W O J S K O W A A K A D E M I A T E C H N I C Z N A

im. Jarosława Dąbrowskiego

P R A C A D Y P L O M O W A

Analiza implementacji algorytmu HIEROCRYPT w strukturach programowalnych.

STUDIA WYŻSZE

ppor. Marcin ROGAWSKI
(imię i nazwisko dyplomanta)

WYDZIAŁ CYBERNETYKI
(wydział)

INSTYTUT MATEMATYKI I KRYPTOLOGII
(instytut)

dr inż. Piotr BORA
(imię i nazwisko kierownika pracy)

W A R S Z A W A - 2003

Spis treści:

I. Wybrane aspekty kryptologii – szyfrowanie symetryczne.

1.1 Zarys historyczny.

Problem efektywnego i poufnego sposobu przekazywania informacji już od starożytności pojawiał się wszędzie tam, gdzie istniało niebezpieczeństwo przejęcia strategicznych wiadomości przez nieprzyjaciela. Realizacja udanych operacji wojskowych, sprawne rządzenie, skuteczność zabiegów ekonomiczno – marketingowych, wszystko to jest uzależnione nie tylko od nieznajomości utajnionej informacji przez każdego, kto nie jest do tego powołany, ale także od integralności, niezaprzeczalności utajnionych danych oraz uwierzytelnienia w przypadku dostępu do nich.

Pierwszą chronologicznie funkcją kryptograficzną realizowaną dla wiadomości, na którą pojawiło się zapotrzebowanie, była jej tajność. Już od początku uzyskiwano ją za pomocą szyfrów symetrycznych, które jednak w niczym nie przypominały współczesnych algorytmów szyfrujących. Przekształcenie Viegenera (Algorytm Cezara możemy uznać tylko za książkę kodową) pomimo że korzysta on z jednej z dwóch najważniejszych technik wykorzystywanych w kryptografii symetrycznej – podstawiania, nie jest w stanie oprzeć się współczesnym narzędziom kryptoanalitycznym. Posiada ono jednak istotną cechę, która nakazuje nam umieścić je w rodzinie szyfrów symetrycznych – szyfrowanie i deszyfrowanie wiadomości odbywa się za pomocą tego samego, tajnego klucza.

Równolegle z pracą ludzi utajniających depesze, zawsze pracowały zespoły próbujących poznać zakodowaną wiadomość. Jednym z najbardziej spektakularnych sukcesów w dziejach kryptoanalizy było złamanie przez francuskiego matematyka, niemieckiego szyfrogramu w czasie niemieckiej ofensywy na Paryż podczas I wojny światowej. Umożliwiło to poznanie rozkazów przeciwnika oraz w konsekwencji skuteczną obronę przez francuskich żołnierzy. Szyfr ten nosił nazwę ADFGVX i wykorzystywał on zarówno podstawianie jak i permutację, czyli drugą najważniejszą technikę stosowaną we współczesnych szyfrach symetrycznych.

Absolutnym fenomenem okazuje się być historia ENIGMY, niemieckiego urządzenia szyfrującego, na którym opierało się bezpieczeństwo informacji przekazywanej wewnątrz systemu komunikacyjnego III Rzeszy, podczas II wojny światowej. To dzięki zespołowi polskich kryptologów, w którym główną rolę odgrywał Marian Rejewski, udawało się łamać codzienne depesze dowództwa niemieckiego i dzięki temu, wg opinii wielu fachowców, udało się skrócić wojnę o co najmniej 2 lata.

Jednak tak naprawdę „złoty wiek" kryptologii przypada na II połowę XX wieku ze szczególnym uwzględnieniem ostatnich lat. Wybuch „zimnej wojny" zdecydowanie przyspieszył

rozwój komputerów, internetu, telekomunikacji, zwiększył ilość transferowanych danych, a co za tym idzie spowodował zwiększenie potrzeby ochrony informacji niejawnej.

Do lat 70 – tych XX wieku kryptologia była jeszcze nauka tajną i specjalną, która miała jedynie zastosowanie w dyplomacji, wojsku i służbach specjalnych. Największymi osiągnięciami do tego okresu były niewątpliwie: zasada Kerkhoffsa, mówiąca o tym, że kryptoanalityk próbujący złamać nasz szyfrogram nie zna tylko klucza, jakim się posłużyliśmy oraz prawo Shannona, które dało początek teorii szacowania bezpieczeństwa szyfru.

Rozwój telekomunikacji i cyfrowego przetwarzania danych, rosnąca rola informacji i oszczędności czasu, we współczesnym świecie, spowodowała, że konieczność ochrony przechowywanych oraz przekazywanych ogólno dostępnymi kanałami wiadomości, stała się indywidualną i naturalną potrzebą każdego człowieka. Jednocześnie szybki rozwój zaawansowanych technik kryptanalitycznych sprawia, że przed projektantami kolejnych algorytmów szyfrowania pojawiają się coraz większe wymagania. Dzisiaj nie wystarcza już tylko udowadnialne bezpieczeństwo szyfru. Musi on dodatkowo spełniać warunki co do efektywności jego implementacji programowej i sprzętowej, szybkości realizowanej operacji szyfrowania/deszyfrowania, a także adapcyjności w nietypowych środowiskach.

1.2 Uzasadnienie wyboru tematu.

Celem pracy jest zaimplementowanie szyfru blokowego o nazwie Hierocrypt. Wybór motywowany jest tym, że został on zgłoszony przez grupę japońskich kryptologów z korporacji TOSHIBA jako propozycja nowego standardu szyfrowania danych dla Europy w konkursie NESSIE (ang. *New European Schems for Signature, Integrity and Encryption*). Pomimo znacznych atutów bezpieczeństwa w postaci, bardzo dużej odporności na wszelkie znane ataki kryptoanalityczne, algorytm został odrzucony podczas pierwszej fazy trwania konkursu. Powodem było udane przeprowadzenie ataku, będącego bardziej efektywnym od minimalnych oszacowań projektantów szyfru, ale także znaczna komplikacja algorytmu generowania podkluczy oraz bardzo słabe wyniki implementacji algorytmu – jeden z najwolniejszych algorytmów w całym konkursie.

Pomimo że algorytm został już zdyskwalifikowany to jednak prezentuje on aktualne trendy w kryptografii i metodach projektowania bezpiecznych szyfrów.

Wraz z rozpoczęciem badań zgłoszonych algorytmów pojawiły się ich implementacje programowe oraz realizacje sprzętowe. Ponieważ wybrany algorytm wskazywał całkiem dobre wyniki, w porównaniu z pozostałymi uczestnikami konkursu, implementacji programowej oraz

katastrofalnie słabe wyniki sprzętowe, wskazanym było, żeby zrealizować dedykowany układ scalony realizujący funkcję szyfrowania i którego głównym kryterium projektowym byłaby jak największa szybkość szyfrowania.

Implementacja algorytmu została przeprowadzona w sprzęcie, z wykorzystaniem układów programowalnych rodziny FLEX 10K firmy ALTERA. Celem nadrzędnym było uzyskanie jak największej szybkości działania układu, dodatkowo zostały wykonane analogiczne realizacje, które pozwalają na przenaszalność na układy innych firm, np. Xilinx. Osiągnąć ją można tylko poprzez realizację projektów w języku programowania HDL. Wykorzystując zaawansowane technologie specyficzne dla układów ALTERY, można osiągnąć lepsze wyniki pod względem szybkości działania, ale niestety odbywa się to kosztem możliwości przeniesienia projektu na układy innych firm.

Mój projekt jest zrealizowany w dwóch różnych odmianach architektury iteracyjnej. Wstępne założenie mówiło o możliwości realizacji także architektur: kombinacyjnej i potokowej, ale niestety stanowisko projektowe MAX PLUS II nie wspiera układów o wystarczającej pojemności dla tego typu rozwiązań.

Projekt architektury iteracyjnej nie jest związany z żadnym konkretnym środowiskiem pracy. Stanowi propozycję układu, który można wykorzystać w dowolnym systemie ochrony informacji. Stąd też pomijam zagadnienia komunikacji z użytkownikiem, sposobu dostarczania, odbierania lub składowania danych.

1.3 Podstawowe pojęcia kryptografii.

Niemal w każdej kryptologicznej publikacji przejawiają się terminy i definicje, które szczególnie przez czytelnika nie posiadającego podstawowych wiadomości kryptologicznych, mogą być nie zrozumiałe. Wprowadzenie w świat kryptologii rozpocznę dlatego od wyjaśnienia podstawowych pojęć, które są związane z tematyką i będą przejawiać się w mojej pracy.

Kryptografia jest nauką o metodach przekształcania informacji do postaci, która umożliwia odczytanie jej treści tylko przez adresata, każda inna osoba natomiast nie jest w stanie poznać zawartości przesyłanych danych. Dzięki temu dwie strony mogą porozumiewać się przy wykorzystaniu jawnego kanału łączności.

Kryptoanaliza jest nauką zajmującą się metodami łamania szyfrów, czyli odzyskiwania: klucza i porcji danych, dla przechwyconego, utajnionego tekstu. .

Kryptologia jest to nauka obejmująca kryptografię i kryptoanalizę.

Tekstem jawnym nazywamy informację, którą chce przesłać nadawca. Informacja ta może być wszelkiego rodzaju: tekst w określonym języku, dźwięk, obraz, dane numeryczne, itd.

Szyfr jest metodą utajnionego zapisywania, w której tekst jawny jest przekształcany na **tekst zaszyfrowany (szyfrogram, kryptogram)**.
Przekształcania tekstu jawnego w zaszyfrowany nazywamy **szyfrowaniem (utajnianiem, kodowaniem)**. Szyfrowanie odbywa się przy użyciu określonego algorytmu i **klucza szyfrującego**. Proces odwrotny, polegający na przekształceniu tekstu zaszyfrowanego w tekst jawny – **deszyfrowaniem (odtajnianiem, dekodowaniem)**. Deszyfrowanie jest możliwe dzięki znajomości algorytmu użytego do szyfrowania i **klucza deszyfrującego**.

Algorytm kryptograficzny jest to ciąg operacji matematycznych realizujących określone zadanie wykorzystywane w kryptografii.

Klucz kryptograficzny – ciąg symboli, od którego zależy wynik przekształcenia kryptologicznego: szyfrowania (klucz szyfrujący), deszyfrowania (klucz deszyfrujący).

Przestrzeń klucza – zakres wartości które może przyjąć klucz. Wielkość tej przestrzeni jest podstawową miarą bezpieczeństwa algorytmu szyfrującego. Atak kryptoanalityczny polegający na pełnym przeszukaniu całej przestrzeni klucza nazywany jest czasami atakiem brutalnym (ang. *brute force*).

1.4 Systemy kryptograficzne.

Kryptosystem jest to piątka (*P*, *C*, *K*, *E*, *D*), gdzie spełnione są następujące warunki:

1. *P* jest skończonym zbiorem możliwych jednostek tekstu jawnego.
2. *C* jest skończonym zbiorem możliwych jednostek szyfrogramu.
3. *K* jest przestrzenią klucza; skończonym zbiorem możliwych kluczy.
4. Dla każdego $k \in K$ istnieje reguła szyfrowania $e_k \in E$ i odpowiadająca reguła deszyfrowania $d_k \in D$. Wtedy $e_k{:}P \to C$ i $d_k{:}C \to P$ są funkcjami takimi, że $d_k(e_k(x)) = x$ dla każdego $x \in P$.

Funkcje e_k, d_k muszą być wzajemnie jednoznaczne:

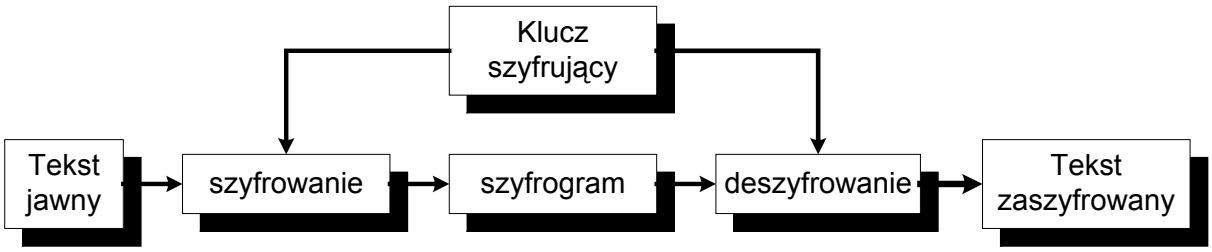
$x_1 \neq x_2 \Rightarrow e_k(x_1) \neq e_k(x_2)$

i podobnie dla funkcji d_k.

Wyróżnia się dwa typy kryptosystemów:

- kryptosystemy symetryczne (jednokluczowe, z kluczem tajnym),
- kryptosystemy asymetryczne (dwukluczowe, z kluczem jawnym).

Symetrycznym systemem kryptograficznym nazywamy system, w którym dla każdej pary odpowiadających sobie kluczy $e,d \in K$ wyznaczenie drugiego z pary kluczy na podstawie znajomości jednego z nich jest zadaniem łatwym obliczeniowo.
Klucze przekazywane są najczęściej przez zaufanego kuriera lub też transmitowane bezpiecznym kanałem łączności.

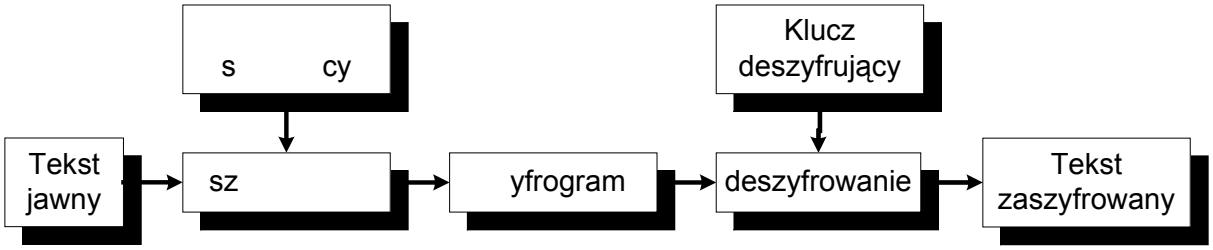


Rys.1.1: Symetryczny system kryptograficzny.

Asymetrycznym systemem kryptograficznym nazywamy system, w którym dowolna para odpowiadających sobie przekształceń, szyfrującego $e_k \in E$ i deszyfrującego $d_k \in D$ posiada własność, iż znając e_k jest praktycznie niemożliwe wyznaczenie dla losowego szyfrogramu $c \in C$

takiej wiadomości $m \in P$, że $e_k(m) = c$. Własność ta równoważna jest temu, iż jest obliczeniowo niemożliwe wyznaczenie klucza deszyfrującego na podstawie znajomości klucza szyfrującego.

W praktyce nie spotyka się systemów, w których własność ta zachodzi dla każdej pary kluczy, lecz przy ogromnych przestrzeniach klucza, z jakimi mamy do czynienia w komputerowej kryptografii, unikanie nieodpowiednich par nie stanowi zbyt wielkiego problemu.
Przykładami asymetrycznych systemów kryptograficznych są: RSA (algorytm Rivesta, Shamira, Adlemana), ElGammal



Rys.1.2: Asymetryczny system kryptograficzny.

Szyfry należące do rodziny szyfrów symetrycznych można podzielić na dwie kategorie: blokowe i strumieniowe.

Szyfry blokowe (ang. *block ciphers*) dzielą tekst jawny P na bloki P_1, P_2, .., o długości takiej jaką specyfikuje algorytm, a następnie każdy z nich jest szyfrowany przy użyciu tego samego klucza K: $E_K(P) = E_K(P_1), E_K(P_2)$....
W szyfrach tego typu jednostką szyfrowania jest grupa bitów zwana blokiem i wynikiem działania takim szyfrem na tekst jawny są bloki zaszyfrowanych wiadomości. Przykładowymi szyframi blokowymi są: DES, IDEA, Rijndael, Serpent, RC6, Mars, Twofish, Camelia, Anubis, Khazad oraz Hierocrypt.

Szyfry strumieniowe (ang. *stream ciphers*) – dzielą tekst jawny na bity p1,p2, ... a następnie każdy element ciągu tekstu jawnego jest szyfrowany odpowiadającym mu elementem z ciągu klucza tzw. strumienia klucza K: k1,k2, ...
$E_K(P) = E_{k1}(p_1), E_{k2}(p_2)$, ...
Wynikiem takiego działania jest ciąg zaszyfrowanych bitów. Przykładami takich algorytmów są: Sober, Leviathan, Snow, Lili-128 oraz stosowany w telefonii cyfrowej A5.

1.5 Algorytm DES i konkurs AES.

W latach 70-tych ubiegłego stulecia w laboratoriach IBM, powstał protoplasta algorytmu DES (ang. Data Encryption Standard). Narodowa Agencja Bezpieczeństwa Stanów Zjednoczonych wprowadziła niewielkie poprawki do budowy algorytmu, który potem stał się standardem na prawie ćwierć wieku.

Konkurs AES (ang. *Advanced Encryption Standard*), zrodził się na potrzebę zastąpienia przestarzałego, algorytmu, którego długość klucza nie była już wystarczającym zabezpieczeniem przed atakiem brutalnym. Koszt i czas takiego ataku stawał się zbyt mały, żeby był on nadal standardem bezpieczeństwa. W wyniku konkursu AES wybrano Rijndaela na algorytm XXI wieku oraz na następcę DES-a. Od kilku już lat zauważamy, że zarówno w darmowym jak i komercyjnym oprogramowaniu oraz we wszelkiego rodzaju sprzęcie krytograficznym rolę DES-a przejął potrójny DES czyli 3DES. Algorytm ten ma taką samą długość bloku oraz podwojoną długość klucza (112 bitów), w związku z czym jest nadal stosowany i uważany za bardzo dobre i bezpieczne rozwiązanie. Konkurs AES, który miał na celu znalezienie nowego standardu dla Stanów Zjednoczonych, pokazał pomysłodawcom konkursu NESSIE, że należy również zwrócić uwagę na to, że nadal potrzebnymi są szyfry, które posiadają długość bloku 64 bity. Stąd też pomysł, żeby rywalizację w kategorii szyfrów blokowych przeprowadzić w dwóch grupach: szyfrów z normalnym poziomem zabezpieczeń (długość bloku 64 bity, długość klucza 128 bitów) oraz z podwyższonym poziomem zabezpieczeń (długość bloku 128 bitów, długość klucza 128, 192, 256 bitów).

Japońscy kryptolodzy (Kenji Ohkuma, Hideo Shimizu, Hirofoumi Muratani, Fumihiko Sano, Shinichi Kawamura, Masahiko Motoyama) z Toshiba Corp. zaproponowali więc algorytmy: Hierocrypt –L1 z blokiem 64 bitowym oraz Hierocrypt –3 z blokiem 128 bitowym.

1.6 Założenia konkursu NESSIE.

Projekt NESSIE (ang. *New European Schemes for Signature, Integrity, and Encryption*) – to konkurs który ma na celu wybranie europejskich standardów kryptograficznych. W trzy letniej rywalizacji, rozpoczętej 1 stycznia 2000 roku mogą zmierzyć się projekty algorytmów kryptograficznych, z kategorii:

- szyfry blokowe (ang. *Block ciphers*),
- synchroniczne szyfry strumieniowe (ang. *Synchronous stream ciphers*),
- samosynchronizujące szyfry strumieniowe (ang. *Self-synchronising stream ciphers*),

- jednokierunkowe funkcje skrótu z kluczem(ang. *Message Authentication Codes (MACs)*),
- funkcje skrótu odporne na kolizje (ang. *Collision-resistant hash functions*),
- jednokierunkowa funkcja skrótu (ang. *One-way hash functions*),
- pseudolosowa funkcja (ang. *Families of pseudo-random functions*),
- niesymetryczny schemat szyfrowania (ang. *Asymmetric encryption schemes*),
- niesymetryczny podpis cyfrowy (ang. *Asymmetric digital signature schemes*),
- niesymetryczny schemat identyfikacji (ang. *Asymmetric identification schemes*).

Termin zgłaszania propozycji do konkursu NESSIE upłynął 29 września 2000 roku i od tamtej pory każdy może uczestniczyć w konkursie jako juror, ponieważ NESSIE ma charakter otwarty.

Ocena propozycji odbywa się w trzech turach pierwsza zakończyła się w lipcu 2001 roku, druga zakończyła się w grudniu 2002r, a wyniki konkursu będą ogłoszone w marcu 2003 r.

Kryteria oceny algorytmów kryptograficznych obejmować będą takie zagadnienia jak bezpieczeństwo szyfru, efektywność i szybkość implementacji programowej i sprzętowej.

Kryterium bezpieczeństwa

- złożoność każdego znanego ataku powinna być przynajmniej równa złożoności ataku polegającego na pełnym przeszukaniu przestrzeni klucza,
- wszystkie algorytmy kryptograficzne powinny być ocenione przez autorów. Atak, który będzie efektywniejszy od podanego przez nich będzie automatycznie eliminował z dalszej części konkursu,
- algorytmy będą oceniane także pod względem odporności na atak polegający na badaniu poboru mocy w różnych środowiskach,

Kryteria Implementacyjne:

- programowa i sprzętowa efektywność będzie porównywana pomiędzy poszczególnymi uczestnikami konkursu, ale także z istniejącymi już implementacjami algorytmów.
- szybkość wykonania kodu, rozmiar potrzebnej pamięci w różnego typu środowiskach będzie również kryterium, które będzie rozważane.
- dodatkowymi atutami może być także łatwość dostosowania się do nietypowych środowisk (np. procesor 48-bitowy).

Inne kryteria:

- prostota projektu oraz rzeczowe uzasadnienie doboru komponentów szyfru (unikanie elementów, które mają pochodzenie losowe).

Wymagania licencyjne:

- możliwość korzystania ze zwycięskiego algorytmu nie powinna być ograniczona przez żadne umowy licencyjne.

1.7 Propozycje algorytmów symetrycznych w konkursie NESSIE.

W kategorii na najlepszy symetryczny szyfr blokowy dokonano podziału na szyfry, których długość bloku wynosi 64 bity (normalny poziom zabezpieczeń) oraz na grupę szyfrów z blokiem 128 bitowym (podwyższony poziom zabezpieczeń).

W pierwszej grupie, selekcji do finałowej fazy konkursu nie przeszły algorytmy:

- CS-Cipher (CS Communication & Systèmes),
- Hierocrypt – L1 (Toshiba Corporation),
- Nimbus (Alexis Machado),
- NUSH (LAN Crypto, Int.).

CS-Cipher nie wykazywał żadnych słabości, ale był najwolniejszym jeśli chodzi o implementację programową i sprzętową. Hierocrypt –L1 okazał się również bardzo wolny, ale także został zdyskwalfikowany z powodu algorytmu generacji podkluczy oraz ataku na 3.5 z 6 rund. Nimbusa odrzucono ponieważ przeprowadzono praktyczny atak przy pomocy 2^8 wybranych tekstów i przy 2^{10} złożoności obliczeniowej ! NUSH nie był odporny na kryptanalizę liniową. Finalistami natomiast zostały algorytmy wobec, których nie było żadnych wątpliwości pod względem bezpieczeństwa oraz wskazywały względnie dobre wyniki implementacyjne. Są to:

- IDEA (Mediacrypt AG),
- Khazad (Paulo Barreto and Vincent Rijmen),
- MISTY1(Mitsubishi Electric Corporation),
- SAFER++$_{64}$ (Cylink Corporation).

W kategorii szyfrów że 128 bitami długości bloku, po pierwszej fazie odrzucono następujące algorytmy:

- NUSH (LAN Crypto, Int.),
- Grand Cru (Johan Borst),
- Noekeon (Joan Daemen, Michael Peeters, Gilles Van Assche, Vincent Rijmen),
- Q (Leslie McBride),
- Hierocrypt-3 (Toshiba Corporation),
- SC2000 (Fujitsu Laboratories LTD.),
- Anubis (Paulo Barreto and Vincent Rijmen).

NUSH w wersji 128 bitowej miał katastrofalne wyniki pod względem bezpieczeństwa. Grand Cru, który bazował na algorytmach będących finalistami z konkursu AES, okazał się najwolniejszy w implementacjach zarówno programowych jak i sprzętowych. Q nie oparł się ani kryptanalizie różnicowej, ani liniowej. W Noekeonie zaniedbano algorytm generowania podkluczy i dlatego też skutecznym okazał się atak z zależnymi kluczami. Anubis został odrzucony za duże podobieństwo do algorytmu Rijndael, SC2000 natomiast był bardzo wolny i niejasne były kryteria projektowania komponentów szyfru, wreszcie Hierocrypt-3 odrzucono z tych samych powodów co Hierocrypt-L1.

Do kolejnego etapu, podobnie jak w przypadku szyfrów z normalnym poziomem zabezpieczeń, zostały zakwalifikowane szyfry, które oparły się wysiłkom kryptoanalityków z całego świata oraz dawały dobre wyniki implementacyjne. Są to:

- SAFER++$_{128}$ (Cylink Corporation),
- Camellia (Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation),
- RC6 (RSA Security Inc.),
- SHACAL (Gemplus).

II. Układy cyfrowe programowane przez użytkownika.

2.1 Układy cyfrowe.

Systemy cyfrowe mogą się opierać o standardowe, uniwersalne elementy wielkiej skali integracji, które mogą być uzupełniane elementami małej i średniej skali integracji. System cyfrowy zbudowany z nich, składa się często z wielu odrębnych układów, co powoduje, że drastycznie zwiększają się koszty produkcji takiego rozwiązania. Taka sytuacja spowodowała rozwój nowego podejścia do procesu wytwarzania cyfrowych układów. Preferuje wytworzenie jednego układu scalonego, który spełniałby rolę nawet najbardziej złożonego układu cyfrowego, w jednym, wyspecjalizowanym układzie cyfrowym ASIC (ang. Application Specific Integrated Circuit). Rozdział ten będzie poświęcony charakterystyce układów cyfrowych ze szczególnym uwzględnieniem układów FLEX 10K. Własności tej rodziny sprawiają bowiem, że najlepiej pasuje ona do propozycji mojego projektu.

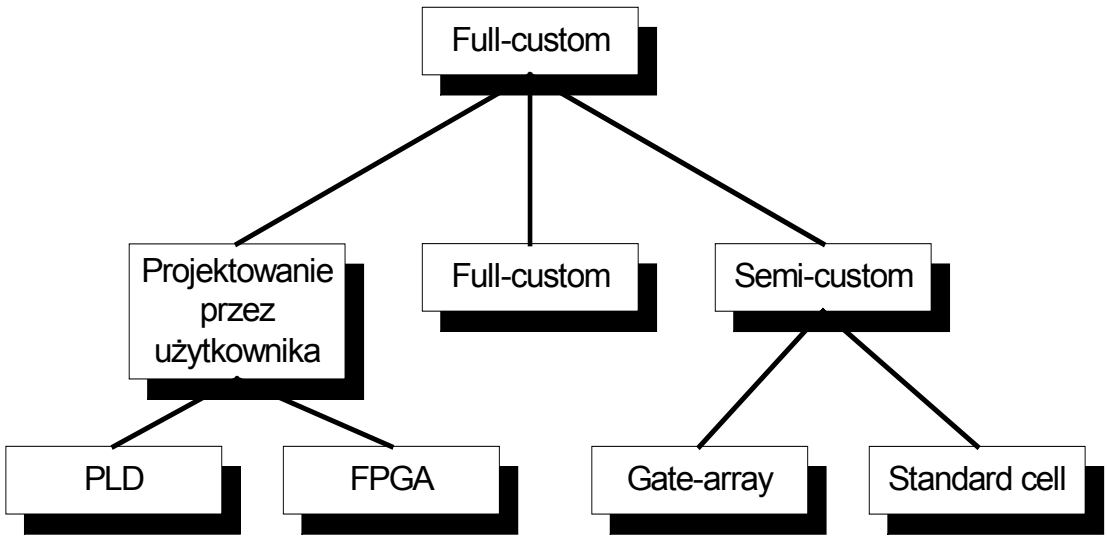
2.1.1 Systematyka układów ASIC.

Układy specjalizowane ASIC, układy produkowane na zamówienie klienta dzielą się na następujące grupy:

- Układy na zamówienie z pełnym cyklem projektowania (ang. *full-custom*) – Proces projektowania tego typu układów polega na przedstawieniu założeń projektowych przez zamawiającego układ. Fizyczna realizacja jest dokonywana przez wyspecjalizowane przedsiębiorstwa przy wykorzystaniu drogich i zaawansowanych narzędzi. Względy ekonomiczne skłaniają producentów tego typu układów do produkcji wielko seryjnej.
- Układy z ograniczonym cyklem projektowania (ang *semi-custom*). Wytwarzanie układu realizowane jest w dwóch etapach. Najpierw za pomocą tanich i łatwo dostępnych narzędzi projektujący tworzy układ. Ma on do dyspozycji gotowe komponenty, takie jak przerzutniki, bramki i multipleksery, z których wykonuje całościowy projekt. Następnie przesyła go dla wyspecjalizowanego przedsiębiorstwa, które wykonuje jego fizyczny odpowiednik. Układy tego typu, ze względu na wewnętrzną strukturę dzielimy na układy: wykorzystujące komórki standardowe (ang. *standard Cells*), układy wykorzystujące matryce bramek (ang. *gate arrays*)
- Układy projektowane przez użytkownika. Producent dostarcza dla klienta możliwość konfigurowania „prefabrykatów". Cechą charakterystyczną tych układów jest to, że są najwolniejsze i najmniej pojemne w porównaniu z wyżej wymienionymi rodzinami. Istnieje

bogate oprogramowanie, które wspiera proces wytwarzania tego typu układów scalonych. Układy te ze względu na wewnętrzną strukturę dzieli się na programowalne układy logiczne PLD (ang. *Programmable Logic Devices*) oraz układy FPGA (ang. *Field Programmable Gate Array*).

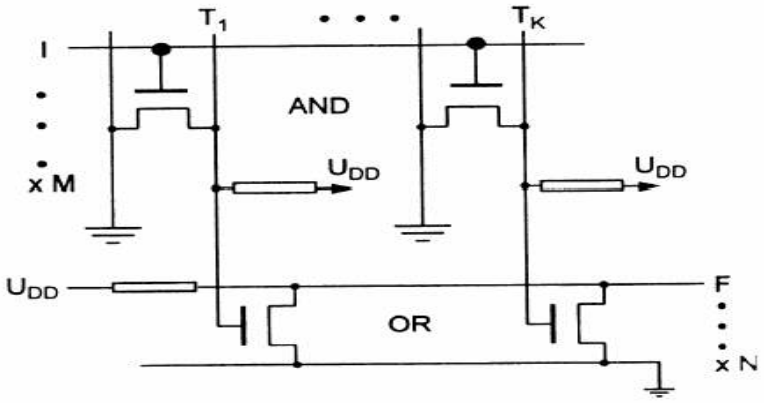
Podział układów ASIC obrazuje rysunek:



Rys.2.1.: Podział układów ASIC.

2.1.2 Układy FPGA i PLD.

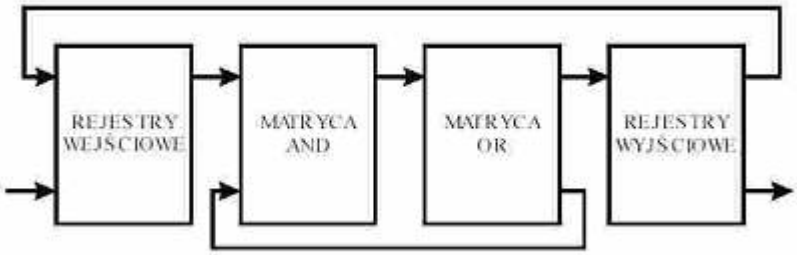
Podstawowym elementem konstrukcyjnym programowalnych modułów logicznych jest matryca tranzystorów, które w zależności od technologii – mogą być bipolarne TTL lub typu MOS. Układy typu PLD tranzystory te ułożone są w formie dwóch matryc: AND i OR.



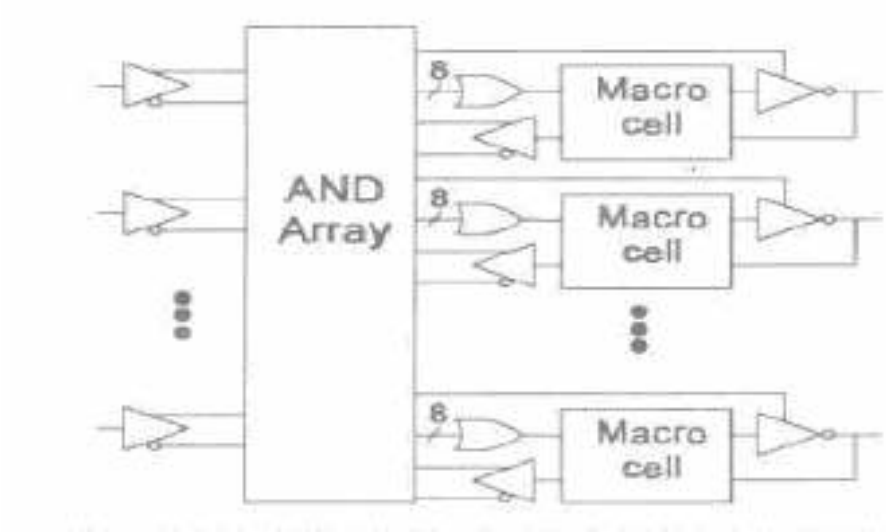
Rys.2.2: Matryca tranzystorów.

Elektrody tranzystorów są dołączone: bramka do poziomych linii wejściowych, Źródło do linii wyjściowych, a także do poziomych lub pionowych linii doprowadzających napięcie zasilania. Połączenia odpowiednich linii z bramkami tranzystorów (tzw. Punkty programowania), są zrealizowane w postaci cienkich tytanowo-wolframowych bezpieczników. Umożliwiają one programowanie matryc prosta metodą przepalenia. W układach tych występują także zespoły elementów wyjściowych: przerzutniki i trójstanowe bufory wyjściowe lub w bardziej rozbudowanych układach multipleksery i bramki OR, zwane makrokomórkami (ang. macrocell).

Najprostszymi układami typu PLD są tzw. SPLD (ang. *simple programmable logic devices*). Są najmniej pojemne ze wszystkich układów tego typu. Ich podział opiera się o możliwości konfiguracyjne matryc AND i OR.



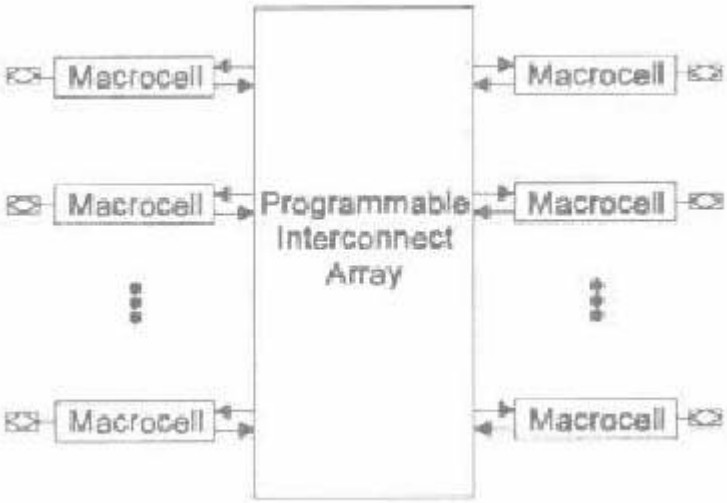
Rys.2.4: Schemat blokowy prostego układu PLD (SPLD).



Rys.2.5: Struktura prostego układu PLD (SPLD).

