpovídajícího rundovního klíče na sběrnici *ROUND_KEY*.
- Signál *SUBST_DONE* informuje jednotku KeyExpansion o skončení činnosti jednotky SubBytes.
- Signál *SUBST_SELECT* slouží k přepnutí vstupu jednotky SubBytes. Bude vysvětleno dále v této sekci.
- Signál *SUBST_ENABLE* slouží pro aktivaci jednotky SubBytes.

- Signál *EXPANSION_DONE* slouží pro informování řídící jednotky hlavního obvodu o skončení expanze klíče.

Pro další vysvětlení činnosti jednotky je potřeba si prohlédnout její schéma na obrázku 4.20. Jednotka KeyExpansion je nejsložitější ze všech jednotek v návrhu obvodu. Vnitřní činnost jednotky je řízena, stejně jako v případě jednotky SubBytes, konečným stavovým automatem (na obrázku označen jako FSM – Control Unit). Abychom pochopili funkčnost jednotky, postačí když si vysvětlíme činnost FSM. Zjednodušený přechodový diagram automatu je zobrazen na obrázku 4.19.



Obr. 4.19 – Přechodový diagram FSM u jednotky KeyExpansion

V počátečním, nebo také klidovém stavu (stav *IDLE*) slouží jednotka KeyExpansion jako paměť rundovních klíčů. To znamená, že vnitřní multiplexory jednotky jsou nastaveny tak, že při vystavení adresy na sběrnici *KEY_ADDR* dojde v dalším hodinovém taktu k vystavení odpovídajícího rundovního klíče na sběrnici *ROUND_KEY*. Při aktivaci signálu *KEY_EXPANSION* dojde k přechodu automatu do stavu *COPY KEY*. V tomto stavu dojde k okopírování vstupního klíče do vnitřní paměti. Počáteční rundovní klíče jsou tvořeny právě bity vstupního klíče. Vstupní klíč je rozdělen pomocí multiplexoru na jednotlivá slova a ty se pak postupně uloží do vnitřní paměti. V tomto stavu je s každým hodinovým taktem zvýšena hodnota adresové sběrnice *write_addr*, jejíž tři nejméně významné bity jsou napojeny na selekční vstup multiplexoru *key_part_mx*, který rozděluje klíč na jednotlivá slova. Z tohoto faktu vyplývá, že s každým hodinovým taktem dojde k zápisu jednoho slova klíče do paměti. Multiplexor, který rozděluje klíč na jednotlivá slova není na obrázku 4.20 vyobrazen, neboť pro něj již nezbyl prostor.

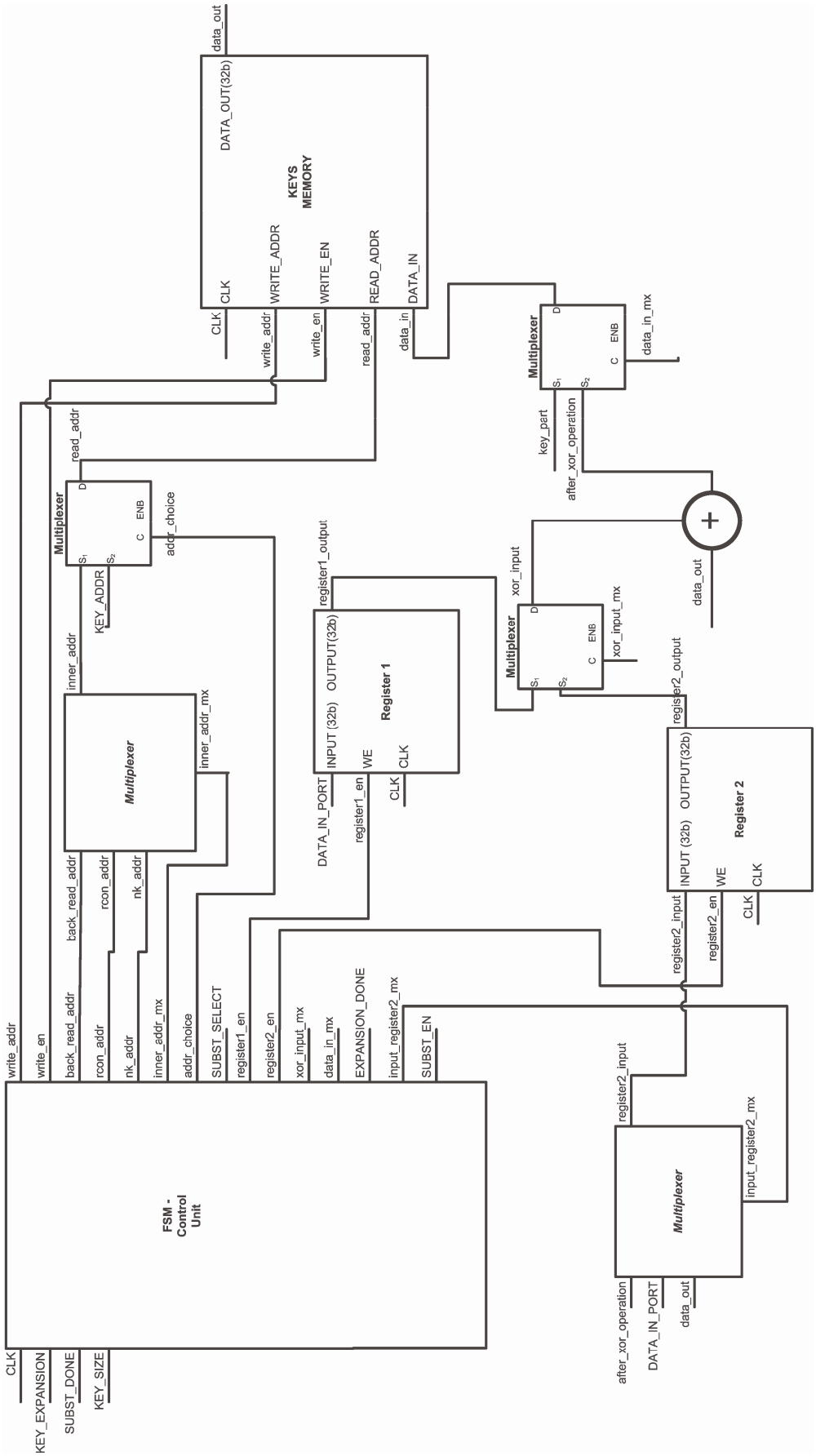
Počet slov se u různých velikostí klíče samozřejmě liší. U velikosti klíče 128 bitů jsou to 4 slova, u velikosti 192 bitů 6 slov a u velikosti 256 bitů pak 8 slov. Abychom mohli sjednotit přechod ze stavu *COPY_KEY* do následujícího stavu pro různé velikosti klíče, obsahuje řídící automat 3 vnitřní čítače s názvy *counter_4, counter_6, counter_8*, které čítají do dané velikosti. Při přetečení čítačů je generováno signály *overflow_4, overflow_6, overflow_8* odpovídající přetečení. Na základě velikosti klíče a odpovídajícího přetečení je aktivován signál *overflow* a je proveden přechod do následujícího stavu *ROT_SUB_Workout*.

