VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ŠIFROVANÁ KOMUNIKACE MEZI DVĚMA FITKITY

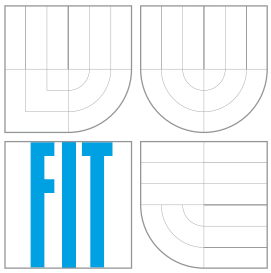
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

STANISLAV BOŘUTÍK

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ŠIFROVANÁ KOMUNIKACE MEZI DVĚMA FITKITY

ENCRYPTED COMMUNICATION BETWEEN TWO FITKITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — STANISLAV BOŘUTÍK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN ŽÁDNÍK
SUPERVISOR

BRNO 2010

Abstrakt

Práce se zabývá problematikou šifrované komunikace mezi párem FITkitů. Výsledkem je implementace systému pro jednosměrný přenos krátkých textových zpráv pomocí rozhranní RS-232. Zprávy jsou šifrovány algoritmem AES-128 v módu CBC a jejich psaní a zobrazení zajišťují klávesnice a LCD displej FITkitu.

Abstract

This thesis is about encrypted communication between two FITkits. Thesis goal is implementation of system for short text message one-way transmission through RS-232 interface. Messages are encrypted by AES-128 algorithm working in CBC mode and their input and display are provided by keyboard and LCD display of FITkit device.

Klíčová slova

FITkit, AES, Rijndael, komunikace, RS-232, kryptografie

Keywords

FITkit, AES, Rijndael, communication, RS-232, cryptography

Citace

Stanislav Bořutík: Šifrovaná komunikace mezi dvěma FITkity, bakalářská práce, Brno, FIT VUT v Brně, 2010

Šifrovaná komunikace mezi dvěma FITkity

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Martina Žádníka a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Stanislav Bořutík
14. května 2010

Poděkování

Děkuji vedoucímu práce Ing. Martinovi Žádníkovi za cenné rady a vstřícný přístup a Ing. Zděňkovi Vašíčkovi za pomoc s řešením implementačních problémů.

Obsah

Kapitola 1

Úvod

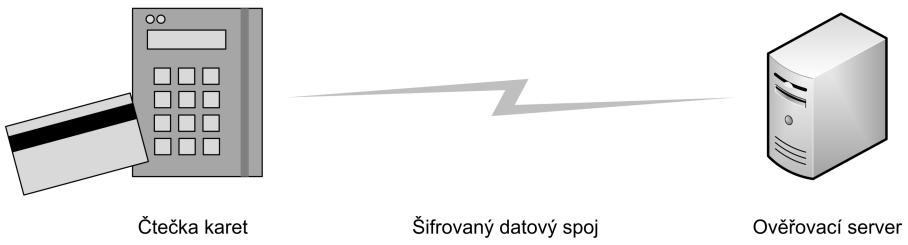
Šifrování již dávno není jen výsadou armády, úřadů, bank a korporací. Proniklo do každodenních činností obyčejných lidí, aniž si to mnohdy uvědomujeme. Používáme mobilní telefon, platební a identifikační karty, posíláme e-maily, ovládáme svůj bankovní účet přes internet, připojujeme se k firemní síti z domova. Všechny tyto a mnohé další činnosti se neobejdou bez použití šifrovacích systémů, které ochrání naše důvěrná data před zneužitím. Informace dnes mají velkou hodnotu a proto je nutné se postarat o jejich bezpečí. Například armáda každého státu operuje s takovými informacemi, jejichž prozrazení by mohlo ohrozit obrovské množství lidských životů.

1.1 Co je to šifrování

Účelem šifrování je skrytí obsahu utajované zprávy před nepovolanými osobami. I když bude mít útočník přístup ke zprávě v zašifrované podobě, nebude moci jejímu obsahu porozumět.[9]

1.2 HW Šifrování

Šifrování v hardwaru nalézá uplatnění u různých vestavných systémů. Příkladem takového systému může být čtečka karet u vchodových dveří do budovy, sloužící pro zabránění vstupu nepovolaných osob. Identifikační údaje z karty jsou odeslány autentizačnímu serveru v šifrované podobě a zde zpracovány. Tím se velmi snížila pravděpodobnost toho, že útočník získá odposlechem na komunikačním médiu tajné identifikační údaje z karty a zneužije je.



Obrázek 1.1: Příklad použití šifrování - komunikace mezi čtečkou čipových karet a autentizačním serverem

Cílem této práce je ukázat, jaké šifrování lze na FITkitu, který má z pohledu moderních kryptografických metod velmi omezené prostředky, implementovat. FITkit obsahuje vhodné periferie (klávesnici, LCD displej) umožňující jednoduchou prezentaci funkce šifrování. Díky použití jazyka VHDL může implementovaný systém sloužit pro budoucí využití při vývoji jednoúčelového šifrovacího čipu (ASIC). Tento čip může nalézt uplatnění například v síťových prvcích pro bezpečný přenos dat (ať už po metalických spojích či bezdrátově) nebo v řadičích paměťových zařízení pro uchovaní citlivých informací. Jednoduše kdekoliv, kde je potřeba rychle a transparentně šifrovat data. Vyvinutý systém může také sloužit jako základ pro budoucí aplikace a projekty, které budou na FITkitu realizovány.

Ve druhé kapitole této práce jsou rozebrány šifry, přicházející v úvahu pro implementaci na FITkitu a následně výběr jedné z nich. Obsahem třetí kapitoly je popis vybraného šifrovacího algoritmu a popis jeho implementace na platformě FITkit. Čtvrtá a pátá kapitola obsahují zhodnocení implementace a závěr této práce.

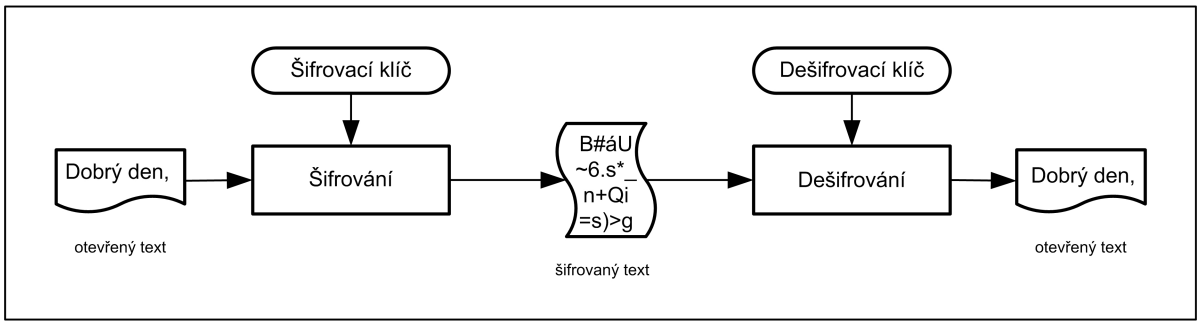
Kapitola 2

Šifrovací algoritmy

2.1 Proces šifrování a dešifrování

Vysvětleme si nejdříve elementární pojmy, které budou dále v textu použity:

Šifrovací algoritmus - pravidla použitá k zašifrování zprávy
Otevřený text (plain text) - text, který chceme zašifrovat
Šifrovaný text (cipher text) - zašifrovaný text v nečitelné podobě
Šifrování - proces zabezpečení zprávy
Dešifrování - proces získávání otevřeného textu z textu šifrovaného



Obrázek 2.1: Proces šifrování a dešifrování

2.2 Rozdělení algoritmů

Šifrovací algoritmy se dělí podle způsobu práce s klíči na

- symetrické
- asymetrické

2.2.1 Symetrické šifry

Dešifrovací klíč lze u těchto šifer odvodit z šifrovacího klíče. Obvykle bývají totožné, proto se také nazývají jako jednoklíčové systémy nebo systémy s tajným klíčem. Odesílatel a příjemce sdílí stejný tajný klíč. Výhodou je jejich rychlost, jsou vhodné pro objemná data.

Nevýhodyou je nutnost přenosu klíče zabezpečeným kanálem, protože bezpečnost algoritmu závisí právě na klíči. Podle způsobu zašifrování dat se dále dělí na blokové či proudové. Např. DES, IDEA, MARS, Serpent, Blowfish, Twofish.

Blokové symetrické šifry

Šifrují zprávu po skupinách bitů pevné délky, zvaných bloky. Každý blok dat je šifrován stejnou funkcí a šifrováním nemění velikost. Velikost bloků je nejčastěji 64b, u AES 128b. Pokud délka dat není přesným násobkem délky bloku, je potřeba je doplnit. Toto doplnění se nazývá "padding" a lze jej provést jednoduše vyplněním nulami nebo některým sofistikovanějším algoritmem.

Proudové symetrické šifry

Šifrují zprávu bit po bitu (dříve po slovech či písmenech). Hodnota bitu může při šifrování nabýt opačné hodnoty nebo zůstane nezměněna. To, zda se změní či nikoliv, určuje šifrovací klíč (bit klíče je 0 nebo 1). Pro šifrování se používá operace XOR. Pokud je na šifrovanou zprávu aplikován znovu klíč, jímž byla zašifrována, získáváme zpět otevřený text - symetrické šifry.

Jejich výhodou je, že pokud při přenosu zprávy dojde k chybě, která ovlivní jeden bit (např. rušením na přenosovém médiu), je špatně dekódován pouze tento bit a ostatní zůstanou v pořádku. Proto se tyto šifry používají v telekomunikacích (například standard pro mobilní telefony GSM používá šifru A5), RC4 se používá pro šifrování komunikace ve Wi-Fi sítích. Dalšími výhodami oproti blokovým šifrám jsou vyšší rychlost a nenáročná implementace.

2.2.2 Asymetrické šifry

Říká se jim také systémy s veřejným šifrovacím klíčem. Tyto systémy používají veřejný šifrovací klíč a tajný dešifrovací klíč, přičemž dešifrovací klíč prakticky nelze z klíče šifrovacího odvodit. Například algoritmus RSA je postaven na problému rozkladu dostatečně velkého čísla na jeho prvočinitele (faktorizace). Velikost bloků u této šifry začíná na 640 bitech, výjimkou nejsou bloky o velikosti 1024 nebo 2048 bitů. Algoritmus El Gamal, který tvoří základ amerického standardu pro digitální podpisy (DSS), je postaven na jiném matematickém problému, známém jako diskrétní logaritmus.

Výhodou je odstranění problému s přenosem klíčů. Velkou nevýhodou je výpočetní náročnost, která je činí nevhodnými pro objemná data.

Výpočetní náročnost asymetrických algoritmů je také důvodem, proč je nebudu dále uvažovat jako kandidáty pro implementaci na FITkitu.

