VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

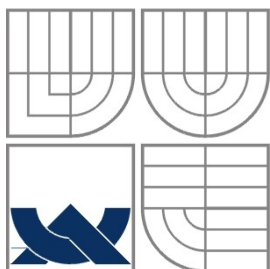
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE ŠIFROVACÍCH ALGORITMŮ POMOCÍ FPGA

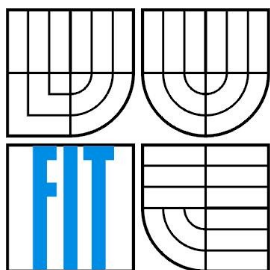
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MIROSLAV GAJDOŠ
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE ŠIFROVACÍCH ALGORITMŮ POMOCÍ FPGA

ACCELERATION OF ENCRYPTION ALGORITHM USING FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MIROSLAV GAJDOŠ
AUTHOR

VEDOUCÍ PRÁCE — ING. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá možností akcelerace šifrovacích algoritmů pomocí rekonfigurovatelných obvodů FPGA a zkoumáním rozdílu rychlosti implementace oproti implementaci softwarové. Práce popisuje základy šifrování a akcelerace algoritmů na FPGA. Dále se zabývá procesem návrhu, implementace, simulace a syntézy výsledné implementace. Provádí rozbor dosaženého řešení. Cílem projektu bylo vytvořit funkční řešení akcelerovaného algoritmu, tím umožnit jeho další použití v reálném provozu a dále vytvoření česky psaného materiálu o této problematice.

Abstract

This work deals with the possibility of acceleration algorithm using reconfigurable FPGA circuits and speed of implementation by examining the difference compared to software implementation. The work describes the basics of encryption and acceleration algorithms on the FPGA. It then addresses the process of design, implementation, simulation and synthesis of the resulting implementation. It made analysis of the achieved solution. The aim of the project was to create a functional solution of accelerated algorithm, thus enabling its use in the real application and, finally, establishment of czech written material on this issue.

Klíčová slova

FPGA, šifrovací algoritmus, akcelerace, VHDL, DES, FITkit, Spartan 3, Virtex 5.

Keywords

FPGA, encryption algorithm, acceleration, VHDL, DES, FITkit, Spartan 3, Virtex 5.

Citace

Miroslav Gajdoš: Akcelerace šifrovacích algoritmů pomocí FPGA, bakalářská práce, Brno, FIT VUT v Brně, 2009

AKCELERACE ŠIFROVACÍCH ALGORITMŮ POMOCÍ FPGA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Václava Šimka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Miroslav Gajdoš
26. 5. 2009

Poděkování

Chtěl bych poděkovat svému vedoucímu práce, Ing. Šimkovi, za jeho vedení, podporu, odbornou pomoc a rady. Bez něj by tato práce nemohla vzniknout.

© Miroslav Gajdoš, 2009
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

Obsah....1
1 Úvod....2
2 Problematika šifrování dat....3
2.1 Úvod do šifrování....3
2.2 Základní pojmy....3
2.3 Šifrovací algoritmy....3
3 Algoritmus DES....6
3.1 Popis algoritmu....6
3.2 Použití DESu v PGP....9
3.3 Kontroverze....10
3.4 Současnost....10
3.5 Implementace na FPGA....10
4 Rekonfigurovatelný hardware....11
4.1 FPGA....11
4.2 Využití....11
4.3 Návrh a programování....12
4.4 FITkit....13
4.5 Karta PICO E-16....14
5 Akcelerace DES na FPGA....15
5.1 Nástroje....15
5.2 Hardware....15
5.3 Návrh....15
5.4 Implementace....20
5.5 Simulace....26
5.6 Syntéza....28
6 Závěr....33
6.1 Analýza dosažených výsledků....33
6.2 Pokračování projektu, navrhovaná vylepšení....33
Literatura....35
Seznam příloh....36

1 Úvod

Tato práce si klade za cíl prozkoumat možnosti implementace šifrovacích algoritmů akcelerovaně pomocí rekonfigurovatelných obvodů, FPGA a porovnání se stávající softwarovou implementací.

Motivací byl nedostatek materiálů na dané téma v českém jazyce a dále zájem o funkci šifrovacích algoritmů a snaha zjistit, jak si povede algoritmus akcelerovaný pomocí FPGA, v porovnání se stejným algoritmem, který již je softwarově implementován.

Realizace algoritmu bude založena na rozboru daného tématu, výběru vhodného algoritmu pro implementaci, výběru části pro akceleraci, implementace algoritmu včetně simulace pomocí jazyku VHDL a konečně naprogramování FPGA a test implementace na hardwaru.

Práce je rozdělena na několik tematických sekcí.

Nejprve uvedu čtenáře do problematiky kryptografie, neboli šifrování. V této části jsou uvedeny jednak základní pojmy z této problematiky, dále pak rozdělení jednotlivých algoritmů z různých hledisek a několik příkladů.

Další kapitola se zabývá rekonfigurovatelným hardwarem, jeho využitím, procesu návrhu a implementace. Dále pak popis 2 zástupců hardwaru, obsahujících FPGA čip.

Kapitola „Algoritmus DES“ se zabývá algoritmem, který jsem si k implementaci na FPGA zvolil. Čtenáře zde seznámím s tímto algoritmem, uvedu příklad jeho užití, jeho rozbor, stav v současnosti a hlediska jeho nasazení pomocí FPGA.

V následující kapitole se potom budu zabývat hlavní náplní této práce – návrhem, implementací v jazyce VHDL, simulací a syntézou vybraného algoritmu. Budu zde diskutovat problémy během prací, stejně tak výsledky simulací a teoretické srovnání hardwarové a softwarové implementace algoritmu.

V závěru pak uvedu vlastní přínos k tématu a nastíním další možnosti pokračování v projektu.

2 Problematika šifrování dat

2.1 Úvod do šifrování

Informace znamenají moc. Proto se už od zrodu mluveného slova člověk zabýval možností utajení důvěrných informací. Jejich znalost nepřítelem by způsobila jen problémy. V dnešní době informačních technologií jsou informace ještě cennější komodita a utajení klíčových dat je otázkou úspěchu v obchodním světě.

Šifrovací algoritmy se časem vyvýjely, od nejjednodušších substitučních, aditivních a transpozičních algoritmů až po dnešní moderní, velmi složité algoritmy nebo speciality typu kvantové kryptografie.

Se složitostí šifer však rostla i výpočetní náročnost těchto algoritmů. To nevyhnutelně vedlo k zapojení výpočetních systémů (počítačů) do procesu šifrování a dešifrování. Způsob jejich implementace pak určuje rychlost zpracování dat a vytížení zdrojů systémů.

2.2 Základní pojmy

Šifrování nebo kryptografie je způsob utajení zprávy převodem do takové podoby, že je čitelná pouze se speciální znalostí. K přečtení takové šifrované zprávy je potřeba tajný klíč. Šifra je pak algoritmus, který pomocí klíče provádí převod prostého textu na jeho nečitelnou podobu.

Symetrická šifra používá pro dešifrování i šifrování stejný klíč. Asymetrická šifra používá 2 různé klíče. Hashovací funkce je v podstatě jednosměrná šifra; z prostého textu vytvoří krátký řetězec, který s velkou pravděpodobností identifikuje původní text.[1]

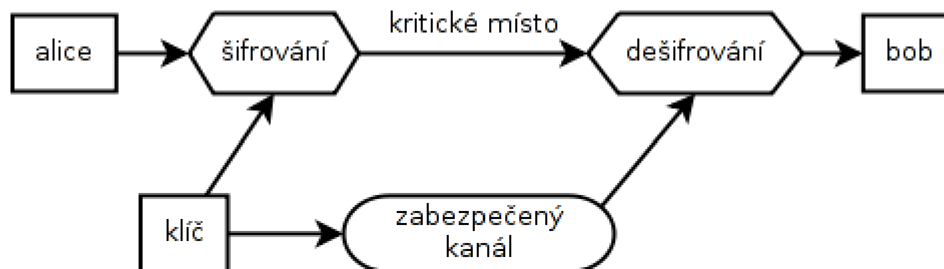
2.3 Šifrovací algoritmy

Šifrovací algoritmy jsou algoritmy, které data buď šifrují, či dešifrují. Moderní šifrovací algoritmy jsou většinou již softwarově implementovány. Lze je v prvé řadě rozdělit na symetrické a asymetrické šifry. Dále je můžeme klasifikovat na blokové a proudové šifry. Blokové šifry fungují tak, že zpracovávají data po blocích pevné velikosti. Naproti tomu proudové šifry zpracovávají data sekvenčně, po jednotlivých znacích; jsou většinou jednodušší, ale míň bezpečné.

2.3.1 Symetrické šifry

Symetrické (též konvenční) šifry používají k šifrování i dešifrování stejný klíč. Obě strany (původce i příjemce zprávy) musí klíč vlastnit. Musí ho tedy vlastnit předem, nebo se domluvit na jeho předání zabezpečeným kanálem. Symetrické šifry jsou obecně jednodušší na implementaci, na šifrování i dešifrování se používá většinou týž algoritmus. Jsou i méně náročné na výpočetní výkon. [2]

Symetrické šifrování lze jednoduše znázornit obrázkem 2.1:



Obrázek 2.1: Princip symetrického šifrování [1]

Alice potřebuje komunikovat s Bobem po nezabezpačeném kanálu. Oscar chce však zprávy zachytávat a číst. Alice a Bob tedy musí před přenosem zprávy zašifrovat. Použijí k tomu symetrickou šifru, která používá jediný klíč. Ten si musí Alice s Bobem předem rozdistribuovat prostřednictvím nějakého zabezpečeného kanálu (např. osobně).

Mezi tyto šifry patří např. DES, AES, IDEA, FISH.

