ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

HARDVERSKA IMPLEMENTACIJA SKEIN ALGORITMA ZA HEŠIRANJE

– Master rad –

Kandidat:

Strahinja Jović 3187/2013

Mentor:

doc. dr Zoran Čiča

Beograd, Septembar 2015.

SADRŽAJ

1. UVOD

U vreme digitalnog doba i obavljanja različite vrste poslova korišćenjem Interneta, veliki značaj se pridaje sigurnosti u digitalnom svetu. Neophodno je da komunikacija putem Interneta i njeni korisnici budu zaštićeni od zlonamernih napada, tj. treba da se obezbedi glavni aspekt telekomunikacionih mreža - sigurnost komunikacije. Zato su razvijeni različiti mehanizmi zaštite od zlonamernih napada poput autentifikacije, digitalnog potpisa, autorizacije, šifrovanja komunikacije i dr. Sigurnost današnjih komunikacija se dobrim delom zasniva na kriptografiji gde se u velikoj meri primenjuju heš funkcije, koje su našle primenu u kriptografiji prilikom šifrovanja komunikacije, provere integriteta poruka, autentifikacije [1].

Predmet rada jeste hardverska implementacija Skein algoritma za heširanje, koji je bio jedan od kandidata za novi SHA-3 (*Secure Hash Algorithm-3*) standard u kriptovanju podataka. Obrađene su četiri različite hardverske implementacije datog algoritma.

Za realizaciju implementacije koristi se VHDL programski jezik, a razvoj i verifikacija dizajna vrši se u ISE razvojnom okruženju za FPGA čipove proizvođača Xilinx. Svi projekti (implementacija Skein algoritma za različite dužine izlaza) će biti dati u elektronskoj formi na priloženom CD-u. Rezultat rada, pored implementacije pomenutog algoritma, je i verifikacija i analiza performansi implementacije. Realizovana implementacija će moći da da se primeni u implementacijama zaštite na mrežnim uređajima za postizanje većeg stepena zaštite u mrežnoj komunikaciji, poput Internet rutera.

Ostatak rada je organizovan na sledeći način: Drugo poglavlje daje definiciju i osobine heš algoritama, njihovu primenu i poznate predstavnike. Potom sledi detaljan opis verzije Skein heš algoritma u trećem poglavlju. Četvrto poglavlje se bavi opisivanjem realizovane implementacije, na primeru verzije koda Skein 256-256, tj. Skein sa 256-tobitnom heš vrednošću. Nakon opisa dizajna po principu crne kutije, detaljno su opisane funkcije i procedure napisane za realizaciju koraka Skein algoritma, a potom i kod koji vrši celokupno heširanje kao i razlike u kodu za ostale dužine heš vrednosti. U petom poglavlju biće prikazan postupak verifikacije Skein modula. Šesto poglavlje sadrži zaključna razmatranja o realizovanom algoritmu.

2. HEŠ ALGORITMI

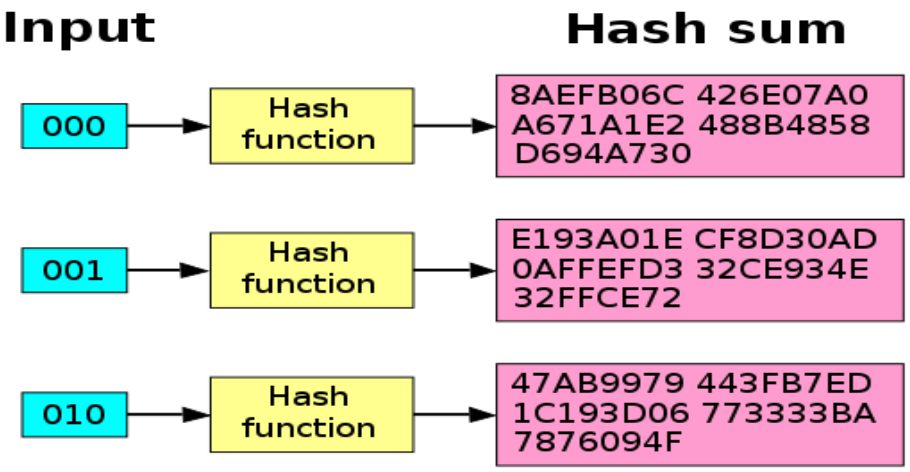
2.1. Definicija heš algoritma i njegova primena

Heš funkcije transformišu ulazni podatak promenjive dužine u numerički heš (sažetak) određene fiksne dužine. Ulazni podaci nad kojima se primenjuje heš funkcija se nazivaju poruka, a vrednost koja se dobija nakon primene funkcije se zove heš vrednost ili sažetak.

Heš funkcije razvijene su za potrebe sigurnosti digitalnog potpisa. Danas, pored pomenutog, pronašle su primenu u mnogim oblastima i razvijen je veliki broj heš algoritama koji omogućavaju proveru integriteta i ispravnosti poslatih podataka, enkripciju ključeva, brzog traženja poruka u bazi podataka, pronalaženja dupliranih zapisa (svaki podatak se propusti kroz heš funkciju i pronađu se podaci koji imaju istu vrednost heša) ili mapiranja poruka sa indeksima tabele [2]. Heš funkcije predstavljaju i efikasan način čuvanja pristupnih lozinki u bazama podataka i sistema autentifikacije kada je neophodno da samo korisnik zna lozinku. Takođe, heš algoritmi se mogu koristiti za generisanje novih ključeva i šifri iz jednog sigurnog ključa i šifre.

Veoma značajna oblast u kojoj heš algoritmi nalaze primenu je kriptografija. Kriptografske heš funkcije se primenjuju u savremenim telekomunikacijama gde je zaštita podataka od zlonamernih napadača od ključne važnosti. Cilj je na neki način osigurati komunikaciju učesnika ako se zna da napadač može imati pristup podacima koji se razmenjuju tokom komunikacije. U polju kriptografije, heš ima primenu u procesu autentifikacije korisnika, provere integriteta poruke, potvrdi lozinke, digitalnom potpisivanju dokumenata i generisanju statistički slučajnih nizova podataka.

Jak heš algoritam može brzo da generiše heš iz ulaznih podataka i da pritom bude praktično nemoguće napadačima da otkriju originalne ulazne podatke na osnovu poznavanja vrednosti izlaznog heša. Takođe, verovatnoća kolizije, gde se za dve ili više različitih poruka generiše identičan heš na izlazu, mora biti zanemarljivo mala. Svaka promena ulazne poruke, ma koliko bila mala, treba da izazove drastičnu promenu izlaznog heša.



Slika 2.1.1. Poruke i njihove heš vrednosti nakon primene SHA-1 algoritma [3]

Za primenu u kriptografiji, glavne osobine koje bi idealna heš funkcija trebalo da poseduje su sledeće:

- jednostavno izračunavanje heša za bilo koju poruku,
- izlaz konstantne dužine za svaki ulaz proizvoljne dužine,
- skoro je nemoguće rekonstruisati ulaz na osnovu poznatog heša,
- skoro je nemoguće pronaći dva ulaza koji daju isti heš.

Stoga, nije lako napraviti novu kriptografsku heš funkciju. Ona bi trebalo da se ponaša što je više moguće kao random funkcija, ali da i dalje bude deterministička. Na slici 2.1.1. se može videti kako promena u samo jednom bitu drastično menja vrednost heša.

2.2. Razvoj heš algoritama

SHA (*Secure Hash Algorithm*) algoritmi za izračunavanje sažetaka poruke su algoritmi koje je američki Nacionalni institut za standarde i tehnologiju - NIST (*National Institute of Standards and Technology*) prihvatio kao kriptografski sigurne funkcije za izračunavanje sažetaka poruke. Specifikacije SHA algoritama javno su objavljene u publikaciji SHS koju izdaje FIPS-a (*U.S. Federal Information Processing Standard*) [4].

2.2.1. Algoritam SHA-0

Krajem osamdesetih i početkom devedesetih godina, najčešće korišćeni algoritam je bio MD5. Institut NIST objavljuje poboljšanu verziju ovog algoritma 1993. godine pod imenom SHA-0, zbog sumnje u sigurnost MD5 algoritma. Prvi SHA algoritam za izračun sažetaka poruke objavljen je pod nazivom SHA (kasnije je naziv SHA zamijenjen nazivom SHA-0). Struktura ovog algoritma ima sličnosti sa MD5. Za SHA-0 algoritam vrlo brzo je ustanovljeno da ova 160-bitna funkcija sadrži značajne manjkavosti. Mane prvog SHA algoritma su ispravljene dodavanjem samo jedne operacije.

2.2.2. Algoritam SHA-1

SHA-1 je 160-bitni algoritam za izračun sažetka poruke koji je nastao 1995. godine revizijom algoritma SHA-0 i čija struktura takođe ima sličnosti sa algoritmom MD5. Ovu funkciju razvila je američka agencija za nacionalnu sigurnost koja je postala prihvaćena širom sveta kao standard za autentifikaciju digitalnih podataka i digitalnog potpisa. Početkom 2005. godine objavljen je uspešan napad kolizijom na skraćenu verziju SHA-1 algoritma sa 53 runde (od ukupno 80 izvorno korišćenih rundi). Ove slabosti ukazuju da SHA-1 algoritam ipak nije onoliko siguran koliko se prvobitno mislilo [4][5].

2.2.3. Algoritam SHA-2

SHA-2 je naziv za više sličnih algoritama za izračunavanje sažetaka poruka. Osnovna razlika je u dužinama ulaznog bloka te u tom smislu grupu SHA-2 čine sledeći algoritmi:

- SHA-256 – algoritam koji ulaznu poruku deli na blokove dužine 256 bitova (32-bitne riječi) i računa heš dužine 256 bita,
- SHA-512 – algoritam koji ulaznu poruku deli na blokove dužine 512 bitova (64-bitne riječi) i računa heš dužine 512 bita,
- SHA-224 – skraćena verzija SHA-256 algoritma za proračun heša dužine 224 bita,
- SHA-384 – skraćena verzija SHA-512 algoritma za proračun heša dužine 384 bita.

Algoritam SHA-2 takođe je, 2001. godine, osmislila NSA (*National Security Agency*). SHA-2 algoritam osmišljen je sa namerom da ukloni neke potencijalne, tada već poznate, ranjivosti SHA-1 funkcije. SHA-2 iako je za sada bezbedan algoritam ipak ima, aritmetički, sličnosti sa SHA-1 algoritmom, čime se u budućnosti može javiti rizik od probijanja SHA-2 algoritma. Time bi nastupili sigurnosni problemi sa brojnim aplikacijama koje koriste i oslanjaju se na sigurnost SHA-2 algoritma. Sa druge strane, svetska industrija u tom slučaju ne bi imala na raspolaganju jedan od osnovnih alata neophodnih za razvoj i sigurno korištenje informacionih sistema [6].

2.2.4. Algoritam SHA-3

Sve uspješniji pokušaji kriptoanalize algoritama za računanje sažetaka poruka podstakli su NIST da počne sa razvojem novog SHA algoritma. Krajem 2007. godine NIST je objavio javni konkurs za izbor novog algoritma za izračunavanje sažetaka poruka.

Dok je SHA-1 i grupu SHA-2 algoritama osmislila NSA, buduća SHA-3 funkcija trebala je biti rezultat udruženog znanja svetske kriptografske zajednice. Osnovni zahtev bio je da predloženi algoritmi (kandidati) moraju u potpunosti zameniti SHA-2 algoritme. Osim same kriptografske sigurnosti algoritama, tokom takmičenja analizirana je i njihova efikasnost, odnosno brzina obrade podataka za različiti dizajn [7].

Godine 2010. objavljena je konačna lista od pet finalista među kojima je Skein algoritam zauzeo mesto kao jedan od najvećih favorita za novi SHA-3 algoritam. Primena i potencijal Skein heš funkcije su velike. Krajem 2012. godine, algoritam Keccak izabran je kao pobednik takmičenja, čime je proglašen algoritmom SHA-3.

3. SKEIN ALGORITAM

Skein je jedan od heš algoritama koji su ušli u finalnu fazu NIST SHA-3 takmičenja. Glavni autori ovog algoritma su Niels Ferguson i Bruce Schneier [8]. SHA-3 algoritmi treba da obrađuju ulaznu poruku promenljive dužine i da za odgovarajuću poruku računaju heš dužine 224, 256, 384, 512 bita (tj. da u potpunosti mogu da zamene SHA-2 algoritam). Skein može da radi sa tri različite inicijalne vrednosti 256, 512, ili 1024 bita. Dužina izračunatog heša za sve tri inicijalne vrednosti kod Skein algoritma je proizvoljna, tako da Skein algoritam u potpunosti može da zameni SHA-2 algoritam. Napomenimo da će radu će biti obradjen Skein algoritam za inicijalnu vrednost ulaznog bloka od 256 bita, za sve četiri vrednosti izlaznog heša .

Skein algoritam se sastoji od tri komponente: Threefish bloka, bloka za ulančavanje podataka – UBI bloka (*Unique Block Iteration*) i opcionog bloka. Preko opcionog bloka u Skein algoritam je moguće ubaciti dodatne argumente (poput javnog ključa, argumenta personalizacije, konfiguracije, itd.). Prve dve komponente biće detaljno opisane, dok će blok sa specijalnim funkcijama biti preskočen jer nije predmet ovog rada.

Skein algoritam može da obradi ulaznu poruku - M maksimalne dužine 2^{99} - 8 bita.

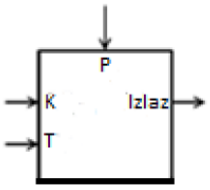
3.1. Threefish

Threefish je blok koji se nalazi unutar UBI bloka, i obrađuje podatke u blokovima različitih dužina. On ulazne podatke, u blokovima dužine 256, 512 ili 1024 bita, obrađuje velikim brojem jednostavnih koraka. Ulazna poruka je podeljena u N_w reči dužine 64 bita. Stoga za različite inicijalne vrednosti ima različit broj N_w reči.

Ulazni podaci algoritma Threefish su sledeći:

- K – ključ (iste dužine kao blok poruke 256, 512 ili 1024 bita),
- T – Tweak vrednost, dužine 128 bita,
- P – ulazni blok podataka ($P=p_0, p_1, \ldots$).

Tweak će biti detaljno obrađen na kraju poglavlja. Dužina ključa mora biti jednaka dužini ulaznog bloka (256, 512 ili 1024 bita).



Slika 3.1.1. Threefish blok dijagram

Threefish se sastoji od operacija na 64-bitnim podacima, poput dodavanja podključa, MIX funkcije i permutacije.

Algoritam Threefish bloka je sastavljen iz rundi. Koriste se 72 runde za Skein 256 i Skein 512, dok Skein 1024 koristi 80 rundi. Broj rundi je označen sa N_r. U svakoj rundi obavlja se funkcija MIX i permutacija. Svake četvrte runde dodaje se novi podključ.

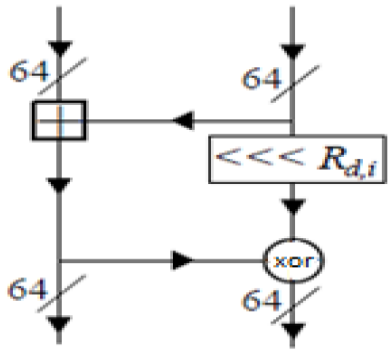
Veličina bloka/Ključa	Broj reči N_w	Runde N_r
256	4	72
512	8	72
1024	16	80

Tabela 3.1.1. Broj rundi i ukupan broj reči za različitu dužinu bloka

Zavisnost N_w i N_r od dužine inicijalne reči data je u tabeli 3.1.1. Broj runde koja se izvršava označava se sa *d* koja uzima vrednost [0, *Nr*-1].

3.1.1. MIX i Permutacija

Na ulazu u Threefish algoritam, poruci se dodaje podključ. Dodavanje podključa se nastavlja u svakoj četvrtoj rundi. Generisanje i dodavanje podključa biće objašnjeno u sledećem poglavlju. Sledeći korak je MIX operacija koja je prikazana na slici 3.1.1.1.



Slika 3.1.1.1. Blok dijagram MIX funkcije

Svaki MIX uzima dve 64-bitne reči kao ulaz, i daje dve 64-bitne reči kao izlaz. MIX operacija koristi jednostavno sabiranje po modulu 2^{64}, pomeranje ulevo (tj. rotacija ulevo) za *R* konstantu i XOR operaciju. Znak + predstavlja operaciju sabiranja po modulu 2^{64}, a znak <<< rotaciju ulevo. U *d*-toj rundi algoritma Threefish, *j*-ta funkcija MIX (oznaka $M_{ixd,j}$) podatke transformiše sledećim algoritmom:

$Y_0 = (X_0 + X_1) \bmod 2^{64}$

$Y_1 = (X_1 <<< R_{(d \bmod 8),j}) \; XOR \; Y_0$

Vrednosti parametra *R* su različite za različite inicijalne vrednosti Skein algoritma, i ponavljaju se na svakih osam rundi. Vrednosti *R* parametra date su u tabeli 3.1.1.1.