Podrobněji je rozdělení popsáno v [9]

2.3 Módy ECB a CBC symetrických blokových šifer

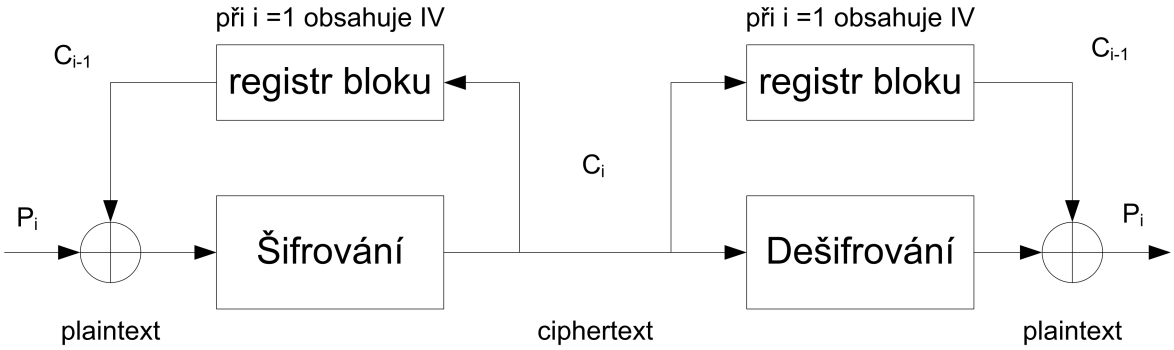
Ačkoliv existuje více módů, v jakých mohou symetrické blokové šifry pracovat, zmíním zde pouze dvě, které jsou relevantní. Další módy jsou popsány např. v [7]

2.3.1 ECB (Electronic Code Book)

Toto je přirozený mód šifry, která pracuje přesně dle své specifikace. Každý blok plaintextu je pomocí šifrovacího klíče zašifrován na odpovídající blok šifrovaného textu nezávisle na ostatních blocích. Šifrovaná podoba dvou zpráv, které byly zašifrovány pomocí stejného klíče, bude shodná. To činí šifru deterministickou a tedy náchylnou k útokům, proto se tento mód používá velmi zřídka.

2.3.2 CBC (Cipher Block Chaining)

V tomto nejčastěji používaném módu je zašifrovaný blok závislý na všech předchozích blocích zprávy. Závislost je zajištěna prostřednictvím operace XOR, jejímiž operandy jsou před šifrováním aktuální vstupní blok a zašifrovaný blok předchozí, při dešifrování dešifrovaný aktuální blok a zašifrovaný předchozí blok. Jelikož první blok zprávy nemá předchůdce, je ke zprávě připojen nultý blok, zvaný inicializační vektor (IV). Ten nemusí být zašifrovaný, ovšem měl by být nedeterministický. Dvě shodné zprávy budou zašifrovány shodně pouze při použití nejen shodných klíčů ale také i shodných IV. Tak je zvýšena bezpečnost šifry, která v takovém módu pracuje. Ač se může zdát, že tento mód zajišťuje integritu zprávy, není to pravda.



Obrázek 2.2: Šifrování a dešifrování v CBC módu

2.4 Algoritmy

2.4.1 DES

Vytvořen firmou IBM, v roce 1975 přijat jako šifrovací standard Americkým národním úřadem pro standardy (NSB). Původně byl zamýšlen pro 5-ti leté období, ale používal se 15 let. Jedná se o symetrickou blokovou šifru. Používá 64 bitů dlouhé bloky, které jsou šifrovány v 16 rundách a pouze 56-bitový klíč, což se časem projevilo jako největší slabina algoritmu. Prudký vývoj výpočetní síly počítačů a hardwaru umožnil útokem hrubou silou [1] šifru rozluštit za nebezpečně krátký čas. Například zařízení Copacobana [16] dokáže nalézt klíč průměrně za necelý týden. Jedním řešením tohoto problému bylo vytvoření varianty 3DES (Triple-DES). Ta otevřený text jednoduše zašifruje třemi běhy algoritmu DES. V jednotlivých bězích je použit jeden nebo dva odlišné klíče.[7] [3]

[1] *Útok hrubou silou (brute-force attack)* - proces zkoušení všech možných kombinací klíče, dokud dešifrovaný text nedává smysl. Pokud šifra neobsahuje žádné bezpečností chyby, pak je útok hrubou silou jediným možným útokem.

Advanced Encryption Standard - AES

V roce 1997 americký Národní Institut pro Standardy a Technologii (NIST) vyhlásil soutěž o nový šifrovací standard, který by nahradil nedostačující šifru DES a byl by schopen dostatečně zabezpečit citlivá vládní data.

Hodnotící kritéria pro výběr nového standardu byla rozdělena do tří hlavních kategorií:

- *Bezpečnost* - Odolnost vůči kryptoanalýze, spolehlivost matematických základů, na kterých je postaven, náhodnost výstupu a relativní bezpečnost vzhledem k ostatním kandidátům
- *Cena* - Cenou je myšlena efektivita algoritmu (tedy rychlost), paměťová náročnost a to, zda není zatížen licencí.
- *Charakteristiky algoritmu a jeho implementace* - Kritériemi byla jednoduchost algoritmu, flexibilita a možnost jeho softwarové i hardwarové implementace.

V roce 1998 bylo představeno 15 kandidátů, do užšího výběru se jich v roce 1999 dostalo pět: MARS, RC6, Rijndael, Serpent a Twofish. Po dlouhých a podrobných analýzách jednotlivých algoritmů byl v roce 2000 za AES zvolen Rijndael. Podrobněji v [5].

2.4.2 MARS

Symetrická bloková šifra vyvinutá firmou IBM. Používá 128-bitový blok dat a volitelnou délku klíče (128-400 bitů). Je postavena na Feistelově síti 3.typu, má kombinovanou strukturu, díky které by měla být odolnější i vůči útokům, které v době jejího vzniku neexistovaly. Kombinuje operace xor, sčítání, odčítání, vyhledávání v tabulkách (S-box), násobení a pevné, na datech závislé rotace. Navržena pro optimální využití operací podporovaných dnešními počítači. Rychlost šifrování a dešifrování je průměrná a na širokém spektru platforem (8-64 bitů) vyrovnaná. Nabízí vyšší bezpečnost než 3DES, je však rychlejší než DES. [2]

2.4.3 RC6

Byl navržen Ronem Rivestem. Je nástupcem RC5, na který sice nebyl nikdy proveden úspěšný útok, studie však odhalily možné teoretické útoky, zejména díky faktu, že počet rotací je nezávislý na všech bitech registru. Na RC5 byly provedeny úpravy jednak zabraňující těmto teoretickým útokům, jednak takové, aby splňoval podmínky na AES. Tak vznikl RC6. Použitím násobení bylo dosaženo velkého rozptylu na jednu rundu, což se odrazilo v menším počtu rund, větší bezpečnosti a rychlosti. Zejména na moderních procesorech, které dokáží efektivně násobit 32bitová čísla, poráží v rychlosti šifrování algoritmus Rijndael. V ostatních implementacích je průměrně rychlý. Oproti jiným algoritmům nepoužívá vyhledávácí tabulky, tudíž je nenáročný na paměť. Čas šifrování / dešifrování je nezávislý na datech, je tedy odolný vůči časovaným útokům. Nejsou známy žádné slabé klíče. [10] [11]

2.4.4 Rijndael

Autory algoritmu jsou belgičtí kryptografové Joan Daemen a Vincent Rijmen. Jedná se o velmi bezpečnou symetrickou blokovou šifru, která je navíc v porovnání s ostatními kandidáty na AES velmi rychlá a jednoduchá na implementaci. Má volitelné délky bloku dat

i klíče (128 až 256 bitů), které určují počet rund (10 až 14) potřebných pro zašifrování jednoho datového bloku a zároveň umožňují volit z více stupňů zabezpečení pro jakoukoliv aplikaci. Je použitelný pro implementaci softwarovou i hardwarovou, v obou dosahuje vysoké propustnosti, která však klesá při použití delších klíčů. Má nízké paměťové nároky, což jej činí vhodným pro zařízení s omezenými prostředky, ve kterých dosahuje nejvyšší propustnosti ze všech kandidátů na AES. Používá operace, které nejlépe odolávají útokům pomocí časové a odběrové analýzy. Jeho matematická struktura byla sice kritizována, nicméně bez vlivu na útoky. [7] [5]

2.4.5 Serpent

Autory této 128-bitové šifry jsou R. Anderson, E. Biham a L. Knudsen. Ve finálním hlasování o výběr AES se umístila na druhém místě. Je dost podobná Rijndaelu. Oproti němu je bezpečnější, ale pomalejší. Důvodem obojího je větší počet rund, používá jich 32. Nejlepší útoky prolomily šifru při použití 11 rund, má tedy velkou bezpečnostní rezervu. Jeho architektura podporuje tzv. "bitslice" implementaci. Ta spočívá v napodobení logických hradel v hardwaru tak, že např. 32-bitový procesor šifruje 32 bloků paralelně. Efektivita je vyšší než u obvyklé implementace, jelikož je celou dobu využito všech 32 bitů procesoru, ne pouze několik při dílčích operacích. [15]

2.4.6 Twofish

Schneierem upravená šifra Blowfish, splňující podmínky na AES. Délka bloku 128 bitů, klíč volitelné délky 128, 192 nebo 256 bitů. Jedná se o 16-ti rundovou Feistelovu síť s bijektivní funkcí "F". Inovací jsou na klíči závislé S-Boxy. Důkladně analyzována, opět nepatentována. Pomalé generování subklíčů, rychlost šifrování/dešifrování je hodně závislá na platformě (optimalizována pro 32-bitové procesory). Má široké možnosti přizpůsobení implementace z hlediska rychlosti, požadavků na paměť a času inicializace klíče. [12]

Tento výčet není úplný, byly zmíněny pouze nejznámější a nejpoužívanější šifry.

Síla šifer

Není vždy nutné používat nejsilnější možný algoritmus. U tajných zpráv, jejichž doba platnosti či cena není vysoká, je zbytečné používat silné šifry, zvláště pokud je taková šifra výpočetně náročná. [3]

Příklad:
Pracovnice pobočky bankovní instituce pracuje s terminálem pro správu účtů klientů banky. Z důvodu bezpečnosti dat je každý den platné jiné heslo pro přístup do systému. Toto heslo je každé ráno na pobočku zasíláno z centrály banky. Pro zabezpečení přenosu hesla stačí použít šifru, na kterou útok hrubou silou na nejvýkonnějším v současnosti dostupném hardwaru trvá déle než jeden den, tedy po dobu, po kterou heslo platí. Samozřejmě je nutné myslet na budoucnost a použít jistou rezervu.

