elektrorevue
ISSN 1213 - 1539

Rok / Year:	Svazek / Volume:	Číslo / Number:	Jazyk / Language
2016	18	3	CZ

Akcelerace šifry AES pomocí programovatelných hradlových polí

Acceleration of AES by using FPGA

David Smékal, Jakub Frolka, Jan Hajný

smekal@phd.feec.vutbr.cz, frolka@feec.vutbr.cz, hajny@feec.vutbr.cz

Faculty of Electrical Engineering and Communication, BUT

DOI: -

Abstract: This article deals with encryption on Field Programmable Gate Array (FPGA). The first part of the article fuses on the analysis of the current state of implementation of asymmetric and symmetric ciphers. The next section describes the encryption algorithm AES and its own implementation using VHDL programming language. In the last part, the test results of our implementation to network card COMBO-80G, based on FPGA Xilinx Virtex-7 are shown.

elektrorevue
ISSN 1213 - 1539

Akcelerace šifry AES pomocí programovatelných hradlových polí

David Smékal, Jakub Frolka, Jan Hajný

Fakulta elektrotechniky a komunikačních technologií VUT v Brně
Email: smekal@phd.feec.vutbr.cz, frolka@feec.vutbr.cz, hajny@feec.vutbr.cz

Abstrakt *– Článek se zabývá šifrováním na programovatelných hradlových polí FPGA (Field Programmable Gate Array). První část článku je zaměřena na analýzu současného stavu implementací asymetrických a symetrických šifer. V další části je popsán šifrovací algoritmus AES a jeho vlastní implementace pomocí programovacího jazyka VHDL. V poslední části, jsou uvedeny výsledky testování implementovaného algoritmu AES na kartě COMBO-80G, založené na FPGA firmy Xilinx řady Virtex-7.*

1 Úvod

V dnešní době se klade velký důraz na bezpečnost a rychlost přenosu dat. Proto v poslední době, kdy se FPGA (Field Programmable Gate Array) stávají více a více dostupnější, se začali odborníci zabývat jejich vhodným využitím pro výkonnostně náročné aplikace. FPGA jsou programovatelná hradlová pole, která umožňují vývoj hardwarově akcelerovaných aplikací. FPGA jsou programovatelné logické obvody, které lze naprogramovat až po jejich výrobě. V dnešní době obsahují především programovatelné logické bloky, RAM paměti a multiplexery. Výhodou těchto logických obvodů je jejich univerzálnost.

1.1 Asymetrické šifrování na FPGA

Jedním z mnoha využití mohou být asymetrické šifry. V roce 1999 se autoři [1] zabývali modulárním mocněním na FPGA řady Xilinx XC 4000, pro RSA bitové délky 1024 bitů zvládal šifrovat za 0,75 ms a dešifrovat za 10,18 ms. Tento návrh byl později v článku [2] upraven, optimalizován a následně implementován do FPGA firmy Xilinx XC40250XV, pro 1024 bit RSA se šifrování zrychlilo na 0,22 ms a dešifrování na 3,1 ms při taktovací frekvenci 45,6 MHz.

V práci [3], navržený šifrátor/dešifrátor, implementovaný na Xilinx V1000FG680-6, pro RSA bitové délky 1024 bitů dosahuje datové propustnosti 48,2 kb/s, která odpovídá času 21,25 ms na 1024 bitový blok dat. V článku [4] je navržena architektura pro mocnění, RSA 1024 bitové délky dosahují datové propustnosti pro šifrování 4,79 Mb/s a dešifrování 375,54 kb/s, které odpovídají časům 0,214 ms pro šifrování a 2,73 ms pro dešifrování. Tyto hodnoty byly získány pouze analýzou pro FPGA Xilinx XC2V6000. V článku [5] využili čipu Xilinx XVC2000E-6, 1024 bitové RSA mocnění je možné provést s frekvencí 69,4 MHz a časem 6,1 ms.

Další výzkumy [6] a [7] se zabývaly návrhům, které jsou odolné proti útokům na postranní kanály. Návrh v [6] byl syntetizován na Virtex2 xc2v6000, pro 1024 bit RSA dosahuje rychlosti do 150 ms. V článku [7] byl navržen algoritmus modulárního mocnění, který byl pro 1024 bitů testován na Virtex 5 XC5VLY50T při frekvenci 274 MHz.

1.2 Symetrické šifrování na FPGA

Další skupina odborníků se zabývala i symetrickými šiframi. V článku [8] z roku 2001 se zohledňuje právě hardwarová implementace na FPGA v různých algoritmech, ze kterých jako vítěz vyšel algoritmus Rijndael, který tvoří základ pro AES (popsáno v části 2). Výsledek vykazuje silný výkon v implementacích na hardwaru.