N_w	4		8				16							
d/j	0	1	0	1	2	3	0	1	2	3	4	5	6	7
0	14	16	46	36	19	37	24	13	08	47	08	17	22	37
1	52	57	33	27	14	42	38	19	10	55	49	18	23	52
2	23	40	17	49	36	39	33	04	51	13	34	41	59	17
3	05	37	44	09	54	56	05	20	48	41	47	28	16	25
4	25	33	39	30	34	24	41	09	37	31	12	47	44	30
5	46	12	13	50	10	17	16	34	56	51	04	53	42	41
6	58	22	25	29	39	43	31	44	47	46	19	42	44	25
7	32	32	08	35	56	22	09	48	35	52	23	31	37	20

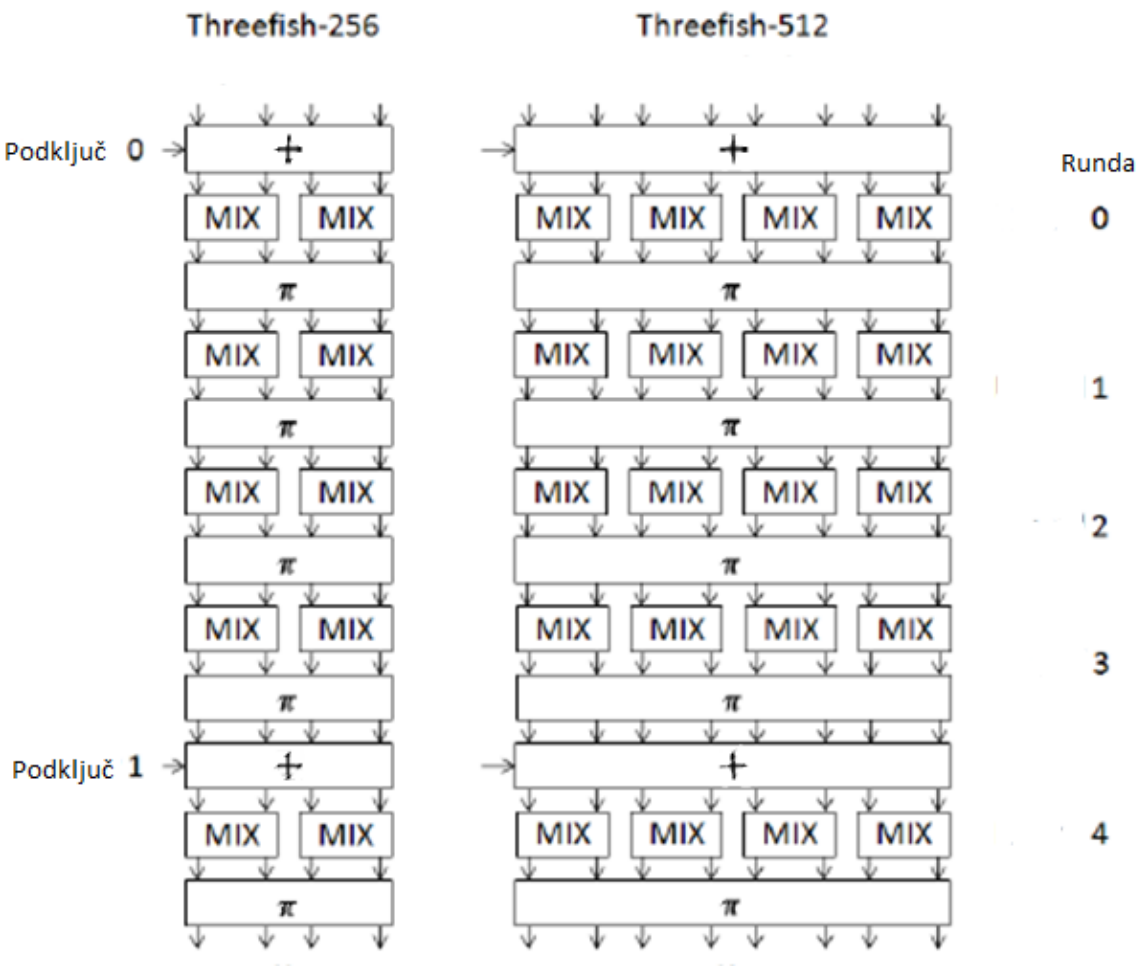
Tabela 3.1.1.1. Faktori rotacije bitova u zavisnost broja N_w

Sledeći korak je premutacija koja nije ništa drugo do zamena pozicija N_w reči. Vrednosti permutacija su iste za svaku rundu, ali zavise od inicijalnih vrednosti. Vrednosti permutacija prikazane su u tabeli 3.1.1.2.

N_w / i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
4	0	3	2	1	-				-				-			
8	2	1	4	7	6	5	0	3	-				-			
16	0	9	2	13	6	11	4	15	10	7	12	3	14	5	8	1

Tabela 3.1.1.2. Funkcija permutacije

Nakon svih završenih rundi radi se još jedno dodavanje podključa.



Slika 3.1.1.2. Threefish 256 i 512 prikazan za 4 runde

Na slici 3.1.1.2. su prikazane strukture Threefish 256 i 512 gde se izvršavaju 4 runde (od 72). Svaka vertikalna strelica predstavlja 64-bitni podatak, dok horizontalne linije predstavljaju dodavanje podključa.

3.1.2. Generisanje podključa

Dodavanje podključa vrši se N_r /4 +1 puta, gde je bitska dužina podključa jednaka inicijalnoj dužini. Podključevi algoritma Threefish računaju se iz tri parametra:

- *K* - ključa ($K=k_0$ *II* k_1 *II...* *II* k_{Nw-1}),
- *T* - Tweak (zapisan kao dve reči t_0 i t_1),
- *C240* konstante (*C240=0x1BD11BDAA9FC1A22*).

Pored ove tri vrednosti biće nam potreban i redni broj dodavanja podključa.

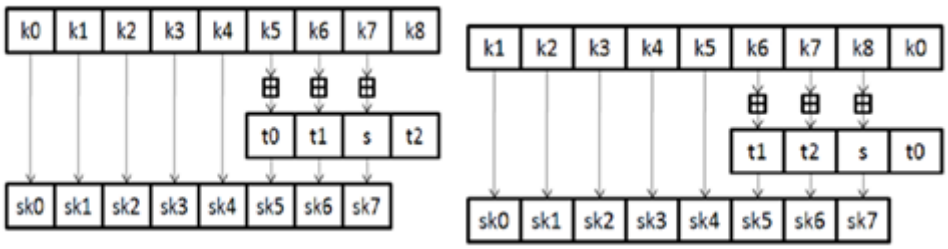
Za algoritam Threefish koji je ostvaren na način da ne traži učitavanje ključa, vrednost *K* je trivijalna (niz bita vrednosti nula). Konstanta *C240* osigurava da, nezavisno o ulaznom ključu, izračunati podključevi ne mogu biti trivijalni. Zapravo za generisanje podključa koriste se proširene vrednosti ključa i promenljive Tweak, kao i redni broj podključa (s = 0,1,..., N_r/4). Algoritam Threefish u svakoj četvrtoj rundi koristi novi podključ. Algoritam za pravljene podključeva je

definisan u tabeli 3.1.2.1. Iz tabele se može videti i algoritam za pravljenje proširenog ključa kao i proširene Tweak vrednosti.

Algoritam za podključ		
Promenljive:	$K = k_0, \ldots k_{Nw-1}$ - ključ	
	$T = t_0, t_1$ - Tweak	
	$sK_s = sk_{s,0},\ldots sk_{s,Nw-1}$ - podključ	
Algoritam:	$k_{Nw} = C240\ XOR\ \sum k_i$	
	$t_2 = t_0\ XOR\ t_1$	
	$sk_{s,i} = k_{(s+i)} \bmod (Nw+1)$	$i = 0,\ldots,N_w\text{-}4$
	$sk_{s,i} = k_{(s+i)} \bmod (Nw+1) + t_{s \bmod 3}$	$i = N_w\text{-}3$
	$sk_{s,i} = k_{(s+i)} \bmod (Nw+1) + t_{(s+1) \bmod 3}$	$i = N_w\text{-}2$
	$sk_{s,i} = k_{(s+i)} \bmod (Nw+1) + s$	$i = N_w\text{-}1$

Tabela 3.1.2.1. Algoritam za generisanje podključa [8]

Generisanje podključeva može jednostavno biti realizovano u hardveru pomoću pomeračkih registara, što je prikazano na slici 3.1.2.1. Slika je prilagođena algoritmu Skein 512-512.

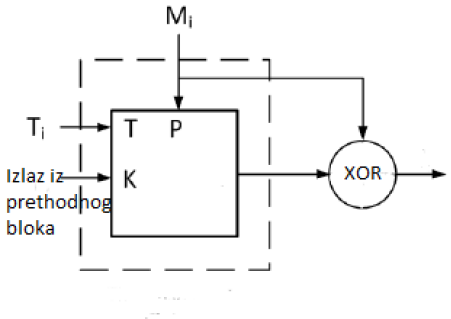


Slika 3.1.2.1. Inicijalna vrednost podključa (levo), i vrednost sledećeg podključa (desno)

Prošireni ključ i Tweak su podeljeni u 64-bitne reči i sačuvani u registrima. Sa svakom novom potrebom za generisanjem podključa, vrednosti ovih registarara se pomeraju levo za 64 bita, dok se vrednost *s* inkrementira. Zatim se te vrednosti upisuju u podključ, gde + predstavlja sabiranje po modulu 2^{64}.

3.2. Ulančavanje podataka - UBI blok

UBI blok ili blok za jedinstveno ulančavanje podataka koristi Threefish za obradu podataka. Nad izlazom svakog Threefish bloka se vrši operacija XOR, zajedno sa ulaznom blokovnom porukom. Zatim se dobijeni podatak zajedno sa jedinstvenom Tweak vrednošću dovodi na ulaz sledećeg bloka. UBI blok tako osigurava jedinstvenost svake iteracije. UBI blok je odgovoran za svaki argument koji se obrađuje u Skein funkciji.

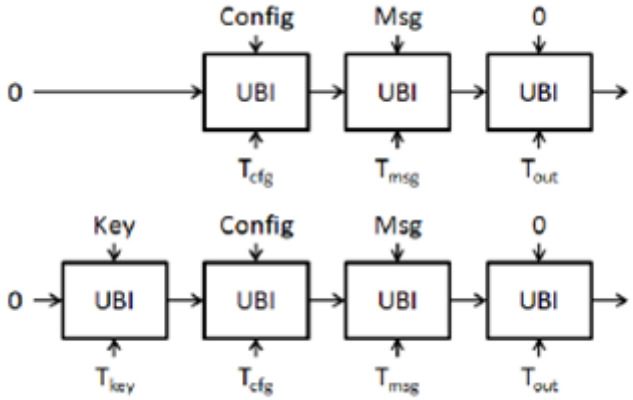


Slika 3.2.1. Jedan lanac u UBI bloku

Ulazni podaci UBI bloka su sledeći:

- G – početna ulazna vrednost (blok podataka dužine N_b bajtova),
- M – niz podataka proizvoljne dužine,
- T – Tweak.

Skein koristi konstrukciju sa argumentnim sistemom Tweak. Svaki UBI blok procesira poruku i ključeve sa jednistvenom promenljivom Tweak, tj. različite vrednosti Tweak promenljive se koriste za različite konfiguracije. Stoga će Tweak vrednost uvek biti jedinstvena za svaki blok.



Slika 3.2.2. Obavezni UBI blokovi. Varijanta bez dodavanja ključa od strane korisnika (gore) i sa dodavanjem ključa (dole) [8]

Obavezni UBI blokovi kod Skein heširanja su konfiguracioni blok, blok za obradu poruke, i izlazni blok, koji su detaljno objašnjeni. Ukoliko korisnik želi da unosi i jedinstveni ključ, potreban je i blok za obradu ključa. Osnovni princip dizajna UBI blokova prikazan je na slici 3.2.2.

3.2.1. Konfiguracioni UBI blok

Konfiguracioni UBI je obavezan deo UBI blokova pri Skein procesiranju. On kao ulaznu poruku koristi konfiguracioni string C, niz nula za ključ (varijanta bez ključa) i konfiguracioni Tweak. Konfiguracioni string C je 256-bitna poruka koja je definisana u specifikaciji. Izlaz iz konfiguracionog UBI bloka se koristi kao ulaz u UBI blok za obradu poruke.

Ukoliko se Skein funkcija koristi samo kao heš algoritam, konfiguracioni UBI se može zameniti sa jednostavnim inicijalnim vektorom - *IV* (*Initalization Vector*) koji se dovodi na ulaz UBI bloka za obradu poruke. Vrednosti *IV* za različite dužine ulaznog bloka date su u tabeli 3.2.1.1.

Skein 256-256			
FC9DA860D048B449	2FCA66479FA7D833	B33BC3896656840F	6A54E920FDE8DA69
Skein 512-512			
4903ADFF749C51CE	B0FEF9CCFD84FAA4	9D77DD663D770CFE	D798CBF3B468FDDA
1BC4A6668A0E4465	7ED7D434E5807407	548FC1ACD4EC44D6	266E17546AA18FF8
Skein 1024-1024			
D593DA0741E72355	15B5E511AC73E00C	5180E5AEBAF2C4F0	03BD41D3FCBCAFAF
1CAEC6FD1983A898	6E510B8BCDD0589F	77E2BDFDC6394ADA	C11E1DB524DCB0A3
D6D14AF9C6329AB5	6A9B0BFC6EB67E0D	9243C60DCCFF1332	1A1F1DDE743F02D4
0996753C10ED0BB8	6572DD22F2B4969A	61FD3062D00A579A	1DE0536E8682E539

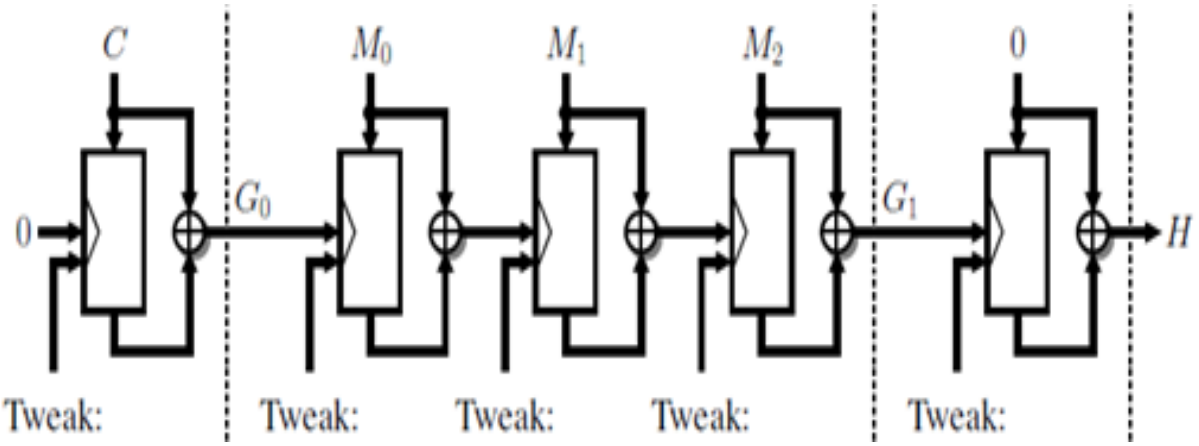
Tabela 3.2.1.1. IV za različite dužine ulaznog bloka

Moguće je proračunati i sve ostale *IV* koje pokrivaju slučajeve različite dužine izlaznog heša kod Skein *dužina_ulaza* - *dužina_izlaza* algoritma.

3.2.2. UBI blok za obradu poruke

UBI blok za obradu poruke (MSG UBI blok) najlakše je objasniti preko slike 3.2.2.1. Ulazni podaci funkcije UBI bloka za obradu poruke su sledeći:

- *G* - izlaz konfiguracionog UBI bloka,
- *M* - ulazna poruka koja se obrađuje - (može biti dužine 2^{99} - 8 bita)
- *T* - 128-bitna Tweak vrednost



Slika 3.2.2.1. Grafički prikaz ulančavanja ulazne poruke *M* (dužine 3*veličine bloka) [8]

Tweak vrednost se računa za svaki deo UBI bloka za obradu poruke, što znači da će svaki Threefis kod UBI bloka za obradu poruke imati specifičnu Tweak promenljivu.

Dužina poruke M određuje koliko blokova Threefis funkcije će biti pozvano u lanac za obradu poruke. Pretpostavka je da je dužina poruke k puta veća od inicijalne dužine poruke koja se obrađuje. Poruka se deli u k blokova (dužine jednake incijalnoj dužini poruke tj. 256, 512, ili 1024 bita), počevši od M_0,.. M_{k-1} (sa slike k=3), gde se svaki blok poruke M_k dovodi na ulaz Threefisha. Ukoliko k nije ceo broj (dužina ulazne poruke u bitima nije višestruki umnožak dužine bloka) neophodno je uraditi padding poruke. Izlaz prethodnog UBI bloka za obradu poruke predstavlja ulaz za sledeći blok UBI bloka za obradu poruke.

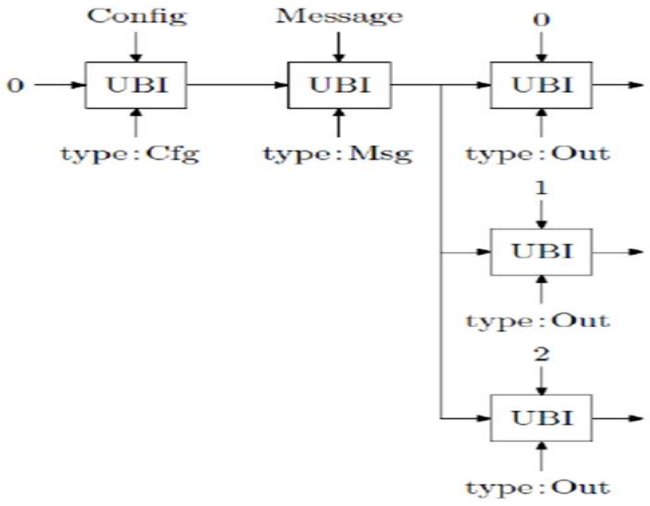
Padding (dopuna) predstavlja dodavanje ekstenzije poruci da bi tako novonastala poruka mogla da se podeli na celobrojni broj blokova. Padding poruke radi se na sledeći način: poruka se podeli u blokove (ukoliko je dužina poruke u bitima veća od inicijalne vrednosti bloka za obradu poruke), tako da se za zadnji blok poruke vrši dopuna. Ukoliko je broj bita zadnjeg bloka poruke umnožak od 8, poruka se dopunjava nulama dok ne dostigne dužinu inicijalne vrednosti. Ukoliko broj bita zadnjeg bloka poruke nije umnožak od 8, onda se na poziciji bita naveće težine u reči upiše vrednost 1 za prvi bit koji nedostaje, a zatim se, ukoliko je potrebno, poruka dopunjava nulama dok ne dostigne dužinu inicijalne vrednosti.