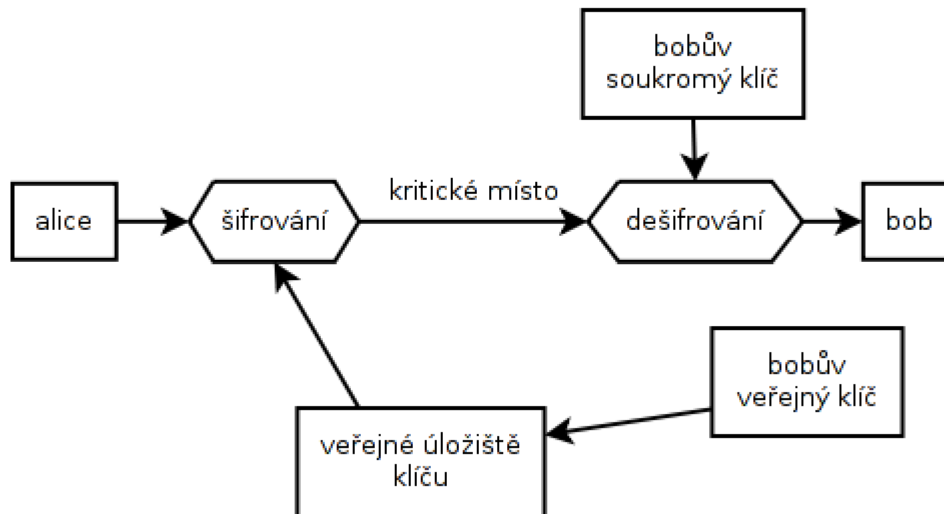
2.3.2 Asymetrické šifry

Asymetrické šifry používají oproti symetrickým šifrám 2 klíče – jeden pro zašifrování a druhý pro dešifrování zprávy. Obvykle se klíč určený k zašifrování zprávy vystaví na veřejném místě (tzv. veřejný klíč). Lze ho pak použít k zašifrování zprávy osobě, která klíč vlastní. Příjemce pak zprávu dešifruje pomocí svého tajného klíče. Odpadá tak problém distribuce klíče zabezpečeným kanálem. [2]

Tento druh šifrování lze navíc použít k ověření autenticity (nebo i integrity) zprávy - podepsáním privátním klíčem původce a jeho ověřením u příjemce původcovým veřejným klíčem.

Tyto algoritmy jsou většinou poměrně složitější k implementaci i náročnější na výpočetní zdroje. Proto se obvykle asymetrická šifra používá pouze při zahájení komunikace subjektů, kdy si subjekty vymění klíč určený pro symetrickou šifru a poté již komunikují kanálem zabezpečeným pouze symetrickou šifrou.

Asymetrické šifrování lze jednoduše znázornit obrázkem 2.2:



Obrázek 2.2: Princip asymetrického šifrování [1]

Alice chce komunikovat s Bobem po nezabezpečeném kanálu. Oscar chce opět zprávy číst, Alice s Bobem tak musí použít šifrování. Tentokrát však Alice zašifruje data asymetrickou šifrou s veřejným klíčem Boba, který získá např. z veřejného úložiště klíčů. Jediný Bob pak má znalost, která mu dovolí zprávu dešifrovat (vlastní soukromý klíč). Odpadá zde problém distribuce tajného klíče. Zároveň se však Alice musí ubezpečit, zda má opravdu autentický klíč (hrozba man-in-the-middle útoku).

Pro příklad použití lze uvést program GnuPG, což je GNU implementace Pretty Good Privacy.

Mezi tyto šifry patří např. RSA, ElGamel, Diffie-Hellman, DSA.

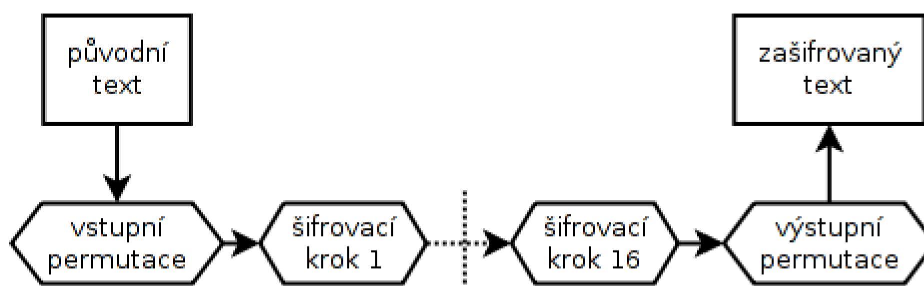
3 Algoritmus DES

DES[1] je symetrická šifra, která byla v roce 1976 vybrána jako oficiální národní standard Spojených států a která se poté masově rozšířila. Je založena na algoritmu se symetrickým, 56-bitovým klíčem.

V této kapitole jsem čerpal z [3].

3.1 Popis algoritmu

DES je archetypální bloková šifra – algoritmus provádí šifrování po blocích stejné, konstantní délky. V případě DES je blok 64 bitů dlouhý. Konkrétní podoba transformace jednotlivých bloků odpovídá klíči – k dešifrování je pak zapotřebí stejný klíč. Klíč je 64 bitů dlouhý, ale pouze 56 bitů je efektivních a ovlivňuje sílu šifry. Zbytek klíče (8 bitů) slouží pouze ke kontrole parity.



Obrázek 3.1: Průběh algoritmu DES

Průběh algoritmu (viz také obr. 3.1) vypadá následovně: blok dat nejprve projde *vstupní permutací*, která pouze změní pořadí bitů. Blok dat poté prochází 16x *šifrovacím krokem* (ten bude popsán dále). Na konci dojde k *výstupní permutaci* (která je inverzní k permutaci vstupní). Výstupem algoritmu je zašifrovaný, nebo dešifrovaný text (dle zvolené operace, průběh se liší pouze pořadím aplikace podklíčů při jednotlivých krocích; viz dále).

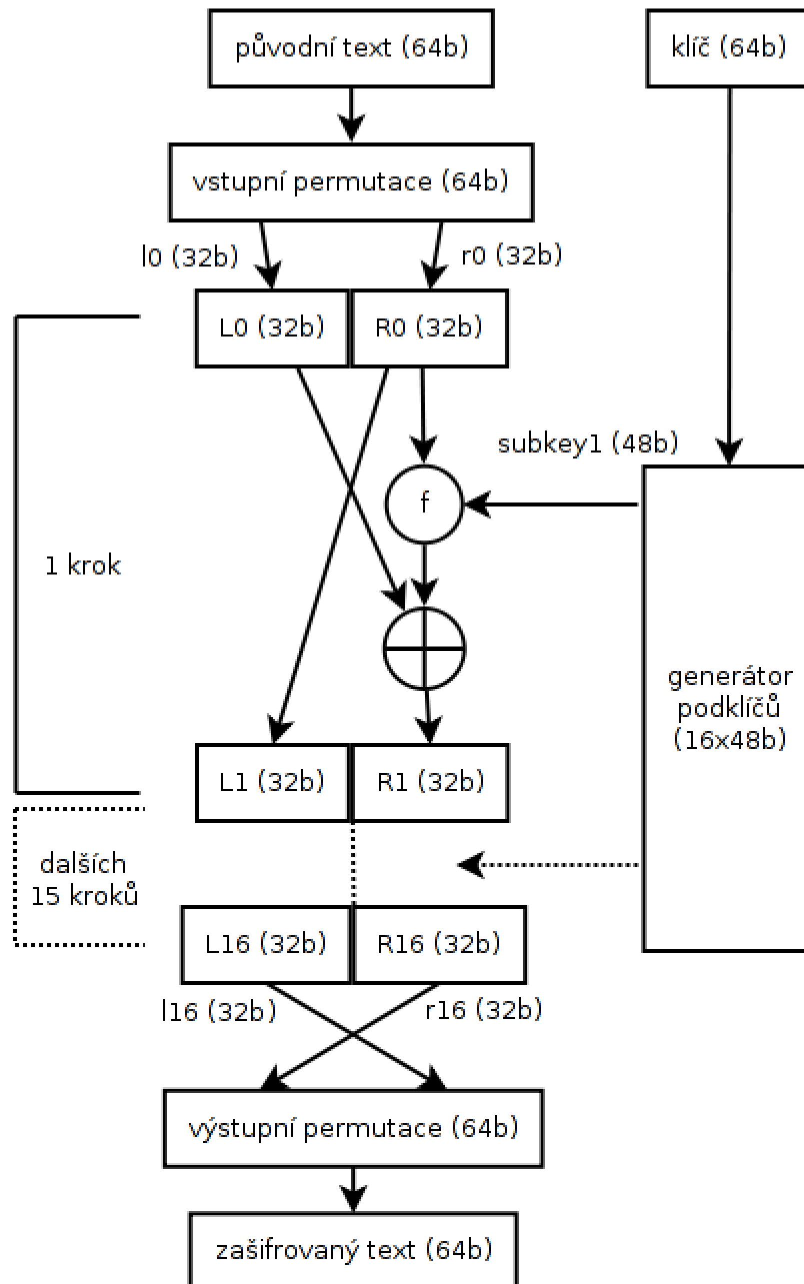
Šifrovací krok vypadá následovně: vstup je rozdělen na 2 poloviny (každá o velikosti 32 bitů) a prochází funkcí F. Každá polovina je ale zpracovávána samostatně; tomuto křížení se říká Feistelovo schéma. Toto uspořádání zajišťuje, že šifrování a dešifrování jsou velmi podobné procesy. Jediný rozdíl je, že při dešifrování jsou podklíče aplikovány v opačném pořadí.

Jako poslední krok se provede výstupní permutace, která je reverzibilní vůči vstupní permutaci.

1 z angl. Data Encryption Standard

Pomocí obměňování permutací a S-boxů (taktika nazývaná *„confusion and diffusion“*) lze zvýšit šanci, že šifra zůstane neprolomena (nebo se prolomení oddálí).

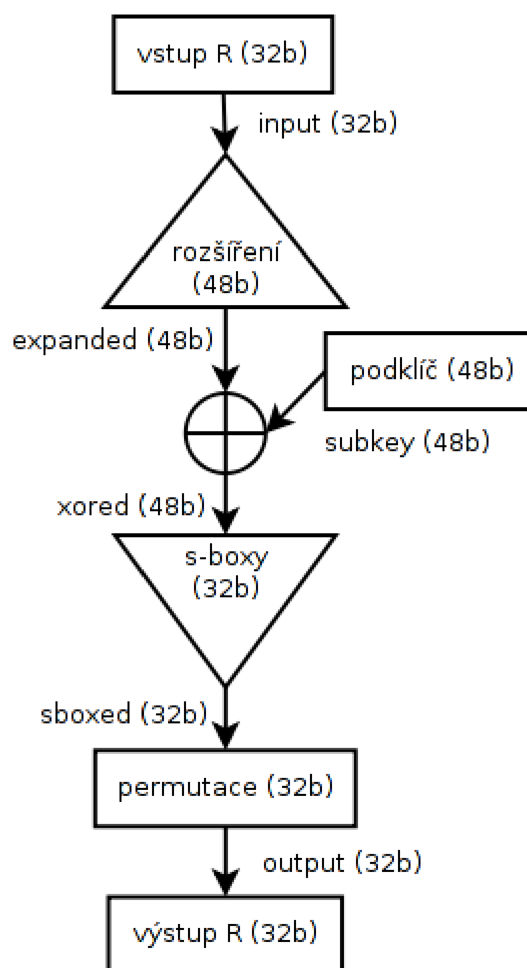
Obrázek 3.2 znázorňuje podrobnější schéma fungování algoritmu, konkrétně lze rozpoznat Feistelovo schéma, které se tu uplatňuje:



Obrázek 3.2: DES - Feistelovo schéma [9]