Vždy, když zpracováváme klíč s indexem *i*, takovým že $i \bmod Nk = 0$ (*Nk* je počet slov klíče), přecházíme do stavu *ROT_SUB_Workout*. V tomto stavu je rundovní klíč vypočítán následujícím způsobem:

```
temp = SubWord(RotWord(round_key[i-1])) xor Rcon[i/Nk]
round_key[i] = round_key[i - Nk] xor temp
```

To znamená, že na předchozí rundovní klíč (i - 1) aplikujeme operaci rotace o jeden byte doleva (jednotka ShiftRows) a následně posunuté slovo substituujeme pomocí jednotky SubBytes. V dalším kroku pak provedeme operaci XOR s konstantou, která se nachází v poli Rcon. Následuje načtení rundovního klíče z pozice i – Nk a opět operace XOR s předchozím výsledkem. Stav *ROT_SUB_Workout* je rozdělen na několik dílčích stavů, které zajišťují provedení jednotlivých kroků zpracování:

- *rot_sub* je název počátečního stavu. V tomto stavu se nastaví adresa (*back_read_addr*) pro načtení rundovního klíče s indexem i – 1. Dále se nastaví vnitřní konfigurace vnitřních datových cest tak, abychom získali výsledek rotace a následné substituce na portu *DATA_IN*. Také musíme pomocí signálu *SUBST_SELECT* vybrat, zda do jednotky SubBytes bude přiveden výstup jednotky ShiftRows nebo výstupní datový port jednotky KeyExpansion. V tomto případě bude vybrán výstup jednotky ShiftRows.



Obr. 4.20 – Schéma jednotky KeyExpansion

- Stav *rotsub_wait* – v tomto stavu se aktivuje jednotka SubBytes pomocí signálu *SUBST_EN*. Po skončení substituce je výsledek uložen do registru 1 a přechází se do dalšího stavu. Po skončení stavu *rotsub_wait* máme tedy vypočten výsledek rotace a substituce.
- Stav *rcon_add* – v tomto stavu dojde k načtení konstanty Rcon[i/Nk] z vnitřní paměti jednotky. K tomuto účelu je potřeba vystavit na čtecí adresu sběrnici *rcon_addr*. V této části popisu je dobré uvést, že jednotka vnitřní paměti je virtuálně rozdělena na dvě poloviny. V první polovině paměti se nacházejí rundovní klíče a ve druhé polovině pak konstanty Rcon. V tomto stavu také zároveň dojde k operaci XOR načtené konstanty a předchozího výsledku, jež je uložen v registru 1. Výsledek je pak uložen do registru 2.
- Stav *iNk* – v tomto stavu dojde k načtení rundovního klíče s indexem i – Nk a provedení operace XOR s výsledkem předchozího stavu, jež je uložen v registru 2. V tomto stavu nastavujeme na čtecí adresu paměti sběrnici *nk_addr*.
- Stav *rotsub_write* – v tomto stavu je již připraven nový rundovní klíč k zapsání do paměti. Je proto povolen zápis do paměti pomocí signálu *write_en*.
- Stav *rotsub_inc* – v tomto stavu dojde k inkrementaci všech potřebných čítačů a adres, následně se pak přechází do stavu *Normal Workout*.

Většina rundovních klíčů je zpracovávána právě ve stavu Normal Workout. V tomto stavu je rundovní klíč vypočten následujícím způsobem:

```
round_key[i] = round_key[i-1] xor round_key[i-Nk]
```

Nejdříve je načten předchozí rundovní klíč a uchován v jednom z registrů. V dalším kroce je pak načten rundovní klíč s indexem i – Nk a je provedena operace XOR s uloženým klíčem. Stejně jako v přechozím případě je i tento stav tvořen několika dílčími stavy, které si dále popíšeme:

- Stav *normal_workout* je počáteční stav. V tomto stavu provedeme nastavení čtecí adresy na sběrnici *back_read_addr* (i - 1). V dalším hodinovém taktu se nám výsledek uloží do registru 2.
- Stav *nw_read_nk* – v tomto stavu dojde k vystavení adresy rundovního klíče s indexem i – Nk. Ten adresujeme pomocí sběrnice *nk_addr*. V dalším taktu pak získáme výsledek operace XOR mezi právě adresovaným rundovním klíčem a klíčem, který máme uchovaný v registru 2.
- Stav *nw_write* – v tomto stavu známe již výsledek a proto povolíme zápis do paměti pomocí signálu *write_en*. V dalším hodinovém taktu pak dojde k zapsání dat do paměti.
- Stav *nw_increment* – v tomto stavu dojde k inkrementaci potřebných čítačů a adres a přechodu do následujícího stavu.

Následujícím stavem může být stav *ROT_SUB Workout* v případě následujícího klíče s indexem *i*, pro který platí *i mod Nk = 0*. Nebo může být následujícím stavem *Ex8 Workout*, pokud pro index *i* platí, že *i mod Nk = 4* a zároveň *Nk = 8*. Pokud není splněna ani jedna z předchozích podmínek, stavový automat zůstává ve stejném stavu a pokračuje zpracováním následujícího rundovního klíče. Pokud jsme v aktuálním stavu zpracovali poslední klíč, bude následujícím stavem *DONE*. Zbývá nám popsat stavy *Ex8 Workout* a *DONE*.