3.2.3. Izlazni UBI blok

Izlazni UBI blok (UBI OUT blok) je obavezan deo UBI blokovnog sistema kod Skein algoritma, i koristi se u finalnoj fazi obrade algoritma. Konačan rezultat algoritma Skein (heš) računa se primenom funkcije izlazne transformacije koja dodatnim menjanjem podataka osigurava izlaznom nizu karakteristike slučajnog niza i obrađuje rezultat na unapred definisanu dužinu (algoritam Skein omogućuje računanje heša proizvoljne dužine).

Parametri izlazne transformacije su sledeći:

- G – poslednja vrednost UBI MSG funkcije, tj. izlaz iz UBI bloka za obradu poruke
- N_0 – nulti element N_w, 64bita



Slika 3.2.3.1. Ponavljanje izlazne transformacije u paraleli za izračun sažetaka većih dužina [8]

Potreban broj UBI blokova u završnoj fazi računa se na sledeći način: N_b - dužina bloka u bajtovima (32 za blok od 256, 64 za blok od 512, 128 za blok od 1024). Ukoliko heš nije veći od $8*N_b$, koristi se samo jedan izlazni UBI blok za finalnu obradu heš algoritma. Ukoliko je heš veći od $8*N_b$, onda je potrebno $[N_o/(8* N_b)]$ UBI blokova za obradu heš algoritma u završnoj fazi.

UBI OUT blokovi rade sa istim ulaznim podatkom *G* (izlaz zadnjeg MSG UBI bloka), a rezultirajući heš se dobija povezivanjem izlaza UBI OUT blokova u paralelu. Jedina vrednost koja se menja je N_0 (0 za prvi blok u paraleli, 1 za drugi...). Na slici 3.2.3.1. je prikazan slučaj kada su nam potrebna tri završna UBI bloka za dobijanje Skein heša određene dužine.

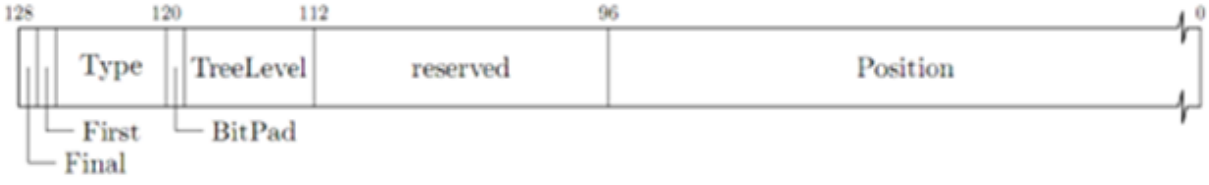
3.3. Tweak

Svaki UBI blok procesira poruku i ključeve sa jedinstvenom Tweak vrednošću. Tweak je 128-bitni podatak koji se koristi za enkripciju svakog bloka. On sadrži podatke koji se UBI blok poziva (konfiguracioni, UBI blok za obradu poruke, izlazni), rednom broju bloka kod UBI bloka za obradu poruke, i ukupnom broju obrađenih bita. Time se može reći da je Tweak jedinstven za svaki ulazni blok.

Ime polja	Pozicija	Opis
Position	0 - 95	Ukupan broj bajtova koji su obrađeni (uključuje se trenutni blok).
Reserved	96 - 111	Rezervisano za buduće potrebe. Postavljeno na 0.
TreeLevel	112 - 118	Setuje se kod opcije Tree heša, 0 za ostale.
BitPad	119	Postavljeno na 1 ukoliko se vrši padding, u suprotnom 0.
Type	120 - 125	Tip za Tweak (Zavisno od stanja UBI bloka).
First	126	Postavljeno na 1 za prvi blok u UBI lancu, 0 za ostale.
Final	127	Postavljeno na 1 za zadnji blok u UBI lancu, 0 za ostale.

Tabela 3.3.1. Parametri Tweak polja

Parametri Tweak promenljive i njihove vrednosti i pozicija dati su u tabeli 3.3.1, dok je raspored polja Tweak promenljive prikazan na slici 3.3.1.



Slika 3.3.1. Raspored bita kod Tweak promenljive [8]

3.3.1. Podešavanje polja kod Tweak promenljive

Potrebno je definisati sva polja promenljive Tweak:

- *Final* (bit 127) = 1 ukoliko je u pitanju zadnji blok poruke koji se obrađuje,

- *First* (bit 126) = 1 ukoliko je u pitanju prvi blok poruke koji se obrađuje,
- *Type* (bits 120-125) = 0/4/48/63. Podešava se na vrednost u zavisnosti od UBI bloka,
- *P* (bit 119) = 1 ukoliko se vrši padding poruke,
- *Tree Level* (bits 112-118) = 0,
- *Reserved* (bits 96-117) = rezervisano, postaviti na 0,
- *Position* (bits 0-95) = ukupan broj bitova *M* poruke koji su obrađeni do sad (ne računaju se biti padding-a).

Pošto je Tweak uglavnom ispunjen nulama, Tweak se u kodu najlakše može implementirati na sledeći način:

Svi biti Tweak promenljive su postavljeni na nula. Tweak sa sobom nosi i informaciju da li je Trefish blok kod UBI bloka za obradu poruke prvi ili poslednji u lancu postavljanjem *First* ili *Final* polja na 1. Ukoliko se poruka obrađuje uz pomoć samo jednog Threefish bloka, oba ova polja su postavljena na 1. Za određen tip Tweak promenljive doda se *Type argument* (pozicija 120-125 bita). Najčešći argumenti *Type* polja kod Tweak promenljive su:

- $T_{key} = 0$,
- $T_{cfg} = 4$,
- $T_{msg} = 48$,
- $T_{out} = 63$.

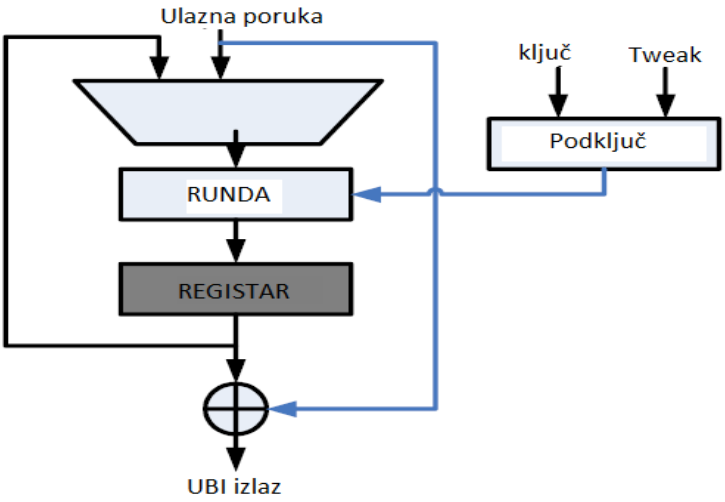
Za svaki obrađeni blok poruke vrednost *Position* se inkrementira za dužinu obrađene poruke.

4. IMPLEMENTACIJA SKEIN ALGORITMA

U ovom poglavlju biće opisan programski kod napisan za potrebe implementacije algoritma. Kod je pisan u VHDL programskom jeziku. Detaljan opis sledi za verziju čija je heš vrednost dužine 256 bita (Skein 256-256), a potom će biti navedene razlike u delovima koda za Skein 256-224, Skein 256-384 i Skein 256-512. Dizajn opisan u ovom poglavlju ne vrši „padding“ proceduru. Dizajn se najpre može predstaviti crnom kutijom sa interfejsima. U narednim poglavljima biće opisani odabir modela, interfejsi dizajna, stanja u kojima se dizajn može naći, tipovi promenljivih, korišćene procedure, kao i unutrašnjost crne kutije.

4.1. Odabir modela

Postoje više različitih modela za implementaciju Skein algoritma. Autor koda se odlučio za iterativni model u kome se zahteva da su sve runde, povezane na red, sa registrovanjem podataka između rundi. Podaci će biti registrovani i na ulazu i izlazu bloka.



Slika 4.1.1. Iterativna implementacija Skein algoritma

Interativni model prikazan na slici 4.1.1. prikazuje bazičnu strukturu iterativnog modela prilagođen Threefish bloku kod Skein algoritma. Ovaj model pored registra za smeštanje podataka koji se obrađuje kod Skein funkcije koristiće i dodatni registar u kome će se čuvati vrednost podključa. Ulazna poruka u modelu se koristi samo u nultoj rundi i nakon završenih rundi za krajnju operaciju XOR. Poseban blok sa podključem je zadužen za pravilno generisanje podključa koji se ubacuje u svakoj četvrtoj rundi. Takođe u modelu mora da se nalazi i kontrolna logika (brojač) za praćenje stanja (praćenje broja obrađenih rundi i kontrola operacija u pojedinoj rundi).

4.2. Interfejsi

Dizajn sadrži ulazne i izlazne interfejse. Ulazni interfejsi su interfejsi sa imenom *clk, reset, prvi, poslednji* i *rec*. Izlazni interfejsi dizajna su *preuzmi, mogu_da_primim_blok_poruke, finalna_obrada_u_toku* i *izlaz*.

Signal *clk* je korišćen kao signal takta. Uzlazna ivica ovog signala koristi se kao okidač za izvršavanje određenog koda, u zavisnosti od faze obrade. U jednom taktu prima se jedan blok poruke i u narednim taktovima vrši se njegova obrada. Jedna runda se izvršava u jednom taktu, dok je za obradu poruke od 256 bita (jedan blok poruke) potrebno nešto više od 124 taktova (po 72 takta za UBI MSG i UBI OUT blok).

Signal *reset* postavlja dizajn u idle stanje, tj. stanje u kojem se čeka prijem poruke. Izlazni signali *preuzmi* i *mogu_da_primim_blok_poruke*, koji će biti objašnjeni nešto kasnije, postavljaju se na neaktivnu vrednost. Signal *izlaz*, koji predstavlja heš vrednost, postavlja se na nultu vrednost.

Signal *prvi* signalizira da je na ulazu prisutan prvi blok poruke i predstavlja okidač za prelazak iz neaktivnog u aktivno stanje, tj. iz *idle* stanja u stanje u kome se vrši obrada.

Signalom *poslednji* signalizira se da je na ulazu prisutan poslednji blok poruke. Ovaj signal predstavlja i okidač za prelazak u UBI OUT blok za obradu poruke.

Ulazni signal *rec* predstavlja jedan blok poruke od 256 bita koju je potrebno obraditi.

Signal *mogu_da_primim_blok_poruke* označava da je obrada jednog bloka poruke završena i da je moguće primiti sledeći blok poruke na obradu.

Signalom *preuzmi* signalizira se da je heš vrednost izračunata i da trenutna vrednost signala izlaz predstavlja heš vrednost.

Kao što je rečeno, signal *izlaz* po završetku obrade predstavlja heš vrednost ulazne poruke. Dok je obrada u toku ili se čeka prijem poruke, signal ima nultu vrednost.

Funkcija crne kutije je izračunavanje heš vrednosti primenom Skein 256-256 algoritma na primljenu poruku. Zavisno od vrednosti signala sa ulaznih interfejsa, menjaju se i vrednosti internih signala. Dalje, pojedini interni signali predstavljaju okidače za promenu vrednosti drugih internih signala. Uloga internih signala je signaliziranje u kojoj fazi obrade se nalazi dizajn, tako da se u odnosu na te vrednosti primenjuje odgovarajuća faza obrade poruke (inicijalizacija stanja, UBI MSG obrada, UBI OUT obrada, postavljanje heš vrednosti kao vrednost signala izlaz). Sledi opis crne kutije.

4.3. Unutrašnjost crne kutije

U ovom potpoglavlju biće opisani tipovi promenljivih definisani za potrebe ovog rada i konačni automat korišćen u dizajnu. Tipovi promenljivih su korisnički definisani tipovi podataka i generisani su sa ciljem da olakšaju implementaciju algoritma.

4.3.1. Tipovi promenljivih

Pored predefinisanih tipova podataka *(std_logic, std_logic_vector* i *integer*), korišćeni su i korisnički definisani tipovi podataka, koji su definisani u zasebnom VHDL paketu. Sledi lista korisnički definisanih tipova i kratko objašnjenje za kakve promenljive se koriste:

- *stanje_niz*: Tip promenljive koji predstavlja niz od četiri elementa, gde je svaki element niz dužine 64 bita, tako da se *niz_stanje* sastoji od 256 bita. Sa ovim tipom je lakše

obrađivati funkcije i procedure Skein algoritma, pa se u ovaj tip promenljive smešta reč sa ulaza koju je potrebno obraditi.

- *stanje_niz_ET*: Tip promenljive koja se koristi za proširenu Tweak vrednost. Predstavlja niz od tri elementa, gde je svaki element niz dužine 64 bita.
- *stanje_niz_SY*: Tip promenljive koja se koristi za upisivanje podključa. Predstavlja niz od devet elementa, gde je svaki element niz dužine 64 bita.
- *konacni_automat*: Tip promenljive koja predstavlja stanje konačnog automata. Ovaj enumerisani tip sadrži logičke nazive za stanja automata. Detaljan opis i razlog upotrebe konačnog automata u ovom dizajnu sledi u narednom odeljku.

4.3.2. Konačni automat

Na osnovu trenutne vrednosti promenljive koja predstavlja stanje konačnog automata, primenjuje se odgovarajući kod, tj. odgovarajuća faza u obradi poruke. Stanja koja su definisana su:

- *idle,*
- *obrada_nulte_runde,*
- *UBI_msg_runda,*
- *gotov_jedan_blok_poruke,*
- *ima_vise_blokova_poruke,*
- *ulazi_u_UBI_OUT,*
- *UBI_out_runda,*
- *gotov_jedan_blok_out_poruke,*
- *zavrsio.*

Stanje *idle* je neaktivno stanje, tj. stanje u kome se nalazi dizajn pre prijema bilo kakve poruke. U ovom stanju se vrši inicijalizacija stanja.

Kada počne prijem poruke, dizajn se nalazi u stanju *idle* gde generiše podključ i prelazi u stanje *obrada_nulte_runde* gde se vrši obrada nulte runde, a zatim se nastavlja u stanje *UBI_msg_runda*. Dizajn ostaje u stanju *UBI_msg_runda* dok se ne izvrši ukupno 72 runde nad blokom poruke.

Potom, dizajn prelazi u stanje *gotov_jedan_blok_poruke* u kome se izvršava operacija XOR rezultata obrade sa ulaznim blokom poruke. Nakon toga, dizajn može da ode u *ima_vise_blokova_poruke* ili *ulazi_u_UBI_OUT* stanje.

U stanje *ima_vise_blokova_poruke* ulazi ukoliko signal *poslednji* nije detektovan, što znači da je ulazna poruka podeljena u više blokova poruke koje treba obraditi UBI MSG blokom pre ulaza u *ulazi_u_UBI_OUT* stanje. Ovo stanje generiše podključ za naredni blok poruke i iz ovog stanja prelazi u stanje *obrada_nulte_runde* koji vrši obradu narednog bloka poruke.

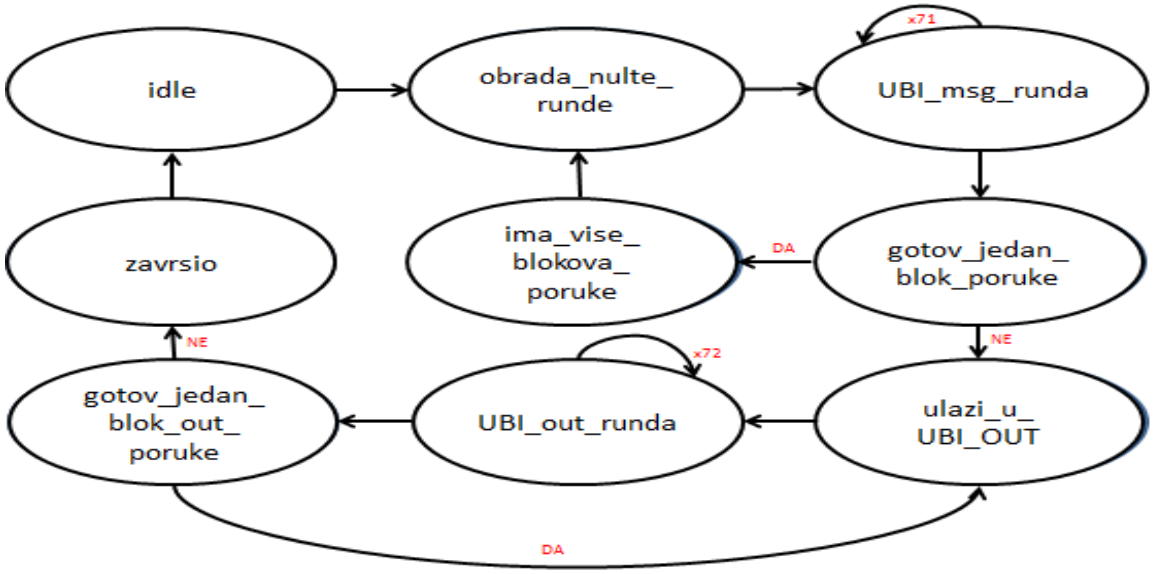
Dizajn ulazi u *ulazi_u_UBI_OUT* stanje ukoliko je bio primljen (i obrađen) poslednji blok poruke. U ovom stanju se generiše ključ za UBI OUT blok i postavljaju se vrednosti nekih signala. Izlaz iz ovog stanja je *UBI_out_runda*.

