REALIZACIJA KRIPTOLOŠKIH ALGORITAMA U SISTEMIMA VELIKIH PROTOKA PODATAKA

Nikola M. Jaćimović, *Bratislav* Ž. Planić
Vojska Srbije, Generalštab, Uprava za telekomunikacije i informatiku (J-6), Centar za primenjenu matematiku i elektroniku, Beograd
e-mail: nikola.jacimovic@vs.rs; bratislav.planic@vs.rs

DOI: 10.5937/vojtehg63-7392
OBLAST: računarske nauke, elektronika, telekomunikacije
VRSTA ČLANKA: originalni naučni članak
JEZIK ČLANKA: srpski

Sažetak:

U radu je analizirana implementacija kriptoloških algoritama tako da se dobiju što je moguće bolje performanse sa aspekta brzine. Na taj način pruža se podrška zaštićene komunikacije između dva učesnika sa što je moguće manjim uticajem na performanse date mrežne infrastrukture. U radu je objašnjena razlika između hardverske i softverske realizacije algoritama za šifrovanje. Prikazane su osnovne karakteristike FPGA čipova i napredne mogućnosti VHDL jezika za dizajniranje koji su korišćeni za implementaciju šifarskog algoritma. Za implementaciju je odabran AES-256 algoritam šifrovanja koji ne samo da predstavlja jedan od najpoznatijih javnih algoritama danas, već se jednako dobro pokazao kako u hardverskoj tako i u softverskoj varijanti. Razvojno okruženje koje je korišćeno je Xilinx ISE Design Suite, a razvojne ploče Xilinx Spartan SP-605 i Xilinx Kintex KC-705. Svi dobijeni rezultati odgovaraju uređajima koji u sebi imaju Spartan®-6, odnosno Kintex®-7 čip.

Ključne reči: *AES, optimizacija dizajna, VHDL, FPGA, šifrovanje.*

Uvod

Konvergencijom podataka, govora i videa u svetu računara, pojavom novih računarskih tehnologija, te povećanjem broja korisnika koji svoje poslovanje oslanjaju na elektronskoj razmeni podataka doveli su do toga da je potražnja za propusnim opsegom sve veća. Istovremeno, sa porastom i realizacijom takvih zahteva javlja se i sve veća potreba za bržom razmenom informacija koje tokom prenosa moraju biti zaštićene.

U jednom takvom okruženju, u današnje vreme, najčešće ni količina podataka ni kapaciteti prenosnog kanala ne predstavljaju problem u pogledu propusne moći čitavog sistema. Razlog za to krije se u sve nižoj ceni

same mrežne opreme. Ono što se po pravilu javlja kao „usko grlo" je sam proces šifrovanja, odnosno dešifrovanja informacije koja se razmenjuje, a koji iziskuje dodatno vreme za obradu. Samim tim, ili će vreme potrebno za predaju biti zakašnjeno, ili će protok razmene podataka biti umanjen.

Prvi slučaj se javlja, na primer, ukoliko se postupak šifrovanja izvršava na višim nivoima OSI (*Open Systems Interconnection*) modela (npr. aplikativni). U takvim situacijama prvo se izvrši šifrovanje, a zatim se šalju podaci. Kao primer može se uzeti slanje elektronske pošte. Mada vreme potrebno za šifrovanje često izgleda zanemarljivo sa aspekta krajnjeg korisnika i pripada vremenu koje odgovara obradi informacija, a ne njihovom prenosu, potrebno je napomenuti da predmet istraživanja ovog rada predstavljaju sistemi velikih protoka, u kojima se svako kašnjenje koje postoji mora uzeti u razmatranje.

Drugi slučaj, smanjenje protoka razmene podataka, dešava se u sistemima koji kriptozaštitu realizuju na fizičkom nivou OSI referentnog modela. To najčešće podrazumeva da je implementacija kriptološkog algoritma i pratećih interfejsa izvršena na hardverskom nivou. Koliko će proces kriptološke obrade podataka uticati na brzinu prenosa u datom sistemu najviše zavisi od kompleksnosti samog algoritma. Međutim, ne sme se zanemariti ni činjenica da i sam način implementacije i postupak dizajniranja pomenutih hardverskih kriptoblokova bitno menja performanse čitavog sistema. Upravo ova oblast predstavlja osnovnu temu rada i praktično dokazuje navedenu tvrdnju.

Dakle, u zavisnosti od složenosti i načina realizacije algoritma koji se koristi za kriptozaštitu podataka, može se razmatrati maksimalna brzina razmene zaštićenih podataka, bez obzira na performanse koje data mrežna infrastruktura pruža.

Imajući to u vidu, a s obzirom na to da se jedan kriptosistem, kao sistem koji obezbeđuje šifrovanje i dešifrovanje podataka, može implementirati i hardverski i softverski, prvo pitanje koje se postavlja jeste koji će od ova dva načina implementacije dati bolje performanse čitavog sistema. Sledeće pitanje je koja vrsta tehnologije će biti korišćena. Poslednji problem, koji u pojedinim slučajevima može biti od presudne važnosti za performanse, predstavlja način same implementacije prethodno odabranih elemenata.

Upravo na ovaj način će se, u daljem razmatranju, izvršiti pristup problemu realizacije kriptoloških algoritama u sistemima velikih protoka.

Hardverski i softverski kriptosistemi

Već je napomenuto da algoritmi u jednom šifarskom sistemu mogu biti implementirani na dva načina: softverski ili hardverski.

Osnovne prednosti softverske realizacije algoritma za šifrovanje su: prenosivost, fleksibilnost i jednostavnost korišćenja i nadgradnje. To znači da se algoritam napisan u nekom programskom jeziku (npr. C/C++,

JAVA, ...) može izvršavati, uz male izmene i bez velikih dodatnih troškova, na svakom računaru. Takođe, ne postoje bitnija ograničenja ni prilikom prenošenja koda sa jedne na drugu aplikaciju.

Uprkos tome, NSA (*National Security Agency* – državna bezbednosna agencija Sjedinjenih Američkih Država) autorizuje isključivo hardversko šifrovanje (Schneier, 1996, p.192). Takođe, za vojne i ozbiljne komercijalne primene hardver se još uvek koristi kao glavni vid realizacije algoritama za šifrovanje. Prema Schneier-u (1996, pp.192-194) postoji nekoliko razloga za to.

Prvi razlog je brzina. Svaki algoritam za šifrovanje sastoji se od mnoštva komplikovanih operacija nad bitovima. S obzirom na to da te operacije nisu podrazumevano ugrađene ni u jedan računar (procesor) opšte namene, softverske realizacije kriptoloških algoritama će raditi neefikasno. Osim toga, šifrovanje je često postupak sa mnogo izračunavanja, a dodeljivanje takvog zadatka osnovnom procesoru je takođe neefikasno. Hardversko šifrovanje podrazumeva šifrovanje podataka na posebnom (specijalnom) čipu unutar uređaja, čime se bitno dobija na brzini. Mada postoje napori pojedinih istraživača da optimiziraju vremensko izvršavanje softverskih rešenja, dobijeni rezultati su uvek lošiji od onih koje nudi specijalizovan hardver.

Drugi razlog je bezbednost. Algoritam za šifrovanje koji se izvršava na računaru opšte namene nema nikakvu fizičku zaštitu koja bi sprečila napadača da upadne u sistem i izvrši određene izmene algoritma. Za razliku od njih, uređaji za hardversko šifrovanje mogu biti dodatno fizički zaštićeni – od načina čuvanja takvih uređaja (zaključane prostorije, video nadzor itd.) do načina realizacije fizičke zaštite na samom uređaju (npr. *tamper-proof*). VLSI čipovi posebne namene mogu da budu obloženi takvom supstancom da će svaki pokušaj pristupanja njihovoj unutrašnjosti dovesti do uništavanja logike čipa (takvi su svojevremeno bili čipovi američke vlade *Clipper* i *Capstone*). Na ovaj način, oprema koja štiti od upada može da spreči i izmenu uređaja za hardversko šifrovanje.

Treći razlog za nadmoćnost hardvera je jednostavnost instalacije. Naime, šifrovanje komunikacije između uređaja koji ne predstavljaju računare opšte namene (telefon, faks, modem itd.) mnogo je jednostavnije i jeftinije realizovati ugradnjom hardvera za šifrovanje nego ugradnjom softvera i određenog mikroprocesora.