Stav *Ex8 Workout* je tvořen opět několika dílčími stavy. Rundovní klíč je zpracováván následujícím způsobem:

```
temp = SubWord(round_key[i-1])
round_key[i] = round_key[i - Nk] xor temp
```

Dílčí stavy jsou následující:

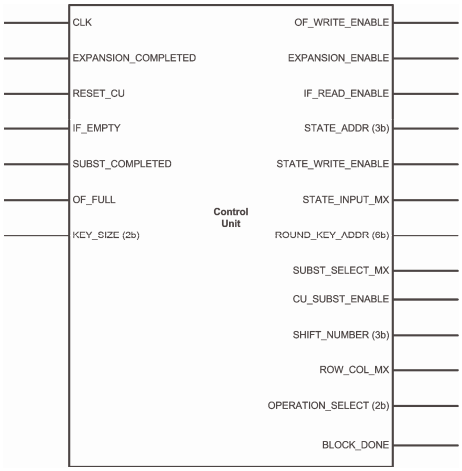
- Stav *ex8* – jedná se počáteční stav, ve kterém provedeme nastavení čtecí adresy na sběrnici *back_read_addr*. V dalším hodinovém taktu pak budeme mít na datovém výstupu jednotky načten rundovní klíč s indexem i – 1. Taky musíme vybrat pomocí signálu *SUBST_SELEC*T vstup jednotky SubBytes. Na rozdíl od stavu *ROT_SUB Workout* nepřivádíme na vstup jednotky SubBytes výstup jednotky ShiftRows, ale přímo datový výstup jednotky KeyExpansion.
- Stav *ex8_wait* – v tomto stavu povolíme činnost jednotky SubBytes a počkáme na dokončení substituce. Po skončení substituce pak bude uložen výstup do registru 2.
- Stav *ex8_ink* – v tomto stavu vystavíme na čtecí adresu sběrnici *ink_addr*. V dalším hodinovém taktu pak získáme na výstupu z paměti rundovní klíč s indexem *i – Nk*. Zároveň budeme vědět výsledek operace XOR.
- Stav *ex8_write* – na datovém vstupu do paměti máme k dispozici spočtený rundovní klíč a připraven k zapsání. V tomto stavu stačí povolit zápis do paměti pomocí signálu *write_en*. S dalším hodinovým taktem pak dojde k zapsání klíče do paměti.
- Stav *ex8_inc* – stejně jako v jiných případech, i v tomto stavu dojde k potřebné úpravě adres a čítačů a následnému přechodu do stavu *Normal Workout*.

Zbývá nám popsat poslední stav *DONE*, do kterého se jednotka dostane po skončení rozvoje klíče. V tomto stavu je aktivován výstupní signál *EXPANSION_DONE*, který informuje řídící jednotku o skončení rozvoje. Řídící jednotka pak reaguje na tento signál deaktivací signálu *KEY_EXPANSION* a jednotka KeyExpansion přejde s následujícím hodinovým taktem do stavu *IDLE*.

V této části jsme si vysvětlili základní princip, jakým jednotka KeyExpansion pracuje. Pro bližší detaily ohledně fungování jednotky (např. podrobné nastavení signálu) je potřeba nahlédnout do zdrojových kódů.

4.9 Jednotka Control Unit

Tato jednotka je jádrem celého obvodu. Slouží k řízení datového toku v obvodu a k nastavování řídících signálů jednotlivých jednotek. Jelikož tato jednotka řídí celou činnost obvodu, byla prakticky její činnost popsána v části 4.1. V této části kapitoly budeme popisovat takové informace, aby byly informace z obou části navzájem disjunktní. Na obrázku 4.21 si můžeme prohlédnout rozhraní této jednotky.



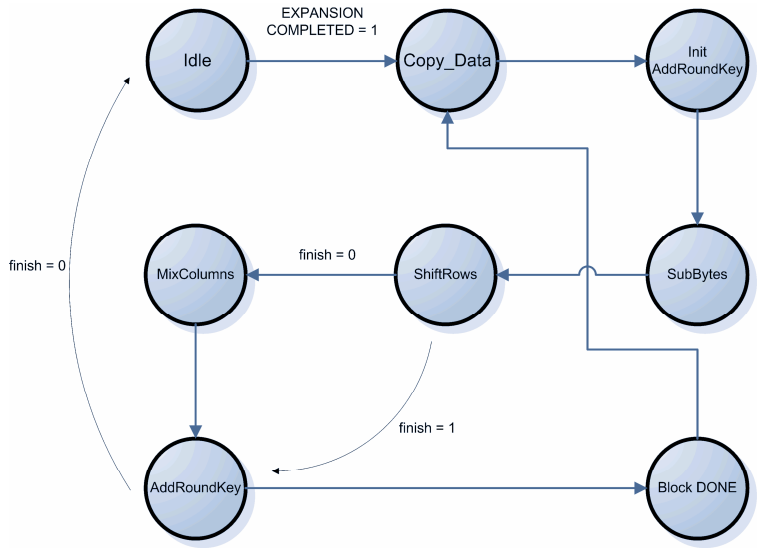
Obr. 4.21 – Rozhraní řídící jednotky

Význam signálů je následující:

- *CLK* je hodinový signál pro synchronizaci jednotky.
- *EXPANSION_COMPLETED* je signál, který informuje řídící jednotku o tom, že expanze klíče byla dokončena. Expanze klíče za aktivuje pomocí signálu EXPANSION_ENABLE.
- *RESET_CU* slouží k uvedení řídící jednotky do počátečního stavu po nahrání nového klíče do obvodu.
- Signál *IF_EMPT*Y informuje řídící jednotku o tom, že vstupní fronta FIFO je prázdná. Naproti tomu signál *OF_FULL* informuje o tom, že výstupní jednotka je plná.
- *KEY_SIZE* slouží k určení velikosti klíče.
- *OF_WRITE_ENABLE* je signál pro aktivaci zápisu do výstupní fronty. Signál *IF_READ_ENABLE* pak pro aktivaci čtení ze vstupní fronty.
- Adresová sběrnice *STATE_ADDR* slouží k adresaci jednotlivých sloupců a řádků stavu.
- *STATE_WRITE_ENABLE* slouží k povolení zápisu dat do stavu.
- *STATE_INPUT_MX* je selekční vstup multiplexoru, který vybírá datový vstup stavu mezi vstupní frontou a výsledkem některé z operací.
- *ROUND_KEY_ADDR* je adresová sběrnice pro adresaci rundovního klíče v jednotce KeyExpansion.
- Signál *SUBST_SELECT_MX* slouží k předání dočasného řízení obvodu jednotce KeyExpansion, která potřebuje ke své činnosti využívat jednotek SuBytes a ShiftRows.

- *CU_SUBST_ENABLE* slouží k aktivaci jednotky SubBytes. O skončení činnosti substituce je řídící jednotka informována skrze signál *SUBST_COMPLETED*.
- Sběrnice *SHIFT_NUMBER* slouží k definování velikosti posunu u jednotky ShiftRows.
- Signál *ROW_COL_MX* představuje selekční vstup multiplexoru, který vybírá vstup jednotky ShiftRows mezi sloupcem a mezi řádkem stavu. V jednotce ShiftRows se totiž posouvají nejen řádky, ale také i sloupce.
- Sběrnice *OPERATION_SELECT* představuje selekční vstup multiplexoru, kterým se vybírá výstup jedné ze čtyř jednotek. Tento výstup je pak určen buď k zápisu do stavu anebo k zápisu do výstupní fronty.
- Signál *BLOCK_DONE* slouží k informaci, že bylo právě dokončeno šifrování jednoho datového bloku a že tento datový blok je uložen ve výstupní frontě. Signál je vystaven právě na dobu jedné periody hodin.