Způsobů jak implementovat jednotlivé operace šifrování v FPGA je mnoho. Efektivnímu návrhu se věnuje práce [9], který testuje různé architektury. Praktický realizace [10] přináší v roce 2007 implementaci šifry AES-128 na čipu Xilinx Virtex-4 (XC4VFX12). Čip vykazuje hodnoty práce s daty až 640 Mb/s. Rychlost zpracování dat jednoho bloku kolem je 30 Mb/s.

V části 3 tohoto článku prezentujeme implementaci šifrovacího algoritmu Advanced Encryption Standard. AES je v současné době velmi perspektivní algoritmus, který je považován za dostatečně bezpečný. Stal se prakticky celosvětovým komerčním standardem. Naše implementace je směřována na platformu síťových karet typu FPGA. FPGA karta je osazena čipem Virtex-7 od společnosti Xilinx. Hlavním zájmem je implementovat všechny moduly šifrovacího algoritmu na hardware.

2 Advanced Encryption Standard

Advanced Encryption Standard, neboli AES [11], je algoritmus používaný k šifrování dat. Jedná se o symetrickou blokovou šifru, která zpracovává data rozdělená do bloků pevně dané délky 128 bitů. Tyto bity jsou uspořádány do matice 4×4, kdy jedna buňka matice odpovídá jednomu bajtu. K šifrování a dešifrování se používá stejný klíč o velikosti 128, 192 nebo 256 bitů. V tomto článku pracujeme s klíčem 128 bitů.

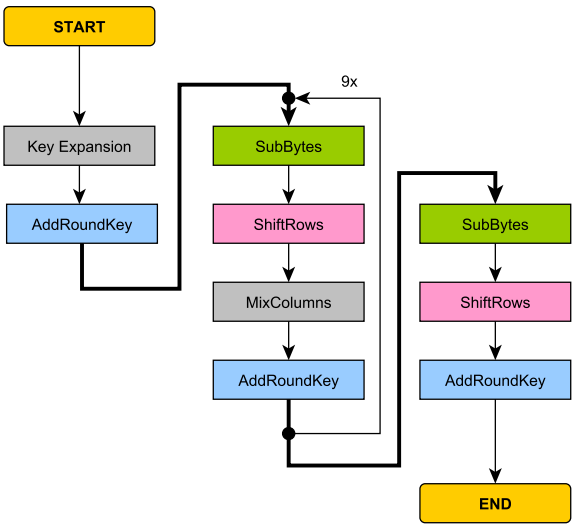
elektrorevue
ISSN 1213 - 1539

Algoritmus lze rozdělit na tři části [12]:

- iniciační část (Key Expansion, AddRoundKey),
- iterační část – tzv. runda (SubBytes, ShiftRows, MixColumns, AddRoundKey),
- závěrečná část (SubBytes, ShiftRows a AddRoundKey).

Na začátku šifrování se provede expanze klíče. Ke stavové matici 4×4 vytvořené z bloku 128 bitů dat se v šifře přixoruje klíč. Poté se devětkrát provede iterace, která se běžně označuje jako runda. Počet provedení rundy se odvíjí v závislosti na použité délce klíče. Počet 9 odpovídá klíči o délce 128 bitů. Každá runda se skládá ze substituce bajtů stavové matice (SubBytes), rotace řádků (ShiftRows), následně substituce sloupců (MixColumns). Na konci každé rundy se k matici přičte rundovní (iterační) klíč (AddRoundKey).

V závěrečné části se opět uskuteční substituce bajtů, rotace řádků a poslední fází je přičtení klíče finální rundy. Ve výsledné matici jsou uloženy bajty šifrového textu. Tyto popsané kroky jsou zobrazeny na Obrázku 1.



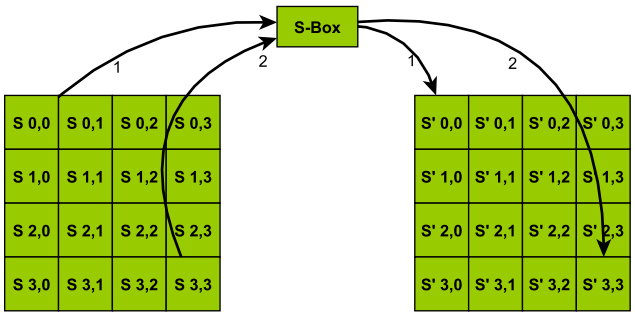
Obrázek 1: Jednotlivé kroky šifrování AES.

