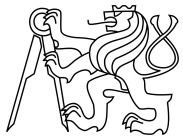
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Hardwarový modul pro šifrování dat v reálném čase
Student:	Daniel Hynek
Vedoucí:	Ing. Matěj Bartík
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Navrhněte a realizujte v jazyce VHDL funkční blok (IP Core) realizující šifrování a dešifrování kryptografického standardu AES (Advanced Encryption Standard). Toto IP bude podporovat všechny základní režimy (ECB, CBC, CFB, OFB, CTR). Dále bude podporovat šifrovací klíče o délce 128, 192 a 256 bitů.

Navrhněte způsoby, jak za běhu systému měnit režimy, které jsou v IP implementované a které zároveň toto IP využívají. Dále navrhněte IP tak, aby bez problémů pracovalo s datovými pakety Internet Protocol získané přes rozhraní Ethernet.

Požadovaná propustnost (pro různé varianty Ethernetu) je:
10G - Xilinx Virtex-6
40G - Xilinx Virtex-7
100G - Xilinx Virtex UltraScale
400G - Xilinx Virtex UltraScale

Pro splnění cíle závěrečné práce je nutné splnit alespoň jednu požadovanou propustnost. Navržené IP důkladně otestujte v simulaci.

Seznam odborné literatury

NIST FIPS-197 - Specification for the ADVANCED ENCRYPTION STANDARD (AES)
NIST Special Publication 800-38A - Recommendation for Block Cipher Modes of Operation

L.S.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 24. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra číslicového návrhu

Bakalářská práce

Hardwarový modul pro šifrování dat v reálném čase

Daniel Hynek

Vedoucí práce: Ing. Matěj Bartík

13. května 2016

Poděkování

Rád bych poděkoval Ing. Matěji Bartíkovi za vedení této práce a připomínky vedoucí k jejímu zlepšení. Dále bych chtěl poděkovat svým rodičům a sestře Světlaně za korekci textu a obrovskou podporu při psaní práce i během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 13. května 2016 .....

Odkaz na tuto práci

Hynek, Daniel. *Hardwarový modul pro šifrování dat v reálném čase.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce má za cíl představení šifry AES a její implementaci v jazyce VHDL pro obvod typu FPGA. Implementovaný modul nabízí běžné operační módy blokových šifer dle standardu *NIST Special Publication 800-38A*. Důraz je kladen na vysokou propustnost v operačních módech ECB a CTR. Propustnosti přesahující 30 Gbit/s je dosaženo za pomoci pipeliningu. Výsledný modul byl podrobený testování v post-translate simulaci pro obvod *Xilinx Virtex-6*.

Klíčová slova VHDL, FPGA, AES, realtime, operační módy blokových šifer, pipelining

Abstract

This bachelor thesis focuses on the AES and its implementation in VHDL language for FPGA. The implemented module offers usual block cipher modes of operation according to standard *NIST Special Publication 800-38A*. High speed is priority especialy in modes ECB and CTR. Speed exceeding 30 Gbit/s is achieved by using pipelining. The module was tested in post-translate simulation for *Xilinx Virtex-6*.

Keywords VHDL, FPGA, AES, realtime, block cipher modes of operation, pipelining

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Přenos dat po síti je dnes samozřejmostí. S množstvím dat, která po síti posíláme, přichází bezpečnostní rizika. Chceme-li si být jisti, že odeslaná data bude číst pouze určený příjemce, nikoliv potencionální útočník, je nutné data šifrovat.

Data nelze přenášet nezabezpečená kanálem, do kterého může útočník proniknout. Obvykle však nelze dokonale zabezpečit celý kanál, kterým data prochází. Vývoj sítí pokročil kupředu a přenosové rychlosti, které byly dříve nedosažitelné, jsou dnes samozřejmostí. Tento fakt jde částečně proti šifrování. Pokud potřebujeme data před odesláním zašifrovat a při příjmu zpětně dešifrovat, vzniká drobné zpoždění. Je tedy třeba používat šifry, které jsou rychlé a zároveň dostatečně bezpečné.

Potřeba šifrovat informace není pouze problémem novodobým. Podle historických zdrojů se první šifry objevily již ve starodávném Egyptě[1]. Velmi známou šifrou je pak šifra Caesarova, ve které jsou všechna písmena zašifrovaného textu posunutá o 3 pozice (A se změní na D, B na E,...).

Dnes se již tyto primitivní šifry k zabezpečení nevyužívají. Vývoj šifer neustále pokračuje s rostoucím výpočetním výkonem počítačů. Díky zvyšujícímu se výkonu a útokům hrubou silou[1] některé šifry přestaly být dostatečně bezpečné. Například DES (Data Encryption Standard) s 56-bitovým klíčem je tímto způsobem velice snadno prolomitelný[2]. AES (Advanced Encryption Standard), o kterém je tato práce, je v dnešní době považován za dostatečně odolný proti útoku hrubou silou. Jediná možnost útoku je tak prostřednictvím postranních kanálů[3]. Další budoucí hrozbou nejen pro AES, ale také další šifry je vývoj kvantových počítačů[4]. Snadným terčem výkonných kvantových počítačů by byly především asymetrický kryptosystém RSA (Rivest-Shamir-Adleman) a nebo algoritmus výměny klíčů Diffie-Hellman.

[1] Útok hrubou silou (brute-force) je typ útoku spočívající ve zkoušení všech možných klíčů. Končí úspěšně ve chvíli, kdy nenarazíme na správný klíč. Složitost takového útoku roste exponenciálně s délkou klíče. Brute-force útoky se obvykle využívají, pouze pokud není jiná možnost.

Kapitola 1

Cíl práce

Cílem této práce je analýza vlastností šifry AES (Advanced Encryption Standard), realizace a testování IP Core realizující tuto šifru. IP Core bude schopný data šifrovat i dešifrovat. Šifrování i dešifrování má volitelnou délku klíče 128, 192, nebo 256 bitů. Dále je popsáno využití základních operačních módů pro blokové šifry[5], a to ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback), OFB (Output Feedback) a CTR (Counter). Jednotlivé konfigurace je možné volit pomocí vstupů. Všechny módy, stejně jako šifrovací a dešifrovací modul, jsou popsané v jazyce VHDL (VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language) umožňujícím popis programovatelných hradlových polí FPGA (Field-Programmable Gate Array).

Modul je cílený na šifrování a dešifrování síťové komunikace pomocí FPGA obvodu. Požadované datové propustnosti jsou 10 Gbit/s pro Xilinx Virtex-6, 40 Gbit/s pro Xilinx Virtex-7, 100 Gbit/s pro Xilinx Virtex UltraScale a 400 Gbit/s pro Xilinx Virtex UltraScale+.

Kapitola 2

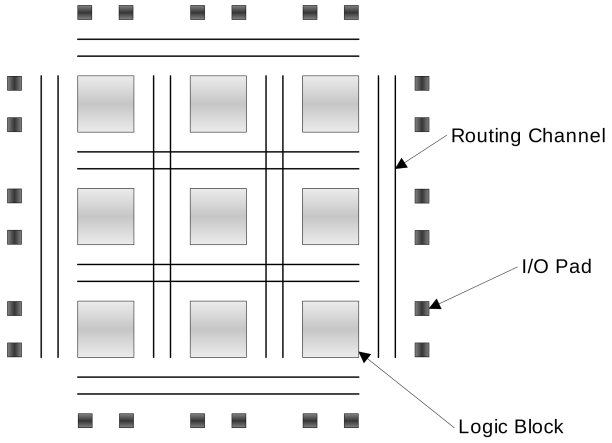
Analýza

2.1 FPGA obvody

FPGA je integrovaný obvod, který umožňuje dodatečné uživatelské naprogramování funkčnosti. Při výrobě tedy není určeno, jak se bude obvod chovat. Chování takového obvodu lze popsat pomocí libovolného HDL (Hardware Description Language) nebo vytvořit schéma obvodu přímo. Nejběžnějšími jazyky pro popis hardware jsou VHDL a Verilog.