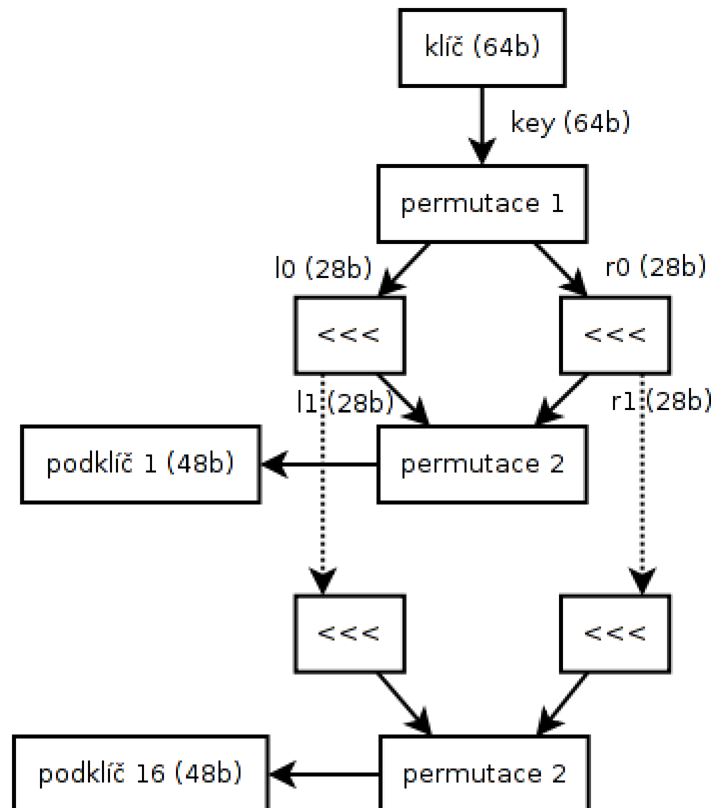
Nyní se podíváme na *funkci f*. Blok dat nejprve projde *expanzní permutací* (některé bity jsou použity vícekrát – součást taktiky *confusion and diffusion* – zde dochází k rozptýlení vlivu jednotlivých bitů). Poté dojde k exkluzivní disjunkci bloku s příslušným podklíčem. Následuje průchod bloku dat *s-boxy*. Ty v podstatě provádí substituci. Mapují 6 vstupních bitů na 4 výstupní. Lze je implementovat jako tzv. lookup-table, kde první a poslední bit vstupu identifikuje řádek a prostřední 4 bity sloupec tabulky. Cílová buňka pak představuje výstup pro daný vstup. Zde se opět uplatňuje taktika *confusion and diffusion* – snaha zastřít vztah vstupního a výstupního textu. Správná volba s-boxů velkou měrou přispívá k odolnosti šifry proti prolomení.

Po průchodu s-boxy už zbývá jen výstupní permutace.



Obrázek 3.3: DES - schéma f-funkce [2]

Z algoritmu už zbývá popsat pouze proces vytváření podklíčů pro jednotlivé kroky šifrování z původního klíče. Původní klíč nejprve projde *permutací 1*, která odstraní 8 paritních bitů. Zbylých 56 bitů je rozděleno na 2 poloviny po 28 bitech a mezi jednotlivými kroky dle daného předpisu postupně rotují (vlevo, buď o 1 nebo 2 místa). V každém kroku pak obě poloviny projdou *permutací 2*, která opět odstraní 8 bitů a její výstup už je podklíčem daného kroku.



Obrázek 3.4: DES - generování podklíčů

Právě generování podklíčů určuje to, zda algoritmus šifruje, nebo dešifruje. Jediný rozdíl obou možností je totiž pořadí aplikace podklíčů v jednotlivých krocích metody.

3.2 Použití DESu v PGP

Algoritmus DES je v současné době užíván jako součást systému PGP[2], což je systém umožňující šifrování a podepisování dat. PGP se dá použít např. v oblasti komunikací (elektronická pošta, instant messaging), kde může poskytnout end-to-end šifrování a autentifikaci. V principu se však jedná o asymetrickou metodu, to znamená, že používá 2 klíče (veřejný a soukromý). Asymetrické šifrování používá ale jen z důvodu vytvoření zabezpečeného kanálu pro výměnu sdíleného klíče pro

2 z angl. Pretty Good Privacy

symetrickou metodu šifrování (např. právě DES). Symetrické šifrování je totiž méně výpočetně náročné (a také rychlejší).

3.3 Kontroverze

Ve svých počátcích byl algoritmus utajován a vzhledem k malé velikosti klíče panovalo podezření, že obsahuje zadní dvířka. Tento back-door měl dovolit agentuře NSA (která spolupracovala na vývoji tohoto algoritmu) jeho prolomení v dostatečně krátkém čase. Tyto obavy se však nikdy nepotvrdily, ale donutily výzkumné ústavy univerzit, aby tento algoritmus důkladně prozkoumaly. Přispěly tak k pochopení moderních blokových šifer a jejich kryptoanalýze.

3.4 Současnost

DES je nyní považován jako nedostatečně bezpečný pro použití v korporátní sféře, a to především kvůli příliš malému klíči. V roce 1999 se podařilo veřejně demonstrovat prolomení DES klíče během 22 hodin. Algoritmus je však stále použitelný díky variantě Triple DES, kdy je DES použit 3-krát (obvykle s klíčem o velikosti 168 bitů, tzn. složený ze tří standardních klíčů).

Původní DES se stále používá např. v PGP / GnuPG. Obecně však byla šifra překonána šifrou AES[3].

3.5 Implementace na FPGA

Vzhledem k relativní jednoduchosti algoritmu by měla být implementace na platformě FPGA poměrně jednoduchá. Možných řešení je však více. Lze se zabývat i optimalizací, a to buď prostorovou, nebo časovou. DES totiž používá kroky, které se mnohonásobně opakují (např. funkce F – opakuje se 16x). Dá se tedy implementovat paralelně (16 výpočetních jednotek) a zkrátit tak čas, kdy se čeká na výsledek funkcí F, 16x.

Jedním z cílů této práce bude i porovnání softwarové a hardwarové implementace. Vzhledem k možnostem paralelního zpracování a taky faktu, že DES obsahuje některé prvky, které mají znevýhodnit softwarovou implementaci (jako například vstupní a výstupní permutace), dá se předpokládat, že FPGA bude ve výhodě.

3 z angl. Advanced Encryption Standard

4 Rekonfigurovatelný hardware

4.1 FPGA

FPGA[4] je polovodičový čip, jehož činnost lze naprogramovat (a později opět přeprogramovat) po výrobě.

FPGA může vykonávat jakoukoliv činnost, takže může nahradit aplikačně-specifický čip. Obsahuje programovatelné komponenty, tzv. logické bloky, které jsou propojeny programovatelnými spoji. Bloky mohou být nastaveny k vykonávání činnosti logických hradel, případně paměťových elementů. [10]

Navíc lze popis jeho činnosti upravovat – obdoba nové verze softwaru. To je velmi výhodné z hlediska vývoje a údržby aplikace – snižuje čas a prostředky potřebné k vývoji a údržbě.

Určitou nevýhodou oproti aplikačně-specifickým čipům je nižší rychlost, vyšší příkon a menší prostorová efektivita. [12]

4.2 Využití

FPGA nalézá uplatnění v oblasti těch druhů algoritmů, kde se dá využít paralelismu. Na jednom FPGA čipu je totiž možné implementovat velké množství výpočetních jednotek. Jedná se např. o code-breaking, útoky typu brute-force, šifrovací algoritmy nebo oblast vysoce výkonných výpočetních systémů.

Vysoká úroveň paralelismu dovoluje dosáhnout velkého průtoku dat i při relativně nízké taktovací frekvenci čipů (běžně méně než 500MHz). Například dnešní generace FPGA umožňuje implementaci stovek FPU na jediném čipu, kde jsou jednotlivá FPU schopna získávat výsledek každý hodinový cyklus. Dnes se již běžně setkáváme i s použitím FPGA v aplikacích typu System On Chip.

Jedním z možných využití FPGA je akcelerace softwarových šifrovacích algoritmů, za všechny lze uvést konkrétně algoritmus DES. Tento algoritmus byl přímo navržen pro hardwarovou implementaci. Obsahuje navíc prvky, které by měli softwarovou implementaci zpomalit, takže k použití FPGA přímo vybízí. [12]

Výhody užití FPGA místo ASIC čipů v určitých aplikacích jsou tedy zřejmé.

Z dalších použití FPGA lze uvést vestavěné systémy. Ty se v dnešní době uplatňují v běžném životě a jejichž význam ještě výrazně poroste. Jedná se o všechna zařízení, která v sobě mají nějakým způsobem vestavěn počítač (mobilní telefon, MP3 přehrávač, televizní přijímač atd.). Typické

4 z angl. Field-Programmable Gate Array

vestavěné systémy se skládají z procesorů, specializovaného hardwaru (např. MP3 kodér/dekodér) a aplikačního software.[4] FPGA lze využít k návrhu těchto zařizení, případně při použití v produkčních sériích lze i upravovat funkci zařízení již v postprodukční fázi.

4.3 Návrh a programování

K definování chování FPGA musí programátor vytvořit schematický design, nebo popis jazykem HDL[5]. Oba přístupy mají jisté výhody i nevýhody, ale z hlediska přehlednosti u projektů vyšší složitosti je popis jazykem HDL zřejmě výhodnější.

Po vytvoření takového popisu se použije nástroj pro syntézu. Ten z textového popisu vytvoří netlist využívající obecné logické bloky. Poté se použije většinou proprietární nástroj výrobce, který z obecného netlistu vytvoří netlist specifický pro konkrétní hardware. Takovýto netlist lze již nahrát do FPGA, čímž se de facto naprogramuje.

Během vývoje je dále vhodné využívat simulátoru. Simulátor umožňuje sledovat průběh algoritmu bez potřeby syntézy, urychluje tak vývoj a usnadňuje kontrolu funkce, odhalování a opravu chyb. Simulací lze i zjistit, zda syntéza proběhla úspěšne.

4.3.1 VHDL

VHDL[6] je programovací jazyk pro popis hardwaru. Používá se pro návrh a simulaci obvodových realizací na FPGA (mimo jiné). Jedná se o typovaný programovací jazyk. Má prostředky pro popis paralelismu, konektivity a explicitní vyjádření času.

Popis tímto jazykem se dělí na 2 části: popis entit a popis architektury.

Popis entit definuje rozhraní (tzn. pouze vstupy a výstupy, z toho hlediska jsou entity černou škříňkou). Rozhraní je vlastně výčet portů, ty mohou být několika typů: *in* (vstup), *out* (výstup), *buffer* (výstup se zpětnou vazbou), *inout* (vstup-výstup), *linkage* (neznámý směr toku).

Popis architektury určuje chování entit. Je rozdělen na 2 části: deklarační a příkazovou. Pro jednu entitu může existovat více architektur (implementací).

Architekturu lze popsat 3 způsoby.