Kada se dizajn nadje u *UBI_out_runda* ostaje u ovom stanju dok se ne izvrši ukupno 72 runde nad blokom poruke. Zatim prelazi u stanje *gotov_jedan_blok_out_poruke*.

U stanju *gotov_jedan_blok_out_poruke* se odlučuje da li je jedna UBI OUT obrada bloka dovoljna za dobijanje heša određene dužine. Ukoliko jeste prelazi se u stanje *zavrsio*, a ukoliko nije vraća se u stanje u *ulazi_u_UBI_OUT* gde treba da se generise još jedna heš vrednost poruke kod UBI OUT bloka.

Kada se dizajn nađe u stanju *zavrsio*, heš se prosleđuje na izlaz i u narednom taktu stanje dizajna prelazi u *idle* stanje.

Slika 4.3.2.1. prikazuje stanja konačnog automata i prelazak iz jednog u drugo stanje.



Slika 4.3.2.1. Stanja konačnog automata i prelazak iz jednog u drugo stanje

4.4. Opis algoritma

U ovom potpoglavlju će prvo biti navedene i objašnjene konstante, funkcije i procedure korišćene u kodu. Nakon toga će biti objašnjen i kod koji prati tok obrade poruke, kao i uloga pre toga objašnjenih procedura. Funkcije, procedure, kao i pojedine konstante korišćene pri implementaciji ovog algoritma definisane su u paketu „*skein_256_paket*“.

4.4.1. Opis konstanti, funkcija i procedura

i) Konstante

Spisak svih konstanti koje se koriste u dizajnu:

- *C240,*
- *IV.*

Konstanta *C240* predstavljena tipom *STD_LOGIC_VECTOR* od 64 elemenata koristi se pri pravljenju proširenog ključa. Zahvaljujući ovoj konstanti, obezbeđuje se jedinstven podključ koji se dodaje poruci. Konstanta *C240* glasi:

```
CONSTANT C240:  STD_LOGIC_VECTOR (63 DOWNTO 0):= X"1bd11bdaa9fc1a22";
```

Konstanta *IV* predstavljena tipom *STD_LOGIC_VECTOR* od 256 elemenata se koristi na ulazu MSG UBI bloka. Ova konstanta zamenjuje ceo konfiguracioni UBI blok, tj. predstavlja rezultat iz UBI konfiguracionog bloka. Jedinstvena je za svaki Skein algoritam, i koristi se kao početna vrednost MSG UBI ključa. Za Skein 256-256 ova konstanta glasi:

```
CONSTANT iv: STD_LOGIC_VECTOR (255 DOWNTO 0):=
X"fc9da860d048b4492fca66479fa7d833b33bc3896656840f6a54e920fde8da69";
```

ii) Funkcije

Funkcija koja se koristi u dizajnu je *f_inicijalizacija*. Ona kao ulaz koristi konstantu *IV*, dok za rezultat vraća ključ. Ova funkcija se koristi kako bi dodelila vrednost *IV* ključu na početku UBI MSG bloka. Pošto svaki Skein algoritam ima jedinstven *IV* koji je tipa *STD_LOGIC_VECTOR* dužine 256 bita, ovim se obezbeđuje jednostavniji upis ključa koji je tipa *stanje_niz*. Nultom elementu kluča se dodeljuje 64 bita *IV* najveće težine, prvom elementu ključa drugih 64 bita *IV* najveće težine, itd... Poziva se uvek za prvi blok UBI MSG i glasi:

```
    FUNCTION f_inicijalizacija (CONSTANT iv: STD_LOGIC_VECTOR) RETURN stanje_niz
IS
        VARIABLE kljuc: stanje_niz;
    BEGIN
        kljuc(0):= iv(255 DOWNTO 192);
        kljuc(1):= iv(191 DOWNTO 128);
        kljuc(2):= iv(127 DOWNTO 64);
        kljuc(3):= iv(63 DOWNTO 0);
        RETURN kljuc;
    END f_inicijalizacija;
```

i) Procedure

Spisak svih Procedura koje se koriste u dizajnu:

- *procedura_Obradi_rec,*
- *procedura_Generisi_Subkey,*
- *procedura_Siftuj_Subkey,*
- *procedura_Runda_0,*
- *procedura_Runda_0_out,*
- *procedura_Runda_1_71,*
- *MIX,*
- *PERMUTE,*
- *ADD_SUBKEY,*
- *procedura_Krajnji_XOR,*
- *procedura_Rezultat.*

Procedura *procedura_Obradi_rec* vrši rotiranje reči sa ulaza UBI MSG bloka u odgovarajućem redosledu i upisuje je u varijablu *lokalna_rec* koja je istog tipa. Uloga ove procedure je da se olakša procedura *procedura_Krajnji_XOR*, koja će biti objašnjena kasnije. Princip rada ove procedure je sledeci: ulaznu reč koja je 256 bita posmatrati u blokovima od 64 bita. Zatim za 64-bitnu reč obrnuti redosled bajtova i upisati dobijeni rezultat u promenljivu *lokalnu_rec*. Kod ove procedure je sledeći:

```
    PROCEDURE procedura_Obradi_rec(VARIABLE lokalna_rec: INOUT
STD_LOGIC_VECTOR(255 DOWNTO 0); SIGNAL rec: IN STD_LOGIC_VECTOR(255 DOWNTO 0))
IS
    BEGIN
        FOR i IN 0 to 3 LOOP
            FOR j IN 0 to 7 LOOP
                FOR d IN 0 to 7 LOOP
                    lokalna_rec(i*64+j*8+d):= rec((i+1)*64-(j+1)*8+d);
                END LOOP;
            END LOOP;
        END LOOP;
    END procedura_Obradi_rec;
```

Procedura *procedura_Generisi_Subkey* koristi se da generise podključ pre ulaza u bilo koji u UBI blokova (MSG, OUT). Ova procedura kao rezultat vraća vrednost produženog ključa, produženu Tweak vrednost i vrednost brojača *s*, koji se upisuju u registar *SUBKEY_REGISTAR*. Ovaj registar će se koristiti u narednih 72 rundi za generisanje podključa. Ova procedura mora biti pozvana minumum jedan takt pre prve runde UBI blokova kako bi postavila ispravnu vrednost u registar *SUBKEY_REGISTAR*. Kao ulazne parametre poziva promenljivu *kljuc* pomoću koga se generiše prošireni ključ, *radi_ubi_out* koji služi za tačno generisanje *Type* polja Tweak promenljive u zavisnosti da li se radi UBI MSG ili UBI OUT blok, signale *prvi*, *poslednji* i *duzina_msg_bloka* koji služe da generišu jednoznačnu Tweak vrednost.

```
    PROCEDURE procedura_Generisi_Subkey(SIGNAL SUBKEY_REGISTAR: OUT
stanje_niz_SY; VARIABLE kljuc: IN stanje_niz; VARIABLE radi_ubi_out: IN
STD_LOGIC; SIGNAL prvi : IN  std_logic; SIGNAL poslednji : IN  std_logic;
VARIABLE duzina_msg_bloka: IN INTEGER) IS
        VARIABLE s: STD_LOGIC_VECTOR (63 DOWNTO 0):=(OTHERS=>'0');
        VARIABLE e_tweak: stanje_niz_ET;
        VARIABLE pomocna0: STD_LOGIC_VECTOR (63 DOWNTO 0);
        VARIABLE pomocna1: STD_LOGIC_VECTOR (63 DOWNTO 0);
    BEGIN
        IF (radi_ubi_out='0') THEN
            pomocna0:=(OTHERS=>'0');
            pomocna1:=(OTHERS=>'0');
            IF(prvi='1')THEN
                pomocna0:= pomocna0 + X"4000000000000000";
            END IF;
            IF(poslednji='1')THEN
                pomocna0:= pomocna0 + X"8000000000000000";
            END IF;
            pomocna0:= pomocna0 + X"3000000000000000";
            FOR i IN 0 TO 100
                iF (i < duzina_msg_bloka) THEN
                    pomocna1:= pomocna1 + X"0000000000000020";
                END iF;
            END LOOP;
            e_tweak(1):= pomocna0;
            e_tweak(0):= pomocna1;
            e_tweak(2):= pomocna0 XOR pomocna1;
```

Ovaj deo koda, ukoliko je *radi_ubi_out* podešen na 0, generiše Tweak za UBI MSG blok. Ukoliko je signal *prvi* ili *poslednji* podešen na 1, postavljaju se vrednosti Tweak polja *first* i *last* na 1. Zatim se proverava koji UBI MSG blok po redu se obrađuje i dodaje se na vrednost Tweak polja *Position*.

```
        ELSE
            pomocna0:= X"ff00000000000000";
            pomocna1:= X"0000000000000008";
            e_tweak(1):= pomocna0;
            e_tweak(0):= pomocna1;
            e_tweak(2):= e_tweak(0) XOR e_tweak(1);
        END IF;
```

Ukoliko je *radi_ubi_out* podešen na 1, generiše se Tweak za UBI OUT blok. Nezavisno od vrednosti *radi_ubi_out* računa se proširenu ključ i sve dobijene vrednosti (proširenu ključ i proširena Tweak vrednost) se upisuju u *SUBKEY_REGISTAR*.

```
        pomocna0:= kljuc(0) XOR kljuc(1) XOR kljuc(2) XOR kljuc(3) XOR C240;
        SUBKEY_REGISTAR(0)<= kljuc(0);
        SUBKEY_REGISTAR(1)<= kljuc(1);
        SUBKEY_REGISTAR(2)<= kljuc(2);
        SUBKEY_REGISTAR(3)<= kljuc(3);
        SUBKEY_REGISTAR(4)<= pomocna0;
        SUBKEY_REGISTAR(5)<= e_tweak(0);
        SUBKEY_REGISTAR(6)<= e_tweak(1);
        SUBKEY_REGISTAR(7)<= e_tweak(2);
        SUBKEY_REGISTAR(8)<= s;
    END procedura_Generisi_Subkey;
```

Procedura *procedura_Siftuj_Subkey* koristi se nakon dodavanja prvog podključa i služi da obezbedi jedinstvenost narednih 18 podključeva koji se dodaju registru *REGISTAR*. Nakon generacije podključa, sa svakom novom potrebom za generisanjem podključa, dovoljno je izvršiti pomeranje ulevo za 64 bita registra produženog kluča i produžene Tweak vrednosti, dok se vrednost *s* inkrementira. Vrednosti nakon pomeraja se upisuju registar *SUBKEY_REGISTAR* i koristiće se kod procedure *ADD_SUBKEY*. Pošto se upis podključa vrši za svaku četvrtu rundu, bitno je obezbediti novi podključ minimum jedan takt pre njegovog dodavanja registru *REGISTAR* kako bi se očitala ispravna vrednost iz registara *SUBKEY_REGISTAR*.

```
    PROCEDURE procedura_Siftuj_Subkey (SIGNAL SUBKEY_REGISTAR: INOUT
stanje_niz_SY) IS
        VARIABLE pomocnaSK: stanje_niz_SY;
    BEGIN
        pomocnaSK := SUBKEY_REGISTAR;
        SUBKEY_REGISTAR(0)<= pomocnaSK(1); --rotiram key
        SUBKEY_REGISTAR(1)<= pomocnaSK(2);
        SUBKEY_REGISTAR(2)<= pomocnaSK(3);
        SUBKEY_REGISTAR(3)<= pomocnaSK(4);
        SUBKEY_REGISTAR(4)<= pomocnaSK(0);
        SUBKEY_REGISTAR(5)<= pomocnaSK(6); --rotiram tweak
        SUBKEY_REGISTAR(6)<= pomocnaSK(7);
        SUBKEY_REGISTAR(7)<= pomocnaSK(5);
        SUBKEY_REGISTAR(8)<= pomocnaSK(8)+X"0000000000000001"; --inkrementiram s
    END procedura_Siftuj_Subkey;
```

Procedura *procedura_Runda_0* vrši rotiranje reči sa ulaza UBI MSG bloka u odgovarajućem redosledu (koji je već opisan u proceduri *procedura_Obradi_rec*) i zatim prosleđuje tu vrednost na promenljivu *pomocni_registar*, tako što nultom elementu pomoćnog registra dodeljuje 64 bita najveće težine rotirane reči. Promenljiva *pomocni_registar* koristi se radi lakše obrade reči sa ulaza. Nakon toga poziva procedure *ADD_SUBKEY* (koja vrši prvo dodavanje podključa), i *MIX* i

PERMUTE procedure koje će kasnije biti objašnjene. Vrednost koja se dobila na kraju izvršavanja ovih procedura se smešta u *REGISTAR*.

```
    ...
        pomocni_registar(0):= pomocna_rec(255 DOWNTO 192);
        pomocni_registar(1):= pomocna_rec(191 DOWNTO 128);
        pomocni_registar(2):= pomocna_rec(127 DOWNTO 64);
        pomocni_registar(3):= pomocna_rec(63 DOWNTO 0);
        ADD_SUBKEY(SUBKEY_REGISTAR, pomocni_registar);
        MIX(pomocni_registar, brojac72_varijabla);
        PERMUTE(pomocni_registar);
        REGISTAR <= pomocni_registar;
    END procedura_Runda_0;
```

Procedura *procedura_Runda_0_out* je ista kao procedura *procedura_Runda_0*, samo se za ulaznu reč uzima reč koja je postavljena na 0 (definisano kod UBI OUT bloka).

Procedura *procedura_Runda_1_71* se sastoji od procedura *ADD_SUBKEY*, *MIX* i *PERMUTE*. Slična je proceduri *procedura_Runda_0*, ali se za obradu ne koristi reč sa ulaza bloka, već se koristi vrednost koja je zapamćena u registru *REGISTAR*. Razlika je i u tome što se procedura *ADD_SUBKEY* ne vrši na početku runde, već na kraju, i to ukoliko je ispunjen uslov da je u pitanju svaka četvrta runda u fazi obrade. Ukoliko uslov nije ispunjen, ova procedura se preskače. Umesto signala *REGISTAR*, u podprocedurama koristiće se promenljiva *pomocni_registar* koja ne unosi kašnjenje u kod. Na kraju procedure *procedura_Runda_1_71* registru *REGISTAR* dodeliće se vrednost promenljive *pomocni_registar* koja je obrađena svim podprocedurama. Kod ove procedure je sledeći:

```
    PROCEDURE procedura_Runda_1_71(SIGNAL SUBKEY_REGISTAR: IN stanje_niz_SY;
SIGNAL REGISTAR: INOUT stanje_niz; VARIABLE brojac72_varijabla: IN INTEGER) IS
        VARIABLE pomocni_registar:stanje_niz;
    BEGIN
        pomocni_registar:=REGISTAR;
        MIX(pomocni_registar, brojac72_varijabla);
        PERMUTE(pomocni_registar);
        IF (((brojac72_varijabla+1) mod 4) = 0) THEN
            ADD_SUBKEY(SUBKEY_REGISTAR, pomocni_registar);
        END IF;
        REGISTAR<=pomocni_registar;
    END procedura_Runda_1_71;
```

Procedura *MIX* služi da obavi MIX operaciju Skein algoritma koja je opisana u poglavlju 3.1.1. Ulazni parametri su promenljiva *pomocni_registar* i brojač *brojac72_varijabla*.

```
    PROCEDURE MIX(VARIABLE pomocni_registar: INOUT stanje_niz; VARIABLE
brojac72_varijabla: IN INTEGER) IS
        VARIABLE pomocna1:stanje_niz;
        VARIABLE pomocna2:stanje_niz;
        VARIABLE brojac:INTEGER;
    BEGIN
        pomocna1:=pomocni_registar;
        pomocna2:=pomocni_registar;
        brojac:= brojac72_varijabla mod 8;
```

Nulti element promenljive *pomocni_registar* nakon MIX operacije treba da ima vrednost sabiranja nultog i prvog elementa promenljive *pomocni_registar* sa ulaza. Ovo se postiže sledećim delom koda (analogno se dobija i treći element):

```
        pomocna2(0):= pomocna1(0)+pomocna1(1);
        pomocna2(2):= pomocna1(2)+pomocna1(3);
```