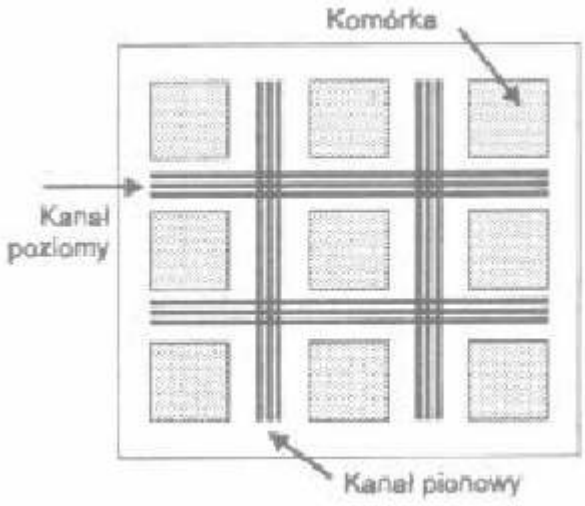
.

Complex PLD, w skrócie CPLD to bardziej złożone układy PLD. Podstawowym elementem jest tutaj programowalna matryca połączeń PIA (ang. *programmable inteconnect array*), do której dołączone są makrokomórki . Strukturę tego rozwiązania przedstawia rysunek poniżej.



Rys.2.6: Struktura złożonego PLD (CPLD).

Układy FPGA (ang. *Field Programmable Gate Array*) należą obok układów PLD (ang. *Programmable Logic Devices*) i CPLD (ang. *Complex PLD*) do grupy układów programowalnych przez użytkownika. Cechą charakterystyczną układów FPGA jest ich regularna struktura wewnętrzna w postaci prostokątnej matrycy Programowalnych Bloków Logicznych oraz również programowalnych połączeń między nimi.



Rys.2.7: Struktura układów programowalnych typu FPGA.

Programowanie układów FPGA polega na właściwym dla danego projektu skonfigurowaniu układów wejścia/wyjścia, bloków logicznych oraz połączeń między nimi.

2.1.3 Układy programowalne ALTERY.

Firma ALTERA rozwija i produkuje obecnie kilka rodzin układów: CLASSIC, MAX 3000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000, FLEX 10K, APEX 20K oraz rodziny Stratix, Mercury, Excalibur, Cyclone. Mają one różne właściwości: strukturę wewnętrzną, pojemność, szybkość działania, ale wszystkie łączy technologia wykonania CMOS, która powoduje przede wszystkim zmniejszony pobór mocy.

Systematyka układów programowalnych określa tego typu układy jako w pełni reprogramowalne, za pomocą sygnałów elektrycznych. Są to tzw. EPLD (ang. *eraseble PLD*). Podrozdział będzie przybliżał rozwiązania konstrukcyjne oraz charakteryzował wymienione powyżej rodziny.

- CLASSIC – rodzina układów programowalnych, która wykonana jest całkowicie w technologii CMOS. Pojedyncza macierz komórek, połączonych globalną szyną danych tworzy architekturę tego typu układów. Konfiguracja wewnętrznych elementów odbywa się poprzez stosowanie nie ulotnej pamięci EPROM, która je tworzy.

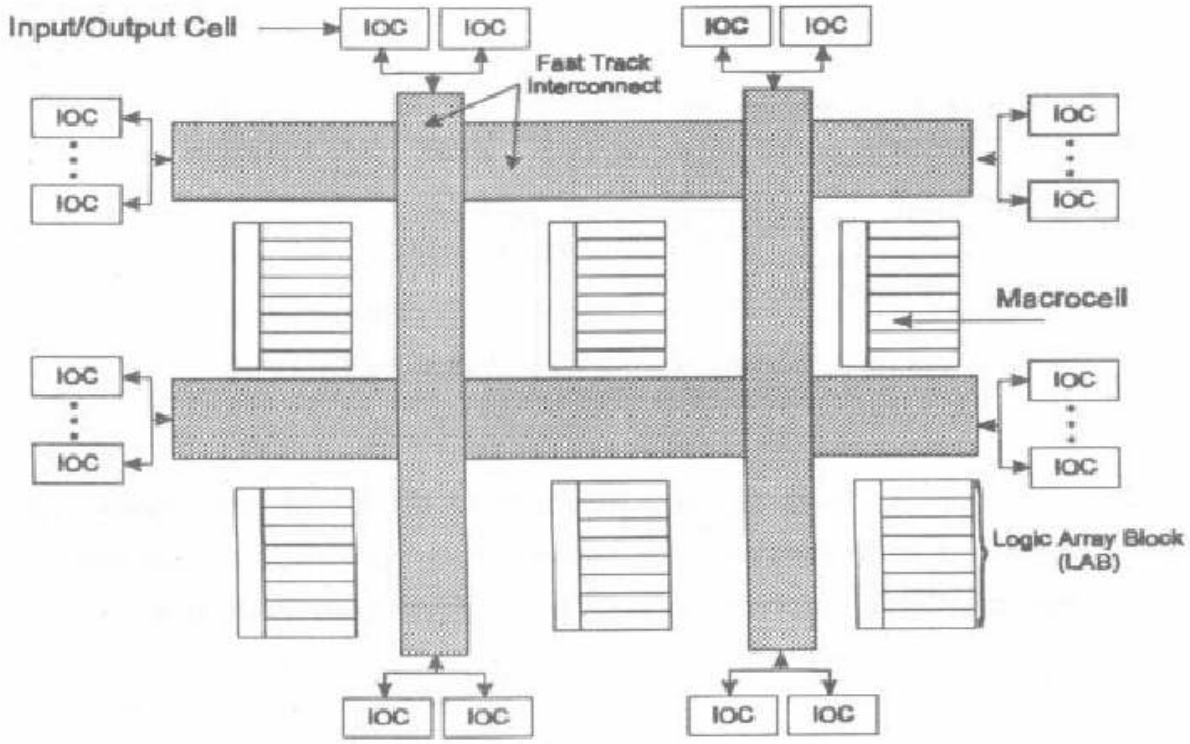
- MAX 3000, MAX 7000 – układy produkowane w technologii CMOS E^2PROM o wysokiej gęstości, bazują na drugiej generacji architektury MAX. Ich pojemność wynosi od 32 do 256 makrokomórek logicznych, które są łączone w bloki po szesnaście. Bloki te nazywa są LABami (ang. *Logic Array Block*).

  Komunikacja między modułami LAB odbywa się dzięki programowalnej tablicy połączeń PIA. Ta ogólnodostępna szyna jest programowalną ścieżką, która pozwala na połączenie każdego sygnału źródłowego z każdym sygnałem docelowym w układzie. W rodzinie MAX 7000 wszystkie dedykowane wejścia, piny we/wy i wyjścia z makrokomórek są podłączone do PIA i dzięki temu są dostępne w całym układzie.

  Rodzina MAX 7000 dostarcza programowalną optymalizację prędkości i mocy (krytyczne pod względem prędkości części projektu mogą pracować przy dużej szybkości i pełnej mocy, podczas gdy pozostałe części mogą pracować przy zredukowanej prędkości i niskiej mocy). Ta cecha pozwala projektantowi konfigurować jedną lub więcej makrokomórek tak , by pracowały one przy 50% lub niższym zasilaniu . Odbywa się to

tylko kosztem dodania nominalnego opóźnienia czasowego . Układy serii MAX 7000 mogą być reprogramowane około 100 razy.

- MAX 9000 - Rodzina ta złożona z układów CMOS o wysokiej gęstości bazuje na trzeciej generacji architektury MAX stworzonej przez Altera Corp. Układy tego typu zbudowane są z makrokomórek (od 320 do 560) rozdzielonych w grupy po 16 makrokomórek każda. Każda makrokomórka zawiera tablicę „programowalna AND\stała OR". W architekturze MAX 9000 połączenia pomiędzy makrokomórkami a pinami WE/WY są realizowane za pomocą połączeń FastTrack - rzędami i kolumnami specjalnych kanałów, które biegną przez cały układ. Ta rozległa struktura połączeń zapewnia określoną wydajność nawet przy skomplikowanych projektach.

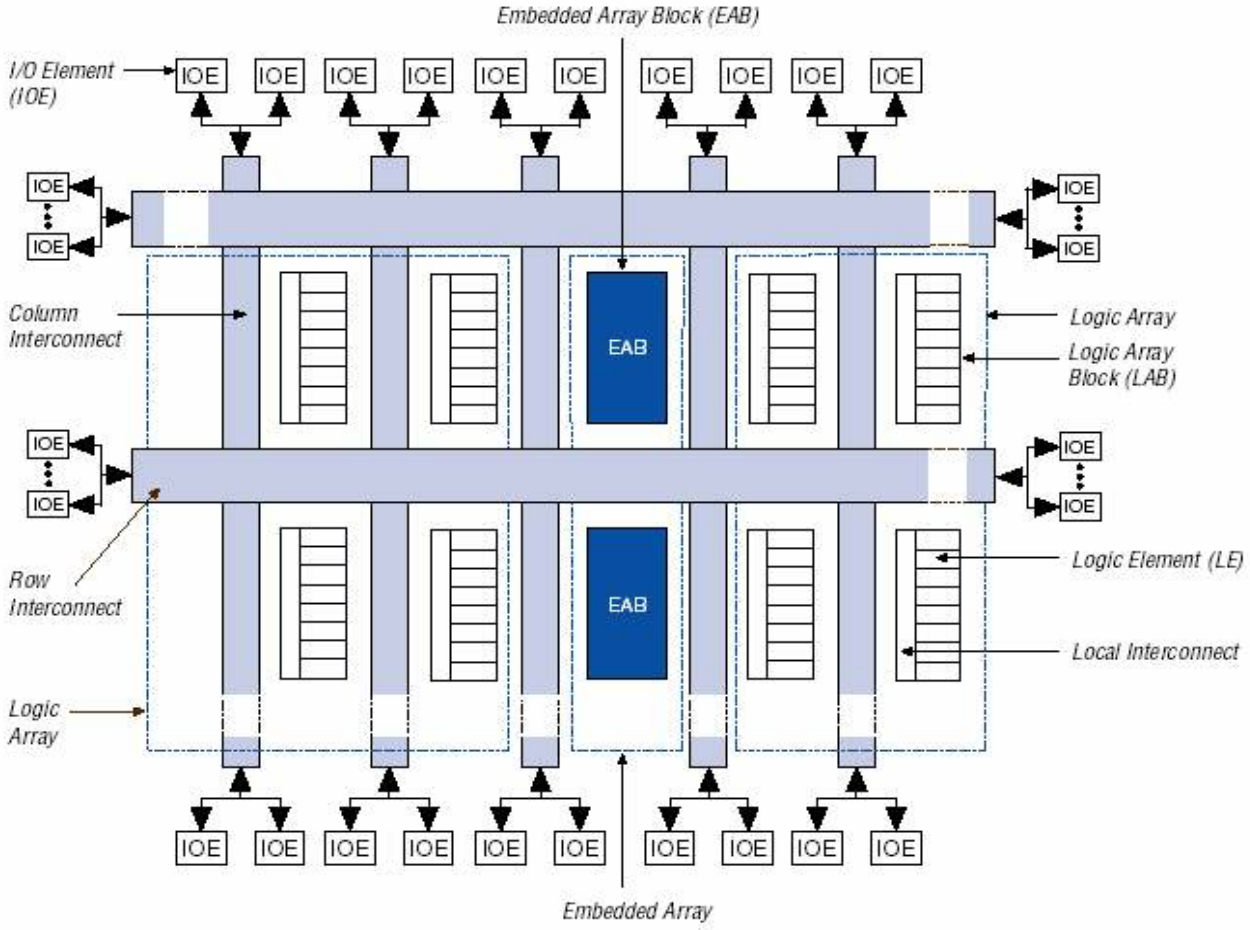


Rys.2.8: Struktura wewnętrzna układów z rodziny MAX9000.

- FLEX 6000, FLEX 8000 - Rodzina FLEX łączy w sobie zalety zarówno układów EPLD jak i FPGA. Duża liczba rejestrów charakterystycznych dla układów FPGA wraz z bardzo szybkimi połączeniami o przewidywalnych opóźnieniach charakterystycznymi dla układów EPLD powoduje, że układy FLEX 8000 nadają się do szerokiego zakresu aplikacji. Dane konfiguracyjne mogą być przechowywane w standardowych układach EPROM lub ładowane z pamięci RAM.

Układy FLEX 8000 mogą być przekonfigurowane po zresetowaniu układu i załadowaniu nowych danych. Ponieważ rekonfiguracja trwa mniej niż 100 ms, zmiany mogą być dokonywane w czasie rzeczywistym podczas normalnej pracy systemu.

- FLEX 10K – chyba najwięcej ciekawych zmian pojawiło się w układach rodziny FLEX 10K, a co najważniejsze zmiany te spowodowały wzrost szybkości działania układów realizowanych przez użytkownika oraz zwiększenie pojemności układów fizycznych. Tworząc układ, projektujący wykorzystujący w najbardziej wyrafinowany z możliwych sposobów możliwości FLEX 10K traci jednak możliwość łatwej przenoszalności projektu na układy innych firm (np. Xilinx). Wyjaśnię tą kwestie w dalszej części. Po pierwsze zmianie uległa struktura LAB i makrokomórki. Zmiana ta polega na tym, że matryca wejściowa jest wspólna dla wszystkich makrokomórek w danym bloku LAB.



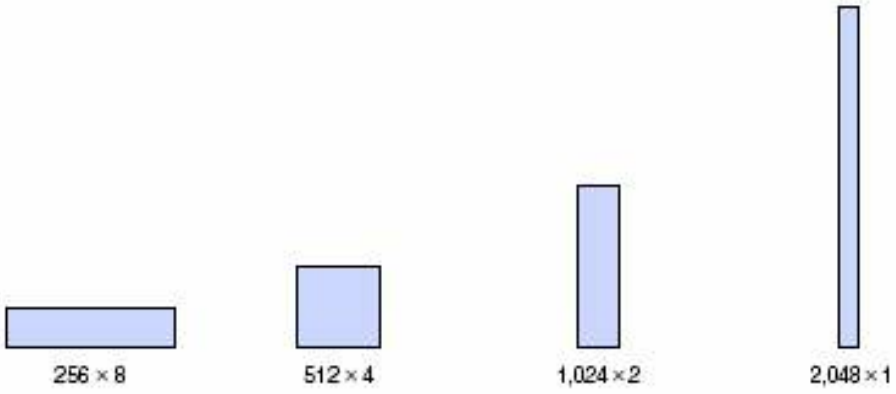
Rys.2.9: Struktura wewnętrzna układów FLEX 10K

Najważniejszą jednak cechą jest to, że układy te posiadają wbudowaną logikę specjalnego przeznaczenia, dzięki której możliwe jest implementowanie funkcji pamięci ROM i RAM i dzięki temu zwiększyła się znacząco pojemność układu.

Wewnętrzna struktura układów FLEX 10K zbudowana jest z bloków matryc logicznych, bloku matryc wbudowanych oraz połączeń typu „fast track" (obrazuje to rysunek powyżej). Wbudowana matryca EAB (ang. *embedded array block*) umożliwia emulację pamięci stałej oraz specjalizowanych funkcji logicznych w różnej konfiguracji wejściowo – wyjściowej. Podczas procesu kompilacji układu zadeklarowana funkcja jest wbudowywana w tą szybką pamięć. Układy innych firm nie stosują tej techniki wobec czego utrudnione jest przenoszenie projektów tego typu na układy spoza oferty ALTERY.

Każdy EAB jest oddzielnym blokiem logicznym, który może pracować niezależnie lub może tworzyć większa strukturę. Połączenia „fast track" umożliwiają podłączenie wejścia i wyjścia EAB do zewnętrznych pinów układu fizycznego, do innego EAB lub do bloku logicznego LAB. Zastosowanie EABów:

- Emulacja funkcji pamięci o pojemności 2kb każdy EAB. Możliwe jest wybranie konfiguracji pamięci jako 256 x 8 bitów, 512 x 4 bity, 1024 x 2 bity lub 2048 pojedynczych bitów.

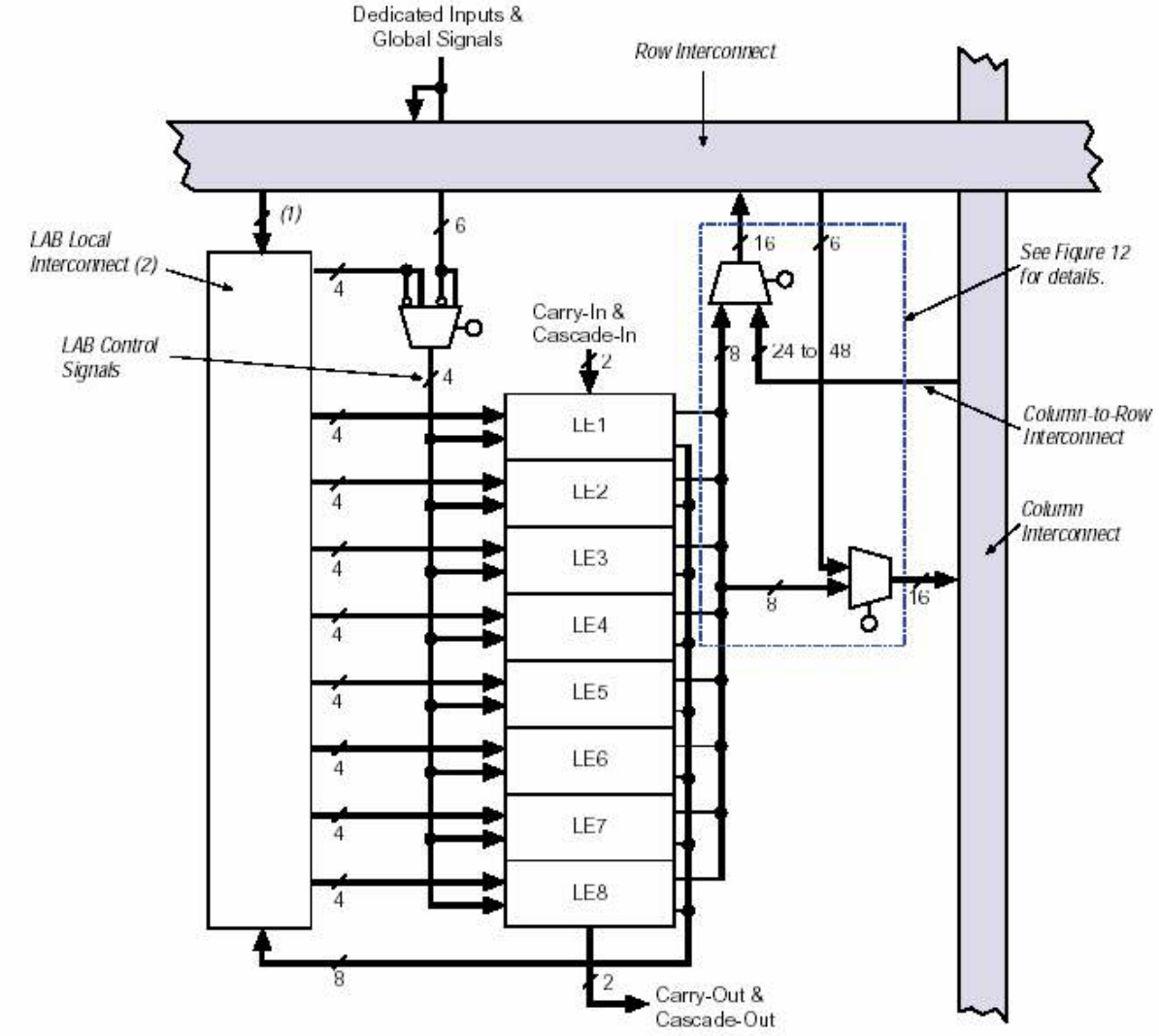


Rys.2.10: Możliwe konfiguracje wybudowanego bloku EAB.

- Implementacja funkcji pamięci RAM. Każdy EAB posiada port adresowy, danych wejściowych, wyjściowych oraz port sterujący, które są przyłączone do kolumn i wierszy szybkich połączeń.
- Implementacja funkcji pamieci ROM, która różni się od konfiguracji pamięci RAM tylko tym, że posiada właściwość „read only memory" – wejście danych jest zablokowane.

- Realizacja specjalizowanych, być może także krytycznych dla szybkości i pojemności projektu, funkcji logicznych.

Podstawowym elementem układów programowalnych pomimo wszystko nadal są komórki logiczne, które tworzą bloki logiczne (LAB). Każdy blok zawiera po 8 elementów logicznych (pokazuje to rysunek poniżej) i związane z nim połączenie lokalne. Każdy element logiczny LE (ang. *logic element*) zbudowany jest z komórki LUT (ang. *lookup table*), programowalnego przerzutnika oraz ścieżek sygnałowych. Wyjścia poszczególnych matryc EAB i LAB połączone są w system szybkich, wewnętrznych magistral, które przebiegają przez cały układ.

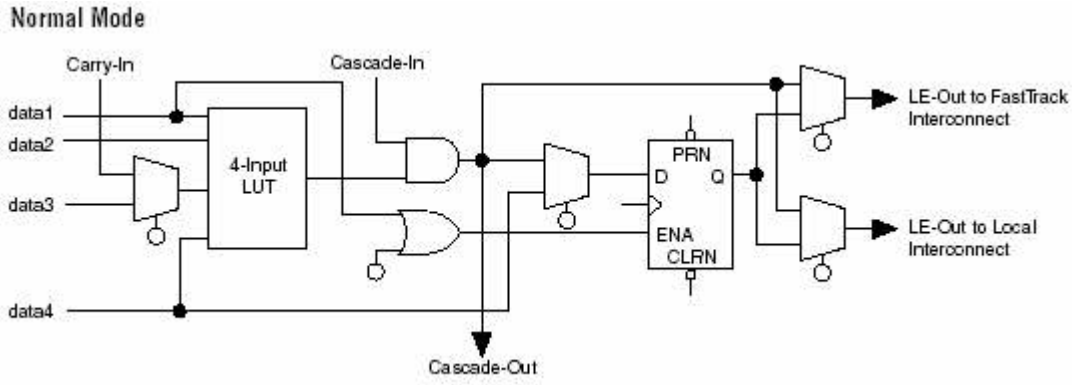


Rys.2.11 Struktura Bloku komórek logicznych (LAB).

Każdy logiczny element LE posiada przerzutnik, który może być zdefiniowany jako przerzutnik typu D, T, JK lub RS.

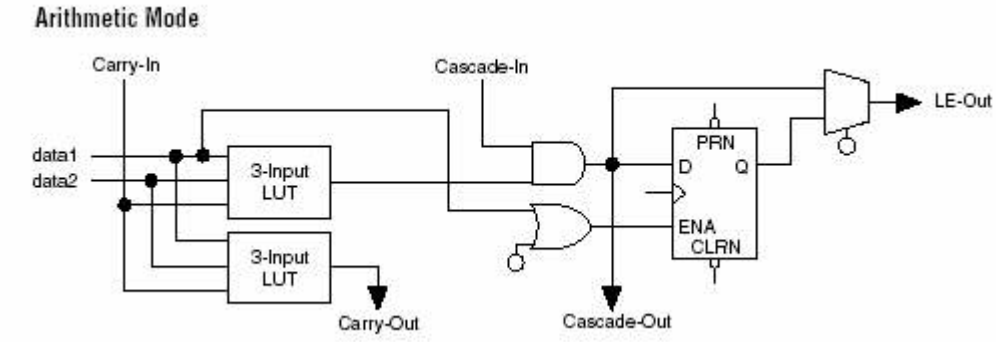
Komórki logiczne można konfigurować w trzech trybach:

- trybie normalnym (ang. *normal mode*),



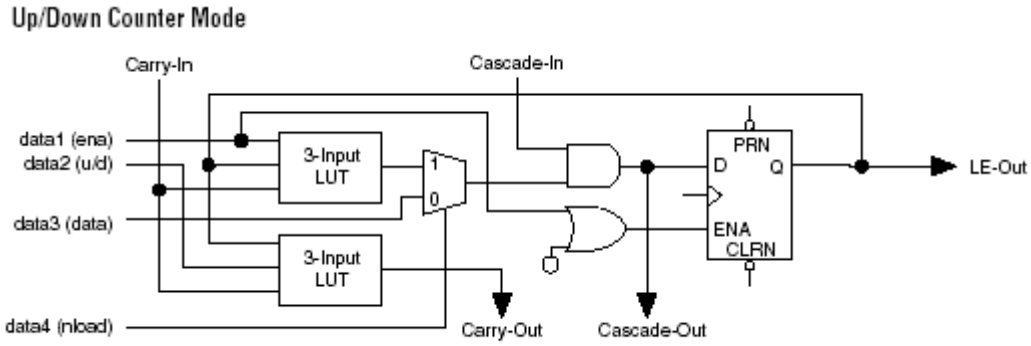
Rys.2.12: Tryb normalny pracy komórki logicznej.

- trybie arytmetycznym (ang. *arithmetical mode*),



Rys.2.13: Tryb arytmetyczny pracy komórki logicznej.

- trybie licznika (ang. *counter mode*),



Rys.2.14: Tryb licznikowy pracy komórki logicznej.

- Poza systemem projektowym dostępne są także rodziny układów: APEX 20k, Stratix, Mercury, Excalibur, Cyclone (Obsługuje je oprogramowanie Quartus).

Do praktycznej realizacji projektu wykorzystam układy FLEX 10K oraz ich odmianę FLEX 10KE. Za cenę szybkości zrezygnuję z możliwości przenaszalności projektu na inne układy, spoza oferty ALTERY. Jednakże sprawdzę także takie projekty, które będą miały uniwersalne cechy i sprawdzę jak wypadają w porównaniu z dedykowanymi dla układów ALTERY.

2.2 System projektowy Altery: Max PLUS II.

System Max PLUS II jest to pakiet 11 programów użytkowych, które stanowią system projektowy. Zarządzać można ich uruchomieniem, bądź zatrzymaniem za pomocą bardzo dobrze zorganizowanego managera. Z pozycji głównego menu można uruchamiać każdy z 11 programów tworzący stanowisko projektowe Altery go:

- Prezenter hierarchii (ang. *hierarchy display*) najbardziej przydatny przy złożonych projektach o skomplikowanej, wielopoziomowej strukturze. Służy do prezentacji struktury projektów hierarchicznych, umożliwia szybki dostęp do plików na wszystkich poziomach hierarchii.

- Edytor graficzny (ang. *graphic editor*) – umożliwia projektowanie układów logicznych na poziomie schematu. Z tego poziomu dostępne są prymitywy i makrofunkcje realizujące podstawowe i rozszerzone funkcje logiczne. Można również zbudować własną bibliotekę podukładów. Program generuje pliki Graphic Design File (.gdf) lub OrCAD Design File (.sch), które zawierają opis układu.

- Edytor symboli (ang. *symbol editor*) Program służący do przeglądu, tworzenia oraz edytowania symboli, reprezentujących układy logiczne.

- Edytor tekstowy (ang. *text editor*) ma podobne możliwości jak zwykły edytor tekstowy Windows. Dodatkowe wyposażenie to zbiór standardowych struktur języka AHDL. Umożliwia on redagowanie tekstowego projektu AHDL. Edytor umożliwia obsługę dowolnego pliku w formacie ASCII. Na potrzeby działania w systemie projektowym wykorzystuje się pliki Text Design File (.tdf) - zawierające opis układu w języku AHDL.

- Edytor przebiegów czasowych (ang. *waveform editor*) – narzędzie ma dwa rodzaje zastosowań: pozwala na projektowanie układu przez podanie przebiegów wejściowych, wyjściowych oraz zmian stanów automatu. Umożliwia też stworzenie pliku wejściowego do symulatora – dzięki temu można wprowadzić wektory testowe dla danego układu. Program generuje Waveform Design File (WDF - rozszerzenie .wdf) zawierający opis układu za pomocą przebiegów lub Simulator Channel File (SCF - rozszerzenie .scf) z opisem przebiegów dla symulatora.

- Edytor planu zasobów (ang. *floorplan editor*) służy do wprowadzania i modyfikacji planu zasobów fizycznego układu, a także do oglądania wyników syntezy logicznej.

- Kompilator (ang. *compiler*). Kompilator MAX+PLUS II składa się z serii modułów i aplikacji, które sprawdzają projekt pod względem występowania w nim błędów, przeprowadzają syntezę logiczną, przydzielają projekt do jednego lub więcej układów Altery i generują pliki wejściowe do symulacji, analizy czasowej i programatorów. Kompilacja projektu jest automatyczna, przy czym użytkownik może wpływać na jej przebieg przy pomocy odpowiednich opcji. Służą do tego komendy kompilatora oraz polecenia menu Assign.

- Symulator (ang. *simulator*). Program pozwala na przetestowanie funkcjonalnej poprawności projektu oraz kontrolę przebiegów czasowych w wybranych węzłach w skompilowanym już projekcie. Umożliwia porównanie wyników symulacji ze spodziewanymi (przez projektanta) oraz (przy pomocy odpowiednich urządzeń) z wyjściami zaprogramowanych już urządzeń. Symulator można uruchomić w dwóch trybach: interakcyjnym lub wsadowym. W trybie interakcyjnym użytkownik sam ustawia wszystkie opcje i parametry symulacji, w trybie wsadowym działaniem programu kierują instrukcje zawarte w pliku <nazwa projektu>.cmd (Command File).
Program obsługuje trzy rodzaje plików pliki typu simulator netlist file w skrócie (.snf) - jest to plik tworzony przez kompilator, jest automatycznie ładowany w momencie uruchomienia Symulatora. Drugi rodzaj to pliki zawierające wektory testowe. Noszą one nazwę Simulator Channel File (.scf) jak również ASCII Vector File (.vec). Ostatnimi są pliki z rozszerzeniem .cmd (ang. *Command File*) - zawierające instrukcje dla symulatora, działającego w trybie wsadowym.

- Analizator czasowy (ang. *timing analyzer*) Program pozwala sprawdzić parametry zaprojektowanego układu (projektu) po wykonaniu kompilacji (syntezy). W szczególności pozwala prześledzić opóźnienia sygnału na ścieżce między dowolnym wejściem i wyjściem oraz znaleźć ścieżkę krytyczną. Pliki wejściowe to Simulator Netlist File (.snf). Pozwala wykonać trzy typy analizy Delay Matrix (analiza czasów opóźnień pomiędzy dowolnymi wejściami i wyjściami układu.), Registered Preformance (analiza układów synchronicznych, pozwalająca na wyznaczenie maksymalnej częstotliwości z jaką układ może pracować), Setup/Hold Matrix (oblicza minimalne okresy czasu o jakie sygnały na wejściu układu muszą wyprzedzać aktywne zbocze zegara oraz jak długo muszą pozostać niezmienione (po zmianie stanu zegara) aby ich wartość została wpisana do przerzutnika).

- Programator (ang. *programmer*) Program pozwalający programować, testować i badać układy z rodzin które wspiera oprogramowanie Max PLUS II .

- Procesor komunikatów (ang. message processor) - wyświetla listę błędów, ostrzeżeń oraz informacje generowane przez pracujące (także w tle) aplikacje systemu. Dane te przechowywane są do momentu, gdy ponownie uruchomisz program , który je przesłał do procesora komunikatów. Program ten współpracuje z wszystkimi programami wchodzącymi w skład systemu projektowego Max PLUS II.

Na koniec trzeba też wspomnieć o systemie pomocy, która stanowi źródło kompletnej oraz aktualnej wiedzy potrzebnej dla projektanta. Dostarcza niezbędnej wiedzy na temat dostępnych układów fizycznych oraz na temat narzędzi wchodzący w skład systemu. Dostępna jest z poziomu każdej z aplikacji.

2.3 Język AHDL.

AHDL (Altera Hardware Description Language) - jest strukturalnym językiem wysokiego poziomu stosowanym w środowisku Altery do opisu układów cyfrowych. Części projektu opisane w tym języku (pliki <nazwa>.tdf AHDL Text Design Files) mogą być łączone z innymi w większe projekty hierarchiczne.

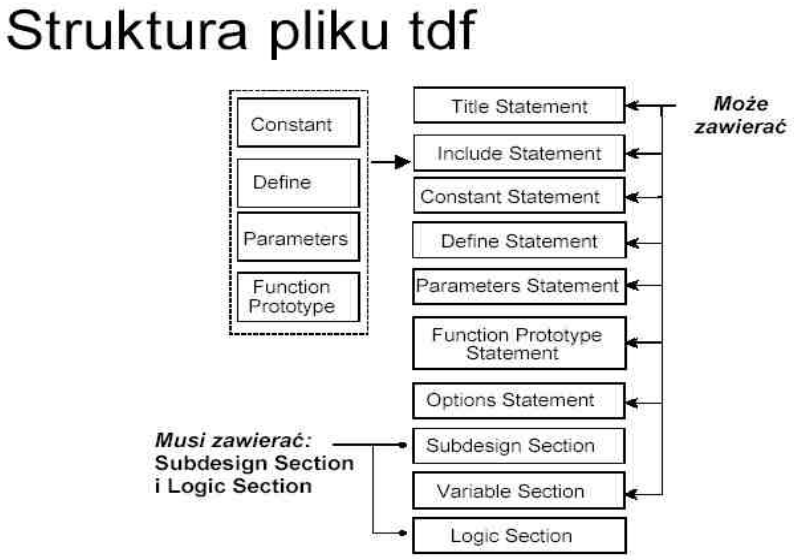
Jest on językiem wysokiego poziomu całkowicie zintegrowanym z systemem MAX+PLUS II. Został stworzony do specyfikowania projektów realizowanych w układach programowalnych firmy ALTERA.

Jego składnia jest bardzo łatwa i intuicyjna oraz pozwala szybko realizować tablice prawdy, układy sekwencyjne, kombinacyjne oraz inne operacje logiczne.

Program w AHDL może być pisany w jakimkolwiek pliku tekstowym, lecz musi posiadać rozszerzenie .tdf (ang. text design file). Można dokonać jego kompilacji, a następnie symulacji.

Bardzo ważnym atutem jest możliwość konstruowania projektów hierarchicznych. Jest bardzo proste zadanie i nawet nie doświadczony projektant szybko opanuje technikę konstrukcji hierarchicznych. Edytor symboli automatyczne generuje symbol, który reprezentuje plik z rozszerzeniem .sym. Reprezentuje go prostokąt z zaznaczonymi wejściami i wyjściami układu logicznego. Można każdy wygenerowany symbol włączyć do pliku .gdf. W ten sposób można połączyć wiele plików .sym w większą całość, realizującą bardziej złożoną funkcję. W taki sam sposób można włączać do zbiorów .gdf makrofunkcje zdefiniowanych przez Alterę.
Wszystkie makrofunkcje zawarte są w plikach o rozszerzeniu .inc w bibliotece makrofunkcji. Plik tekstowy ASCII .inc (Include File) może być dołączony do pliku .tdf poprzez instrukcję Include.

2.3.1 Struktura podstawowego pliku w AHDL.



Rys.2.15: Struktura pliku .tdf.

Omówię teraz najważniejszymi elementy języka AHDL. Przedstawiać będę je w kolejności nie przypadkowej. Determinować ją będzie kolejność w jakiej powinny występować w każdym zbiorze TDF. Niektóre z nich są obligatoryjne, inne opcjonalne.

- instrukcja **TITLE** (opcjonalnie) – pozwala na redagowanie komentarzy, które pojawią się w

zbiorze wynikowym kompilacji - Report File;
- instrukcja **INCLUDE** (opcjonalnie) – pozwala na realizacje złożonych projektów w postaci hierarchicznej. Zbiór włączany jest za pomocą Include File;
- instrukcja **CONSTANT** (opcjonalnie) – pozwala na specyfikowanie symbolicznych nazw, którymi są zastępowane stałe;
- instrukcja **DEFINE** (opcjonalnie) – daje możliwość zdefiniowania funkcji numerycznych, których obliczone wartości są argumentami innych funkcji;
- sekcja **SUBDESIGN** – pozwala na deklaracje wejść, wyjść i portów dwukierunkowych. Mogą być następujące typy portów: *input* – wejście, *output* – wyjście, *bidir* - port dwukierunkowy, *machine input, machine output,* które pozwalają na pobieranie i przesyłanie stanów urządzenia sekwencyjnego;
- sekcja zmiennych **VARIABLE** (opcjonalna) – deklarowanie zmiennych, które reprezentują i przechowują informację. Sekcja VARIABLE może zawierać także deklarację: *node*, *register* (np. DFF), *state machine*, *machine allias*;
- sekcja logiczna **LOGIC** – służy do określenia operacji logicznych.