2.5 Šifrování na FITkitu

Dosavadní práce, týkající se kryptografie na platformě FITkit, se zabývaly implementací šifry AES v jazyce C i VHDL [4], šifrováním telefonních hovorů [13], šifrovanou komunikací mezi PC a FITkitem [6] a hašovacím algoritmem MD5 [8]. Ještě je tu dostatek prostoru pro

prozkoumání možností šifrování na platformě FITkit. I přes své poměrně omezené zdroje má jistě co nabídnout.

2.6 Výběr šifry

Nyní stojíme před otázkou, která šifra bude vhodná k implementaci na platformě FITkit. Jelikož má tato platforma omezené výpočetní prostředky, bude vhodnější zvolit některou ze symetrických šifer. Navíc je potřeba ponechat rezervu pro obvody komunikace s uživatelem (klávesnice, LCD displej), komunikační rozhranní RS-232 a obvody pro zajištění komunikace mezi MCU a FPGA prostřednictvím sběrnice SPI. Při výběru vhodné šifry jsem vycházel z [14]

Algoritmus **DES** je v dnešní době snadno prolomitelný ve velmi krátkém čase [3], proto bych považoval jeho implementaci za "zpátečnickou".

Výhodou **MARSu** je kompaktnost, nevýhodou je nejen vysoká latence díky třem odlišným typům rund, hojnému použití S-boxu a komplikovanému generování subklíčů, ale i použití násobení, které je v hardwaru drahé.

I když má **RC6** dobré možnosti ke snížení velikosti na úkor rychlosti, je stále dosti náročným na výpočetní zdroje.

Serpent je velmi podobný DESu, používá operace vhodné pro implementaci v hardwaru. Ovšem jeho S-Boxy v hardwaru zaberou značnou část prostředků. Podle srovnávacích implementací ve výše uvedeném dokumentu nejvíc ze všech kandidátů na AES.

Rijndael a **Twofish** jsou nejméně náročné na zdroje FPGA, proto by byly nejvhodnější pro implementaci na FITkitu.

Rozhodl jsem se pro algoritmus Rijndael díky jeho rozšířenosti a z toho vyplývající větší možnosti spolupráce s jinými zařízeními. Jeho kompletní implementace (myšleno šifrování i dešifrování) je objemnější než v případě algoritmu Twofish, avšak implementace pouze jedné funkce je méně objemná. Implementace v [4], která obsahuje pouze šifrování, zabrala 75 % CLB slices FPGA na FITkitu. Proto jsem se rozhodl pro jednosměrnou komunikaci, kdy jeden FITkit bude pouze šifrovat a druhý pouze dešifrovat.

Pozn.: dále v textu bude algoritmus Rijndael označován jako AES.

Kapitola 3

Návrh řešení

V této kapitole bude popsán jak algoritmus AES, tak i vlastní implementace celého systému. Informace týkající se specifikace AES jsou převzaty z [1].

3.1 Matematická příprava

Pro pochopení matematického pozadí šifry AES je nutné si vysvětlit některé operace v Galoisově poli GF(2^8), protože byty jsou interpretovány jako prvky tohoto pole.

3.1.1 Sčítání

Součet dvou prvků odpovídá sečtení stejných mocnin v polynomiálním reprezentaci těchto prvků. Prakticky je součet proveden operací XOR. Operace odčítání je naprosto stejná.

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2 \quad \text{(polynomiální reprezentace)}$$

$$\{01010111\} \oplus \{10000011\} = \{11010100\} \quad \text{(binární reprezentace)}$$

$$\{57\} \oplus \{83\} = \{d4\} \quad \text{(hexadecimální reprezentace)}$$

3.1.2 Násobení

Násobení v polynomiální reprezentaci odpovídá násobku polynomů modulo neredukovatelným polynomem [1] 8. stupně. Pro AES je tento polynom:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

což odpovídá $\{01\}\{1b\}$ v hexadecimální reprezentaci.

Operace modulo m(x) zajišťuje, aby byl výsledný polynom menšího stupně než 8. a bylo možné jej reprezentovat jako byte. Násobení neodpovídá žádná jednoduchá operace na bytové úrovni, jako je tomu u sčítání.

[1] Polynom je neredukovatelný, pokud je dělitelný pouze číslem 1 a sám sebou.

Příklad:

$$\{57\} \bullet \{83\} = \{c1\}$$

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) = x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \; modulo \; x^8 + x^4 + x^3 + x + 1 = x^7 + x^6 + 1 = \{c1\}$$

Násobení polynomem x

Výsledkem násobení binárního polynomu s polynomem x (tedy bytem $\{00000010\} = \{02\}$) je polynom:

$$b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

Konečný výsledek $x \bullet b(x)$ získáme redukováním předchozího modulo $m(x)$. Pokud je nejvyšší bit $b_7 = 0$, výsledek je již v redukované formě. Pokud $b_7 = 1$, provedeme redukci operací XOR s polynomem $m(x)$. Na bytové úrovni lze tedy násobení číslem $\{02\}$ implementovat jako posun o jeden bit doleva následovaný podmíněným XORem s hodnotou $\{1b\}$. Tato operace je označována jako *xtime()*. Násobení většími čísly než $\{02\}$ lze realizovat opakovanou operací *xtime()* se sčítáním mezivýsledků, viz. následující příklad:

$$\begin{aligned}
&\{57\} \bullet \{13\} = \{fe\} \\
&\{57\} \bullet \{02\} = xtime(\{57\}) = \{ae\} \\
&\{57\} \bullet \{04\} = xtime(\{ae\}) = \{47\} \\
&\{57\} \bullet \{08\} = xtime(\{47\}) = \{8e\} \\
&\{57\} \bullet \{10\} = xtime(\{8e\}) = \{07\}
\end{aligned}$$

nyní mezivýsledky sečteme:

$$\{57\} \bullet \{13\} = \{57\} \bullet (\{01\} \oplus \{02\} \oplus \{10\}) = \{57\} \oplus \{ae\} \oplus \{07\} = \{fe\}$$

3.1.3 Polynomy s koeficienty v GF(2^8)

Koeficienty těchto polynomů jsou samy prvky GF(2^8), tedy byty, narozdíl od předešlých polynomů, jejichž koeficienty představovaly bity. Tyto polynomy maximálně 3. stupně jsou definovány takto:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

Mohou být také reprezentovány jako slovo ve formátu $[a_0, a_1, a_2, a_3]$.

Sčítání těchto polynomů se provádí jako součet koeficientů odpovídajících stupňů, což odpovídá XORu odpovídajících bytů. To můžeme zjednodušit na XOR celých slov. Součet dvou polynomů a(x) a b(x) s koeficienty v GF(2^8) tedy vypočítáme:

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

Násobení je prováděno ve dvou krocích. Prvním je roznásobení rovnice $c(x) = a(x) \bullet b(x)$ a jeho úprava do následujícího tvaru:

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

kde:

$$\begin{aligned}
c_0 &= a_0 \bullet b_0 \\
c_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \\
c_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \\
c_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3 \\
c_4 &= a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\
c_5 &= a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\
c_6 &= a_3 \bullet b_3
\end{aligned}$$

Výsledek $c(x)$ není 4-bytovým slovem. Proto je druhým krokem redukce $c(x)$ modulo $x^4 + 1$, čímž získáme polynom maximálně 3. stupně. Ten již může být reprezentován jako 4-bytové slovo:

$$d(x) = d_3x^3 + d_2x^2 + d_1x + d_0$$

kde:

$$\begin{aligned}
d_0 &= (a_0 \bullet b_0) \oplus (a_3 \bullet b_1) \oplus (a_2 \bullet b_2) \oplus (a_1 \bullet b_3) \\
d_1 &= (a_1 \bullet b_0) \oplus (a_0 \bullet b_1) \oplus (a_3 \bullet b_2) \oplus (a_2 \bullet b_3) \\
d_2 &= (a_2 \bullet b_0) \oplus (a_1 \bullet b_1) \oplus (a_0 \bullet b_2) \oplus (a_3 \bullet b_3) \\
d_3 &= (a_3 \bullet b_0) \oplus (a_2 \bullet b_1) \oplus (a_1 \bullet b_2) \oplus (a_0 \bullet b_3)
\end{aligned}$$

V maticovém tvaru:

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

Protože $x^4 + 1$ není neredukovatelným polynomem nad GF(2^8), není násobení plně invertibilní. Proto AES specifikuje polynom, ke kterému existuje inverze:

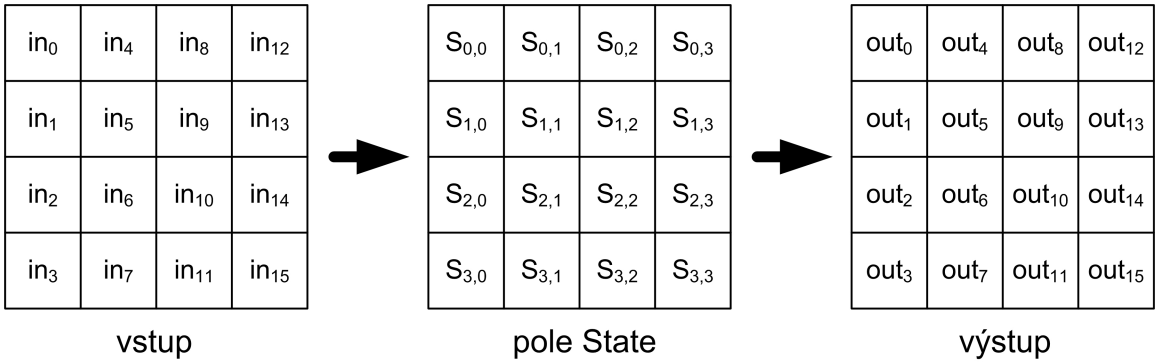
$$\begin{aligned}
a(x) &= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \\
a^{-1}(x) &= \{0b\}x^3 + \{0d1\}x^2 + \{09\}x + \{0e\}
\end{aligned}$$

3.2 Popis AES

AES je blokový symetrický šifrovací algoritmus. Standard AES pracuje se 128 bitovými bloky dat, které šifruje a dešifruje klíči o délkách 128, 192 a 256 bitů, ačkoliv samotný algoritmus Rijndael podporuje pro délky bloků i klíčů hodnoty 128, 160, 192, 224 a 256 bitů .

3.2.1 Stav (state) *s*

Stav je dvourozměrné pole velikosti 4 řádky * 4 sloupce, prvkem pole je byte. Na počátku se do něj vloží blok otevřeného resp. šifrovaného textu (v případě šifrování resp. dešifrování), v průběhu šifrování jsou do něj ukládány výsledky dílčích transformací a na konci šifrování je z něj přečten zašifrovaný resp. dešifrovaný blok dat.