2.1.1 Struktura FPGA

Obecně FPGA obvod obsahuje CLB (Configurable Logic Blocks) ve 2D matici, které lze propojovat přepínatelnými spoji. Tato struktura je znázorněna na Obrázku 2.1. Bloky jsou složeny z menších jednotek, takzvaných slice.



Obrázek 2.1: Struktura FPGA. Převzato z [6]

Každý slice obsahuje určitý počet LUT (Look-Up Table), klopných obvodů (flip-flop) a multiplexerů. LUT je několikavstupové hradlo, které lze napro-

gramovat a zrealizovat tak kombinační funkci. Funkce LUT je realizovaná pomocí vnitřní paměti SRAM (Static Random Access Memory). Flip-flop slouží jako paměť výstupů z LUT. Multiplexer zase vybírá mezi výstupy z LUT. Programovat lze i to, které bloky budou propojené a které nikoli. Výsledkem je, že máme propojené přesně ty bloky, které potřebujeme pro vytvoření komplexnější funkce. Dále je běžné, že slice obsahují další paměti, bloky pro řízení hodinového kmitočtu, signály pro urychlení přenosu aritmetických operací a další prvky, které jsou již specifické pro každý čip[7].

2.1.2 Xilinx FPGA

Společnost Xilinx patří mezi největší výrobce FPGA. Vyrábí tři různé modelové řady FPGA obvodů 7. generace (pro tuto práci využívám 6. generaci řady Virtex). Tyto řady jsou rozdělené podle zaměření. Nejnižší řadou je Artix, který nabízí nejpříznivější cenu a vzhledem k menšímu počtu zdrojů také nejnižší spotřebu. Pokud požadujeme vysoký výkon, je vhodné využít řadu Virtex. Kompromisem je pak Kintex, který leží mezi Artixem a Virtexem z hlediska výkonu i spotřeby. Za vyšší ceny jsou pak modely Virtex a Kintex ve verzích UltraSCALE a UltraSCALE+ s vyšší mírou integrace 20 nm, resp. 16 nm oproti běžným 28 nm[8][9][10].

2.1.3 Xilinx Virtex-6

Virtex-6 obsahuje LUT, které mohou být konfigurovatelné jako šestivstupové, nebo dva pětivstupové. Výstup každého LUT může být uložen do flip-flopu. Slice je potom složený ze čtyř LUT, kde každý LUT má dva flip-flopy, multiplexer a logiku pro zrychlení přenosu aritmetických operací.

Tabulka 2.1: Porovnání konfigurací Virtex-6[11]

Device	Logic cells	Slices	Max user I/0
XC6VLX75T	74 496	11 640	360
XC6VLX130T	128 000	20 000	600
XC6VLX195T	199 680	31 200	600
XC6VLX240T	241 152	37 680	720
XC6VLX365T	364 032	56 880	720
XC6VLX550T	549 888	85 920	1 200
XC6VLX760	758 784	118 560	1 200
XC6VSX315T	314 880	49 200	720
XC6VSX475T	476 160	74 400	840
XC6VHX250T	251 904	39 360	320
XC6VHX255T	253 440	39 600	480
XC6VHX380T	382 464	59 760	720
XC6VHX565T	566 784	88 560	720

Virtex-6 existuje ve více než deseti různých konfiguracích. Tabulka 2.1 porovnává jednotlivé modely. Je patrné, že rozdíly parametrů jsou velké, a tak i v rámci jedné řady je široká možnost volby.

2.2 Jazyk VHDL

VHDL (VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language) je striktně typový jazyk pro popis hardwaru vycházející z programovacího jazyka Ada. Ada i VHDL byly vyvinuty pro americkou armádou, na žádost *U.S Department of Defense*. Naopak druhý velmi používaný jazyk pro popis hardwaru (Verilog) pochází od společnosti *Automated Integrated Design Systems*. Verilog složil původně pouze pro interní potřeby společnosti. Vychází z programovacího jazyka C. Kód Verilogu je stručnější, ale nenabízí tak široké možnosti využití jako jazyk VHDL[12]. Ukázka několika rozdílů v syntaxi VHDL a Verilogu je zobrazena v Listing 1.

Listing 1 Ukázka příkladu popisu synchronního flip-flopu s resetem ve VHDL(nahoře) a Verilogu(dole) pro porovnání základní syntaxe.

```
--VHDL
--DFF se synchronním resetem

DFlipFlop : PROCESS( clk )
BEGIN
IF RISING_EDGE(clk) THEN
        IF reset = '1' THEN
                q <= '0';
        ELSE
                q <= in;
        END IF;
END IF;
END PROCESS DFlipFlop;
```

```
//Verilog
//DFF se synchronním resetem

always @ ( posedge clk )
if ( ~reset ) begin
        q <= 1'b0;
end
else begin
        q <= data;
end
```

VHDL má velmi široké možnosti využití. Lze jej využít jak pro popisování designu, tak pro testování. V případě, že chceme daný kód syntetizovat,

je nutné využívat pouze omezenou množinu jazyka, kterou syntetizovat lze. Záleží také na syntézním nástroji, které konstrukce je schopný syntetizovat a které nikoli.

2.3 Algoritmus AES

AES je bloková symetrická šifra. Tedy data jsou šifrována po blocích, vždy stejnou transformací[13]. V případě AES jde o 128-bitové bloky. Na jeden šifrovací cyklus je zašifrováno celých 128 vstupních bitů najednou. Že je šifra symetrická pak znamená použití pouze jednoho klíče pro šifrování i dešifrování. Šifra AES je standardizována v *Advanced Encryption Standard (AES) (FIPS PUB 197)*[14].

Jedná se o nástupce šifry DES (Data Encryption Standard), resp. 3DES (Triple Data Encryption Standard), která byla nahrazena z důvodu nedostatečné bezpečnosti. AES byl vybrán institutem NIST (National Institute of Standards and Technology) z několika kandidátů veřejného výběru. Vítězem se stala šifra Rijndael, kterou navrhl Vincent Rijmen a Joan Daemen. AES je její podmnožinou. Oproti AES nabízí Rijdael možnost využít bloky i klíče o velikosti násobku 32 bitů v rozmezí 128 až 256 bitů[15].

AES je založený na rundovních operacích. Počet rund N_r je 10, 12, resp. 14 v závislosti na délce klíče, který má 128, 192, nebo 256 bitů.

2.3.1 Popis šifrování

Před vstupem do první rundy proběhne přidání rundovního klíče na vstupní vektor funkcí XOR (eXclusive OR). Výstup funkce XOR je vstupem do první rundy. V následujících $N_r - 1$ rundách dojde k postupnému provedení operací *SubBytes*, *ShiftRows*, *MixColumns* a *AddRoundKey*. V rundě N_r proběhne pouze *SubBytes*, *ShiftRows* a *AddRoundKey*. Funkce *MixColumns* je vynechána. Výstup z rundy $N_r - 1$ je vždy vstupem pro rundu N_r. Výstup z poslední rundy je výsledkem šifrování.