Tego typu operacje definiuje się za pomocą równań boolowskich, warunków logicznych i tablic prawdy. Zawarte są w niej wszelkie kombinacje na wartościach sygnałów oraz sposoby ich przepływu. Zaczyna się słowem *begin* a kończy *end*. W sekcji tej występują operatory logiczne i arytmetyczne, instrukcje warunkowe IF, instrukcje wyboru CASE oraz tablice prawdy.

2.4 Proces specyfikacji układu w strukturach programowalnych.

Proces projektowania układów programowalnych obejmuje trzy etapy.

- Wprowadzenie specyfikacji projektu, najpierw formułowany jest opis działania projektowanego układu za pomocą schematu logicznego, opisu tekstowego lub wykresów czasowych. Do specyfikacji działania służy nam edytor graficzny (ang.Graphic Editor), dzięki któremu możemy konstruować schematy logiczne i blokowe. Pozwala on wykorzystywać do tego celu bibliotekę standardowych bloków funkcjonalnych TTL i makrofunkcji oraz stosować globalne oznaczenia sygnałów i linii połączeniowych. Nie mniejsze znaczenie ma edytor tekstowy (ang. Text Editor), który umożliwia przygotowanie formalnego opisu układu w odpowiednim języku specyfikacji sprzętu, np. AHDL (Altera Hardware Description Langguage), VHDL (Very High Speed Integrated Circuits Hardware Description Language). Innym sposobem opisu może też być opis zależności między sygnałami wejściowymi i

wyjściowymi. Tę formę specyfikacji umożliwia edytor wykresów czasowych (ang. Waveform Editor). Tworzenie specyfikacji układu ułatwia możliwość stosowania wielopoziomowej struktury hierarchicznej i dowolnego łączenia dostępnych środków opisu.

- Przetwarzanie wprowadzonego projektu, obejmuje weryfikację formalną projektu i przetworzenie danych (kompilację). Wynikiem tej operacji jest szczegółowa konstrukcja funkcjonalna i rozmieszczenie jej elementów w wybranej strukturze układu programowalnego. Proces kompilacji projektu polega na optymalizacji jego struktury logicznej i dekompozycji na mniejsze części tak, aby mogły one być rozmieszczone i odpowiednio połączone ze sobą w zadanej strukturze programowalnej. Podczas kompilacji projektu tworzone są dane dotyczące modelu fizycznego funkcjonowania układu programowalnego i mapy punktów połączeń niezbędnych do symulacji układu.

- Weryfikowanie i programowanie opracowanego modelu struktury.
  Na koniec jest weryfikacja i programowanie. Ma ona na celu sprawdzenie czy poprawnie została opracowana konstrukcja pod względem funkcjonalnym i zachowania się w czasie. Polega to na przeprowadzeniu eksperymentów symulacyjnych, które są formułowane w języku przebiegów czasowych (Waveform Editor). W trakcie działania symulacji działania rzeczywistego układu, informacje podane na wejście układu zostają przetworzone na informacje pojawiające się na wyjściu układu. W końcu działania układu będziemy mogli zweryfikować za pomocą wykresów czasowych. Końcowym etapem jest realizacja projektu w układzie programowalnym w procesie programowania.

III. Algorytm HIEROCRYPT.

3.1 Wprowadzenie.

Hierocrypt jest szyfrem blokowym, którego autorami są kryptolodzy TOSHIBA Corp. W 2000 roku algorytm został zgłoszony na konkurs NESSIE, który ma wyłonić nowe standardy kryptologiczne dla Europy. Choć wiadome już jest, że Hierocrypt nim nie zostanie, to jednak posiada on bardzo dobre cechy: ma udowadnialne bezpieczeństwo przed większością znanych ataków, jest zgodny z nowymi trendami w projektowaniu szyfrów blokowych, to jeden z szyfrów, w którym zastosowana jest strategia „szerokiej ścieżki" (ang. Wide Trail Strategy), osiąga bardzo dobre wyniki w implementacjach programowych.

Został odrzucony ponieważ nie był odporny na atak typu SQUARE oraz atak z wykorzystaniem różniczek niemożliwych. Także nie jasnym był algorytm generacji pokluczy do poszczególnym rund oraz osiągał jedne z najsłabszych wyników w implementacji sprzętowej i w efekcie już po pierwszej fazie konkursu Hierocrypt został odrzucony.

Algorytm, którego protoplastą był szyfr SHARK, jest iteracyjny, składa się z różnej ilości rund, których ilość jest zależna od długości klucza. Dla wersji Hierocrypt-3 ta zależność przedstawia się następująco: klucz 128 bitowy - 6 rundy, 192 bitowy – 7 rund, 256 bitowy – 8 rund. Podyktowane jest to potrzebą zwiększenia marginesu bezpieczeństwa, wartości prawdopodobieństw różnicowych i liniowych najlepszych charakterystyk w stosunku do ataku polegającego na przeszukaniu pełnej przestrzeni klucza.

W pracy będę się zajmował się będę wersją algorytmu, o długości bloku 128 bitów, oraz z długością klucza 128 i 256 bitów i ta też wersję dalej będę nazywał Hierocrypt.

3.2 Podstawy matematyczne.

Operacje w Hierocrypcie zdefiniowane są na poziomie bajtów oraz słów cztero, ośmio i szesnasto bajtowych. Bajty reprezentują elementy ciała $GF(2^8)$. Dla dobrego zrozumienia działania szyfru, a także dla zrozumienia założeń projektowych przyjętych przez autorów szyfru potrzeba wyjaśnić kilka matematycznych oraz wynikających z teorii kodów koncepcji.

3.2.1 Wprowadzenie do problematyki działań w ciele $GF(2^n)$.

Strukturą algebraiczną nazywamy wyróżnione działanie wewnętrzne w danym zbiorze F oraz ten zbiór.

Grupą nazywamy strukturę algebraiczną złożoną ze zbioru F i wewnętrznego działania, które spełnia warunki:

- działanie to spełnia prawo łączności,
- każdy element zbioru posiada element odwrotny,
- istnieje element neutralny wyróżnionego działania,

Jeżeli działanie spełnia prawo przemienności to taką grupę nazywamy przemienną albo abelową.

Ciałem nazywamy zbiór F z działaniami dodawania (+) oraz (*), które mają wyróżnione elementy nazywane neutralnymi, ze względu na ich właściwość. Są to odpowiednio: zero i jedynka. Własności jakie musi spełnić taka struktura, żeby można było ją określać mianem ciała:

- zbiór F z każdym z działań tworzy grupę abelową,
- oba działania spełniają prawo rozdzielności.

Ciałem skończonym nazywamy ciało, które posiada skończoną ilość elementów. Ciałem, którym będziemy się zajmować w przypadku omawiania algorytmu Hierocrypt jest $GF(2^n)$.

$GF(2^n) = ((a_1, a_2, a_3, \dots , a_{n-2}, a_{n-1}): ai = \{0, 1\}, 0 \le i \le n-1), +, *, 0, 1)$.

Operacje w Hierocrypcie są przeprowadzane w ciele $GF(2^n)$, gdzie n = 8.

Elementy z tego ciała zwykle przedstawia się jako:

- ciąg bitowy

np. $a = a_7\, a_6\, a_5\, a_4\, a_3\, a_2\, a_1\, a_0$ gdzie $a = \{0, 1\}$

- wielomian

np. $a_7 x^7 + a_6 x^6 + a_5 x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a\, x + a_0$

Reprezentacja wielomianowa jest wzajemnie jednoznaczna z elementami zapisanymi w postaci bitowej.

Struktura algebraiczna ciała $GF(2^8)$ ma określone działania:

Dodawanie – polega na binarnym dodawaniu bajtów (wielomianów) – jest to operacja EXOR oznaczana jako $\oplus$.

Np. Zapisane w postaci szesnastkowej bajty

$57 \oplus 83 = D4$

możemy przedstawić w postaci binarnej i również zsumować modulo dwa, czyli przeprowadzić operacje EXOR zdefiniowana dla tego ciała:

$01010111 \oplus 10000011 = 11010100$ (notacja binarna)

oraz w postaci wielomianowej. Operacje sumowania modularnego dokonujemy na współczynnikach wielomianów:

$(x^6 + x^4 + x^2 + x^1 + x^0) \oplus (x^7 + x^1 + x^0) = x^7 + x^6 + x^4 + x^2$ (notacja wielomianowa)

Mnożenie – polega na mnożeniu binarnym wielomianów modulo binarny wielomian nierozkładalny stopnia 8. Takim wybranym wielomianem nierozkładalnym dla algorytmu jest

$$x^8 + x^6 + x^5 + x + 1.$$

Np.:

$57 * 83 = C1$ (notacja szesnastkowa)

$(x^6 + x^4 + x^2 + x^1 + x^0) * (x^7 + x^1 + x^0) = (x^7 + x^6 + x^0)$

3.2.2 „Strategia szerokiej ścieżki”.

Hierocrypt jest algorytmem symetrycznym, który został zaprojektowany wg najnowszych trendów kryptograficznych. Projekt ten opiera się o strategię budowania szyfrów, która określana jest mianem „strategii szerokiej ściezki” (ang. *wide trail strategy*) Umożliwia ona projektowanie szyfrów blokowych o dużej i równomiernej odporności na kryptoanalizę liniową i różnicową. Została wprowadzona przez Joana Deamena w jego pracy doktorskiej z 1995 roku. Jako sposób konstruowania algorytmów kryptograficznych odpornych na kryptoanalizę różnicową i kryptoanalizę liniową.

Strategia Deamena budowania szyfrów blokowych zakłada, że przekształcenie rundy składa się z kilku przekształceń, które w jednakowy sposób traktują każdy bit. Niech x oznacza tekst wejściowy, k klucz (rundy) przekształcenia rundy $\rho[k](x)$. Wejście x jest dzielone na bloki p-bitowe x_i, $0 \le i < q$. Przekształceniami wchodzącymi w skład rundy szyfru zaprojektowanego tą metodą są:

1. nieliniowa warstwa podstawień γ operująca osobno na każdym z p-bitowych bloków;
2. liniowa warstwa dyfuzji θ mieszająca p-bitow bloki;
3. dodawanie klucza $\sigma[k]$.

Nieliniową warstwę podstawień stanowią równolegle działające S-skrzynki o wymiarach $p \times p$. Wybieramy je w taki sposób, że nie ma niezerowych różnic (xor) wejściowych, które przechodzą z dużym prawdopodobieństwem na określone różnice wyjściowe oraz korelacje wejściowo-

wyjściowe są małe. Chodzi tutaj o dużą równomierność rozkładu badanych własności w konstruowanych na potrzeby kryptoanalizy tablic xor-profili różnicowych i liniowych skrzynek podstawieniowych. Warstwę liniowej dyfuzji wybieramy tak, aby nie było żadnych charakterystyk różnicowych lub liniowych na małej liczbie aktywnych S-skrzynek. Strategia szerokiej ścieżki Deamena skłania projektantów szyfrów do tego, aby budować szyfry, których siła nie tylko opiera się na dobrze zaprojektowanej warstwie nieliniowej, ale także na odpowiednim dobraniu warstwy linowej.

Kolejną operacją jest dodawanie klucza, które specyfikuje jak klucze rund są mieszane z wejściem do każdej rundy.

Przekształcenie rundy dane jest więc następująco:

$$y = \rho[k](x) = \theta \circ \gamma \circ \sigma[k].$$

Przy czym kolejność tych przekształceń może się różnić w różnych projektach.
Użycie terminu „liniowy" implikuje wybór dotyczący operacji, które będziemy uważać za liniowe: xor, dodawanie modularne, dzielnie lub też inne zaproponowane przez projektanta.
Ponieważ xor jest najczęściej wybieraną operacją dla kryptoanalizy, więc będzie uznawany za operację liniową. Dodam, że w Hierocrypcie właśnie ta operacja jest operacją liniową.

Ten sposób budowania szyfrów blokowych zakłada, że każdy składnik rozpatrywany jest osobno. Warstwa dyfuzji wybierana jest tak aby miała jednolite i dobre własności rozpraszania, a S-skrzynki tak aby miały jednolite własności nieliniowe. Własności składników oceniane są bez brania pod uwagę szczegółów ich połączenia. Nie podejmuje się prób zrównoważenia słabości w warstwie nieliniowej dodatkowymi własnościami warstwy liniowej i odwrotnie. Całość staje się równomiernie silna dzięki temu, że wszystkie „ogniwa" są silne.

Strategia szerokiej ścieżki dała też podwaliny pod teorię szacowania odporności algorytmu na najważniejsze ataki kryptoanalityczne, czyli kryptoanalizę różnicową i liniową. Definiuje ona wielkości zwane liczbami różnicową i liniową, dzięki którym możliwe jest szacowanie maksymalnego prawdopodobieństwa charakterystyk kryptoanalitycznych badanego szyfru.

Jedynym miejscem, gdzie prawdopodobieństwo charakterystyki różnicowej jest mniejsze od 1 to aktywne S-skrzynki w warstwie nieliniowej. Jeśli δ jest maksymalnym prawdopodobieństwem przejścia niezerowej różnicy wejściowej na ustaloną różnicę wyjściową, a B jest ograniczeniem dolnym na liczbę aktywnych S-skrzynek w charakterystyce różnicowej, to prawdopodobieństwo jej jest ograniczone z góry przez:

$$p \leq \delta^B.$$

Jeśli S-skrzynki wybieramy, tak aby miały małą wartość δ, a liniową warstwę dyfuzji, tak aby wartość B była duża, to powyższe ograniczenie górne może mieć bardzo małą wartość.

Korelacja wejściowo-wyjściowa może być ograniczona w ten sam sposób. Dla warstw liniowych każdy wybór bitów wejściowych ma korelację równą 1 z dokładnie jednym wyborem bitów wyjściowych i korelację =0 dla wszystkich pozostałych wyborów. Jeśli maksymalną korelację wejściowo-wyjściową aktywnej S-skrzynki oznaczymy przez λ, a dolne ograniczenie na liczbę aktywnych S-skrzynek w relacji liniowej przez B to korelacja wejściowo-wyjściowa szyfru jest ograniczona z góry przez:

$$p \leq \lambda^B.$$

W ten sposób pojawiły się liczby B przekształceń różnicowego i liniowego, które mówią o maksymalnej możliwej liczbie aktywnych s-skrzynek danego, liniowego, bądź różnicowego przekształcenia.

W celu jak najlepszego uodpornienia szyfru blokowego na najbardziej znane ataki należy tak skonstruować warstwę dyfuzji szyfru, aby liczby B, różnicowa i liniowa miały jak największe wartości, bo dzięki temu uzyskamy najlepszy efekt lawinowości szyfru.

Najlepiej poznanym i zbadanym sposobem na uzyskanie dobrych wyników dyfuzyjnych jest zastosowanie rozwiązań dostarczonych przez teorię kodów MDS (ang. *maximum distance separable*).

3.2.3 Teoria kodów MDS.

Projektowanie warstw dyfuzyjnych dla szyfrów blokowych z najlepszymi właściwości propagującymi jest ściśle związane z teorią kodowania oraz z funkcjami boolowskimi.

Wagą hamminga binarnego ciągu nazywamy liczbę jedynek w tym ciągu. **Odległością hamminga** dwóch binarnych ciągów nazywamy liczbę ilości pozycji, na których te ciągi się różnią. W skrócie mówiąc teoria kodów dostarcza nam narzędzi, w postaci kodów MDS, do budowania warstw dyfuzyjnych, w których każde niezerowe słowo kodowe ma określoną maksymalną odległość hamminga od najbliższego mu słowa. Dzięki tej własności ciągów binarnych możliwe jest rozpropagowanie różnic wyjściowych pomiędzy dwoma wektorami binarnymi, które różnią się na wejściu do pewnej nieliniowej operacji na przykład na jednej pozycji, czyli ich odległość hamminga wynosi jeden.

Na potrzeby dalszego rozważania wyjaśnię teraz pojęcie **przestrzeni liniowej (wektorowej)**. Niech V będzie dowolnym niepustym zbiorem, a K ustalonym ciałem. Elementy zbioru V będziemy nazywać wektorami, a ciała K skalarami. Mówimy, że V jest przestrzenia liniową nad ciałem K, jeśli określone są funkcje:

$VxV \rightarrow V:(u, v) \rightarrow u+v.$

$KxV \rightarrow V(\lambda, v) \rightarrow \lambda v.$

Zwane odpowiednio dodawaniem wektorów i mnożeniem wektora przez skalar, takie, że spełnione są następujące warunki:

- V jest grupą abelową względem dodawania;
- $\lambda (u + v) = \lambda u + \lambda v$ ($\lambda \in K, u, v \in V$);
- $(\lambda + \mu)u = \lambda u + \mu u$ ($\mu, \lambda \in K, u \in V$);
- $\lambda (\mu u) = (\lambda \mu)u$ ($\mu, \lambda \in K, u \in V$);
- $1*v = v$ ($v \in V$, 1 jest jedynką ciała K);

Niepusty podzbiór W przestrzeni V nad ciałem K, będący przestrzenią liniową względem działań: dodawania wektorów w V oraz mnożenia wektorów z V przez skalar, nazywa się **podprzestrzenią liniową** przestrzeni V. Wynika z definicje następujący fakt niepusty zbiór W przestrzeni liniowej V, jest jej podprzestrzenią wtedy i tylko wtedy gdy W jest podgrupą V

Liniowym [n, k, d] kodem nad $GF(2^p)$ jest k wymiarowa podprzestrzeń przestrzeni wektorowej $(GF(2^p)^n$, w której każde dwa różne wektory mają odległość Hamminga równą, co najmniej d (d jest największą liczbą o tej własności).

Odległość d kodu liniowego równa jest minimalnej wadze każdego niezerowego słowa kodowego. Kod liniowy można opisać każdą z dwóch następujących macierzy: generującej G dla [n, k, d] kodu C lub też za pomocą macierzy kontroli parzystości H.

Bardzo ważnym faktem dotyczącym teorii kodów jest ograniczenie Singletona, które mówi, że jeśli C jest [n, k, d] kodem, to $d \leq n - k + 1$.

Kod, który spełnia to ograniczenie nosi nazwę kodu MDS (*Maximal Distance Separable*). Stosując ten fakt do n wymiarowego przekształcenia liniowego mamy, ograniczenie górne dla liczby B różnicowej i liniowej.

Przekształcenia, które osiągają tę granicę nazywane są optymalnymi (różnicowymi) przekształceniami dyfuzyjnymi. Dla algorytmu Hierocrypt taką macierzą generującą jest macierz występująca w przekształceniu mds_L. Zapewnia ona to, że po dwóch rundach aktywnych skrzynek podstawieniowych jest maksymalna ilość.

3.3 Założenia projektowe algorytmu.

Projekt Hierocrypt był docelowo tworzony na europejski standard symetrycznego blokowego szyfrowania i dlatego też najważniejszymi kryteriami są założenia zaproponowane przez organizatorów konkursu NESSIE:

- **Bezpieczeństwo** (ang. *security*)

Najbardziej fundamentalnym kryterium bezpieczeństwa szyfru blokowego jest długość klucza, która ma zapobiec efektywności i skuteczności ataku polegającego na pełnym przeszukaniu przestrzeni klucza (ang. *exhaustive attack or key search*). Oprócz długości klucz miarą bezpieczeństwa szyfru jest jego odporność na znane ataki kryptanalityczne: kryptanalizę różnicową (ang. *Differential cryptanalysis*) , kryptanalizę liniową (ang. *Linear cryptanalysis*), kryptanalizę różnicową wyższego rzędu (ang. *High-order differential cryptanalysis*), atak interpolacyjny (ang. *Interpolation attack*), atak na szyfry typu SQUARE (ang. *SQUARE-dedicated attack*), różniczki obcięte (ang. *Truncated differential attack*), różniczki niemożliwe (ang. *Impossible differential attack*).

- **Szybkość działania** (*ang. performance*)

Autorzy szyfru chcieli osiągnąć jak najlepszą szybkość implementacji programowych i sprzętowych – najważniejszym założeniem dotyczącym tego punktu, oprócz jak największej prędkości działania szyfru , był dobór operacji składowych części dotyczących przekształcania danych i generowania podkluczy do rund aby te dwie części algorytmu miały jak najbardziej zbliżony czas działania.

- **Efektywność implementacji** (ang. *implementation efficiency*)

Implementacja programowa – niewielka ilość kodu oraz małe wymagania pamięciowe – RAM.
Implementacja sprzętowa – niewielka ilość kodu oraz małe wymagania pamięciowe – ROM.

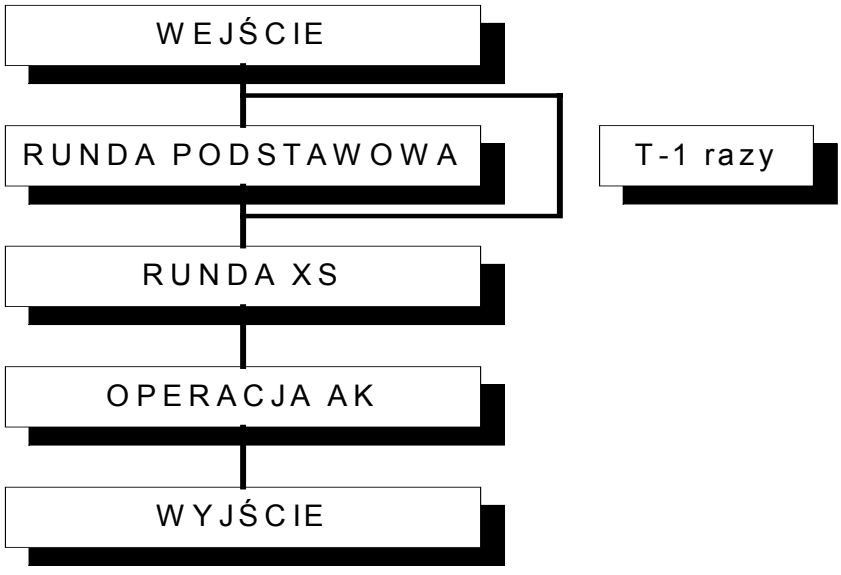
3.4 Specyfikacja operacji szyfrowania i deszyfrowania.

3.4.1 Operacja szyfrowania.

Hierocrypt–3 jest blokowym algorytmem iteracyjnym, w którym w zależności od dobranego klucza ustalona jest liczba rundy, które należy wykonać. T-rundowy algorytm składa się z T-1 operacji rundy ρ, jednej operacji XS oraz operacji dodania klucza AK (ang. *post-whitening*). Pełna specyfikacja każdej z operacji będzie przedstawiona w rozdziale 3.6.

$P_{(128)} = X(0)_{(128)}$,
$X(i)_{(128)}, = \rho(X(i-1)_{(128)}, K(i)_{(256)})$ $\quad i = 1,2, .. T-1$
$X(T)_{(128)}, = XS(X(T-1)_{(128)}, K(T)_{(256)})$
$C_{(128)}, = AK(X(T)_{(128)}, K(T+1)_{(128)})$
Gdzie T oznacza ilość rund (6,7 lub 8).

$X(i)_{(128)}$ jest wartością wyjściową po i-tej rundzie. Tekst wejściowy $P_{(128)}$ jest równy wartości $X(0)_{(128)}$. Natomiast $X(T-1)_{(128)}$ jest wartością wejściową operacji XS a $X(T)_{(128)}$ jest wartością wyjściową z tej operacji. Tekst szyfrogramu jest wynikiem operacji dodania ostatniego podklucza z wartością wyjściową operacji XS.



Rys. 3.1: Schemat blokowy algorytmu szyfrowania Hierocrypt-3

3.4.2 Operacja deszyfrowania.

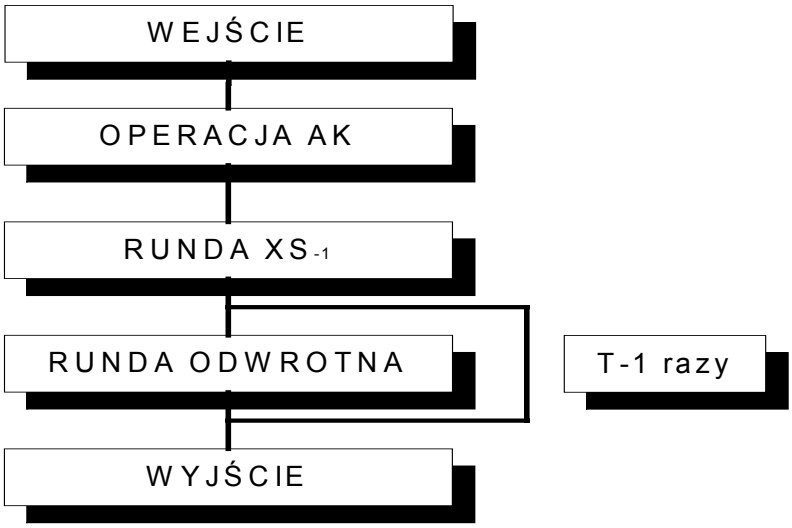
Operacja deszyfrowania polega na odwróceniu kolejności działania szyfrującego. Najpierw następuje dodanie ostatniego podklucza, potem operacja XS i w końcu T-1 razy operacja odwróconej rundy.

$X(T)_{(128)} = C_{(128)} \oplus K(T+1)_{(128)}$

$X(T-1)_{(128)} = XS^{-1}(X(T)_{(128)}) , K(T)_{(256)}))$,

$X(i-1)_{(128)} = \rho^{-1}(X(i)_{(128)}) , K(i)_{(256)}))$ $i = T-1, ...2, 1.$

$P_{(128)} = X(0)_{(128)}$,



Rys. 3.2: Schemat blokowy algorytmu deszyfrującego.

$X(i)_{(128)}$ jest wartością wyjściową po i-tej rundzie. Tekst szyfrogramu $C_{(128)}$ jest równy wartości $X(T+1)_{(128)}$. Natomiast $X(T-1)_{(128)}$ jest wartością wejściową operacji XS^{-1} ,a $X(T)_{(128)}$ jest wartością wyjściową z tej operacji. Tekst jawny jest wynikiem ostatniej operacji odwrotności rundy ρ^{-1}.

3.5 Specyfikacja operacji algorytmu.

3.5.1 Operacja podstawienia.

Jest to operacja, która jako jedyna w algorytmie jest operacją nieliniową i działa ona na każdy bajt w bloku danych równolegle działającymi skrzynkami podstawieniowymi (ang. *substitution box* w skrócie s-box) o wymiarach 8 x 8.

Skrzynka używana w Hierocrypcie jest oczywiście nielosowa, w myśl idei projektowania wg „strategii szerokiej ścieżki", która proponuje nam sboxy, które powstały na bazie deterministycznego działania. Procedura wyznaczenia skrzynek w Hierocrypcie wygląda następująco:

- Skrzynka jest wymiarów 8 x 8 więc brane będą pod uwagę wektory 8 bitowe, a operacje będą wykonywane w ciele GF(28), a pierwsza operacja polega na przepermutwaniu bitów obrębie każdego elementu, wg procedury:

I	1	2	3	4	5	6	7	8
Π(i)	3	7	5	8	6	2	4	1

- Potęgowanie modularne w ciele GF(2^8) z zadanym wielomianem pierwotnym
  $z^8 + z^6 + z^5 + z + 1$
  (Power($x_{(8)}$) = $x_{(8)}^{247}$). Potęga 247 została

wybrana ponieważ była to najwyższa wartość wykładnika dla którego charakterystyki różniocwa I liniowa sboxą miały wartość 2-6, czyli najlepszą możliwą dla tego typu skrzynek.

- Liniowa opracja, polegająca na dodaniu do każdego elementu wartości 0x07. Miała ona na celu uniemożliwienie pewnego trywialnego przejścia w skrzynce (element 0x00 przechodził na ten sam element), a także spowodować, żeby parametry statystyczne skrzynki dawały jak najbardziej losowy charakter. Korelacja wartości wejściowo – wyjściowych skrzynki ma najmniejszą wartość właśnie dla wartości 0x07.

3.5.2 Operacja MDS lower level.

Funkcja ta realizuje liniową transformację, która jest reprezentowana prze z macierz wymiaru 4x4 gdzie elementy macierzy są elementami z ciała $GF(2^8)$

$Y_{(32)} = mds_L(X_{(32)})$

$x_{1(8)} \| x_{2(8)} \| x_{4(8)} \| x_{4(8)} = X_{(32)}$

$y_{1(8)} \| y_{2(8)} \| y_{4(8)} \| y_{4(8)} = Y_{(32)}$

$Y_{1(8)}$		C4	65	C8	8B		$x_{1(8)}$
$Y_{2(8)}$		8B	C4	65	C8		$x_{2(8)}$
$Y_{3(8)}$	=	C8	8B	C4	65	*	$x_{3(8)}$
$Y_{4(8)}$		65	C8	8B	C4		$x_{4(8)}$

$y_{1(8)} = C4*x_{1(8)} \oplus 65*x_{2(8)} \oplus C8*x_{4(8)} \oplus 8B*x_{4(8)}$

$y_{2(8)} = 8B*x_{1(8)} \oplus C4*x_{2(8)} \oplus 65*x_{4(8)} \oplus C8*x_{4(8)}$

$y_{3(8)} = C8*x_{1(8)} \oplus 8B*x_{2(8)} \oplus C4*x_{4(8)} \oplus 65*x_{4(8)}$

$y_{4(8)} = 65*x_{1(8)} \oplus C8*x_{2(8)} \oplus 8B*x_{4(8)} \oplus C4*x_{4(8)}$

Transformacja ta operuje na słowach 32 bitowych, czyli na całym bloku operują cztery takie macierze. Polega na wymnożeniu odpowiednich wartości ustalonych z odpowiednimi wartościami wejściowymi w ciele $GF(2^8)$, dla którego wybranym wielomianem charakterystycznym jest $x^8 + x^6 + x^5 + x + 1$.

Autorzy szyfru mówią o niej jako o operacji spełniającej rolę dyfuzji lokalnej. Powstała ona na bazie teorii kodów i zapewnia ona rozpropagowywanie różnic charakterystyk liniowych i różnicowych w stopniu maksymalnym. Macierz wykorzystywana w tym przekształceniu jest macierzą kodu MDS.

3.5.3 Operacja MDS higher level.

Funkcja ta realizuje liniową transformację, składającą się z operacji xor pomiędzy 8 bitowymi blokami danych dobieranymi wg macierzy MDS_H.

$Y_{(32)} = MDS_H(X_{(32)})$

$x_{1(8)} \| x_{2(8)} \| \ldots \| x_{15(8)} \| x_{16(8)} = X_{(128)}$

$y_{1(8)} \| y_{2(8)} \| \ldots \| y_{15(8)} \| y_{16(8)} = Y_{(128)}$

$Y_{1(8)}$		1	0	1	0	1	0	1	0	1	1	0	1	1	1	1	1	$x_{1(8)}$
$Y_{2(8)}$		1	1	0	1	1	1	0	1	1	1	1	0	0	1	1	1	$x_{2(8)}$
$Y_{3(8)}$		1	1	1	0	1	1	1	0	1	1	1	1	0	0	1	1	$x_{3(8)}$
$Y_{4(8)}$		0	1	0	1	0	1	0	1	1	0	1	0	1	1	1	0	$x_{4(8)}$
$Y_{5(8)}$		1	1	1	1	1	0	1	0	1	0	1	0	1	1	0	1	$x_{5(8)}$
$Y_{6(8)}$		0	1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	$x_{6(8)}$
$Y_{7(8)}$		0	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1	$x_{7(8)}$
$Y_{8(8)}$	=	1	1	1	0	0	1	0	1	0	1	0	1	1	0	1	0	* $x_{8(8)}$
$Y_{9(8)}$		1	1	0	1	1	1	1	1	1	0	1	0	1	0	1	0	$x_{9(8)}$
$Y_{10(8)}$		1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	1	$x_{10(8)}$
$Y_{11(8)}$		1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	$x_{11(8)}$
$Y_{12(8)}$		1	0	1	0	1	1	1	0	0	1	0	1	0	1	0	1	$x_{12(8)}$
$Y_{13(8)}$		1	0	1	0	1	1	0	1	1	1	1	1	1	0	1	0	$x_{13(8)}$
$Y_{14(8)}$		1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	1	$x_{14(8)}$
$Y_{15(8)}$		1	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	$x_{15(8)}$
$Y_{16(8)}$		0	1	0	1	1	0	1	0	1	1	1	0	0	1	0	1	$x_{16(8)}$

Operacja ta operuje na całej długości bloku. Określana ona jest często jako globalna dyfuzja . Sposób jej działania pokazuje przykład wyliczenia jednego z wyjściowych bajtów

$$y_{1(8)} = x_{1(8)} \oplus x_{3(8)} \oplus x_{5(8)} \oplus x_{7(8)} \oplus x_{1(8)} \oplus x_{1(8)} \oplus x_{9(8)} \oplus$$
$$x_{10(8)} \oplus x_{12(8)} \oplus x_{13(8)} \oplus x_{14(8)} \oplus x_{15(8)} \oplus x_{16(8)}$$

3.5.4 Specyfikacja rundy szyfru

Funkcja rundy szyfru to złożenie dwóch operacji:

Operacji XS oraz MDS_H. Pierwsza funkcja jest kompozycją operacji: równolegle działających skrzynek podstawieniowych, dodania klucza, czterech równoległych operacji mds_L oraz ponownego dodania klucz i ostatecznego dokonania podstawienia za pomocą tych samych skrzynek.

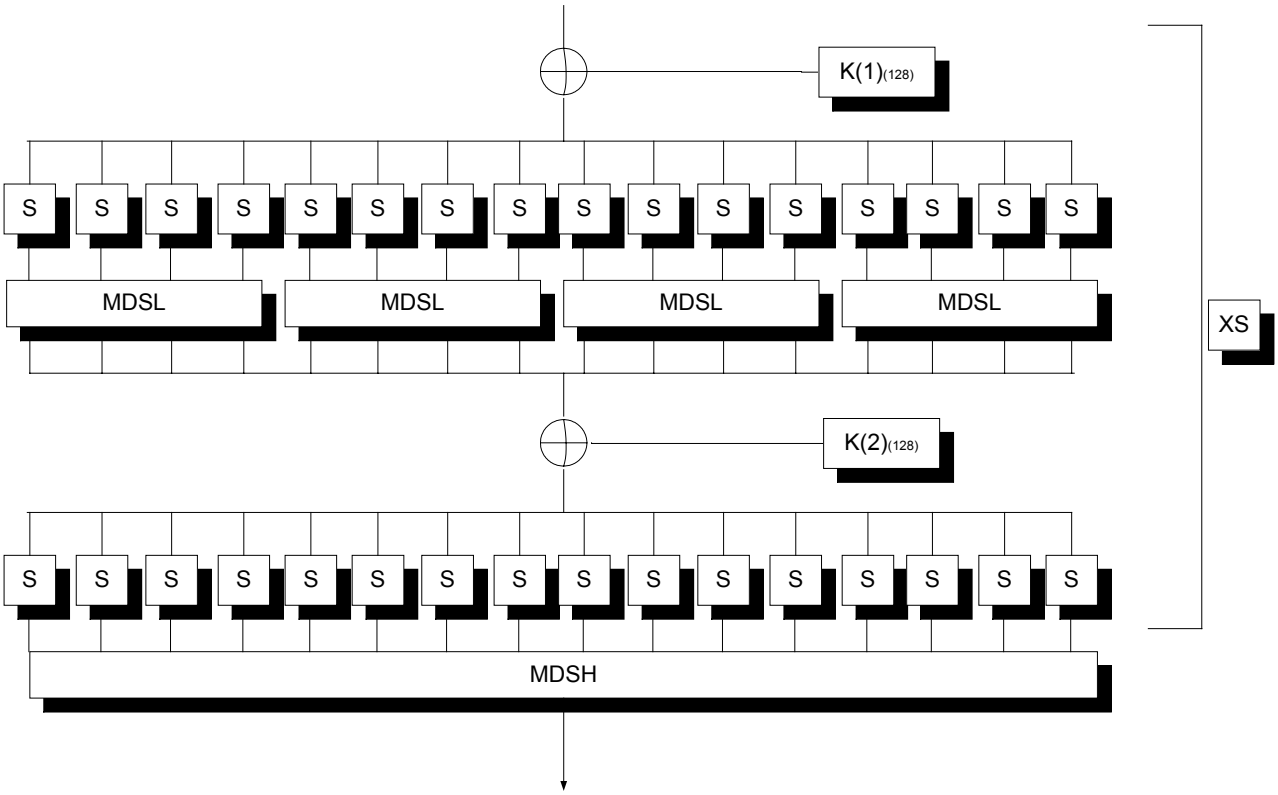
$$Y_{(128)} = XS(X_{(128)}, K_{(256)})$$

$$K1(128)||K2(128) = K(256)$$

$$XS(X_{(128)}, K_{(256)}) = S(MDS_L(S(X_{(128)} \oplus K_{1(128)})) \oplus K_{2(128)})$$

I wreszcie funkcja rundy ma ostatecznie postać:

$$\rho\ (X_{(128)}, K_{(256)}) = MDS_H\ (XS\ (X_{(128)}, K_{(256)})$$



Rys.3.3: Funkcja rundy algorytmu blokowego Hierocrypt.