Dešifrování je opačný postup šifrování, kde výstupem bude zpráva, která byla na počátku celého šifrovacího procesu. Dešifrovací algoritmus se provádí v opačném pořadí za použití inverzních transformací, neboť všechny operace jsou reverzibilní. Inverzní operace je změna transformací v opačném pořadí nebo s použitím jiných hodnot. K šifrovací transformaci SubBytes je inverzní transformací InvSubBytes využívající inverzní substituční tabulku. Dále k ShiftRows je inverzní transformací InvShiftRows. K MixColumns je to InvMixColumns s použitím inverzní mixovací matice než tomu bylo u transformace v šifrování. Mixovací matice pro dešifrování je uvedana v Tabulce 1. Poslední transformace InvAddRoundKey je inverzní sama k sobě, jedná se pouze o operaci XOR.

2.1 SubBytes – substituce bajtů

Jedná se o prostou nelineární bajtovou substituci, kde každému vstupnímu bajtu je přiřazena předem daná hodnota výstupního bajtu. Přiřazení se provádí dle známé tabulky. Každý bajt se rozdělí na dvě hexadecimální číslice. Pomocí první se určí řádek a pomocí druhé sloupec v tabulce. V dané buňce se pak přečte substituovaný bajt.

Vstupní blok dat, tedy 16 bajtů, rozdělíme na jednotlivé bajty, se kterými následně pracujeme. Lépe řečeno porovnáváme aktuální bajt se substituční tabulkou, kterou máme definovanou jako výběrové přiřazení. Deklarace přiřazení se skládá ze všech 256 možných kombinací vstupního bajtu. Jestliže vstupní blok dat je 16 bajtů, každý bajt se nahradí novou hodnotou určenou ze substituční tabulky. Operace znázorněna na Obrázku 2. Substituční tabulka S-Box je zobrazena na Obrázku 3.



Obrázek 2: Transformace SubBytes.

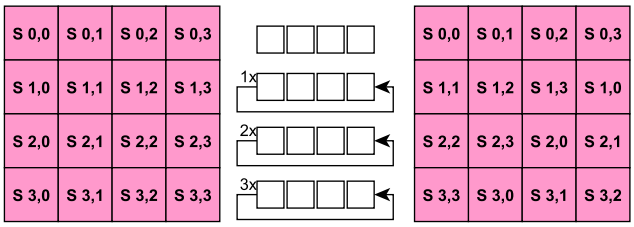
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc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

Obrázek 3: Substituční tabulka S-Box.

2.2 ShiftRows – rotace řádků

Při rotaci řádků se upravují jednotlivé řádky matice následujícím způsobem. V prvním řádku matice se neprovede řádná změna. Ve druhém řádku se provede rotace vlevo o jeden bajt, ve třetím řádku rotace vlevo o dva bajty a ve čtvrtém řádku se provede rotace vlevo o tři bajty. Transformace je znázorněna na Obrázku 4.

elektrorevue
ISSN 1213 - 1539

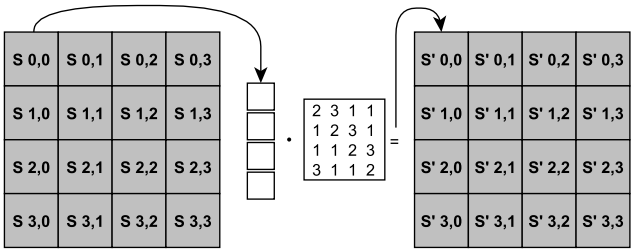


Obrázek 4: Transformace ShiftRows.

2.3 MixColumns – kombinace sloupců

Operace pracuje na každém sloupci jednotlivě. Jedná se o promíchání hodnot sloupce násobením v Galoisově poli GF(2^8) využívající generující polynom $g(x) = x^8 + x^4 + x^3 + x + 1$, princip viz Obrázek 5.

Transformace MixColumns vychází z vynásobení získané matice tzv. mixovací maticí viz Tabulka 1. Sloupec původní matice se násobí mixovací maticí, a vznikne tak sloupec nové transformované matice.



Obrázek 5: Transformace Mixcolumns.

Tabulka 1: Mixovací matice pro šifrování (vlevo) a dešifrování (vpravo).