Algorithm 1 Šifrování

```
function ENC( byte in[ 4N_b ], byte out[ 4N_b ], word w[ N_b( N_r + 1 ) ] )
    byte state[4, N_b] ← in
     ADDROUNDKEY ( state, w[0, N_b − 1] )
    for round ← 1 to N_r − 1 do
        SUBBYTES ( state )
        SHIFTROWS ( state )
        MIXCOLUMNS ( state )
        ADDROUNDKEY ( state, w[ round N_b, (round+1) N_b − 1] )
    end for
     SUBBYTES ( state )
     SHIFTROWS ( state )
     ADDROUNDKEY ( state, w[ N_r N_b, (N_r + 1) N_b − 1] )
    out ← state
end function
```

2.3.2 Popis dešifrování

Dešifrování probíhá v inverzním pořadí. Nejprve je přidán poslední rundovní klíč na vstupní vektror funkcí XOR. Výstup funkce XOR je vstupem do první rundy. V každé z $N_r - 1$ rund dojde k provedení operací *ShiftRowsInverse*, *SubBytesInverse*, *AddRoundKey* a *MixColumnsInverse*. V rundě N_r proběhne pouze *ShiftRowsInverse*, *SubBytesInverse* a *AddRoundKey*. Chybí *MixColumnsInverse*, jehož inverze chyběla i v šifrování. Oproti šifrování je změněné pořadí operace *AddRoundKey* a *MixColumnsInverse*. Tato změna umožňuje ponechat stejnou expanzi klíče jako u šifrování.

Algorithm 2 Dešifrování

```
function DEC( byte in[ 4N_b ], byte out[ 4N_b ], word w[ N_b( N_r + 1 ) ] )
    byte state[4, N_b] ← in
     ADDROUNDKEY ( state, w[ N_r N_b, (N_r + 1) N_b − 1] )
    for round ← N_r − 1 downto 1 do
        INVSUBBYTES ( state )
        INVSHIFTROWS ( state )
        ADDROUNDKEY ( state, w[ round N_b, (round+1) N_b − 1] )
        INVMIXCOLUMNS ( state )
    end for
     INVSHIFTROWS ( state )
     INVSUBBYTES ( state )
     ADDROUNDKEY ( state, w[0, N_b − 1] )
    out ← state
end function
```

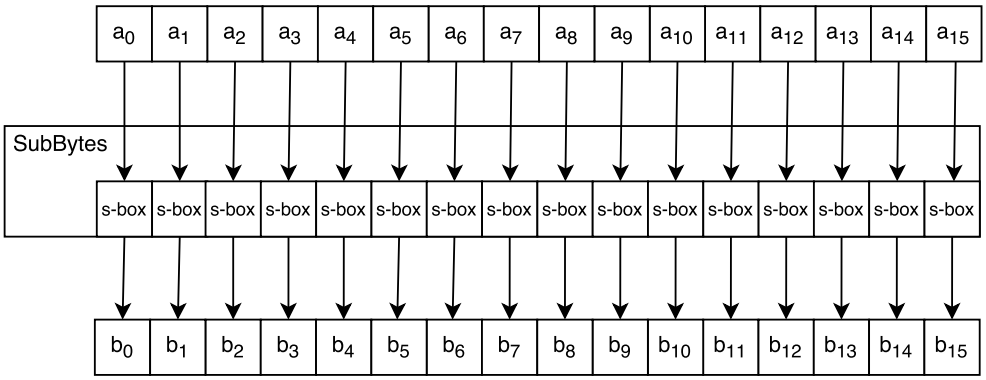
2.3.3 Ekvivalentní inverzní šifra

Díky tomu, že *MixColumns* a *MixColumnsInverse* jsou lineární s ohledem na sloupce vstupu, lze pořadí těchto operací zaměnit. Výsledkem je dešifrování pomocí totožného pořadí inverzních funkcí. Tento přístup má tu nevýhodu, že je nutné změnit expanzi klíče. Nelze tedy pro dešifrování využít klíč expandovaný pro šifrování.

2.3.4 Rundovní funkce

- **AddRoundKey**
  Provede přixorování (provedení funkce XOR) příslušného rundovního klíče ke vstupu.

- **Funkce SubBytes**
  Skládá se ze šestnácti s-boxů (Obrázek 2.2). Nahradí všechny byty vstupního vektoru za jiné. Náhrady jsou definované v s-boxech Obrázek 2.3.

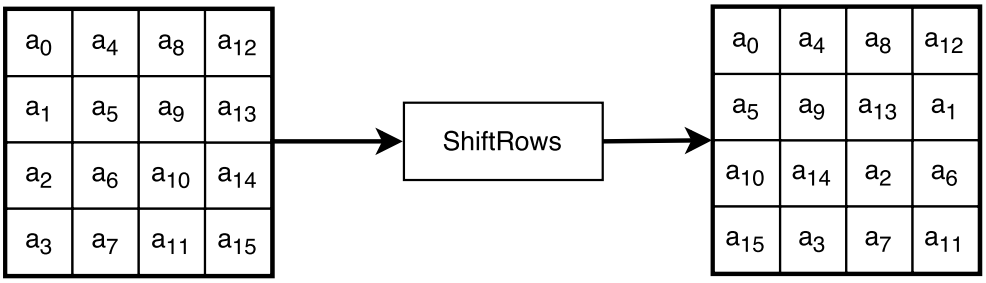


Obrázek 2.2: SubBytes

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

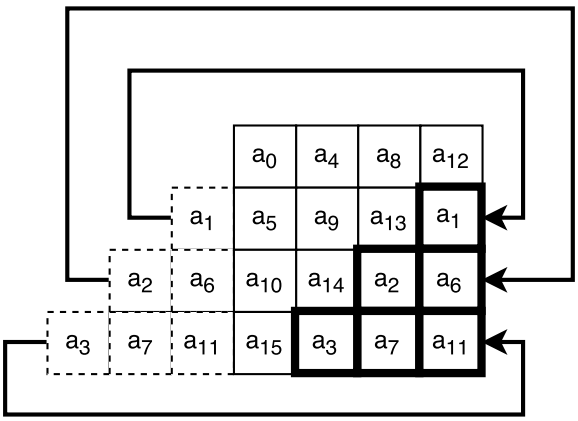
Obrázek 2.3: S-box tabulka. Převzato z [14]

- **Funkce ShiftRows**
  Provede promíchání bytů cyklickým posunutím řádků matice vektoru o offset odpovídající indexu řádku[2] doleva.



Obrázek 2.4: Funkce ShiftRows

[2]Řádky jsou indexovány od nuly.



Obrázek 2.5: Schéma posunu ShiftRows

- **Funkce MixColumns**
  Jedná se o sloupcovou operaci. Jednotlivé sloupce vezme jako polynomy z Galoisova tělesa GF(2^8) a vynásobí je modulo $x^4 + 1$ polynomem a(x) = $\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$. Tuto operaci lze interpretovat také jako násobení maticí **M**.

$$\mathbf{M} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

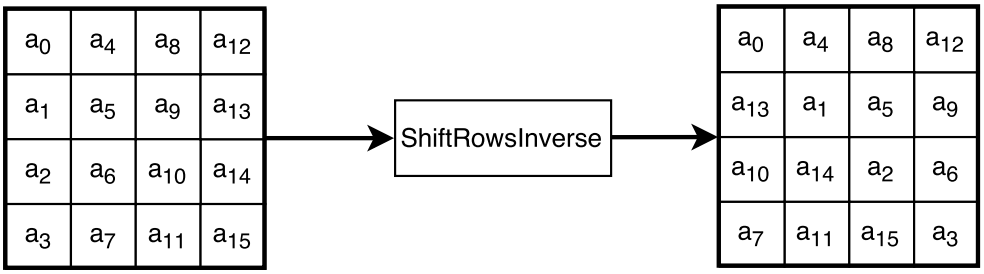
$$\begin{bmatrix} b_{4i} \\ b_{4i+1} \\ b_{4i+2} \\ b_{4i+3} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{4i} \\ a_{4i+1} \\ a_{4i+2} \\ a_{4i+3} \end{bmatrix}, \text{ kde } 0 \leq i \leq 3$$

- **Funkce SubBytesInverse**
  Funkce je totožná s *SubBytes*. Jediná změna je ve využití inverzních s-boxů místo klasických. Náhrady jsou tedy definované v inverzních s-boxech. 2.6

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

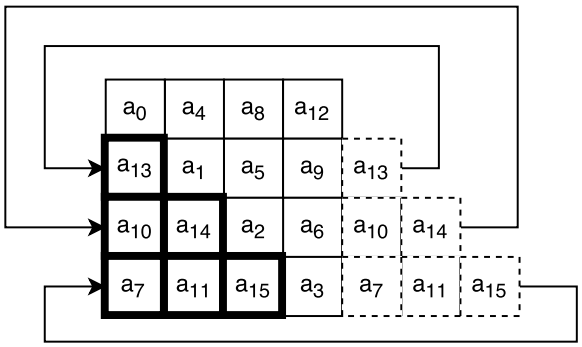
Obrázek 2.6: Inverzní s-box tabulka. Převzato z [14]

- **Funkce ShiftRowsInverse**
  Vrátí promíchání bytů opačným cyklickým posunutím řádků matice vektoru. Posun je tedy o offset odpovídající indexu řádku[3] doprava.