Imajući to u vidu, sasvim je jasno da će se za primenu u sistemima velikih protoka radije koristiti hardverska realizacija kriptoalgoritama od softverske. To se u praksi svakodnevno potvrđuje od strane mnogih vodećih organizacija koje u svojim proizvodima (kriptouređajima) koriste ovaj pristup zaštite informacija (mada su podaci o tehničkoj realizaciji aktuelnih rešenja veoma oskudni i teško se mogu naći na zvaničnim sajtovima).

Zbog niza prednosti, kao jedna od najzahvalnijih tehnologija hardverske realizacije kriptoloških algoritama nameće se primena programabilnih čipova.

FPGA logička kola

Danas je na tržištu dostupan veliki broj različitih tipova programabilnih logičkih kola (*Programmable Logic Device* – PLD kola), koja se razlikuju po načinu programiranja, složenosti, brzini rada i broju pinova.

Osnovna prednost FPGA (*Field Programmable Gate Array*) čipova u odnosu na ostala PLD kola jeste što je proces projektovanja dosta ubrzan, a proces testiranja i *debug*-ovanja znatno olakšan zbog mogućnosti jednostavnog rekonfiguriranja delova sistema u toku projektovanja (El Mezeni, et al., 2008). Kada se uzme u obzir i činjenica da je cena razvoja takvog dizajna niža od drugih načina programiranja hardvera (npr. izrada ASIC kola), jasno je zašto su FPGA kola veoma privlačna za dizajniranje.

Da bi se napravio efikasan FPGA dizajn, HDL (*Hardware Description Language*) opis mora se prilagoditi zahtevima FPGA arhitekture, tako da bude iskorišćeno što više ponuđenih hardverskih resursa. Osim toga, potrebno je proučiti određeni alat za sintezu, dok je vrsta HDL-a koji se koristi (VHDL, Verilog,...) od manjeg značaja (Gschwind, Salapura, 1995) (Farooq, et al., 2012).

Kao jedno od rešenja koja se mogu primeniti za dizajniranje kompleksne logike u ovom radu korišćen je VHDL (***V****ery High Speed Integrated Circuit* ***H****ardware* ***D****escription* ***L****anguage*) – jezik za opis hardvera.

Primena FPGA čipova u kriptozaštiti

Usled komercijalnih potreba koje se ogledaju u smanjenju troškova dizajniranja kola, ali i u pogledu smanjenja rizika i bržeg pojavljivanja na tržištu, upotreba FPGA programabilnih kola se povećala. Drugim rečima, za neke aplikacije više nije neophodno prebacivanje prototipskog dizajna sa FPGA na ASIC kola, već se takvi proizvodi pojavljuju na tržištu sa već programiranim FPGA kolom (Good, Benaissa, 2005).

U sistemima kriptozaštite prednost upotrebe ovakvih čipova je višestruka (Wollinger, et al. 2004):

– Prilagodljivost algoritama (eng. *algorithm agility*) koja se ogleda u zameni kriptografskih algoritama usled različitih potreba (promena trenutne sesije, narušavanje tajnosti kriptoparametara, itd.) nezavisno od komunikacionih protokola.

– Nadogradnja algoritma (eng. *algorithm upload*) je veoma bitna karakteristika, s obzirom da neophodne modifikacije (ili izmene) već implementiranog bloka za šifrovanje u slučaju drugih tehnologija (na primer ASIC čipova) mogu biti nemoguće.

– Modifikacija algoritma (eng. *algorithm modification*) koji podrazumeva izmenu osnovnih kriptoparametara (npr. *S-box* tabela).

– Propusnost (eng. *throughput*), budući da procesori opšte namene ne moraju biti optimizovani za brzo izvršavanje određenih operacija.

– Cena (eng. *cost efficiency*) koja se posmatra sa dva aspekta: cena razvoja i cena hardverske komponente. Kada se posmatra razvoj na jednom FPGA čipu, dolazi se do zaključka da je to mnogo jeftinije nego npr. ASIC implementacija. To, takođe, rezultira bržom pojavom proizvoda na tržištu, što u današnje vreme može biti veoma značajno.

Naravno, kako bi navedene prednosti bile što bolje iskorišćene, potrebno je efikasno koristiti resurse čipa koji su na raspolaganju. U zavisnosti od toga koliko će se prilikom dizajniranja voditi računa o optimizaciji kritičnih putanja signala, mogu se dobiti različite performanse sistema. S obzirom na to, u sledećem poglavlju biće dat detaljniji prikaz naprednih tehnika VHDL jezika za dobijanje optimiziranog dizajna.

VHDL optimizacija

Inicijalna struktura sintetizovane logike direktno se dobija od strukture njenog opisa. Dakle, da bi ostvarili neophodne optimizacije u vidu efikasnog iskorišćenja ponuđenih resursa na samom FPGA čipu, potrebno je prilagoditi opis tog dizajna i kodovati ga na način kojim će se, alatu za sintezu, precizno definisati kako da implementira opisanu logiku. Osim toga, sofisticirani alati za optimizaciju često nisu dovoljno dobri za postizanje neophodnih performansi određenog dizajna. Pod pojmom performansi digitalnog kola možemo posmatrati tri osnovne grupe – brzinu rada, površinu koju zauzima na čipu i napajanje (eng. *power*). U ovom radu od posebnog je interesa brzina, pa se ostale performanse mogu posmatrati kao manje restriktivne kategorije.

Postoje tri osnovne definicije brzine u zavisnosti od konteksta problema: propusnost (eng. *throughput*), kašnjenje (eng. *latency*) i vremensko usaglašavanje (eng. *timing*) (Kilts, 2007).

Propusnost podrazumeva količinu podataka koji se mogu obraditi za jedan takt. Najčešće se izražava jedinicom bps (eng. *bits per second*).

Kašnjenje se odnosi na vreme koje protekne od prijema ulaznog podatka (signala) do dobijanja odgovarajućeg izlaza. Uglavnom se meri vremenskim jedinicama ili brojem taktova.

Vremensko usaglašavanje podrazumeva logička zadržavanja između sekvencijalnih elemenata. Naime, ako naš dizajn ne ispunjava uslov vremenskog usaglašavanja, to znači da je kašnjenje signala na kritičnom putu (put signala koji ima najveće kašnjenje) između dva flip-flopa veće od zadatog perioda takta. To se može desiti iz više razloga, kao što su – kompleksna kombinaciona logika, kompleksno rutiranje signala itd. Merna jedinica za *timing* je period takta i frekvencija rada.

Propusnost (throughput)

Da bi dobili veću propusnost signala, uvodi se koncept *pipeline*-a. Ova tehnika se koristi u skoro svim uređajima sa visokim performansama, kao što su centralne procesorske jedinice, stekovi mrežnih protokola, ali i uređaji za šifrovanje.

Najčešće *pipeline* nastaje tako što se podeli neka kompleksna operacija na više jednostavnijih. Kako se svaki kriptoalgoritam sastoji od mnoštva složenih operacija nad bitovima, lako je zaključiti da se tehnika *pipeline*-a mora javljati u svakom dobro optimizovanom kodu za šifrovanje. U zavisnosti od načina implementacije razlikuju se dve vrste *pipeline*-a: instrukcijski (koji predstavlja način rada raznih procesorskih jedinica) i hardverski (koji se koristi prilikom dizajniranja logike).

Implementacija instrukcijskog *pipeline*-a pomoću VHDL-a je moguća, ali samo ukoliko se dizajnira mikroprocesor na FPGA. S obzirom na to da je, na početku ovog rada, objašnjena prednost hardverske realizacije algoritama sa aspekta brzine izvršavanja, i da takva implementacija ne podrazumeva procesor kao hardversku podlogu, to implementacija instrukcijskog *pipeline*-a pomoću VHDL-a neće biti razmatrana u daljem radu.

S druge strane, kako se svaki FPGA dizajn sastoji od skupa elemenata za obradu informacija povezanih u niz, tako da izlaz iz jednog elementa predstavlja ulaz u sledeći, ideja hardverskog *pipeline*-a sastoji se u obradi manje količine podataka za kraći period. Drugim rečima, ukoliko je dizajn kompleksan, potrebno je prepoznati i razviti ga u manje, jednostavnije korake. To se postiže pamćenjem međurezultata složenih operacija. Na taj način povećava se maksimalna frekvencija takta, smanjuje vreme sinteze koda i, što je u ovom slučaju najvažnije, povećava propusnost sistema.

Bez obzira na to što je ovaj način pisanja koda nešto komplikovaniji od uobičajenog, on se ne sme zaobilaziti ukoliko je potrebno maksimalno povećanje brzine rada algoritma. To se posebno odnosi na velike projekte koji mogu biti usko grlo u pogledu performansi za čitav dizajn.