3.5.5 Operacja odwrotna do podstawiania.

Operacja ta, podobnie jak kolejne operacje z rozdziałów 3.5.6, 3.5.7 i 3.5.8 są wykorzystywana przy algorytmie deszyfrującym. Jeżeli chodzi o sposób wyznaczenia operacji odwrotnej do skrzynki podstawieniowej to polega on na dokonaniu permutacji odwrotnej elementów z $GF(2^n)$ w stosunku do operacji podstawienia omówionej w rozdziale 3.5.1

3.5.6 Operacja odwrotna do MDS lower level.

Funkcja ta realizuje liniową transformację, która jest reprezentowana prze z macierz wymiaru 4x4 gdzie elementy macierzy są elementami z ciała $GF(2^8)$

$Y_{(32)} = mds_L^{-1}(X_{(32)})$

$x_{1(8)} \| x_{2(8)} \| x_{4(8)} \| x_{4(8)} = X_{(32)}$

$y_{1(8)} \| y_{2(8)} \| y_{4(8)} \| y_{4(8)} = Y_{(32)}$

$Y_{1(8)}$		82	C4	34	F6		$x_{1(8)}$
$Y_{2(8)}$		F6	82	C4	34		$x_{2(8)}$
$Y_{3(8)}$	=	34	F6	82	C4	*	$x_{3(8)}$
$Y_{4(8)}$		C4	34	F6	82		$x_{4(8)}$

$$y_{1(8)} = 82*x_{1(8)} \oplus C4*x_{2(8)} \oplus 34*x_{4(8)} \oplus F6*x_{4(8)}$$
$$y_{2(8)} = F6*x_{1(8)} \oplus 82*x_{2(8)} \oplus C4*x_{4(8)} \oplus 34*x_{4(8)}$$
$$y_{3(8)} = 34*x_{1(8)} \oplus F6*x_{2(8)} \oplus 82*x_{4(8)} \oplus C4*x_{4(8)}$$
$$y_{4(8)} = C4*x_{1(8)} \oplus 34*x_{2(8)} \oplus F6*x_{4(8)} \oplus 82*x_{4(8)}$$

Operacja ta operuje na słowach 32 bitowych, czyli na całym bloku operują cztery takie macierze. Jest ona odwrotną do operacji mds_L. Jej działanie jest dokładnie takie same jak w mds_L – polega na realizacji wymnożenia elementów wejściowych z $GF(2^8)$ przez odpowiednie elementy stałe, podane w macierzy kodu. Wybranym wielomianem charakterystycznym ciała jest wielomian $x^8 + x^6 + x^5 + x + 1$.

Sposób wyznaczenia odwrotności macierzy mds_L, polega na obliczeniu macierzy odwrotnej do macierzy generującej kod MDS. Macierz ta także jest macierzą kodu MDS, więc w oczywisty sposób dyfuzja tej macierzy także jest maksymalna. Przy okazji tej operacji warto wspomnieć, że autorzy szyfru tak dobierali operację mds_L i operację do niej odwrotną aby suma wag hamminga wektorów należących do macierzy generujących kod były jak najbardziej zbliżone.

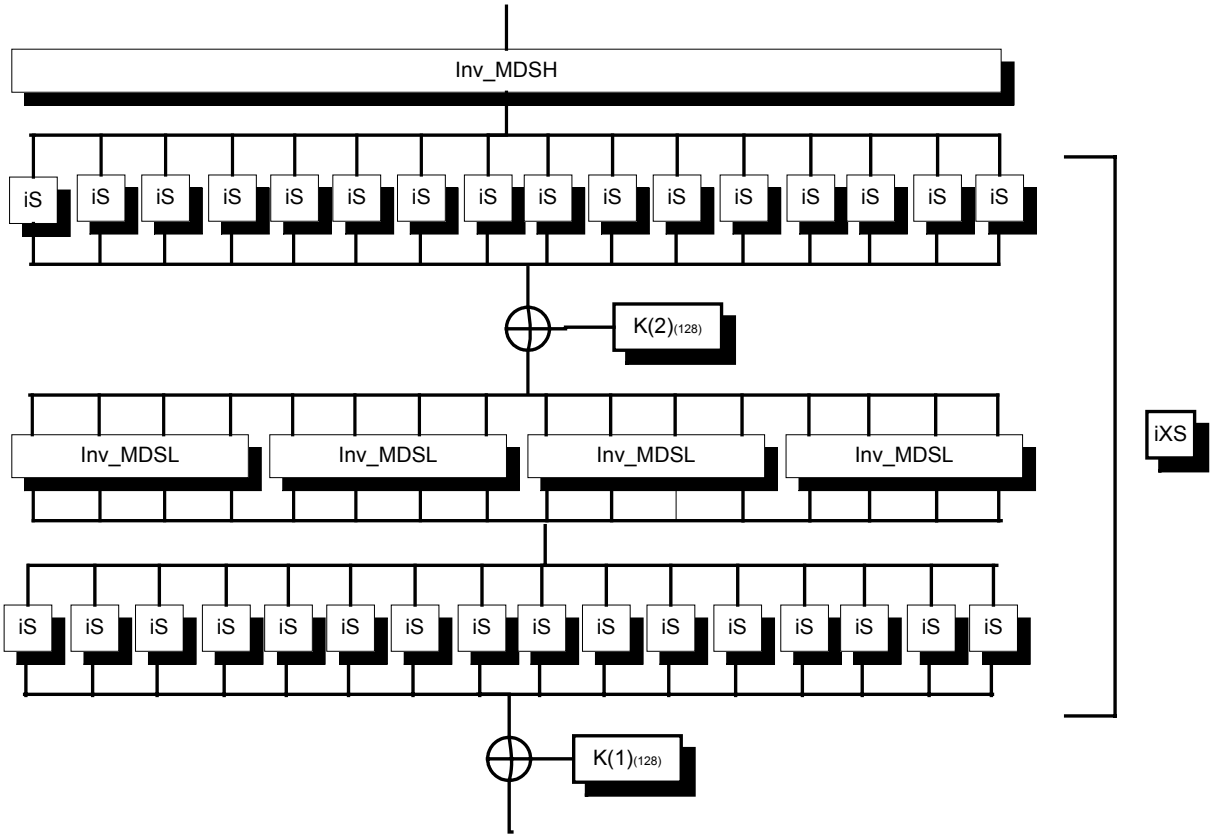
3.5.7 Operacja odwrotna do MDS higher level.

Funkcja ta realizuje liniową transformację, składającą się z operacji xor pomiędzy 8 bitowymi blokami danych dobieranymi wg macierzy MDS_H.

$$Y_{(128)} = MDS_H^{-1}(X_{(128)})$$
$$x_{1(8)} \,||\, x_{2(8)} \,||\, \ldots \,||\, x_{15(8)} \,||\, x_{16(8)} = X_{(128)}$$
$$y_{1(8)} \,||\, y_{2(8)} \,||\, \ldots \,||\, y_{15(8)} \,||\, y_{16(8)} = Y_{(128)}$$

3.5.8 Specyfikacja rundy odwrotnej.

Funkcja rundy szyfru to złożenie dwóch operacji:
Operacji XS^{-1} (na rysunku ma oznaczenie iXS) oraz MDS_H^{-1}(na rysunku oznaczone jako Inv_MDSH). Pierwsza funkcja jest kompozycją operacji: równolegle działających odwróconych skrzynek podstawieniowych, dodania klucza, czterech równoległych operacji mds_L ponownego dokonania podstawienia za pomocą tych samych skrzynek i finalnego dodania podklucza.



Rys.3.4: Operacja odwrotna do operacji rundy.

3.6 Algorytm generacji podkluczy.

3.6.1 Wprowadzenie do algorytmu generacji podkluczy.

Duża komplikacja algorytmu generacji podkluczy w Hierocrypcie była jedną z głównych przyczyn odrzucenia go już po pierwszej fazie trwania konkursu NESSIE.

Podstawą jego działania była idea generacji dwóch rodzajów, zależnych od siebie podkluczy dla każdej rundy:

- klucza przejściowego rundy
- klucza głównego rundy.

Główną przyczyna takiego wyboru algorytmu było dążenie autorów do swoistej symetrii sposobu wyznaczania podkluczy. Symetria ma polegać na tym, że dla każdej wersji algorytmu Hierocrypt, sześcio, siedmio lub ośmio rundowego, wyznaczane podklucze przejściowe mają być symetryczne względem środkowej rundy. Dla algorytmu z sześcioma rundami takimi samymi podkluczami przejściowymi są 1 i 7, 2 i 6 oraz 3 i 5 podklucz. Podobnie jest w wersjach algorytmu z dłuższymi kluczami głównymi.

Powodem takiego wyboru było założenie, że wszelkie operacje składowe algorytmu powinny trwać w jak najbardziej zbliżonym czasie. Implementacja algorytmu deszyfrowania często przeprowadzana jest w następujący sposób: najpierw musi dojść do wyznaczenia ostatniego podklucza rundy, a następnie poprzez realizację odwrotności rund klucza wyznaczane są od końca wszystkie podklucze rundy. Aby uniknąć swoistego marnowania czasu na wykonanie całego algorytmu generacji podkluczy twórcy szyfru zastosowali symetrię przy okazji wyznaczania podkluczy przejściowych. Oczywistym stał się fakt, że trzeba było zapewnić niepowtarzalność i jak największą niezależność podkluczy rund, w związku z tym potrzeba było dodatkowego algorytmu wyznaczania podkluczy rund z podkluczy przejściowych.

Przyjmijmy oznaczenie:

K – klucz główny

Z – podklucz przejściowy

$K_{i(64)}$ – i–ta część klucza głównego

$Z^{(-1)}{}_{i(64)}$ – i-ta część pierwszego podklucza przejściowego

G() – wybrana stała.

$X_{(n)}$ – binarna wartość wejściowa dowolnej funkcji o długości n

$Y_{(n)}$ – binarna wartość wyjściowa dowolnej funkcji o długości n.

3.6.2 Operacja M5e.

Operacja $\mathbf{M_{5E}}$ jest częścią składową algorytmu generacji podkluczy. Składa się z konkatenacji dwóch 32-bitowych liniowych transformacji, gdzie każdy 8 bitowy blok danych jest uważany za element ciała $GF(2^n)$. Operacja ma charakter dyfuzyjny.

$Y_{(64)} = M_{5E}\ (X_{(64)})$

$x_{1(8)} \parallel x_{2(8)} \parallel .... \parallel x_{8(8)} = X_{(64)}$

$y_{1(8)} \parallel y_{2(8)} \parallel .... \parallel y_{8(8)} = Y_{(64)}$

$$\begin{bmatrix} y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \\ y_{4(8)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \\ x_{4(8)} \end{bmatrix}$$

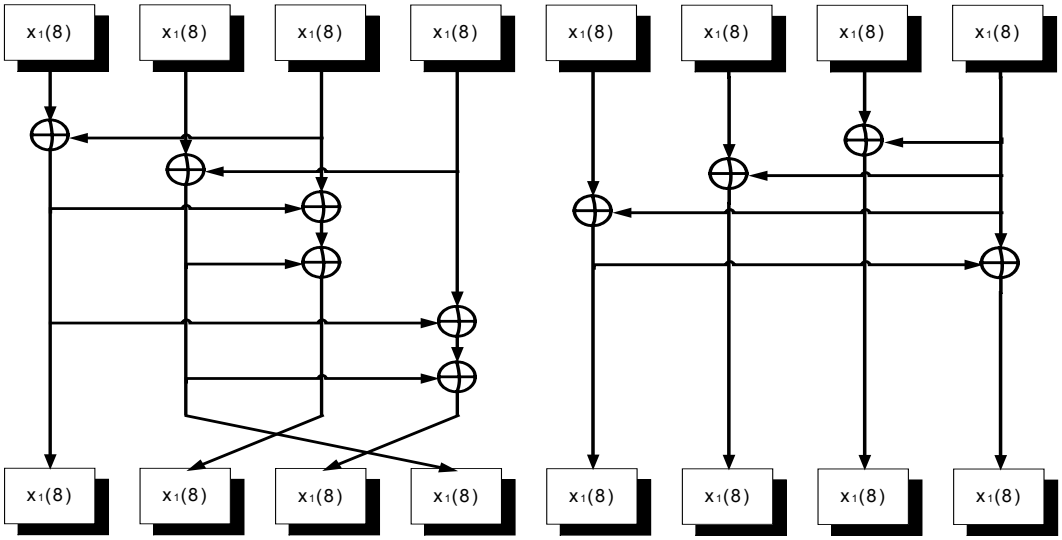
$$\begin{bmatrix} y_{5(n)} \\ y_{6(n)} \\ y_{7(n)} \\ y_{8(8)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} * \begin{bmatrix} x_{5(8)} \\ x_{6(8)} \\ x_{7(8)} \\ x_{8(8)} \end{bmatrix}$$

64-bitowy blok wejściowy jest dzielony na 8 wektorów 8 bitowych i na tych wektorach realizowana jest operacja mnożenia przez macierz M5e.

Np. pierwsze osiem bitów z całego 64 bitowego bloku wyliczamy w następujący sposób:

$y_{1(8)} = x_{1(8)} \oplus x_{3(8)}$;

Realizacja tej operacji może wyglądać w następujący sposób:



Rys.3.5: Schemat przykładowej realizacji operacji M5e.

3.6.3 Operacja Mb3.

Operacja $\mathbf{M_{B3}}$ składa się z konkatenacji dwóch 32-bitowych liniowych transformacji, gdzie każdy 8 bitowy blok danych jest uważany za element ciała $GF(2^n)$. Jest ona odwrotnością operacji M5e i podobnie jak ona występuje w algorytmie generacji podkluczy.

$Y_{(64)} = M_{B3}\ (X_{(64)})$

$x_{1(8)} \parallel x_{2(8)} \parallel .... \parallel x_{8(8)} = X_{(64)}$

$y_{1(8)} \parallel y_{2(8)} \parallel .... \parallel y_{8(8)} = Y_{(64)}$

$y_{1(8)}$		0	1	0	1		$x_{1(8)}$
$y_{2(8)}$		1	0	1	0		$x_{2(8)}$
$y_{3(8)}$	=	1	1	0	1	*	$x_{3(8)}$
$y_{4(8)}$		1	0	1	1		$x_{4(8)}$
$y_{5(n)}$		1	1	0	0		$x_{5(8)}$
$y_{6(n)}$		0	1	1	0		$x_{6(8)}$
$y_{7(n)}$	=	1	0	1	1	*	$x_{7(8)}$
$y_{8(8)}$		1	0	0	1		$x_{8(8)}$

3.6.4 Operacja permutacji $P^{(n)}$.

Operacja $P^{(n)}$ składa się z liniowej transformacji dla danych wejściowych $X_{(4n)}$, które są podzielone na 4 n–bitowe bloki $x_{i(n)}$
(i = 1, 2, 3, 4), gdzie każdy element jest uważany jako element z ciała $GF(2^n)$.

$Y_{(4n)} = P^{(n)}(X_{(4n)})$

$x_{1(n)} \,||\, x_{2(n)} \,||\, x_{4(n)} \,||\, x_{4(n)} = X_{(4n)}$

$y_{1(n)} \,||\, y_{2(n)} \,||\, y_{4(n)} \,||\, y_{4(n)} = Y_{(4n)}$

$y_{1(n)}$		1	0	1	0		$x_{1(n)}$
$y_{2(n)}$		0	1	0	1		$x_{2(n)}$
$y_{3(n)}$	=	0	1	1	1	*	$x_{3(n)}$
$y_{4(n)}$		1	0	1	1		$x_{4(n)}$

Funkcja jest parametryzowana wielkością słowa na jakim operuje operacja. W algorytmie generacji podkluczy funkcja ta ma wartości parametru n=16 oraz 32.
Powstała ona poprzez wymnożenie dwóch inwolucyjnych macierzy.

1	0	1	0		1	0	0	0		1	0	1	0
0	1	0	1		0	1	0	0		0	1	0	1
0	1	1	1	=	0	1	1	0	*	0	0	1	0
1	0	1	1		1	0	0	1		0	0	0	1

3.6.5 Operacja odwrotna do permutacji $P^{-1(n)}$.

Operacja powstała poprzez wymnożenie macierzy inwolucyjnych z poprzedniej funkcji, ale w innej kolejności.

1	1	1	0		1	0	1	0		1	0	0	0
1	1	0	1		0	1	0	1		0	1	0	0
0	1	1	0	=	0	0	1	0	*	0	1	1	0
1	0	0	1		0	0	0	1		1	0	0	1

Powstała w ten sposób macierz dyfuzyjna $P^{-1(n)}$ jest operacją która jest odwrotną do $P^{(n)}$. Jednakże w algorytmie wykorzystywana jest tylko dla długości słowa n=32.

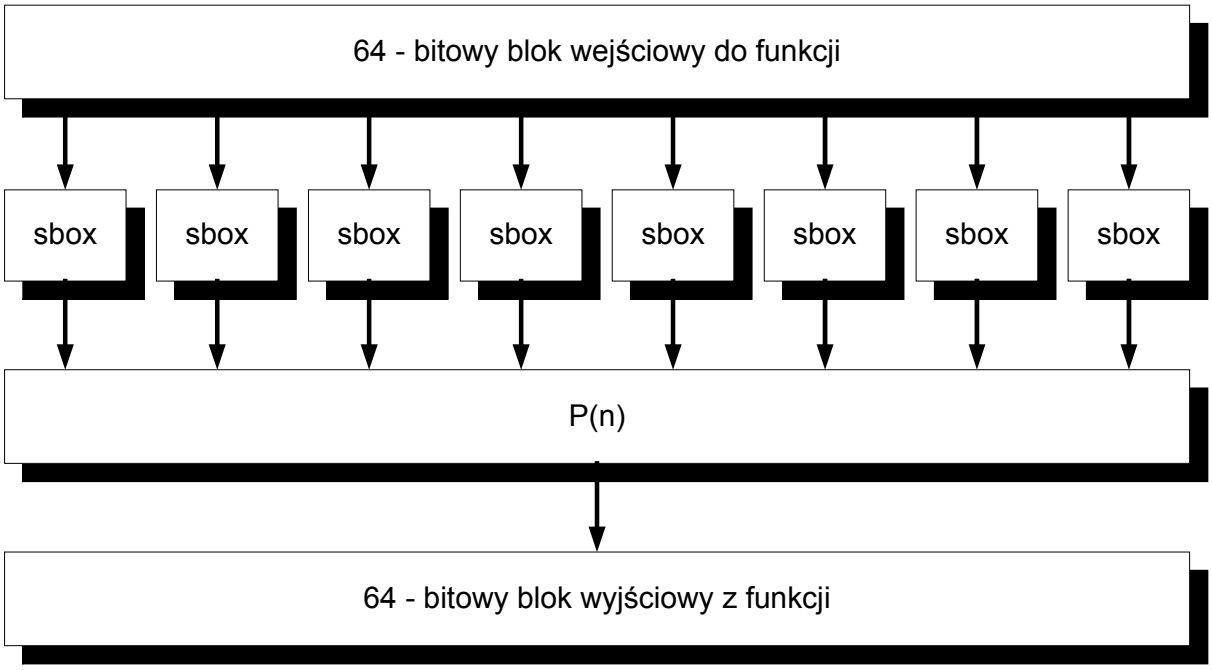
3.6.6 Operacja Fσ.

Funkcja $\mathbf{F_\sigma}$ jest nieliniową funkcją, która składa się z 8 funkcji sbox 8x8, oraz operacji $P^{(n)}$ (dla n =16),dane wejściowe są podawane w postaci ośmiu ośmio-bitowych bloków danych, a na wyjściu operacji mamy blok 64 – bitowy.

$Y_{(64)} = F_\sigma (X_{(64)})$

$x_{1(8)} \,||\, x_{2(8)} \,||\, ... \,||\, x_{8(8)} = X_{(64)}$

$P^{(16)}(s(y_{1(8)}) \,||\, s(y_{2(8)}) \,||\, ... \,||\, s(y_{8(8)})) = Y_{(64)}$



Rys.3.6: Operacja F_σ.

3.6.7 Operacja przygotowania podklucza.

Długość klucza głównego ma wpływ na ilość rund szyfru. Bez względu na to czy klucz ten jest 128, 192 czy 256 bitowy zawsze podklucz rundy ma długość 256 bitów. Pierwszą operacją, która zapewnia ustandaryzowanie do długości 256 bitów jest operacja określana jako padding. Operacja przygotowanie polega na podstawieniu, dla klucza głównego o długości:

- 128-bit key

$K_{128} = K_{1(64)} \,||\, K_{2(64)}$

$Z^{(-1)}{}_{1(64)} = K_{1(64)}, \quad Z^{(-1)}{}_{2(64)} = K_{2(64)}, \quad Z^{(-1)}{}_{3(64)} = K_{1(64)}, \quad Z^{(-1)}{}_{4(64)} = H_3 || H_2,$

- 192-bit key

$K_{192} = K_{1(64)} \| K_{2(64)} \| K_{3(64)}$

$Z^{(-1)}{}_{1(64)} = K_{1(64)}, \quad Z^{(-1)}{}_{2(64)} = K_{2(64)}, \quad Z^{(-1)}{}_{3(64)} = K_{3(64)}, \quad Z^{(-1)}{}_{4(64)} = H_2 \| H_3,$

- 256-bit key

$K_{256} = K_{1(64)} \| K_{2(64)} \| K_{3(64)} \| K_{4(64)}$

$Z^{(-1)}{}_{1(64)} = K_{1(64)}, \quad Z^{(-1)}{}_{2(64)} = K_{2(64)}, \quad Z^{(-1)}{}_{3(64)} = K_{3(64)}, \quad Z^{(-1)}{}_{4(64)} = K_{4(64)},$

3.6.8 Operacja wybielająca podklucza.

Funkcja wybielająca (ang. p*re-whitening*) (σ_0 – function) różni się od operacji aktualizacji podklucza przejściowego (σ – function) jedynie brakiem występowania operacji $P^{(32)}$.

$Z^{(0)}{}_{(256)} = \sigma_0(Z^{(-1)}{}_{(256)}, G^{(0)}{}_{(64)})$

Definicja tej funkcji wygląda następująco:

$Z^{(0)}{}_{3(64)} = M_{5E}(Z^{(-1)}{}_{3(64)} \oplus G^{(0)}{}_{(64)}),$

$Z^{(0)}{}_{4(64)} = M_{5E}(Z^{(-1)}{}_{4(64)}),$

$Z^{(0)}{}_{1(64)} = Z^{(-1)}{}_{2(64),}$

$Z^{(0)}{}_{2(64)} = Z^{(-1)}{}_{1(64)} \oplus F_0(Z^{(-1)}{}_{2(64)}, Z^{(-1)}{}_{3(64)})$

$G^{(0)}{}_{(64)} = G_0(5) = H_1 \| H_0,$

3.6.9 Operacja aktualizacji podklucza przejściowego.

Operacja ta posiada oznaczenie σ

$Z^{(t)}{}_{(256)} = \sigma(Z^{(t-1)}{}_{(256)}, G^{(t)}{}_{(64)}),$

Definicja funkcji:

$W^{(t-1)}{}_{1(64)} \| W^{(t-1)}{}_{2(64)} = P^{(32)}(Z^{(t-1)}{}_{1(64)} \| Z^{(t-1)}{}_{2(64)})$

$Z^{(t)}{}_{1(64)} = Z^{(t-1)}{}_{2(64)},$

$Z^{(t)}{}_{2(64)} = Z^{(t-1)}{}_{1(64)} \oplus F_\sigma(Z^{(t-1)}{}_{2(64)} \oplus Z^{(t)}{}_{3(64)}),$

$Z^{(t)}{}_{3(64)} = M_{5E}(W^{(t-1)}{}_{1(64)}) \oplus G^{(t)}{}_{(64)},$

$Z^{(t)}{}_{4(64)} = M_{5E}(W^{(t-1)}{}_{2(64)}),$

3.6.10 Operacja odwrotnej aktualizacji podklucza przejściowego σ^{-1}.

Operacja ta posiada oznaczenie σ^{-1}

$Z^{(t)}_{(256)} = \sigma^{-1}(Z^{(t-1)}_{(256)}, G^{(t)}_{(64)})$,

Definicja finkcji:

$Z^{(t)}_{1(64)} = Z^{(t-1)}_{2(64)} \oplus F_\sigma(Z^{(t-1)}_{1(64)} \oplus Z^{(t-1)}_{3(64)})$,

$Z^{(t)}_{2(64)} = Z^{(t-1)}_{1(64)}$,

$W^{(t)}_{1(64)} = M_{B3}(Z^{(t-1)}_{3(64)} \oplus G^{(t)}_{(64)})$,

$W^{(t)}_{2(64)} = M_{B3}(Z^{(t-1)}_{4(64)})$,

$Z^{(t)}_{3(64)} || Z^{(t)}_{4(64)} = P^{(32)\,-1}(W^{(t-1)}_{1(64)} || W^{(t-1)}_{2(64)})$

3.6.11 Operacja generacji podklucza rundy.

Wyznaczenie podkluczy właściwych poszczególnych rund jest zależne od wartości kluczy przejściowych. Dla rund o numerach mniejszych niż wartość t_{turn} ($t_{turn} = 4$ dla klucza głównego o długości 128, 192 bity oraz $t_{turn} = 5$ dla klucza 256) operacja ta przebiega następująco:

Operacja wyznaczania podkluczy rundy dla $1 <= t <= t_{turn}$.

$V^{(t)}_{(64)} = F_\sigma(Z^{(t-1)}_{2(64)} \oplus Z^{(t-1)}_{3(64)})$,

$K^{(t)}_{1(64)} = Z^{(t-1)}_{1(64)} \oplus V^{(t)}_{(64)}$,

$K^{(t)}_{2(64)} = Z^{(t)}_{3(64)} \oplus V^{(t)}_{(64)}$,

$K^{(t)}_{3(64)} = Z^{(t)}_{4(64)} \oplus V^{(t)}_{(64)}$,

$K^{(t)}_{4(64)} = Z^{(t-1)}_{2(64)} \oplus Z^{(t)}_{4(64)}$,

Operacja wyznaczania podkluczy rundy dla $t_{turn}+1 <= t <= T+1$.

$V^{(t)}_{(64)} = F_\sigma(Z^{(t-1)}_{1(64)} \oplus Z^{(t)}_{3(64)})$,

$K^{(t)}_{1(64)} = Z^{(t)}_{1(64)} \oplus Z^{(t-1)}_{3(64)}$,

$K^{(t)}_{2(64)} = W^{(t)}_{1(64)} \oplus V^{(t)}_{(64)}$,

$K^{(t)}_{3(64)} = W^{(t)}_{2(64)} \oplus V^{(t)}_{(64)}$,

$K^{(t)}_{4(64)} = Z^{(t-1)}_{1(64)} \oplus W^{(t)}_{2(64)}$,

3.6.12 Wybrane wartości stałych zależnych od rundy.

Wprowadzono, aby uniknąć zagrożenia ataku z kluczami zależnymi. Stałe do poszczególnych rund, są wyznaczone poprzez konkatenację 2 z 4 32-bitowych wartości wybranych, jak podają autorzy, kompletnie irracjonalnie:

H_0 = 0x5A8279999 = trunc(2^{-2}/4),
H_1 = 0x6ED9EBA1 = trunc(3^{-2}/4),
H_2 = 0x8F1BBCDC = trunc(5^{-2}/4),
H_3 = 0xCA62C1D6 = trunc(10^{-2}/4),

Gdzie trunc(x) = $\lfloor 2^{32}x \rfloor$.

Wartości stałe zależne od rundy:

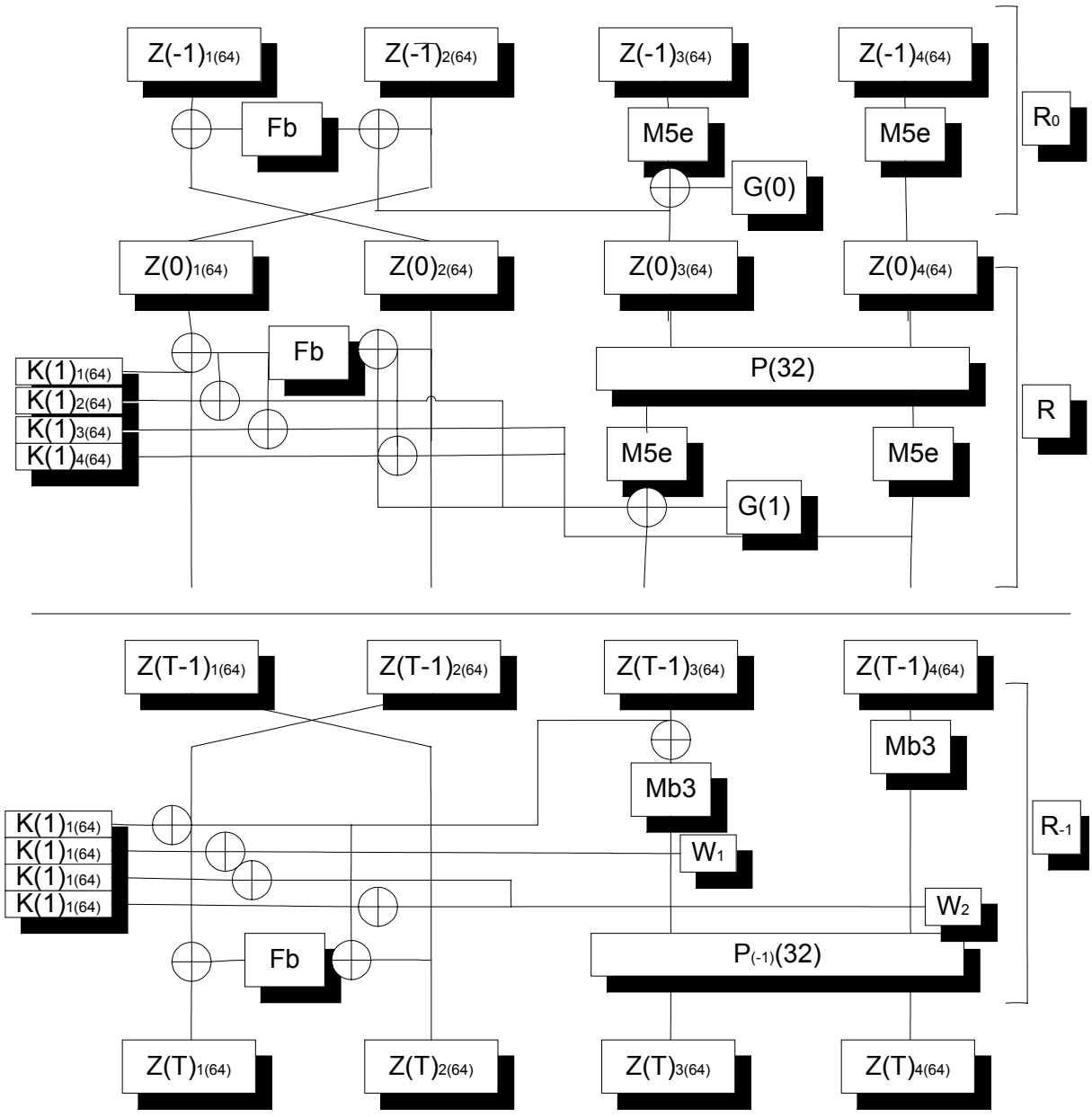
$G_0(0) = H_3 \,||\, H_0$, $G_0(1) = H_2 \,||\, H_1$,
$G_0(2) = H_1 \,||\, H_3$, $G_0(3) = H_0 \,||\, H_2$,
$G_0(4) = H_2 \,||\, H_3$, $G_0(5) = H_1 \,||\, H_0$,

3.6.13 Algorytm generacji podklucza rundy dla różnych długości klucza głównego.

Tabela generacji podkluczy z klucza głównego 128 bitowego.

Klucz rundy	t	operacja	$G^{(t)}_{(64)}$
-	-1 (PAD)	-	$H_3 \,||\, H_2$
-	0 (PW)	σ_0	$G_0(5)$
$K^{(1)}_{(256)}$	1	σ	$G_0(0)$
$K^{(2)}_{(256)}$	2	σ	$G_0(1)$
$K^{(3)}_{(256)}$	3	σ	$G_0(2)$
$K^{(4)}_{(256)}$	4	σ	$G_0(3)$
$K^{(5)}_{(256)}$	5	σ^{-1}	$G_0(3)$
$K^{(6)}_{(256)}$	6	σ^{-1}	$G_0(2)$
$K^{(7)}_{(256)}$	7	σ^{-1}	$G_0(1)$

Tabele, które specyfikują algorytm generacji podkluczy do poszczególnych rund algorytmu przy różnym klucz głównym wyglądają podobnie, a cechą wspólną jest symetryczność przekształceń aktualizacji klucza przejściowego względem środkowej rundy.



Rys.3.7: Algorytm generacji podklucza rundy.

3.7 Bezpieczeństwo szyfru.

Najbardziej fundamentalnym kryterium bezpieczeństwa szyfru blokowego jest długość klucza, która ma zapobiec efektywności i skuteczności ataku polegającego na pełnym przeszukaniu przestrzeni klucza (ang. *exhaustive attack or key search*). Oprócz długości klucz miarą bezpieczeństwa szyfru jest jego odporność na znane ataki kryptanalityczne: kryptanalizę różnicową (ang. *Differential cryptanalysis*) , kryptanalizę liniową (ang. *Linear cryptanalysis*), kryptanalizę różnicową wyższego rzędu (ang. *High-order differential cryptanalysis*), atak interpolacyjny (ang. *Interpolation attack*), atak na szyfry typu SQUARE (ang. *SQUARE-dedicated attack*), różniczki obcięte (ang. *Truncated differential attack*), różniczki niemożliwe (ang. *Impossible differential attack*).

Podczas procesu oceny własnej algorytmu, dokonanej przez autorów szyfru, nie wykryto żadnych poważniejszych usterek, ani wad algorytmu, które mogłyby go dyskwalfikować. Algorytm był projektowany tak, aby można było potwierdzić jego bezpieczeństwo za pomocą oszacowania prawdopodobieństw najlepszych charakterystyk.