Obrázek 2.7: Funkce ShiftRowsInverse

[3]Řádky jsou indexovány od nuly.



Obrázek 2.8: Schéma posunu ShiftRowsInverse

- **MixColumnsInverse**
  Jedná se o sloupcovou operaci. Jednotlivé sloupce vezme jako polynomy v GF(2^8) a vynásobí je modulo $x^4 + 1$ polynomem $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$. Tuto operaci lze interpretovat také jako násobení maticí **N**.

$$\mathbf{N} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

$$\begin{bmatrix} b_{4i} \\ b_{4i+1} \\ b_{4i+2} \\ b_{4i+3} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} a_{4i} \\ a_{4i+1} \\ a_{4i+2} \\ a_{4i+3} \end{bmatrix}, \text{ kde } 0 \leq i \leq 3$$

2.3.5 Expanze klíče

Klíče aplikované v jednotlivých rundách pomocí funkce *AddRoundKey* se neopakují. Pokud označíme počet rund N_r, pak je pro šifrování i dešifrování použito N_r+1 128-bitových klíčů. Vzhledem k tomu, že klíč ze vstupu je nanejvýš 256-bitový, je nutné jej expandovat na požadovanou délku. První klíč je aplikovaný přímo na vstupní data a zbylé klíče jsou použity v N_r šifrovacích rundách, vždy na konci rundy po proběhnutí poslední transformace.

Šifrovací klíč je rozdělen na 32-bitové části w_i. Na vstupu máme prvních N_k částí, kde N_k je 4, 6, nebo 8 v závislosti na délce klíče. Části w_0 až w_{N_k-1} jsou zkopírovány ze vstupního klíče, další části se generují podle Algorithm 3. Generování klíče je shodné pro AES-128 (AES využívající 128-bitový klíč)

a AES-192 (AES využívající 192-bitový klíč). V případě AES-256 (AES využívající 256-bitový klíč) se liší. Rozdíl je v částech klíče, kde $i-4$ je násobkem 8, pro část w_i.

Funkce používané při expanzi klíče

- SubWord nahradí každý ze čtyř bytů za jiný pomocí s-boxů Obrázek 2.3.
- RotWord provede cyklický posun doleva, slovo [a_0, a_1, a_2, a_3] se změní na slovo [a_1, a_2, a_3, a_0], kde a_i je jeden byte.
- Rcon je rundovní konstanta definovaná jako prvek GF(2^8) s hodnotou x^{i-1}, kde x je {02}.

  RC(1) = 0x01
  RC(2) = 0x02
  RC(3) = 0x04
  ...
  RC(i) = RC[i−1]·0x02 ▷ Nedojde-li k přenosu
  RC(i) = (RC[i−1]·0x02) ⊕ 0x1b [4] ▷ Došlo k přenosu
  Rcon(i) = [RC(i), {00}, {00}, {00}]

Algorithm 3 Pseudokód expanze klíče

```
function KeyExpansion( byte key[ 4N_k ], word w[ N_b( N_r + 1 ) ], N_k )
    word temp
    i ← 0
    while i < N_k do
        w[ i ] ← word( key[ 4i ], key[ 4i + 1 ], key[ 4i + 2 ], key[ 4i + 3 ] )
        i ← i + 1
    end while
    i ← N_k
    while i < N_b(N_r + 1) do
        temp ← w[i − 1]
        if i mod N_k = 0 then
            temp ← SubWord ( RotWord (temp)) ⊕ Rcon (i/N_k)
        else if N_k > 6 and (i mod N_k = 4) then
            temp ← SubWord (temp)
        end if
        w[i] ← w[ i - N_k ] ⊕ temp
        i ← i + 1
    end while
end function
```

[4] Hodnota 0x1b je ireducibilní polynom v GF(2^8).

Při využití ekvivalentní inverzní šifry je nutné na konci expanze klíče použít funkci *MixColumnsInverse* na všechny rundovní klíče kromě prvních 128 bitů. Jedná se o jedinou změnu v plánování klíče.

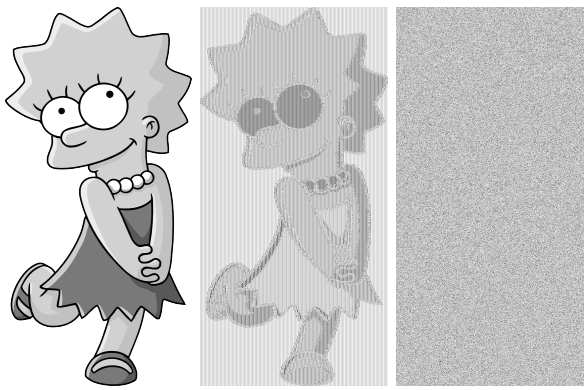
2.4 Operační módy blokových šifer

Operační módy spočívají v úpravě vstupních bloků (PB), nebo výstupních bloků (CB) tak, aby nebyl při dvou shodných blocích na vstupu stejný výstup. Dochází tedy k určité provázanosti šifrování jednotlivých bloků.

Použití operačních módů může přinést i nějaká omezení. Prvním je, že v módech CBC, CFB a OFB je nutné čekat na ukončení šifrování jednoho bloku, aby mohlo začít šifrování bloku následujícího. Dále jsou pak kladeny vyšší požadavky na kvalitu přenosu zašifrovaných bloků. Ztráta, případně chyba v přenosu jednoho bloku může mít vliv na ostatní bloky dat.

Mezi základní operační módy podle standardu *NIST Special Publication 800-38A*[5] patří ECB (Electronic Codebook), CBC (Cipher Block Chaining), CFB (Cipher Feedback), OFB (Output Feedback) a CTR (Counter). V praxi nejpoužívanějšími módy jsou ECB a CBC[16].

Pokud šifrujeme větší množství dat, je použití módu ECB nežádoucí a mělo by být samozřejmostí využití některého pokročilejšího módu. Z dat zašifrovaných pomocí ECB lze poznat strukturu dat před šifrováním. Data jsou tedy zašifrovaná, ale prozrazují více informací, než by bylo žádoucí. Problematika použití ECB je znázorněna na Obrázku 2.9



Obrázek 2.9: Ukázka, jak dopadne zašifrování datového obsahu obrázku ve formátu BMP za použití různých operačních módů. Postupně zleva nezašifrováno, zašifrováno pomocí AES-128 a operačního módu ECB, zašifrováno pomocí AES-128 a operačního módu CBC. Originální obrázek je převzatý z [17].

2.4.1 ECB (electronic codebook mode)

Každý blok dat je šifrovaný nebo dešifrovaný bez ohledu na ostatní bloky. Výsledkem je, že dva bloky shodné před zašifrováním, budou shodné i po zašifrování. ECB je nejjednodušší operační mód, kde díky nezávislosti bloků lze snadno paralelizovat šifrování i dešifrování. Chyba v přenosu či ztráta dat ovlivní pouze postižený blok, nemá žádný vliv na ostatní bloky.