Strukturální popis je popis na vysoké úrovni. Užívá se hlavně pro simulaci.

Behaviorální popis je vhodný pro syntézu.

Dataflow popis umožňuje detailní časové simulace.

5 z angl. Hardware Description Language

6 z angl. Very-high-speed-integrated-circuit Hardware Description Language

4.4 FITkit

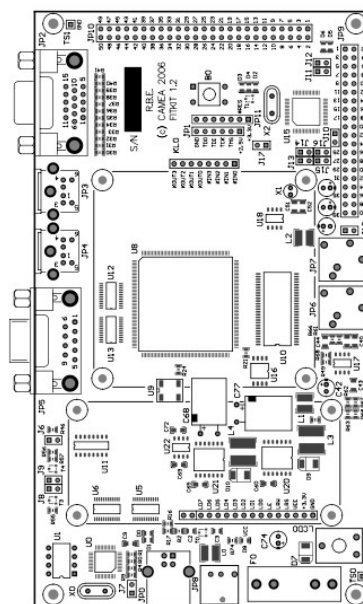
Platforma FITkit je výuková platforma Fakulty informačních technologií VUT. Byla nasazena, aby umožnila studentům vyzkoušet si návrh a realizaci nejen softwarových, ale i hardwarových projektů.

4.4.1 Součásti FITkitu

FITkit obsahuje výkonný mikrokontrolér s nízkým příkonem a řadu periferií. Důležitým aspektem je též využití pokročilého reprogramovatelného hardwaru na bázi hradlových polí FPGA, jenž lze, podobně jako software na počítači, neomezeně modifikovat pro různé účely dle potřeby – uživatel tedy nemusí vytvářet nový hardware pro každou aplikaci znovu. [4]

FITkit obsahuje jmenovitě následující součásti:

- MCU rodiny MSP430 (Texas Instruments)
- FPGA Spartan 3 XC3S50-4PQ208C (Xilinx)
- USB převodník FT2232C
- audio rozhraní
- konektory PS2
- rozhraní VGA
- konektor RS232
- DRAM 8x8Mbit
- klávesnice
- řádkový LCD displej
- rozšiřující konektory



Obrázek 4.1: FITkit [4]

4.4.2 FPGA na FITkitu

FPGA na kitu je programovatelné hradlové pole XC3S50-4PQ208C řady Spartan 3 firmy Xilinx.

- rozhraní: až 124 uživatelských vstupů/výstupů (I/O), podpora až 23 různých I/O standardů
- 192 konfigurovatelných logických bloků (CLBs) uspořádaných do matice o 16 řádcích a 12 sloupcích, 1728 logických buněk, 50000 logických hradel
- paměť RAM 12 kbitů distribuovaných, 72 kbitů v jednom bloku
- 2 jednotky pro správu hodin (DCMs)
- 4 blokové dvouportové paměti BRAM s kapacitou 2kB
- 4 násobičky 18x18 bitů [5]

4.5 Karta PICO E-16

Karta PICO E-16 firmy Pico Computing je koprocesor určený pro zapojení do ExpressCard/34 slotu počítače. Je určena pro hardwarové akcelerování algoritmů. Díky zapojení do ExpressCard slotu lze dosáhnout vysoké datové propustnosti. K akceleraci šifrování je tedy velmi vhodná.

Karta bohužel v době psaní této práce nebyla fyzicky k dispozici, jistě by byla zajímavější alternativou k FITkitu.

4.5.1 FPGA na kartě PICO E-16

FPGA na kartě je programovatelné hradlové pole XC5VLX50 řady Virtex-5 LX50 firmy Xilinx.

- rozhraní: až 560 uživatelských vstupů/výstupů (I/O)
- 7200 konfigurovatelných logických bloků (CLBs) uspořádaných do matice o 120 řádcích a 30 sloupcích, 46 080 logických buněk
- paměť RAM 96x18 a 48x36 kilobitů
- 6x2 jednotky pro správu hodin (DCMs) [6]

5 Akcelerace DES na FPGA

Jak už bylo řečeno výše, pro demonstraci akcelerace šifrování na FPGA jsem si vybral DES, a to z několika důvodů. Za prvé, je to jedna z nejrozšířenějších šifer. Používá se již několik desetiletí, jsou k dispozici materiály potřebné k její implementaci. Za druhé, je k akcelerování na FPGA velmi vhodná.

DES byl projektován jako rychlý a malý algoritmus; probíhá zde minimální množství výpočtů a většina operací sestává ze substitucí a permutací. Právě to jej činí ideální k hardwarové implementaci.

5.1 Nástroje

Pro zápis zdrojových kódů budu používat jednoduchý textový editor (gedit).

Pro simulace implementace VHDL souborů budu používat ModelSim SE 6.2j. Je to nástroj, který umožňuje simulovat chování obvodů zapsaných pomocí VHDL (jde defacto o průmyslový standard mezi simulátory HDL [7]). Lze tak okamžitě kontrolovat správnost implementace; zjistit, zda se chová tak, jak má. K tomu je vhodné používat tzv. testbench, který obalí navrhovaný design, posílá data na vstup a sleduje výstup. Bohužel jsem musel použít tuto starší verzi. Je to totiž jediná verze, kterou se mi podařilo zprovoznit pod operačním systémem GNU/Linux, který používám.

Dále budu používat Xilinx ISE Design Suite. Jedná se o vývojové prostření, které je určeno pro vývoj hardwarových implementací. Umožňuje tvorbu a úpravu zdrojových VHDL kódů. Obsahuje taky množství nástrojů potřebných i dalších pro překlad, analýzu, syntézu a mapování programů pro programovatelný hardware.

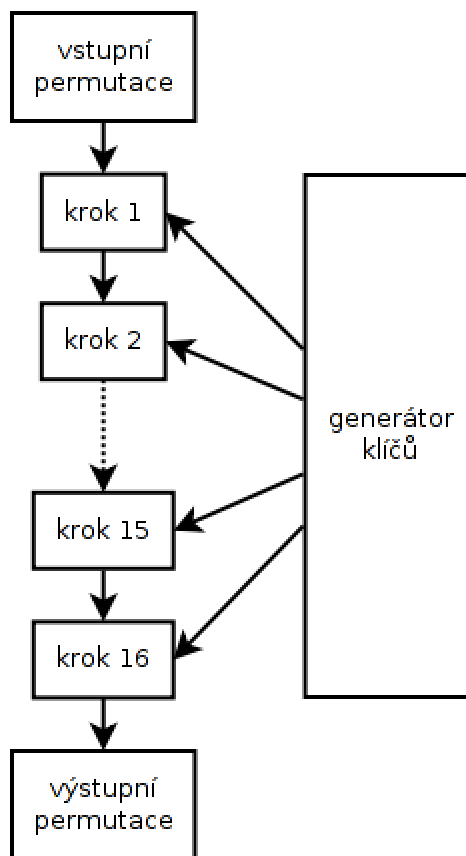
5.2 Hardware

Jako cílové platformy budu používat již zmíněné karty *FITkit* a *PICO E16*. Kartu FITkit mám zapůjčenou, proto můžu testovat implementaci přímo na ní. Karta PICO však v době psaní této kapitoly není k dispozici (fakulta vlastní pouze 1 kus). Uvidíme tedy, zda budu moci implementaci vyzkoušet i na této kartě.

5.3 Návrh

V rozboru algoritmu DES jsem již uvedl, že se algoritmus skládá z několika základních komponent. V tomto případě bude vhodné preferovat strukturální popis obvodu, zápis bude přehledný a logicky

strukturovaný. Návrh budu provádět odshora dolů, tzn. začnu od top entity, kompletního algoritmu a budu pokračovat až k jednotlivým součástem obvodu. Tabulky a obrázky uvedené v této kapitole jsou z [9].



Obrázek 5.1: Návrh - DES - schéma

Algoritmus DES sestává tedy z následujících komponent: počáteční permutace, 16ti kroků šifrovací funkce, konečné permutace a generátoru podklíčů. Celý algoritmus lze implementovat jako tzv. černou skříňku. Ta se navenek prezentuje pouze svým rozhraním. Minimální rozhraní musí obsahovat jednak vstupy, a to pro šifrovaný text a klíč. Dále musí obsahovat výstup pro zašifrovaný text. Na vstupu můžu ještě předpokládat synchronizační režii (hodinový signál) a signál pro asynchronní reset.

5.3.1 Jednotlivé komponenty

Vstupní permutace provádí mapování vstupu na výstup dle následující tabulky:

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

Tabulka 5.1: Mapování vstupu na výstup vstupní permutací [9]

Jak lze vidět v tabulce, vstupní permutace nebude potřebovat žádné logické zdroje. Její zapojení se dá provést jednoduchým napojením vstupních pinů na příslušné výstupní piny. (Pro upřesnění, tabulka se interpretuje následujícím způsobem: první pin výstupu bude napojen na 58. bit vstupu, druhý pin výstupu bude napojen na 50. pin vstupu atd.)

Šifrovací funkce je funkce, která provádí vlastní šifrování. V designu se 16x opakuje. Jde o strukturovanou komponentu, která sestává z následujících součástí:

- *Expanzní permutace:* provádí rozšíření vstupu z 32 bitů na 48 bitů. Některé bity jsou přitom použity vícekrát. Permutaci lze popsat následující tabulkou (interptretuje se stejným způsobem jako tabulka předchozí):

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

Tabulka 5.2: Mapování vstupu na výstup expanzní permutací [9]

  - Opět, jako v případě vstupní permutace, se jedná o prosté mapování vstupních bitů na výstupní. Implementace proto zřejmě nebude spotřebovávat žádné logické zdroje.

- *Exkluzivní disjunkce:* provádí zmíněnou operaci nad dvěma vstupy – výstupem expanzní permutace a příslušným podklíčem daného kroku (viz generování podklíčů). Implementace tohoto kroku bude provedena pomocí operace exkluzivní disjunkce nad všemi jednotlivými bity obou vstupů.

- *Substituce pomocí s-boxů:* s-boxy jsou jádrem bezpečnosti šifry DES. Provádí nelineární transformaci. Fungují na principu nahrazení šestice vstupních bitů čtveřicí výstupních bitů. Jde v podstatě o tzv. lookup table, kdy první a poslední bit určují řádek tabulky a prostřední 4 bity určují sloupec tabulky. Vybraná buňka je pak výstupem pro daný vstup. Pro představu uvádím část tabulky, určující chování s-boxu číslo 1 ve standardní implementaci:

s-box 1	x0000x	x0001x	x0010x	x0011x	x0100x	x0101x	x0110x	x0111x
0yyyy0	14	4	13	1	2	15	11	8
0yyyy1	0	15	7	4	14	2	13	1
1yyyy0	4	1	14	8	13	6	2	11
1yyyy1	15	12	8	2	4	9	1	7

Tabulka 5.3: Část tabulky definující s-box 1 [9]

- Následuje další *permutace*, opět pouze mapuje vstupní bity na výstupní (dle následující tabulky) a nespotřebovává žádné logické zdroje.