Obrázek 3.1: Vstup a výstup dat do a ze stavu

3.2.2 Algoritmus šifrování

Základem algoritmu je runda, což je jedna iterace, která se skládá ze čtyř transformací:

- SubBytes
- ShiftRows
- MixColumns
- AddRoundKey

Na počátku je blok plaintextu uložen do stavu a je provedena pouze transformace AddRoundKey, jejímiž vstupy jsou sloupce stavu a 4 slova šifrovacího klíče. Poté je stav transformován skrze **Nr** rund, přičemž v poslední rundě je vynechána operace MixColumns. Tím je zašifrování jednoho 128b bloku ukončeno, výsledek je uložen ve stavu. Jednotlivé transformace jsou vysvětleny dále.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(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
```

Obrázek 3.2: Algoritmus šifrování

	Délka klíče (Nk slov)	Délka bloku (Nb slov)	Počet rund (Nr)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Tabulka 3.1: Přehled konstant pro různé délky klíčů

SubBytes

Transformace SubBytes nahrazuje každý byte stavu pomocí nelineární substituce. K tomu používá tzv. S-box, ten je vytvořen pomocí dvou transformací:

1. Multiplikativní inverze v GF(2^8),
2. Afinní transformace nad GF(2^8).

S-box můžeme vyjádřit tabulkou hodnot v hexadecimálním tvaru, tu najdeme v C

Například substituci bytu hodnoty 73h provedeme nalezením odpovídající hodnoty v řádku 7 a sloupci 3. Nová hodnota tedy bude 8Fh.

ShiftRows

Transformace ShiftRows jednoduše rotuje řádky stavu o různý počet bytů doleva, tedy na nižší index c v rámci řádku. První řádek zůstává beze změny. Rotace jednotlivých řádků je

dána následující rovnicí:

$$s'_{r,c} = s_{r,(c+shift(r,Nb))modNb} \quad \text{pro } 0 < r < 4 \text{ a } 0 \leq c < \mathbf{Nb}$$

Například pro $Nb = 4$ (AES-128) není první řádek stavu rotován vůbec, druhý řádek o jeden byte, třetí o dva a čtvrtý o tři byty.

MixColumns

Tato transformace pracuje se stavem po sloupcích. S každým sloupcem pracuje jako s polynomem nad GF(2^8) a násobí jej modulo $x^4 + 1$ fixním polynomem:

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

což můžeme vyjádřit jako násobení matic $s'(x) = a(x) \otimes s(x)$:

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

vynásobením matic získáme následující rovnice:

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \end{aligned} \tag{3.1}$$

AddRoundKey

Transformace AddRoundKey provádí exkluzivní logický součet (XOR) každého sloupce stavu s jedním z **Nb** slov $[w_i]$ odpovídajícího rundovního klíče z Key Schedule:

$$\left[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}\right] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{round} * Nb + c] \quad \text{pro } 0 \leq c < \mathbf{Nb}$$

Jak již bylo zmíněno, poprvé se transformace AddRoundKey provádí před první rundou, kdy je round = 0 a z Key Schedule jsou tedy použita slova $[w_0]$ až $[w_{Nb-1}]$, což je původní šifrovací klíč.

3.2.3 Expanze klíče

KeyExpansion slouží k vygenerování sady rundovních klíčů z šifrovacího klíče **K**. Tato sada je nazývána *Key Schedule* a tvoří ji (Nr+1) rundovních klíčů, z nichž každý má **Nb** 32-bitových slov. Key Schedule je jednorozměrné pole slov $[w_i], 0 \leq i \ < \mathbf{Nb}(\mathbf{Nr} + \mathbf{1})$.

Používají se transformace SubWord, která je až na 32bitovou šířku dat shodná s transformací SubBytes a RotWord, což je prostá cyklická rotace slova o jeden byte doleva. Pole rundovních konstant **Rcon** obsahuje 32bitové konstanty $[x^{i-1},\{00\},\{00\},\{00\}]$, kde x^{i-1} je mocninou $x = 02$ v poli GF(2^8), $i \geq 1$.

Prvních **Nk** slov Key Schedule tvoří zkopírovaný šifrovací klíč. Každé následující slovo **w[i]** se vypočítá jako exkluzivní logický součet předcházejícího slova **w[i-1]** a slova **w[i-Nk]**. Pokud je **i** násobkem **Nk**, jsou na **w[i-1]** aplikovány transformace RotWord a SubWord, následované operací XOR s rundovní konstantou **Rcon[i]**. V případě 256-bitového šifrovacího klíče a splnění podmínky $i-4 \; mod \; Nk = 0$ je na **w[i-1]** aplikována transformace SubWord.

```
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
```

Obrázek 3.3: Algoritmus expanze klíče

3.2.4 Algoritmus dešifrování

Algoritmus dešifrování se od šifrovacího liší pouze odlišným pořadím transformací a sestupným číslováním prováděných rund.

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

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

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

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

  out = state
end
```

Obrázek 3.4: Algoritmus dešifrování

InvShiftRows

Jedná se o inverzní transformace k ShiftRows. Poslední tři řádky stavu jsou rotovány o různý počet bytů "doleva". Rotace jednotlivých řádků je dána následujícím vzorcem:

$$s'_{r,(c+shift(r,Nb))modNb} s_{r,c} \text{ for } 0 < r < 4 \text{ and } 0 \leq c < \mathbf{Nb}$$

InvSubBytes

Tato transformace, stejně jako SubBytes, nahrazuje všechny byty stavu pomocí substituční funkce. Ta je nyní složena z:

1. Inverze afinní transformace nad GF(2^8),
2. Multiplikativní inverze v GF(2^8).

Inverzní S-Box ve formě tabulky hexadecimálních hodnot najdete v C

InvMixColumns

Tato transformace pracuje se stavem po sloupcích. S každým sloupcem pracuje jako s polynomem nad GF(2^8) a násobí jej modulo $x^4 + 1$ fixním polynomem:

$$a(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$$

což můžeme vyjádřit jako násobení matic $s^{'}(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s^{'}_{0,c} \\ s^{'}_{1,c} \\ s^{'}_{2,c} \\ s^{'}_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

vynásobením matic získáme následující rovnice:

$$\begin{aligned} s^{'}_{0,c} &= (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s^{'}_{1,c} &= (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c}) \\ s^{'}_{2,c} &= (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c}) \\ s^{'}_{3,c} &= (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \end{aligned} \tag{3.2}$$

Inverzní transformace k AddRoundKey

Jedná se o totožnou transformaci s AddRoundKey, jelikož obsahuje pouze operaci XOR.

3.3 Mapování na platformu FITkit

3.3.1 FITkit

FITkit [17] je platforma určená pro rozšíření výuky programování hardwaru na FIT VUT v Brně. Skládá se z následujících komponent (platí pro FITkit verze 1.2):

- 16-bitový RISC procesor TI MSP430F168IPM (48 kB FLASH a 2 kB RAM paměti),
- FPGA pole Xilinx XC3S50-4PQ208C (Spartan 3A),
- DRAM 64 Mbit,
- Alfanumerická klávesnice (16 tlačítek),
- Řádkový LCD displej (16 znaků),
- Konektory RS232, VGA, PS/2,
- Audio rozhranní.

FPGA pole Xilinx XC3S50

Toto rekonfigurovatelné hradlové pole je jedním z nejlevnějších, nevyniká tedy výkonem. Je ideální pro použití ve spotřební elektronice, jako jsou například domácí síťová a zobrazovací zařízení. Obsahuje 50 000 logických hradel, 1728 logických buněk, 192 CLB (konfigurovatelných logických bloků), z nichž každý tvoří 4 "slice". Každá slice obsahuje dva 4-vstupové funkční generátory (LUT - LookUpTable) a dva paměťové klopné obvody. Dále v něm najdeme 4 dvouportové paměti BlockRAM o kapacitě 2kB a 4 násobičky 18x18 bitů. [18]

3.3.2 Implementace

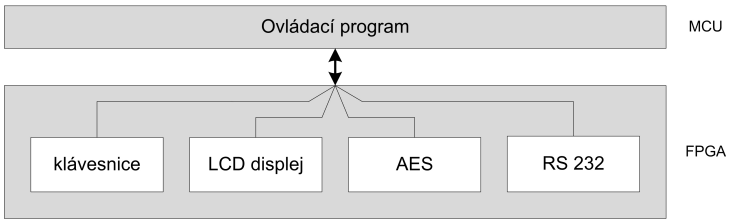
Z důvodů omezených prostředků FPGA pole podporuje implementace pouze 128-bitový šifrovací klíč (označováno jako AES-128) a ze stejných důvodů musela být realizována pouze jednosměrná komunikace mezi FITkity. Jeden FITkit (dále označovaný jako šifrovací modul) tedy umožňuje zprávy pouze psát, šifrovat a odesílat a druhý (dešifrovací modul) umožňuje jejich příjem, dešifrování a zobrazení. Komunikace mezi FITkity je zajištěna pomocí rozhranní RS-232, konkrétně jsem použil null-modem kabel. Aplikaci je možno jednoduše přeložit a nahrát pomocí terminálu QDevKit. K syntéze VHDL kódu bylo použito prostředí Xilinx ISE WebPack 9.1i, program pro MCU byl přeložen sadou MSP-430 s GCC verze 3.2.3. Vycházel jsem z aplikace Piškvorky, která také řeší komunikaci mezi FITkity přes rozhranní RS-232. Tuto aplikaci lze nalézt v SVN FITkitu [17].

V procesoru běží program, který má na starosti zasílání a příjem dat do a z jednotlivých komponent uvnitř FPGA pole a interakci s uživatelem (vstup dat pomocí klávesnice a výstup pomocí LCD displeje), viz obr. 3.5.

3.3.3 Top-level entita

Z dostupných top-level entit je použita enita *tlv_pc_ifc*, která zpřístupňuje PC rozhranní FITkitu (RS-232, VGA a PS/2). K aktivaci tohoto rozhranní je nutné, aby byla propojka J6 spojená. Architektura této entity je popsána v souboru message.vhd. Tato architektura obsahuje řadiče klávesnice, LCD displeje, řadič přerušení, řadič rozhranní RS-232 a šifrovací

komponentu Encryptor (resp. dešifrovací Decryptor). Ke každé komponentě náleží adresový dekodér sběrnice SPI, která je na FITkitu využita pro komunikaci mezi MCU a FPGA polem.
Další text bude zaměřen pouze na popis šifrovacího (resp. dešifrovacího) obvodu v FPGA, podpůrné komponenty zde nebudu popisovat, veškeré informace k nim lze nalézt na [17].