- **Šifrování**
  $\mathrm{CB}_i = \mathrm{ENC}_{key}(\mathrm{PB}_i)$
- **Dešifrování**
  $\mathrm{PB}_i = \mathrm{DEC}_{key}(\mathrm{CB}_i)$

2.4.2 CBC (cipher block chaining)

Při použití módu CBC je šifrování řetězeno, není tedy možné jej paralelizovat. Dešifrování paralelizovat lze. Chyba v libovolném bloku vstupu znamená chybu v daném bloku a bloku následujícím. Při ztrátě některého bloku přijdeme o daný blok a blok následující.

- **Šifrování**
  $\mathrm{CB}_0 = \mathrm{ENC}_{key}(\mathrm{PB}_0 \oplus \mathrm{IV}$[5]$)$
  $\mathrm{CB}_i = \mathrm{ENC}_{key}(\mathrm{PB}_i \oplus \mathrm{CB}_{i-1})$ pro i = 1 .. n
- **Dešifrování**
  $\mathrm{PB}_0 = \mathrm{DEC}_{key}(\mathrm{CB}_0) \oplus \mathrm{IV}$
  $\mathrm{PB}_i = \mathrm{DEC}_{key}(\mathrm{CB}_i) \oplus \mathrm{CB}_{i-1}$ pro i = 1 .. n

2.4.3 CFB (cipher feedback)

CFB umožňuje zmenšit velikost bloku pouze na segment s bitů, $1 \leq s \leq b$, kde b je maximální velikost bloku šifry. Chyba v libovolném bloku vstupu znamená chybu v daném bloku a bloku následujícím. Při ztrátě některého bloku přijdeme o daný blok a blok následující.

- **Šifrování**
  $\mathrm{I}_0 = \mathrm{IV}$
  $\mathrm{I}_j = \mathrm{LSB}^{6}_{b-s}(\mathrm{I}_{j-1}) | CB^{\#}_{j-1}$[7]
  $\mathrm{O}_j = \mathrm{ENC}_{key}(\mathrm{I}_j)$
  $CB^{\#}_j = PB^{\#}_j \oplus MSB^{8}_s(O_j)$

[5] Zkratka IV označuje inicializační vektor.
[6] Funkce $\mathrm{LSB}_n()$ vrátí n nejméně významných bitů
[7] $CB^{\#}_i$ a $PB^{\#}_i$ je šifrovaný, resp. nešifrovaný segment.
[8] Funkce $\mathrm{MSB}_n()$ vrátí n nejvíce významných bitů

- **Dešifrování**
  $I_0 = IV$
  $I_j = LSB_{b-s}(I_{j-1})|CB^{\#}_{j-1}$
  $O_j = ENC_{key}(I_j)$
  $PB^{\#}_j = CB^{\#}_j \oplus MSB_s(O_j)$

2.4.4 OFB (output feedback)

Při použití módu OFB se na vstupní bloky přixoruje stream vygenerovaný opakovaným šifrováním inicializačního vektoru. Z blokové šifry se tak stává proudová. Chyba v libovolném bloku vstupu znamená pouze chybu v daném bloku výstupu. Při ztrátě některého bloku jsou všechny následující bloky chybné.

- **Šifrování**
  $I_0 = IV$
  $I_i = O_{i-1}$
  $O_i = ENC_{key}(I_i)$
  $CB_i = PB_i \oplus O_i$ pro i = 1 .. n−1
  $CB_n = PB_n \oplus MSB_u(O_n)$

- **Dešifrování**
  $I_0 = IV$
  $I_i = O_{i-1}$
  $O_i = ENC_{key}(I_i)$
  $PB_i = CB_i \oplus O_i$ pro i = 1 .. n−1
  $PB_n = CB_n \oplus MSB_u(O_n)$

2.4.5 CTR (counter)

Pomocí šifrovací transformace je zašifrován obsah čítače (T_i), výstup šifrování je přixorován na vstupní blok. Z blokové šifry se tak stává proudová. Poslední blok nemusí být kompletní. V případě šifrování i dešifrování se používá pouze šifrovací funkce. Lze paralelizovat generování streamu. Chyba v libovolném bloku vstupu znamená pouze chybu v daném bloku výstupu. Při ztrátě některého bloku jsou všechny následující bloky chybné.

- **Šifrování**
  $O_i = ENC_{key}(T_i)$
  $CB_i = PB_i \oplus O_i$
  $CB_n = PB_n \oplus MSB_u(O_n)$

- **Dešifrování**
  $O_i = ENC_{key}(T_i)$
  $PB_i = CB_i \oplus O_i$
  $PB_n = CB_n \oplus MSB_u(O_n)$

Z uvedených vlastností jednotlivých módů vyplývá, že nejvhodnějšími módy pro maximalizaci propustnosti jsou módy ECB a CTR. Mód ECB je sice nejsnazší pro implementaci, ale není dostatečně bezpečné. Jako bezpečnější alternativa tak poslouží implementovaný mód CTR.

Tabulka 2.2: Další vlastnosi základních opeačních módů

	Padding posledního bloku	Paralelizace širování/dešifrování	Využití dešifrovací funkce
ECB	✓	✓/✓	✓
CBC	✓	✗/✓	✓
CFB	✓	✗/✓	✗
OFB	✗	✗/✗	✗
CTR	✗	✓/✓	✗

Kapitola 3

Návrh a realizace

Hlavní modul se skládá ze dvou komponent popsaných v jazyce VHDL. První komponenta slouží pro šifrování a druhá pak pro dešifrování. Obě komponenty využívají předdefinované rundovní funkce vytvořené na základu šifrovacích funkcí pro AES-128 poskytnutých vedoucím této práce Ing. Matějem Bartíkem[18]. Bylo však potřeba předělat samotné šifrování, aby bylo možné rozšiřování funkčnosti. Funkce, u kterých není potřeba inverzní chování, jsou sdílené pro obě komponenty.

3.1 Popis chování

Z důvodu snazší synchronizace je chování šifrovacího a dešifrovacího modulu velice podobné. Vždy dochází k celé expanzi klíče před samotným zahájením šifrování nebo dešifrování. Toto chování není pro šifrování nezbytně nutné, ale zjednodušuje návrh, protože není nutné hlídat, jaká část klíče je již připravena, nebo čekat na její přípravu. Dešifrování z důvodu, že využívá nejprve naposledy expandovanou část klíče, musí mít již proběhlou expanzi celou.

Oddělená expanze klíče znamená také možnost zvýšení propustnosti. Není nutné expandovat klíč během procesu samotného šifrování, což zbytečně nezvyšuje délku šifrování. Je zde však prodleva před samotným započetím šifrování nebo dešifrování prvního bloku, než je klíč expandován. Dále je použita pro dešifrování Ekvivalentní inverzní šifra popsaná v kapitole 2.3.3. Expandovaný klíč tedy není stejný jako při šifrování.

Vždy jsou posílány vstupy jak do šifrovací komponenty, tak do dešifrovací komponenty. Výstup z komponent je vybírán podle toho, zda je zvoleno šifrování nebo dešifrování a dále zpracováván v souladu s vybraným operačním módem.

3.2 Propustnost

Šifra AES je složená z 10, 12, nebo 14 rund v závislosti na zvolené délce klíče. Rundy na sebe navazují, paralelní zpracování jednotlivých rund tedy není možné[19]. Lze však poměrně efektivně využít pipelining, který díky dostatečnému množství zdrojů na FPGA není problém implementovat. Obecný princip pipeliningu je popsán v kapitole 3.3. V případě potřeby následného navyšování propustnosti je zde možnost použít více šifrovacích a dešifrovacích komponent.