Pipeline ima veliku primenu u iterativnim sistemima koji koriste iste registre i resurse za obradu podataka tokom čitavog procesa izračunavanja, i u kojima se ne može početi sa novom obradom podataka dok se prethodna operacija ne obavi u potpunosti. Ovaj koncept poznat je pod nazivom „razvijanje petlji" (eng. *unrolling the loop*) (Kilts, 2007), a sastoji se u tome da se dati signal propušta kroz nivoe *pipeline*-a, koristeći nezavisne resurse za izračunavanje međurezultata. Tako, dok se računa vrednost X_k u drugom, sledeća vrednost X_{k+1} može se istovremeno obrađivati u prvom nivou *pipeline*-a.

Jedini nedostatak razvijanja petlje je povećanje prostora koji dizajn zauzima na čipu. Iterativna implementacija zahteva jedan registar, jedan kombinacioni blok za obradu signala i određenu kontrolnu logiku, dok *pipeline* implementacija zahteva, osim kontrolne logike, posebne registre za međurezultate, ali i po jedan kombinacioni blok za obradu signala za svaki nivo *pipeline*-a.

Kašnjenje (latency)

Kašnjenje dizajna predstavlja vremensku razliku između trenutka pojave signala na ulazu do dobijanja validnog izlaznog signala. Da bi to kašnjenje trajalo što kraće, potrebno je da se protok podataka sa ulaza na izlaz odvija što je moguće brže. To se ostvaruje optimiziranjem vremena potrebnog za obradu međurezultata. Često, smanjenje kašnjenja podrazumeva paralelizam, uklanjanje *pipeline*-a ili logičke prečice, što može prouzrokovati smanjenje propusnosti ili maksimalnog takta dizajna.

Posmatrajući problem razvijanja petlji ponovo, može se uočiti da ne postoji optimizacija kašnjenja kod iterativne implementacije, zbog toga što se rezultat jedne operacije obrade signala svakako mora snimiti u registar, pre nego što se izvrši sledeća obrada. Međutim, kod *pipeline* implementacije postoji jednostavan način za smanjenje kašnjenja. Lako se uočava da se posle svakog nivoa *pipeline*-a rezultat dobijene obrade smešta u registre i čeka narednu ivicu takta kako bi prešao u sledeći nivo. Uklanjanjem tih registara može se izvršiti minimizacija vremena kašnjenja kola, čime se u potpunosti dobija kombinaciona mreža.

Nedostatak ovakve realizacije digitalnog kola je u vremenu. Naime, *pipeline* implementacije bi teoretski mogle da se izvršavaju pri taktu sa periodom čije je trajanje približno kašnjenju jedne operacije obrade signala, dok eliminacijom registara *pipeline*-a taj period mora biti jednak zbiru kašnjenja svih operacija za obradu signala, plus kašnjenje kontrolne logike kritičnog puta.

Vremenska usaglašenost (timing)

Maksimalna brzina signala takta (eng. *clock speed*) u kolu određena je maksimalnim kašnjenjem između bilo koja dva sekvencijalna elementa. Mada ona dosta zavisi od odnosa kompromisa učinjenih u sferi „brzina”/„prostor na čipu”, sama ideja brzine signala takta javlja se na mnogo nižem nivou apstrakcije. Drugim rečima, bez detaljnog uvida u kompletnu implementaciju, ne možemo jasno reći da li će, na primer, *pipeline* dizajn biti brži od iterativnog ili ne, i slično.

Ukoliko se kao sekvencijalni elementi u dizajnu koriste D-flip-flopovi, maksimalna brzina, ili maksimalna frekvencija, može se definisati na sledeći način (Kilts, 2007):

$$F_{max} = \frac{1}{T_{clk-q} + T_{logic} + T_{routing} + T_{setup} - T_{skew}} \quad (1)$$

gde je:
F_{max} – maksimalna dopuštena frekvencija takta;

$T_{clk\text{-}q}$ – vreme od pojavljivanja ivice takta do pojavljivanja podatka na *Q* izlazu flip-flopa;
T_{logic} – propagaciono kašnjenje logike između flip-flopova;
$T_{routing}$ – kašnjenje usled rutiranja signala između flip-flopova;
T_{setup} – minimalno vreme za koje podatak mora stići na *D* ulaz flip-flopa, pre nego što se pojavi naredna ivica takta;
T_{skew} – propagaciono kašnjenje takta između izlaznog i ulaznog flip-flopa

Brzina takta može se poboljšati korišćenjem različitih metoda. To su:

– dodavanje registara koji čuvaju međurezultate na kritičnoj putanji. Ovu tehniku poželjno je primenjivati u dizajnima sa velikim nivoima *pipeline*-a, kako kašnjenje zbog dodatog takta ne bi mnogo narušilo dobijene performanse, a ukupna funkcionalnost kola ostala nepromenjena;

– paralelne strukture, odnosno reorganizacija kritične putanje, tako da se logičke strukture implementiraju u paraleli. Ovu tehniku bi trebalo koristiti kad god nam funkcija koja se izvršava serijski dozvoljava da se razdvoji i obrađuje u paraleli. Na taj način, bitno se može smanjiti i maksimalno kašnjenje;

– spljoštene (eng. *flatten*) logičke strukture kao tehnika koja se posebno primenjuje kod logike ulančanih prioriteta kodiranja. Na primer, u kodu se može javiti neki od prioritetnih zahteva, koji realno nije neophodan za dizajn. Alat za sintezu će ga prepoznati i implementirati, iako ignorisanje tog zahteva ne bi uopšte uticalo na funkcionalnost kola, a istovremeno bi, uvođenjem paralelizma, mogle da se dobiju bolje performanse;

– balansiranje registara predstavlja ravnomernu preraspodelu logike između registara radi minimiziranja najgoreg slučaja kašnjenja između bilo koja dva registra. Ovu tehniku trebalo bi koristiti kad god postoji velika neujednačenost između kritičnog i njemu susednog puta. Pošto je brzina takta ograničena najgorim slučajem putanje, ponekad se na ovaj način, uz veoma male izmene dizajna, može dobiti velika optimizacija;

– preraspodela puta toka podataka je tehnika koja se koristi kad god je više putanja kombinovano sa kritičnom, i ta kombinovana putanja se može preraspodeliti tako da se deo logike kritičnog puta pomeri bliže odredišnom registru.

Implementacija AES-256 (Rijndael) kriptološkog algoritma

Osnovni koncept AES algoritma

U zavisnosti od operacija koje se izvršavaju nad nizom otvorenog teksta, algoritmi za šifrovanje mogu biti efikasniji kao softversko nego kao hardversko rešenje, i obrnuto. Pojedini algoritmi, kao što je AES, pokazali

su se jednako dobri za implementaciju u obe ove varijante (Karimian, et al., 2012). Radi dobijanja objektivnih rezultata, za konkretan primer efikasne hardverske realizacije kriptosistema korišćen je algoritam AES-256 (Rijndael).

Šifarski algoritam *Advanced Encryption Standard* (*AES*) jedan je od najpopularnijih šifarskih algoritama danas (Jevremović, 2011). Odobrio ga je *NIST* 2000. godine, kao federalni standard za obradu informacija, a njegov opis je dokumentovan u publikaciji poznatoj kao FIPS-197 (http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001).

AES algoritam predstavlja simetrični blokovski algoritam koji može koristiti ključeve dužine 128, 192 i 256 bitova. Osnovna jedinica za obradu podataka kod AES algoritma je bajt i sve operacije se izvršavaju nad dvodimenzionalnim nizovima bajtova koji se nazivaju stanja. Dužina ulaznog bloka, izlaznog bloka i stanja je 128 bitova. Ovi nizovi sadrže po četiri reda bajtova, od kojih svaki sadrži *N* bajtova (*N* je veličina bloka podeljena sa 32) (Veinović, Jevremović, 2011).