3.8 Propozycje ataków na szyfr.

Podczas otwartego procesu oceniania, trwającego podczas pierwszej fazy konkursu NESSIE, algorytm Hierocrypt był poddany szczegółowym analizom. Wynikiem tego były ataki, które choć obniżające tylko w nie wielkim stopniu margines bezpieczeństwa, stały się przyczyną odpadnięcia algorytmu z dalszej rywalizacji o miano najlepszego blokowego symetrycznego algorytmu.

Paul Barreto, twórca między innymi algorytmów Khazad i Anubis, startujących w ramach konkursu zaproponował atak typu SQUARE na 3 rundy dla wersji 128 bitowej oraz ataki 3,5 rundowe dla wersji z dłuższym kluczem. Twórcy algorytmu utrzymują nadal zdanie, że atak ten, nie jest niczym innym jak przedstawiona przez nich ocena własna algorytmu. Twierdzą, że liczba rund nie jest przypadkowo dobrana – chodziło im bowiem między innymi o uodpornienie algorytmu przeciw atakowi tego typu.

Kolejną próbą kryptoanalityczną było szukanie różniczek niemożliwych. Cheon znalazł różniczki dla dwóch rund i pokazał dalej, że atak tego typu można przeprowadzić nawet dla trzech rund, jednak złożoność tego ataku jest tak ogromna, że nawet gdyby Hierocrypt miał tylko trzy rundy to i tak atak ten nie byłby bardziej efektywny niż pełne przeszukiwanie przestrzeni klucza.

Co ciekawe także szukano słabości w algorytmie generowania podkluczy, pomimo że wcześniej już stwierdzono dużą złożoność jego i trudność analizy. Pomysł wyznaczaniem

podkluczy rund z podkluczy przejściowych raczej nie miał przeciwników. Stwierdzono natomiast, że runda klucza powinna być tak skonstruowana, żeby nie było prostych relacji pomiędzy podkluczami z kolejnych rund. Dokonano analizy algorytmu pod tym kątem i stwierdzono, że jest bardzo duża ilość takich właśnie prostych relacji pomiędzy bitami podkluczy rund. Nie udało się jednak dokonać nowego ataku przy wykorzystaniu tego faktu.

Algorytm Hierocrypt pomimo że wydaje się być bardzo mocnym pod względem odporności na znane ataki został odrzucony ponieważ zaproponowane ataki zostały uznane za poważne argumenty, naruszające margines bezpieczeństwa.

IV. Projekt implementacji sprzętowej algorytmu Hierocrypt.

4.1 Założenia projektowe.

Następujące założenia projektowe postanowiłem przyjąć w momencie kiedy zacząłem realizować projekt implementacji sprzętowej algorytmu Hierocrypt w strukturach programowalnych:

- Realizowaną wersją będzie Hierocrypt-3, który będzie korzystał z 128 bitowego bloku danych używając 128 bitowego klucza.
- Przeprowadzić implementację w układzie oferowanym przez firmę ALTERA – układ ma być z rodziny układów FLEX 10K. Bardzo ważnym założeniem jest konieczność zrealizowania funkcji szyfrowania i deszyfrowania w jak najmniejszej liczbie układów. (Co najwyżej po jednym dla każdej z funkcji: szyfrowania i deszyfrowania).
- Realizację projektu będę realizował w systemie projektowym ALTERY Max+PlusII.
- Najważniejszym kryterium projektowym jest jednak uzyskanie jak największej szybkości szyfrowania układem. Jest to spowodowane tym, że algorytm jest oceniany jako ten, który ma słabe predyspozycje jeśli chodzi o szybkość przetwarzania, szczególnie sprzętowego. Była to zresztą jedna z przyczyn jego odpadnięcia z konkursu.
- Dodatkowym założeniem będzie też możliwość przenoszalności projektu na inne układy spoza oferty firmy ALTERA, ale jest to założenie, które postawiłem przed projektem jeszcze przed faktycznym zapoznaniem z specyfikacją szyfru.

4.2 Wnioski wynikające z założeń projektowych.

Przyjęte założenia projektowe dokładnie przeanalizowane pozwoliły na wyciągnięcie wniosków dotyczących implementacji:

- Algorytm Hierocrypt, który opiera się na nieinwolucyjnym SPN trzeba będzie zrealizować w co najmniej dwóch układach rodziny FLEX10K. Jeden będzie przeznaczony na realizację funkcji szyfrowania, natomiast drugi będzie wykonywał operację deszyfrowania dla wersji algorytmu ze 128 bitowym blokiem danych oraz z wykorzystaniem 128 bitowego klucza.
- Bez względu na architekturę realizującą funkcję szyfrowania, ta wersja algorytmu wymaga wyznaczenia 6 podkluczy 256 bitowych oraz jednego 128 bitowego do operacji AK, która spełnia rolę „wybielenia końcowego” (ang. *post whitening*).

- Dwa ostatnie założenia mogą być sprzeczne ponieważ dostępne w układach ALTERY wbudowane szybkie pamięci mogą w znaczny sposób przyczynić się do poprawienia parametrów realizowanego układu, ale jednocześnie mogą przekreślić szansę na przenaszalność układu na inne układy. Istnieje więc potrzeba zbadania dwóch rodzajów rozwiązań.

4.3 Realizacja szyfrowania z wykorzystaniem algorytmu Hierocrypt.

Algorytmy startujące w ramach konkursu NESSIE, a także wcześniej w konkursie AES były oceniane także pod względem wyników implementacji w różnego typu architekturach w sensie rozwiązania logicznego.

Pierwsze rozwiązanie, określane dalej jako iteracyjne (ang. *loop*) polega na równoległym wyznaczaniu klucza rundy i realizację rundy danych. W pierwszym takcie obliczany będzie podklucz do rundy pierwszej i runda zerowa, w kolejnych taktach jednocześnie realizowane będą obliczenia podklucza dla rundy następnej i obliczenia rundy aktualnej. Rozwiązanie takie jest zalecane dla projektów w których czas wykonywania się rundy klucza i szyfrowania są zbliżone. Jest to rozwiązanie które ma jednak możliwość na realizację w wielu różnych wersjach. Sprawdzona zostanie szybkość implementacji, przy różnych długościach trwania rundy – jeden oraz dwa cykle przypadające na rundę.
Głównymi zaletami tego typu rozwiązania są:

- duża efektywność implementacji (nie wielka ilość zużytych komórek logicznych oraz bitów pamięci),
- bardzo dobry współczynnik, szybkości przetwarzania w stosunku do ilości elementów logicznych.

Głównymi wadami są

- rozwiązanie, które wymaga skomplikowanego sterowania,
- implementacja zdecydowanie najwolniejsza.

Kolejnym sposobem realizacji algorytmu szyfrowania w strukturach programowalnych jest realizacja w postaci funkcji kombinacyjnej (ang. *unrolled*). Polega ona na realizacji całego procesu szyfrowania w ciągu jednego taktu zegara. Rozwiązanie to ma dwie odmiany: pierwsza jest zastosowywana dla algorytmów w których runda generowania podklucza zajmuje mniej czasu niż długość trwania rundy, natomiast druga w przypadku odwrotnym. Przykładowym algorytmem, który można zaimplementować używając pierwszej

odmiany funkcji kombinacyjnej jest algorytm DES, w którym tak naprawdę generacja podkluczy nie ma iterowanego charakteru. Szyfry typu Rijndael, czy Hierocrypt – czyli takie w których całościowo algorytm generacji podkluczy stanowi „wąskie gardło" realizuje się w ten sposób, że podklucze poszczególnych rund dostarczane są do odpowiednich rejestrów przed wykonaniem jednotaktowej operacji szyfrowania. Rozwiązanie to nie będzie możliwe do realizacji w układach FLEX10K, ze względu na minimalną pojemność w stosunku do wymagań pamięciowych rozwiązania. W przypadku próby realizacji w wielu układach opóźnienia powstające na drodze komunikacji pomiędzy nimi są na tyle duże, że praktycznie nie możliwe będzie uzyskanie korzystnego wyniku szybkości. Najważniejsze korzyści wynikające ze stosowania tego typu architektury:

- sterowanie posiada zdecydowanie najmniej skomplikowany charakter,
- duża szybkość przetwarzania,

Dużo poważniejsze są jednak wady tego rozwiązania:

- wymaga ogromnej ilości komórek logicznych,
- posiada zdecydowanie najgorszy stosunek szybkości do zajętości układu,
- często wymaga oddzielnego generowania podkluczy (za pomocą software), gdy algorytm generowania podkluczy prezentuje zbyt dużą komplikację.

Ostatnim rozwiązaniem jest projekt potokowej (ang. *pipelined*) realizacji działania algorytmu, z jednoczesnym wyznaczaniem podkluczy rundy. Eliminowane są w ten sposób wady obu powyższych rozwiązań. Pozwala na realizację operacji szyfrowania w jednym takcie zegara (po napełnieniu potoku). Dedykowany procesor kryptograficzny potokowy zapewnia najszybsze działanie algorytmu, ale ma bardzo poważną wadę – komplikuje architekturę, utrudnia sterowanie oraz wymaga układu o dużej ilości komórek pamięci, co także powoduje dyskwalifikację tego rozwiązania na tego typu układach.

Ponieważ dwie ostatnie propozycje mają wymagania, których nie jest w stanie spełnić żaden z układów rodziny FLEX10K kontynuowane będą pracę nad najlepszą realizacją operacji szyfrowania w postaci funkcji iteracyjnej.

4.3.1 Szyfrowanie w wersji iteracyjnej.

Ponieważ powstało wiele różnych odmian tej architektury i nie powstawały one w sposób chronologiczny zdecydowałem się najpierw przedstawić wszystkie cechy wspólne rozwiązań które ma ją charakter iteracyjny.

Przede wszystkim układ SZYFROWANIE obsługuje wersję szyfru, która operuje na bloku danych o długości 128 bitów oraz 128 bitowym kluczem. Wynikiem takiego działania jest 128 bitowy szyfrogram zaszyfrowany przy użyciu algorytmu Hierocrypt. Został zrealizowany jako specjalizowany układ cyfrowy o wyjściach i wejściach przedstawionych na rysunku.

Rys.4.1: Wejścia i wyjścia układu szyfrowanie.

Wejścia układu:

- ZEGAR - wejście zegarowe,
- START_SETUP – wejście zezwolenia startu układu. Stan wysoki na tym wejściu powoduje ustawienie wewnętrznych elementów w stan początkowy. Rozpoczyna się proces przygotowania układu do stanu w którym może być rozpoczęte szyfrowanie właściwe. Ideę rozwiązania z wcześniejszymi operacjami przygotowawczymi, rozpoczętymi przed szyfrowaniem, wyjaśnię w dalszej części rozdziału. Podczas narastającego zbocza tego sygnału na wejściu KEY[127..0] powinien znajdować się sesyjny klucz. Po zakończeniu operacji ustawienia układu do stanu gotowości (linia wyjściowa GOTOWY w stanie wysokim) wszelkie sygnały na wejściu tej linii są ignorowane.
- RESET – jest to wejście, które powoduje całkowite przerwanie pracy układu. Na liniach wyjściowych GOTOWY i PRACA pojawia się stan niski. Układ jest przygotowany do wymiany klucza sesyjnego.
- START_SZ – wejście zezwolenia startu szyfrowania układu. Na narastającym zboczu zegara na wejściu IN[127..0] powinien znajdować się 128 bitowy blok danych do zaszyfrowania.

Przejście do stanu niskiego rozpoczyna proces szyfrowania oraz powoduje wysterowanie linii PRACA. Dodatkowe impulsy pojawiające się na tej linii są ignorowane gdy nie spełnione są dwa warunki, na wyjściu informacyjnym PRACA musi być stan niski, natomiast na wyjściu GOTOWY stan wysoki.

- IN[127..0] – 128 bitowa magistrala danych wejściowych, na nią podawany jest blok danych do zaszyfrowania.
- KEY[127..0] – 128 bitowa magistrala klucza, na wejście to podawany jest klucz sesji, za pomocą, którego będzie realizowana operacja szyfrowania.

Wyjścia z układu:

- GOTOWY – wyjście informacyjne układu. Wysoki stan tej linii informuje, że układ jest gotowy do pracy, mówi on, że klucz sesji jest już przygotowany. Dopiero po pojawieniu się jego, możliwe jest włączenie układu do pracy. Stan niski informuje, że w rejestrze odpowiedzialnym za przechowywanie klucza głównego znajduje się nieważny klucz.
- PRACA – wyjście informacyjne układu, mówiące o jego stanie. Wysoki stan informuje, że układ jest w stanie pracy, niski stan mówi, że na wyjściu magistrali OUT[127..0] znajduje się wynik operacji szyfrowania.
- OUT [127..0] – 128 bitowa magistrala danych wyjściowych, na jej liniach pojawia się wynik operacji szyfrowania Hierocryptem.

Oprogramowanie oraz sprzęt mające spełniać rolę utajniającą wymagają klucza sesyjnego, dzięki któremu mogą realizować swoją funkcję. Prawie zawsze odbywa się to w ten sposób, że podczas całej transmisji zaszyfrowanych danych używany jest tylko jeden klucz. Realizacja kolejnej transmisji wiąże się już z potrzebą wymiany klucza.

W algorytmie Hierocrypt, który niewątpliwie mógłby w przyszłości być wykorzystywany do zabezpieczenia poufności przepływających danych zarówno w rozwiązaniach programowych jak i sprzętowych, cechą która wpływa na konieczność zastosowania dodatkowych wcześniejszych przygotowań klucza jest występowanie operacji wybielenia klucza (ang. *pre - whitening*). Rozpoczęcie procesu ustawienia wstępnego klucza powoduje narastające zbocze na linii START_SETUP.

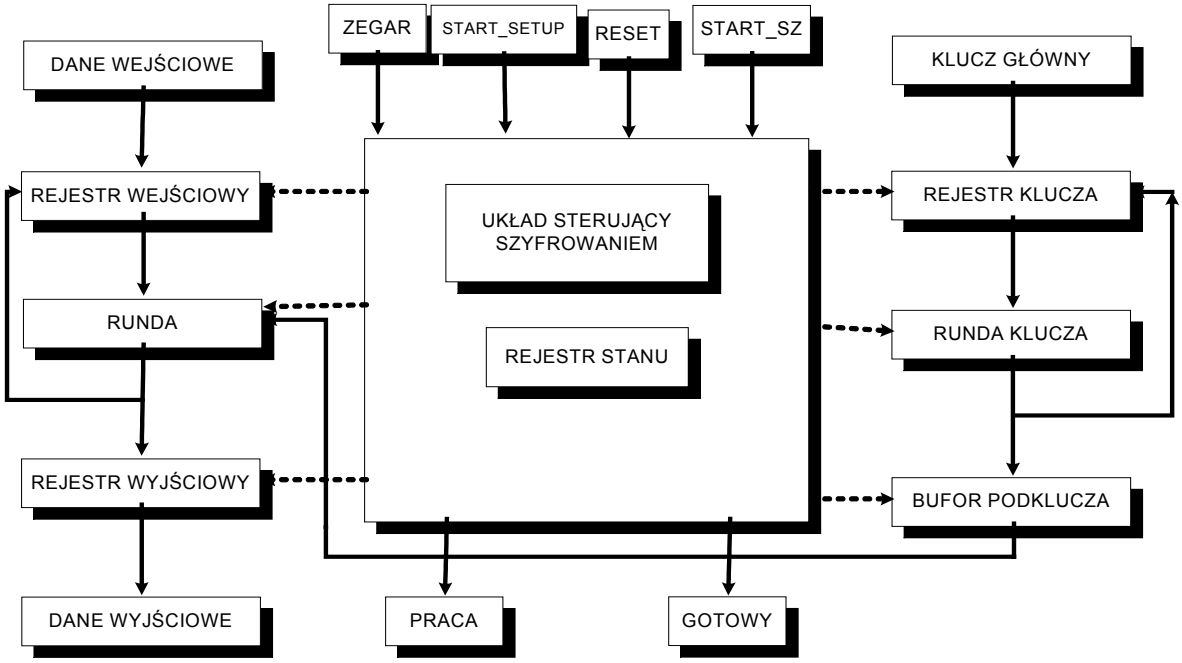
Rozwiązanie pierwsze. Po zakończeniu procesu ustawienia wstępnego w dwóch rejestrach przechowywana jest wartość klucza, który będzie używany już bezpośrednio do generacji odpowiednich podkluczy. Występowanie drugiego rejestru wynika, z potrzeby jednokrotnego w całej transmisji przeprowadzania procesu ustawiania klucza. Zapamiętuje on wartość klucza sesyjnego otrzymanego podczas ustawiania z dostarczonego z zewnątrz klucza głównego.

Drugi główny rejestr, będzie przetrzymywał natomiast wartości aktualnych kluczy przejściowych. Zakończenie tej fazy działania układu sygnalizowane jest przez wyjście informacyjne GOTOWY, które ustawione jest wówczas na stan wysoki.

Inny sposób na zrealizowanie układu, który będzie spełniał rolę urządzenia szyfrującego polega na wykorzystaniu symetrii algorytmu generacji podkluczy. Do wyznaczenia wszystkich podkluczy do poszczególnych rund wystarczy zapamiętać pięć podkluczy przejściowych oraz pięć dodatkowych wartości przejściowych z rundy generacji podkluczy. Realizacja ta wymaga wykorzystania 1600 bitów: 5 podkluczy 256 bitowych oraz 5 wartości 64 bitowych. Drugie rozwiązanie różni się więc przede wszystkim samym procesem wstępnego ustawienia, podczas którego wartości te zostają wygenerowane i zapamiętane. Zakończenie tej fazy działania układu jest sygnalizowane przez pojawienie się stanu wysokiego na wyjściu informacyjnym GOTOWY.

Dopóki na wyjściu tym nie pojawi się taka wartość stanu, nie możliwym jest uruchomienie operacji szyfrowania za pomocą linii START_SZ. Narastające zbocze na tej linii, przy jednoczesnym spełnieniu warunku na wysokość stanu na linii GOTOWY, powoduje uruchomienie układu oraz wysterowanie wyjścia informacyjnego PRACA. Dopóki na wyjściu tym będzie stan wysoki, dotąd każde narastające zbocze na linii START_SZ będzie ignorowane. Wykonanie operacji dodania ostatniego podklucza jest operacją, która zamyka proces szyfrowania algorytmem Hierocrypt, po niej następuje wysterowanie linii informacyjnej PRACA sygnałem w stanie niskim. Wraz z opadającym zboczem na tym wyjściu przepisywana jest do głównego rejestru klucza wartość klucza, która została zapamiętana po operacji ustawienia klucza. Dzięki temu układ jest w stanie takim, w jakim był po tej operacji. Układ jest gotowy do przyjęcia następnej 128 bitowej porcji danych.

Po zakończeniu sesji należy przywrócić układ do stanu sprzed użycia. Wraz ze stanem wysokim na linii wejściowej RESET czyszczona jest zawartość wszystkich rejestrów będących w użyciu podczas całego procesu szyfrowania.



Rys.4.2: Blok logiczny SZYFROWANIE został wykonany za pomocą edytora graficznego i nosi nazwę *szyfrowanie.gdf*

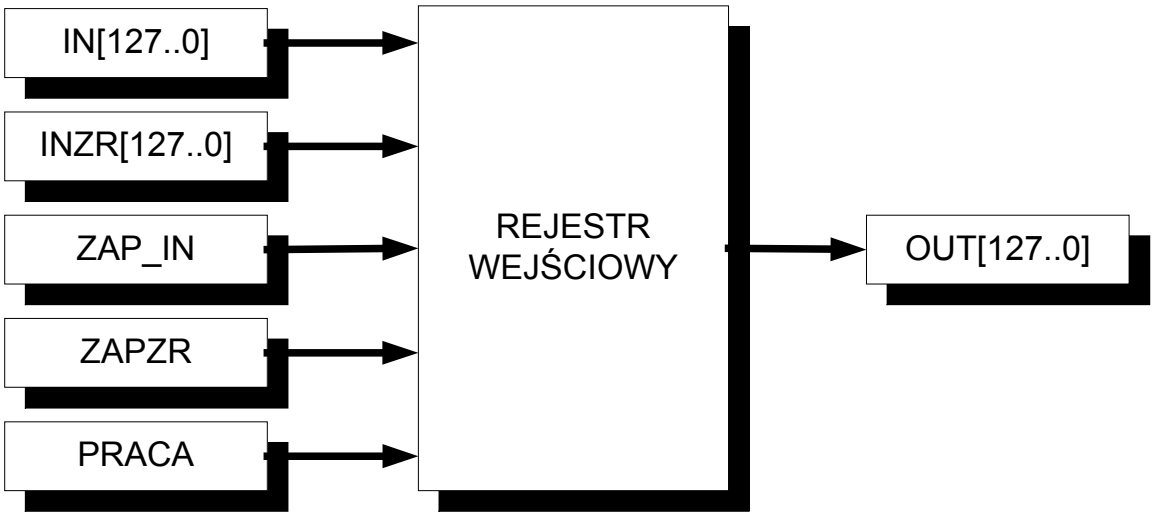
Ponieważ oba pomysły na realizacje funkcji szyfrowania w układzie programowalnym są w dużej mierze podobne do siebie dlatego też opisałem dokładnie pierwszy pomysł, natomiast w drugim wymieniłem wszystkie różnice .

4.3.1.1 Projekt z krótkim okresem ustawienia początkowego.

Układ szyfrowanie zrealizowany został w siedmiu blokach logicznych:

- rejestr wejściowy,
- runda szyfrowania,
- rejestr klucza przejściowego,
- runda klucza,
- rejestr podklucza rundy,
- rejestr wyjściowy,
- sterowanie,

4.3.1.1.1 Rejestr wejściowy - pełni rolę bufora danych wejściowych do każdej z rund szyfrowania. Schemat blokowy z zaznaczonymi wejściami i wyjściami bloczka logicznego przedstawia rysunek:



Rys.4.3: Schemat blokowy rejestru wejściowego.

Wejścia układu:

- IN[127..0] – 128 bitowe wejście dla danych wejściowych układu,
- INZR[127..0] – 128 bitowe wejście na, które wchodzi blok danych z wyjścia z układu runda szyfrowania,
- ZAP_IN – narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „niskim" na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali IN[127..0] do rejestru wewnętrznego bloczka.
- ZAPZR – narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim" na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali INZR[127..0] do rejestru wewnętrznego bloczka.
- PRACA – stan „niski" na tej linii wejściowej umożliwia zapis wartości wejściowej danych, stan „wysoki" umożliwia zapis wartości pojawiających się na magistrali wychodzącej z wyjścia układu rundy szyfru.

Wyjścia z układu:

- OUT[127..0] – 128 bitowe wyjście, na którym przechowywane są dane dla rundy szyfrowania podczas wykonywania się obliczeń rundy.

Układ rejestr wejściowy został zaimplementowany w języku AHDL i nosi nazwę *rejestr_wejściowy.tdf.*

4.3.1.1.2 Runda szyfrowania – realizuje operacje rundy szyfrowania algorytmem Hierocrypt, realizowana jest przez operacje: dodania podklucza, podstawienia oraz mnożenia przez macierz kodu MDS, wszystko do w postaci dwóch warstw.

Układ realizujący tą funkcję został tak zaprojektowany, że w zależności od sygnału sterującego OSTATNIA może wykonywać albo rundę podstawową, albo końcową.

Rys.4.4: Schemat blokowy operacji runda szyfrowania.

Wejścia układu:

- IN[127..0] – 128 bitowa magistrala z danymi wejściowymi do rundy. Na wejście podawany jest blok danych wyjściowych z rejestru wejściowego.
- KEY[255..0] – 256 bitowa magistrala z danymi podklucza do rundy. Na wejście podawany jest blok danych z rejestru podklucza rundy.
- OSTATNIA – wejście ustawiające tryb pracy układu: runda podstawowa/ końcowa. Stan „wysoki" determinuje wykonywanie się rundy podstawowej szyfru, natomiast „niski" rundy końcowej.

Wyjścia z układu:

- OUT[127..0] – 128 bitowa magistrala wyjściowa rundy szyfrowania.

Runda szyfrowania składa się z 3 bloków logicznych:

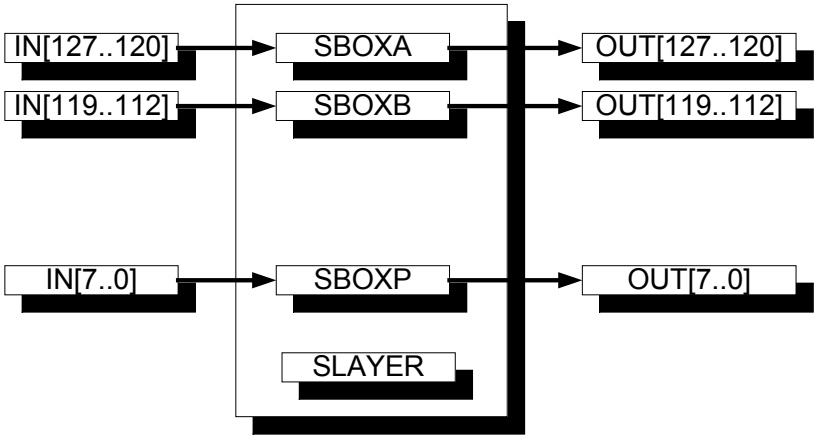
- skrzynek podstawieniowych.
- mnożenia przez macierz MDS lower level.
- mnożenia przez macierz MDS higher level.

Kolejność wykonywania poszczególnych operacji w rundzie szyfru obrazuje rysunek.

Rys.4.5: Schemat blokowy rundy szyfrowania

Układ rundy szyfrowania został zaprojektowany w języku AHDL, a plik w którym przechowywany jest jej projekt nazywa się *runda.tdf*.

Warstwa skrzynek podstawieniowych (ang. *sbox layer*) – układ, który realizuje operacje nieliniową w szyfrze składa się z równolegle działających 16 skrzynek podstawieniowych, które operują na ośmio bitowych blokach. Każdy bajt stanowi dane wejściowe do bloku logicznego sbox. Wyjście z układu jest wynikiem działania bloków sbox na przypisanych mu bajtach.



Rys.4.6: Schemat blokowy operacji równoległego podstawiania bajtów.

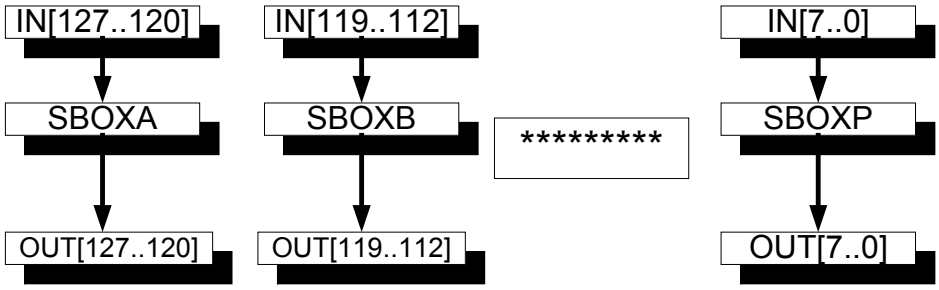
Wejścia układu:

- IN[127..0] – 128 bitowe wejście danych podzielone na 16 bloków po 8 bitów. Bloki te odpowiadają kolejnym grupom ośmiu bitów z magistrali danych wchodzących do operacji.

Wyjścia z układu:

- OUT[127..0] - 128 bitowe wyjście danych podzielone na 16 bloków po 8 bitów. Bloki te odpowiadają kolejnym grupom ośmiu bitów z magistrali danych wychodzącej z operacji.

Istota działania tej operacji przedstawiona jest na rysunku poniżej



Rys.4.7: Schemat funkcjonalny działania operacji równoległego podstawiania bajtów.

Układ ten został zaimplementowany w języku AHDL i nosi nazwę slayer.tdf.

Sbox jest blokiem logicznym o 8 wejściach i 8 wyjściach. Realizuje on trzy operacje, które zostały przedstawione w rozdziale 3.5.1. Są to operacja permutacji wewnątrz każdego bajta, mnożenia modularnego w ciele GF(2^8) oraz operacja afinicznego dodania stałej.

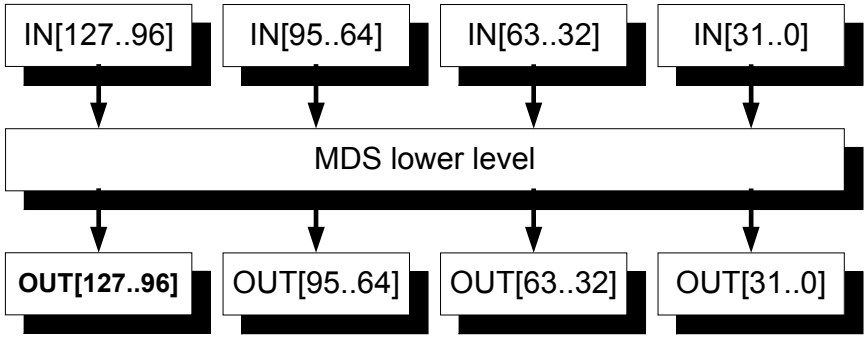
Układ sbox może być zaimplementowany na dwa sposoby:

- realizacja za pomocą tablicy prawdy o 8 wejściach i 8 wyjściach. Implementacja tą metodą zajmuje 253 komórki logiczne. Czas dostępu do tak zaimplementowanej pamięci wynosi ok. 19ns. Zaletą rozwiązania wykorzystującego komórki logiczne jest możliwość przenoszalności układu na inne układy spoza oferty firmy ALTERA.
- realizacja za pomocą wbudowanej specjalizowanej pamięci o szybkim czasie dostępu EAB. Emuluje ona pamięć typu ROM i w konfiguracji 256 x 8 świetnie nadaje się do wykorzystania w projekcie. Czas dostępu do tego typu pamięci wynosi 13ns. Jeden blok pamięci EAB będzie wykorzystywany przez jednego sboxa. W algorytmie Hierocrypt każda runda wymaga pracy 40 sboxów (2 x 16 skrzynek w funkcji rundy oraz 8 w operacji generacji podklucza), natomiast najwięcej EABów spośród układów rodziny FLEX10K posiada jej odmiana FLEX10KE – 24.

Założenie projektowe było takie, że liczyć się będzie przede wszystkim szybkość układu, natomiast badał będę również wyniki implementacji, które mają właściwość przenoszalności. Zdecydowałem się więc na realizację projektu iteracyjnego (typ jeden cykl – jedna runda), w którym wykorzystam wszystkie 24 EAB w ten sposób, że 8 z nich wykorzystam do skrzynek podstawieniowych algorytmu generowania podkluczy, natomiast pozostał 16 do jednej z warstw podstawieniowych z rundy szyfru. Drugi sposób będzie polegał na wykonaniu implementacji z wykorzystaniem tablic prawdy.

Warstwa mnożenia przez macierz MDS lower level – blok logiczny, którego idea działania jest opisana w rozdziale 3.5.2. 128 bitowe wejście jest dzielone na cztery bloki po 32 bity. Operacja polega na wymnożeniu przez macierz kodu MDS każdego 32 bitowego słowa. Operacja mnożenia macierzy wymaga dwóch operacji w ciele GF(2^8): modularnego mnożenia oraz dodawania. Układ składa się z czterech bloków mnożenia modulo wybrany wielomian nierozkładalny.

Zasada działania układu przedstawiona jest poniżej:



Rys.4.8: Schemat blokowy operacji MDS lower level.

Blok logiczny MDS lower level został zaimplementowany w języku AHDL i nosi nazwę *mdsl_layer.tdf*.

Wejścia układu:

- IN[127..0] – 128 bitowe wejście do układu, które dzielone jest na 4 bloki po 32 bity,

Wyjście z układu:

- OUT[127..0] – 128 bitowe wyjście z układu, na które składają się 4 wyjścia z funkcji MDS_L

Układ ten składa się z czterech układów MDS_L.

Układ MDS_L.

Wejścia do układu:

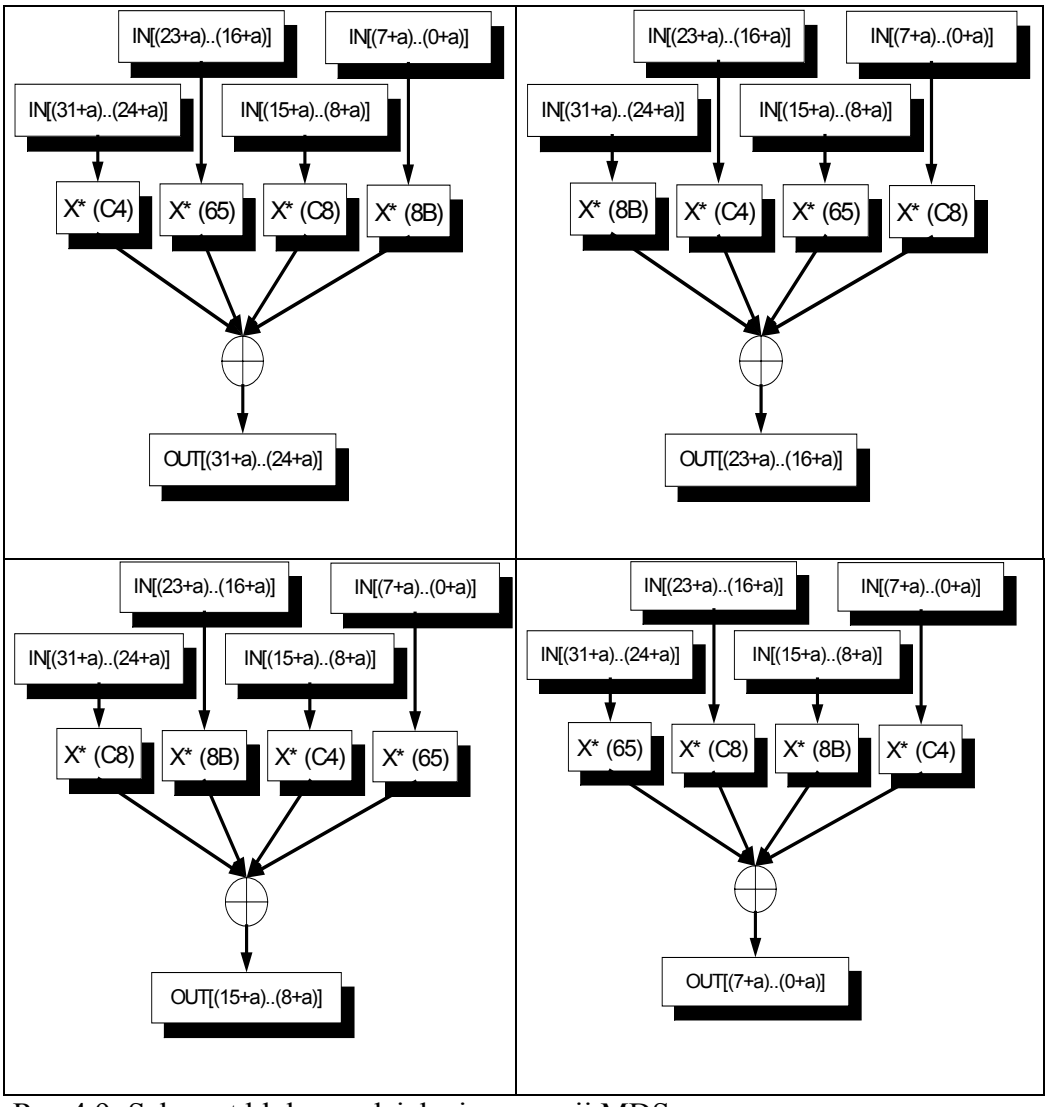
- IN[31..0] – 32 bitowa szyna wejściowa do układu. Słowo wchodzące do układu dzielone jest na 4 bloki bajtowe, na których wykonywana jest operacja mnożenia przez macierz kodu MDS.

Wyjścia z układu:

- OUT[31..0] – 32 bitowa szyna wyjściowa jest złożeniem 4 bajtowych bloków wyjściowych z operacji mnożenia macierzowego.

Układ został zaprojektowany w języku AHDL i ma nazwę mdsl.tdf.