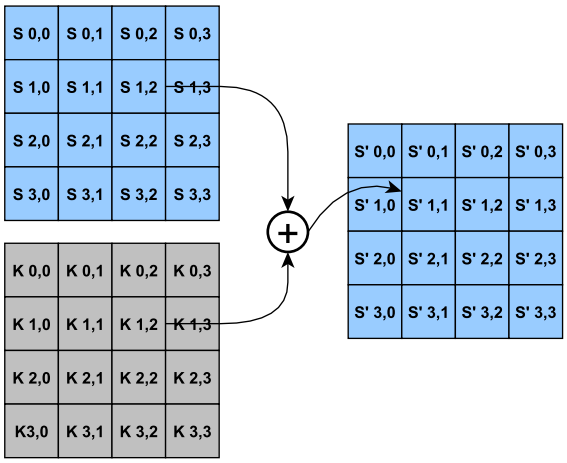
2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

14	11	13	9
9	14	11	13
13	9	14	11
11	13	9	14

Násobení jedničkou znamená ponechání původní hodnoty. Násobení dvojkou znamená posuv původní hodnoty o jeden bit vlevo. Násobení trojkou znamená posuv původní hodnoty o jeden bit vlevo a následné sečtení (XOR) s původní hodnotou [13].

2.4 AddRoundKey – přičtení rundovního klíče

Přičtení rundovního klíče je poslední transformací. Následná operace sečtení matice a klíče je prováděna pomocí exklusivního logického součtu XOR. Vstupní blok dat je stejně velký jako šifrovací klíč, tedy 128 bitů. Obě tyto hodnoty jsou známé, takže nezbývá než provést logickou operaci XOR, viz Obrázek 6.



Obrázek 6: Transformace Addroundkey.

2.5 Key Expansion – expanze klíče

Na počátku každého šifrování se provede expanze klíče (Key Expansion). Expanze klíče slouží k sestrojení klíčů z hlavního šifrovacího klíče na tzv. rundovní klíče (Round Keys). Získané rundovní klíče jsou použity v transformaci AddRoundKey, která se aplikuje při každé iteraci (rundě). Pokud je velikost klíče 128b je potřeba celkem 10 dílčích klíčů. Pro iniciační část se použije původní vstupní klíč a pro dalších 10 iterací se použijí nově vypočítané klíče. Počet iterací se odvíjí od délky šifrovacího klíče. Tabulka 2 prezentuje přesný počet iterací v závislosti na délce klíče.

Tabulka 2: Počtu iterací v závisloti na délce klíče.

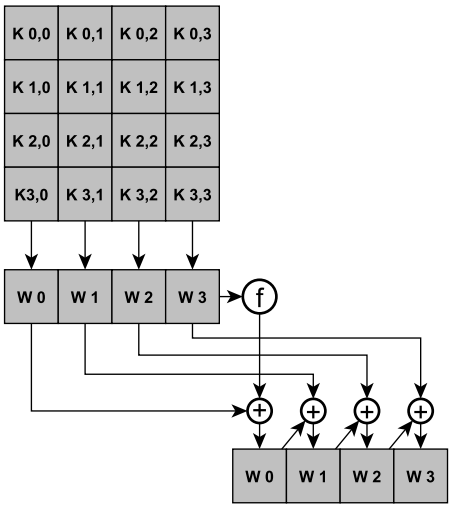
Délka klíče	128	192	256
Počet iterací	10	12	14

Hodnoty klíče jsou reprezentovány maticí 4×4 bajtů, podobně jako vstupní data algoritmu AES. Pro snazší pochopení expanzi klíče znázorňuje Obrázek 7. Operace realizuje expanzi klíče s délkou 16 bytů. Hodnoty sloupců chápeme jako prvky pole, které jsou označeny W (word). Klíč má 4 slova s délkou 4 byty. Získaná slova mají hodnoty indexu 0 až 4. Při počtu 10 rund je potřeba 44 slov. Sestrojení nového klíče probíhá z hodnot předchozího klíče.

Poslední slovo je podrobeno operaci funkce f, ze které vychází výpočet ostatních slov. Operaci tvořící funkci f znázorňuje Obrázek 8. Funkce f se skládá ze tří operací:

- RotWord – funkce rotuje sloupec směrem nahoru o 1 bajt,
- SubWord – funkce provede substituci SubBytes ze substituční tabulky (S-Box),
- XOR s Rcon – bajty sloupců se XORují s rundovní konstantou Rcon viz Tabulka 3.

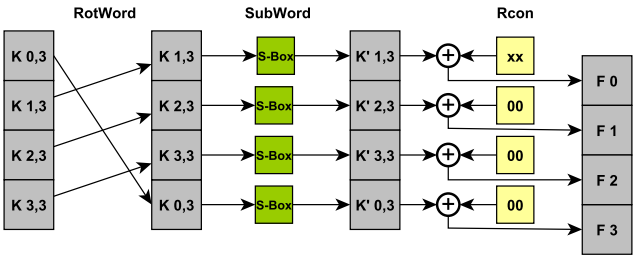
elektrorevue
ISSN 1213 - 1539



Obrázek 7: Expanze klíče.