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

Tabulka 5.4: Mapování vstupu na výstup permutací [9]

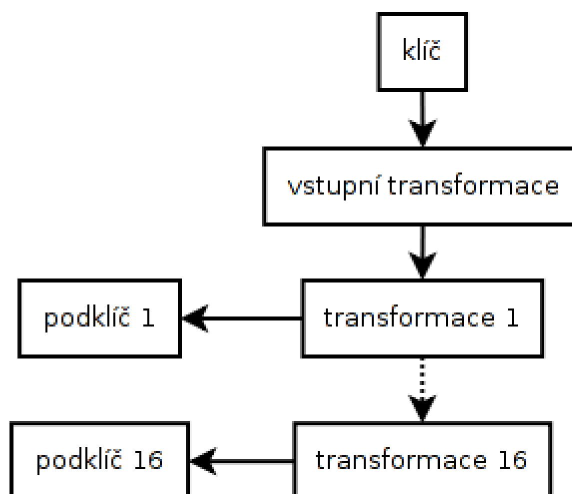
- Poslední operací v rámci šifrovacího kroku je exkluzivní disjunkce výsledku předchozí operace s levou polovinou vstupu šifrovacího kroku. Platí pro ni stejné závěry jako pro exkluzivní disjunkci rozšířeného datového bloku s podklíčem daného kroku.

Výstupní permutace je inverzní operací ke vstupní permutaci. To znamená, že pokud bychom na jeden blok dat aplikovali nejprve vstupní a následně výstupní permutaci, tak dostaneme původní blok dat. Platí pro ni stejné závěry jako pro permutaci vstupní.

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

Tabulka 5.5: Mapování vstupu na výstup výstupní permutace [9]

Generátor klíčů je součást obvodu, která přivádí na vstup jednotlivých kroků šifrovací funkce odpovídající podklíče. Jako vstup slouží klíč, který je redukován vstupní transformací. Redukovaný klíč poté postupně prochází transformacemi, jejichž výstupem je již příslušný podklíč. Jednotlivé transformace sestávají z rotace (každá polovina klíče zvlášť) a následné permutace. Permutace jsou ve všech krocích stejné, mění se pouze počet levotočivých rotací.



Obrázek 5.2: Návrh - DES - schéma generátoru klíčů

krok	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
počet rotací	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Tabulka 5.6: Počet rotací klíče v jednotlivých krocích generování [9]

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

Tabulka 5.7: Mapování vstupu na výstup první a druhé permutace generátoru klíčů [9]

Transformace, permutace i rotace lze realizovat pouze správným propojením vstupů a výstupů, proto zde opět platí, že tato část obvodu nebude používat logické zdroje.

5.3.2 Optimalizace

Optimalizací bude použití zřetězeného zpracování dat. Algoritmus mj. sestává z 16ti stejných, opakujících se za sebou zapojených kroků šifrovací metody.

5.4 Implementace

Implementaci jsem prováděl v jazyce VHDL. Popis implementace jednotlivých komponent jsem pro přehlednost rozdělil do souborů – každou komponentu do jednoho souboru.

5.4.1 top.vhdl

Tento soubor popisuje komponentu zapouzdřující celý algoritmus (entita DES). Popis rozhraní:

```
entity DES is
      port (
            plaintext   :     in std_logic_vector(1 to 64);
            key         :     in std_logic_vector(1 to 64);
            ciphertext  :     out std_logic_vector(1 to 64);
            clk         :     in std_logic;
            encrypt     :     in std_logic;
            reset       :     in std_logic
      );
end;
```

Rozhraní entity obsahuje 64bitový vektor pro vstup dat (`plaintext`), další pro vstup klíče (key) a poslední pro výstup zašifrovaného textu (`ciphertext`). Dále obsahuje vodiče pro vstup řídících signálů – `clk` pro synchronizační hodinový signál, `encrypt` pro výběr režimu algoritmu (v logické jedničce provádí šifrování, jinak dešifruje) a asynchronní `reset`.

Entita obsahuje následující komponenty: vstupní permutaci (`DES_INITIAL_PERMUTATION`), generátor klíčů (`DES_KEY_SCHEDULER`), komponenty jednotlivých kroků šifrovací funkce (`DES_STEP`) – celkem 16krát a výstupní permutaci (`DES_FINAL_PERMUTATION`).

Obsahuje dále řadu signálů sloužících k propojení uvedených komponent. Jsou to signály `l0` – `l16` a `r0` – `r16`, které propojují jednotlivé kroky šifrovací funkce. Dále jsou to signály `subkey1` – `subkey16` a `subkey1x` – `subkey16x`, které přivádějí podklíče k odpovídajícím krokům. Jsou jich 2 sady a mezi sebou jsou propojovány na základě signálu `encrypt` (rozhoduje o pořadí připojení podklíčů).

5.4.2 des_initial_permutation.vhd

Tento soubor popisuje komponentu provádějící vstupní permutaci. Rozhraní je následující:

```
entity DES_INITIAL_PERMUTATION is  port (
      input                  :     in std_logic_vector(1 to 64);
      output_l, output_r     :     out std_logic_vector(1 to 32)
      );
end;
```

Rozhraní obsahuje 64bitový vektor pro vstup dat a dva 32bitové vektory pro výstup. Komponenta provádí přiřazení vstupu na oba výstupy dle tabulky 5.1 na úrovni jednotlivých bitů.

5.4.3 des_final_permutation.vhd

Tento soubor popisuje komponentu provádějící výstupní permutaci. Rozhraní je následující:

```
entity DES_FINAL_PERMUTATION is port (
      input_l, input_r  :     in std_logic_vector(1 to 32);
      output            :     out std_logic_vector(1 to 64)
      );
end;
```

Rozhraní obsahuje dva 32bitové vektory pro vstup a jeden 64bitový pro výstup dat. Komponenta provádí přiřazení obou vstupů na výstup dle tabulky 5.5 na úrovni jednotlivých bitů. Implementace se od vstupní permutace liší minimálně.

5.4.4 des_step.vhd

Tento soubor popisuje komponentu jednoho kroku šifrovací funkce. Rozhraní je následující:

```
entity DES_STEP is port (
      input_l, input_r  :     in std_logic_vector(1 to 32);
      subkey            :     in std_logic_vector(1 to 48);
      output_l, output_r      :     out std_logic_vector(1 to 32);

      reset, clk        :     in std_logic
      );
end;
```

Rozhraní obsahuje dva 32bitové vstupní a výstupní datové vektory, 48bitový vektor pro podklíč, vstup pro hodinový signál a asynchronní reset. Komponenta obsahuje dále signály `feisteled` a `xored`.

Skládá se z následujících komponent: funkce f (DES_FEISTEL - `feistel`), exkluzivní disjunkce pro 32 bitů (XOR32 - `xor32`) a dvou 32bitových registrů (REGISTER32 – `reg_l`, `reg_r`).

Data na pravém vstupu (`input_r`) jsou vyvedena na levý výstupní registr (`reg_l`) a na vstup funkce f, do které je přiveden i signál `subkey`. Výstupem funkce f je signál `feisteled`, který je

připojen společně s levým vstupem `input_l` na `xor32`. Výstupem `xor32` je signál `xored`, který je přiveden na pravý výstupní registr `reg_r`.

Přítomnost obou výstupních registrů a synchronizačního hodinového signálu je zde nutná kvůli zřetězenému zpracování.

5.4.5 des_feistel.vhd

Tento soubor popisuje komponentu funkce f. Rozhraní je následující:

```
entity DES_FEISTEL is port (
      input       :     in std_logic_vector(1 to 32);
      subkey      :     in std_logic_vector(1 to 48);
      output      :     out std_logic_vector(1 to 32)
      );
end;
```

Rozhraní obsahuje 32bitový vstupní a výstupní vektor a 48bitový vstupní vektor pro podklíč. Dále obsahuje vnitřní signály `expanded`, `xored` a `sboxed`.

Skládá se z následujících komponent: expanzní permutace (DES_FEISTEL_EXPANSION – `expansion`), 48bitové exkluzivní disjunkce (XOR48 – `xor48b`), substitučních s-boxů (DES_FEISTEL_SBOX – `sboxes`) a permutace (DES_FEISTEL_PERMUTATION – `permutation`).

Jednotlivé součásti jsou za sebou jednoduše propojeny v tomto pořadí: `expansion`, `xor48b`, `sboxes`, `permutation`. Na vstup `xor48b` je přiveden také podklíč `subkey`.

5.4.6 des_feistel_expansion

Tento soubor popisuje komponentu expanzní permutace. Rozhraní je následující:

```
entity DES_FEISTEL_EXPANSION is port (
      input       :     in std_logic_vector(1 to 32);
      output      :     out std_logic_vector(1 to 48)
      );
end;
```

Rozhraní obsahuje 32bitový vstupní vektor a 48bitový výstupní vektor. Komponenta provádí expanzní permutaci dle tabulky 5.2 přiřazením vstupu na výstup na úrovni jednotlivých bitů.

5.4.7 xor48.vhd

Tento soubor popisuje komponentu exkluzivní disjunkce pro 2 48bitové vektory. Rozhraní je následující:

```
entity XOR48 is port (
      input1, input2    :     in std_logic_vector(1 to 48);
      output            :     out std_logic_vector(1 to 48)
      );
end;
```

Rozhraní obsahuje 2 vstupní a jeden výstupní 48bitový vektor. Komponenta přivádí na výstup exkluzivní disjunkci obou vstupů.

5.4.8 des_feistel_sbox.vhd

Tento soubor popisuje komponentu substitučních boxů (s-boxů). Rozhraní je následující:

```
entity DES_FEISTEL_SBOX is port (
      input       :     in std_logic_vector(1 to 48);
      output      :     out std_logic_vector(1 to 32)
      );
end;
```

Rozhraní obsahuje vstupní 48bitový vektor a výstupní 32bitový vektor.

Komponenta obsahuje 8 s-boxů. Každý takový s-box má 6bitový vstup a 4bitový výstup. Každý s-box je implementován jako lookup table. Následující kód ukazuje implementaci části prvního s-boxu:

```
-- S1
with input(1 to 6) select
output(1 to 4) <=
      "1110" when "000000", "0000" when "000001",
      "0100" when "100000", "1111" when "100001",
      "0100" when "000010", "1111" when "000011",
      "0001" when "100010", "1100" when "100011",
atd...
```

Stejným způsobem jsou zapsány všechny s-boxy. K vytvoření tabulek hodnot byla použita tabulka 5.3.