Díky implementovanému pipeliningu je možné dosáhnout požadované propustnosti jak pro šifrování, tak pro dešifrování. Při pipeliningu nemá na propustnost ani vliv to, zda zpracováváme 10, nebo 14 rund. Jediný projev vyššího počtu rund je v delší době do výstupu prvního bloku. Další bloky poté mohou následovat v každém dalším taktu.

3.2.1 Operační módy

Operační módy znamenají další vrstvu nad šifrováním AES. Z operačních módů popsaných v kapitole 2.4 nebrání použití pipeliningu pouze módy ECB a CTR, které nemají závislost jednoho vstupního bloku na výstupu předchozího. Módy CBC, CFB a OFB pro šifrování bloku potřebují vždy dokončené předchozí šifrování, vyjma prvního bloku. Tato serializace na úrovni módů znemožňuje využít pipelining pro šifrování u těchto módů.

Propustnost s operačními módy ECB a CTR je tedy shodná s propustností samotných komponent. ECB je mód, který nijak nemodifikuje ani vstup do komponent ani jejich výstup, nebude z hlediska propustnosti problematický. U módu CTR je pouze nutné zapamatování vstupu, aby bylo možné po zašifrování čítače přičíst výstup šifrování pomocí funkce XOR na blok dat, který byl na vstupu o několik taktů dříve.

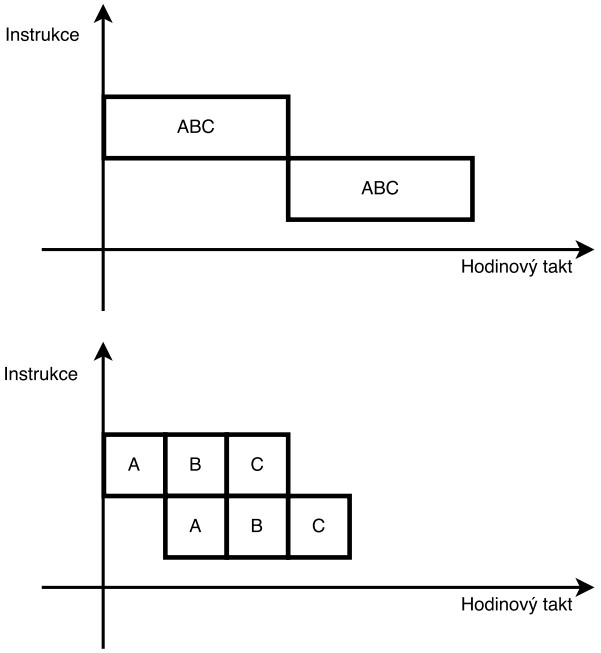
Mód CFB umožňuje zmenšení velikosti bloku šifry způsobem uvedeným v kapitole 2.4.3. Tato možnost není po dohodě s vedoucím práce implementovaná, protože by zbytečně snižovala propustnost. V případě potřeby využití tohoto módu spolu s menšími bloky bude nutné tyto bloky skládat stejně jako u ostatních módů externí logikou.

Z důvodu nevyužitelnosti paralelního zpracování většinou módů není použita duplikace šifrovacích a dešifrovacích komponent.

3.3 Pipelining

Pipelining je technika umožňující zvýšení frekvence obvodu. Je hojně využíván tam, kde by jinak bylo mezi vstupem a výstupem velké množství samotné kombinační logiky. Každé logické hradlo, každý prvek obvodu má nějaké drobné zpoždění. Pokud máme za sebou zřetězené stovky hradel, zpoždění narůstá. Frekvence obvodu pak může být jenom tak rychlá, jak rychlá je nejpomalejší

část (kritická cesta). Aby nebylo nutné čekat s hodinovým taktem na doběhnutí kritické cesty využívá se rozdělení na menší bloky (stage). Každá stage má na svém konci registr, do kterého se uloží výsledek a v dalším taktu pokračuje dál do následující stage. Rozdělení umožní vkládat na vstup s každým taktem nová data a zároveň umožní zvýšení hodinového taktu. Data na výstup se dostanou po uplynutí počtu taktů odpovídajícímu počtu stagí. Po uplynutí této doby je na výstupu v každém dalším taktu další výsledek.

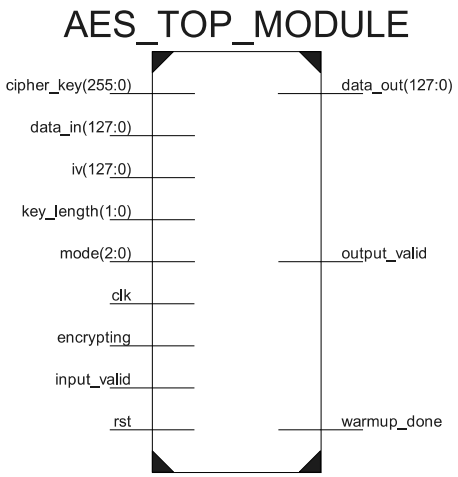


Obrázek 3.1: Dvě instrukce spouštěné bez implementovaného pipeliningu nahoře, dvě instrukce rozdělené na tři stage s využitím pipeliningu dole.

Pipelining je také využívaný v mikroprocesorech pro urychlení zpracování instrukcí. Jak může vypadat urychlení výpočtu pro příklad dvou instrukcí, je zachycené na Obrázku 3.1. Stejný princip platí i pro pipelining využívaný v integrovaných obvodech.

3.4 Vstupy a výstupy

Vzhledem k možnostem různého nastavení modulu je vysoký požadavek na I/O bloky. Dohromady je potřeba 651 I/O bloků. Podrobný popis vstupních signálů je v Tabulce 3.1 a výstupní signály jsou v Tabulce 3.2.



Obrázek 3.2: I/O schéma modulu

Tabulka 3.1: Vstupní signály

Název	Označení	Popis	Šířka (v bitech)
Vstupní data	`data_in`	Data, která mají být zašifrovaná nebo nešifrovaná.	128
Inicializační vektor	`iv`	Inicializační vektor, který je používaný pro operační módy CBC, CFB, OFB a CTR. V módu ECB není vektor použitý.	128
Klíč	`cipher_key`	Pokud je potřeba klíč kratší než 256 bitů, použije se 128, resp. 192 nejméně významných bitů.	255
Operačního módu	`mode`	Slouží pro výběr použitého operačního módu: 0x0 - ECB 0x1 - CBC 0x2 - CFB 0x3 - OFB ostatní - CTR	3
Délka klíče	`key_length`	Volba délky klíče: 0x0 - 128 bitů 0x1 - 192 bitů ostatní - 256 bitů	2
Validita vstupu	`input_valid`	Při požadavku na šifrování je nutné nastavit na 1. Pro správnou funkčnost je potřeba nastavovat jen v taktech, kdy chceme šifrovat nový vstup.	1
Režim	`encrypting`	Výběr mezi šifrování a dešifrováním: 0x0 - data jsou dešifrována 0x1 - data jsou šifrována	1
Hodinový signál	`clk`	Signál pro přivedení hodinového taktu.	1
Reset	`rst`	Signál pro resetování modulu.	1

Tabulka 3.2: Výstupní signály

Název	Označení	Popis	Šířka (v bitech)
Validita výstupu	`output_valid`	Signalizuje, že výstupní data jsou platná	1
Expanze klíče	`warmup_done`	Signalizuje, že klíč byl expandován a je možné začít šifrovat data	1
Výstupní data	`data_out`	Zašifrovaná, případně dešifrovaná data	128

3.5 Popis funkcionality

Vždy před začátkem prvního šifrování nebo dešifrování s daným klíčem je tento klíč expandován. Dokončení expanze klíče je indikováno signálem `warmup_done`. Nepředpokládají se příliš časté změny klíčů.