Tabulka 3: Rundovní konstanty Rcon.

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00



Obrázek 8: Funkce f tvořící operaci expanze klíče.

Po provedení těchto tří operací, máme nové slovo označené F, které vstupuje dále do procesu viz předchozí Obrázek 7. Po ukončení výpočtu máme nový 16 bajtový klíč, který je určen pro jednu iteraci. Z tohoto klíče se stejným způsobem vypočítají další klíče s použitím následné konstanty Rcon. Na konci expanzi klíče máme k dispozici celkem 11 unikátních iteračních (rundovních) klíčů.

3 Vlastní implementace

Vlastní implementace je navržena pro šifrovací algoritmus AES v jazyce VHDL (Very High Speed Integrated Circuit Hardware Description Language) [13]. Funkční výstup je použit a otestován v FPGA síťové kartě, která vykonává šifrování. Jedná se o síťovou FPGA kartu COMBO-80G [14] od společnosti INVEA-TECH, umožňující vývoj hardwarově akcelerovaných aplikací. Karta je osazena výkonným FPGA čipem Virtex–7 firmy Xilinx. Podporuje technologie 40G a 10G Ethernet. Využívá sběrnici PCI Express pro vysokorychlostní přenosy dat mezi kartou a hostitelským počítačem.

Pro návrh je použito vývojové prostředí Vivado® od společnosti Xilinx. Implementace ve Vivado se skládá z logických a fyzických návrhů. Jde o převod ze zdrojového kódu programu na konfigurační soubor pro programovatelný logický obvod.

Zaměříme se na 4 základní bloky, které je třeba implementovat v jazyce VHDL. Dále na expanzi klíče a finální tvorbu firmwaru pro síťovou kartu. Za pomoci vytvořených testbenchů pro jednotlivé komponenty lze odsimulovat výše uvedené kroky algoritmu. Následně se sjednotily všechny komponenty do jednoho projektu Vivada a provedla se finální simulace. Ta představuje implementaci a ověření výsledků celého procesu šifrování.

Poslední fází je tvorba firmwaru. Ten je vytvořen za pomoci vývojového frameworku NetCOPE [15] od společnosti INVEA-TECH. Ve frameworku jsou implementované řadiče a rozhraní pro práci s periferiemi karet Combo. Framework NetCOPE se dělí na dvě části, kdy první částí je myšlen firmware určený pro FPGA kartu a druhou částí je software navržený pro hostitelským počítač. Komunikace mezi firmwarem na síťové kartě a softwarovou aplikací na počítači probíhá přes rozhraní PCI Express.

3.1 SubBytes

Vstupní blok dat je rozdělen na jednotlivé bajty, se kterými pracujeme. Porovnáváme aktuální bajt se substituční tabulkou, kterou máme definovanou jako výběrové přiřazení. Deklarace přiřazení se skládá ze všech 256 možných kombinací vstupního bajtu. Každý bajt se nahradí novou hodnotou určenou ze seznamu. Následující Kód 1 v jazyce VHDL vysvětluje zápis substituce s několika bajty z tabulky.

```
architecture behavior of SubTable is
begin
  with inp select
      outp <= x"63" when x"00",
              x"7c" when x"01",
              x"77" when x"02",
               ...
               ...
              x"54" when x"FD",
              x"BB" when x"FE",
              x"16" when others;
end architecture;

architecture structural of SubBytes is
begin
 gen: for i in 0 to 15 generate
 sub : entity work.SubTable port map(
           inp => vstup(8*i+7 downto 8*i),
           outp => vystup(8*i+7 downto 8*i));
 end generate gen;
end architecture;
```

Kód 1: SubBytes.

elektrorevue
ISSN 1213 - 1539

3.2 ShiftRows

Je důležité si uvědomit, že matici reprezentujeme vektorem. První 4 bajty tvoří první řádek, další 4 bajty druhý řádek atd. Pracujeme proto s indexem jednotlivých bitů, které transformujeme dle teoretických pravidel. Přesný postup je prezentován na ukázce Kódu 2.

```
architecture structural of ShiftRows is
begin
 TX(127downto96)<= RX(127downto96);
 TX(95downto64) <= RX(87downto64)&RX(95downto88);
 TX(63downto32) <= RX(47downto32)&RX(63downto48);
 TX(31downto0)  <= RX(7downto0)&RX(31downto8);
end structural;
```

Kód 2: ShiftRows.