Prvi i treći element promenljive *pomocni_registar* treba rotirati ulevo za *R* konstantu, koja se ponavlja na svakih 8 rundi, pa se za njeno kontrolisanje koristi brojač *brojac72_varijabla* po modulu 8. Pomeranje ulevo je odrađeno sledećim delom koda:

```
        IF (brojac=0) THEN          --RADE SE ROTACIJE NA LEVO
            pomocna2(1):= pomocna1(1)(49 DOWNTO 0)& pomocna1(1)(63 DOWNTO 50); -
-treba da se rotira za 14  element 1.
            pomocna2(3):= pomocna1(3)(47 DOWNTO 0)& pomocna1(3)(63 DOWNTO 48);--
treba da se rotira za 16 element 3.
        ELSIF (brojac=1) THEN
            pomocna2(1):= pomocna1(1)(11 DOWNTO 0)& pomocna1(1)(63 DOWNTO 12);--
treba da se rotira za 52
            pomocna2(3):= pomocna1(3)(06 DOWNTO 0)& pomocna1(3)(63 DOWNTO 07);--
treba da se rotira za 57
        ELSIF (brojac=2) THEN
            pomocna2(1):= pomocna1(1)(40 DOWNTO 0)& pomocna1(1)(63 DOWNTO 41);--
treba da se rotira za 23
            pomocna2(3):= pomocna1(3)(23 DOWNTO 0)& pomocna1(3)(63 DOWNTO 24);--
treba da se rotira za 40
        ELSIF (brojac=3) THEN
            pomocna2(1):= pomocna1(1)(58 DOWNTO 0)& pomocna1(1)(63 DOWNTO 59);--
treba da se rotira za 5
            pomocna2(3):= pomocna1(3)(26 DOWNTO 0)& pomocna1(3)(63 DOWNTO 27);--
treba da se rotira za 37
        ELSIF (brojac=4) THEN
            pomocna2(1):= pomocna1(1)(38 DOWNTO 0)& pomocna1(1)(63 DOWNTO 39);--
treba da se rotira za 25
            pomocna2(3):= pomocna1(3)(30 DOWNTO 0)& pomocna1(3)(63 DOWNTO 31);--
treba da se rotira za 33
        ELSIF (brojac=5) THEN
            pomocna2(1):= pomocna1(1)(17 DOWNTO 0)& pomocna1(1)(63 DOWNTO 18);--
treba da se rotira za 46
            pomocna2(3):= pomocna1(3)(51 DOWNTO 0)& pomocna1(3)(63 DOWNTO 52);--
treba da se rotira za 12
        ELSIF (brojac=6) THEN
            pomocna2(1):= pomocna1(1)(05 DOWNTO 0)& pomocna1(1)(63 DOWNTO 06);--
treba da se rotira za 58
            pomocna2(3):= pomocna1(3)(41 DOWNTO 0)& pomocna1(3)(63 DOWNTO 42);--
treba da se rotira za 22
        ELSE
            pomocna2(1):= pomocna1(1)(31 DOWNTO 0)& pomocna1(1)(63 DOWNTO 32);--
treba da se rotira za 32
            pomocna2(3):= pomocna1(3)(31 DOWNTO 0)& pomocna1(3)(63 DOWNTO 32);--
treba da se rotira za 32
        END IF;
```

Zatim nad tako dobijenim prvim i trećim elementom promenljive *pomocni_registar* treba izvršiti operaciju XOR sa već dobijenim nultim i drugim elementom:

```
        pomocna2(1):= pomocna2(0) XOR pomocna2(1);-- rotiran 1.el i operacija
XOR nad 0.el i.
        pomocna2(3):= pomocna2(2) XOR pomocna2(3); -- rotiran 3.el i operacija
XOR nad 2.el
        pomocni_registar:=pomocna2;
    END MIX;
```

Procedura *PERMUTE* koristi se kako bi se permutovale vrednosti u registru. Realizovana je kao jednostavna zamena mesta 64-bitnih reči za promenljivu *pomocni_registar* koja je tipa *stanje_niz*. Redosled zamene je: 64-bitna reč na poziciji 0, 1, 2, 3 se rotira po redosledu 0, 3, 2, 1. Kod procedure je sledeći:

```
PROCEDURE PERMUTE(VARIABLE pomocni_registar: INOUT stanje_niz) IS
    VARIABLE pomocna: stanje_niz;
BEGIN
    pomocna:=pomocni_registar;
    pomocni_registar(0):=pomocna(0);
    pomocni_registar(1):=pomocna(3);
    pomocni_registar(2):=pomocna(2);
    pomocni_registar(3):=pomocna(1);
END PERMUTE;
```

Procedura *ADD_SUBKEY* služi za dodavanje podključa poruci koja se obrađuje. Dodavanje podključa je organizovano na sledeći način za prvo dodavanje u bloku obrade: nultom elementu poruke dodaje se nulti elemenat podključa (tj. prvi element ključa), prvom elementu poruke prvi i peti element podključa (tj. drugi element ključa i prvi element Tweak promenljive), drugom elementu poruke se dodaje drugi i šesti element podključa (tj. treći element ključa i drugi element Tweak promenljive) i trećem elementu poruke se dodaje treći i osmi element podključa (tj. treći element ključa i promenljiva *s*). Pre sledećeg dodavanja vrši se procedura *procedura_Siftuj_Subkey*, koja obezbeđuje da su elementi podključa ažurirani, tako da je moguće korišćenje procedure *ADD_SUBKEY* u sledećoj rundi bez izmena u kodu za ovu proceduru. Kod opisane procedure je sledeći:

```
    PROCEDURE ADD_SUBKEY(SIGNAL SUBKEY_REGISTAR: IN stanje_niz_SY; VARIABLE
pomocni_registar: INOUT stanje_niz) IS
    BEGIN
        pomocni_registar(0):=SUBKEY_REGISTAR(0) + pomocni_registar(0);
        pomocni_registar(1):=SUBKEY_REGISTAR(1) + SUBKEY_REGISTAR(5) +
pomocni_registar(1);
        pomocni_registar(2):=SUBKEY_REGISTAR(2) + SUBKEY_REGISTAR(6) +
pomocni_registar(2);
        pomocni_registar(3):=SUBKEY_REGISTAR(3) + SUBKEY_REGISTAR(8) +
pomocni_registar(3);
    END ADD_SUBKEY;
```

Procedura *procedura_Krajnji_XOR* poziva se na kraju obrade svakog UBI bloka. Služi da obavi operaciju XOR poruke sa ulaza UBI bloka i poruke sa izlaza UBI bloka i smesti je u promenljivu *kljuc*. Reč na ulazu u blok smeštena je u promenljivu *lokalna_rec*, a reč sa izlaza UBI bloka smeštena je u registru *REGISTAR*. U promenljiva *kljuc* je namerno upisana vrednost XOR operacije, jer će se na ulazu u sledeći blok baš ova vrednost koja je dodeljena ovoj promenljivoj koristiti kao ključ za generisanje podključa (ukoliko je u pitanju zadnji blok, vrednost *kljuc* koristiće se za dobijanje heša). Kod procedure je sledeći:

```
    PROCEDURE procedura_Krajnji_XOR(SIGNAL REGISTAR: IN stanje_niz; VARIABLE
lokalna_rec: IN STD_LOGIC_VECTOR(255 DOWNTO 0); VARIABLE kljuc: INOUT
stanje_niz) IS
    BEGIN
        kljuc(0):= lokalna_rec(255 DOWNTO 192) XOR REGISTAR(0);
```

```
        kljuc(1):= lokalna_rec(191 DOWNTO 128) XOR REGISTAR(1);
        kljuc(2):= lokalna_rec(127 DOWNTO 64) XOR REGISTAR(2);
        kljuc(3):= lokalna_rec(63 DOWNTO 0) XOR REGISTAR(3);
    END procedura_Krajnji_XOR;
```

Procedura *procedura_Rezultat* služi da prosledi rezultat heša (koji je tipa *stanje_niz* i sadržan je u promenljivoj *kljuc*) promenljivoj *hash_final* koja je niz od 256 bita, u odgovarajućem redosledu, tj. ova procedura vrši rotiranje reči sa izlaza UBI OUT bloka i analogna je proceduri *procedura_Obradi_rec*. Kod ove procedure je sledeći:

```
    PROCEDURE procedura_Rezultat( VARIABLE kljuc: IN stanje_niz; VARIABLE
hash_final: OUT STD_LOGIC_VECTOR(255 DOWNTO 0)) IS
        VARIABLE pomocna_rec:STD_LOGIC_VECTOR(255 DOWNTO 0);
    BEGIN
        pomocna_rec(255 DOWNTO 192):=kljuc(0);
        pomocna_rec(191 DOWNTO 128):=kljuc(1);
        pomocna_rec(127 DOWNTO 64):=kljuc(2);
        pomocna_rec(63 DOWNTO 0):=kljuc(3);
        FOR i IN 0 to 3 LOOP
            FOR j IN 0 to 7 LOOP
                FOR d IN 0 to 7 LOOP
                    hash_final(i*64+j*8+d):= pomocna_rec((i+1)*64-(j+1)*8+d);
                END LOOP;
            END LOOP;
        END LOOP;
    END procedura_Rezultat;
```

4.4.2. Opis rada top-level entiteta

Nakon što su objašnjene funkcije i procedure potrebne za implementaciju algoritma, biće opisan kod u arhitekturi top-level entiteta kojim se realizuje celokupna obrada poruke. Pre nego što se počne sa opisom koda, radi uvida u celokupan kod biće prikazana arhitektura top-level entiteta:

```
ARCHITECTURE shema OF skein_256 IS
    SIGNAL stanje_automata: konacni_automat;
    SIGNAL REGISTAR: stanje_niz;
    SIGNAL SUBKEY_REGISTAR: stanje_niz_SY;
    SIGNAL signalizacija_da_se_ide_u_UBIout: STD_LOGIC;
BEGIN

    PROCESS(reset,clk)
        VARIABLE brojac72_varijabla: INTEGER;
        VARIABLE kljuc: stanje_niz;
        VARIABLE lokalna_rec: STD_LOGIC_VECTOR(255 DOWNTO 0);
        VARIABLE duzina_msg_bloka: INTEGER :=1;
        VARIABLE radi_ubi_out: STD_LOGIC :='0';
        VARIABLE hash_final: STD_LOGIC_VECTOR (255 DOWNTO 0);
    BEGIN
        IF (reset='1')THEN
        stanje_automata<= idle;
        signalizacija_da_se_ide_u_UBIout<='0';
        preuzmi<='0';
        finalna_obrada_u_toku<='0';
        mogu_da_primim_blok_poruke<='0';
        izlaz<=(OTHERS=>'0');
        duzina_msg_bloka:=1;
```

```
ELSIF (clk'EVENT AND clk='1') THEN
        CASE (stanje_automata) IS
            WHEN idle =>
                preuzmi<='0';
                finalna_obrada_u_toku<='0';
                radi_ubi_out:='0';
                mogu_da_primim_blok_poruke<='1';
                duzina_msg_bloka:=1;
                IF (prvi='1') THEN
                    brojac72_varijabla:=0;
                    procedura_Obradi_rec(lokalna_rec, rec);
                    kljuc:= f_inicijalizacija(iv);
                    procedura_Generisi_Subkey(SUBKEY_REGISTAR, kljuc,
                        radi_ubi_out, prvi, poslednji,
                        duzina_msg_bloka);
                    stanje_automata<= obrada_nulte_runde;
                    mogu_da_primim_blok_poruke<='0';
                    IF(poslednji='1')THEN
                        signalizacija_da_se_ide_u_UBIout<='1';
                    ELSE
                        signalizacija_da_se_ide_u_UBIout<='0';
                    END IF;
                END IF;

            WHEN obrada_nulte_runde =>
                procedura_Runda_0(rec, SUBKEY_REGISTAR,REGISTAR,
                    brojac72_varijabla);
                stanje_automata<= UBI_msg_runda;

            WHEN UBI_msg_runda =>
                brojac72_varijabla:=brojac72_varijabla+1;
                mogu_da_primim_blok_poruke<='0';
                procedura_Runda_1_71(SUBKEY_REGISTAR, REGISTAR,
                    brojac72_varijabla);
                IF ((brojac72_varijabla+2) MOD 4 = 0 ) THEN
                    procedura_Siftuj_Subkey(SUBKEY_REGISTAR);
                END IF;
                IF (brojac72_varijabla=71) THEN
                    stanje_automata<= gotov_jedan_blok_poruke;
                ELSE
                    stanje_automata<= UBI_msg_runda;
                END IF;
            WHEN gotov_jedan_blok_poruke =>
                procedura_Krajnji_XOR(REGISTAR, lokalna_rec, kljuc);
                brojac72_varijabla:=0;
                IF(signalizacija_da_se_ide_u_UBIout='1')THEN
                    stanje_automata<= ulazi_u_UBI_OUT;
                ELSE
                    stanje_automata<= ima_vise_blokova_poruke;
                    mogu_da_primim_blok_poruke<='1';
                END IF;
            WHEN ima_vise_blokova_poruke =>
                duzina_msg_bloka:=duzina_msg_bloka+1;
                procedura_Obradi_rec(lokalna_rec, rec);
                procedura_Generisi_Subkey(SUBKEY_REGISTAR, kljuc,
                     radi_ubi_out, prvi, poslednji, duzina_msg_bloka);
                mogu_da_primim_blok_poruke<='0';
                stanje_automata<= obrada_nulte_runde;
                IF(poslednji='1')THEN
```

```
                    signalizacija_da_se_ide_u_UBIout<='1';
                ELSE
                    signalizacija_da_se_ide_u_UBIout<='0';
                END IF;
            WHEN ulazi_u_UBI_OUT =>
                finalna_obrada_u_toku<='1';
                radi_ubi_out:='1';
                procedura_Generisi_Subkey(SUBKEY_REGISTAR, kljuc,
                       radi_ubi_out, prvi, poslednji, duzina_msg_bloka);
                stanje_automata<= UBI_out_runda;
            WHEN UBI_out_runda =>
                IF brojac72_varijabla = 0 THEN
                    procedura_Runda_0_out(SUBKEY_REGISTAR, REGISTAR,
                        brojac72_varijabla);
                    brojac72_varijabla:=brojac72_varijabla+1;
                    stanje_automata<= UBI_out_runda;
                ELSE
                    procedura_Runda_1_71(SUBKEY_REGISTAR, REGISTAR,
                        brojac72_varijabla);
                    brojac72_varijabla:=brojac72_varijabla+1;
                    IF ((brojac72_varijabla+1) MOD 4 = 0 ) THEN
                        procedura_Siftuj_Subkey(SUBKEY_REGISTAR);
                    END IF;
                    IF (brojac72_varijabla = 72) THEN
                        stanje_automata<= gotov_jedan_blok_out_poruke;
                    ELSE
                        stanje_automata<=UBI_out_runda;
                    END IF;
                END IF;
            WHEN gotov_jedan_blok_out_poruke=>
                lokalna_rec := (OTHERS=>'0');
                procedura_Krajnji_XOR(REGISTAR, lokalna_rec, kljuc);
                brojac72_varijabla:=0;
                stanje_automata<= zavrsio;
                procedura_Rezultat(kljuc, hash_final); -
            WHEN zavrsio =>
                izlaz<=hash_final;
                preuzmi<='1';
                finalna_obrada_u_toku<='0';
                radi_ubi_out:='0';
                stanje_automata<= idle;
                mogu_da_primim_blok_poruke<='0';
        END CASE;
    END IF;
END PROCESS;

END shema;
```

Kako je u pitanju sekvencijalni kod, on se nalazi unutar procesa. U listi osetljivosti nalazi se *reset* i signal takta *clk*. Dizajn ima mogućnost asinhronog reseta, što znači da se kod izvršava svaki put kada dođe do promene vrednosti signala takta ukoliko signal za reset nije aktivan, u suprotnom se određeni signali postavljaju na inicijalne vrednosti.

Signali definisani unutar arhitekture su: *stanje_automata, REGISTAR, SUBKEY_REGISTAR, signalizacija_da_se_ide_u_UBIout.*

Signal *stanje_automata* je tipa *konacni_automat* i njegova trenutna vrednost predstavlja fazu u kojoj se dizajn nalazi.

Signal *REGISTAR* je tipa *stanje_niz* i u njemu se čuva rezultat obrade nakon svake runde.

Signal *SUBKEY_REGISTAR* je tipa *stanje_niz _SY* i u njemu se smešta vrednost podključa.

U deklarativnom delu procesa definisana su varijable: *brojac72_varijabla, kljuc, lokalna_rec, duzina_msg_bloka, radi_ubi_out, hash_final.*

Varijabla *brojac72_varijabla* koja je tipa *integer* koristi se kao brojač rundi kod UBI blokova.

Varijabla *kljuc* tipa *stanje_niz* služi za čuvanje *IV* na početku obrade kao i rezultata krajnje XOR operacije na izlazu UBI bloka.

Varijabla *lokalna_rec* tipa *stanje_niz* služi da se čuvanje ulazne reči koja se obrađuje i koristi se kod krajnje XOR operacije.

Varijabla *duzina_msg_bloka* služi da se naznači o kom se MSG bloku obrade radi, ukoliko se poruka sastoji iz više blokova i koristi se pri generisanju Tweak vrednosti.

Varijabla *radi_ubi_out* služi da naglasi da se radi o UBI OUT bloku za obradu poruke, i to pri generisanju Tweak vrednosti.

Varijabla *hash_final* koja je tipa *STD_LOGIC_VECTOR (255 DOWNTO 0)* koristi se pri krajnjoj obradi heš vrednosti i pritom je istog tipa kao i signal koji se šalje na izlaz, pa se lako može dodeliti izlaznom signalu.

Naredbe koje se izvršavaju u slučaju aktivnog signala za reset su sledeće:

```
stanje_automata<= idle;
signalizacija_da_se_ide_u_UBIout<='0';
preuzmi<='0';
finalna_obrada_u_toku<='0';
mogu_da_primim_blok_poruke<='0';
izlaz<=(OTHERS=>'0');
duzina_msg_bloka:=1;
```

Ukoliko je u trenutku reseta obrada bila u toku, ona će se prekinuti. Signali *signalizacija_da_se_ide_u_UBIout, preuzmi, finalna_obrada_u_toku, mogu_da_primim_blok_poruke* se postavljaju na 0, signal *duzina_msg_bloka* se postavlja na 1, dok se vrednost *izlaz* koji predstavlja heš postavlja na 0. Ukoliko je signal za reset neaktivan, prelazi se u *idle* stanje. Kasnije u zavisnosti od vrednosti signala *stanje_automata*, izvršava se odgovarajući kod. U tu svrhu je iskorišćena CASE struktura. Kod koji se izvršava je onaj koji se nalazi u okviru WHEN dela koji sadrži vrednost koja se poklapa sa trenutnom vrednosti signala *stanje_automata*.