Broj rundi kod ovog algoritma zavisi od dužine ključa i iznosi 10, 12 ili 14 rundi za dužine ključa od 128, 192 i 256 bitova respektivno. Za procese šifrovanja i dešifrovanja podataka AES algoritam koristi sledeće transformacije:

– nelinearnu zamenu bajtova na osnovu supstitucione tabele (*ByteSub*),

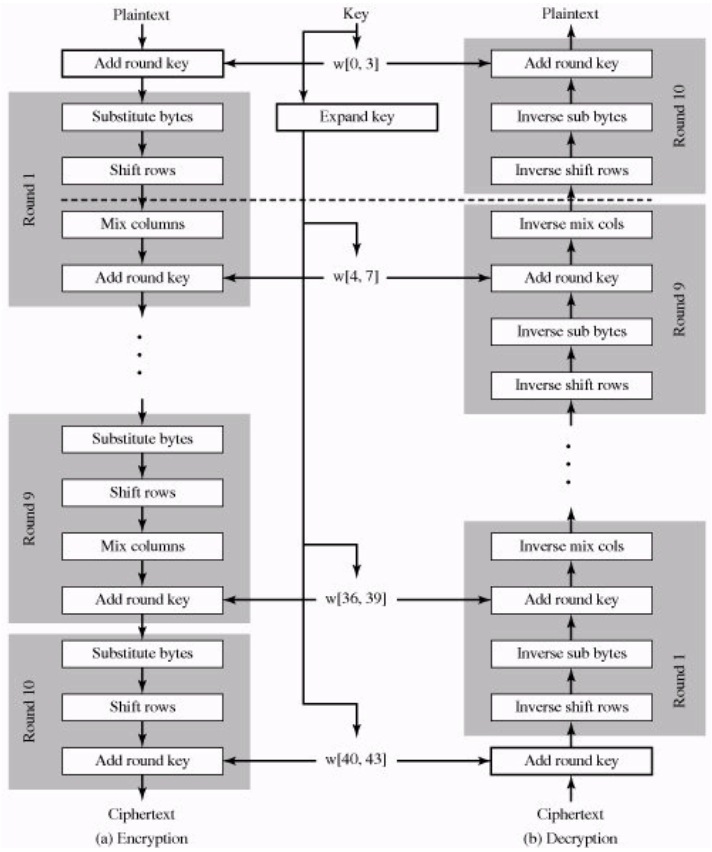
– promenu mesta bajtova unutar istog reda (*ShiftRow*),

– transformaciju bajtova unutar iste kolone (*MixColumn*) i

– sabiranje po modulu dva sa odgovarajućim delom ključa (*AddRoundKey*).

Celokupan proces šifrovanja prikazan je dijagramom na slici 1 (Marković, 2012).

Na početku procesa šifrovanja ulazni niz kopira se u matricu stanja. Nakon dodavanja *RoundKey* vrednosti matrica stanja se transformiše u rundama. Nakon poslednje runde, matrica stanja se kopira u izlazni niz. *RoundKey*, kao jedan od ulaznih parametara svake runde, jednodimenzionalni je niz reči od po četiri bajta, dobijen na osnovu *KeyExpansion* rutine. Ova rutina od glavnog ključa generiše tzv. raspored ključeva dužine *Nr* reči (*Nr* je broj ciklusa) od po četiri bajta (32 bita) (Veinović, Jevremović, 2011). U poslednjem ciklusu šifrovanja se ne odvija transformacija bajtova unutar iste kolone (*MixColumn*).



Slika 1 –AES algoritam
Figure 1 – AES algorithm
Рис. 1 – алгоритм AES

Algoritam ekspanzije ključa (*KeyExpansion* rutina) kao ulaznu vrednost uzima ključ *K*. Rezultat njegovog izvršenja predstavljaće niz 4-bajtnih reči *W* dužine 4x(*Nr*+1)(gde *Nr* predstavlja broj rundi u algoritmu). Tako da, ukoliko je npr. dužina ključa 128 bita, algoritam će imati 10 rundi, pa će niz *W* biti dužine 4x(10+1)=44 reči od 32 bita.

Ako sa w_i predstavimo *i*-ti element niza *W*, onda će način formiranja niza *W* biti (Marković, 2012):

$w_0 = k_0$
$w_1 = k_1$
$w_2 = k_2$
$w_3 = k_3$
$w_4 = w_0 \text{ xor } temp_1$
$w_5 = w_1 \text{ xor } w_4$

$w_6 = w_2 \text{ xor } w_4$
$w_7 = w_3 \text{ xor } w_4$
$w_8 = w_4 \text{ xor } temp_2$
:
:
$w_{43} = w_{39} \text{ xor } w_{42}$
$w_{44} = w_{40} \text{ xor } temp_{11}$

gde su k_0, k_1, k_2 i k_3 elementi ključa K. Može se primetiti da za izračunavanje w_i postoje dva slučaja:
– $w_i = w_{i-Nk}$ *xor* w_{i-1} za $i \bmod Nk \neq 0$,
– $w_i = w_{i-Nk}$ *xor* $temp_k$ za $i \bmod Nk = 0$, gde je $temp_k = ByteSub(S_1, W_{i-1})$ *xor* $rcon_k$, pri čemu je funkcija *ByteSub* zapravo funkcija supstitucije

VHDL implementacija AES algoritma

Pregled dosadašnjih rešenja

Odmah nakon objavljivanja AES algoritma, on postaje predmet mnogih istraživanja u oblasti pronalaženja pogodne arhitekture za hardversku implementaciju. Odabir te arhitekture i način realizacije konkretne logičke strukture zavisi od sistemskih zahteva u pogledu brzine i resursa.

Danas postoji niz radova koji obrađuju problem implementacije ovog algoritma za šifrovanje u FPGA ili ASIC čipove. Autori, osim osnovne implementacije, uglavnom obrađuju dve vrste optimizacije – optimizaciju prostora koji dizajn zauzima na čipu i optimizaciju vremena izvršavanja. Takođe, česta oblast istraživanja je i ukupna potrošnja snage određenog dizajna, što je posebno popularno za aplikacije koje se razvijaju za potrebe prenosnih uređaja. S obzirom na to da je za potrebe ovog rada od posebnog interesa optimizacija vremena izvršavanja algoritma, u ovom poglavlju detaljnije će biti prikazani trenutni rezultati do kojih su došli istraživači u svetu.

Jedno od prvih istraživanja na ovu temu je svakako rad (Gaj, Chodowiec, 2001) koji se pojavio iste godine kada je AES i prihvaćen kao standard. Performanse koje su tada dobijene pružale su adekvatnu podršku algoritma za rad na brzinama do 12,2 Gb/s.

Ukoliko se uzmu u obzir i drugi parametri na čipu, mogu se dobiti znatne razlike u brzini. Tako je u radu (Rodriguez-Henriquez, et al. 2003), s obzirom na to da on obrađuje paralelnu optimizaciju i prostora i vremena izvršavanja, dobijena brzina od 4121 Mb/s. Razlog ovakvih performansi je obrnuto srazmeran međusobni uticaj ove dve vrste optimizacija. Osnovni način vremenske optimizacije u ovom radu je *pipeline* tehnika i dobijeni rezultat je (i pored vođenja računa o prostoru na čipu) za 27,23% brži u odnosu na rešenje opisano u (McLoone, McCanny, 2001).

Značaj i rezultati koji se mogu dobiti pomoću korišćenja *pipeline*-a su dosta dobro objašnjeni u radu (Hodjat, Verbauwhede, 2003) gde su prikazana tri načina realizacije AES-128 algoritma na ASIC kolima. Prvi način realizacije predstavlja najbržu verziju AES-128 algoritma koja može dostići brzinu od 77,6 Gb/s, i ima 41 *pipeline* stanje (svaka runda je razbijena na 4 stanja). Druga verzija dizajna podrazumeva jedno *pipeline* stanje za jednu rundu, odnosno ukupno 11 stanja. Brzina koja se dobila ovom tehnikom je 48,2 Gb/s i oko 25% uštede prostora na čipu. Konačno, treći način realizacije podrazumeva izvršavanje 2 runde za jedno stanje, odnosno ukupno 5 *pipeline* stanja. Očekivano, ova verzija je i najsporija i iznosi 23,1 Gb/s. Na osnovu prikazanih rezultata može se zaključiti da različitim pristupom kodovanja možemo dobiti i do 3 puta brže izvršavanje algortima, dok je ušteda na prostoru između najbrže i najsporije verzije algoritma oko 50%.

Postoji nekoliko radova u oblasti implementacije efikasnog AES algoritma koji za potrebe optimizacije uvode novi koncept u pogledu hardverske realizacije *S-box* tabele. Naime, implementacija operacije zamene bajtova (*SubByte*) uglavnom se izvršava pomoću *look-up* tabele, čime se, zbog vremena potrebnog za prolazak kroz memorijske blokove FPGA, dobija kašnjenje koje nije moguće optimizirati. Realizovanjem ove operacije pomoću aritmetike dozvoljava se dublji nivo *pipeline*-a, a time i veća propusnost dizajna. Jedan od radova na ovu temu je svakako i rad (Good, Benaissa, 2005), u kojem su autori predstavili AES-128 verziju algortima koja radi na brzini od 25 Gb/s.