3.3 MixColumns

Vstupem komponenty MixColumns je výstup bloku ShiftRows. V prvním kroku si vytvoříme 4 sloupce ze vstupní matice, které budeme transformovat. Každý sloupec můžeme považovat za jeden vstupní signál o velikosti 32 bitů, reprezentující sloupec původní matice. Uvádíme zde příklad výpočtu prvního bajtu nové matice. V tabulce 1 pro šifrování se jedná o první řádek, kdy násobíme první bajt dvojkou, druhý bajt trojkou a třetí i čtvrtý bajt jsou ponechány beze změny. Výsledek je získán exklusivním součtem nových bajtů.

```
input2 => column(31downto24), output2 => output2);
input3 => column(23downto16), output3 => output3);

result <= output2 xor output3 xor
    column(15to8) xor column(7to0);
```

Kód 3: Práce se sloupci v MixColumns.

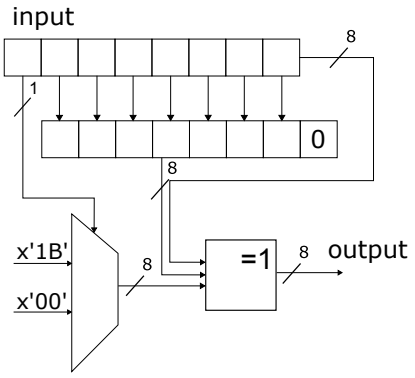
Pro násobení bajtu, je vytvořena entita, které předáme data a ona nám vrátí vypočtený výsledek. Vstup se násobí číslem 2 a 3, resp. 9, 11, 13, 14 u dešifrování. Způsoby, jak výpočet vyřešit, jsou dva. První způsob nazveme logický. Bajt vynásobíme požadovanou hodnotou a uložíme. U operace násobení je třeba dávat pozor na velikost výsledného bajtu. Jestliže vynásobíme bajt o velikosti větší než $7F_h$ dvěma resp. bajt o velikosti větší než 55_h třemi, vznikne nám číslo o devíti bitech. V tomto kroku je třeba postupovat podle [12] a bajt „xorovat“ s $1B_h$. Tato hodnota vychází z násobení v GF(2^8) viz 2.3.

V první řadě nás zajímá, zda při násobení překročíme hodnotu, kdy dojde k přetečení o jeden bit. Jestliže ano, s výsledkem je nutné provést operaci XOR s hodnotou $1B_h$. Násobení dvěma se provede posunutím bitových hodnot o jednu pozici vlevo a přidá se logická nula. Pro lepší pochopení je třeba uvést schéma na Obrázku 9, popisující jednotlivé kroky, a uvedený zdrojový Kód 4.

Druhá možnost, je zcela odlišná. Implementaci transformace MixColumns je možné vyřešit způsobem podobný tomu, jak je řešena transformace SubBytes. Tedy substitucí vstupních dat dle předem známých hodnot. Vstupní

```
architecture logic of MultiplyTwo is
 begin
 output <= input(6downto0)&'0' when input(7)='0'
      else input(6downto0)&'0' xor x"1B";
 end logic;
```

Kód 4: Operace násobení dvěma v MixColumns.



Obrázek 9: Schéma realizace násobení třemi [13].

bajt může nabývat 256 různých hodnot. Jestliže všechny tyto kombinace předem vynásobíme hodnotou, kterou chceme znát na výstupu, můžeme vytvořit substituční tabulku, podle které přiřadíme výstupu správnou hodnotu. Tento způsob můžeme pojmenovat lookup. Architektura obsahuje vyhledávací tabulku, ve které jsou všechny předem vypočítané hodnoty. V ukázkovém zdrojovém Kódu 5 je možné vidět, jakým způsobem jsou hodnoty interpretovány. Vstupem entity je bajtový signál input, kterému se přiřadí nová hodnota a uloží se do signálu output.

```
architecture lookup of MultiplyTwo is
 begin
    with input select
      output <= x"00" when x"00",
                x"02" when x"01",
                x"04" when x"02",
                  ...
                x"E1" when x"FD",
                x"E7" when x"FE",
                x"E5" when others;
 end lookup ;
```

Kód 5: Implementace sub. tab. násobení dvěma.

3.4 AddRoundKey

Poslední částí je přičtení šifrovacího klíče. Tento proces je proveden pomocí exkluzivní disjunkce (XOR). Vstupní blok dat je stejně velký jako šifrovací klíč, tedy 16 bajtů.