Obrázek 3.5: Návrh systému

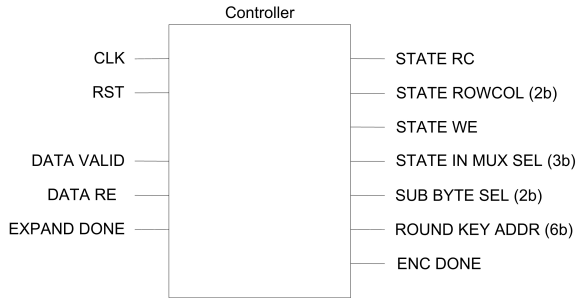
3.3.4 Šifrovací modul

Šifrovací modul umožňuje zadání šifrovacího klíče a samotné zprávy pomocí klávesnice, zašifrování zprávy a její odeslání pomocí rozhranní RS-232.

Po spuštění modulu je uživatel vyzván k zadání šifrovacího klíče pomocí klávesnice. Zadávání alfanumerických znaků je stejné jako např. na mobilním telefonu, tj. pro zadání znaku, který se na této klávese nachází na druhé a další pozici, je potřeba opakovaný stisk této klávesy v určeném časovém intervalu. Znak mezera je na klávese 1, písmeno Z na klávese 0. Klíč může být maximálně 16 znaků dlouhý (= 128 bitů). Protože se jedná o demonstrační verzi, znaky klíče jsou během zadávání zobrazovány na displeji. Pro reálné použití by byla vhodná jednoduchá úprava pro zobrazování zástupných znaků (např. hvězdiček) místo znaků klíče. Během zadávání je možno klávesou C přepínat velikost písmen a klávesou D smazat poslední znak. Po potvrzení klávesou A je klíč odeslán do šifrovací komponenty Encryptor a uživatel je vyzván k zadání zprávy. V případě, že je zpráva delší jak 16 znaků (velikost LCD displeje), může si ji uživatel posunovat pomocí klávesy * doleva a klávesou # doprava. Po zadání zprávy a stisku klávesy B je přes RS-232 odeslán *NOB - Number Of Block* byte, jehož hodnota je počet 128-bitových (tedy 16-bytových) bloků, do kterých je nutno zprávu rozdělit, následovaný 15-ti náhodnými byty, reprezentujícími zbytek IV. Poté je první blok zprávy po XORu s IV zaslán do komponenty Encryptor. Ta po zašifrování bloku vyvolá přerušení, obslužný program v MCU z něj přečte zašifrovaný blok, uloží jej zpět na jeho místo ve zprávě a po znacích jej pošle přes RS-232. Poté je proveden XOR dalšího bloku se zašifrovaným předchozím a zaslán k zašifrování. Tak se celý cyklus opakuje až do konce zprávy. Pokud délka zprávy není dělitelná 16-ti znaky, je poslední blok doplněn na plnou délku znaky o hodnotě 00_h. Po zašifrování a odeslání celé zprávy je možno odeslat další, k jejímu zašifrování bude použit stejný šifrovací klíč a nově vygenerovaný IV. Následuje popis jednotlivých komponent v FPGA, ze kterých se šifrovací obvod skládá. Jeho celé schéma lze nalézt v příloze B.1.

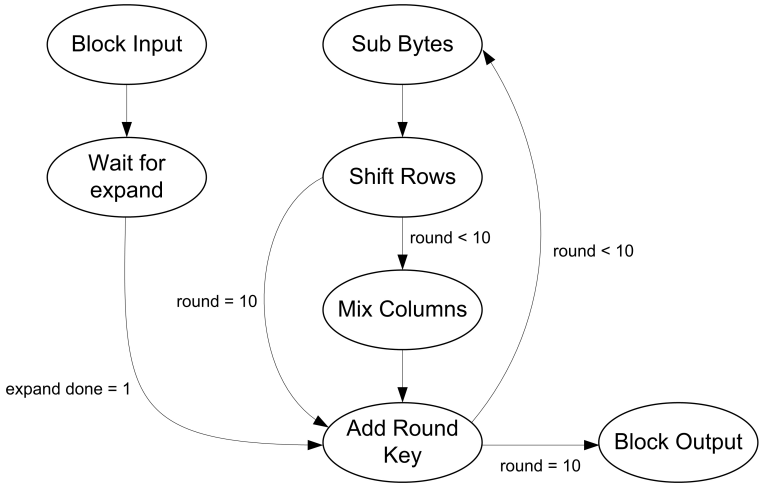
Komponenta Controller

Jedná se o řídící blok celého šifrovacího obvodu realizovaný jako Mealyho konečný automat.



Obrázek 3.6: Komponenta Controller

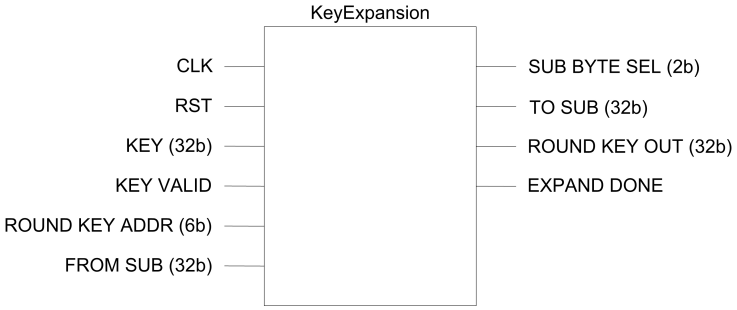
Aktivní signál `DATA_VALID` oznamuje platná slova bloku dat k zašifrování, která je třeba zapsat do stavu. To se provede přepojením datového vstupu stavu na vstup šifrovacího obvodu pomocí `STATE_IN_MUX_SEL`, určením způsobu adresace stavu po sloupcích (`STATE_RC = 1`), vystavením adresy sloupce na `STATE_ROWCOL` a povolením zápisu do stavu signálem `STATE_WE`. Poté se čeká na dokončení expanze klíče jednotkou *KeyExpansion*, která dokončení oznámí aktivací signálu `EXPAND_DONE`.
Nyní může začít proces šifrování. V jeho průběhu je pomocí signálu `STATE_IN_MUX_SEL` přepínán vstup stavu na výstup komponenty, zajišťující aktuální transformaci. Adresování stavu je zajištěno signály `STATE_RC` a `STATE_ROWCOL`. V případě transformace AddRoundKey je na `ROUND_KEY_ADDR` vystavována adresa slova rundovního klíče, které je uloženo v komponentě *KeySchedule*.
Ukončení šifrování bloku je oznámeno aktivací signálu `CIPH_DONE` napojeného na řadič přerušení a na výstup stavu je vystaven první sloupec stavu. Přerušení je zpracováno a je zahájeno čtení zašifrovaného bloku do MCU. Při čtení je dekodérem SPI sběrnice aktivován signál připojený na vstup `DATA_RE`, který oznámí přečtení prvního sloupce. Poté můžeme postupně vystavit další sloupce. Po přečtení posledního čeká komponenta na vstup dalšího bloku k zašifrování.



Obrázek 3.7: Diagram přechodů mezi skupinami stavů konečného automatu komponenty Controller

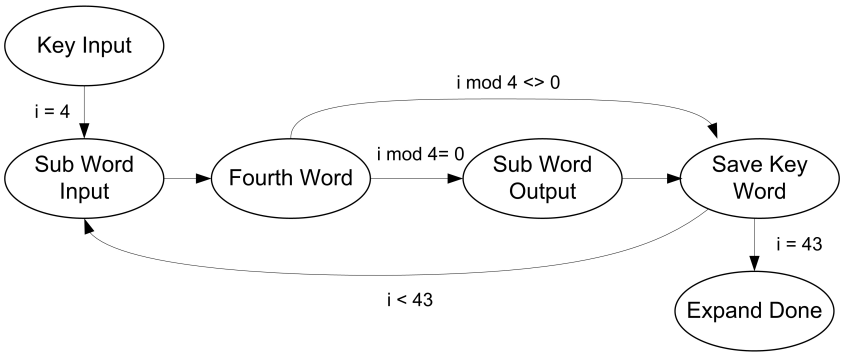
Komponenta KeyExpansion

Tato komponenta má na starosti expanzi šifrovacího klíče, vysvětlenou v 3.2.3. Tato kompnenta v sobě zahrnuje také komponentu pro uložení rundovních klíčů *KeySchedule*, která bude popsána dále.



Obrázek 3.8: Komponenta KeyExpansion

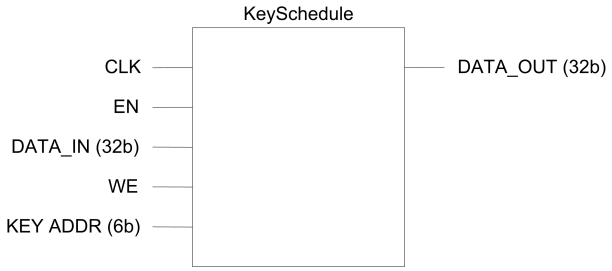
Na počátku čeká komponenta na přijmutí šifrovacího klíče na 32-bitovém portu `KEY`. To je komponentě oznámeno dekodérem SPI sběrnice prostřednictvím aktivního signálu na vstupu `KEY_VLD`. Šifrovací klíč je uložen do *KeySchedule* a je započata samotná expanze. Protože je jednou z jejích operací operace *SubWord()*, která je shodná s operací *SubBytes()*, je pro její realizaci využita komponenta *SubBytes*. Vstup do této komponenty je skrze `TO_SUB` a výstup je na portu `FROM_SUB`. Selekce bytu pro substituci je prováděna pomocí 2-bitového portu `SUB_BYTE_SEL`.
Ukončení expanze je dalším komponentám oznámeno aktivním signálem `EXPAND_DONE`. Poté komponenta zůstává ve stavu, kdy umožňuje čtení rundovních klíčů z komponenty *KeySchedule*. Ty jsou adresovány pomocí portu `ROUND_KEY_ADDR` a vystavovány na `ROUND_KEY_OUT`. Expanze 128-bitového klíče trvá od načtení klíče po ukončení expanze 211 hodinových taktů.



Obrázek 3.9: Diagram přechodů mezi skupinami stavů konečného automatu komponenty KeyExpansion

Komponenta KeySchedule

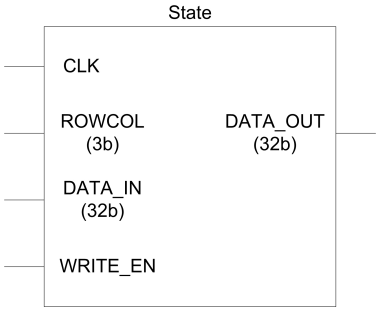
Jedná se o paměť pro uložení rundovních klíčů vzniklých expanzí šifrovacího klíče. V případě AES-128 je potřeba uložit 44 32-bitových slov (viz. 3.2.3). To už je v případě FPGA pole FITkitu nezanedbatelný prostor, proto je tato paměť implementována v BlockRAM.