Není potřeba zde popisovat chování této jednotky do detailů, neboť tomu odpovídá popis v části 4.1. Pro ještě bližší detaily ohledně nastavení všech signálů a sběrnic je potřeba se podívat do zdrojových kódů řídící jednotky. Zde si uvedeme jen základní přechodový diagram této jednotky, který je k vidění na obrázku 4.22. Každý stav tohoto diagramu je složen z více dalších podstavů. Jednotka představuje konečný stavový automat s počátečním stavem *IDLE*. V tomto stavu je aktivován signál *KEY_EXPANSION*, který aktivuje rozvoj klíče pomocí jednotky KeyExpansion.



Obr. 4.22 – Přechodový diagram řídící jednotky

Po skončení rozvoje se přechází do stavu *Copy_Data*, v němž se provede okopírování datového bloku ze vstupní fronty do stavu (4 datová slova). Pokud je ve vstupní frontě méně než 4 slova, čeká automat, dokud nebudou k dispozici další data. Po načtení všech 4 slov do stavu se provádí počáteční operace AddRoundKey. Dále pak následuje hlavní cyklus AES algoritmu, který sestává v opakování operace SubBytes, ShiftRows, MixColumns a AddRoundKey. Počet iterací tohoto cyklu je dán

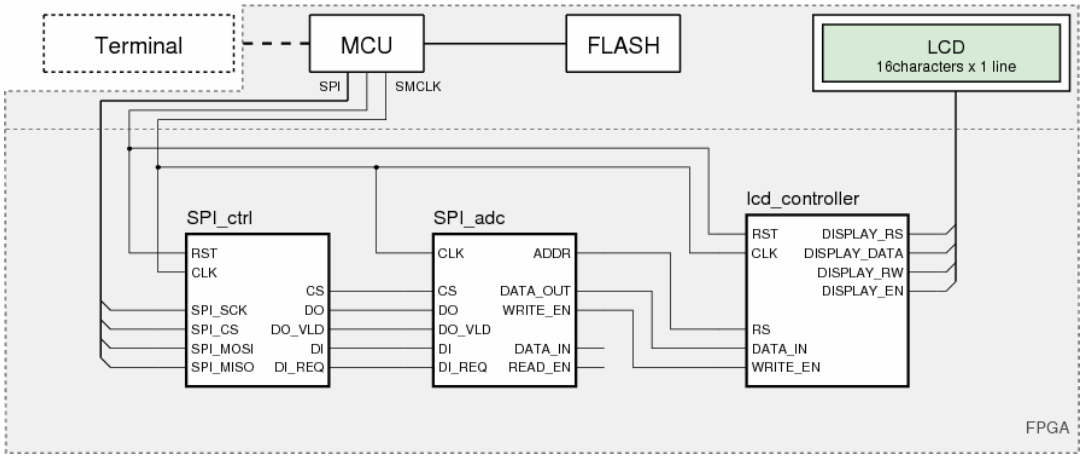
hodnotou Nr – 1. Konstanta Nr byla popsána v části 2.9. Na závěr pak následuje provedení posledního cyklu, ovšem s vynecháním operace MixColumn. Po skončení zpracování bloku dat, přejde automat do stavu *Block DONE*, ve kterém je informována řídící jednotka o skončení šifrování pomocí signálu *block_done*. Po dalším hodinovém taktu dojde k přechodu do stavu *Copy_Data* a dochází ke zpracování dalšího bloku dat. V této části je dobré se zmínit ještě o signálu RESET_CU a jeho funkci: při aktivaci tohoto signálu dochází k přechodu právě do stavu *IDLE*.

Kapitola 5

5 Implementace softwarového řešení

V softwarové implementaci bylo úkolem vytvořit SW realizaci algoritmu AES pro mikrokontrolér, který se nachází na platformě FITKit. Důvodem vytvoření této realizace byla potřeba porovnat vlastnosti HW realizace z předchozí kapitoly vůči softwarovému řešení. Výsledky těchto srovnání jsou pak k dispozici v následující 6. kapitole. Mikrokontrolér na FITKitu obsahuje 48 kB Flash paměti pro uložení programu a 2 kB RAM pro práci s daty. Díky tomu nebylo potřeba generovat různé údaje za chodu a veškeré údaje bylo možné uchovat buď v paměti Flash (konstanty, např. hodnoty S-boxu) nebo v paměti RAM (např. rundovní klíče). SW implementace je tvořena jednak softwarovou částí, a jednak hardwarovou částí. Nejprve si popíšeme HW část.

HW část slouží k tisku údajů na LCD displej FITKitu. Tento LCD displej je připojen na piny hradlového pole FPGA a může zobrazit maximálně 16 znaků. V hradlovém poli byl realizován řadič LCD displeje, který je skrze SPI dekodér připojen k vnitřní sběrnici SPI. K vnitřní sběrnici je připojen zároveň i SPI řadič, který zajišťuje komunikaci LCD řadiče a mikrokontroléru. Vše ilustruje následující obrázek 5.1.



Obr. 5.1 – Blokové schéma HW části

Pokud chce uživatel zobrazit na displeji určitá data, použije k tomu daný příkaz v terminálu a na základě tohoto příkazu se zavolá příslušná obslužná rutina. HW část se nachází v adresáři *top* a je tvořena VHDL souborem s názvem *top_level.vhd*. Výstupem HW části tedy je konfigurační soubor hradlového pole FPGA.

SW část je pak tvořena ovladači LCD displeje a samotnou realizací algoritmu AES. SW realizaci můžeme nalézt v adresáři *sw*. Program je rozmístěn do tří souborů, jejichž funkci si popíšeme v další části. Výstupem SW části je program, který je spuštěn v mikrokontroléru.