Ukoliko se dizajn nalazi u *idle* stanju, tj. stanju u kome se čeka prijem poruke, izvršava se sledeći kod:

```
preuzmi<='0';
finalna_obrada_u_toku<='0';
radi_ubi_out:='0';
mogu_da_primim_blok_poruke<='1';
duzina_msg_bloka:=1;
IF (prvi='1') THEN
    brojac72_varijabla:=0;
    procedura_Obradi_rec(lokalna_rec, rec);
```

```
            kljuc:= f_inicijalizacija(iv);
            procedura_Generisi_Subkey(SUBKEY_REGISTAR, kljuc,
                radi_ubi_out, prvi, poslednji,
                duzina_msg_bloka);
            stanje_automata<= obrada_nulte_runde;
            mogu_da_primim_blok_poruke<='0';
            IF(poslednji='1')THEN
                signalizacija_da_se_ide_u_UBIout<='1';
            ELSE
                signalizacija_da_se_ide_u_UBIout<='0';
            END IF;
        END IF;
```

Signali *preuzmi*, *finalna_obrada_u_toku*, i *radi_ubi_out* se podešavaju na vrednost 0, signali *mogu_da_primim_blok_poruke*, *duzina_msg_bloka* se stavljaju na vrednost 1. Čeka se signal *prvi* koji nagoveštava da treba da se počne sa obradom poruke koja je stigla na ulaz.

Kada se primi prvi blok poruke, brojač rundi se postavlja na 0. Vrši se obrada reči sa ulaza u proceduri *procedura_Obradi_rec* i smešta se u promenljivu *lokalnu_reč* koja će da se koristi za krajnju operaciju XOR, zatim se vrši postavljanje ključa na *IV*. Onda se za dobijeni ključ generiše podključ u proceduri *procedura_Generisi_Subkey* i taj podključ se upisuje u registar *SUBKEY_REGISTAR* . Signal *mogu_da_primim_blok_poruke* se postavlja na 0. U zavisnosti da li je primljena reč ujedno i poslednja reč (signal *poslednji*), podešava se i signal *signalizacija_da_se_ide_u_UBIout.* Stanje automata se postavlja u stanje *obrada_nulte_runde.*

Kada se dizajn nađe u stanju *obrada_nulte_runde*, izvršava se sledeći kod:

```
procedura_Runda_0(rec, SUBKEY_REGISTAR,REGISTAR,
    brojac72_varijabla);
stanje_automata<= UBI_msg_runda;
```

Radi se nulta runda UBI MSG bloka koja kao rezultat runde vraća upis u registar *REGISTAR*, a koristi podatke sa ulaza bloka koji treba da se obradi tj. reč sa ulaza, stanje brojača i vrednost iz registra *SUBKEY_REGISTAR*. Ostale runde (runda 1-71) koriste stanje iz registra *REGISTAR* za obradu umesto ulaznog bloka poruke. Stanje automata se postavlja u stanje *UBI_msg_runda.*

Kada se dizajn nađe u stanju *UBI_msg_runda*, izvršava se sledeći kod:

```
brojac72_varijabla:=brojac72_varijabla+1;
procedura_Runda_1_71(SUBKEY_REGISTAR, REGISTAR,
       brojac72_varijabla);
IF ((brojac72_varijabla+2) MOD 4 = 0 ) THEN
    procedura_Siftuj_Subkey(SUBKEY_REGISTAR);
END IF;
IF (brojac72_varijabla=71) THEN
    stanje_automata<= gotov_jedan_blok_poruke;
ELSE
    stanje_automata<= UBI_msg_runda;
END IF;
```

Brojač se inkrementira za 1 kako bi ispravno pratio runde. Radi se nova runda obrade poruke u proceduri *procedura_Runda_1_71*, i rezultat runde se smešta u *REGISTAR*. Zatim se vrši provera da li je *(vrednost trenutne runde + 1) po mod 4 =0*. Ukoliko je ovo ispunjeno radi se generisanje podključa (tj. njegovo ažuriranje) za sledeću rundu preko procedure *procedura_Siftuj_Subkey*. Ovo

je potrebno uraditi pre korišćenja podključa u svakoj četvrtoj rundi. Zatim se proverava da li je brojač stigao do 71, što bi značilo da je završena obrada jednog bloka poruke i da *stanje_automata* treba da se promeni u *gotov_jedan_blok_poruke*. Ukoliko brojač još nije dostigao vrednost 71, ostaje se u stanju *UBI_msg_runda.*

Kada se dizajn nađe u stanju *gotov_jedan_blok_poruke*, izvršava se sledeći kod:

```
procedura_Krajnji_XOR(REGISTAR, lokalna_rec, kljuc);
brojac72_varijabla:=0;
IF(signalizacija_da_se_ide_u_UBIout='1')THEN
    stanje_automata<= ulazi_u_UBI_OUT;
ELSE
    stanje_automata<= ima_vise_blokova_poruke;
    mogu_da_primim_blok_poruke<='1';
END IF;
```

U ovom stanju se radi operacija XOR poruke na ulazu u UBI MSG blok, i poruke na izlazu u proceduri *procedura_Krajnji_XOR.* Kao rezultat ove procedure dobija se ključ koji će biti korišćen u daljem kodu. Zatim se brojač rundi postavlja na vrednost 0 i vrši se provera da li je obrađeni blok poruke ujedno bio i poslednji blok UBI MSG bloka. Ukoliko je to ispunjeno, stanje automata prelazi na *ulazi_u_UBI_OUT*, a ukoliko to nije slučaj stanje automata se podešava na vrednost *ima_vise_blokova_poruke*, a signal *mogu_da_primim_blok_poruke* se postavlja na 1 i naznačava da je moguće primiti sledeći blok poruke na obradu.

Kada se dizajn nađe u stanju *ima_vise_blokova_poruke*, izvršava se sledeći kod:

```
duzina_msg_bloka:=duzina_msg_bloka+1;
procedura_Obradi_rec(lokalna_rec, rec);
procedura_Generisi_Subkey(SUBKEY_REGISTAR, kljuc,
     radi_ubi_out, prvi, poslednji, duzina_msg_bloka);
mogu_da_primim_blok_poruke<='0';
stanje_automata<= obrada_nulte_runde;
IF(poslednji='1')THEN
    signalizacija_da_se_ide_u_UBIout<='1';
ELSE
    signalizacija_da_se_ide_u_UBIout<='0';
END IF;
```

Inkrementira se promenljiva *duzina_msg_bloka*, koja će se koristiti za generisanje Tweak vrednosti. Zatim se vrše već opisane procedure *procedura_Obradi_rec* i *procedura_Generisi_Subkey* (potrebna jer svaki blok poruke ima jedinstven podključ, dok se kao ulazna promenljiva ključ sada koristi ključ koji je dobijen nakon procedure *krajnji_XOR*). U ovom stanju se kao kod prijema prvog bloka poruke vrši provera da li je novopristigli blok poruke ujedno i poslednji blok poruke. Stanje automata se podešava na stanje *obrada_nulte_runde* kako bi se novopristigli blok poruke obradio u 72 runde, dok signal *mogu_da_primim_blok_poruke* dobija vrednost 0.

Kada se dizajn nađe u stanju *ulazi_u_UBI_OUT*, izvršava se sledeći kod:

```
finalna_obrada_u_toku<='1';
radi_ubi_out:='1';
procedura_Generisi_Subkey(SUBKEY_REGISTAR, kljuc,
     radi_ubi_out, prvi, poslednji, duzina_msg_bloka);
stanje_automata<= UBI_out_runda;
```

Signal *finalna_obrada_u_toku* se postavlja na 1, i signalizira da se stiglo do obrade poruke u UBI OUT bloku, tj. da je završen UBI MSG blok. Promenljiva *radi_ubi_out* se postavlja na 1, i koristi se kod generisanja Tweak vrednosti. Procedura *procedura_Generisi_Subkey* je opisana, a ovde se poziva kako bi dala jedinstven podkjuč za UBI OUT blok. Stanje automata se zatim prebacuje u *UBI_out_runda.*

Kada se dizajn nađe u stanju *UBI_out_runda*, izvršava se sledeći kod:

```
IF brojac72_varijabla = 0 THEN
    procedura_Runda_0_out(SUBKEY_REGISTAR, REGISTAR,
         brojac72_varijabla);
    brojac72_varijabla:=brojac72_varijabla+1;
    stanje_automata<= UBI_out_runda;
ELSE
    procedura_Runda_1_71(SUBKEY_REGISTAR, REGISTAR,
        brojac72_varijabla);
    brojac72_varijabla:=brojac72_varijabla+1;
    IF ((brojac72_varijabla+1) MOD 4 = 0 ) THEN
        procedura_Siftuj_Subkey(SUBKEY_REGISTAR);
    END IF;
    IF (brojac72_varijabla = 72)
        stanje_automata<= gotov_jedan_blok_out_poruke;
    ELSE
        stanje_automata<=UBI_out_runda;
    END IF;
END IF;
```

Ukoliko se u ovo stanje automata ulazi iz *ulazi_u_UBI_OUT* stanja, tada se brojač rundi postavlja na 0 i izvršava se procedura *procedura_Runda_0_out.* Ova procedura radi isto što i *procedura_Runda_0,* samo što je kod *procedura_Runda_0_out* ulazna reč postavljena na 0. Zatim se brojač rundi inkrementira i opet se ulazi u ovo stanje automata sa inkrementiranim brojačem. Sada se izvršava kod koji je isti kao u stanju *UBI_msg_runda* (radi se runda 1-71, vrši se ažuriranje podključa i čeka se da se uradi ukupno 72 rundi). Nakon završenih rundi stanje automata se prebacuje u *gotov_jedan_blok_out_poruke.*

Kada se dizajn nađe u stanju *gotov_jedan_blok_out_poruke*, izvršava se sledeći kod:

```
lokalna_rec := (OTHERS=>'0');
procedura_Krajnji_XOR(REGISTAR, lokalna_rec, kljuc);
brojac72_varijabla:=0;
stanje_automata<= zavrsio;
procedura_Rezultat(kljuc, hash_final);
```

Promenljiva *lokalna_rec* dobija vrednost sve nule kako bi se odradila procedura *procedura_Krajnji_XOR* (nije potrebno raditi XOR operaciju sa promenljivom koja ima sve nule, ali potreba za XOR operacijom postoji kada je izlazni heš u bitima veći od inicijalne vrednosti što će biti prikazano za Skein 256-512). Nakon procedure *procedura_Krajnji_XOR* funkcija obrade heša kod koda Skein 256-256 je završena i heš je upisan u promenljivu *kljuc*. Brojač rundi se postavlja na 0 i radi se procedura *procedura_Rezultat.* Ova procedura upisuje vrednost heša iz promenljive *kljuc* u promenljivu *hash_final* koji je istog tipa kao signal *izlaz.* Menja se stanje automata u stanje *zavrsio.*

Kada se dizajn nađe u stanju *zavrsio*, izvršava se sledeći kod:

```
izlaz<=hash_final;
preuzmi<='1';
```

```
finalna_obrada_u_toku<='0';
radi_ubi_out:='0';
stanje_automata<= idle;
```

U stanju *zavrsio* heš se postavlja na izlaz, aktivira se signal *preuzmi*, i signali *finalna_obrada_u_toku* i *radi_ubi_out* se postavljaju na 0. Stanje automata se zatim postavlja na *idle.*

4.5. Razlike u kodu za ostale duzine hes vrednosti

Kako postoje određene razlike u koracima algoritma između Skein 256-224, Skein 256-256, Skein 256-384 i Skein 256-512, postoje i razlike u VHDL kodu.

4.5.1. Razlika u kodu između Skein 256-256 i Skein 256-224

Skein 256-224 koristi Skein 256-256 za dobijanje heša, i nakon toga radi funkciju odsecanja. Stoga su razlike u kodu minimalne.

Prva razlika se ogleda u *IV* konstanti. *IV* konstanta je postavljena na odgovarajući *IV* za Skein 256-224 koji glasi:

```
CONSTANT iv: STD_LOGIC_VECTOR (255 DOWNTO 0):=
X"C6098A8C9AE5EA0B876D568608C5191C99CB88D7D7F53884384BDDB1AEDDB5DE";
```

Pošto se kao izlaz koristi signal *izlaz* tipa *STD_LOGIC_VECTOR(223 DOWNTO 0),* potrebno je pravilno dodeliti dobijeni 256-bitni heš ovom signal. To se postiže pravilnim odscanjem varijable *hash_final* kada se dizajn nađe u stanju *zavrsio.*

```
izlaz<=hash_final(255 DOWNTO 32);
```

4.5.2. Razlika u kodu između Skein 256-256 i Skein 256-512

Kod koda za Skein 256-512 javlja se slučaj da je bitska dužina heša na izlazu veća od dužine bloka. To znači da Skein 256-512 mora da koristi više UBI OUT blokova za obradu izlaza iz MSG OUT bloka. Sledi opis razlika u kodu između Skein 256-256 i Skein 256-512.

Prva razlika između Skein 256-256 i Skein 256-512 se ogleda u *IV* konstanti. *IV* konstanta je postavljena na odgovarajući *IV* za Skein 256-224 koji glasi:

```
CONSTANT iv: STD_LOGIC_VECTOR (255 DOWNTO 0):=
X"C4CE5631EA6550429BBEEFDC80F03B55771E5CBFA3DD7ED0BE5B58CB3DAB065D";
```

Sledeća razika razlika je u proceduri *procedura_Runda_0_out:*

```
PROCEDURE procedura_Runda_0_out(SIGNAL SUBKEY_REGISTAR: INOUT stanje_niz_SY;
SIGNAL REGISTAR: OUT stanje_niz; VARIABLE brojac72_varijabla: IN INTEGER;
VARIABLE brojac_out :IN STD_LOGIC) IS
        VARIABLE pomocni_registar:stanje_niz;
    BEGIN
        pomocni_registar(0):= X"0000000000000000";
        pomocni_registar(1):= X"0000000000000000";
        pomocni_registar(2):= X"0000000000000000";
        pomocni_registar(3):= X"0000000000000000";
```

```
    IF (brojac_out = '1') THEN
        pomocni_registar(0):= X"0000000000000001";
    END IF;
    ADD_SUBKEY(SUBKEY_REGISTAR, pomocni_registar);
    MIX(pomocni_registar, brojac72_varijabla);
    PERMUTE(pomocni_registar);
    REGISTAR<=pomocni_registar;
END procedura_Runda_0_out;
```

U proceduri *procedura_Runda_0_out* ubačen je deo koda koji proverava da li se obrađuje nulti ili prvi UBI OUT blok:

```
    IF (brojac_out = '1') THEN
        pomocni_registar(0):= X"0000000000000001";
    END IF;
```

Ukoliko se obrađuje nulti UBI OUT blok, nema promene u odnosu na proceduru *procedura_Runda_0_out* kod koda za Skein 256-256. Ukoliko se obrađuje prvi UBI OUT blok, prvi element ulazne reči u blok se postavlja na vrednost 1. Za proveru koji je UBI OUT blok u obradi koristi se promenjliva b*rojac_out*.

Slede razlike koje se nalaze u top-level delu. Dodatno je ubačena promenljiva *brojac_out*. Uloga ove promenljive je da broji UBI OUT blokove (da li se obrađuje nulti ili prvi blok) i kontroliše kod u procedurama u zavisnosti od stanja u kome se nalazi.

```
VARIABLE brojac_out: STD_LOGIC :='0';
```

Uvedena je dodatna promenljiva *izlaz_iz_UBI_MSG_bloka* koja je tipa *stanje_niz*. Uloga ove promenljive je da pamti stanje na izlazu iz UBI MSG bloka, pošto oba UBI OUT bloka koriste ovo stanje za dalju obradu (blokovi UBI OUT rade serijski, a oba bloka uzimaju ovu vrednost kao ključ).

```
VARIABLE izlaz_iz_UBI_MSG_bloka: stanje_niz;
```

Uvedena je promenljiva *hash_final_512*. Koristi se pri krajnjoj obradi heš vrednosti i u nju se upisuju pojedinačne 256-bitne heš vrednosti sa izlaza UBI OUT blokova. Pritom je istog tipa kao i signal koji se šalje na izlaz, pa se lako može dodeliti izlaznom signalu.

```
VARIABLE hash_final_512: STD_LOGIC_VECTOR (511 DOWNTO 0);
```

Slede razlike u kodu kod konačnog automata.

Kada se automat nađe u stanju *gotov_jedan_blok_poruke* izmenjen je deo koda*:*

```
IF(signalizacija_da_se_ide_u_UBIout='1')THEN
    izlaz_iz_UBI_MSG_bloka:=kljuc;
    stanje_automata<= ulazi_u_UBI_OUT;
ELSE
```

Ukoliko se signalizira da iz ovog stanja treba preći u stanje *ulazi_u_UBI_OUT* potrebno je sačuvati izlaz iz UBI MSG bloka u promenljivu *izlaz_iz_UBI_MSG_bloka.* Ova potreba se javlja jer prvi UBI OUT blok mora da koristi ovu vrednost kao ulaz ključa u obradi.