5.4.9 des_feistel_permutation.vhd

Tento soubor popisuje komponentu permutace ve funkci f. Rozhraní je následující:

```
entity DES_FEISTEL_PERMUTATION is port (
      input       :     in std_logic_vector(1 to 32);
      output      :     out std_logic_vector(1 to 32)
      );
end;
```

Rozhraní obsahuje jeden vstupní a jeden výstupní 32bitový vektor. Jako ostatní permutace pouze mapuje vstup na výstup na úrovni jednotlivých bitů. K implementaci bylo použito tabulky 5.4.

5.4.10 xor32.vhd

Tento soubor popisuje komponentu exkluzivní disjunkce pro 2 32bitové vektory. Rozhraní je následující:

```
entity XOR32 is port (
      input1, input2    :     in std_logic_vector(1 to 32);
      output            :     out std_logic_vector(1 to 32)
      );
end;
```

Rozhraní obsahuje 2 vstupní a jeden výstupní 32bitový vektor. Komponenta přivádí na výstup exkluzivní disjunkci obou vstupů.

5.4.11 register32.vhd

Tento soubor popisuje komponentu 32bitového registru. Rozhraní je následující:

```
entity REGISTER32 is port (
      input       :     in std_logic_vector(1 to 32);
      output      :     out std_logic_vector(1 to 32);
      reset, clk  :     in std_logic
      );
end;
```

Rozhraní obsahuje jeden vstupní a jeden výstupní 32bitový vektor, hodinový signál `clk` a asynchronní `reset` pro vynulování obsahu registru.

Tato komponenta je jako jediná popsána pomocí procesu, kde je v sensitivity listu signál `clk` a `reset`, na které komponenta reaguje. V případě, že je signál `reset` v úrovni logické 1, je na výstup registru přiveden vektor nul. V opačném případě je při náběžné hraně hodinového signálu (`clk`) přiveden na výstup vstupní vektor.

```
process(clk,reset)
begin
      if reset='1' then -- asynchronni reset aktivni pri log1
            output <= (others => '0');
      elsif (clk'event and clk='1') then  -- nabezna hrana hodin
            output <= input;
      end if;
end process;
```

5.4.12 des_key_scheduler.vhd

Tento soubor popisuje komponentu generátoru podklíčů pro jednotlivé kroky šifrovací funkce. Rozhraní je následující:

```
entity DES_KEY_SCHEDULER is port (
      input :     in std_logic_vector(1 to 64);
      output1, output2, output3, output4,
      output5, output6, output7, output8,
      output9, output10, output11, output12,
      output13, output14, output15, output16
            :     out std_logic_vector(1 to 48)
      );
end;
```

Rozhraní obsahuje vstupní 64bitový vektor nesoucí šifrovací klíč a 16 výstupních 48bitových vektorů pro jednotlivé podklíče. Dále využívá 2x16 interní signálů pro přenos dat mezi jednotlivými kroky výpočtu podklíče.

Skládá se z komponenty DES_KEY_PERMUTED_CHOICE_1 (`pc1`) a 16ti komponent DES_KEY_PERMUTED_CHOICE_2 (`pc2_1 – pc2_16`). Klíč nejprve projde přes `pc1` a následuje 16 kroků, kdy klíč vždy rotuje a poté projde přes `pc2` na odpovídající výstup.

Zde bych zmínil způsob provedení rotace. V jazyce VHDL existují operátory ror a rol, které provádí rotaci operadnu. Bohužel operand nemůže být vektorem, proto jsem se místo převodů spokojil s následující konstrukcí:

```
l1 <= l0(2 to 28) & l0(1);    r1 <= r0(2 to 28) & r0(1);
```

Rotace bylo dosáhnuto mapováním vstupních bitů 2-28 na výstupní bity 1-27 a vstupního bitu 1 na výstupní bit 28.

5.4.13 des_key_permuted_choice_1.vhd

Tento soubor popisuje komponentu první permutace generátoru klíčů. Rozhraní je následující:

```
entity DES_KEY_PERMUTED_CHOICE_1 is port (
      input                  :     in std_logic_vector(1 to 64);
      output_l, output_r     :     out std_logic_vector(1 to 28)
      );
end;
```

Rozhraní obsahuje 64bitový vstupní vektor a dva 28bitové výstupní vektory. Komponenta provádí permutaci (mapování vstupu na výstupy na úrovni bitů) dle tabulky 5.7.

5.4.14 des_key_permuted_choice_2.vhd

Tento soubor popisuje komponentu první permutace generátoru klíčů. Rozhraní je následující:

```
entity DES_KEY_PERMUTED_CHOICE_2 is port (
      input_l, input_r  :     in std_logic_vector(1 to 28);
      output            :     out std_logic_vector(1 to 48)
      );
end;
```

Rozhraní obsahuje dva 28bitové vstupní vektory a jeden 48bitový výstupní vektor. Komponenta provádí permutaci (mapování vstupu na výstupy na úrovni bitů) dle tabulky 5.7.

5.5 Simulace

Simulace je proces validace popisu algoritmu bez nutnosti syntézy na cílovou platformu. Pro simulaci jsem použil ModelSim SE PLUS 6.2j (novější verze se mi nepodařilo na mém operačním systému zprovoznit).

5.5.1 Testbench

Pro potřeby simulace jsem musel vytvořit tzv. testbench, což je komponenta, která zapouzdří algoritmus DES a bude s ním komunikovat. Bude posílat na vstup potřebná data a zachytávat výstup. [11]

Testbench je popsán v souboru `testbench.vhd a` je označen jako komponenta `tb_fa`.V popisu komponenty `DES` jsem již uvedl její rozhraní. Sestává ze vstupního 64bitového vektor `plaintext`, který obsahuje původní data, dále vstupní 64bitový vektor se šifrovacím klíčem `key` a 64bitový výstupní vektor `ciphertext` se zašifrovanými daty. Navíc obsahuje komponenta ještě vstupy pro tři jednoduché signály – `clk` (hodinový synchronizační signál), `encrypt` (volba režimu algoritmu – šifrování/dešifrování) a asynchronní `reset`.

Průběh simulace bude vypadat následovně:

- Po celou dobu bude na `clk` přiváděn hodinový signál s periodou 10 ns:

```
clk <= '1'; wait for 5 ns; clk <= '0'; wait for 5 ns;
```

- Nejprve provedu asynchronní reset (`reset` nastavím na logickou 1). Na výstupu `ciphertext` by měl být vektor samých nul:

```
reset <= '1';
```

- Po jednom hodinovém taktu přivedu na `key` vybraný testovací šifrovací klíč:

```
key <= X"FFFFFFFF00000000";
```

- Nastavím režim algoritmu na šifrování (`encrypt` na logickou jedničku). Tento signál by měl správnou aplikaci šifrovacích podklíčů v jednotlivých krocích (v původním pořadí – 1-16):

```
encrypt <= '1';
```

- Po dalším hodinovém taktu zruším reset (`reset` na logickou 0) a počkám další takt.
- Nyní už budu přivádět na vstup `plaintext` testovací data – celkem 4krát 64bitový blok. Mezi každým vstupem musím počkat 1 takt:

```
plaintext <= X"0123456789ABCDEF";
   clk <= '1'; wait for 5 ns; clk <= '0'; wait for 5 ns;
```

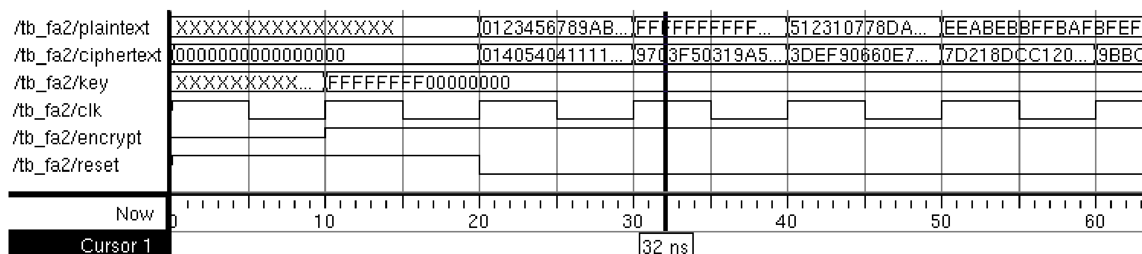
- Spávné výsledky se na výstupu `ciphertext` začnou objevovat až 16 taktů po prvním vstupu. Proto je třeba po posledním vloženém bloku dat vložit minimálně 16 taktů. Důvodem je zřetězené zpracování, kdy design obsahuje 16 šifrovacích kroků a za každým krokem je umístěn registr reagující na hodinový signál. Po průchodu prvního bloku už však další bloky budou na výstupu každý další takt.
- Po průchodu posledního ze 4 bloků vyzkouším dešifrování zašifrovaných dat.
- Nastavím režim (`encrypt` na logickou nulu).
- Poté přivedu během 4 taktů 4 bloky zašifrovaných dat k dešifrování. Postup je stejný jako při šifrování. Po 16 taktech po posledním vloženém bloku by měly být na výstupu již všechny dešifrované bloky.

Poznámka k průběhu fungování algoritmu: během průchodu jednotlivých bloků nelze měnit režim algoritmu (šifrování/dešifrování). Došlo by totiž k aplikaci nesmyslné sekvence podklíčů a na výstupu algoritmu by nebyla správná data. Proto je třeba při změně režimu počkat, až projdou všechna předchozí data (16 taktů po posledním vstupu) a teprve poté režim změnit.

5.5.2 Výsledky simulace

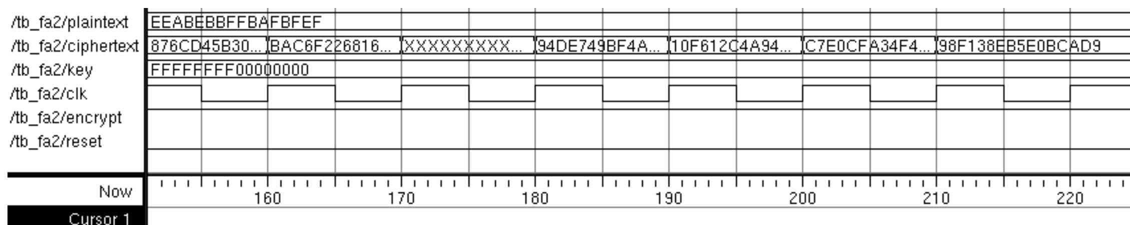
Na následujících obrázcích budeme sledovat průběh simulace tak, jak proběhla v ModelSimu (jedná se o obrazový výstup z ModesSimu).

Na obrázcích je v levé části vidět seznam signálů: `plaintext` (vstupní data), `ciphertext` (výstupní data), `key` (šifrovací klíč), `clk` (hodinový signál), `encrypt` (volba režimu) a `reset`. V pravé části je pak vidět průběh těchto signálů v čase (vypsáno hexadecimálně). Časová osa ve spodní části je v nanosekundách.