Operację mnożenia macierzowego pokazuje rysunek poniżej:



Rys.4.9: Schemat blokowy dzialania operacji MDS_L .

Oznaczenie „a” na schemacie ma parametryzować zapis. Taka operacja wykonywana jest na wszystkich 4 32 bitowych słowach. „a” może więc przyjmować wartości a = {0, 32, 64, 92} dla poszczególnych 32 bitowych słów.

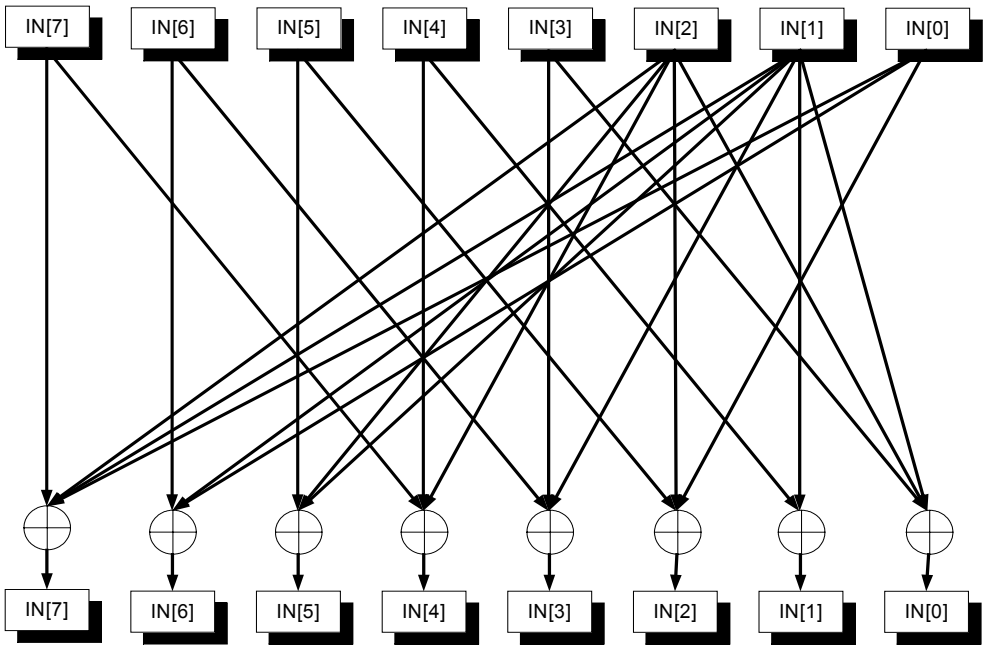
Oznaczenie „X * (wartość)” ma oznaczać mnożenie modularne w ciele GF(2^8) wektora wejściowego przez wybrany wektor składowy macierzy generujące kod MDS.

Układ ten składa się z 16 bloków logicznych:

- 4 x mnożenie przez C4,
- 4 x mnożenie przez 65,
- 4 x mnożenie przez C8,
- 4 x mnożenie przez 8B,

Mnożenie przez C4, mnożenie przez 65, mnożenie przez C8, mnożenie przez 8B – realizują operację mnożenia modularnego dowolnego wielomianu binarnego stopnia mniejszego niż 8 przez ustalony wielomian binarny stopnia mniejszego niż 8modulo nierozkładalny wielomian (rozdział 5.3.2). Każdy z tych układów ma ośmio bitowe wejście, które reprezentuje dowolny wielomian stopnia mniejszego niż 8 oraz ośmio bitowe wyjście, które reprezentuje wynik operacji mnożenia ustalonego wielomianu przez zadany wielomian wejściowy.

Schemat blokowy mnożenie przez C4 wyjaśnia skomplikowny charakter przekształcenia



Rys.4.10: Schemat blokowy mnożenia przez wektor binarny reprezentowany przez C4 szesnastkowo.

Ze względu na skomplikowany charakter schematów blokowych ograniczę się tylko do dodania, że mnożenie przez 65, mnożenie przez C8 i mnożenie przez 8B realizuje się w taki sam sposób.

Układy te zostały zaprojektowane w języku AHDL i noszą nazwę *mul_c4xa.tdf, mul_65xa.tdf, mul_c8xa.tdf, mul_8bxa.tdf.*

Warstwa mnożenie przez macierz MDS higher level.

Jest blokiem logicznym realizującym funkcję mieszania globalnego.

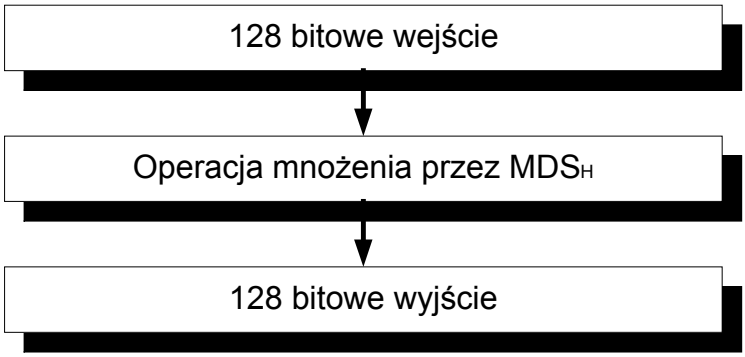
Wejścia układu:

- IN[127..0] – 128 bitowa magistrala danych wejściowych,

Wyjścia układu:

- OUT[127..0] – 128 bitowa magistrala danych, wejściowych które przeszły operację mnożenie przez binarną macierz MDS_H (rozdział 3.5.3)

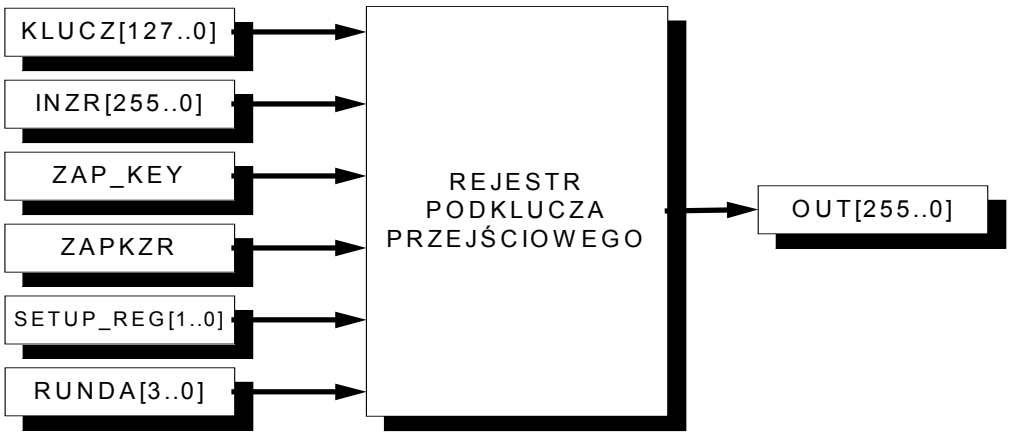
Schemat blokowy układu:



Rys.4.11: Schemat blokowy operacji MDS_H.

Układ został zaimplementowany w języku AHDL i nosi nazwę mdsh.tdf

4.3.1.1.3 Rejestr klucza przejściowego. – pełni rolę bufora danych wejściowych do każdej z rund klucza. Schemat blokowy z zaznaczonymi wejściami i wyjściami bloczka logicznego przedstawia rysunek:



Rys.4.12: Schemat blokowy układu rejestr podklucza przejściowego.

Wejścia układu:

- KLUCZ[127..0] – 128 bitowe wejście dla klucza głównego,
- INZR[255..0] – 256 bitowe wejście na, które wchodzi blok danych z wyjścia z układu rundy klucza,
- ZAP_KEY - narastające zbocze na tej linii wejściowej, powoduje zapisanie danych znajdujących się na magistrali KLUCZ[127..0] do rejestru wewnętrznego bloczka.
- ZAPKZR - narastające zbocze na tej linii wejściowej, powoduje zapisanie danych znajdujących się na magistrali INZR[127..0] do rejestru wewnętrznego bloczka.
- SETUP_REG[1..0] – szyna sterująca układu, określa numer stanu algorytmu przygotowującego klucz.
- RUNDA[3..0] – szyna sterująca układu. Numer rundy określa sposób zachowania się wyjścia z układu.

Wyjścia z układu:

- OUT[255..0] – 256 bitowe wyjście, które wystawia zawartość rejestru przechowującego wartość klucza przejściowego do rundy klucza.

Układ rejestr podklucza przejściowego został zaimplementowany w języku AHDL i nosi nazwę *im_rejestr.tdf.*

4.3.1.1.4 Runda klucza.

Układ runda klucza wylicza na podstawie wartości na wejściu RUNDA[3..0] oraz IN[255..0] wartość następnego klucza przejściowego oraz podklucza do następnej rundy.

Schemat blokowy układu wraz z zaznaczonymi wejściami i wyjściami układu znajduje się na rysunku poniżej:

Rys.4.13: Schemat blokowy rundy klucza.

Wejścia układu:

- IN[255..0] – 256 bitowe wejście dla podklucza przejściowego na podstawie, którego wyliczana będzie nowa wartość podklucza przejściowego i podklucza rundy.
- RUNDA[3..0] – 4 bitowa szyna sterująca. Wartość podawana na nią pochodzi z bloku sterującego i jest niezbędna do wybrania odpowiedniej stałej do rundy klucza.

Wyjścia z układu:

- OUT[255..0] – 256 bitowe wyjście z układu będące wynikiem operacji uaktualnienia podklucza przejściowego, bądź jej odwrotności. Na tej magistrali pojawia się więc wartość nowego podklucza przejściowego.
- KEY[255..0] – 256 bitowe wyjście z układu będące wynikiem operacji wyznaczenia podklucza rundy.

W zależności od wartości, podanej z układu STEROWNIE na szynę RUNDA[3..0] dobierana jest wartość stałej do rundy wg tabeli:

Numer rundy	Wartość stałej rundy G[31..0]
0	H"6ED9EBA15A827999"
1	H"CA62C1D65A827999"
2	H"8F1BBCDC6ED9EBA1"
3	H"6ED9EBA1CA62C1D6"
4	H"5A8279998F1BBCDC"
5	H"5A8279998F1BBCDC"
6	H"6ED9EBA1CA62C1D6"
7	H"8F1BBCDC6ED9EBA1"

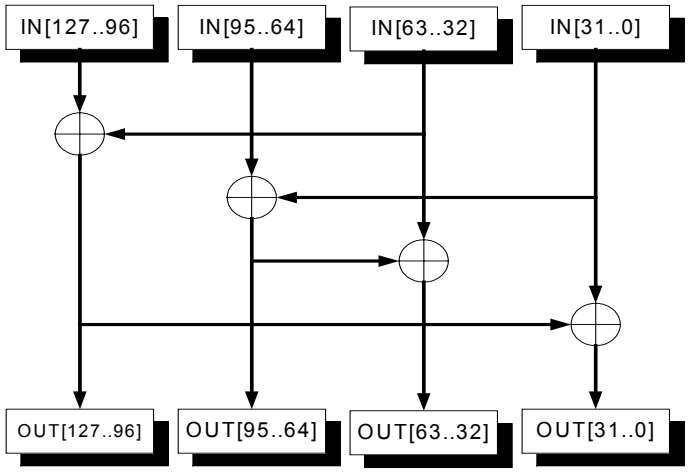
Układ logiczny rundy klucza został zaimplementowany w języku AHDL i nosi nazwę *key_round.tdf*

Runda klucza składa się z bloczków logicznych:

- $P^{(128)}$,
- $P^{-1(128)}$,
- M5e,
- Mb3,
- Funkcję σ.

Funkcja $P^{(128)}$. – układ logiczny reprezentujący tą transformację posiada magistralę wejściową 128 bitową oraz 128 bitową magistralę wyjściową. Spełnia on role funkcję operacji $P^{(128)}$ (rozdział 3.6.4) .

Schemat funkcjonalny pokazuje ideę działania operacji:



Rys.4.14: Schemat funkcjonalny działania operacji $P^{(128)}$.

Opis wejść i wyjść układu:

- IN[127..0] – 128 bitowa magistrala z danymi wejściowymi do operacji. Dane te dzielone są na bloki po 32 bity każdy i realizowana jest na nich funkcja wg schematu
- OUT[127..0] – 128 bitowa magistrala z danymi wyjściowymi z operacji.

Układ został zaimplementowany w AHDL i nosi nazwę *perm128.tdf.*

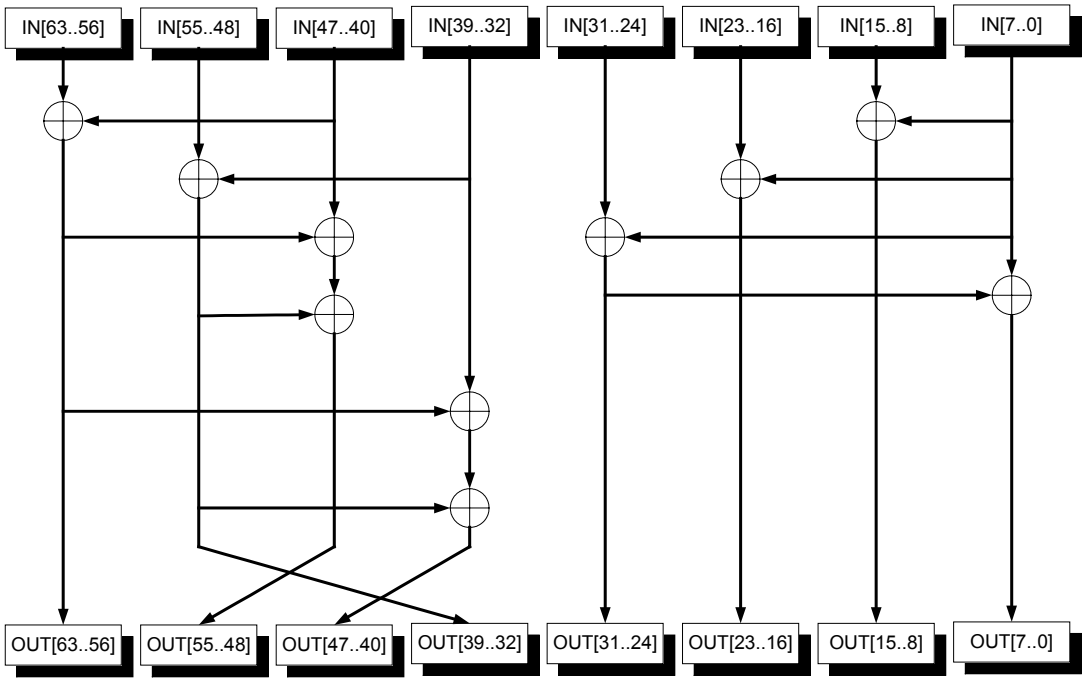
Funkcja $P^{-1(128)}$. – układ logiczny reprezentujący tą transformację posiada magistralę wejściową 128 bitową oraz 128 bitową magistralę wyjściową. Spełnia on role funkcję operacji $P^{-1(128)}$ (rozdział 3.6.5).

Działa w sposób odwrotny do układu $P^{(128)}$ i został zaimplementowany w AHDL.

Nosi nazwę *invperm128.tdf*

M5e – układ logiczny reprezentujący tą transformację posiada magistralę wejściową i wyjściową 64 bitową. Spełnia on rolę funkcji M5e (rozdział 3.6.2).

Schemat funkcjonalny przedstawia ideę działania układu:



Rys.4.15: Schemat funkcjonalny M5e.

Opis wejść i wyjść układu:

- IN[63..0] – 64 bitowa magistrala z danymi wejściowymi do operacji. Dane te dzielone są na bloki po 8 bitów każdy i realizowana jest na nich funkcja wg schematu
- OUT[63..0] – 64 bitowa magistrala z danymi wyjściowymi z operacji.

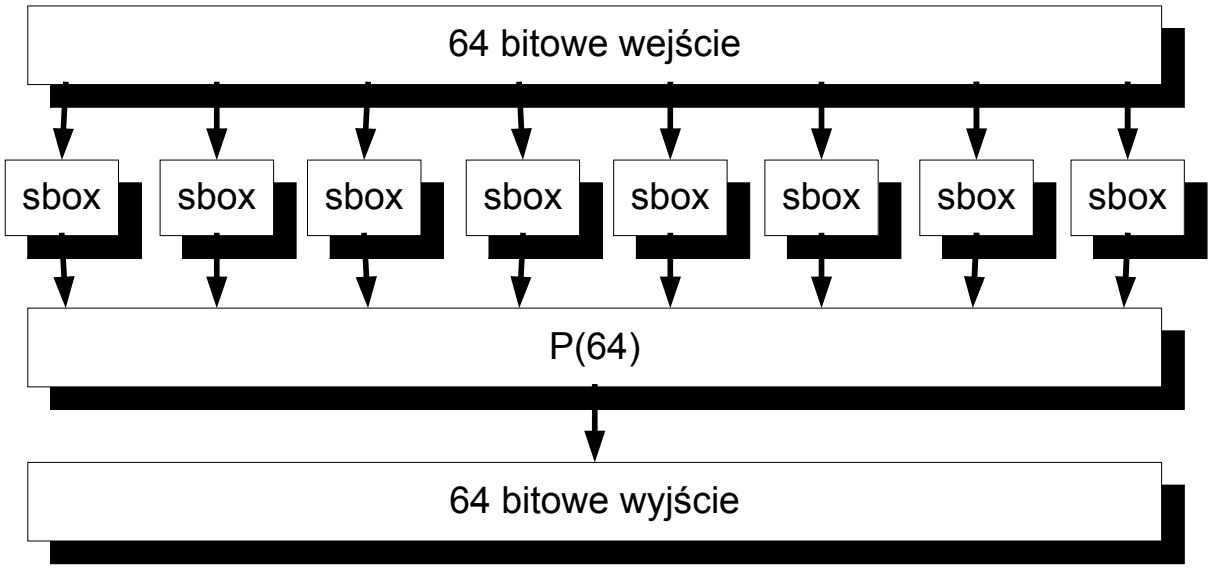
Układ został zaimplementowany w AHDL i nosi nazwę *m5e.tdf.*

Mb3 - układ logiczny reprezentujący tą transformację posiada magistralę wejściową i wyjściową 64 bitową. Spełnia on rolę funkcji Mb3 (rozdział 3.6.3).

Działa w sposób odwrotny do układu M5e i został zaimplementowany w AHDL. Nosi nazwę *mb3.tdf*

Funkcja σ – układ logiczny jest realizacją transformacji funkcji σ. 64 bitowe wejście jest dzielone na bloki 8 bitowe, które przechodzą operację podstawienia w skrzynkach podstawieniowych – sbox, a następnie na całej długości bloku realizowana jest operacja $P^{(64)}$.

Schemat blokowy funkcji.



Rys.4.16: Schemat blokowy funkcji σ.

Opis wejść i wyjść układu:

- IN[63..0] – 64 bitowa magistrala z danymi wejściowymi do operacji.
- OUT[63..0] – 64 bitowa magistrala z danymi wyjściowymi z operacji.

Układ został zaimplementowany w AHDL i nosi nazwę *funkcja_p.tdf.*

Składa się z układów logicznych:

- warstwy 8 równolegle działających sboxów,
- z funkcji $P^{(4n)}$. Dla n =16.

4.3.1.1.5 Rejestr podklucza rundy - pełni rolę bufora podklucza do każdej z rund. Schemat blokowy z zaznaczonymi wejściami i wyjściami bloczka logicznego przedstawia rysunek:

Rys.4.17 Schemat blokowy układu logicznego rejestr podklucza.

Opis wejść:

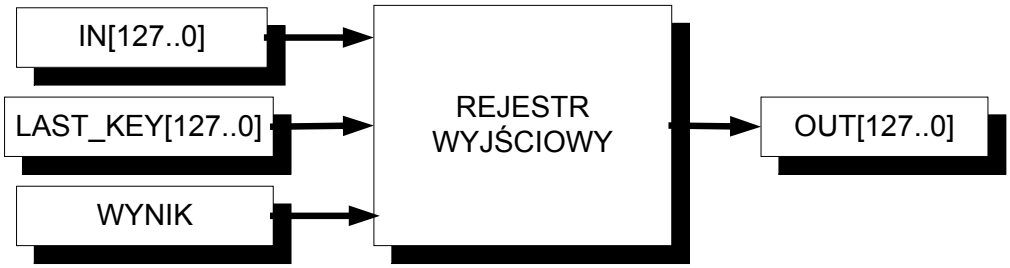
- IN[255..0] – magistrala wejściowa. Wychodzi z układu logicznego runda klucza, służy do wprowadzenia wartości podklucza do bufora, który jest odpowiedzialny za dostarczenie odpowiedniego podklucza do rundy.
- PRACA - stan „niski" na tej linii wejściowej uniemożliwia zapis wartości wejściowej danych – układ jest wyłączony, stan „wysoki" umożliwia zapis wartości pojawiających się na magistrali wychodzącej z wyjścia układu rundy klucza.
- ZAPKZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim" na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali IN[255..0] do rejestru wewnętrznego bloczka.

Opis wyjść:

- OUT[255..0] – magistrala wyjściowa wystawiająca wartość podklucza do rundy podstawowej oraz końcowej.
- LAST_KEY[127..0] – magistrala wyjściowa wystawiająca wartość ostatniego podklucza do operacji AK, zamykającej operacje szyfrowania.

Układ rejestr podklucza zaimplementowany został w języku AHDL i nosi nazwę *rejestr_podklucza.tdf*

4.3.1.1.6 Rejestr wyjściowy - pełni rolę bufora, do którego wrzucany jest wynik operacji szyfrowania. Schemat blokowy z zaznaczonymi wejściami i wyjściami bloczka logicznego przedstawia rysunek



Rys.4.18: Schemat blokowy układu logicznego rejestr wyjściowy.

Opis wejść:

- IN[127..0] – magistrala wejściowa. Służy do wprowadzenia wartości danych przetworzonych już po sześciu rundach do bufora, w którym realizowana jest ostatnia operacja AK.
- LAST_KEY[127..0] – magistrala wejściowa wystawiająca wartość ostatniego podklucza do operacji AK, zamykającej operacje szyfrowania.
- WYNIK - narastające zbocze na tej linii wejściowej, powoduje zapisanie danych powstałych w wyniku operacji xor na danych znajdujących się na magistrali IN[127..0] i LAST_KEY[127..0] do rejestru wewnętrznego, w którym przechowany jest wynik operacji szyfrowania.

Opis wyjść:

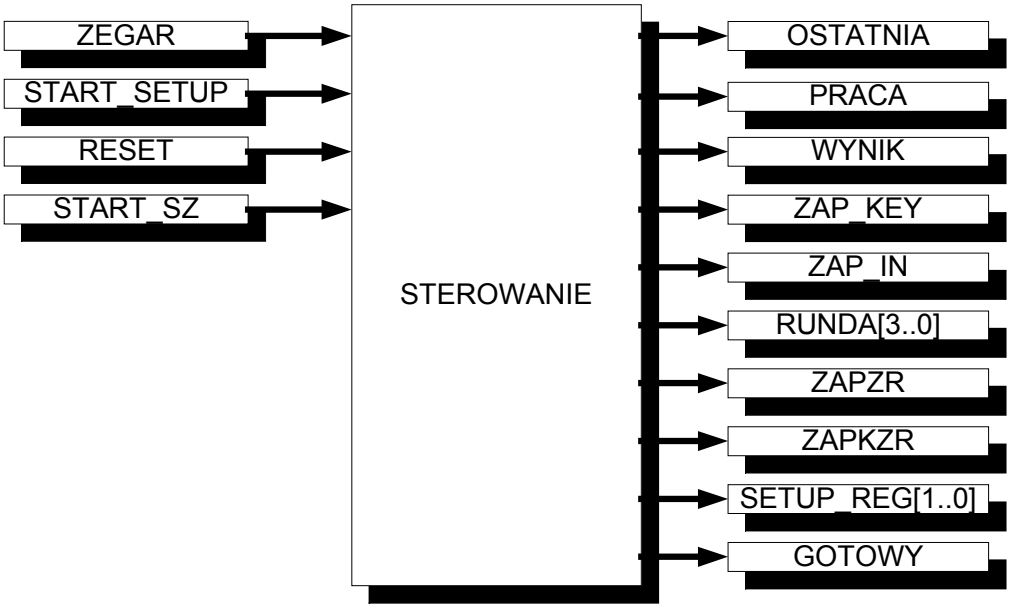
- OUT[127..0] – magistrala wyjściowa wystawiająca wartość wyniku na zewnątrz układu.

Układ rejestr wyjściowy zaimplementowany został w języku AHDL i nosi nazwę *rejestr_wyjściowy.tdf*

4.3.1.1.7 Sterowanie.

Blok logiczny STEROWANIE kieruje procesem szyfrowania. Wykonuje następujące zadania:

- generuje impulsy i sygnały, które zapewniają poprawną realizację operacji szyfrowania.
- nie pozwala na zapoczątkowanie kolejnego szyfrowania przed zakończeniem aktualnego,
- uniemożliwia także operację zmiany klucza głównego w trakcie operacji przygotowujący klucz,
- generuje sygnały PRACA i GOTOWY układu, które mówią w jakim aktualnie stanie znajduje się układ.
- zlicza wszystkie stany układu na podstawie, których generowane są odpowiednie sygnały i impulsy sterujące rundami szyfru i klucza.



Rys.4.19: Schemat blokowy układu logicznego STEROWANIE.

Wejścia układu:

- ZEGAR - wejście zegarowe,
- START_SETUP – wejście zezwolenia startu układu. Stan wysoki na tym wejściu powoduje ustawienie wewnętrznych elementów w stan początkowy. Rozpoczyna się proces przygotowania układu do stanu w którym może być rozpoczęte szyfrowanie właściwe. Podczas narastającego zbocza tego sygnału na wejściu KEY[127..0] powinien znajdować się sesyjny

klucz. Po zakończeniu operacji ustawienia układu do stanu gotowości (linia wyjściowa GOTOWY w stanie wysokim) wszelkie sygnały na wejściu tej linii są ignorowane.

- RESET – jest to wejście, które powoduje całkowite przerwanie pracy układu. Na liniach wyjściowych GOTOWY i PRACA pojawia się stan niski. Układ jest przygotowany do wymiany klucza sesyjnego.
- START_SZ – wejście zezwolenia startu szyfrowania układu. Na narastającym zboczu zegara na wejściu IN[127..0] powinien znajdować się 128 bitowy blok danych do zaszyfrowania. Przejście do stanu niskiego rozpoczyna proces szyfrowania oraz powoduje wysterowanie linii PRACA. Dodatkowe impulsy pojawiające się na tej linii są ignorowane gdy nie spełnione są dwa warunki, na wyjściu informacyjnym PRACA musi być stan niski, natomiast na wyjściu GOTOWY stan wysoki.

Wyjścia z układu :

- GOTOWY – wyjście informacyjne układu. Wysoki stan tej linii informuje, że układ jest gotowy do pracy, mówi on, że klucz sesji jest już przygotowany. Dopiero po pojawieniu się jego, możliwe jest włączenie układu do pracy. Stan niski informuje, że w rejestrze odpowiedzialnym za przechowywanie klucza głównego znajduje się nieważny klucz.
- PRACA – wyjście informacyjne układu, mówiące o jego stanie. Wysoki stan informuje, że układ jest w stanie pracy, niski stan mówi, że na wyjściu magistrali OUT[127..0] znajduje się wynik operacji szyfrowania.
- OSTATNIA – sygnał determinujący rodzaj wykonywanej rundy. Sygnał „wysoki” powoduje wykonanie rundy podstawowej szyfru, „niski” rundę końcową.
- WYNIK – impuls, którego narastające zbocze powoduje zapisanie wyniku szyfrowania do bufora wyjściowego projektu.
- ZAP_KEY – narastające zbocze tego sygnału, przy jednoczesnym stanie „niskim” na linii sterującej PRACA, powoduje zapisanie klucza głównego, do rejestru klucza przejściowego.
- ZAP_IN – narastające zbocze tego sygnału, przy jednoczesnym stanie „niskim” na linii sterującej PRACA, powoduje zapisanie danych wejściowych, do rejestru wejściowego.
- RUNDA[3..0] – 4 bitowa szyna wystawiająca numer rundy aktualnie wykonywanej.
- ZAPZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim” na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali wychodzącej z rundy szyfru do rejestru wewnętrznego bloczka logicznego odpowiedzialnego za zapis stanów danych szyfru.

- ZAPKZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim” na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali wychodzącej z rundy szyfru i wchodzącej do rejestru wewnętrznego bloczka logicznego odpowiedzialnego za zapisywanie podkluczy przejściowy (im_klucz), oraz podkluczy rundy (rejestr podklucza).
- SETUP_REG[1..0] – szyna sterująca układu, określa numer stanu algorytmu przygotowującego klucz.

Blok logiczny STEROWANIE został zaimplementowany w języku AHDL i nosi nazwę *sterowanie.tdf*

4.3.1.2 Projekt z długim okresem ustawienia początkowego.

Realizacja algorytmu szyfrowania z koncepcją dłuższego, wcześniejszego przygotowania układu polega na wyznaczeniu wszystkich niezbędnych danych do bezpośredniego wyznaczania podkluczy rund. Tymi niezbędnymi danymi są podklucze przejściowe z rund klucza: zerowej do czwartej oraz 64 bitowe wartości danych wychodzący z funkcji σ, opisanej w rozdziale 3.6.6. Po ich wyznaczeniu układ jest gotowy do pracy, a sygnalizowane to jest „wysokim” stanem na linii informacyjnej GOTOWY.

Wraz z pojawieniem się na wejściu START_SZ „narastającego” zbocza sygnału rozpoczyna się praca struktury, sygnalizowana stanem „wysokim” na wyjściu informacyjnym PRACA. Podczas pierwszego cyklu zegara na wejściu danych powinien znajdować się blok 128 bitowy do zaszyfrowania. Podczas kolejnych cykli realizowana są operacje jednoczesnego wykonania rundy szyfru oraz wygenerowania podklucza do rundy następnej. Inaczej wygląda cykl siódmy, podczas którego wykonuje się runda ostatnia i dodanie 128 bitowego podklucza. Połączenie tych dwóch operacji pozwala zaoszczędzić, jeden cykl, ale jest tylko możliwe do zrealizowania gdy wyznaczony zostanie wcześniej podklucz zamykający operację szyfrowania. Tak naprawdę jest on wyznaczany „w locie” bowiem nie ma nigdzie rejestru przetrzymującego jego wartość. Podklucz ten wyznaczany jest przez funkcje realizowaną przez układ GENERACJA_PODKLUCZA dla magistrali wyjściowej OUTG[127..0].

Układ szyfrowanie zrealizowany został w dziewięciu blokach logicznych:

- rejestr wejściowy,
- runda szyfrowania,
- rejestr klucza przejściowego,
- runda klucza,
- rejestr podklucza rundy,
- rejestr wyjściowy,
- sterowanie,
- pamięć podkluczy przejściowych,
- generacja podkluczy głównych rundy.

4.3.1.2.1 Rejestr wejściowy.

Układ logiczny został już opisany w rozdziale 4.2.1.1.1.

4.3.1.2.2 Runda szyfrowania.

Układ ten został już opisany w rozdziale 4.2.1.1.2.

4.3.1.2.3 Rejestr klucza przejściowego.

Układ ten został już opisany w rozdziale 4.2.1.1.3.

4.3.1.2.4 Runda klucza.

Układ ten został już opisany w rozdziale 4.2.1.1.4.

4.3.1.2.5 Rejestr podklucza rundy.

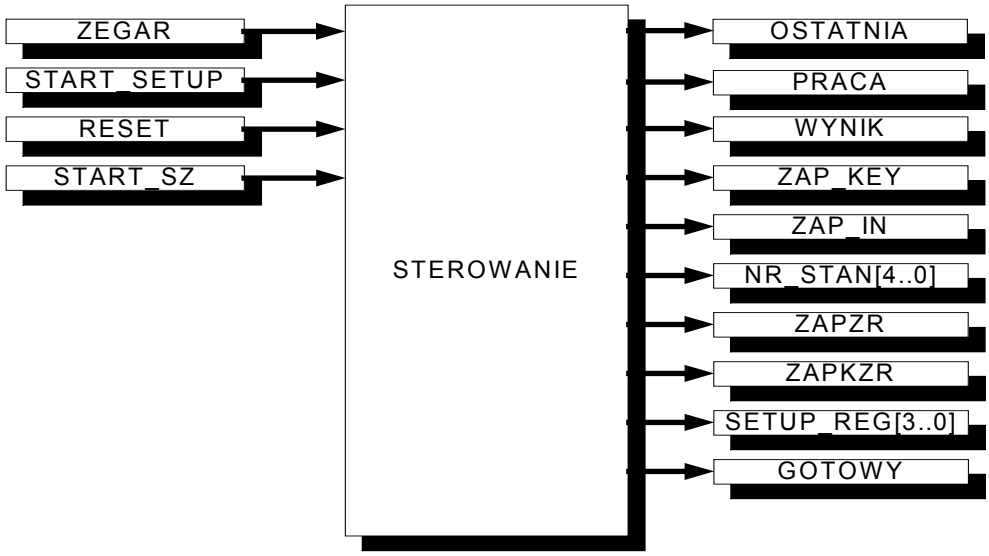
Układ ten został już opisany w rozdziale 4.2.1.1.5.

4.3.1.2.6 Rejestr wyjściowy.

Układ ten został już opisany w rozdziale 4.2.1.1.6.

4.3.1.2.7 Sterowanie.

Blok logiczny STEROWANIE kieruje procesem szyfrowania.



Rys.4.20: Schemat blokowy układu logicznego STEROWANIE.

Wejścia układu:

- W układzie STEROWANIE tego projektu nie zmieniło się nic jeżeli chodzi o sygnały wejściowe. Pełnią one takie same role. Wejścia układu zostały opisane w rozdziale 4.3.1.1.7

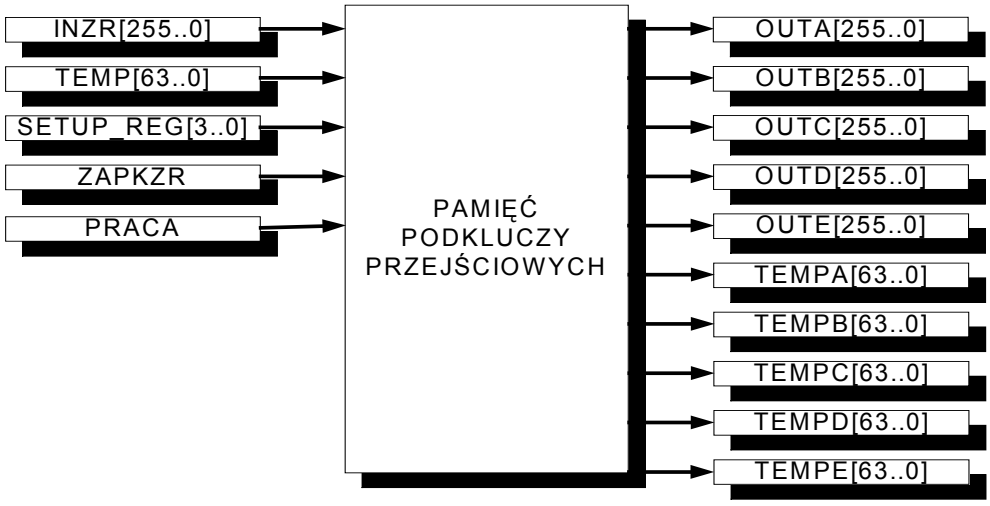
Wyjścia z układu :

- Poniżej opisane są tylko różnice pomiędzy wyjściami tego elementu projektu, a układem z opisanym w rozdziale 4.3.1.1.7
- NR_STAN[4..0] – 5 bitowa szyna wystawiająca numer aktualnego stanu wewnętrznego w układzie logicznym STEROWANIE. (w poprzedniej wersji wyjście to odpowiadało szynie RUNDA[3..0].
- ZAPKZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim” na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali wychodzącej z rundy klucza i wchodzącej do odpowiedniego rejestru wewnętrznego bloczka logicznego odpowiedzialnego za zapisywanie podkluczy przejściowych oraz dodatkowych danych potrzebnych do wygenerowania odpowiednich podkluczy rund (pamięć_imklucz).