Kada se automat nađe u stanju *gotov_jedan_blok_out_poruke:*

```
lokalna_rec := (OTHERS=>'0');
```

```
IF (brojac_out= '1') THEN
    lokalna_rec := X"0000000000000001
                     0000000000000000
                     0000000000000000
                     0000000000000000";
END IF;
procedura_Krajnji_XOR(REGISTAR, lokalna_rec, kljuc);
brojac72_varijabla:=0;
IF (brojac_out='0') THEN
    lokalna_rec := lokalna_rec + '1';
    stanje_automata<= ulazi_u_UBI_OUT;
    procedura_Rezultat(kljuc, hash_final);
    hash_final_512(511 DOWNTO 256):=hash_final;
    kljuc:=izlaz_iz_UBI_MSG_bloka;
    brojac_out:= '1';
ELSE
    stanje_automata<= zavrsio;
    procedura_Rezultat(kljuc, hash_final);
    hash_final_512(255 DOWNTO 0):=hash_final;
END IF;
```

U ovom stanju dodate su dve IF grane u kodu. Prva koja glasi:

```
IF (brojac_out= '1') THEN
    lokalna_rec := X"0000000000000001
                     0000000000000000
                     0000000000000000
                     0000000000000000";
END IF;
```

Prva grana služi za proveru koji se UBI OUT blok obrađuje. Ukoliko je prvi blok u pitanju ulazna reč u blok nije niz nula, već prvi element niza treba da dobije vrednost 1. Dobijena vrednost koristiće se u proceduri *procedura_Krajnji_XOR*. Druga IF-ELSE grana spaja dva dobijena heša od 256 bita:

```
IF (brojac_out='0') THEN
    lokalna_rec := lokalna_rec + '1';
    stanje_automata<= ulazi_u_UBI_OUT;
    procedura_Rezultat(kljuc, hash_final);
    hash_final_512(511 DOWNTO 256):=hash_final;
    kljuc:=izlaz_iz_UBI_MSG_bloka;
    brojac_out:= '1';
ELSE
    stanje_automata<= zavrsio;
    procedura_Rezultat(kljuc, hash_final);
    hash_final_512(255 DOWNTO 0):=hash_final;
END IF;
```

Ukoliko se završila obrada prvog bloka, *brojac_out* još uvek se nalazi u stanju 0. Ulazi se u prvi deo IF-ELSE petlje gde se poziva procedura *procedura_Rezultat* koja vraća vrednost 256-bitnog heša, tj. dobija se prva polovina heša. Ovaj deo heša se zatim upisuje u 256 bita najveće težine 512-bitnog heša (promenljiva *hash_final_512*). Ključ za prvi UBI OUT blok se stavlja na vrednost izlaza iz UBI MSG bloka (zapamćen u promenljivoj *izlaz_iz_UBI_MSG_bloka*), lokalna reč se postavlja za ulaznu reč u prvi UBI OUT blok, brojač dobija vrednost 1. Stanje automata se postavlja na *ulazi_u_UBI_OUT* kako bi se još jednom izvršio UBI OUT blok.

Nakon izvršenog prvog UBI OUT bloka, *brojac_out* ima vrednost 1, pa se ulazi u ELSE granu koda. Tu se zatim radi procedura *procedura_Rezultat* koja daje drugu polovinu 512-bitnog heša. Ovaj heš se smešta na 256 bita najmanje težine promenljive *hash_final_512*. Stanje atomata menja se u stanje *zavrsio*.

Kada se stanje automata nađe u stanju *zavrsio*, potrebno je poslati 512-bitni heš na izlaz pa je promena u kodu sledeća:

```
izlaz<=hash_final_512;
```

Ovde je uvedena izmena da se umesto prosleđivanja promenljive *hash_final* na izlaz, vrši prosleđivanje promenljive *hash_final_512*.

4.5.3. Razlika u kodu između Skein 256-512 i Skein 256-384

Skein 256-384 koristi Skein 256-512 za dobijanje heša, i nakon toga radi funkciju odsecanja, pa su i u ovom slučaju razlike u kodu minimalne.

Prva razlika se ogleda u *IV* konstanti. *IV* konstanta je postavljena na odgovarajući *IV* za Skein 256-384 koji glasi:

```
    CONSTANT iv: STD_LOGIC_VECTOR (255 DOWNTO 0):=
X"8F8F63673A7B611D5C3F50B7C9DA10A6D88BAD1FD8A81272F1AAFD31293EFCBB";
```

Pošto se kao izlaz koristi signal *izlaz* tipa *STD_LOGIC_VECTOR(383 DOWNTO 0)*, potrebno je pravilno dodeliti dobijeni 512-bitni heš ovom signal. To se postiže pravilnim odscanjem varijable *hash_final* kada se dizajn nađe u stanju *zavrsio*.

```
izlaz<=hash_final_512(511 DOWNTO 128);
```

5. OPIS PERFORMANSI I VERIFIKACIJA DIZAJNA

5.1. Opis performansi

Izvršavanjem procesa analize i sinteze u slučaju sve četiri dužine izlaza, dobijene su informacije o performansama svake od implementacija. Proces analize i sinteze izvršen je u ISE razvojnom okruženju za FPGA čipove proizvođača Xilinx. U pitanju je dobra procena vrednosti i one su date u tabeli 5.1.1. Za sve četiri dužine izlaza izabran je uređaj XC6VLX760 familije Virtex 6.

Naziv	Skein 256-224	Skein 256-256	Skein 256-384	Skein 256-512
Broj slajs registara	1987 (1%)	2051 (1%)	2564 (1%)	2820 (1%)
Broj slajs LUT-ova	13250 (2%)	13283 (2%)	13732 (2%)	13798 (2%)
Broj potpuno iskorišćenih parova LUT-FF	1339 (9%)	1400 (10%)	1806 (12%)	1924 (13%)
Broj globalnih taktova (BUFG/BUFGCTRL)	1 (3%)	1 (3%)	1 (3%)	1 (3%)
Broj pinova	487/1760 (28%)	519/1760 (29%)	647/1760 (37%)	775/1760 (44%)
Maksimalna frekvencija	12.805MHz	12.805MHz	12.805MHz	12.805MHz

Tabela 5.1.1. Procenjene performanse za različite dužine izlaza

Za svaku od četiri implementacije korišćen je jedan globalni takt, *clk*. Može se primetiti da povećanjem dužine izlaznog heša dizajn troši više resursa. Registri se prvenstveno troše na čuvanje stanja, pa je otuda nešto veća potrošnja registarskih bita u slučajevima većih izlaza jer oni imaju veći broj bita u stanju. Prema podacima iz tabele 5.1.1. može se primetiti da se broj pinova na čipu menja sa povećanjem veličine (odnosno broja bita) izlaza, što je jednim delom i posledica većeg rezultujućeg heša.

Ovakva implementacija Skein algoritma predstavlja dizajn koji bi trebalo da zauzima najmanju površinu, tj. da iskorišćava najmanje resursa na čipu u odnosu na ostale implementacije. Ovim pristupum, samo jedna poruka može da se procesira istovremeno. Ovakvim dizajnom smanjuje se protok heširanja. Maksimalna frekvencija je ista za bilo koju dužinu izlaza. Ova realizacija zahteva nešto kompleksniju kontrolnu logiku koja podrazumeva uvođenje mašine stanja koja će da broji runde, kao i stanja prilikom svake od rundi. I pored ove dodatne kontrolne logike ušteda u hardverskim resursima koja se ostvari implementacijom je značajna.

5.2. Verifikacija dizajna

Verifikacija će se obaviti korišćenjem fajla sa test vektorima sa sajta instituta NIST [9] preuzeti sa zvaničnog sajta instituta NIST pod imenom *skein_golden_kat_short_internals.txt.* Celokupan fajl će biti priložen putem CD-a. U pomenutom fajlu prikazane su vrednosti stanja nakon svakog izvršenog bloka. Finalno, obaviće se verifikacija celokupnog dizajna

Test vektor koji se koristi kao poruka je niz brojeva različitih duzina (256, 512, 1024 bita), predstavljenih u obliku bajtova u heksadecimalnom obliku: 01 02 03 04...40. Verifikacija dizajna biće prikazana za Skein 256-256 verziju algoritma za test vektor od 1024 bita (četiri bloka poruke).

U fajlu *skein_golden_kat_short_internals.txt*, početno stanje je sledeće:

Message data:

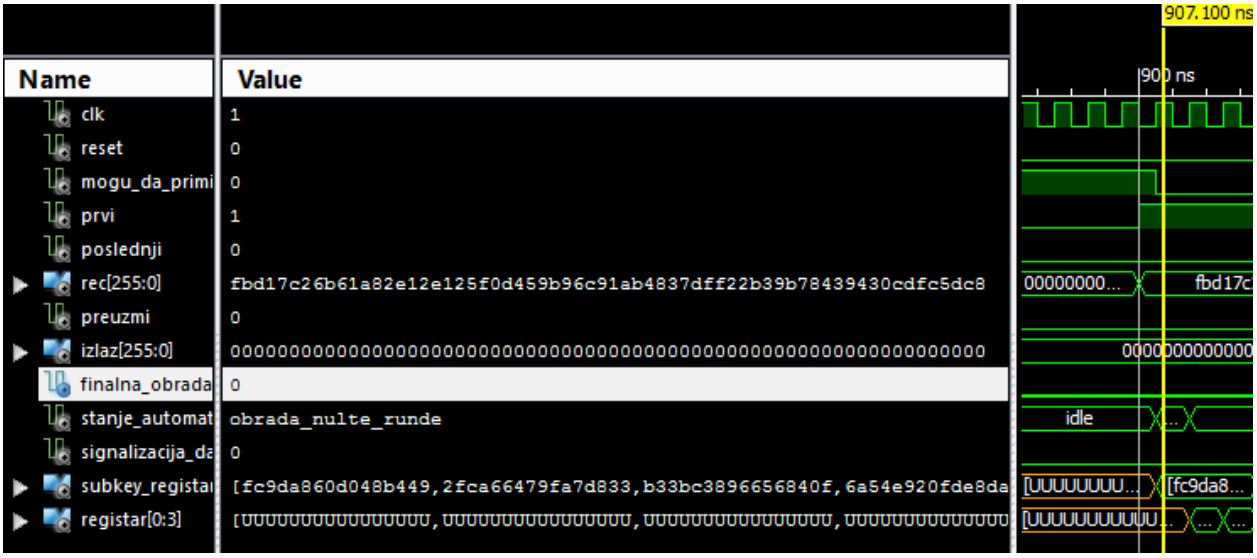
FBD17C26B61A82E1 2E125F0D459B96C9 1AB4837DFF22B39B 78439430CDFC5DC8

78BB393A1A5F79BE F30995A85A129233 39BA8AB7D8FC6DC5 FEC6F4ED22C22BB

E7EB61981892966D E5CEF576F71FC7A8 0D14DAB2D0C03940 B95B9FB3A727C66A

6E1FF0DC311B9AA2 1A3054484802154C 1826C2A27A091415 2AEB76F1168D4410

Obzirom da je dužina bita za obradu četiri puta veća od broja bita koji mogu da se obrade u jednom bloku, UBI MSG blok će biti pozvan četiri puta za poruku. Za datu početnu vrednost poruke koja se obrađuje u blokovima od po 256 bita, signal *rec* treba da sadrži prvih 256 bita poruke, što je prikazano na slici 5.2.1.



Slika 5.2.1. Početno stanje signala *rec*

Napomena: Ova slika i sve naredne dobijene su korišćenjem opcije Print Screen, nakon pokretanja ISim simulatora. Poklapanjem podataka iz fajla *skein_golden_kat_short_internals.txt* i podataka dobijenim u ISim simulatoru verifikovaće se dizajn.

Sledeće stanje koje je dato u fajlu *skein_golden_kat_short_internals.txt* je stanje na izlazu nakon obrade prvog bloka poruke i glasi:

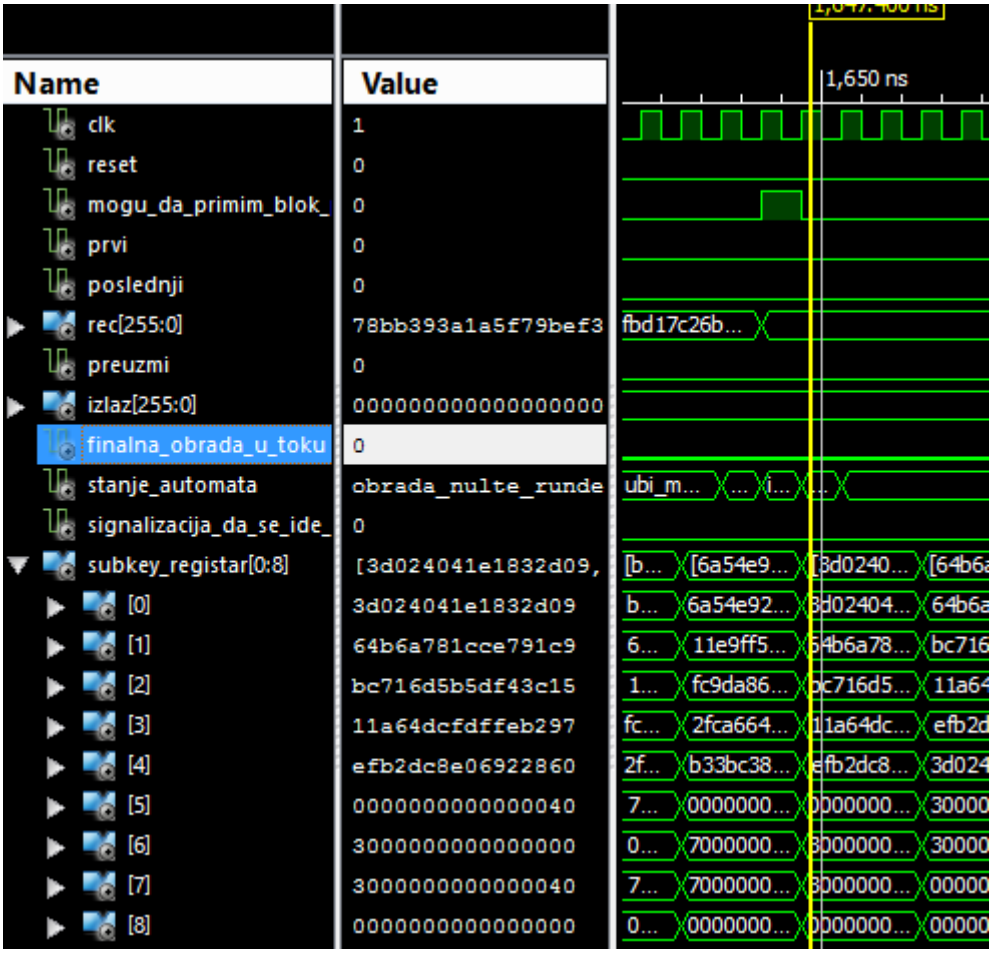
Skein-256: [state after plaintext feedforward] =

3D024041E1832D0 64B6A781CCE791C9 BC716D5B5DF43C15 11A64DCFDFFEB297

Takođe pored ovog stanja, na siglal *rec* dovodi se sledeći blok porukeod 256 bita, tj. biti:

78BB393A1A5F79BE F30995A85A129233 39BA8AB7D8FC6DC5 FEC6F4ED22C22BB

Stanje na izlazu prvog MSG UBI bloka u dizajnu predstavlja ključ na ulazu sledećeg bloka, pa je ovaj stanje vidljivo na slici 5.2.2. ako se posmatra signal *subkey_registar* na pozicijama 0-3.



Slika 5.2.2. Izlaz iz prvog UBI MSG bloka smešten u signalu *subkey_registar* [0-3]

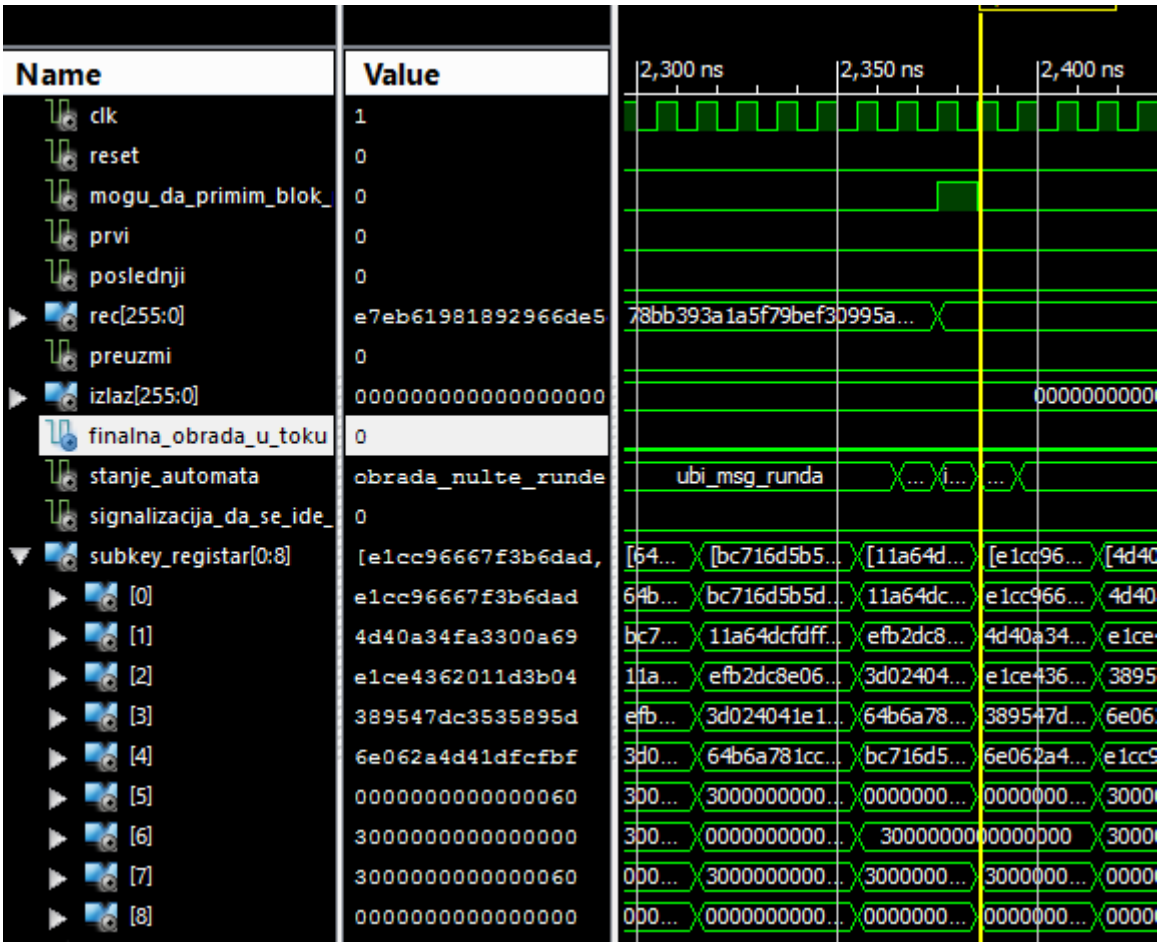
Sa slike 5.2.2 delimično se može videti da se stanje signala *rec* (tj. prvih 64 bita poruke) promenilo na stanje signala koje je predviđeno za drugi blok poruke nakon što se signal *mogu_da_primim_blok_poruke* podigao na 1.