5.1 Popis základního programu

Ovládání programu: Veškerá činnost programu je ovládána uživatelem z terminálu (po připojení k FITKitu). Po zadání příkazu a jeho odeslání mikrokontroléru, je volána funkce *USER_CMD_Decode(char *UString, char *String)*, která provede rozpoznání zadaného příkazu. Seznam příkazů můžeme v terminálu vypsat pomocí příkazu „*help*“. Dostupné příkazy jsou:

- *CLOCKS* – vypíše na LCD displej počet hodinových taktů, které odpovídají délce průběhu zpracování programu. Programem se myslí expanze klíče a zašifrování jednoho bloku dat (128 bitů).
- *INUP* – tento příkaz slouží k vypsání první poloviny vstupního datového bloku (8 bytů) na LCD displej v hexadecimálním tvaru (jsou zobrazeny bity 255 až 128). Každý byte je zobrazen pomocí dvou hexadecimálních znaků a proto je tedy na displeji využito všech 16 znaků.
- *INDOWN* – jedná se o příkaz pro vypsání druhé poloviny vstupního datového bloku na LCD displej (jsou zobrazeny bity 127 až 0).
- *OUTUP* – jedná se o komplementární příkaz k příkazu INUP. V tomto případě dojde k vypsání první poloviny zašifrovaného datového bloku na LCD displej.
- *OUTDOWN* – jak název napovídá, příkaz vypíše na LCD displej druhou polovinu zašifrovaného datového bloku.

Vstupy programu: základním vstupem programu, podle kterého se řídí následující činnost programu, je konstanta *KEY_SIZE*. Tato konstanta definuje velikost klíče v počtu bitech. Možnosti u AES algoritmu jsou 128, 192 a 256 bitů. Na základě této konstanty se následně vyberou správné hodnoty konstant Nk a Nr a také se provede zvolení správné hodnoty ukazatele na klíč. Konstanta Nk je počet slov daného klíče (Nk = velikost klíče / 32). Konstanta Nr pak udává počet iterací algoritmu AES. Dále se provádí zvolení ukazatele na jeden ze 3 vstupních klíčů. První klíč je uložen v poli bytů pod názvem *klic_128* a jak název napovídá, má velikost 128 bytů. Obdobně zbývající dva klíče jsou uloženy v polích s názvy *klic_192* a *klic_256*. Dalším a posledním vstupem programu je samotný datový blok (128 bitů), který je určen k zašifrování. Jedná se o proměnnou s názvem *input*. Jak je patrno, zpracovává se v tomto programu pouze jeden datový blok. Program samozřejmě lze snadným způsobem upravit tak, aby šifroval více bloků. Pro naše potřeby ovšem stačí zašifrovat jen jeden blok.

Průběh programu: V prvních krocích programu je provedena inicializace mikrokontroléru, LCD displeje a čítače. Čítač je nastaven na kontinuální mód, což znamená že při přetečení začne čítat znovu od počáteční hodnoty (nuly). Aby ovšem nedošlo k přetečení čítače, je jeho pracovní frekvence nastavena na osminu pracovní frekvence mikrokontroléru. Při výpisu výsledků je pak výstupní hodnota násobena číslem 8. Inicializaci čítače provádí funkce *timerB_init()* a reset čítače pak funkce *timerB_clear()*. Dalším krokem je, jak již bylo popsáno, nastavení konstant Nr, Nk a výběr ukazatele

na jeden z klíčů. Dále následuje volání funkce *key_expansion(key, round_keys, Nk, Nr),* která provede rozvoj klíče (parametr *key*) a výsledné rundovní klíče uloží do pole bytů s názvem *round_keys*. Po tomto rozvoji je pak volána funkce void *aes_cipher(input, output, round_keys, Nr)*, která provede zašifrování vstupního bloku *input* a výstup uloží do pole s názvem *output*. Před započetím operace rozvoje klíče je resetován čítač. Od této chvíle je započato měření doby průběhu operací *key_expansion* a *aes_cipher*. Na závěr programu je pak uložena hodnota čítače do proměnné *counter*. Zbývající zakomentovaná část programu sloužila k otestování dob jednotlivých operací. Tyto doby jsou použity v další kapitole pro srovnání vlastností HW implementace vůči SW implementaci.

Výstup programu: po skončení programu máme uložen vstupní a výstupní datový blok v proměnných *input* a *output*. Hodnota (v hodinových cyklech), udávající dobu trvání programu, je uložena v proměnné counter. Po skončení šifrování dat je program zacyklen. V zacyklené smyčce je pak volána obsluha terminálu a uživatel může provádět výpisy výstupních informací pomocí již dříve zmíněných příkazů.

5.2 Popis jednotlivých operací

Všechny operace AES šifry jsou uloženy v souboru *aes_cipher.c*. V této části si uvedeme krátký popis ke každé operaci:

- *key_expansion (unsigned char * key, unsigned char * keys, unsigned char Nk, unsigned char Nr)* – jedná se o operaci, která provede expanzi klíče. Na začátku operace je okopírováno Nk slov do výstupního pole rundovních klíčů (pole *keys*). Dále pak následuje klasické zpracování dle algoritmu, který již byl popsán.
- *ShiftRows (unsigned char * slovo, unsigned char posuv)* – podle názvu funkce je možné poznat, že se jedná o rotaci řádků. Daný řádek předáváme do funkce parametrem *slovo*. Proměnná posuv pak udává o kolik bytů se bude rotovat. Možnosti jsou o jeden, dva či tři byty.
- *SubBytes (unsigned char * slovo)* – funkci předáváme ukazatel na pole 4 bytů, nad kterými se provede substituce pomocí s-boxu (proměnná *sbox*).
- *AddRoundKey (unsigned char * slovo, unsigned char * round_keys, unsigned char key_index)* – funkci je předán ukazatel na pole 4 bytů, které budou XORovány s odpovídajícím rundovním klíčem. Parametr *round_keys* je ukazatel na začátek pole rundovních klíčů a parametr *key_index* udává index odpovídajícího rundovního klíče.
- *MixColumns (unsigned char * slovo)* – provede operaci násobení sloupce (parametr *slovo*) s maticí.
- *aes_cipher (unsigned char * input, unsigned char * output, unsigned char * round_keys, unsigned char Nr)* – jedná se o samotný algoritmus AES. V první části algoritmu je

okopírován vstup (parametr *input*) do stavu (proměnná *state*) a provedena počáteční operace AddRoundKey. Dále pak následuje hlavní cyklus, ve kterém jsou postupně prováděny základní operace. Tyto operace pracují se stavem po řádcích či sloupcích (tedy 32 bitech) z důvodu zachování podobnosti s HW řešením. Díky tomuto pak můžeme snadněji provést porovnání SW a HW implementace. Počet iterací algoritmu je dán parametrem Nr. Na konci algoritmu je provedena ještě jedna iterace algoritmu AES, ovšem s vynecháním operace MixColumn. Na závěr je stav okopírován na výstup (proměnná *output*).

Kapitola 6

6 Porovnání HW a SW implementace

V této kapitole provedeme finální srovnání HW a SW implementace. Zaměříme se především na rozbor urychlení hardwarové implementace vůči softwarové implementaci. Dále pak rozebereme množství využitých zdrojů u jednotlivých implementací a také dosažené rychlosti.