Sopstveno rešenje

Za efikasnu implementaciju sopstvenog hardverskog rešenja kriptološkog algoritma AES moraju se uzeti u obzir tehnike optimizacije VHDL dizajna, a kao polazna pretpostavka razmatranje mogućnosti njihovog dostizanja i/ili boljih rezultata od onih koji su trenutno dostupni. Takođe, mora se uzeti u obzir i kašnjenje koje se može javiti usled korišćenja različitih kriptografskih modova rada.

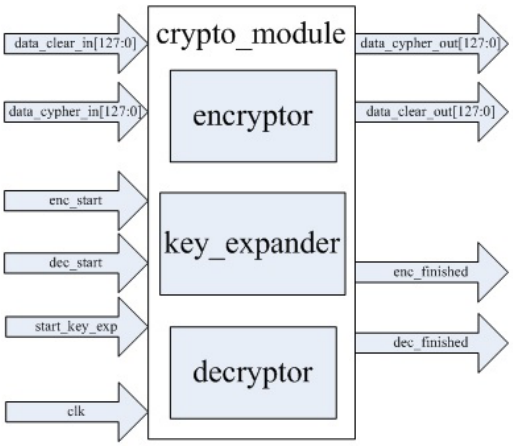
Poznato je da je najjednostavniji mod za realizaciju mod elektronske kodne knjige ECB (*Electronic Code Book*). S obzirom na to da ovaj mod ne podrazumeva nikakva dodatna preprocesiranja bloka otvorenog teksta, on nam omogućava rad u *pipeline* režimu, čime se propusnost bitno povećava. S druge strane, ECB mod je najmanje otporan na napad, s obzirom na to da se identični blokovi otvorenog teksta uvek preslikavaju u iste blokove šifrata. Ostali modovi rada (CBC, OFB...) povećavaju otpornost na napade, ali bitno utiču na performanse, upravo zbog toga što ograničavaju efikasnost *pipeline*-a.

U ovom radu, a radi boljeg uočavanja rezultata koji se mogu dobiti optimizacijom dizajna, koristiće se ECB mod rada i verzija algoritma AES-256. Razvoj algoritma izvršen je u programskom okruženju Xilinx ISE 14.2. Ovo programsko okruženje, osim prevođenja koda, izvršava i detaljnu analizu dizajna. Vremenski parametri koji se koriste u ovom radu su parametri koji su dobijeni od razvojnog okruženja.

Struktura dizajna

Algoritam je podeljen na dva dela – jedan za šifrovanje i drugi za dešifrovanje. Ova dva dela algoritma su nezavisna, iako dele jednu zajedničku komponentu – blok za ekspanziju ključa. S obzirom na to da se ekspanzija ključa izvršava jedanput, na početku rada algoritma, i da se dobijeni potključevi rundi ne menjaju tokom rada, već su isti za svaki naredni blok otvorenog teksta, dobijeni potključevi se smeštaju na interni ROM. Njihovo čitanje može se vršiti simultano i nezavisno od blokova za šifrovanje, odnosno za dešifrovanje otvorenog teksta.

Na slici 2 vidi se blok kompletnog kriptomodula, koji sadrži blokove za šifrovanje, dešifrovanje i ekspanziju ključa.



Slika 2 – Blok kriptomodula namenjenog za šifrovanje, dešifrovanje i ekspanziju ključa
Figure 2 – Block of a cryptomodule for encryption, decryption and key expansion
Рис. 2 – Блок криптологического модуля шифрования, дешифрования и расширения (увеличения длины) ключа

S obzirom na to da je ukupna brzina izvršavanja algoritma jednaka maksimalnoj brzini njegove najsporije komponente, razvijanjem ovog algoritma i posebnom vremenskom analizom svake od njegovih celina dobiće se opšti uvid u maksimalnu brzinu kompletnog kriptomodula.

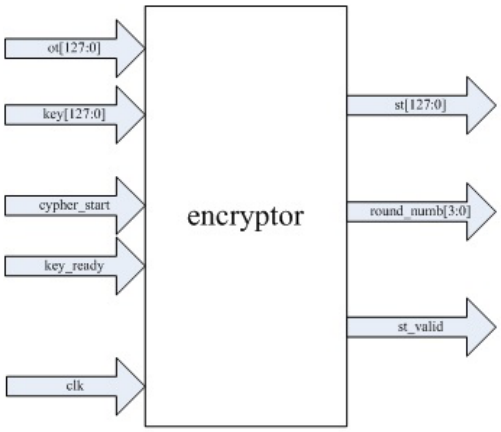
Kako blok za šifrovanje i blok za dešifrovanje imaju istu složenost, u daljem razmatranju posebna pažnja posvećena je samo bloku za šifrovanje.

Blok za šifrovanje

Ulazni i izlazni portovi bloka za šifrovanje prikazani su na slici 3.

Sa slike se može uočiti da se kao ulazni signali (na slici su predstavljeni sa leve strane) osim dva 128-bitna vektora, koji predstavljaju blok otvorenog teksta (*ot*) i potključ za trenutnu rundu (*key*), i ulaznog signala

takta (*clk*), javljaju i dva kontrolna signala (odnosno porta) *key_ready* i *cypher_start*. Ukoliko je signal *key_ready* jednak '1' to znači da se završila ekspanzija ključa i da je vrednost potključa runde validna i u skladu sa brojem runde koji je prosleđen signalom *round_numb* bloku za ekspanziju ključa. Drugi signal *cypher_start* predstavlja kontrolu validnosti otvorenog teksta koji se dovodi na ulaz bloka za šifrovanje. Dakle, ukoliko je vrednost ovog signala jednaka '1' na ulazu *ot[127:0]* nalazi se validan blok otvorenog teksta i proces šifrovanja može da počne.

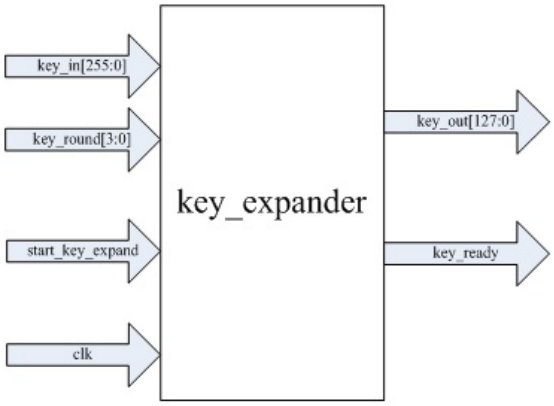


Slika 3 – Blok za šifrovanje
Figure 3 – Encryption block
Рис. 3 – Блок шифрования

Kao rezultat dobijamo blok šifrata na izlazu i kontrolni signal *st_valid* koji označava validnost bloka šifrata. Važno je napomenuti da je blok šifrata na izlazu validan samo jedan takt, što mora biti dovoljno za njegovo prosleđivanje na liniju. Na ovaj način se bloku za dešifrovanje, koji kao svoj ulaz prihvata i *st* i *st_valid* signale iz *encryptor*-a, jasno daje do znanja kada treba da započne proces dešifrovanja.

Blok za ekspanziju ključa

Sledeći blok koji je specifičan jeste blok za ekspanziju ključa. Ovaj deo algoritma dizajniran je tako da u matrici ključeva čuva sve potključeve rundi. Na taj način smanjeno je vreme čekanja validnog potključa na minimum, s obzirom na to da ostvarena veza predstavlja kombinacionu logiku koja ne zavisi od takta (u pitanju je multiplekser kome blok za šifrovanje prosleđuje signale za selekciju potključa). Na slici 4 dat je prikaz bloka za ekspanziju ključa.



Slika 4 – Blok za ekspanziju ključa
Figure 4 – Key expansion block
Рис. 4 – Блок расширения (увеличения длины) ключа

Potrebno je napomenuti da se signal *key_ready* ovog bloka vezuje za signal *key_ready* blokova za šifrovanje, odnosno dešifrovanje i predstavlja znak da je ekspanzija ključa završena i da se na ulazu u multiplekser nalaze validne vrednosti potključeva rundi. Na slici nisu prikazani signali koji se odnose na blok za dešifrovanje (ulazni signal broja runde i izlazni signal ključa za tu rundu) radi jednostavnijeg prikaza.

Brzina izvršavanja VHDL implementacije AES-a

Brzina izvršavanja hardverske realizacija AES algoritma na FPGA čipu zavisi od načina dizajniranja samog algoritma, ali i od vrste FPGA čipa na koji se implementira. Zbog toga će se u daljem radu izvršiti analiza uticaja oba navedena faktora zasebno, tako što će se najpre izvršiti vremenska optimizacija algoritma intervencijom u samom kodu dizajna, a zatim će se dobijena rešenja testirati na dve različite serije FPGA čipova istog proizvođača: *Xilinx Spartan-6* i *Xilinx Kintex-7*.