Blok logiczny STEROWANIE został zaimplementowany w języku AHDL i nosi nazwę *sterowanie.tdf*

4.3.1.2.8 Pamięć podkluczy przejściowych.

Układ logiczny, który odpowiedzialny jest za przechowywanie w pamięci rejestrów wartości podkluczy przejściowych, które są niezbędne do wygenerowania odpowiednich podkluczy rund.



Rys.4.21: Schemat blokowy układu logicznego PAMIEC_IMKLUCZ.

Wejścia układu:

- INZR[255..0] - 256 bitowe wejście na, które wchodzi blok danych z wyjścia z układu rundy klucza. Dane z magistrali zapisywane są do jednego z rejestrów wewnętrznych odpowiedzialnych za przechowywanie podkluczy przejściowych.
- TEMP[63..0] – magistrala wejściowa, dzięki której do układu odpowiedzialnego za zapamiętywanie wartości przejściowych, które wykorzystywane są do generowania podkluczy rund (wartości wyjściowe z funkcji σ).
- SETUP_REG[3..0] - szyna sterująca układu, określa numer stanu algorytmu przygotowującego klucz.
- ZAPKZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „niskim” na linii wejściowej PRACA i GOTOWY, powoduje zapisanie danych znajdujących się na magistrali wychodzącej z rundy klucza i wchodzącej do rejestru wewnętrznego bloczka logicznego odpowiedzialnego za zapisywanie podkluczy przejściowych oraz dodatkowych 64-bitowych wartości (pamiec_imklucz). Na szynie tej sygnały są interpretowane tylko podczas algorytmu generacji podkluczy przejściowych, po ustawieniu stanu „wysokiego” na linii GOTOWY ,żaden impuls na linii ZAPKZR nie wpływa na zmianę stanu żadnego z rejestrów.

- GOTOWY – wejście blokujące możliwość zapisu do rejestrów wewnętrznych jakichkolwiek wartości, które zostały wygenerowane nie podczas procesu ustawiania układu.

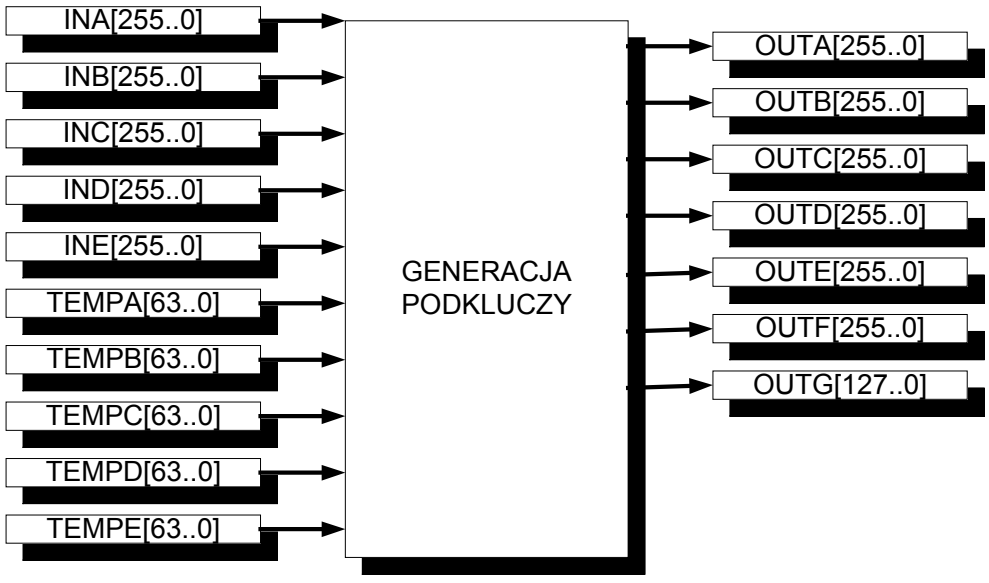
Wyjścia z układu:

- OUTA[255..0], OUTB[255..0], OUTC[255..0], OUTD[255..0], OUTE[255..0] – magistrale wystawiające wartości odpowiednich podkluczy do układu generującego poszczególne podklucze rund.
- TEMPA[63..0], TEMPB[63..0], TEMPC[63..0], TEMPD[63..0], TEMPE[63..0] – 64 bitowe magistrale wystawiające na zewnątrz układu wartości przejściowe niezbędne do prawidłowego wygenerowania podkluczy odpowiednich rund.

Blok logiczny PAMIEC_IMKLUCZ został zaprojektowany w języku AHDL i nosi nazwę *pamiec_imklucz.tdf*

4.3.1.2.9 Generacja podkluczy głównych rundy.

Układ logiczny odpowiadający za generację odpowiednich podkluczy rund.



Rys.4.22: Schemat blokowy układu logicznego GENERACJA_PODKLUCZY.

Wejścia układu:

- INA[255..0], INB[255..0], INC[255..0], IND[255..0], INE[255..0], TEMPA[63..0], TEMPB[63..0], TEMPC[63..0], TEMPD[63..0], TEMPE[63..0] – magistrale wejściowe, które na wystawiają zapamietane podczas operacji ustawienia układu, podklucze przejściowe, oraz niezbędne dane dodatkowe – dane wyjściowe z funkcji σ.

Wyjścia układu:

- OUTA[255..0], OUTB[255..0], OUTC[255..0], OUTD[255..0], OUTE[255..0], OUTF[255..0], OUTG[127..0] – magistrale wyjściowe, wystawiają podklucze do poszczególnych rund. 256-bitowe prowadzą podklucze bezpośrednio do rejestru podklucza, natomiast jedyna 128 bitowa magistrala prowadzi bezpośrednio do rejestru wynikowego, reprezentowanego przez REJESTR_WYJŚCIOWY.

Blok logiczny GENERACJA_PODKLUCZY został zaprojektowany w języku AHDL i nosi nazwę *generacja_podkluczy.tdf*.

4.3.2 Szyfrowanie w wersji kombinacyjnej i potokowej.

Realizacja różnych wersji architektury iteracyjnej spowodowała, że należało zweryfikować założenie możliwości realizacji funkcji kombinacyjnej i potokowej w strukturach programowalnych możliwych do zaprogramowania w systemie projektowym Max PLUS II. Z jednej strony ilość potrzebnych komórek pamięci, a z drugiej czas realizacji takiego przedsięwzięcia uniemożliwił możliwość zrealizowania tego typu architektur. Najbardziej pojemne układy rodziny FLEX10KE dostępne w systemie projektowym ledwo przekraczają 10 tys. komórek logicznych oraz 49kB pamięci specjalnego przeznaczenia. Realizacja tego typu architektur wymagałaby zdecydowanie więcej, prawdopodobnie około 60 tys. komórek logicznych. Istnieje możliwość zaimplementowania w ten sposób projektu, aby wykorzystywał on więcej niż jeden układ, ale takie rozwiązanie powoduje duże opóźnienia przesyłania sygnałów wewnątrz projektu co w efekcie nie daje znacznego przyśpieszenia realizacji.

4.4 Realizacja deszyfrowania z wykorzystaniem algorytmu Hierocrypt.

Realizacja algorytmu deszyfrowania, operacji odwrotnej do szyfrowania, polega na podaniu na wejście układu zaszyfrowanego tekstu. Na wyjściu tego układu otrzymujemy tekst jawny. Ponieważ Hierocrypt jest algorytmem symetrycznym do realizacji obu operacji potrzebny jest jeden klucz główny.

4.4.1 Deszyfrowanie w wersji iteracyjnej.

Przy realizacji deszyfrowania formie architektury iteracyjnej podobnie jak przy szyfrowaniu, dzięki symetrii algorytmu generacji podkluczy przejściowych, podklucze główne do rund generowane są na bieżąco. Ponieważ szyfr nie posiada właściwości inwolucyjności, w związku z tym wyznaczanie tekstu jawnego wiążę się z przeprowadzeniem odwrotnych operacji rund, w odwrotnej kolejności ich działania oraz podania w odwrotnej kolejności podkluczy do rund.

Układ DESZYFROWANIE służy do deszyfrowania 128 bitowego bloku danych wejściowych przy uzyciu 128 bitowego klucza. Wynikiem tego działania jest 128 bitowy blok tekstu jawnego. Zrealizowany układ spełniający funkcje deszyfrującą prezentuje schemat blokowy.

Rys.4.23: Schemat blokowy układu logicznego DESZYFROWANIE.

Wejścia układu:

- ZEGAR - wejście zegarowe,
- START_SETUP – wejście zezwolenia startu układu. Stan wysoki na tym wejściu powoduje ustawienie wewnętrznych elementów w stan początkowy. Rozpoczyna się proces przygotowania układu do stanu w którym może być rozpoczęte deszyfrowanie właściwe. Podczas narastającego zbocza tego sygnału na wejściu KEY[127..0] powinien znajdować się sesyjny klucz. Po zakończeniu operacji ustawienia układu do stanu gotowości (linia

wyjściowa GOTOWY w stanie wysokim) wszelkie sygnały na wejściu tej linii są ignorowane.

- RESET – jest to wejście, które powoduje całkowite przerwanie pracy układu. Na liniach wyjściowych GOTOWY i PRACA pojawia się stan niski. Układ jest przygotowany do wymiany klucza sesyjnego.
- START_SZ – wejście zezwolenia startu deszyfrowania układu. Na narastającym zboczu zegara na wejściu IN[127..0] powinien znajdować się 128 bitowy blok danych do odszyfrowania. Przejście do stanu niskiego rozpoczyna proces odszyfrowywania oraz powoduje wysterowanie linii PRACA. Dodatkowe impulsy pojawiające się na tej linii są ignorowane gdy nie spełnione są dwa warunki, na wyjściu informacyjnym PRACA musi być stan niski, natomiast na wyjściu GOTOWY stan wysoki.
- IN[127..0] – 128 bitowa magistrala danych wejściowych, na nią podawany jest blok danych do odszyfrowania.
- KEY[127..0] – 128 bitowa magistrala klucza, na wejście to podawany jest klucz sesji, za pomocą, którego będzie realizowana operacja deszyfrowania.

Wyjścia z układu:

- GOTOWY – wyjście informacyjne układu. Wysoki stan tej linii informuje, że układ jest gotowy do pracy, mówi on, że klucz sesji jest już przygotowany. Dopiero po pojawieniu się jego, możliwe jest włączenie układu do pracy. Stan niski informuje, że w rejestrze odpowiedzialnym za przechowywanie klucza głównego znajduje się nieważny klucz.
- PRACA – wyjście informacyjne układu, mówiące o jego stanie. Wysoki stan informuje, że układ jest w stanie pracy, niski stan mówi, że na wyjściu magistrali OUT[127..0] znajduje się wynik operacji deszyfrowania.
- OUT [127..0] – 128 bitowa magistrala danych wyjściowych, na jej liniach pojawia się wynik operacji deszyfrowania Hierocryptem.

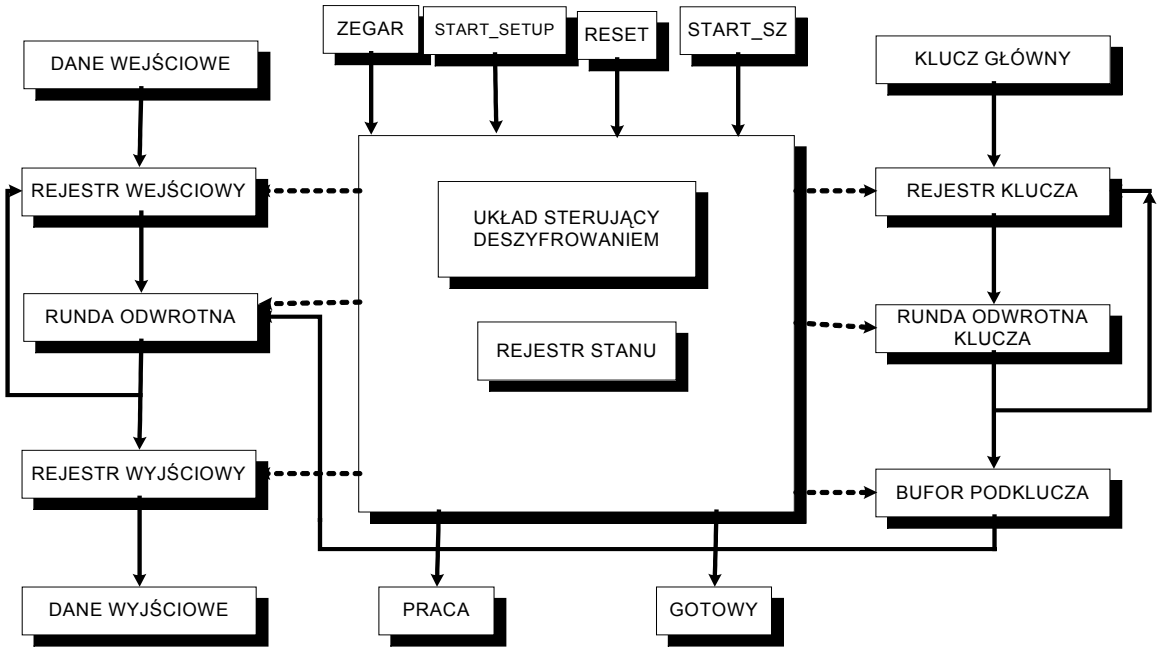
Szczególnie dwa rozwiązania układu realizującego deszyfrowanie, odpowiadające dwóm sposobom szyfrowania wydają się być dobrymi. Pierwsze polega na jednoczesnym realizowaniu operacji rundy, wyznaczania podklucza przejściowego i głównego do następnej rundy. Za pomocą narastającego zbocza na linii „START_SETUP" zostanie uruchomiony mechanizm wyznaczania podklucza przejściowego oraz związanego z nim podklucza do operacji odwrotnej AK (ang. *add key*), która wcześniej pełniła rolę zamykającej. Następnie realizowane już są jednoczesne operacje rundy deszyfrowania i generacji obu podkluczy do następnej rundy. Proces przygotowania układu

kończy się wraz w wygenerowaniem drugiego podklucza przejściowego. Wówczas to w dwóch rejestrach wejściowych układu zapisana jest jego wartość. W jednym z nich wartość będzie przechowywana aż do momentu zakończenia pracy z układem i do pojawienia się zbocza narastającego sygnału na linii RESET. Wartość ta będzie każdorazowo przepisywana do drugiego rejestru, po zakończeniu pracy nad szyfrowaniem bloku danych. Ten drugi rejestr zawsze przechowuje wartość aktualną klucza przejściowego.

Sposób drugi polega na wykorzystaniu dłuższego procesu przygotowawczego układu. Jest analogicznym do drugiego projektu realizacji operacji szyfrowania. Po pojawieniu się narastającego zbocza sygnału na linii sterującej, wejściowej START_SETUP, rozpoczyna się proces ustawiania układu. W czasie jego trwania podobnie jak w przypadku szyfrowania, wyznaczane są wszystkie niezbędne dane. Zakończenie tego wstępnego ustawienia układu sygnalizowane jest poprzez pojawienie się stanu „wysokiego" na linii informacyjnej układu GOTOWY.

Wraz z pojawieniem się na wejściu START_SZ „narastającego" zbocza sygnału rozpoczyna się praca struktury, sygnalizowana stanem „wysokim" na wyjściu informacyjnym PRACA. Podczas pierwszego cyklu zegara na wejściu danych powinien znajdować się blok 128 bitowy. Podczas kolejnych cykli realizowana są operacje jednoczesnego wykonania rundy odwrotnej szyfru oraz wygenerowania podklucza do rundy następnej.

Schemat blokowy operacji deszyfrowania obu wersji przedstawia się następująco:



Rys.4.24: Blok logiczny DESZYFROWANIE został wykonany za pomocą edytora graficznego i nosi nazwę *deszyfrowanie.gdf*

4.4.1.1 Projekt z krótkim procesem ustawienia początkowego.

Układ deszyfrowanie w tej wersji architektury iteracyjnej zrealizowany został w siedmiu blokach logicznych:

- rejestr wejściowy,
- runda deszyfrowania,
- rejestr klucza przejściowego,
- runda klucza,
- rejestr podklucza rundy,
- rejestr wyjściowy,
- sterowanie

4.4.1.1.1 Rejestr wejściowy.

Blok logiczny rejestr wejściowy działa analogicznie jak układ rejestru wejściowego dla operacji szyfrowania. Został on opisany w rozdziale 4.3.1.1.1.

4.4.1.1.2 Runda deszyfrowania

Realizuje operacje rundy deszyfrowania algorytmem Hierocrypt, realizowana jest przez operacje: dodania podklucza, podstawienia oraz mnożenia przez macierz kodu MDS, wszystko do w postaci dwóch warstw.

Układ realizujący tą funkcję został tak zaprojektowany, że w zależności od sygnału sterującego OSTATNIA może wykonywać albo rundę podstawową, albo końcową.

Rys.4.25: Schemat blokowy operacji runda deszyfrowania.

Wejścia i wyjścia układu logicznego maja analogiczną rolę jak w przypadku rundy szyfrowania, a sam układ został zaprojektowany w języku AHDL i nosi nazwę *runda.tdf*

Runda szyfrowania składa się z 3 bloków logicznych:

- odwrotnych skrzynek podstawieniowych.
- mnożenia przez macierz odwrotna do MDS lower level.
- mnożenia przez macierz odwrotna do MDS higher level.

Kolejność wykonywania poszczególnych operacji w rundzie szyfru obrazuje rysunek.

Rys.4.26: Schemat blokowy rundy deszyfrowania

Warstwa mnożenia przez odwrotną macierz kodu MDS higher level.

Operacja, która realizowana jest w analogiczny sposób do swojej odwrotności. Działanie jej opisane jest w rozdziale 4.3.1.1.3 w specyfikacji operacji MDS higher level.

Warstwa odwrotnych skrzynek podstawieniowych.

Operacja, która realizowana jest w analogiczny sposób do swojej odwrotności. Działanie jej opisane jest w rozdziale 4.3.1.1.3 w specyfikacji operacji podstawiania za pomocą skrzynek podstawieniowych.

Warstwa mnożenia przez odwrotną macierz kodu MDS lower level.

Operacja ta jest odwrotnością operacji mnożenia przez macierz kodu MDS lower level, której działanie opisane jest w rozdziale 4.3.1.1.3 w specyfikacji tej operacji. Działanie tego

elementu szyfru opisuje rozdział 3.5.2. Układ został zaimplementowany w języku AHDL i nosi nazwę *inv_mds_layer.tdf.*

Składa się on z czterech układów:

- mnożenia elementu z ciała GF(2^8) przez 34h
- mnożenia elementu z ciała GF(2^8) przez 82h
- mnożenia elementu z ciała GF(2^8) przez C4h
- mnożenia elementu z ciała GF(2^8) przez F6h

Pełna realizacja mnożenia przez macierz kodu MDS wymaga czterokrotnego wystąpienia każdego z układów. Ponieważ mnożenie odbywa się w ciele w związku z czym niezbędnym jest wybranie wielomianu charakterystycznego, którym dla algorytmu Hierocrypt jest wielomian
$x^8 + x^6 + x^5 + x + 1$.
Znajomość jednego z czynników mnożenia oraz znajomość właśnie tego wielomianu powoduje, że operacja ta jest bardzo łatwo implementowana w postaci układu równań.

Schematy działania poszczególnych układów przedstawiają się następująco:

Mnożenie przez 34h

OUT[7] = IN[5] $ IN[4] $ IN[3] $ IN[2];
OUT[6] = IN[7] $ IN[4] $ IN[3] $ IN[2] $ IN[1];
OUT[5] = IN[7] $ IN[6] $ IN[5] $ IN[4] $ IN[1] $ IN[0];
OUT[4] = IN[6] $ IN[2] $ IN[0];
OUT[3] = IN[5] $ IN[1];
OUT[2] = IN[4] $ IN[0];
OUT[1] = IN[7] $ IN[3];
OUT[0] = IN[6] $ IN[5] $ IN[4] $ IN[3];

Mnożenie przez 82h

OUT[7] = IN[6] $ IN[4] $ IN[3] $ IN[2] $ IN[0];
OUT[6] = IN[5] $ IN[3] $ IN[2] $ IN[1];
OUT[5] = IN[6] $ IN[3] $ IN[1];
OUT[4] = IN[6] $ IN[5] $ IN[4] $ IN[3];
OUT[3] = IN[5] $ IN[4] $ IN[3] $ IN[2];

OU [2] = IN[7] $ IN[4] $ IN[3] $ IN[2] $ IN[1];

OUT[1] = IN[7] $ IN[6] $ IN[3] $ IN[2] $ IN[1] $ IN[0];

OUT[0] = IN[7] $ IN[5] $ IN[4] $ IN[3] $ IN[1];

Mnożenie przez C4h

OUT[7] = IN[7] $ IN[2] $ IN[1] $ IN[0];

OUT[6] = IN[6] $ IN[1] $ IN[0];

OUT[5] = IN[5] $ IN[2] $ IN[1];

OUT[4] = IN[7] $ IN[4] $ IN[2];

OUT[3] = IN[6] $ IN[3] $ IN[1];

OUT[2] = IN[5] $ IN[2] $ IN[0];

OUT[1] = IN[4] $ IN[1];

OUT[0] = IN[3] $ IN[2] $ IN[1];

Mnożenie przez F6h

OUT[7] = IN[6] $ IN[4] $ IN[3] $ IN[1] $ IN[0];

OUT[6] = IN[7] $ IN[5] $ IN[3] $ IN[2] $ IN[0];

OUT[5] = IN[3] $ IN[2] $ IN[0];

OUT[4] = IN[7] $ IN[6] $ IN[4] $ IN[3] $ IN[2] $ IN[0];

OUT[3] = IN[6] $ IN[5] $ IN[3] $ IN[2] $ IN[1];

OUT[2] = IN[7] $ IN[5] $ IN[4] $ IN[2] $ IN[1] $ IN[0];

OUT[1] = IN[7] $ IN[6] $ IN[4] $ IN[3] $ IN[1] $ IN[0];

OUT[0] = IN[7] $ IN[5] $ IN[4] $ IN[2] $ IN[1];

Gdzie IN[i] – jest i-tym bitem bajtu wejściowego, a OUT[j] – jest j –tym bitem bajtu wyjściowego.

Wszystkie te układy zostały zaimplementowane w języku AHDL i noszą nazwy: *mul_34xa.tdf, mul_82xa.tdf, mul_c4xa.tdf, mul_f6xa.tdf,*

4.4.1.1.3 Rejestr klucza przejściowego.

Blok logiczny rejestr klucza przejściowego działa analogicznie jak układ rejestr klucza przejściowego operacji szyfrowania. Został on opisany w rozdziale 4.3.1.1.3.

4.4.1.1.4 Runda klucza.

Blok logiczny runda klucza działa analogicznie jak układ runda klucza operacji szyfrowania. Został on opisany w rozdziale 4.3.1.1.4.

4.4.1.1.5 Rejestr podklucza rundy.

Blok logiczny rejestr podklucza rundy działa analogicznie jak układ rejestr poklucza rundy operacji szyfrowania. Został on opisany w rozdziale 4.3.1.1.5.

4.4.1.1.6 Rejestr wyjściowy.

Blok logiczny rejestr wyjściowy działa analogicznie jak układ rejestr wyjściowy operacji szyfrowania. Został on opisany w rozdziale 4.3.1.1.6.

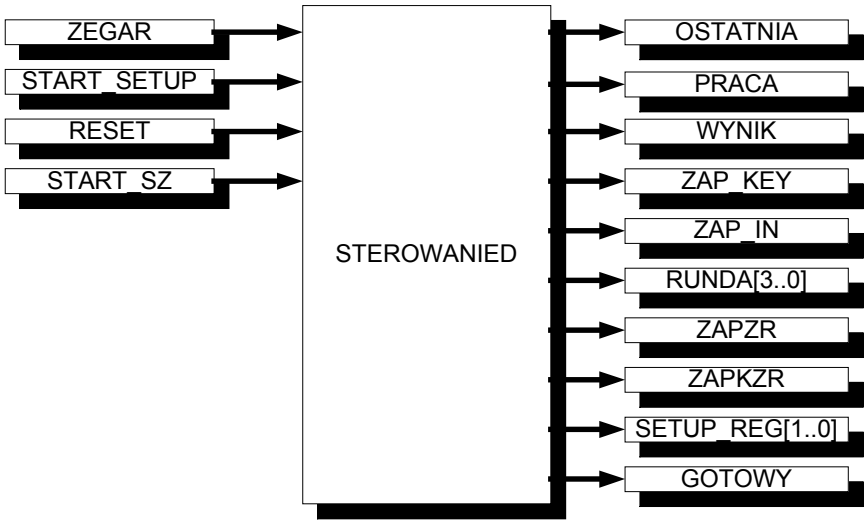
4.4.1.1.7 Sterowanie.

Układ ten nosi nazwę STEROWANIED i determinuje sposób działania całego układu deszyfrującego. Za pomocą sygnałów wychodzących z linii wyjściowych kieruje operacją poprawnego wykonania operacji deszyfrowania.

Realizuje on następujące zadania:

- generuje impulsy i sygnały, które zapewniają poprawną realizację deszyfrowania.
- uniemożliwia rozpoczęcie kolejnego procesu deszyfrowania przed zakończeniem poprzedniego,
- za pomocą linii informacyjnych GOTOWY i PRACA określa stan układu,
- zlicza wewnętrzne stany układu i na ich podstawie generuje odpowiednie impulsy i sygnały.

Schemat blokowy układu STEROWANIED:



Rys.4.27: Schemat blokowy układu logicznego STEROWANIED.

Wejścia układu:

- ZEGAR - wejście zegarowe,
- START_SETUP – wejście zezwolenia startu układu. Stan wysoki na tym wejściu powoduje ustawienie wewnętrznych elementów w stan początkowy. Rozpoczyna się proces przygotowania układu do stanu w którym może być rozpoczęte deszyfrowanie właściwe. Podczas narastającego zbocza tego sygnału na wejściu KEY[127..0] powinien znajdować się sesyjny klucz. Po zakończeniu operacji ustawienia układu do stanu gotowości (linia wyjściowa GOTOWY w stanie wysokim) wszelkie sygnały na wejściu tej linii są ignorowane.
- RESET – jest to wejście, które powoduje całkowite przerwanie pracy układu. Na liniach wyjściowych GOTOWY i PRACA pojawia się stan niski. Układ jest przygotowany do wymiany klucza sesyjnego.
- START_SZ – wejście zezwolenia startu deszyfrowania układu. Na narastającym zboczu zegara na wejściu IN[127..0] powinien znajdować się 128 bitowy blok danych do odszyfrowania. Przejście do stanu niskiego rozpoczyna proces deszyfrowania oraz powoduje wysterowanie linii PRACA. Dodatkowe impulsy pojawiające się na tej linii są ignorowane gdy nie spełnione są dwa warunki, na wyjściu informacyjnym PRACA musi być stan „niski”, natomiast na wyjściu GOTOWY stan „wysoki”.

Wyjścia z układu :

- GOTOWY – wyjście informacyjne układu. Wysoki stan tej linii informuje, że układ jest gotowy do pracy, mówi on, że klucz sesji jest już przygotowany. Dopiero po pojawieniu się jego, możliwe jest włączenie układu do pracy. Stan niski informuje, że w rejestrze odpowiedzialnym za przechowywanie klucza głównego znajduje się nieważny klucz.
- PRACA – wyjście informacyjne układu, mówiące o jego stanie. Wysoki stan informuje, że układ jest w stanie pracy, niski stan mówi, że na wyjściu magistrali OUT[127..0] znajduje się wynik operacji szyfrowania.
- OSTATNIA – sygnał determinujący rodzaj wykonywanej rundy. Sygnał „wysoki" powoduje wykonanie rundy podstawowej szyfru, „niski" rundę końcową.
- WYNIK – impuls, którego narastające zbocze powoduje zapisanie wyniku deszyfrowania do bufora wyjściowego projektu.
- ZAP_KEY – narastające zbocze tego sygnału, przy jednoczesnym stanie „niskim" na linii sterującej PRACA, powoduje zapisanie klucza głównego, do rejestru klucza przejściowego.
- ZAP_IN – narastające zbocze tego sygnału, przy jednoczesnym stanie „niskim" na linii sterującej PRACA, powoduje zapisanie danych wejściowych, do rejestru wejściowego.
- RUNDA[3..0] – 4 bitowa szyna wystawiająca numer rundy aktualnie wykonywanej.
- ZAPZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim" na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali wychodzącej z rundy szyfru do rejestru wewnętrznego bloczka logicznego odpowiedzialnego za zapis stanów danych szyfru.
- ZAPKZR - narastające zbocze na tej linii wejściowej, przy jednoczesnym stanie „wysokim" na linii wejściowej PRACA, powoduje zapisanie danych znajdujących się na magistrali wychodzącej z rundy szyfru i wchodzącej do rejestru wewnętrznego bloczka logicznego odpowiedzialnego za zapisywanie podkluczy przejściowy (im_klucz), oraz podkluczy rundy (rejestr podklucza).

SETUP_REG[1..0] – szyna sterująca układu, określa numer stanu algorytmu przygotowującego klucz.

STEROWANIED zostało zaimplementowane w języku AHDL i nosi nazwę *sterowanied.tdf.*

4.4.1.2 Projekt z długim procesem ustawienia początkowego.

Realizacja algorytmu deszyfrowania z koncepcją dłuższego, wcześniejszego przygotowania układu nie różni się mocno od realizacji operacji szyfrowania w tej architekturze. Po zakończeniu wstępnego przygotowania układu na wyjściu informacyjnym GOTOWY pojawia się stan „wysoki". Układ jest w stanie rozpocząć pracę polegającą na deszyfrowaniu danych.

Wraz z pojawieniem się na wejściu START_SZ „narastającego" zbocza sygnału rozpoczyna się praca struktury, sygnalizowana stanem „wysokim" na wyjściu informacyjnym PRACA. Stan ten utrzymuje się tak długo jak tylko trwa operacja deszyfrowania bloku danych.

Układ DESZYFROWANIE zrealizowany został w dziewięciu blokach logicznych:

- rejestr wejściowy,
- runda szyfrowania,
- rejestr klucza przejściowego,
- runda klucza,
- rejestr podklucza rundy,
- rejestr wyjściowy,
- sterowanie,
- pamięć podkluczy przejściowych,
- generacja podkluczy głównych rundy.

4.4.1.2.1 Rejestr wejściowy.

Układ logiczny został już opisany w rozdziale 4.2.1.1.1.

4.4.1.2.2 Runda deszyfrowania.

Układ ten został już opisany w rozdziale 4.4.1.1.2.

4.4.1.2.3 Rejestr klucza przejściowego.

Układ ten został już opisany w rozdziale 4.2.1.1.3

4.4.1.2.4 Runda klucza.

Układ ten został już opisany w rozdziale 4.2.1.1.4.

4.4.1.2.5 Rejestr podklucza rundy.

Układ ten został już opisany w rozdziale 4.2.1.1.5.

4.4.1.2.6 Rejestr wyjściowy.

Układ ten został już opisany w rozdziale 4.2.1.1.6.

4.4.1.2.7 Sterowanie.

Układ ten działa analogicznie jak opisany w rozdziale 4.4.1.1.7.

4.4.1.2.8 Pamięć podkluczy przejściowych.

Układ ten został już opisany w rozdziale 4.3.1.2.8.

4.4.1.2.9 Generacja podkluczy głównych rundy.

Układ ten został już opisany w rozdziale 4.3.1.2.9.

4.4.2 Deszyfrowanie w wersji kombinacyjnej i potokowej.

W rozdziale 4.3.2 znajduje się wyjaśnienie dlaczego tego typu realizacje operacji szyfrowania w dostępnych strukturach programowalnych nie były możliwe do realizacji.

V. Analiza otrzymanych wyników implementacji.

Ostatnim etapem projektowania układu logicznego spełniającego rolę dedykowanego procesora kryptograficznego w strukturach programowalnych jest jego weryfikacja oraz fizyczne programowanie. Niestety na tym etapie pracy dyplomowej będę musiał się ograniczyć do przeprowadzenia weryfikacji symulacyjnej zrealizowanych projektów. Taki sposób badania determinuje użycie analizatora przebiegów czasowych (Waveform editor) oraz Symulatora. Narzędzia te pozwalają na sprawdzenie poprawności funkcjonalnej, a także parametrów czasowych zrealizowanego układu. Mechanizm Symulatora przetwarza parametry wejściowe i na ich podstawie tworzy odpowiednie informacje wyjściowe. Opis działania układu jest przedstawiany za pomocą przebiegów czasowych.

Decydującymi faktami determinującymi wybór sposobu realizacji układu cyfrowego były przyjęte założenia projektowe, działanie algorytmu oraz specyfika dostępnych struktur programowalnych. Najważniejszym założeniem było osiągnięcie jak najlepszych wyników jeśli chodzi o szybkość działania wytworzonego układu. Z tym faktem wiąże się także konieczność realizacji całej pojedynczej funkcji algorytmu: szyfrowania bądź deszyfrowania w co najwyżej jednej strukturze programowalnej. Wskazanym też było sprawdzenie jakie wyniki osiągają implementacje alternatywne, wolniej działające, ale niosące za sobą właściwość przenoszalności, która determinowana jest sposobem realizacji skrzynek podstawieniowych. W rozdziale 4.3.1.1.2 w akapicie dotyczącym właśnie warstwy nieliniowej zaproponowałem dwa sposoby realizacji. Pierwszy polegał na wykorzystaniu wbudowanych bloków pamięci w konfiguracji 256x8, natomiast drugi wykorzystywał do realizacji sboxów tablice prawdy.
Niestety realizacja skrzynek podstawieniowych w postaci tablic prawdy jest bardzo mało efektywna implementacyjnie – jedna skrzynka zajmuje aż 330 komórek logicznych, a po zastosowaniu dekompozycji funkcjonalnej 253. Nie ma żadnego układu w rodzinie FLEX10K, który pomieścił by w całej strukturze realizowany projekt. Dlatego też ten sposób realizacji architektury iteracyjnej został przeze mnie potraktowany w sposób marginalny.

W algorytmie Hierocrypt w jednej rundzie szyfrowania wykorzystywanych jest 32 skrzynki, a w rundzie klucza dodatkowo 8. Zastosowane przeze mnie układy FLEX10KE posiadają największą ilość wbudowanych matryc pamięci – 24.

Podczas implementacji oraz w czasie symulacji realizowanego projektu okazało się, ze operacją, która najdłużej trwa w algorytmie była runda klucza, dlatego też zdecydowałem się zrealizować 8 sboxów, wchodzących w jej skład, w EABach. Pozostałe 16 zastosowałem do realizacji jednej warstwy nieliniowej w rundzie szyfru.