6.1 Naměřené výsledky u HW implementace

Na úvod této sekce bych rád uvedl, že obvod byl navržen s ohledem na velikost dostupných zdrojů, které nabízí hradlové pole FPGA na platformě FITKit. Typ hradlového pole na FITKitu patří k jedněm z nejmenších průmyslově vyráběných FPGA a jelikož tedy nebylo mnoho prostoru pro paralelizaci výpočtu, jsou operace AES algoritmu prováděny sekvenčním způsobem. Na druhou stranu je ovšem většina operací prováděna pomocí kombinační logiky a zpracování výpočtu jedné operace tak trvá jeden hodinový takt. V HW implementaci jsou data zpracovávána po slovech (32 bitů). V následujícím textu si popíšeme počet hodinových taktů, které připadají na jednotlivé operace AES algoritmu a zpracování jednoho datového slova:

- Jednotka ***KeyExpansion*** – od vystavení aktivačního signálu po vystavení signálu, kterým jednotka KeyExpansion informuje řídící jednotku o konci rozvoje, uběhne následující počet taktů:
  - pro klíč o velikosti 128 bitů = **236** hodinových taktů
  - pro klíč o velikosti 192 bitů = **248** hodinových taktů
  - pro klíč o velikosti 256 bitů = **302** hodinových taktů
- Operace ***SubBytes*** – tato operace je provedena během 12 hodinovýh taktů, včetně zápisu do stavu. Těchto 12 hodinových taktů připadá na zpracování jednoho slova (32 bitů) a na zpracování celého stavu tedy připadá 4 x 12 = **48** taktů. Je potřeba si také povšimnout jednotky SubBytes, která se při této operaci využívá. 5 taktů, z uvedených 12, trvá právě samotná substituce pomocí jednotky SubBytes.
- Operace ***ShiftRows*** – operace je provedena během 6 hodinových taktů, včetně zápisu slova do stavu. Na zpracování celého stavu pak připadne 4 x 6 = **24** hodinových taktů.
- Operace ***MixColumns*** a ***AddRoundKey*** trvají obě 8 hodinových taktů, včetně i zápisu do stavu. Na zpracování celého stavu pomocí jedné z operací pak připadne 4 x 8 = **32** hodinových taktů.

- Zašifrování jednoho bloku dat – od načtení bloku dat ze vstupní fronty po zápis zašifrovaného bloku do výstupní fronty a vystavení signálu *BLOCK_DONE,* uběhne následující počet hodinových taktů:
    - při šifrování 128–bitovým klíčem: **546** hodinových taktů
    - při šifrování 192–bitovým klíčem: **654** hodinových taktů
    - při šifrování 256–bitovým klíčem: **762** hodinových taktů

Předchozí hodnoty byly naměřeny během simulace obvodu. Následující hodnoty, které jsou vyneseny do tabulky 6.1, pak byly naměřeny při syntéze obvodu. Syntéza obvodu byla provedena pomocí programu Precision od firmy Mentor Graphics.

Využití zdrojů obvodu 3S50pq208			
Typ zdroje	Počet použitých prvků	Počet dostupných prvků	Celkové využití zdroje
IOs	330	124	266.13%
Global Buffers	1	8	12.50%
Function Generators	1153	1536	75.07
CLB Slices	577	768	75.13%
Dffs or Latches	577	1908	30.24%
Block RAMs	0	4	100.00%
Block Multipliers	0	0	0.00%
Block Multiplier Dffs	0	144	0.00%

Tabulka 6.1 – Výsledky výstupu syntézy

Tyto hodnoty bylo převzaty přímo z výstupu aplikace. Hodnoty se týkají typu hradlového pole, které je součástí FITKitu (XC3S50-4PQ208C od firmy Xilinx). První sloupec popisuje dostupné zdrojové komponenty hradlového pole FPGA. Druhý sloupec popisuje počet použitých prvků dané komponenty cílové technologie a třetí sloupec popisuje maximální dostupný počet prvků dané komponenty. Poslední sloupec pak popisuje procentuální využití zdrojů.

První řádek tabulky s názvem IOs udává počet vstupních a výstupních pinů obvodu. Je zde vidět, že náš obvod využívá přibližně 2,6-krát více pinů, než nám může daný typ FPGA nabídnout. To ovšem vůbec nic neznamená, neboť všechny piny hradlového pole FPGA na platformě FITKit mají pevně přiřazenou svoji funkčnost. Pokud bychom chtěli realizovat náš obvod na FITKitu, je zapotřebí využít upraveného SPI dekodéru, ke kterému připojíme náš obvod. SPI dekodér je pak pomocí vnitřní

SPI sběrnice připojen k SPI řadiči, který by zajišťoval komunikaci mikrokontroléru a našeho obvodu. Mikrokontrolér by v tomto případě hrál roli vnější jednotky, která nahrává do AES obvodu data k šifrování a zároveň je pak z obvodu čte.

Hradlové pole FPGA je složeno z tzv. CLB (angl. Configurable Logic Block). Každý tento blok je rozdělen na menší logické elementy, tzv. slices. Každý jeden slice pak obsahuje 2 funkční generátory, 2 registry, 2 multiplexory, carry logiku a pomocnou logiku pro aritmetiku. Na třetím řádku můžeme vidět počet využitých funkčních generátorů. Náš obvod používá 75% těchto prvků. Funkční generátor může představovat buď 4-vstupové logické hradlo (tzv. LUT – Look-Up Table), nebo paměť o velikost 16 bitů - tzv. RAM16, popř. 16-bitový posuvný registr tzv. SRL16.

Čtvrtý řádek udává počet jednotek slice (75% využití). Pátý řádek pak udává celkový počet registrů v obvodu (30% využití). Na šestém řádku se nachází využití pamětí typu BlockRAM. Vidíme, že v našem obvodu jsou využity všechny 4 paměti. Na posledních dvou řádcích se nachází využití vestavěných násobiček a souvisejících registrů. Ty v našem obvodu vůbec nepoužíváme. Detailní informace o tom, kolik prvků využívají konkrétní části obvodu, můžeme nalézt v adresáři *synthesis_results* v souboru *aes_cipher_area.rep*.

Dalším důležitým údajem syntézy je maximální frekvence, na které může náš obvod pracovat. Hodnota této frekvence je **79.472** MHz, což nám dává dobu periody hodinového signálu **12.583** ns.