Vždy při změně některého z parametrů ovlivňujících chování modulu (šifrovací klíč, inicializační vektor, délka klíče, mód nebo směr šifrování) je pro deterministické chování potřeba reset. Když je klíč expandovaný (`warmup_done` je nastavený na '1'), stačí při každém novém vstupu změnit na jeden takt signál `input_valid` na '1'. Signál `input_valid` slouží ke správné indikaci dokončení šifrování, tedy nastavování signálu `output_valid`.

3.6 Place and Route report

Výsledky place and route zobrazují využití zdrojů pro FPGA Xilinx Virtex-6 (XC6VLX760) jsou v Tabulce 3.3. Z výsledků je zřejmé, že zdrojů zůstalo na FPGA stále dost nevyužitých. Jediný zdroj, který je využitý více než z poloviny, jsou vstupní a výstupní bloky.

Tabulka 3.3: Place and Route report

Element	Used	Available	[%]
Slice Logic Utilization			
Number of Slice Registers	8,903	948,480	1 %
Number used as Flip Flops	8,903		
Number of Slice LUTs	48,796	474,240	10 %
Number used as logic	48,593	474,240	10 %
Number used as Memory	131	132,480	1 %
Number used exclusively as route-thrus	72		
Slice Logic Distribution			
Number of occupied Slices	16,163	118,560	13 %
Number of LUT Flip Flop pairs used	49,529		
Number with an unused Flip Flop	40,720	49,529	82 %
Number with an unused LUT	733	49,529	1 %
Number of fully used LUT-FF pairs	8,076	49,529	16 %
IO Utilization			
Number of bonded IOBs	651	1,200	54 %
Minimum period 4.046 ns (Maximum frequency: 247.158 MHz)			

Kapitola 4

Testování

Testbench AES modulu je napsaný stejně jako modul v jazyce VHDL. Vzhledem k tomu, že není technicky možné provést otestování všech možných vstupů, je modul testován jen na zlomku možných vstupních dat.

Všechny testy prošly post-translate simulací pro FPGA Xilinx Virtex-6 (XC6VLX760). Simulace proběhla v Xilinx ISim.

4.1 Typy testů

Testy lze rozdělit do tří kategorií podle toho, na jakou funkčnost jsou zaměřeny, resp. odkud jsou převzaty.

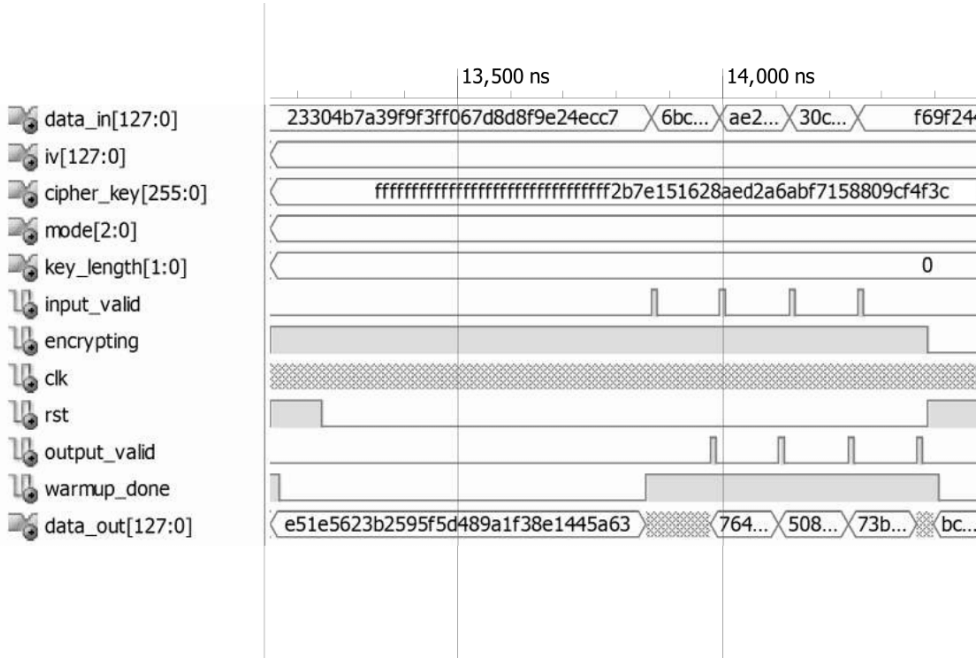
4.1.1 Testy podle vzorů standardu *FIPS PUB 197*

Standard *FIPS PUB 197* obsahuje ukázky vstupů a výstupů základní šifry AES, tedy oproštěné od operačních módů. Aby bylo možné testovat pouze funkčnost šifry AES, je simulován mód ECB, který nijak nemění vstup ani výstup samotné šifry. Testy obsahují pro každou délku klíče jeden vstup pro šifrování a následně inverzně na dešifrování. Dohromady tedy obsahuje šest vstupů. Testy slouží pouze pro ověření, zda šifra funguje korektně, takže nekladou žádné nároky na rychlost. Standard obsahuje rozepsání jednotlivých rund až na úroveň rundovních funkcí. Je tak možné velice dobře zkontrolovat funkčnost samotné šifry AES.

4.1.2 Testy podle vzorů standardu *NIST SP 800-38A*

Standard *NIST Special Publication 800-38A* popisuje operační módy pro blokové šifry. Obsahuje také ukázkové vstupy pro jednotlivé operační módy. Pro každý operační mód a pro všechny délky klíčů jsou čtyři vstupní vektory. Vždy je popsáno jak šifrování, tak dešifrování. Mód CFB zde má popsány dohromady tři varianty, jedná se o 1-bitové, 8-bitové a 128-bitové CFB. Počet bitů

udává, kolik bitů má výstup resp. vstup šifrování. Za účelem maximalizace propustnosti je implementován pouze 128-bitový CFB. Ostatní varianty tedy nejsou ani testovány. Dohromady tato část obsahuje 120 různých kombinací vstupů. Není zde opět testována rychlost. Testuje se zde především správnost průchodu jednotlivými operačními módy a v druhé řadě také správnost samotné šifry.



Obrázek 4.1: Část simulace s ukázkou šifrování AES v módu CBC s klíčem o velikosti 128 bitů.

4.1.3 Náhodně vygenerované testy

Poslední skupina testů obsahuje vstupní data vygenerovaná pomocí webu random.org. Data by tak měla být skutečně náhodná, jelikož služba využívá pro generování čísel atmosférický šum. Jako referenční výstupy jsou zde použity výstupy z OpenSSL. Testování je zaměřené na pipelining, v každém takt jsou na vstup přiváděna nová data. Dohromady je zde použito 96 různých kombinací vstupů, polovina pro ECB a polovina pro CTR (ostatní módy nepodporují pipelining). Opět je testováno pro všechny délky klíčů šifrování i dešifrování.

Závěr

V rámci této bakalářské práce jsem vytvořil šifrovací a dešifrovací modul pro šifru AES. Moduly podporují všechny délky klíčů, tedy 128, 192 a 256 bitů. Moduly jsou zastřešené operačními módy pro blokové šifry. Podporovány jsou všechny základní módy popsané ve standardu *NIST Special Publication 800-38A*, tedy ECB, CBC, CFB, OFB a CTR.

Pouze módy ECB a CTR dokáží využít pipelining a díky tomu lze dosáhnout frekvence 247 MHz. Frekvence 247 MHz při zpracování 128-bitových bloků znamená propustnost zhruba 31,6 Gbit/s[9]. Ostatní módy jsou bez pipeliningu, protože by nebylo možné jej použít pro šifrování i dešifrování. Délka operace je závislá na délce klíče a při frekvenci 247 MHz odpovídá propustnosti 2,87 Gbit/s pro 128 bitů, 2,43 Gbit/s pro 192 bitů a 2,11 Gbit/s pro 256-bitové šifrování.