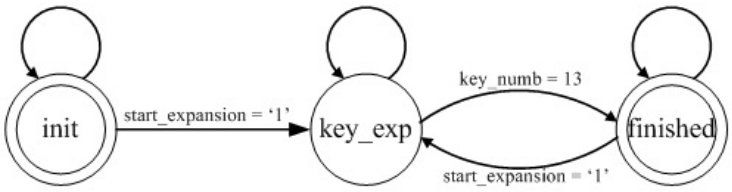
Kao što je rečeno, u ovom radu izvršena je vremenska analiza svakog bloka zasebno, kao i njihova optimizacija.

Vremenska analiza bloka za ekspanziju ključa

Prva verzija *key_expander*-a dizajnirana je pomoću mašine stanja. Inicijalno se posmatraju stanja u kojima se može naći blok za ekspanziju, i prilikom procesa kodovanja svakom stanju dodeli se po jedna perioda takta. To podrazumeva da se kompletna obrada signala u datom stanju izvršava za tačno jedan takt i dolaskom naredne uzlazne ivice takta prelazi se u sledeće stanje. Kod ovakvog pristupa i uz dobro modelovanu

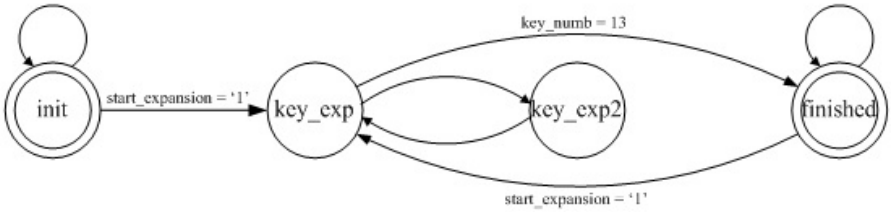
mašinu stanja greške u dizajniranju svedene su na minimum. Loša strana ovog pristupa, sa vremenskog aspekta, ogleda se u različitoj složenosti svakog stanja. Naime, dovoljno je da postoji samo jedno stanje koje podrazumeva kompleksnu i vremenski zahtevnu obradu signala, pa da se maksimalna frekvencija takta čitavog sistema uspori. Upravo taj pristup pokazan je na primeru bloka za ekspanziju ključa.

Prva neoptimizovana verzija dizajna podrazumeva mašinu stanja prikazanu na slici 5. Maksimalna frekvencija ovakvog dizajna za *Spartan-6* čipove iznosi 102.558MHz (Jaćimović, Planić, 2014), dok je kod *Kintex-7* serije ta vrednost 226.827MHz. Lako se uočava da „usko grlo" ovakvog sistema predstavlja stanje *key_exp_state*, jer ono podrazumeva obavljanje nekoliko sukcesivnih operacija za jedan takt, a da bi sve one mogle da postave svoje vrednosti pre naredne uzlazne ivice takta, perioda trajanja takta mora biti duža.



Slika 5 – Mašina stanja neoptimiziranog bloka za ekspanziju ključa
Figure 5 – State machine of the non-optimizing key expanding block
Рис. 5 – Автомат неоптимизированного блока расширения (увеличения длины) ключа

Njegovom optimizacijom *pipeline* tehnikom i razbijanjem na dva podstanja (*key_exp_state1* i *key_exp_state2*) od kojih svako generiše tačno po jedan potključ (slika 6), dobija se znatno poboljšanje u brzini, odnosno maksimalnoj frekvenciji na kojoj može raditi ovakvo kolo, a koja iznosi 154.616MHz za *Spartan-6* (Jaćimović, Planić, 2014), odnosno 306.607MHz za *Kintex-7* čipove. Dakle, prilično jednostavnom intervencijom u kodu dobilo se vremensko poboljšanje nešto veće od 50% u prvom, odnosno oko 36% u drugom slučaju.

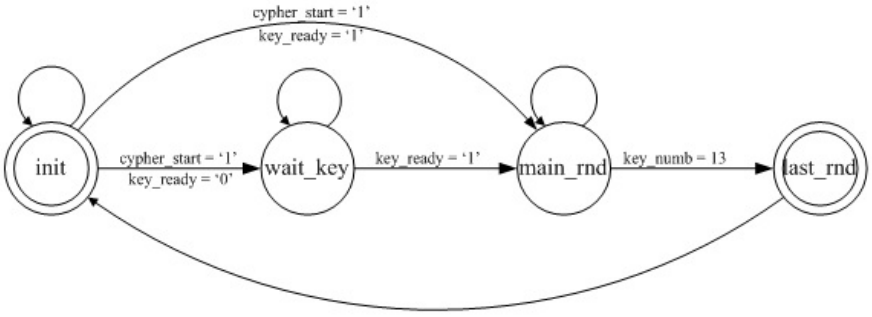


Slika 6 – Mašina stanja optimizovanog bloka za ekspanziju ključa
Figure 6 – State machine of the optimizing key expanding block
Рис. 6 – Автомат оптимизированного блока расширения (увеличения длины) ключа

Najveće poboljšanje vremena izvršavanja dao je tzv. sekvencijalni način kodovanja. Ovim pristupom dizajniramo ponaosob svaki podblok i smeštamo ga i povezujemo u jednu celinu. S obzirom na to da skoro kompletnim izgledom logike upravlja dizajner, a razvojno okruženje manjim delom, uz dobro poznavanje i modelovanje sistema na hardverskom nivou mogu se dobiti optimalna rešenja. Rezultati vremenske analize ovako dobijenog dizajna pokazali su najbolje rezultate – za *Spartan-6* čipove dobijena maksimalna frekvencija koja iznosi 269.920MHz (Jaćimović, Planić, 2014), a za *Kintex-7* 336.655MHz, što znači da ukoliko blok za šifrovanje bude u stanju da podrži ovu frekvenciju, čitav sistem može raditi na jednom taktu.

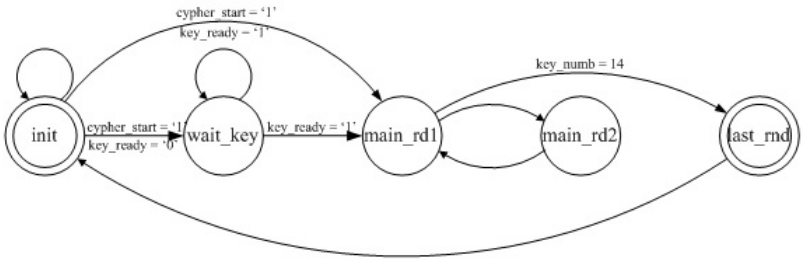
Vremenska analiza bloka za šifrovanje

Prva verzija koda, čija je mašina stanja prikazana na slici 7, implementirana na *Spartan-6* čipu, u stanju je da teorijski podrži rad na frekvenciji od 212.326MHz, odnosno maksimalna brzina između dva uređaja za kriptozaštitu može biti do 27,2 Gbps (Jaćimović, Planić, 2014). Kod čipova iz serije *Kintex-7* dobijaju se (očekivano) bolje performanse i one iznose 274.907 MHz, odnosno 35,2Gbps.



Slika 7 – Mašina stanja neoptimiziranog bloka za šifrovanje
Figure 7 – State machine of the non-optimizing encryption block
Рис. 7 – Автомат неоптимизированного блока шифрования

I ovde je, kao i kod bloka za ekspanziju ključa, „usko grlo" glavna runda (*main_round*) koja ima najkompleksniju kombinacionu logiku u odnosu na ostala stanja (init, wait_key i last_rnd). Uvođenjem *pipeline* tehnike optimizacije dizajna, svako stanje glavne runde razbijamo na dva podstanja. Pri tome, operacije *SubByte* i *ShiftRows*, kao i operacije *MixColumn* i *AddRoundKey* moguće je spojiti u jedan izraz. Na taj način izbegavamo ponavljanje kombinacionih blokova između dva flip-flopa. Mašina stanja ovakvog dizajna prikazana je na slici 8.