Sledeće stanje koje je dato u fajlu *skein_golden_kat_short_internals.txt* je stanje na izlazu nakon obrade drugog bloka poruke i glasi:

Skein-256: [state after plaintext feedforward] =

E1CC96667F3B6DAD 4D40A34FA3300A69 E1CE4362011D3B04 389547DC3535895D

Stanje na izlazu drugog MSG UBI bloka u dizajnu predstavlja ključ na ulazu sledećeg bloka, pa je ovo stanje vidljivo na slici 5.2.3. ako se posmatra signal *subkey_registar* na pozicijama 0-3. Sa slike se može videti i da je stanje signala *rec* (tj. prvih 64 bita poruke) postavljeno na bite predviđene za treći blok poruke.



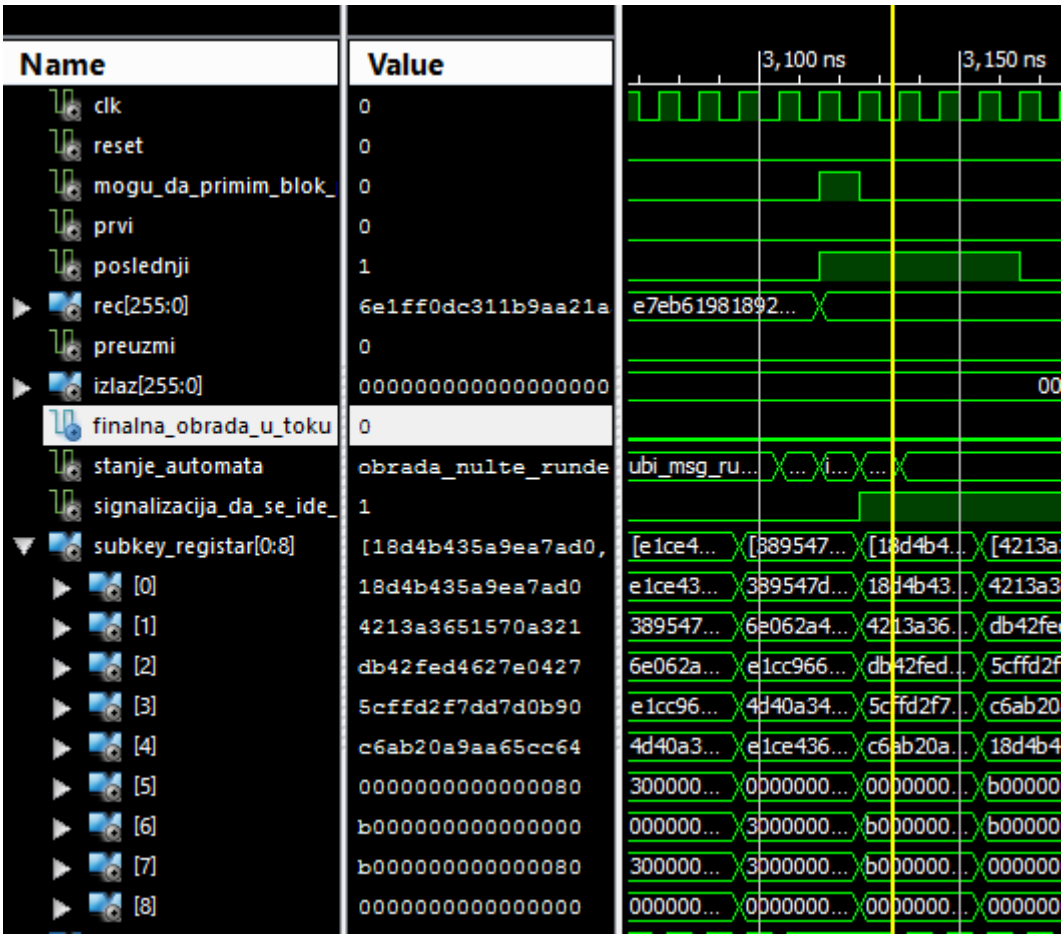
Slika 5.2.3. Izlaz iz drugog UBI MSG bloka smešten u signalu *subkey_registar* [0-3]

Sledeće stanje koje je dato u fajlu *skein_golden_kat_short_internals.txt* je stanje na izlazu nakon obrade trećeg bloka poruke i glasi:

Skein-256: [state after plaintext feedforward] =

18D4B435A9EA7AD0 4213A3651570A321 DB42FED4627E0427 5CFFD2F7.DD7D0B90

Stanje na izlazu trećeg MSG UBI bloka u dizajnu predstavlja ključ na ulazu sledećeg bloka, pa je ovo stanje vidljivo na slici 5.2.4. ako se posmatra signal *subkey_registar* na pozicijama 0-3. Sa slike se može videti i da je stanje signala *rec* (tj. prvih 64 bita poruke) postavljeno na bite predviđene za četvrti blok poruke.



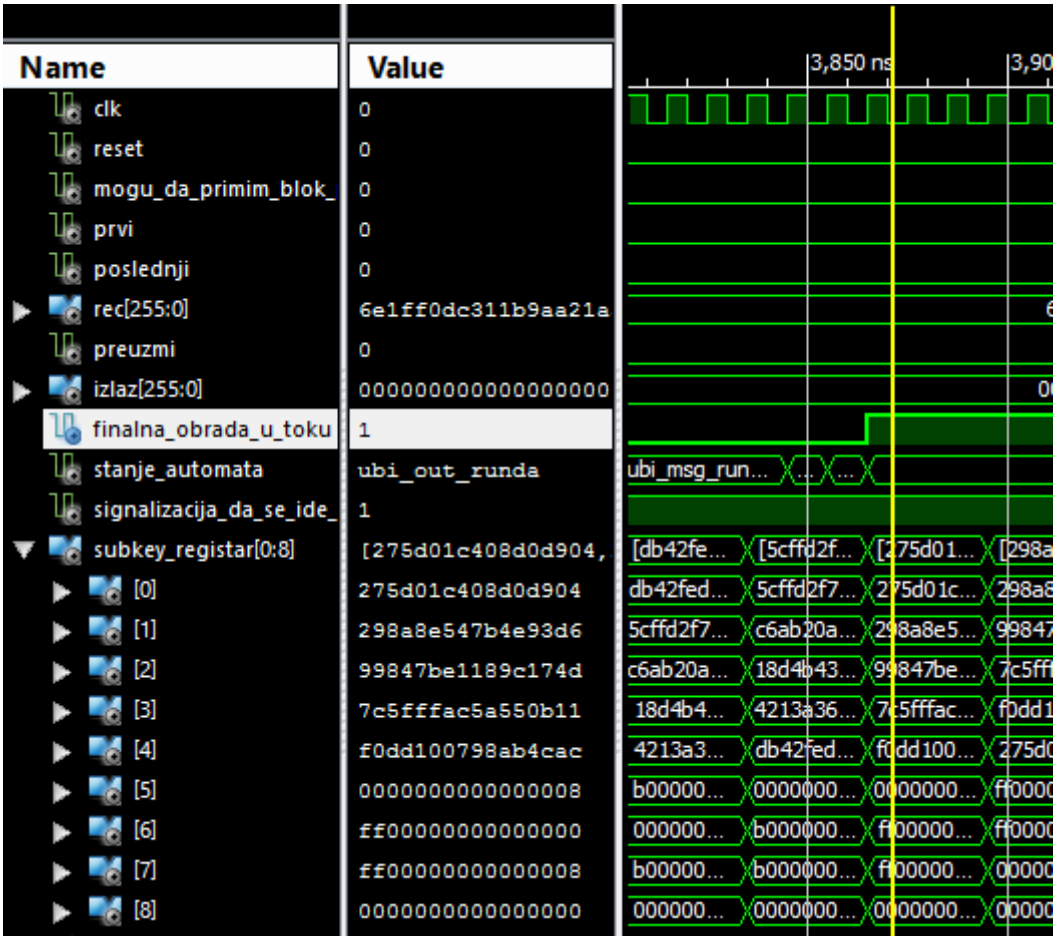
Slika 5.2.4. Izlaz iz trećeg UBI MSG bloka smešten u signalu *subkey_registar* [0-3]

Sledeće stanje koje je dato u fajlu *skein_golden_kat_short_internals.txt* je stanje na izlazu nakon obrade četvrtog bloka poruke i glasi:

Skein-256: [state after plaintext feedforward] =

275D01C408D0D904 298A8E547B4E93D6 99847BE1189C174D 7C5FFFAC5A550B11

Stanje na izlazu četvrtog MSG UBI bloka u dizajnu predstavlja ključ za UBI OUT blok, pa je ovo stanje vidljivo na slici 5.2.5. ako se posmatra signal *subkey_registar* na pozicijama 0-3. Sa slike se može videti i da je stanje signala *rec* ostalo nepromenjeno pri ulazu u UBI OUT blok, kao i da je vrednost signala *finalna_obrada_u_toku* postavljena na 1.



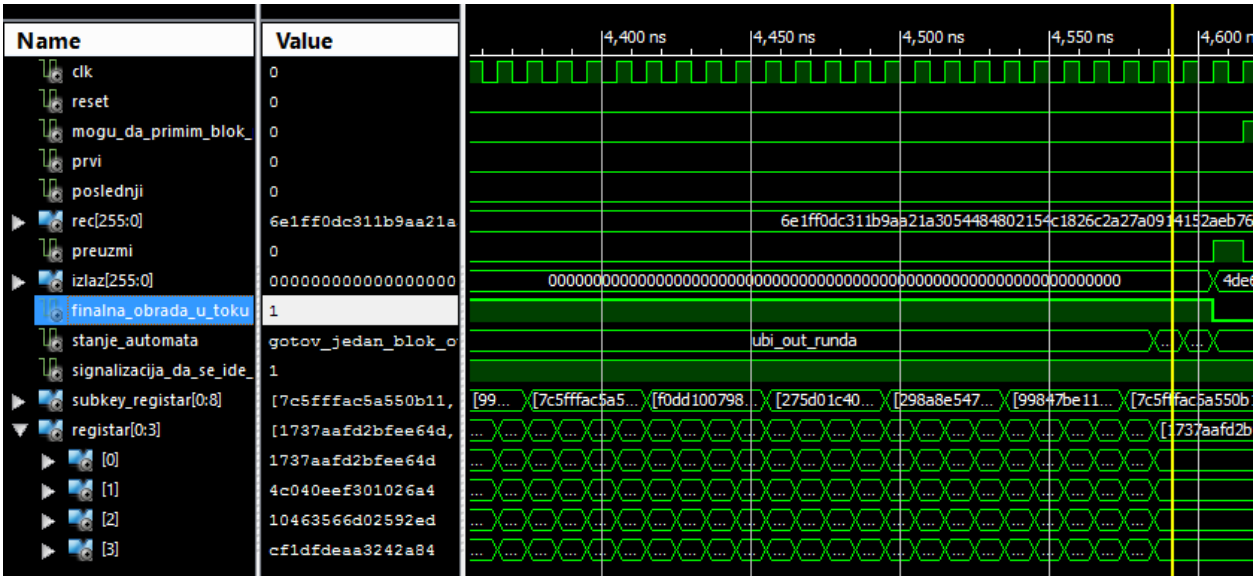
Slika 5.2.5. Izlaz iz četvrtog UBI MSG bloka smešten u signalu *subkey_registar* [0-3]

Sledeće stanje koje je dato u fajlu *skein_golden_kat_short_internals.txt* je stanje na izlazu UBI OUT bloka

Skein-256: [state after plaintext feedforward] =

1737AAFD2BFEE64D 4C040EEF301026A4 10463566D02592ED CF1DFDEAA3242A84

Stanje na izlazu UBI OUT bloka se može videti na slici 5.2.6. ako se posmatra signal *registar* na pozicijama 0-3. Sa slike se može videti da će uskoro biti aktivan i signal *preuzmi*, nakon što se prođe kroz stanja automata *gotov_jedan_blok_ubi_out_poruke* i *zavrsio*.



Slika 5.2.6. Izlaz iz UBI OUT bloka smešten u signalu *registar* [0-3]

Sledeće stanje koje je dato u fajlu *skein_golden_kat_short_internals.txt* je stanje heš vrednosti koje se dobija primenom algoritma za Skein 256-256 i glasi:

Skein-256: Final output=

4DE6FE2BFDAA3717 A4261030EF0E044C ED9225D066354610 842A24A3EAFD1DCF

Ovo stanje se razlikuje od stanja na izlazu UBI OUT bloka zbog dodatne transformacije koda koja se radi nakon UBI OUT bloka pre slanja heš vrednosti na izlaz. Krajnji rezultat, tj. heš vrednost poruke smeštena u signalu *izlaz* može se videti na slici 5.2.7.

Slika 5.2.7. Heš vrednost poruke smeštena u signalu *izlaz*

Poklapanjem podataka o stanju na izlazu i heš vrednosti, verifikovan je celokupan dizajn.

Uspešno je testirano ponašanje dizajna i u slučaju drugih ulaznih podataka (poruka), kao i ponašanje dizajna za različite dužine heš izlaza. Zbog analogije u principu testiranja, ali i da bi se izbegle redudantnosti u tekstu teze, prikazana je verifikacija za samo jednu dužinu heš izlaza.

6. ZAKLJUČAK

Funkcije za izračunavanje sažetaka poruke su funkcije koje za ulazne vrednosti proizvoljne dužine računaju vrednost fiksne dužine koju nazivamo sažetak ili heš. Trenutna primena ovih funkcija u sistemima komunikacije je vrlo raširena. Njima je moguće brzo proveriti celovitosti podataka, skratiti pretraživanje baza podataka, računati pristupne kodove, a neizbežan su deo i sistema za digitalno potpisivanje.

Skein je vrlo jednostavan algoritam za implementaciju. Svaki od koraka algoritma može se raščlaniti na niz jednostavnih operacija: XOR operacija, rotiranje sadržaja promenljive, obična dodela vrednosti. One čine algoritam vrlo jednostavnim za realizaciju. Dizajniran je tako da bude brz, siguran, razumljiv, otporan na kolizije i fleksibilan. Razvijen je za tri osnovne dužine ulaza 256, 512 i 1024 bita, a podržava promenljivu izlaznu dužinu u zavisnosti od potreba implementacije.

Realizovana implementacija zahteva dosta vremena za izvršavanje analize i sinteze jer iako se koriste proste operacije kombinaciona logika je suviše velika. Brzina Skein algoritma bi se mogla povećati korišćenjem hash-tree moda i paralelne implementacije. Takođe Skein algoritima podržava i veliki broj dodatnih funkcionalnosti koje lako mogu biti implementirane, ali nisu predmet ovoga rada.

Krajem 2012. godine američki Nacionalni institut za informacije i tehnologiju je, nakon petogodišnjeg takmičenja, proglasio pobednika izbora za novi nominalni heš algoritam, algoritam Keccak. Iako Skein nije odneo titulu pobednika njegove preformanse i mogućnost implementacije doveli su ga u samo finale takmičenja.

LITERATURA

[1] Wikipedia: Kriptografija. Preuzeto sa: http://sh.wikipedia.org/wiki/Kriptografija

[2] Milica Kovinić, „Uvod u kriptografiju i infrastrukturu javnih kljuceva“, 2010.

[3] Wikipedia: Avalanche effect. Preuzeto sa: https://en.wikipedia.org/wiki/Avalanche_effect

[4] Iva Jeličić, „Programsko ostvarenje novih funkcija za izračunavanje sažetka poruke“, 2014.

[5] Wikipedia: SHA-1. Preuzeto sa: https://en.wikipedia.org/wiki/SHA-1

[6] Wikipedia: SHA-2. Preuzeto sa: https://en.wikipedia.org/wiki/SHA-2

[7] Wikipedia: SHA-3. Preuzeto sa: https://en.wikipedia.org/wiki/SHA-3

[8] The Skein Hash Function Family, Version 1.3. [Online]. Preuzeto sa: http://www.skein-hash.info/downloads

[9] NIST Skein CD [Online]. Preuzeto sa: http://www.skein-hash.info/sites/default/files/skein_NIST_CD_101308.zip