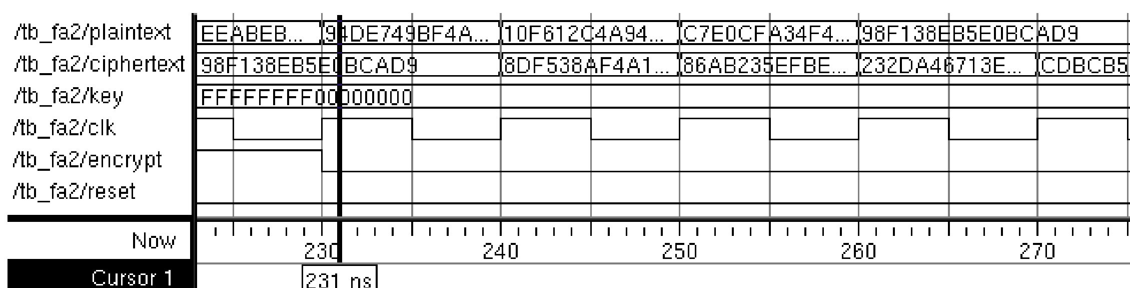
Obrázek 5.3: Průběh simulace - šifrování - vstup

Na obrázku 5.3 můžeme vidět aktivní reset – na výstupu jsou samé nuly. Signál `key` byl po jednom taktu nastaven na požadovanou hodnotu a signálem `encrypt` bylo zvoleno šifrování. V dalším taktu jsou již na vstup (`plaintext`) přiváděny bloky dat.

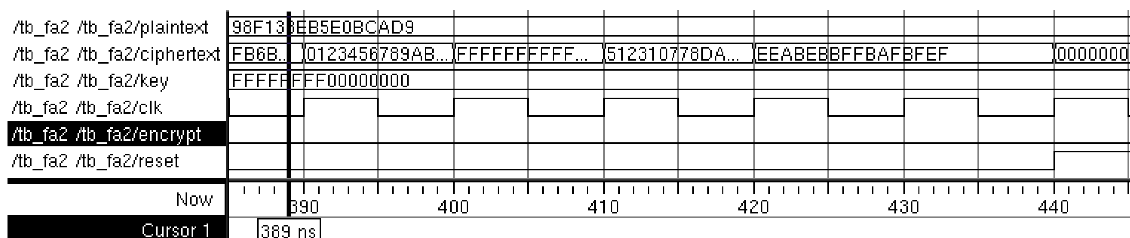


Obrázek 5.4: Průběh simulace - šifrování - výstup

Na obrázku 5.4 můžeme vidět, že v čase 180ns – tedy 16 taktů po vložení prvního bloku dat na vstup (0123456789ABCDEF) – se objevuje první odpovídající výstup (94DE749BF4A32CD4); následují zbylé 3 bloky. Simulace šifrování tedy proběhla úspěšně.



Obrázek 5.5: Průběh simulace - dešifrování - vstup



Obrázek 5.6: Průběh simulace - dešifrování - výstup

Na obrázcích 5.5 a 5.6 je pak vidět pokračování simulace pro dešifrování. Na prvním obrázku jsou na vstup přivedeny výsledky předchozího šifrování spolu se změnou signálu `encrypt` do log.0 (volba režimu na dešifrování). Na druhém obrázku je vidět, že po 16ti taktech se na výstupu objevují správné hodnoty. Úplně na konci simulace je pak vidět opět aktivní reset.

Dle simulace proběhl tedy návrh a zápis algoritmu v jazyku VHDL úspěšně, algoritmus pracuje dle očekávání.

5.6 Syntéza

Syntéza je klíčovým krokem mezi vysoce úrovňovým návrhem a fyzickým rozmístěním prvků celého procesu. Behaviorální syntéza je mechanismus, kdy je abstraktní model převeden na fyzicky

realizovatelný model nižší úrovně. RTL syntéza je pak mechanismus, který umožňuje mapování VHDL kódu na konkrétní logické prvky cílové platformy.

5.6.1 Cílové platformy

Jako cílové platformy pro syntézu DES algoritmu jsem si zvolil FPGA čipy Spartan 3 a Virtex 5. Oba čipy pocházejí z dílny Xilinx a ve vývojovém prostředí Xilinx ISE Webpack je lze zvolit jako cíle pro syntézu.

Překlad probíhá v prostředí ISE automaticky, v rámci následujících podkapitol se budu věnovat rozboru zajímavých částí výpisů z průběhu překladu.

5.6.2 Implementace pro Spartan 3

Syntéza:

Substituční boxy byli syntetizátorem implementovány jako paměti ROM:

```
Synthesizing Unit <DES_FEISTEL_SBOX>.
  Related source file is "/home/mirek/fitkit-
svn/apps/bak/fpga/des_feistel_sbox.vhd".
  Found 64x4-bit ROM for signal <input_25_30$rom0000>.
  Found 64x4-bit ROM for signal <input_31_36$rom0000>.
  Found 64x4-bit ROM for signal <input_37_42$rom0000>.
  Found 64x4-bit ROM for signal <input_43_48$rom0000>.
  Found 64x4-bit ROM for signal <input_1_6$rom0000>.
  Found 64x4-bit ROM for signal <input_13_18$rom0000>.
  Found 64x4-bit ROM for signal <input_7_12$rom0000>.
  Found 64x4-bit ROM for signal <input_19_24$rom0000>.
  Summary:
    inferred   8 ROM(s).
Unit <DES_FEISTEL_SBOX> synthesized.
```

V obvodu bylo po syntéze přítomno 128 pamětí ROM (64 4bitových buněk), 32 registrů (2x 16 – pro každý krok z 16ti 2), 16 32bitových a 16 48bitových exkluzivních disjunkcí (v každém kroku jedna a jedna)-

```
Macro Statistics
# ROMs                                                 : 128
 64x4-bit ROM                                          : 128
# Registers                                            : 32
 32-bit register                                       : 32
 Flip-Flops                                            : 1024
# Xors                                                 : 32
 32-bit xor2                                           : 16
 48-bit xor2                                           : 16
```

V logu se poté objevila informace, která oznamovala nedostatek prostoru k umístění celého algoritmu na tento čip:

```
Optimizing block <DES> to meet ratio 100 (+ 5) of 768 slices :
WARNING:Xst:2254 - Area constraint could not be met for block
 <DES>, final ratio is 230.
```

Toto bylo potvrzeno v následující tabulce, která shrnuje využití zdrojů cílové platformy:

```
 Number of Slices:            1927  out of    768   250%(*)
 Number of Slice Flip Flops:  1024  out of   1536    66%
 Number of 4 input LUTs:      3748  out of   1536   244%(*)
 Number of IOs:                195
 Number of bonded IOBs:        187  out of    124   150%(*)
 Number of GCLKs:                1  out of      8    12%
WARNING:Xst:1336-(*)More than 100% of Device resources are used
```

Co se týče časování, následující výpis informuje o maximální možné pracovní frekvenci, která činí 185.415MHz:

```
 Minimum period: 5.393ns (Maximum Frequency: 185.415MHz)
 Minimum input arrival time before clock: 11.835ns
 Maximum output required time after clock: 6.216ns
 Maximum combinational path delay: No path found
```

Mapování:

Zpráva z procesu mapování již jen potvrzuje nedostatek prostoru na čipu Spartan 3 pro tuto implementaci:

```
Logic Utilization:
  Number of Slice Flip Flops:     1,024 out of   1,536   66%
  Number of 4 input LUTs:         3,751 out of   1,536  244%
   (OVERMAPPED)
Logic Distribution:
  Number of occupied Slices:      2,129 out of     768  277%
   (OVERMAPPED)
  Number of Slices containing only related logic:
                                  2,129 out of   2,129  100%
  Number of Slices containing unrelated logic:
                                      0 out of   2,129    0%
  Total Number of 4 input LUTs:   3,751 out of   1,536  244%
   (OVERMAPPED)
  Number of bonded IOBs:            187 out of     124  150%
(OVERMAPPED)
  Number of BUFGMUXs:                 1 out of       8   12%
```

Na čipu je nedostatek logických buněk. Navíc implementace v současné podobě potřebuje na čipu 187 vstupně výstupních pinů, čip jich ale má pouze 124.

Závěr:

Implementaci nebylo možné na čip Spartan 3 fyzicky umístit, protože nemá dostatek logických buněk a vstupů/výstupů. Co se týče logických buněk, jejich využití by bylo možné změnit pouze výraznou úpravou implementace. Jednou z možností by bylo například vzdát se zřetězeného zpracování a použít stavového automatu. Toto řešení by však znamenalo výraznou ztrátu na datové propustnosti.

Pro snížení počtu využitých vstupů/výstupů by pak bylo třeba upravit způsob předávání dat. Toho lze dosáhnout například úpravou datového vstupu a vstupu pro klíč na vstupy užívající sekvenční předávání dat (např. po 32 bitech během 2 hodinových taktů).

5.6.3 Implementace pro Virtex 5

Syntéza:

Substituční boxy byly opět implementovány jako lookup table pamětí ROM (8 4bitových ROM pamětí na každý krok šifrování).

Počet registrů a exkluzivních disjunkcí je samozřejmě také stejný.

```
 Slice Logic Utilization:
  Number of Slice Registers:          1024  out of  28800     3%
  Number of Slice LUTs:               1792  out of  28800     6%
  Number used as Logic:               1792  out of  28800     6%
 Slice Logic Distribution:
  Number of LUT Flip Flop pairs used: 2309
  Number with an unused Flip Flop:    1285  out of   2309    55%
  Number with an unused LUT:           517  out of   2309    22%
  Number of fully used LUT-FF pairs:   507  out of   2309    21%
  Number of unique control sets:         1
 IO Utilization:
  Number of IOs:                       195
  Number of bonded IOBs:               187  out of    220    85%
 Specific Feature Utilization:
  Number of BUFG/BUFGCTRLs:              1  out of     32     3%
```

Z předcházejícího výpisu, který souhrnně informuje o využití zdrojů čipu, lze vyčíst, že implementaci bude možné na čip umístit. Dostačuje počet logických buněk i počet vstupů/výstupů.

Následující výpis informuje o maximální možné pracovní frekvenci, která činí téměř 500MHz:

```
   Minimum period: 2.028ns (Maximum Frequency: 493.170MHz)
   Minimum input arrival time before clock: 2.657ns
   Maximum output required time after clock: 2.775ns
   Maximum combinational path delay: No path found
```

Mapování:

Následující výpis potvrzuje využití zdrojů čipu po procesu mapování:

```
Slice Logic Utilization:
  Number of Slice Registers:         1,024 out of 28,800    3%
  Number used as Flip Flops:         1,024
  Number of Slice LUTs:              1,792 out of 28,800    6%
  Number used as logic:              1,792 out of 28,800    6%
Slice Logic Distribution:
  Number of occupied Slices:           624 out of  7,200    8%
  Number of LUT Flip Flop pairs used: 2,302
  Number with an unused Flip Flop:   1,278 out of  2,302   55%
  Number with an unused LUT:           510 out of  2,302   22%
  Number of fully used LUT-FF pairs:   514 out of  2,302   22%
  Number of slice register sites lost to control set
restrictions:                            0 out of 28,800    0%
```

5.6.4 Srovnání se softwarovou implementací

Jako referenční stroj pro porovnání jsem vybral notebook Toshiba Tecra A7 s procesorem CoreDuo s taktovací frekvencí 1.83GHz. Jako referenční softwarová implementace byla zvolena implementace algoritmu DES použitá v GnuPG verze 1.4.9.