Obrázek 3.10: Komponenta KeySchedule

Komponenta State

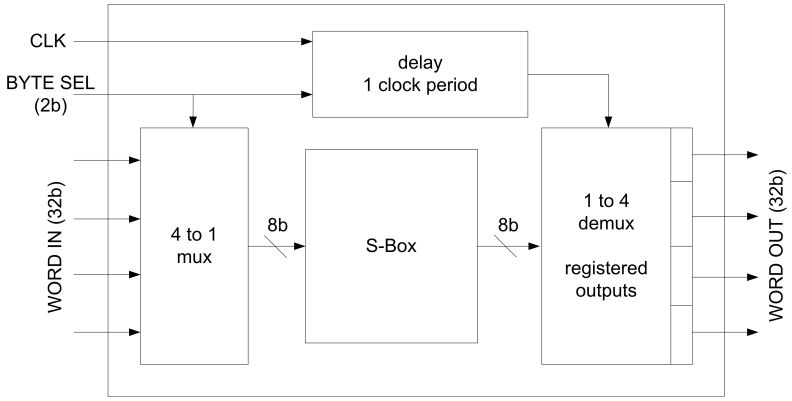
Tato komponenta slouží k uchování mezivýsledků jednotlivých transformací, všechny z ní čtou a zapisují do ní. Proto je této komponentě předřazen 5-kanálový multiplexer, na jehož vstupy jsou přivedeny výstupy všech 4 transformačních komponent a vstup bloku plaintextu k zašifrování. Protože transformace *ShiftRows()* pracuje se stavem po řádcích, zatímco ostatní transformace po sloupcích, je nutné ze stavu číst a zapisovat do něj oběma způsoby. Z tohoto důvodu nebylo možné tuto komponentu implementovat v BlockRAM. Paměť je adresována pomocí 3b signálu `ROWCOL`, jehož nejvyšší bit určuje způsob přístupu (po řádcích, resp. po sloupcích) a zbývající dva bity označují číslo řádku, resp. sloupce.



Obrázek 3.11: Komponenta State

Komponenta SubBytes

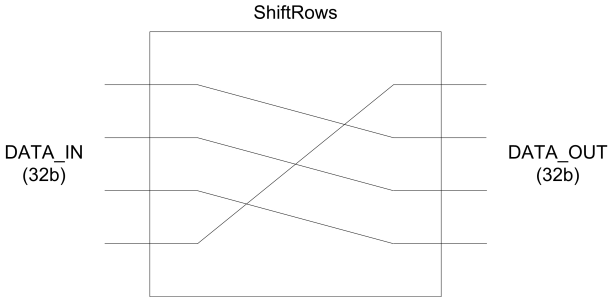
Multiplexer pomocí `BYTE_SEL` postupně vybírá byty vstupního slova, reprezentujícího sloupec stavu, přivádí je na vstup S-Boxu, který provede jejich nahrazení. Pomocí demultiplexeru s registrovanými výstupy je na výstupu jednotky zpět složen celý substituovaný sloupec připravený k zápisu do stavu. Příklad substituce slova lze nalézt v D.2. S-box je implementován v paměti BlockRAM, která je využita jako ROM pamět. Jejím obsahem jsou substituované hodnoty uložené na adresách odpovídajících vstupním bytům. Nahrazovaný byte je tedy přiveden na adresový vstup paměti a z výstupního portu je přečten substituovaný byte. Tato transformace trvá nejdelší dobu, 33 hodinových taktů. Je to zapříčiněno substitucí po bytech, která byla nutná z důvodu optimalizace velikosti obvodu.



Obrázek 3.12: Blokové schema komponenty SubBytes

Komponenta ShiftRows

Tato komponenta pouze změní pořadí bytů vstupního slova jejich jednoduchým prohozením tak, aby provedla rotaci bytů o jeden doleva. Toto řešení jsem zvolil pro jeho jednoduchost a úsporu prostředků FPGA pole. Pro transformaci řádku č.1 stačí jeden průchod, avšak řádky č.2 a 3 vyžadují 2, resp. 3 průchody, přičemž vždy je výstup předchozího průchodu zapsán do stavu a v dalším hodinovém cyklu je z něj přečten pro další průchod. Těchto 6 průchodů trvá 18 hodinových taktů. Simulace této komponenty je v D.1.



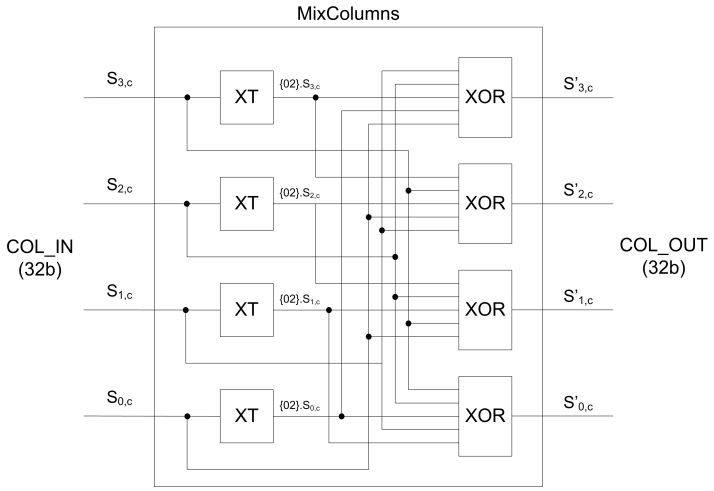
Obrázek 3.13: Komponenta ShiftRows

Komponenta MixColumns

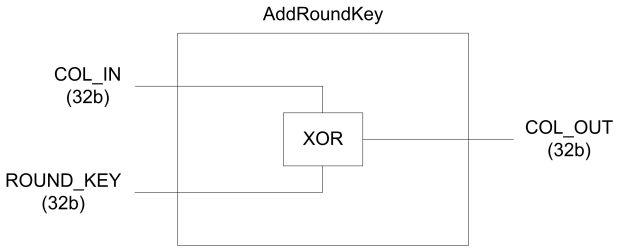
Tato komponenta realizuje rovnice 3.1. Při násobení konstantami je využito násobení polynomem x, popsaném v 3.1.2. Protože je nejvyšší konstantou hodnota $\{03\}$, vystačíme s vynásobením bytů vstupního slova hodnotou $\{02\}$ v GF^2, tedy jednou úrovní operace *xtime()* (na následujícím obrázku se jedná o blok XT). Celá transformace trvá 12 hodinových taktů.

Komponenta AddRoundKey

Tato komponenta představuje pouze 32b XOR dvou vstupů - slova rundovního klíče z Key Schedule (port `ROUND_KEY`) a odpovídajícího sloupce stavu (port `COL_IN`) dle rovnice 3.2.2. Tato transformace trvá 12 hodinových taktů.



Obrázek 3.14: Komponenta MixColumns



Obrázek 3.15: Komponenta AddRoundKey

3.3.5 Dešifrovací modul

Dešifrovací modul umožňuje zadání šifrovacího klíče pomocí klávesnice, přijetí zprávy pomocí rozhranní RS-232 a její zobrazení na LCD displeji.

Po spuštění modulu je uživatel vyzván k zadání šifrovacího klíče pomocí klávesnice. Ten může být maximálně 16 znaků dlouhý (= 128 bitů). Během zadávání klíče je možno klávesou C přepínat mezi velkými a malými písmeny a klávesou D smazat poslední znak. Po potvrzení klíče klávesou A čeká na příjem zprávy. Jako první je přijat 16-ti bytový IV, jehož prvním bytem je hodnota NumberOfBlocks. Po přijetí počtu bloků, který odpovídá této hodnotě, je zpráva kvůli nutnosti XORu aktuálního dešifrovaného bloku s předchozím zašifrovaným blokem (první blok je XORován s IV) postupně od konce dešifrována a poté zobrazena na LCD displeji FITkitu. V případě, že je zpráva delší jak 16 znaků (velikost LCD displeje), může si ji uživatel posunovat pomocí klávesy * doleva a klávesou # doprava. Po stisknutí klávesy B je zpráva vymazána z paměti i z displeje a čeká se na příjem další zprávy.

Dešifrovací modul (schéma viz. příloha B.2) je až na několik výjimek shodný se šifrovacím modulem. Tyto výjimky budou nyní popsány:

Controller

V této komponentě je oproti šifrovacímu modulu odlišné pořadí provádění transformací a opačné sestupné číslování rund (viz. 3.2.4).

InvSubBytes

Tato komponenta provádí inverzní transformaci k *SubBytes*, používá proto jinou substituční tabulku **??**, způsob práce je však shodný.

InvMixColumns

Tato komponenta se oproti *MixColumns* liší na první pohled v jediné jednoduché věci, a to v hodnotách koeficientů polynomu, kterým se násobí vstupní sloupec, viy. rovnice 3.2. Ovšem toto násobení je realizováno operací *xtime()* 3.1.2. Protože je nejvyšší koeficient $\{0e\}$, potřebujeme tři stupně operací *xtime()*. Také se nám zvýší počet XORů pro součty jednotlivých násobků.

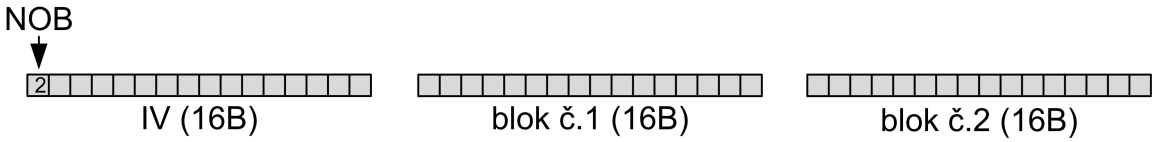
KeyExpansion

Expanze klíče zůstává naprosto beze změny. Uvádím ji zde z důvodu, že používá transformaci SubWord stále podle neinvertovaného S-Boxu a tedy jí vygenerovaný KeySchedule je při shodném šifrovacím klíči totožný s tím, který vygeneruje šifrovací modul. V dešifrovacím modulu je tedy zapotřebí S-Box i inverzní S-Box. Proto jsou v tomto modulu využity 3 BlockRAM paměti v FPGA poli FITkitu.

3.3.6 Přenos zpráv

Maximální délka zprávy je omezena na 1024 znaků z důvodu neexistence funkce *realloc* pro dynamickou realokaci proměnné uchovávající zprávu v knihovně použitého překladače. Prvotní myšlenka nahrazení realokace alokací nové paměti, překopírováním zprávy do ní a zrušením původní paměti se ukázala neprůchozí z toho důvodu, že by takto bylo možné využít maximálně polovinu volného paměťového prostoru (během kopírování z jednoho paměťového místa do druhého by musely být tyto oba přítomny v paměti).