Slika 8 – Mašina stanja optimiziranog bloka za šifrovanje
Figure 8 – State machine of the optimizing encryption block
Рис. 8 – Автомат неоптимизированного блока шифрования

Nakon optimizacije, povećanje brzine iznosi oko 16% za *Spartan-6*, odnosno oko 23% u slučaju *Kintex-7* čipova, što dovodi do zaključka da je, u ovom slučaju, alat za sintezu prepoznao i sa većim uspehom izvršio optimizaciju logike već u prvom dizajnu, što nije bio slučaj sa blokom za ekspanziju ključa. Bez obzira na to, direktnom intervencijom u VHDL kodu dobija se nezanemarljivo poboljšanje brzine, pa je sada ostvarena podrška telekomunikacionim sistemima do 31,6 Gbps kod *Spartan-6* (Jaćimović, Planić, 2014), odnosno 43,3 Gbps kod *Kintex-7* čipova.

Najbolji rezultat dobijen je opet sekvencijalnom tehnikom opisivanja logike. Integralno kolo koje implementira logiku na ovaj način podržava rad sa taktom brzine 273.459 MHz, odnosno može vršiti šifrovanje podataka na brzini od 35 Gbps (oko 4,4 GB/s), na *Spartan-6* familiji (Jaćimović, Planić, 2014). Ukoliko bi ovakav dizajn bio implementiran na novijoj *Kintex-7* arhitekturi, maksimalna frekvencija iznosila bi 375.340 MHz, odnosno bila bi pružena podrška za rad telekomunikacionim sistemima brzine do 48 Gbps (oko 6 GB/s). Svi dobijeni rezultati navedeni su u tabeli 1.

Zaključak

U sistemima velikih protoka podataka neophodno je imati efikasnu implementaciju kriptoloških algoritama. Vreme potrebno za šifrovanje, odnosno dešifrovanje podataka često predstavlja usko grlo za takve sisteme. Opravdanje za ovakvu tvrdnju može se naći u činjenici da je svaki sistem brz onoliko koliko je brza njegova najsporija komponenta. U takvim situacijama jedno od najoptimalnijih rešenja je hardverska realizacija algoritma za šifrovanje.

U ovom radu razmatrana je implementacija algoritma na FPGA čipove, jer se ova tehnologija poslednjih godina pokazala kao veoma pogodna za primenu iz više razloga. Na prvom mestu je reprogramabilnost koja nam pruža mogućnost izmene kriptološkog algoritma ili njegovih delova ukoliko se za to pojavi potreba. Druga veoma bitna osobina zastupljenosti FPGA na tržištu je i njihova cena.

Tabela 1 – Uporedni prikaz dobijenih rezultata
Table 1 – Comparative review of the obtained results
Таблица 1 – Полученные результаты в систематизированном виде

Način realizacije algoritma AES-256 (*Version of the implementation of AES-256*)	Maksimalna frekvencija bloka za ekspanziju ključa (*Maximum frequency of the Key-Expansion blok*) (MHz)		Maksimalna frekvencija bloka za šifrovanje (*Maximum frequency of the Encryption blok*) (MHz)		Maksimalna brzina (*Maximum speed*) (Gbps)	
	Spartan-6	Kintex-7	Spartan-6	Kintex-7	Spartan-6	Kintex-7
Neoptimizovano rešenje (*Nonoptimized solution*)	102.558	226.827	212.326	274.907	27,2	35,2
Rešenje sa *pipeline*-om (*Solution with pipeline*)	154.616	306.607	246.935	338.593	31,6	43,3
Strukturno dizajniranje (*Structural design*)	269.920	336.655	273.459	375.340	35	48

Na osnovu dobijenih rezultata može se primetiti značaj koji sam način korišćenja VHDL-a, kao jezika za opis hardvera, ima u pogledu performansi koje se mogu dobiti. Za dobijanje dizajna koji mora da pruži podršku rada na većim frekvencijama, *pipeline* se nameće kao nezaobilazni koncept u procesu pisanja koda. Na taj način dobit u brzini može iznositi i preko 50% za isti algoritam i hardversku podlogu. Strukturnim dizajniranjem taj procenat može biti i znatno veći (tabela 1).

Naravno, kao što se moglo i očekivati, uvođenjem novijih hardverskih čipova (u ovom slučaju serija *Kintex-7*) dobijaju se znatno bolje performanse. Razlog za to ogleda se u činjenici da su novije serije FPGA čipova u pogledu performansi bolje realizovane u odnosu na prethodne. Detaljnije o tome moguće je obavestiti se na zvaničnom sajtu proizvođača (http://www.xilinx.com). Međutim, i pored boljih tehničkih karakteristika samog hardvera, i ovde znatno poboljšanje performansi pruža dobro optimizovan VHDL kod – u konkretnom slučaju nešto manje od 50%.

Dalja istraživanja mogu se voditi u smeru optimizacije prostora koje dizajn zauzima na čipu, ali i smanjenja potrošnje snage koja je potrebna da se jedan takav sistem napaja. Time bi se otvorila vrata za realizaciju pokretnih kriptoloških sistema velikih protoka podataka. Drugi smer u kojem se istraživanja mogu nastaviti predstavlja proučavanje i dizajniranje interfejsne logike koja bi omogućila međusobno povezivanje ovakvih realizacija kriptoalgoritma i testiranje brzine njihovog rada u realnim uslovima.

Literatura

El Mezeni, D., & Novaković, M. 2008. Embedded FPGA Linux sistemi. . U: 16. Telekomunikacioni forum TELFOR 2008, Beograd, November 25-27.. , str.925-928

Farooq, U., Marrakchi, Z., & Mehrez, H. 2012. *Tree-Based Heterogeneous FPGA Architectures*.New York: Springer Science.

Federal Information Processing Standards Publication 197 2001. Preuzeto sa http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Gaj, K., & Chodowiec, P. 2001. Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays. . U: RSA Security Conference, San Francisco, April 8-12. , str.84-99

Good, T., & Benaissa, M. 2005. AES on FPGA from the Fastest to the Smallest. . U: 7th International Workshop Cryptographic Hardware and Embedded Systems – CHES 2005, Edinburgh, August 29–September 1. , str.427-440

Gschwind, M., & Salapura, V. 1995. A VHDL Design Methodology for FPGAs. . U: 5th International Workshop Field-Programmable Logic and Applications - FPL '95, Oxford, August 29–September 1. , str.208-217

Hodjat, A., & Verbauwhede, I. 2003. Speed-Area Trade-off for 10 to 100Gbit/s Throughput AES Processor. . U: 37th Asilomar Conference on Signals, Systems, and Computers, November 9-12. , str.2147-2150

Preuzeto sa http://www.xilinx.com.

Jaćimović, N., & Planić, B. 2014. Vremenska optimizacija hardverske implementacije AES-256 algortima. . U: 22. Telekomunikacioni forum TELFOR 2014, Beograd, Novembar 25-27. , str.427-430

Jevremović, A. 2011. *Integracija sopstvenih kriptoloških sistema u standardnu računarsko-telekomunikacionu infrastrukturu*.Univerzitet Singidunum. Doktorska disertacija.

Karimian, G.H., Rashidi, B., & Farmani, A. 2012. A High Speed and Low Power Image Encryption with 128-Bit AES Algorithm. *International Journal of Computer and Electrical Engineering,* 4(3), pp367-372.

Kilts, S. 2007. *Advanced FPGA Design: Architecture, Implementation, and Optimization*.USA: John Wiley & Sons Inc..

Marković, M. 2012. *Osnove kriptozaštite*.Banja Luka.

McLoone, M., & McCanny, J.V. 2001. High Performance FPGA Rijndael Algorithm Implementations. . U: 3rd International Workshop Cryptographic Hardware and Embedded Systems – CHES 2001, Paris, May 14–16. , str.65-76

Rodríguez-Henríquez, F., Saqib, N.A., & Díaz-Pérez, A. 2003. 4.2 Gbit/s single-chip FPGA implementation of AES algorithm. *Electronics Letters,* 39(15), pp1115-1116. doi:10.1049/el:20030746

Schneier, B. 1996. *Applied Cryptography, 2nd ed*.USA: John Wiley & Sons Inc..

Veinović, M., & Jevremović, A. 2011. *Računarske mreže*.Beograd: Univerzitet Singidunum.

Wollinger, T., Guajardo, J., & Paar, C. 2004. Security on FPGAs: State of the Art Implementations and Attacks. *ACM Transactions on Embedded Computing Systems,* 3(3), pp534-574. doi:10.1145/1015047.1015052

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ШИФРОВАНИЯ В СИСТЕМАХ С ВЫСОКИМ ИНФОРМАЦИОННЫМ ТРАФФИКОМ

ОБЛАСТЬ: компьютерные науки, электроника, телекоммуникации
ТИП СТАТЬИ: оригинальная научная статья
ЯЗЫК СТАТЬИ: сербский

Резюме.