Softwarovou implementaci jsem testoval tak, že jsem ji nechal zpracovat milion bloků dat beze změny klíče nebo režimu (bylo použito pouze šifrování). Čas trvání programu byl měřen pomocí programu GNU time. Pro zmíněných milion bloků trvala operace 0,676 sekundy. Po přepočtení mi vyšla datová propustnost 94,7 Mbit/s.

U hardwarové implementace syntetizátor xst určil maximální frekvenci 493 MHz, další analýza však určila frekvenci nižší – 358 MHz. Datová propustnost (po zaplnění vnitřní fronty zřetězené linky) by tedy měla činit 23000 Mbit/s. Toto číslo se však zdá nadsazené, nepodařilo se mi zjistit, zda byla chyba způsobena analyzátorem ve výpočtu frekvence, či zda to je jenom teoreticky dostupné maximum, kdy frekvence bude v praxi výrazně nižší. Každopádně datová propustnost dalších hardwarových implementací (např. uvedených v [8] nebo [9]) je v řádu stovek megabitů až desítek gigabitů za sekundu, takže jsme si oproti softwarové implementaci polepšili.

6 Závěr

V této práci jsem se zabýval tématy šifrování a akcelerování algoritmů pomocí FPGA. Po prostudování dostupných zdrojů o těchto tématech jsem chtěl zúročit nabyté znalosti a implementovat jeden z šifrovacích algoritmů právě pomocí FPGA. Zvolený algoritmus, symetrickou šifru DES, jsem nejprve popsal jazykem VHDL, poté provedl simulaci a nakonec vysyntetizoval pro 2 konkrétní FPGA čipy.

Vytvořená implementace může být použita k akceleraci šifrování a výrazně tak ušetřit čas pro větší objem dat. Zároveň jsem se při práci dozvěděl spoustu nových informací a nabyl mnoho zkušeností.

6.1 Analýza dosažených výsledků

Návrh i implementace algoritmu probíhali s ohledem na maximální rychlost algoritmu. Využil jsem zřetězeného zpracování dat, které výrazně urychlí průchod dat algoritmem. Díky tomu se však nepodařilo syntetizovat algoritmus na první ze dvou cílových platforem, čip Spartan 3 – algoritmus by spotřeboval příliš mnoho prostředků (víc než je k dispozici).

Čip Virtex 5 již však byl pro implementaci dostačující. Po provedení syntézy určil syntetizátor maximální pracovní frekvenci obvodu 358 MHz – to znamená, že datová propustnost by činila neuvěřitelných 23 Gb/s. Tento údaj je zřejmě hodně nadsazený (v porovnání s dalšími implemetacemi), nepodařilo se mi však zjistit, kde by mohla být chyba.

Bohužel jsem nemohl ověřit funkci algoritmu v praxi. V případě čipu Spartan 3, který je součástí výukové platformy FITkit, proto, že implementaci nebylo možno na čip vměstnat a úpravy algoritmu by byly příliš rozsáhlé (nezbýval na ně už ani čas). V případě čipu Virtex 5 byl pak problém nedostupnosti karty PICO E-16, která byla po celou dobu mojí práce zapůjčena mimo fakultu. Věřím však, že by byla implementace v reálu funkční a podpořila většinu závěrů vyvozených z rozboru, simulace a analýzy v průběhu syntézy.

6.2 Pokračování projektu, navrhovaná vylepšení

Rád bych se tomuto projektu věnoval i v budoucnu. Určitě bych chtěl nejprve vyzkoušet implementaci zprovoznit na reálném hardwaru, např. na čipu Virtex 5 na kartě PICO E-16.

Jako další rozšíření by mohla být realizace rozhraní či softwaru, pomocí kterého by bylo jednoduché implementaci obsluhovat (volit režim, nahrávat klíč a data, získávat jednoduše výstup). Vhodný by byl program s grafickým prostředím, kde by ovládání programu sestávalo pouze z výběru

souboru a zadání klíče. Program už by pak sám použil FPGA k zašifrování a uživateli by vrátil soubor hotových dat. Možné další vylepšení by mohlo sestávat v zakomponování do již existujícího programu, např. GnuPG. Šifrování algoritmem DES by se tak výrazně urychlilo. Hlavně pro větší objem dat by byl rozdíl velmi patrný.

Literatura

[1] Wikipedia, Kryptografie [online]. 11.4.2009 [cit. 2.5.2009]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Kryptografie>

[2] Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. Boca Raton, CRC Press, 2001, ISBN 0-8493-8523-7

[3] Wikipedia, Data Encryption Standard [online]. 15.4.2009 [cit. 5.5.2009]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Data_Encryption_Standard>

[4] Vašíček, Z.: FITkit – Úvod [online]. 2009 [cit. 7.5.2009]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/uvod.html>

[5] Vašíček, Z., Strnadel, J.: FITkit – FPGA [online]. 2009 [cit. 7.5.2009]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/docs/hardware/hw_fpga.html>

[6] Kolektiv autorů: Xilinx DS100 Virtex-5 Family Overview [online]. 6.2.2009 [cit. 8.5.2009]. Dostupný z WWW: <http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf>

[7] ASICentru: Modelsim [online]. 2009 [cit. 8.5.2009]. Dostupný z WWW: <http://www.asicentrum.cz/cz_01_03_01_01e.php>

[8] Wilson, P.: *Design Recipes for FPGAs*. Bodmin, MPG Books Ltd, ISBN 978-0-7506-6845-3

[9] Rodriguez-Henriquez, F., Saqib, N.A., Diaz-Perez, A., Koc, C.K..: *Cryptographic Algorithms on Reconfigurable Hardware* . New York, Springer Science+Business Media, 2006, ISBN 978-0-387-33883-5

[10] Gokhale, M., Graham, P.: *Reconfigurable Computing : Accelerating Computation with Field-Programmable Gate Arrays* . Dordrecht,, Springer, 2005, ISBN 978-0-387-26105-8

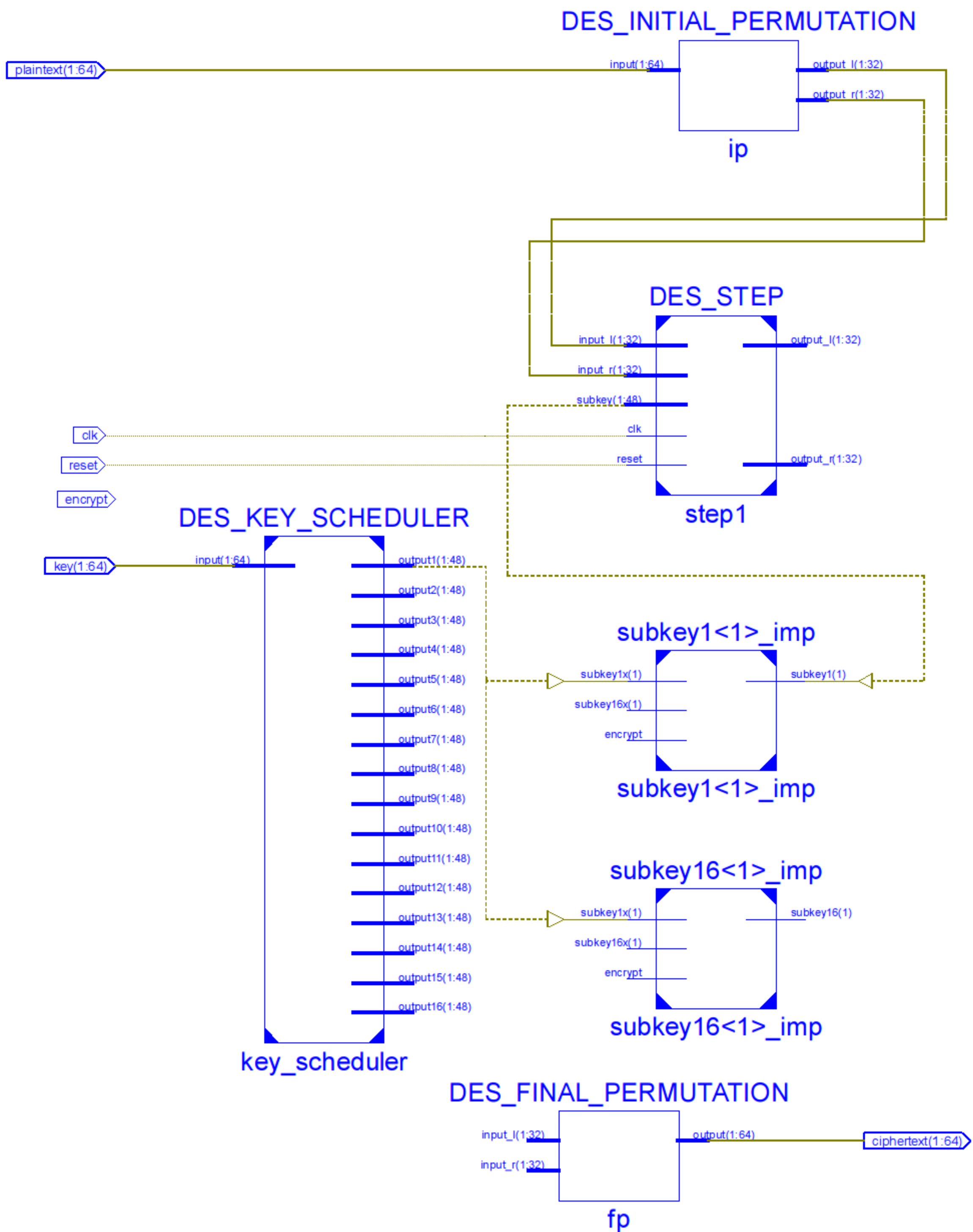
[11] Kilts, S.: *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Hoboken, Wiley-IEEE Press, 2007, ISBN 978-0470054376

[12] Hauck, S., DeHon, A.: *Reconfigurable Computing.* San Francisco, Morgan Kaufmann, 2007, ISBN 978-0-12-370522-8

Seznam příloh

Příloha 1. Zjednodušené schéma celého obvodu
Příloha 2. Schéma šifrovací funkce
Příloha 3. CD

Příloha 1. Zjednodušené schéma célého obvodu



Příloha 2. Schéma šifrovací funkce