Na obrázku 3.16 je zachycen formát zprávy při přenosu z jednoho FITkitu na druhý. Nultý blok je inicializační vektor (IV), jehož první byte obsahuje hodnotu NOB (NumberOfBlocks) a zbývající dva bloky obsahují samotnou zašifrovanou zprávu.



Obrázek 3.16: Formát šifrované zprávy dlouhé 2 bloky

3.3.7 Implementační problémy

Při implementaci šifrovacího modulu jsem se několikrát setkal s omezenými prostředky FPGA pole, konkrétně s množstvím slices. Proto bylo nutné některé komponenty implementovat v BlockRAM a jiné pozměnit tak, abych upřednostnil jimi zabrané zdroje před

počtem taktů, za jaký dokáží danou operaci provést. Nejvíce se to podepsalo na rychlosti komponenty SubBytes. Snížení počtu slices také pomohlo vypnutí řazení registrů (Register Ordering)[2].

Implementace dešifrovacího modulu, který vznikl úpravami modulu šifrovacího, opět překročila množství slices a tak si vyžádala další optimalizace v podobě zrušení některých signálů. Důvodem byl výše popsaný nárůst složitosti komponenty InvMixColumns oproti její neinverzní variantě, který ovšem částečně vyvážilo odstranění 2-bitového a 32-bitového multiplexeru, které v šifrovacím modulu přepínaly vstup a ovládání komponenty SubBytes. Tu používaly jednotky KeyExpansion a Controller. V dešifrovacím modulu používá každá svoji substituční jednotku, proto tyto multiplexery nejsou nutné.

Některé optimalizace dokonce zároveň zvýšily rychlost, např. zrušení signálu, který spouštěl expanzi klíče až po načtení bloku dat. Také je v obou modulech použita jen ta polovina řadiče RS-232 (Tx/Rx), která je potřeba. Další z variant, jak získat více prostředků, byla změna atributu syntézy *OPT_MODE*, který určuje způsob optimalizace syntetizovaného obvodu. Při syntéze pomocí QDevKitu je používána hodnota *SPEED*, která se snaží obvod optimalizovat tak, aby redukovala počet úrovní logických hradel a tím dosáhla co nejvyšší maximální frekvence, na které může obvod pracovat. Ovšem po syntéze s hodnotou *AREA*, která se snaží o co nejmenší počet hradel, se během časové simulace objevily problémy s časováním, konkrétně *HOLD Violation*[3] [18]

[2] Register Ordering je optimalizační technika, která seskupuje klopné obvody, reprezentující bity stejného registru

[3] Hold Time je časový úsek, po který musí byt vstup obvodu stabilní po aktivní hraně hodinového signálu.

Kapitola 4

Zhodnocení

Ovládací program v MCU zabírá 20,1 kB z 48kB dostupné FLASH paměti. Zašifrování jednoho 128-bitového bloku od jeho zapsání do stavu po ukončení šifrování (bez expanze klíče) trvá 750 hodinových taktů. V tabulce 4.1 je uvedena maximální pracovní frekvence a propustnost implementace. Vztahuje se k šifrovacímu i dešifrovacímu modulu, jelikož jsou prakticky shodné. *FITkit* je fyzická implementace na FITkitu, jejíž maximální frekvence je omezena generátorem hodinového signálu DCM v FPGA. *Kompletní (TA)* je maximální frekvence kompletní implementace (včetně řadiče klávesnice, LCD displeje, RS-232, přerušení a dekodérů sběrnice SPI) podle nástroje Timing Analyzer ze sady ISE WebPack. *Pouze AES (TA)* obsahuje jen šifrovací resp. dešifrovací komponentu a k ní náležící SPI dekodér.

	Frekvence (MHz)	Propustnost (Mbit/s)
FITkit	50	8,53
Kompletní (TA)	88	15,02
Pouze AES (TA)	107	18,26

Tabulka 4.1: Maximální frekvence obvodu a propustnost

Tabulky 4.2 a 4.3 znázorňují využití prostředků FPGA pole Xilinx XC3S50 na FITkitu. Oba obvody (šifrovací i dešifrovací) jsou z hlediska dostupných slices naplno využity a to více kombinační logikou (4-vstupové LUTs) než klopnými obvody. Šifrovací obvod využívá 2 paměti BlockRAM pro KeySchedule a S-Box, dešifrovací využívá ještě jednu navíc pro inverzní S-Box.

Jednotky	Využito	Celkový počet	Procentní využití
Slices	766	768	99%
Slice Flip Flops	740	1536	48%
4-input LUTs	1250	1536	81%
BlockRAMs	2	4	50%

Tabulka 4.2: Přehled využití zdrojů FPGA pole FITkitu - šifrovací modul

Jednotky	Využito	Celkový počet	Procentní využití
Slices	766	768	99%
Slice Flip Flops	806	1536	52%
4-input LUTs	1327	1536	86%
BlockRAMs	3	4	75%

Tabulka 4.3: Přehled využití zdrojů FPGA pole FITkitu - dešifrovací modul

4.1 Bezpečnost implementace

Samotný algoritmus je i ve verzi se 128b klíčem v současné době dostatečně bezpečný. Ovšem bezpečnost ovlivňuje i samotná implementace. Použitím módu CBC oproti více zranitelnému ECB, možností změny klíče pro každou relaci a unikátním inicializačním vektorem pro jednotlivé zprávy se určitě zvýšila.

Kapitola 5

Závěr

Dle zadání jsem implementoval komunikaci formou zasílání textových zpráv mezi dvěma FITkity přes rozhranní RS-232. Zprávy jsou zadávány a zobrazovány pomocí klávesnice a LCD displeje na FITkitu a šifrovány algoritmem AES-128 (Rijndael), pracujícím v módu CBC. Šifrování je implementováno pomocí jazyka VHDL v FPGA poli FITkitu, kde se také nachází řadiče podpůrných komponent. Vše je řízeno ovládacím programem v MCU.

Při řešení jsem se setkal s mnoha problémy, z nichž nejzávažnější byly omezené prostředky FPGA pole. Proto jsem byl nucen, oproti původnímu plánu, implementovat komunikaci pouze jednosměrně.

Výzvou pro budoucí vývoj by mohla být právě snaha o implementaci obousměrné šifrované komunikace. Toho by se dalo docílit dalšími optimalizacemi šifrovacího obvodu, například zúžením datové šířky na jeden byte, což ovšem přinese oběť v podobě prudkého poklesu rychlosti. Projekt by se také dal přepracovat do takové podoby, kdy by FITkity šifrovaly komunikaci mezi dvěma počítači prostřednictvím terminálu. Ta by se nemusela omezit na rozhranní RS-232, protože mnohem praktičtější rozšiřující ethernetový a bezdrátový modul pro FITkit verze 2 je již navržen. Další možností by mohla být podobná implementace, avšak s výběrem jiného algoritmu, například TwoFISH se jeví jako perspektivní.

Pro komunikaci více než dvou FITkitů se zdají být nejlepší právě výše zmíněné rozšiřující moduly. Tak by mohly být FITkity propojené skrze switch po ethernetové síti, bezdrátově prostřednictvím AP nebo v Ad-Hoc módu.

Literatura

[1] Federal Information Processing Standards Publication 197 - Announcing the Advanced Encryption Standard (AES). 2001, [cit. 2010-05-01].
URL http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[2] Burwick, C.: MARS - a candidate cipher for AES. 1999, [cit. 2010-05-01].
URL http://www.research.ibm.com/security/mars.pdf

[3] Cooper, O.: An introduction to modern cryptography.
URL http://www.cs.bris.ac.uk/~cooper/Cryptography/crypto.html

[4] Hradil, D.: *Hardwarová akcelerace šifrování*. Diplomová práce, FIT VUT v Brně, 2007.

[5] James Nechvatal: Report on the Development of the AES. 2000.

[6] Kouřil, M.: *Šifrovaná komunikace mezi FITkitem a PC*. Diplomová práce, FIT VUT v Brně, 2007.

[7] Mao, W.: *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, 2003, ISBN 0-13-066943-1.

[8] Novotňák, J.: *Hašovací algoritmy v FPGA*. Diplomová práce, FIT VUT v Brně, 2008.

[9] Piper, F.; Murphy, S.: *Kryptografie: Průvodce pro každého*. Dokořán, 2006, ISBN 80-7363-074-5.

[10] Rivest, R. L.; Robshaw, M.; Sidney, R.; aj.: The RC6 Block Cipher. 1998, [cit. 2010-05-01].
URL ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf

[11] Robshaw, M.: RC6 and the AES. 2006, [cit. 2010-05-01].
URL ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6+aes.pdf

[12] Schneier, B.: WWW stránky. 2010, [cit. 2010-05-01].
URL http://www.schneier.com

[13] Vávra, J.: *Šifrování telefonních hovorů*. Diplomová práce, FIT VUT v Brně, 2008.

[14] Weaver, N.; Wawrzynek, J.: A Comparison of the AES Candidates Amenability to FPGA Implementation. 2000.

[15] WWW stránky: Oficialní stránky šifry Serpent. 2006, [cit. 2010-05-01].
URL http://www.cl.cam.ac.uk/~rja14/serpent.html

[16] WWW stránky: Projekt Copacabana. 2008, [cit. 2010-05-01].
URL http://www.copacobana.org

[17] WWW stránky: Platforma FITkit. 2010, [cit. 2010-05-01].
URL http://merlin.fit.vutbr.cz/FITkit

[18] WWW stránky: Webové stránky firmy Xilinx. 2010, [cit. 2010-05-01].
URL http://www.xilinx.com