6.2 Naměřené výsledky u SW implementace

Softwarovou implementaci jsem se snažil připodobnit HW implementaci takovým způsobem, aby bylo možné dané realizace porovnávat. Jedná se např. o měření zpracování jednoho datového slova. V SW implementaci se zpracovává stav také po sloupcích či řádcích. Ve skutečnosti jsou ovšem data zpracovávána mikrokontrolérem po bytech a to z důvodu 16-bitové RISCové architektury mikroprocesoru. V dalším textu si popíšeme doby jednotlivých operací v hodinových taktech:

- Rozvoj klíče pomocí procedury ***key_expansion*** :
  - klíč o velikosti 128 bitů = **26264** hodinových taktů
  - klíč o velikosti 192 bitů = **29176** hodinových taktů
  - klíč o velikosti 256 bitů = **39256** hodinových taktů
- Operace ***SubBytes*** je realizována pomocí stejnojmenné operace. Operace je provedena během **176** hodinových taktů. Zpracování celého stavu pak trvá 176 x 4 = **704** hodinových taktů.
- Operace ***ShiftRows*** trvá při různém posuvu různý počet hodinových taktů.
  - při posuvu o jeden byte = **192** hodinových taktů
  - při posuvu o dva byty = **120** hodinových taktů
  - při posuvu o tři byty = **200** hodinových taktů

- Při sečtení předchozích hodnot pak získáme dobu zpracování celého stavu 192 + 120 + 200 = **512** taktů

Odlišnosti v počtech taktů pro různé posuvy jsou dány tím, že se slovo zpracovává pokaždé jiným způsobem. To je dáno především faktem, že mikroprocesor je 16-bitový a k datům přistupuje po bytech.

- Operace ***MixColumns*** trvá **480** hodinových taktů. Zpracování celého stavu pak 4 x 480 = **1920** taktů.
- Operace ***AddRoundKey*** trvá **248** hodinových taktů. Zpracování celého stavu pak 4 x 248 = **992** taktů.
- Zašifrování jednoho datového bloku dat – provádí se pomocí operace ***aes_cipher*** a trvá:
  - při šifrování 128-bitovým klíčem: **45248** hodinových taktů
  - při šifrování 192–bitovým klíčem: **54244** hodinových taktů
  - při šifrování 256–bitovým klíčem: **63424** hodinových taktů

Tyto hodnoty byly získány výpisem testovacího programu, který je součástí SW implementace. Dalšími důležitými údaji u SW implementace je velikost, kterou zabírá program ve Flash paměti a kolik RAM zabírají dočasná data. Velikost programu je **24412** bytů, což odpovídá přibližně polovině paměti Flash. Zabranou paměť RAM pak můžeme spočítat sečtením bytů jednotlivých proměnných. To činí přibližně **450** bytů, což odpovídá přibližně 22% zaplněné paměti RAM (2 kB).

Další důležitou a zároveň poslední informací je frekvence, na které mikrokontrolér pracuje. Hodnota této frekvence je **7.3728** MHz, což nám dává dobu periody **135,634** ns.

6.3 Srovnání SW a HW implementace

Při srovnání HW a SW implementace musíme vzít v úvahu fakt, že HW implementace pracuje se stavem po slovech a SW implementace po bytech. Sice byla vytvořena u SW implementace abstrakce, která dává dojem, že je se stavem pracováno po slovech, ovšem ve skutečnosti pracuje mikrokontrolér s daty po bytech. Dalo by se říci, že pokud pracujeme s daty po bytech, tak je zpracování 4x pomalejší, než při zpracování po slovech. Toto ovšem není tak úplně pravda, neboť musíme vzít v úvahu také určitou režii, která se týká např. načítání dat do ALU jednotky, ukládání dat do paměti a podobně. Při popisu každé operace uvedeme zrychlení, které je vypočteno na základě počtu hodinových taktů a další podrobnosti, na které je potřeba vzít ohled. Na závěr si pak uvedeme celkovou datovou propustnost jednotlivých implementací a reálné zrychlení HW realizace.

- Operace ***KeyExpansion*** – pokud podělíme počet taktů SW a HW implementace získáme následující zrychlení:
  - pro 128-bitový klíč: 26264 / 236 = **111**
  - pro192-bitový klíč: 29176 / 248 = **117**

- pro 256-bitový klíč: 39256 / 302 = **129**

Jak je vidět, se vzrůstající velikostí klíče, se zrychlení ještě více zvětšuje. Zrychlení je tedy řádově 100-násobné.

- Operace ***SubBytes*** – tato operace je prováděna po bytech v obou implementacích a zrychlení HW implementace při zpracování celého stavu je: 704 / 48 = **14**, což není tak výrazné jako v případě expanze klíče, ale samozřejmě není ani nijak zanedbatelné. Další možné teoretické zrychlení HW implementace by bylo možné pomocí paralelizace. To by spočívalo v použití více s-boxů. S-box se ve většině případů realizuje pomocí paměti. V hradlovém poli FPGA jsou ovšem k dispozici bohužel jen 4 paměti ve formě komponent BlockRAM. Tyto paměti byly všechny využity.
- Operace ***ShiftRows*** – HW implementace realizuje operaci posuvu jednoho slova v jednom taktu. SW implementace pak posouvá slovo přeskládáním jednotlivých bytů. Zrychlení je 512 / 24 = **21**. Pokud bychom teoreticky uvážili, že by 32-bitový mikrokontrolér zpracoval stav 4-krát rychleji, pak by teoretické zrychlení HW implementace bylo 512 / 4 / 24 = 5.
- Operace ***MixColumn*** – zrychlení HW implementace je 1920 / 32 = **60**. SW implementace pracuje s daty po bytech i v případě 32-bitového mikrokontroléru a není potřeba zde uvažovat o jiném zrychlení jako v případě předchozí operace.
- Operace ***AddRoundKey*** – zrychlení HW implementace je 992 / 32 = **31**. Operace AddRoundKey představuje v základu pouze operaci XOR a na 32-bitovém mikrokontroléru by mohla být provedena 4x rychleji. Teoretické zrychlení by pak bylo 992 / 4 / 32 = 7.
- Zašifrování jednoho datového bloku :
  - zrychlení při šifrování klíčem od délce 128 bitů: 45248 / 546 = **82**
  - zrychlení při šifrování klíčem od délce 192 bitů: 54244 / 654 = **82**
  - zrychlení při šifrování klíčem od délce 256 bitů: 63424 / 762 = **83**

  Jak lze vidět, vyšlo zrychlení téměř shodně pro všechny velikosti klíče.

Toto porovnání bylo založeno na počtu hodinových taktů. Dále si uvedeme výkonnosti obou řešení při daných frekvencích. Výkon u AES šifry bude znamenat počet datových bloků (128 bitů), které dané řešení zašifruje za jednu sekundu. Výkon vypočítáme podle vzorce:
výkon = frekvence řešení / počet taktů [bloků/s], kde počet taktů odpovídá zpracování jednoho datového bloku.