В этой статье анализируется применение алгоритмов шифрования, с учетом требований производительности. Таким образом, обеспечивая поддержку защищенного соединения, оказывается наименьшее возможное влияние на производительность сетевой инфраструктуры. Также в работе разъяснены различия между аппаратным и программным способами реализации алгоритмов шифрования. Приведены основные характеристики FPGA-чипов(ПЛИС) и перспективы использования языка проектирования VHDL при разработке алгоритмов шифрования. Для применения был выбран алгоритм шифрования AES-256, который не только является одним из самых известных на сегодняшний день, но одинаково хорошо зарекомендовал себя как в аппаратных, так и в программных версиях. При разработке использовался пакет программ Xilinx ISE Design Suite, предназначенный для реализации цифровых систем на базе ПЛИС фирмы Xilinx и аппаратно-программные наборы Spartan®-6 FPGA SP605 и Xilinx Kintex-7 FPGA KC705. Полученные результаты поддерживаются всеми устройствами на базе ПЛИС Spartan®-6 или Kintex®-7.

Ключевые слова: *AES, оптимизация архитектуры, VHDL, FPGA, шифрование.*

IMPLEMENTATION OF CRYPTOLOGY ALGORITHMS IN HIGH BIT RATE SYSTEMS

FIELD: Computer Sciences, Electronics, Telecommunications
ARTICLE TYPE: Original Scientific Paper
ARTICLE LANGUAGE: Serbian

Summary:

This paper analyzes the implementation of cryptology algorithms in order to obtain the best possible performance in terms of speed, thus providing support for protected communication between two participants with the smallest possible impact on the performance of the given network infrastructure. The paper explains the difference between the hardware and software implementation of encryption algorithms. It shows the main characteristics of FPGA chips and the advanced technics of the VHDL design language that were used for the implementation of crypto algorithms. The AES-256 encryption algorithm is selected for the implementation since it has proven to be good both in hardware and in software versions. The development environment used is the Xi-

linx ISE Design Suite as well as the development boards Xilinx Spartan SP-605th and Xilinx Kintex KC-705th. All the results correspond to the devices that contain Spartan®-6 and Kintex®-7 chips.

Introduction

At the present time, there is an increasing need for faster exchange of information which must be protected during transmission. However, the "bottleneck" of the entire system is the process of encryption and decryption. Therefore, in search of optimal solutions,there are three main issues: which method of implementation, hardware or software, will give better performance, which technologies are to be used, and which way of the crypto algorithm coding to apply.

Hardware and software cryptosystems

Despite various advantages offered by the software implementation of crypto algorithms, NSA, for example, authorizes only hardware encryption for several reasons such as speed, security, and ease of installation (Schneier, 1996 p.192-194). Thus, in high bit rate systems, hardware implementation will be used, and, as one of the most rewarding technologies of such implementation, the application of programmable chips imposes itself.

FPGA logic circuits

The structure of FPGA (Field Programmable Gate Array) logic gates is made of a large number of identical logic cells subsequently linked to achieve a desired function. The design process is significantly accelerated by using these circuits, so that the implementation can quickly hit the market, and the price of such a development design is lower than other ways of programming hardware. One of the hardware description languages (VHDL, Verilog,etc.) is used for designing logic. Regardless of the language used, the performance of the final design will be most affected by how we use different optimization techniques when writing the code.

Application of FPGA chips in cryptography

Due to commercial needs reflected in the reduction of the costs of designing circuits, and in terms of risk reduction and faster appearance on the market, the use of FPGA programmable circuits has increased, so that now these chips are not only used in the prototype device (in order to be replaced by ASIC chips during production), but also in mainstream production. In crypto-systems, the advantages of such chips are multiple (Wollinger, et al., 2004): algorithm agility, algorithm upload, algorithm modification, throughput and cost efficiency.

VHDL optimization

There are three basic definitions of speed, depending on the context of the problem: throughput (amount of data that can be processed in one cycle), latency (the time that elapses from the receipt of the input signal to obtain the corresponding outputs) and timing (logic retention between sequential elements) (Kilts, 2007).

In order to get a higher signal bandwidth, the concept of pipelines is introduced. This concept involves dividing some of more complex operations to more simpler ones. The idea is to perform the processing of small amounts of data in a shorter period of time. This increases the maximum clock frequency, reduces the synthesis time, and increases the throughput of the system.In order to achieve the design with as little latency as possible, the data flow from the input to the output is necessary to take place as quickly as possible. This is achieved by optimizing the time required for the processing of intermediate results. Clock speed can be improved by using different methods such as adding registers, parallel structures, flatten logical structure, registers balancing and the redistribution of the data flow.

The basic concept of the AES algorithm

The AES (Advanced Encryption Standard), as one of the most popular cipher algorithms today, is well demonstrated both in software and in hardware variants. In this paper we used the AES-256.

A review of past solutions

Today, there are a number of papers dealing with the problem of the implementation of this crypto algorithm into the FPGA or ASIC chips. The authors, in addition to the basic implementation, mainly deal with two types of optimization - optimization area and time optimization.

Own solution

For an effective implementation of the AES algorithm, all optimization techniques of VHDL designs must be taken into account and, as a starting assumption, a possibility of achieving the same and/or better results than those currently available.In this paper, the ECB mode and the AES-256 were used for better observing the results that may be obtained by optimizing the design. The development of the algorithm was executed in the programming environment Xilinx ISE 14.2. The time parameters used in this paper are the parameters obtained from the development environment.

Structure of the design

The algorithm is divided into three parts – encryption block, decryption block and key-expansion block. Since the key-expansion is executed once, at the beginning of the algorithm, and since the resulting round keys do not change during the operation, these subkeys are stored in the internal ROM.

Encryption block

The main purpose of this block is the encryption of the input signals and sending the resulting ciphertext to the output. It is important to note that the ciphertext block on the output is valid for only one clock, which should be enough for its referral to the line.

Key-expansion block

This part of the algorithm is designed so that the matrix of keys keeps all subkeys of rounds. The waiting time for a valid subkey is thus reduced to a minimum.

Speed of the VHDL implementation of the AES

This paper presents a timing analysis of each block separately as well as their optimization.

Time analysis of the key-expansion block

The first version of the key_expander is designed using state machines. The first, unoptimized, design version involves the state machine shown in Figure 5. The maximum frequency of this design is 102.558MHz for the Spartan-6, and 226.827MHz for the Kintex-7 chip. It is easy to observe that the "bottleneck" of such a system is the state key_exp_state. By optimizing it and breaking it into two substates (key_exp_state1 and key_exp_state2), each of which generates exactly one subkey,a significant improvement is achieved in terms of speed and the maximum frequency which is 154.616MHz (Figure 6) (306.607MHz for the Kintex-7). The best results are obtained by the so-called sequential coding – the received maximum frequency is 269.920MHz for the Spartan-6 and 336.655MHz for the Kintex-7.

Time analysis of the encryption block

The first version of the code, the state machine of which is shown in Figure 7, is able to successfully execute at a frequency of 212.326MHz (in the case of the Spartan-6), i.e. the maximum speed between the two crypto devices can be up to 27,2Gbps. In the case of the Kintex-7, the maximum frequency is 274.907MHz (or 35,2Gbps). As the most complex combinational logic is contained in the main round, the introduction of pipeline optimization techniques divides each state of the main round into two substates. The state machine of this design is shown in Figure 8. After the optimization, the increase in speed is about 16% (Spartan-6) or 23% (Kintex-7), which is a theoretically gained support to telecommunication systems up to 31,6Gbps or 43,3Gbps, respectively. The best result was obtained again with the sequential technique of describing the logic and for the Spartan-6 it is 273.459MHz, and data encryption can be done at a speed of 35Gbps (about 4,4Gb/s). In the case of the Kintex-7, those results are 375.340MHz or 48Gbps (or 6GB/s).

Conclusion

This paper considers the implementation of the algorithm on FPGA chips. Based on these results, we can notice the importance that a way of using VHDL has on the performance. To obtain a design that has to support the operation at higher frequencies, the pipeline imposes itself as an indispensable concept in the process of code writing.

Keywords: *AES; Design optimization; VHDL; FPGA; encryption.*

Datum prijema članka / Paper received on / Дата получения работы: 20. 12. 2014.
Datum dostavljanja ispravki rukopisa / Manuscript corrections submitted on / Дата получения исправленной версии работы: 17. 01. 2015.
Datum konačnog prihvatanja članka za objavljivanje / Paper accepted for publishing on / Дата окончательного согласования работы: 19. 01. 2015.