3.5 Syntéza kódu na firmware

V prvním kroku se navržené jednotlivé bloky transformací odsimulovaly v prostředí Vivado a odladily drobné chyby v návrhu. Cílem bylo simulovat testovanou entitu

elektrorevue
ISSN 1213 - 1539

```
RX  <= x"04E0482866CBF8068119D326E59A7A4C";
KEY <= x"A088232AFA54A36CFE2C397617B13905";

architecture structural of AddRoundKey is
begin
  TX <= RX xor KEY;
end structural;
```

Kód 6: Trasnformace AddRoundKey.

pro všechny možné kombinace a snímat výstupní signály. Po ověření správné funkčnosti algoritmu se přistoupilo k syntéze, tvorbě firmwaru.

Vývoj byl prováděn na syntézním serveru, kde za pomocí vývojového softwaru Vivado byl vytvořen firmware pro síťovou kartu. Pro vývoj byl použit NetCOPE verze 8.02.02 z května roku 2015, který je vhodně navržen pro práci s kartou COMBO-80G. Vivado na syntézním serveru je verze 2013.4 z prosince roku 2013. Všechny zdrojové kódy se vnoří do balíku NetCOPE a spustí se syntéza. Na výstupu celého procesu získáme firmware ve formátu mcs.

4 Testování na zařízení FPGA

Po dokončení syntézy je možné zjistit zabrané zdroje na čipu, díky kterým máme přehled o složitosti implementace. Nejčastěji se zabrané zdroje uvádí v hodnotách LUTs (Look Up Table). Jedná se o logické buňky (neboli paměťová místa) v zařízeních FPGA, které se využívají k vytvoření (popisu) logických funkcí, pomocí tzv. pravdivostní tabulky, kde jsou definovány výstupní hodnoty v závislosti na vstupních. Zabrané zdroje komponenty, která realizuje navržené šifrování, jsou uvedeny v Tabulce 4.

Tabulka 4: Zabrané zdroje komponenty.

Zdroje	Využito	Dostupné	%
LUTs	3192,7	432368	1

Lze vidět, že samotná operace není příliš náročná a nezabere velké množství čipu. Musí se ale počítat s celým návrhem včetně NetCOPE, který se stará o ovládání celé karty a dalších věcí (viz NetCOPE [15]). Tyto informace jsou uvedeny v Tabulce 5.

Tabulka 5: Zabrané zdroje celého firmwaru.

Zdroje	Využito	Dostupné	%
LUTs	132492	432368	31
LUT logiky	111102	432368	25
LUT paměti	21390	173992	12
Registry	115750	864736	13
Paměť	457	1470	31

Do FPGA karty se nahrává vytvořený firmware v souboru mcs. Po úspěšném nabootování programu je možné na fyzickém hardwarovém zařízení otestovat správnost a rychlost šifrování bloků dat. Data jsou posílána do karty z obsluhujícího uživatelského počítače. Data jsou uložena do paměti karty a následně šifrována nebo dešifrována, podle zvoleného firmwaru.

Rychlost šifrování dat je závislá na frekvenci jádra samotné karty. V našem případě je takt jádra nastaven na 100 MHz. Šifrování proběhne ve dvou hodinových taktech, proto uvažujeme s dobou výpočtu 20 ns. Za tuto dobu je zpracován jeden blok dat 128 bitů. Z toho je možné určit teoretickou rychlost zpracování. Jestliže zpracujeme za 20 ns 128 bitů dat, propustnost šifrování je 6 400 Mb/s.

Reálná rychlost je však nižší. Provedli jsme řadu testů, kdy jsme do registrů karty nahráli data a nechali zašifrovat. Data se nahrávala do vstupních registrů programu a po uložení posledního slova se spustilo šifrování. Výsledek je následující:

- 987 MB za 1,55 s ⇒ 5 094 Mbit/s,
- 4934 MB za 7,63 s ⇒ 5 173 Mbit/s,
- 9868 MB za 15,7 s ⇒ 5 028 Mbit/s.

Nepříliš rozdílná rychlost oproti teoretické rychlosti je způsobena přístupem k datům, která chceme šifrovat. Data jsou sice uložena na kartě, ale k procesu šifrování přistupují v cyklech, které řídí limitující časovač paměti. Tím vzniká velká prodleva mezi čtením a zápisem dat do registru. Dále je třeba uvažovat chybu v měření doby, kdy je šifrování spuštěno a kdy je u konce.

Shrnutí testování rychlosti šifrování je uvedeno v Tabulce č. 6. Průměrná rychlost šifrování je 5 100 Mbit/s.

Tabulka 6: Výsledek testu šifrování.