Na základě této práce byl vyvinut modul, který je možné nad rámec této práce nadále vyvíjet. Prostor pro vývoj vidím zejména ve zrychlení výpočtu pomocí paralelizace. Dále by bylo možné analyzovat možnosti útoku postranními kanály a provést jejich ošetření. Modul byl navrhován pro maximalizaci výkonu, není tedy řešena ani přílišná optimalizace zdrojů, je tak možné modul optimalizovat i z tohoto hlediska.

[9]Všechny rychlosti jsou měřené pro FPGA Xilinx Virtex-6 (XC6VLX760)

Použité zdroje

[1] DAMICO, Tony M. A Brief History of Cryptography. In: StudentPulse [online]. 2009 [cit. 2016-05-07]. Dostupné z: http://www.studentpulse.com/a?id=41

[2] KESSLER, Gary C. An Overview of Cryptography. Gary Kessler Associates [online]. 2016 [cit. 2016-05-07]. Dostupné z: http://www.garykessler.net/library/crypto.html#desmath

[3] SMITH, Kenneth James. Methodologies for power analysis attacks on hardware implementations of AES. Rochester, New York, 2009. Rochester Institute of Technology. Dostupné z: http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=4201&context=theses

[4] WILLIAMS, Elliot. Quantum computing kills encryption. Hackaday [online]. Pasadena: Supplyframe, Inc., 2015 [cit. 2016-04-20]. Dostupné z: http://hackaday.com/2015/09/29/quantum-computing-kills-encryption/

[5] DWORKIN, Morris. National Institute of Standards and Technology Special Publication 800-38A Recommendation for Cipher Modes of Operation. Washington: U.S. Government printing office, 2001. Dostupné z: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[6] DIEBUCHEBOT. Wikimedia commons [online]. [cit. 2016-05-07]. Dostupné z: https://commons.wikimedia.org/wiki/File:Fpga_structure.svg

[7] AL-ARAJI, Saleh R, Zahir M HUSSAIN a Mahmoud A AL-QUTAYRI. Digital phase lock loops: architectures and applications. 1. vyd. Dordrecht: Springer, 2006. ISBN 978-038-7328-638.

[8] XILINX. All Programmable 7 Series Product Tables and Product Selection Guide. 2015 [cit. 2016-05-07]. Dostupné z: http://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf

[9] XILINX. UltraScale FPGA Product Tables and Product Selection Guide. 2015 [cit. 2016-05-07]. Dostupné z: http://www.xilinx.com/support/documentation/selection-guides/ultrascale-fpga-product-selection-guide.pdf

[10] XILINX. UltraScale+ FPGAs Product Tables and Product Selection Guide. 2015 [cit. 2016-05-07]. Dostupné z: http://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf

[11] XILINX. Virtex-6 Family Overview. 2015 [cit. 2016-05-07]. Dostupné z: http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[12] BUDZYŃ, G. Programmable Logic Design. VHDL vs Verilog. Wrocław University of Science and Technology 2015 [cit. 2016-05-07]. Dostupné z: http://www.ue.pwr.wroc.pl/pld/pld_12.pdf

[13] LÓRENCZ, R. Bezpečnost. Blokové šifry, DES, 3DES, AES, operační módy blokových šifer. Praha ČVUT FIT 2015. Dostupné z: https://edux.fit.cvut.cz/courses/BI-BEZ/_media/bez_n4.pdf

[14] Advanced Encryption Standard (AES) (FIPS PUB 197). National Institute of Standards and Technology, 2001. Dostupné také z: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[15] DAEMEN, Joan a Vincent RIJMEN. The design of Rijndael: AES–the Advanced Encryption Standard. New York: Springer, 2002. ISBN 35-404-2580-2.

[16] AL TAMIMI, Abdel-Karim. Performance Analysis of Data Encryption Algorithms [online]. 2008 [cit. 2016-05-07]. Dostupné z: http://www.cse.wustl.edu/~jain/cse567-06/ftp/encryption_perf/

[17] ELFACEITOSO. Deviant Art [online]. [cit. 7.5.2016]. Dostupný z: http://elfaceitoso.deviantart.com/art/Lisa-Vector-136475106 pod licencí CC BY-NC 3.0 http://creativecommons.org/licenses/by-nc/3.0/

[18] BARTÍK, Matěj. Implementace šifrování AES-128 ve VHDL [online]. 2014 [cit. 3.11.2016]. Dostupné z https://edux.fit.cvut.cz/courses/MI-SOC/tutorials/06/start

[19] NORMAN, Chris. Parallel AES Implementation [online]. 2011 [cit. 7.5.2016]. Dostupné z: http://www.cse.buffalo.edu/faculty/miller/Courses/CSE633/Chris-Norman-Fall-2011.pdf

Příloha A

Použití

- Vstupní data se přivádějí vždy na `data_in` a směr šifrování se vybírá pomocí `encrypting` ('1' šifruje, '0' dešifruje).
- Inicializační vektor se využívá ve všech módech kromě ECB. Pro správnou funkčnost by měl být nastaven už před resetem modulu.
- Šifrovací klíč se používá ve velikosti definované pomocí `key_length` (hodnoty viz. Tabulka 3.1), přičemž vždy je využit daný počet nejméně významných bitů. Pro správnou funkčnost by měl být nastaven už před resetem modulu.
- Operační mód se vybírá pomocí `mode` (hodnoty viz. Tabulka 3.1).
- V okamžiku, kdy se přivede blok vstupních data je potřeba nastavit signál `input_valid` na hodnotu '1' po dobu jednoho taktu.
- Hodinový signál, pomocí kterého je obvod synchronizovaný, se přivede na `clk`, maximální frekvence pro XC6VLX760 je 247 MHz.
- Pro resetování modulu slouží signál `rst`. Pro korektní funkčnost je potřeba resetovat po každé rekonfiguraci modulu (změna klíče, délky klíče, operačního módu, inicializačního vektoru nebo směru šifrování).
- Po vyresetování modulu probíhá expanze klíče, jejíž dokončení oznamuje signál `warmup_done`, který je po expanzi nastaven na '1'.
- Indikace korektního výstupu probíhá pomocí `output_valid`. Funguje v závislosti na správném nastavení `input_valid`. Vždy po dobu jednoho taktu, kdy jsou platná výstupní data je nastaven na hodnotu '1'.
- V případě módů CBC, CFB a OFB, které nejsou pipelinované je nutné s novým vstupem počkat do signalizace dokončení předchozího vstupu. U ECB a CTR lze vkládat nové vstupy ihned za sebou.

Příloha B

Seznam použitých zkratek

3DES Triple Data Encryption Standard

AES Advanced Encryption Standard

AES-128 Advanced Encryption Standard využívající 128-bitový klíč

AES-192 Advanced Encryption Standard využívající 192-bitový klíč

AES-256 Advanced Encryption Standard využívající 256-bitový klíč

BMP Bitmap

CB Ciphered Block

CBC Cipher Block Chaining

CFB Cipher Feedback

CLB Configurable Logic Blocks

CTR Counter

DES Data Encryption Standard

ECB Electronic Codebook

FPGA Field-Programable Gate Array

GF(2^8) Galois Field 2^8

HDL Hardware Description Language

IV Initialization Vector

LUT Look-Up Table

NIST National Institute of Standards and Technology

OFB Output Feedback

PB Plain Block

RSA Rivest-Shamir-Adleman

SRAM Static Random Access Memory

VHDL VHSIC Hardware Description Language

VHSIC Very High-Speed Integrated Circuit

XOR Exclusive OR

Příloha C

Obsah přiloženého DVD

- `readme.txt` ..... stručný popis obsahu DVD
- `project` ..... adresář s projektem, soubory z P&R a simulace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `BP_Hynek_Daniel_2016.pdf` ..... text práce ve formátu PDF
- `ZZP_Hynek_Daniel_2016.pdf` ... zadání závěrečné práce ve formátu PDF