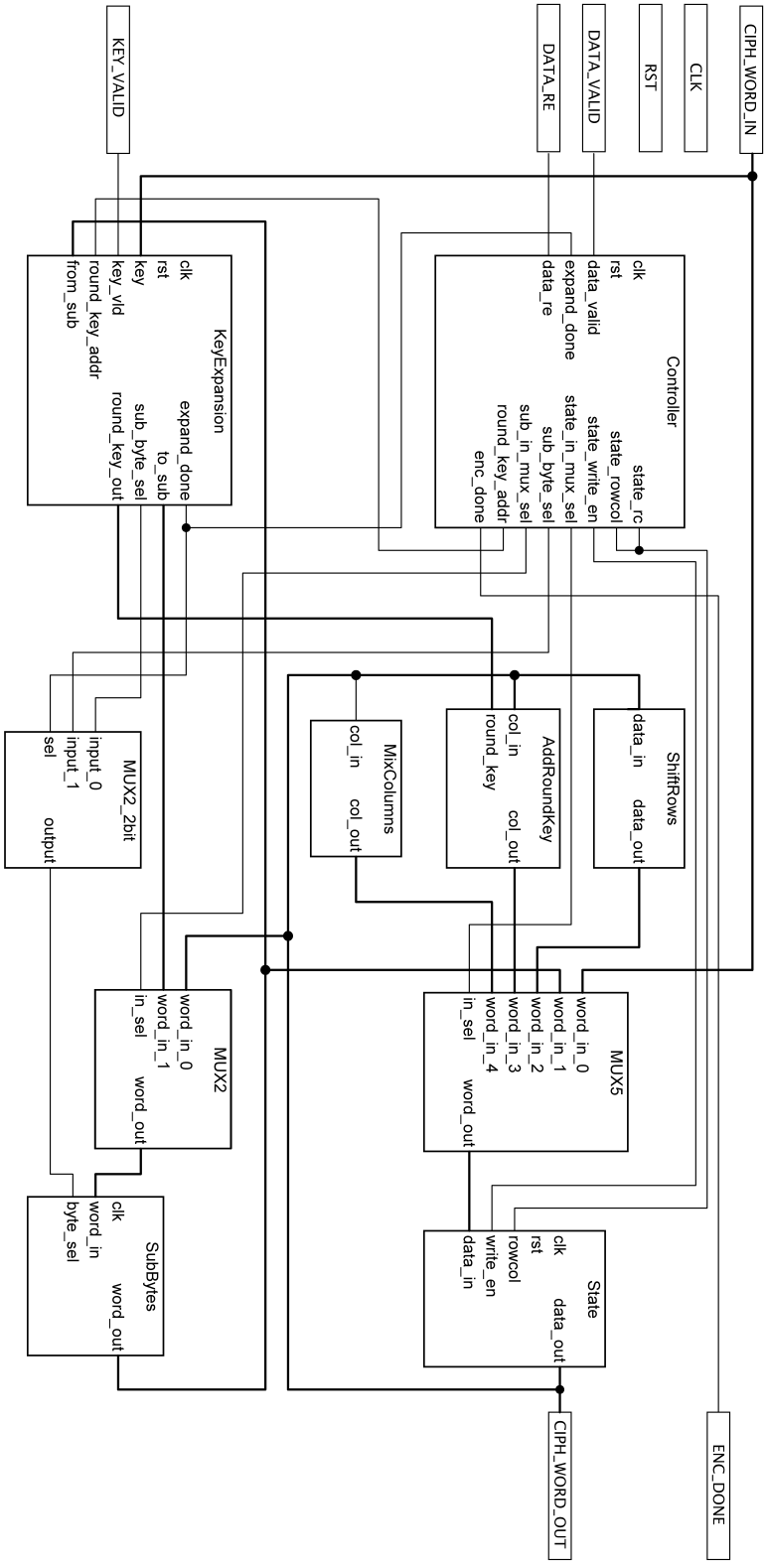
Příloha A

Obsah přiloženého CD

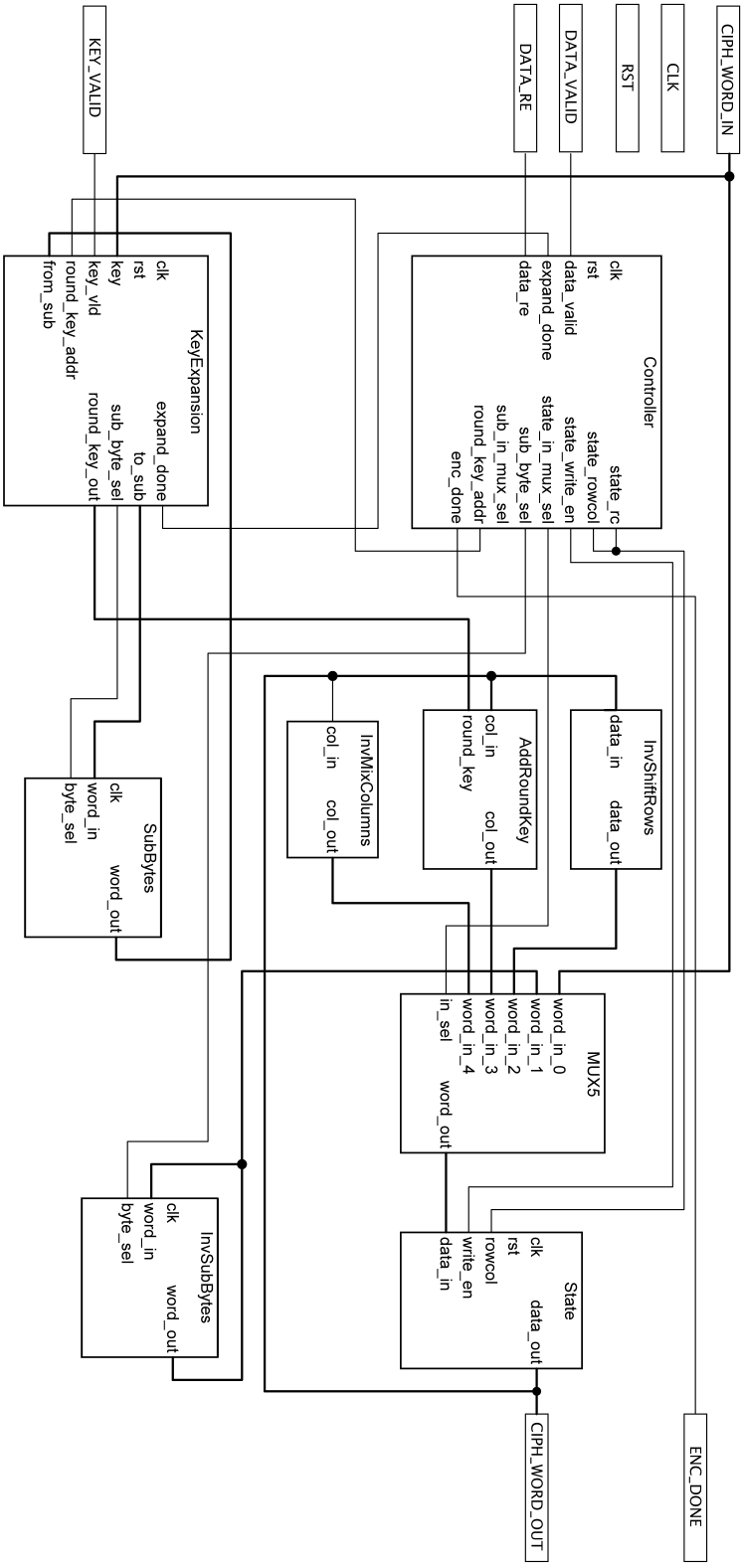
- Zdrojové kódy pro FPGA(VHDL) a MCU(C),
- testbenche pro simulaci šifrovacího a dešifrovacího obvodu,
- tato zpráva ve formátu pdf včetně zdrojového kódu.

Příloha B

Schéma šifrovacího a dešifrovacího obvodu



Obrázek B.1: Schema komponenty Encryptor (tučně zvýrazněné jsou 32b datové sběrnice)



Obrázek B.2: Schema komponenty Decryptor (tučně zvýrazněné jsou 32b datové sběrnice)

Příloha C

S-boxy

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

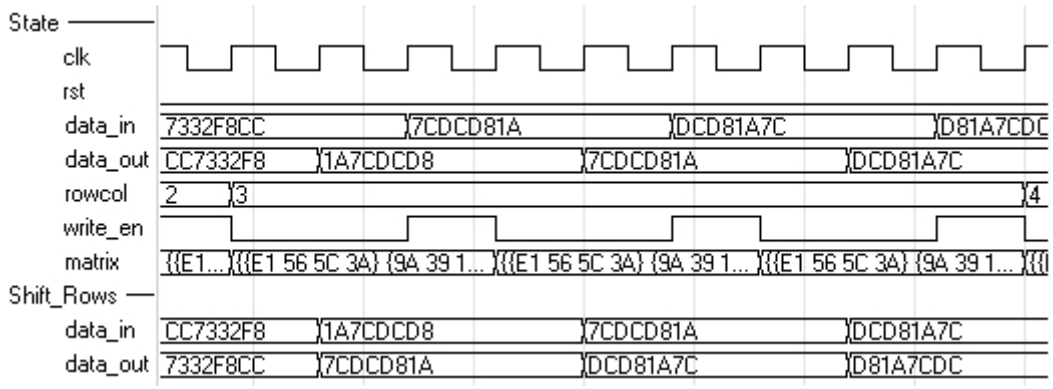
Tabulka C.1: S-Box

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

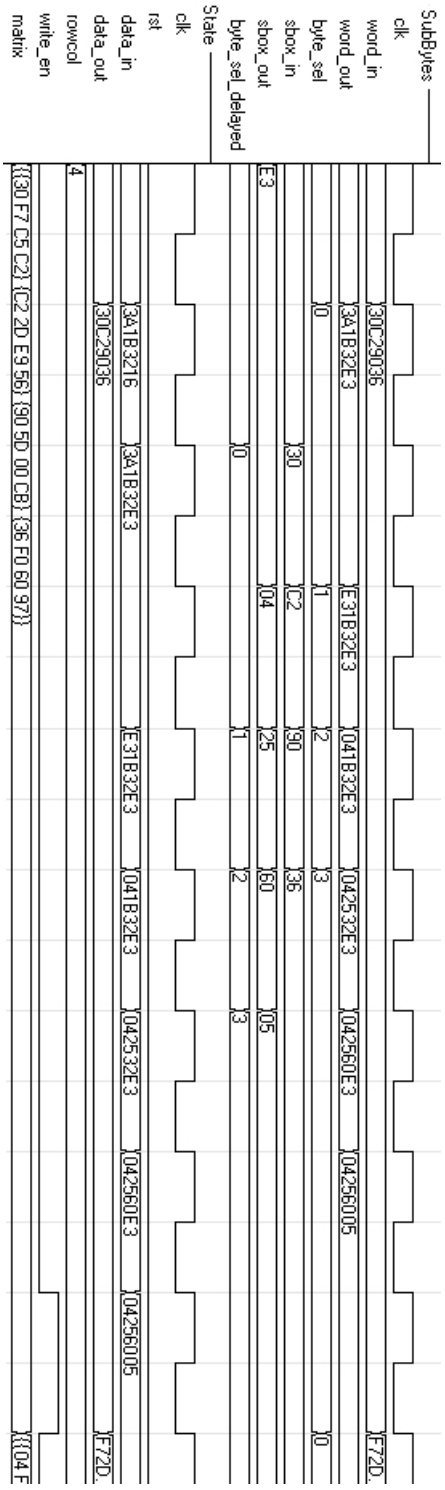
Tabulka C.2: Inverzní S-Box

Příloha D

Simulace - časové průběhy signálů



Obrázek D.1: Operace ShiftRows - posun posledního řádku o 3 byty doleva



Obrázek D.2: Operace SubBytes - substituce slova po bytech