Test č.	data [MB]	výsledná rychlost [Mbit/s]
1	987	5 094
2	4934	5 173
3	9868	5 028

5 Závěr

Článek se zabývá popisem symetrické blokové šifry AES, dále pak konkrétním typem AES-128. Zaměřuje se na aplikaci algoritmu na hardware, konkrétně na FPGA technologii. V tomto článku jsou popsány jednotlivé bloky algoritmu, jak jsou navrženy pomocí programovacího jazyka VHDL. Zaměřili jsme se na implementaci šifry na síťovou FPGA kartu COMBO-80G osazenou čipem Virtex–7.

V poslední části prezentujeme výsledky testování rychlosti šifrování. Je zde uvedena reálná rychlost a diskutován rozdíl oproti teoretické rychlosti.

Poděkování

Výzkum byl podpořen projektem GAČR 14-25298P "Research into cryptographic primitives for secure authentication and digital identity protection" a Národním prgramem udržitelnosti LO1401.

elektrorevue
ISSN 1213 - 1539

Literatura

[1] BLUM, T., C. PAAR. *Montgomery modular exponentiation on reconfigurable hardware.* Proceedings 14th IEEE Symposium on Computer Arithmetic [online]. ISBN 0-7695-0116-8. DOI: 10.1109/ARITH.1999.762831. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=762831

[2] BLUM, T., C. PAAR. *High-radix Montgomery modular exponentiation on reconfigurable hardware.* IEEE Transactions on Computers [online]. 2001, roč. 50, č. 7, s. 759–764. ISBN 00189340. DOI: 10.1109/12.936241. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=936241

[3] DALY, A., W. MARNANE. *Efficient architectures for implementing montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic.* Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays [online]. New York, 2002, s. 40. ISBN 1581134525. DOI: 10.1145/503048.503055. Dostupné z: http://portal.acm.org/citation.cfm?doid=503048.503055

[4] MCIVOR, C., M. MCLOONE, J.V. MCCANNY. *Modified Montgomery modular multiplication and RSA exponentiation techniques.* IEE Proceedings - Computers and Digital Techniques [online]. 2004, roč. 151, č. 6, s. 402. ISSN 13502387. DOI: 10.1049/ip-cdt:20040791. Dostupné z: http://digital-library.theiet.org/content/journals/10.1049/ip-cdt_20040791

[5] AMANOR, D.N., C. PAAR, J. PELZL, V. BUNIMOV, M. SCHIMMLER. *Efficient hardware architectures for modular multiplication on FPGAs.* International Conference on Field Programmable Logic and Applications [online]. 2005, s. 539–542. ISBN 0-7803-9362-7. DOI: 10.1109/FPL.2005.1515780. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1515780

[6] CIET, M., M. NEVE, E. PEETERS, J.-J. QUISQUATER. *Parallel FPGA Implementation of RSA with Residue Number Systems.* Midwest Symposium on Circuits and Systems [online]. 2003, s. 806–810. ISBN 0-7803-8294-3. DOI: 10.1109/MWSCAS.2003.1562409. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1562409

[7] FOURNARIS, A. P. *Fault and simple power attack resistant RSA using Montgomery modular multiplication.* Proceedings of 2010 IEEE International Symposium on Circuits and Systems [online]. 2010, s. 1875–1878. ISBN 978-1-4244-5308-5. DOI: 10.1109/ISCAS.2010.5537879. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5537879

[8] ELBIRT, A.J., W. YIP, B. CHETWYND, C. PAAR. *An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists.* [online]. DOI: 10.1109/92.931230. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=931230

[9] ARRAG, S., A. HAMDOUN, A. TRAGHA, A. KHAMLICH. *Design and Implementation A different Architectures of mixcolumn in FPGA.* International Journal of VLSI Design & Communication Systems [online]. 2012. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1209/1209.3061.pdf

[10] WIEBE, J. H. *AES-128 Implementation on a Virtex-4 FPGA.* IEEE International Symposium on Signal Processing and Information Technology [online]. Giza, 2007, s. 68-73. ISBN 978-1-4244-1834-3. DOI: 10.1109/ISSPIT.2007.4458043. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4458043

[11] FIPS-197. *ADVANCED ENCRYPTION STANDARD (AES).* USA: NIST, 2001. Dostupné z: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[12] BURDA, K. *Aplikovaná kryptografie.* Praha: VUTIUM, 2013, 1. vyd., 255 s. ISBN 978-80-214-4612-0.

[13] SMÉKAL, D. *Zabezpečení vysokorychlostních komunikačních systémů*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015, 55 s.

[14] COMBO-80G FPGA karta. [online]. Dostupné z: https://www.invea.com/cs/produkty-sluzby/fpga-karty/combo-80g

[15] Vývojový framework NetCOPE. [online]. Dostupné z: https://www.invea.com/cs/produkty-sluzby/fpga-development-kit/netcope