U HW implementace je výkon:

- pro klíč o velikosti 128 bitů - 79 472 000 / 546 = **145 553 *bloků/s* = 2274 *kB/s***
- Pro klíč o velikosti 192 bitů - 79 472 000 / 654 = **121 516 *bloků/s* = 1898 *kB/s***
- Pro klíč o velikosti 256 bitů - 79 472 000 / 762 = **104 293 *bloků/s* = 1629 *kB/s***

U SW implementace je pak výkon:

- Pro klíč o velikosti 128 bitů - 7 372 800 / 45 248 = **162 *bloků/s* = 2,54 *kB/s***
- Pro klíč o velikosti 192 bitů - 7 372 800 / 54 244 = **135 *bloků/s* = 2,12 *kB/s***
- Pro klíč o velikosti 256 bitů - 7 372 800 / 63 424 = **116 *bloků/s* = 1,82 *kB/s***

Nakonec je potřeba ještě zjistit skutečné zrychlení HW realizace oproti SW řešení. Skutečné zrychlení tedy je:

- Pro klíč o velikosti 128 bitů – 2274 / 2,54 = **895**
- Pro klíč o velikosti 192 bitů – 1898 / 2,12 = **895**
- Pro klíč o velikosti 256 bitů – 1629 / 1,82 = **895**

Závěrem této kapitoly tedy můžeme říci, že jsme díky HW realizaci dokázali zrychlit šifrování dat 895-krát. To je řádově téměř tisícinásobné zrychlení oproti SW realizaci. Taky ovšem nesmíme zapomenout na fakt, že vnější jednotkou našeho obvodu je mikrokontrolér, který jí dodává vstupní data k šifrování. Z toho důvodu je datový tok našeho obvodu omezen rychlostí, s jakou mikrokontrolér dokáže data do AES obvodu zapisovat a taky z něj číst.

Kapitola 7

7 Závěr

Hlavním cílem této diplomové práce bylo navrhnout obvod, který by realizoval algoritmus AES. Důvodem pro vytvoření této hardwarové realizace bylo urychlení šifrování dat vůči klasickému softwarovému zpracování. Práce se skládala z několika dílčích kroků: prvním krokem bylo nastudování potřebné literatury. Literatura se týkala zejména algoritmu AES [6, 7, 8] a jazyka VHDL [9, 10, 11]. Dále bylo potřeba nastudovat informace týkající se platformy FITKit [3, 4] a základů kryptografie [1, 2]. Na základě těchto znalostí byl vytvořen teoretický úvod, který je popsán ve druhé a třetí kapitole.

Na základě těchto nabitých informací byla prvně navržena softwarová implementace. To z toho důvodu, abychom následně mohli porovnat navržený obvod vůči SW zpracování. SW implementace byla realizována pomocí mikrokontroléru, který je součástí platformy FITKit. Se vstupními daty je v této implementaci pracováno po bytech, neboť mikrokontrolér má 16-bitovou RISC architekturu a implementace AES algoritmu mohou pracovat s daty buď po slovech (32 bitů) anebo po bytech. Vytvořený program provádí expanzi klíče a zašifrování jednoho vstupního bloku dat. Po skončení programu má uživatel možnost pomocí příkazů, které zadává v terminálu, zobrazit na displeji FITKitu vstupní a výstupní datový blok a také počet hodinových taktů, které odpovídají délce zpracování programu. Součástí programu je i analýza, ve které se provádělo měření počtu taktů, které odpovídají jednotlivým procedurám AES algoritmu. Toto měření už ovšem není součástí uživatelského programu. SW implementace je popsána v kapitole č. 5.

Dalším krokem diplomové práce byl návrh hardwarového obvodu. Obvod byl navržen s ohledem na dostupné zdroje hradlového pole FPGA, které je součástí platformy FITKit. Po návrhu tohoto obvodu bylo potřeba obvod verifikovat pomocí simulace. Po úspěšném odladění byl obvod syntetizován pro technologii FPGA. Výsledky syntézy jsou popsány v kapitole č. 6. Pracovní frekvence obvodu je přibližně 80 MHz a celkový výkon se pak pohybuje okolo 2 MB/s.

Po vytvoření a ověření funkčnosti obou implementací byla vytvořena analýza, která rozebírala vlastnosti a především urychlení hardwarové realizace vůči softwarové implementaci. Byly měřeny počty taktů všech procedur AES algoritmu u obou realizací a zrychlení těchto procedur oproti SW implementaci. Detailní analýzu výsledků můžeme nalézt v kapitole 6. U HW implementace bylo dosaženo poměrné vysokého zrychlení: 895-krát rychlejší šifrování než u SW implementace. V úvahu je také potřeba vzít frekvence, na kterých HW a SW řešení pracují. SW řešení pracuje totiž jen na frekvenci 7.37 MHz, což je téměř 11-krát pomaleji než u HW obvodu.

Kapitola 8

8 Literatura

[1] Wikipedia - the free encyclopedia, květen 2007
<http://en.wikipedia.org/wiki/Main_Page>

[2] Doc. Dr. Ing. HANÁČEK, P.: Přednášky předmětu Kryptografie, květen 2007
<https://www.fit.vutbr.cz/study/courses/KRY/private/kry.htm>

[3] Platforma FITKit, květen 2007
<http://merlin.fit.vutbr.cz/FITkit/>

[4] Texas Instrument, Mikrokontrolér MSP430F168, květen 2007
<http://focus.ti.com/docs/prod/folders/print/msp430f168.html>

[5] NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation, December 2001
<http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>

[6] Federal Information Processing Standards Publication (FIPS PUB) 197: Advanced Encryption Standard (AES)
<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>

[7] Daemen, J., Rijmen, V.: AES Proposal - Rijndael
<http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf>

[8] Rijndael Home Page
<http://csrc.nist.gov/CryptoToolkit/aes/rijndael/ >

[9] Glauert, W.: VHDL Tutorial
<http://www.vhdl-online.de/tutorial>

[10] Ashenden, P.: VHDL CookBook
<http://tech-www.informatik.uni-hamburg.de/vhdl/doc/cookbook/VHDL-Cookbook.pdf>

[11] Zhang, W.: VHDL Tutorial – Learn by Example
<http://csold.cs.ucr.edu/content/esd/labs/tutorial>

[12] Geißler, Bulach: Manual for Synthesis and Optimization, květen 2007
<http://mikro.e-technik.uni-ulm.de/vhdl/anl-engl.syn/html/synt-all-e.html>

[13] Glauert, W.: FSM Tutorial, květen 2007
< http://www.vhdl-online.de/fsm/>

[14] Palermo, G.: FSM in VHDL, květen 2007
< http://www.vhdl-online.de/fsm/>