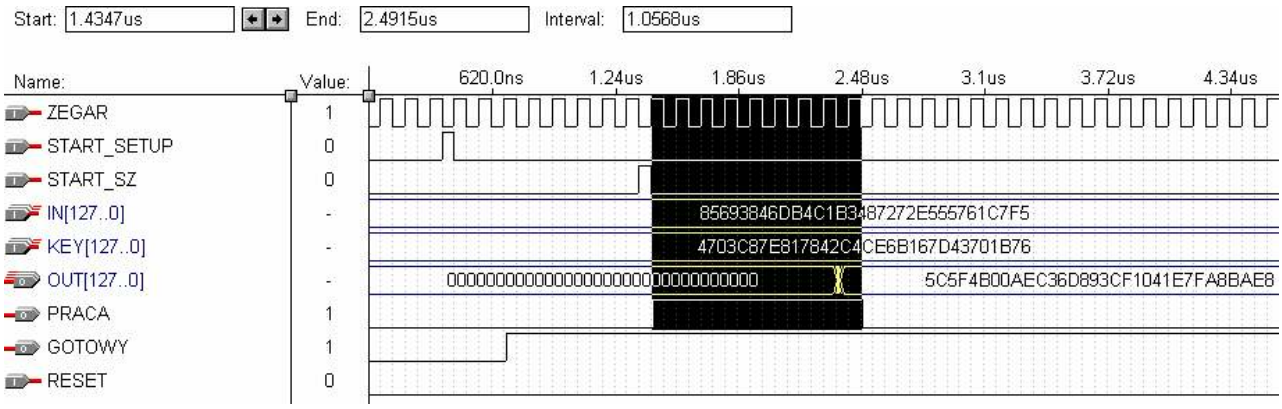
Opis różnic działania projektu z długim i krótkim czasem ustawienia początkowego przedstawiłem w rozdziale 4.3.1 dotyczącym opisu sposobu implementacji architektury iteracyjnej. Dodam tylko, że zrealizowałem dodatkowo dwa rodzaje algorytmów przygotowania wstępnego układu: pierwszy, w którym runda klucza przejściowego trwa 1 cykl zegara oraz drugi z 3 cyklową runda klucza. O tym wszystkim mówi ten rozdział.

5.1 Poprawność funkcjonalna układu.

Poprawność funkcjonalna układu mogła być przeprowadzona za pomocą danych testowych, które zostały dostarczone wraz ze specyfikacją szyfru do konkursu NESSIE. Sprawdzenie jej polega na porównaniu wyników szyfrowania oraz deszyfrowania otrzymanego układu z wynikami, które są wektorami testowymi. Przeprowadziłem symulację dla każdego z układów, które otrzymałem w wyniku syntezy logicznej.

Układy wykorzystujące wbudowane matryce pamięci.

- **wersja z krótkim okresem przygotowania początkowego.**



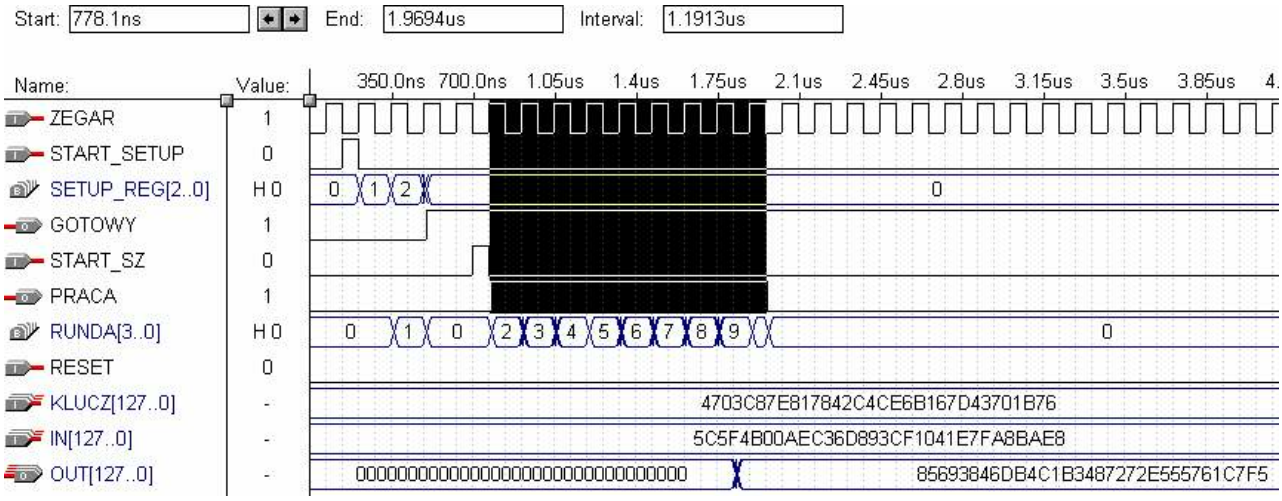
Rys.5.1: Wyniki symulacji układu SZYFROWANIE (krótki okres przygotowania wstępnego).

Po pojawieniu się „narastającego zbocza” na linii START_SETUP rozpoczyna się przygotowanie układu. Długość trwania tego procesu to 2 cykle zegara.

Na potrzeby symulacji zdecydowałem taktować układ zegarem o długości trwania cyklu 124ns. Cała operacja szyfrowania trwa 9 cykli zegara. W oknie interval rysunku widać czas trwania całego procesu szyfrowania – 1,06 μs.

Wykorzystując wzór thr = (dł. bloku danych)/(czas cyklu)*(ilość cykli)

możemy obliczyć że szybkość takiego rozwiązania jest ok. 115 Mb/s.

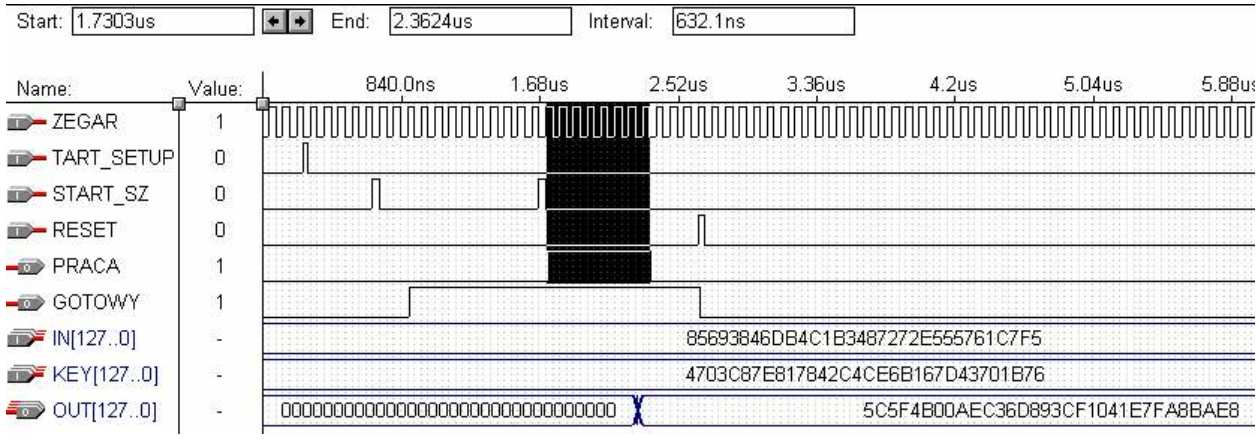


Rys.5.2: Wyniki symulacji układu DESZYFROWANIE (krótki okres przygotowania początkowego).

Wyniki szyfrowania i deszyfrowania oczywiście zgadzają się. Wartości wyjściowe operacji szyfrowania podane na wejściu układu DESZYFROWANIE, przy tym samym kluczu, dają na wyjściu wartości takie same jak na wejściu SZYFROWANIA. Są także zgodne z danymi testowymi zaproponowanymi przez twórców szyfru (dostępne w załączniku nr 1.).

Czas trwania cyklu w procesie deszyfrowania 140ns, ilość cykli – 9, szybkość deszyfrowania – 101Mb/s. W oknie interval widać, ze proces ten trwa 1,20 µs.

- **wersja z długim okresem przygotowania początkowego (1 runda klucza – 1 cykl)**



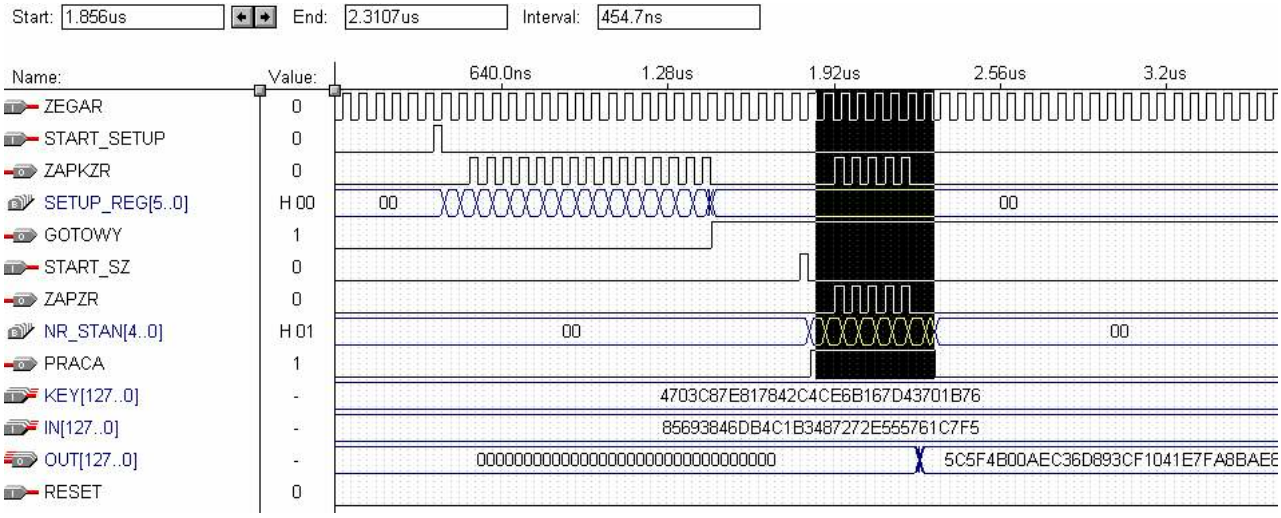
Rys.5.3: Wyniki symulacji układu SZYFROWANIE (długi czas przygotowania wstępnego).

Na rysunku widać, że impuls, który pojawił się linii START_SZ, w momencie gdy układ nie był jeszcze gotowy do pracy został zignorowany i nie rozpoczął się proces szyfrowania. Dopiero

gdy na wyjściu informacyjnym GOTOWY pojawił się stan „wysoki” układ został zaczął pracować. Pojawienie się „narastającego zbocza” na linii RESET spowodowało, ze układ został wprowadzony w stan początkowy.

Okres zegara jakim taktowany jest ten układ wynosi 82ns, a cały proces szyfrowania trwa 8 cykli – czyli 632,1ns. Wykorzystując wzór na szybkość działania otrzymujemy wynik 190Mb/s (8 cykli zegara potrzebnych na ustawienie układu).

- **wersja z długim okresem przygotowania początkowego (1 runda klucza – 3 cykl)**



Rys.5.4: Wyniki symulacji układu SZYFROWANIE (długi czas przygotowania wstępnego 3cykle – runda klucza).

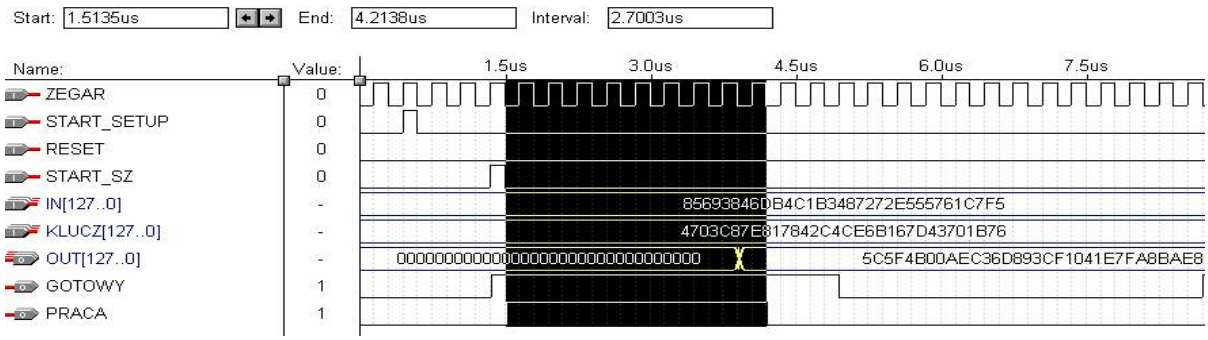
Projekt przechodzi w fazę procesu wstępnego przygotowania po pojawieniu się na linii START_SETUP „narastającego” zbocza. Rozpoczyna się, trwający 17 cykli okres wyznaczenia wszystkich niezbędnych wartości, dzięki którym będzie możliwe wyznaczenie podkluczy rund. Po jego zakończeniu pojawia się stan „wysoki” na wyjściu GOTOWY.

Rozpoczęcie działania następuje w chwili pojawienia się odpowiedniego impulsu na linii START_SZ. Proces szyfrowania trwa 8 cykli a okres zegara taktującego układ trwa tutaj 60ns (cały proces szyfrowania – 480ns). Dzięki temu prędkość szyfrowania w tak zorganizowanym projekcie trwa już 250 Mb/s.

Układy nie wykorzystujące wbudowanych matryc pamięci.

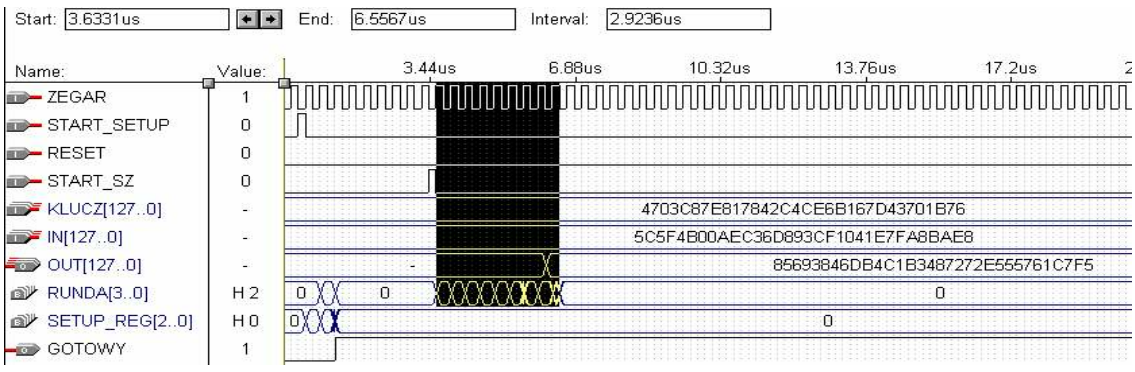
Chcąc zrealizować układ cyforwy, który nie wykorzystywałby pamięci specjalnego przeznaczenia w konfiguracji 256x8 należało zastosować tablice prawdy do realizacji skrzynek podstawieniowych.

Ten sposób wymaga jednak bardzo dużej ilości komórek logicznych, a co za tym idzie powoduje wykorzystanie większej niż jedna struktur programowalnych na funkcję szyfrowania bądź deszyfrowania. Takie „rozbicie" architektury na więcej układów powoduje, ze sygnały i impulsy cyfrowe są obarczone dodatkowymi opóźnieniami pomiędzy układami i powoduje to wyraźne zwolnienie pracy układu. Szybkość przetwarzania w takim układzie jest zdecydowanie mniejsza i wynosi 47Mb/s.



Rys.5.5 Wyniki symulacji układu SZYFROWANIE (krótki czas przygotowania oraz sboxy w postaci tablicy prawdy).

Układ taktowany jest zegarem o długości okresu 300ns, a cały proces trwa 9 cykli- cały proces szyfrowania 2,70 μs. Przy deszyfrowaniu wyniki są jeszcze słabsze: przede wszystkim okres zegara nie może być mniejszy niż 342ns, a ilość potrzebnych cykli wynosi 9. Stąd szybkość deszyfrowania takim układem jest rzędu 41Mb/s.



Rys.5.6: Wyniki symulacji układu DESZYFROWANIE (krótki czas przygotowania oraz sboxy jako tablice prawdy) .

5.2 Wyniki syntezy logicznej.

Wyniki syntezy logicznej zrealizowanych układów przedstawię w następującej kolejności: zacznę od układów, które wykorzystują wbudowaną logikę specjalnego przeznaczenia ponieważ jak się okazało zastosowanie jej pozwoliło na zmieszczenie całego projektu w dwóch strukturach (SZYFROWANIE i DESZYFROWANIE) oraz dało najlepsze wyniki pod względem szybkości działania. Następnie przedstawię alternatywne rozwiązanie, które dzięki wykorzystywaniu tylko elementów języka HDL w projekcie daje możliwość przenoszalności na układy innych firm.

Układy wykorzystujące wbudowane matryce pamięci:

- **wersja z krótkim okresem przygotowania początkowego.**

Wyniki syntezy logicznej dla układu SZYFROWANIE.

Wykorzystana struktura - EPF10K200EBC600-2.

Liczba wykorzystanych:

- wyprowadzeń wejściowych - 260,
- wyprowadzeń wyjściowych – 130,
- komórek pamięci – 8599,
- bitów pamięci wbudowanej – 49150,

Nazwa układu	Liczba zajmowanych komórek LE / bitów EAB	Procentowy udział zajętości układu
SZYFROWANIE	8599 / 49150	86% / 100 %
Warstwy sboxów (runda)	4048 / 32768	40% / 67%
MDS lower level	2688	27 %
MDS higher level	480	4,8 %
M5e	128	1,3 %
Mb3	128	1,3 %
Permutacja 128	128	1,3 %
Permutacja 64	64	0,6 %
Permutacja odw. 128	128	1,3 %
Warstwa sbox (klucz)	0 / 16384	0% / 33%
Sterowanie	33	0,3 %
Rejestr wejściowy	257	2,6 %
Rejestr wyjściowy	128	1,3 %
Rejestr imklucz	1004	10 %
Rejestr podklucza	385	3,9 %
EPF10K200EBC600-2	9986/49150	100% / 100%

Wyniki syntezy logicznej dla układu DESZYFROWANIE.

Wykorzystana struktura - EPF10K200EBC600-2.

Liczba wykorzystanych:

- wyprowadzeń wejściowych - 260,
- wyprowadzeń wyjściowych – 130,
- komórek pamięci – 9055,
- bitów pamięci wbudowanej – 49150,

Nazwa układu	Liczba zajmowanych komórek LE / bitów EAB	Procentowy udział zajętości układu
DESZYFROWANIE	9055 / 49150	91% / 100 %
Warstwy sboxów (runda)	3960 / 32768	39% / 67%
Odw. MDS lower level	2688	27 %
Odw. MDS higher level	480	4,8 %
M5e	128	1,3 %
Mb3	128	1,3 %
Permutacja 128	128	1,3 %
Permutacja 64	64	0,6 %
Permutacja odw. 128	128	1,3 %
Warstwa sbox (klucz)	0 / 16384	0% / 33%
Sterowanie	33	0,3 %
Rejestr wejściowy	257	2,6 %
Rejestr wyjściowy	128	1,3 %
Rejestr imklucz	1004	10 %
Rejestr podklucza	385	3,9 %
EPF10K200EBC600-2	9986/49150	100% / 100%

- **wersja z długim okresem przygotowania początkowego (1 cykl – 1 runda klucza).**

Wyniki syntezy logicznej dla układu SZYFROWANIE.

Wykorzystana struktura - EPF10K200EBC600-1.

Liczba wykorzystanych:

- wyprowadzeń wejściowych - 260,
- wyprowadzeń wyjściowych – 130,
- komórek pamięci – 9497,
- bitów pamięci wbudowanej – 49150,

Nazwa układu	Liczba zajmowanych komórek LE / bitów EAB	Procentowy udział zajętości układu
SZYFROWANIE	9497 / 49150	95% / 100 %
Warstwy sboxów (runda)	4048 / 32768	40% / 67%
MDS lower level	2688	27 %
MDS higher level	480	4,8 %
M5e	128	1,3 %
Mb3	128	1,3 %
Permutacja 128	128	1,3 %
Permutacja 64	64	0,6 %
Permutacja odw. 128	128	1,3 %
Warstwa sbox (klucz)	0 / 16384	0% / 33%
Sterowanie	31	0,3 %
Rejestr wejściowy	130	1,3 %
Rejestr wyjściowy	128	1,3 %
Rejestr imklucz	1004	10 %
Rejestr podklucza	256	2,6 %
Generacja podklucza	1668	17%
Pamieć imklucz	1668	17%
EPF10K200EBC600-1	9986/49150	100% / 100%

Wyniki syntezy logicznej dla układu DESZYFROWANIE.

Wykorzystana struktura - EPF10K200EBC600-1.

Liczba wykorzystanych:

- wyprowadzeń wejściowych - 260,
- wyprowadzeń wyjściowych – 130,
- komórek pamięci – 9756,
- bitów pamięci wbudowanej – 49150,

Nazwa układu	Liczba zajmowanych komórek LE / bitów EAB	Procentowy udział zajętości układu
DESZYFROWANIE	9501 / 49150	95% / 100 %
Warstwy sboxów (runda)	3960 / 32768	39% / 67%
Odw. MDS lower level	2688	27 %
Odw. MDS higher level	480	4,8 %
M5e	128	1,3 %
Mb3	128	1,3 %
Permutacja 128	128	1,3 %
Permutacja 64	64	0,6 %
Permutacja odw. 128	128	1,3 %
Warstwa sbox (klucz)	0 / 16384	0% / 33%
Sterowanie	33	0,3 %
Rejestr wejściowy	257	2,6 %
Rejestr wyjściowy	128	1,3 %
Rejestr imklucz	1004	10 %
Rejestr podklucza	385	3,9 %
Generacja podklucza	1668	17%
Pamieć imklucz	1668	17%
EPF10K200EBC600-1	9986/49150	100% / 100%

- **wersja z długim okresem przygotowania początkowego (3 cykle – 1 runda klucza).**

Wyniki syntezy logicznej dla układu SZYFROWANIE.

Wykorzystana struktura - EPF10K200EBC600-1.

Liczba wykorzystanych:

- wyprowadzeń wejściowych - 260,
- wyprowadzeń wyjściowych – 130,
- komórek pamięci – 9761,
- bitów pamięci wbudowanej – 49150,

Nazwa układu	Liczba zajmowanych komórek LE / bitów EAB	Procentowy udział zajętości układu
SZYFROWANIE	9761 / 49150	97% / 100 %
Warstwy sboxów (runda)	3960 / 32768	40% / 67%
MDS lower level	2688	27 %
MDS higher level	480	4,8 %
M5e	128	1,3 %
Mb3	128	1,3 %
Permutacja 128	128	1,3 %
Permutacja 64	64	0,6 %
Permutacja odw. 128	128	1,3 %
Warstwa sbox (klucz)	0 / 16384	0% / 33%
Sterowanie	36	0,4 %
Rejestr wejściowy	130	1,3 %
Rejestr wyjściowy	128	1,3 %
Rejestr imklucz	1004	10 %
Rejestr podklucza	256	2,6 %
Generacja podklucza	1668	17%
Pamieć imklucz	1668	17%
EPF10K200EBC600-1	9986/49150	100% / 100%

Wyniki syntezy logicznej dla układu DESZYFROWANIE.

Wykorzystana struktura - EPF10K200EBC600-1.

Liczba wykorzystanych:

- wyprowadzeń wejściowych - 260,
- wyprowadzeń wyjściowych – 130,
- komórek pamięci – 9924,
- bitów pamięci wbudowanej – 49150,

Nazwa układu	Liczba zajmowanych komórek LE / bitów EAB	Procentowy udział zajętości układu
DESZYFROWANIE	9620 / 49150	96% / 100 %
Warstwy sboxów (runda)	4448 / 32768	44% / 67%
Odw. MDS lower level	2688	27 %
Odw. MDS higher level	480	4,8 %
M5e	128	1,3 %
Mb3	128	1,3 %
Permutacja 128	128	1,3 %
Permutacja 64	64	0,6 %
Permutacja odw. 128	128	1,3 %
Warstwa sbox (klucz)	0 / 16384	0% / 33%
Sterowanie	33	0,3 %
Rejestr wejściowy	257	2,6 %
Rejestr wyjściowy	128	1,3 %
Rejestr imklucz	1004	10 %
Rejestr podklucza	385	3,9 %
Generacja podklucza	1668	17%
Pamieć imklucz	1668	17%
EPF10K200EBC600-1	9986/49150	100% / 100%

Z danych zamieszczonych w tabelach podsumowujących syntezę logiczną wyciągnąłem następujące wnioski:

- realizacja szyfrowania i deszyfrowania w jednym układzie nie jest możliwa;
- Suma wielkości wszystkich bloczków logicznych jest znacznie większa niż zajętość rzeczywista całego układu – wynika to z oddzielnej kompilacji poszczególnych układów. Podczas kompilacji całego układu następuje redukcja niektórych elementów, które kompilowane oddzielenie zajmują pewne obszary struktury programowalnej (np. Operacja generacja podklucza).
- Układ SZYFROWANIE jest duży i złożony. Zawiera 40 bloków logicznych Sbox (32 runda szyfru, 8 runda klucza). Każdy z nich zajmuje po 253 komórki logiczne. W rezultacie liczba elementów logicznych niezbędnych do realizacji wszystkich Sbox-ów wynosi 4048 oraz 49150. Jest to 40% wszystkich elementów logicznych układu oraz 100% bitów pamięci zgroamadzonej w 24 EABach. Podobna sytuacja zachodzi dla układu DESZYFROWANIE. Zawiera 8 bloków logicznych Sbox (runda klucza) oraz 32 bloki logiczne InvSbox. Układ InvSbox zajmuje 248 komórek logicznych. W rezultacie liczba elementów logicznych niezbędnych do realizacji wszystkich Sboxów i InvSbox-ów wynosi 3960 oraz 49150 bitów pamięci dodatkowej (odpowiednio 39% i 100% zajętości zasobów struktury).

5.3 Ocena otrzymanych wyników.

Symulacyjne testy otrzymanych układów pozwoliły na sprawdzenie działanie projektów zarówno pod względem funkcjonalności, ale i także pod względem wydajności. Faktyczną szybkość działania układu można określić za pomocą szybkości taktowania zegarem – minimalną długością trwania cyklu zegara, przy której wyniki działania opracowanego projektu nadal są zgodne z danymi testowymi. Żeby określić ten minimalny czas trwania cyklu potrzeba zidentyfikować operację, która trwa najdłużej. W przypadku pierwszego projektu są to czas trwania rundy klucza, w drugim czas potrzebny na wyznaczenie danej 64 bitowej (wyjście z funkcji σ) , która niezbędna jest do wyznaczenia podkluczy rund. Natomiast w ostatnim projekcie tą operacją krytyczną z punktu widzenia jej długości trwania jest czas trwania rundy szyfrowania.

Przyjęte przeze mnie wartości długości trwania cyklu zegara dla poszczególnych projektów są minimalnymi, dla których projekty te dają poprawne wyniki.

Poniżej przedstawiam porównanie programowych oraz sprzętowych implementacji firmy TOSHIBA algorytmu Hierocrypt z implementacjami sprzętowymi wykonanymi przeze mnie.

Rodzaj implementacji	Szybkość działania
TOSHIBA: Programowa Visual C++ (Pentium III 650Mhz)	114 Mb/s
TOSHIBA: Programowa DEC C(Alpha 21263 463MHz)	87,3 Mb/s
TOSHIBA: Programowa Forte C (Ultra Sparc IIi 400MHz)	62,2 Mb/s
TOSHIBA: Sprzętowa AHDL (ALTERA: Max PLUS II)	52,6 Mb/s
Wersja z krótkim okresem ustawienia układu	115 Mb/s
Wersja z długim okresem ustawienia układu (1 cykl)	190 Mb/s
Wersja z długim okresem ustawienia układu (3cykle)	250 Mb/s

Przedstawione przeze mnie rozwiązania spełniają wcześniej przyjęte założenia. Założenie o przenoszalności układu, nie było możliwe do pogodzenia z koniecznością realizacji projektu w co najwyżej dwóch układach: jednego przeznaczonego na funkcję szyfrowania, drugiego dla deszyfrowania. Projekty wykorzystujące specjalne pamięci o szybkim dostępie, występujące tylko w układach firmy ALTERA, pozwalają na uzyskanie największych szybkości szyfrowania i deszyfrowania, a także pozwalają na wykorzystanie tylko 2 układów (żadna z wersji układu SZYFROWANIE realizująca sboxy w formie tablic prawdy nie mieściła się w mniejszej ilości układów niż 5).

Wyniki osiągnięte przez projekty wykorzystujące EABy, potwierdzają możliwość tworzenia specjalizowanych struktur sprzętowych, realizujących skomplikowane operacje kryptologiczne. Przeprowadzone badania nad implementacją algorytmu Hierocrypt sugerują, że może on być bardzo szybko działać zaimplementowany w strukturach programowalnych. Zaprezentowane rozwiązania świadczą na korzyść algorytmu i wydaje się nieco pochopną decyzja o dyskwalifikacji szyfru z konkursu NESSIE, z powodu jego słabej implementacji sprzętowej.

5.4 Koncepcja rozwoju projektu.

Na podstawie wyników otrzymanych podczas procesu symulacji widać, że działaniem, które w największym stopniu wpływa na przyspieszenie szybkości algorytmu jest stosowanie szybkich pamięci. Układy FLEX10KE, choć zapewne najlepiej przystosowane, spośród dostępnych w MAX PLUS II, do możliwości algorytmu Hierocrypt, nie są technologicznie najwłaściwszymi dla zrealizowanego projektu.

W systemie projektowy Quartus, także firmy ALTERA, są dostępne układy typu Stratix, które zawierają znaczenie więcej EABów. Dzięki zastosowaniu tego typu struktur możliwym byłoby zrealizowanie wszystkich operacji podstawienia za pomocą implementacji skrzynek w tego typu pamięciach.

W ramach architektury iteracyjnej możliwym do zrealizowania stałby się także pomysł, który opiera się na idei polegającej na wzroście prędkości wraz ze wzrostem pojemności. Opiszę teraz pokrótce sposób realizacji takiego rozwiązania.

Algorytm Hierocrypt jest zorientowany na architektury 8 bitowe. Operację rundy stanowią ośmio bitowe operacje podstawieniowe oraz ośmio bitowe składowe opercji mnożenia przez macierz kodu MDS. Mnożenie przez wektory z ciała $GF(2^8)$ modulo wielomian pierwotny powoduje permutację elementów tego ciała, zatem operacja ta jest swoistym podstawieniem, które można połączyć razem z poprzedzającym ją sboxem. Na każdy wyjściowy bajt przypadała w przypadku operacji MDS lower level operacja dodawania czterech wektorów powstałych w wyniku mnożenia odpowiedniej wartości z macierzy generującej kod MDS z wartością ośmio bitową.

W ramach 32 bitowego słowa realizowana jest więc operacja:

$$y_{1(8)} = C4*x_{1(8)} \oplus 65*x_{2(8)} \oplus C8*x_{4(8)} \oplus 8B*x_{4(8)}$$
$$y_{2(8)} = 8B*x_{1(8)} \oplus C4*x_{2(8)} \oplus 65*x_{4(8)} \oplus C8*x_{4(8)}$$
$$y_{3(8)} = C8*x_{1(8)} \oplus 8B*x_{2(8)} \oplus C4*x_{4(8)} \oplus 65*x_{4(8)}$$
$$y_{4(8)} = 65*x_{1(8)} \oplus C8*x_{2(8)} \oplus 8B*x_{4(8)} \oplus C4*x_{4(8)}$$

Zauważmy więc, że każdy bajt $x_{n(8)}$, mnożony jest przez odpowiedni element z macierzy i jest składową cząstkową każdego bajtu wyjściowego z operacji MDS. Każdą operację mnożenia można więc połączyć z każdą operacją podstawienia. Każdy bajt wejściowy do takiego złożenia operacji będzie miał 4 bajtów wyjściowych. Wynika to z faktu, że będą cztery rodzaje skrzynek podstawieniowych:

- sbox z Hierocrypt + mnożenie przez C4;
- sbox z Hierocrypt + mnożenie przez 8B;
- sbox z Hierocrypt + mnożenie przez C8;
- sbox z Hierocrypt + mnożenie przez 65;

Z każdych 4 bajtów wyjściowych następnie zostanie wyznaczony pojedynczy bajt wyjściowy. Będzie to realizowane w sposób jaki prezentuje przykład:

$$y_{1(8)} = SBOX\_C4(x_{1(8)}) \oplus SBOX\_65(x_{2(8)}) \oplus SBOX\_C8(x_{4(8)}) \oplus SBOX\_8B(x_{4(8)}).$$

Gdzie:

SBOX_C4 – oznacza złożenie operacji podstawienia z mnożeniem przez C4.

SBOX_8B – oznacza złożenie operacji podstawienia z mnożeniem przez 8B.

SBOX_C8 – oznacza złożenie operacji podstawienia z mnożeniem przez C8.

SBOX_65 – oznacza złożenie operacji podstawienia z mnożeniem przez 65.

Determinuje to użycie 64 sboxów w tej warstwie, a w całej rundzie szyfru 80. W taki sposób cały algorytm będzie wymagał użycia razem z rundą klucza 88 EABów.

Szacuję więc, ze gdyby zrealizować w następujący sposób architekturę iteracyjną korzystając w jej odmianie z długim okresem przygotowania wstępnego, gdzie runda klucza trwa 3 cykle, możliwym byłoby osiągnięcie nawet dwukrotnego przyspieszenia szyfru.

Zastosowanie struktur programowalnych Stratix pozwoliłoby prawdopodobnie na zrealizowanie architektur kombinacyjnej oraz potokowej, które działałyby zapewne jeszcze szybciej.

Podsumowanie

Zaproponowana przez TOSHIBA Corp. implementacja sprzętowa przy wykorzystaniu struktur programowalnych firmy ALTERA okazała się bardzo mało efektywna. Projekt realizacji algorytmu Hierocrypt jako funkcji iteracyjnej, zajmuje ponad 22000 komórek logicznych i niestety nie mieści się w żadnym z układów rodziny FLEX10K dostępnym w środowisku MAX PLUS II. Wykorzystanie aż pięciu układów tego typu spowodowało powstanie znacznych opóźnień przesyłania sygnału pomiędzy oddzielnymi strukturami, a co za tym idzie w bardzo znaczący sposób zmniejszyła się szybkość działania projektu (52,6Mb/s). Algorytm Hierocrypt, startujący w ramach konkursu NESSIE, postrzegany był więc jako jeden z najwolniejszych oraz najmniej efektywnych.

Otrzymane przeze mnie wyniki szybkości oraz efektywności implementacji są zdecydowanie lepsze. Układy SZYFROWANIE i DESZYFROWANIE, wykorzystujące dodatkową wbudowaną pamięć, zajmują ponad dwukrotnie mniej miejsca całościowo i dzięki temu każdy z nich, w każdej wersji, mieści się w EPF10K200EBC600-1(2).

Samo wykorzystanie technologii szybkich, dodatkowych matryc pamięci wbudowanych, pozwoliło przede wszystkim na zmieszczenie projektu każdej z funkcji w jednym układzie. Dzięki tej realizacji architektury iteracyjnej udało się otrzymać już wynik znaczenie lepszy od zaproponowanego przez autorów szyfru układu – 115 Mb/s przy szyfrowaniu i 101 Mb/s przy deszyfrowaniu.

Dalsze działanie zmierzające do identyfikacji operacji najwolniej działającej w algorytmie przyczyniło się do zmiany struktury procesu przygotowania układu. Tą najwolniej działającą operacją jest runda generacji podklucza, która charakteryzuje się symetrią. Wykorzystanie tego faktu i podzielenie jej na dwie części (podklucze przejściowe generowane są podczas ustawiania , a główne w czasie pracy układu) spowodowało wzrost szybkości działania do 190 Mb/s, przy praktycznie znikomym wzroście ilości potrzebnych komórek logicznych.

Wersja ostatnia, która w procesie ustawiania początkowego wykorzystuje po trzy cykle zegara na rundę podklucza oraz jeden cykl na rundę szyfru działa najszybciej – 250 Mb/s, pięć razy szybciej niż projekt TOSHIBA Corp.. Implementacja ta wydaje się być najlepszą, ponieważ niemożliwym jest już zwiększanie liczby cykli w procesie generacji podklucza z jednoczesnym wzrostem prędkości układu. Najwolniej działającą operacją w tej realizacji architektury iteracyjnej jest runda szyfru. Dalsze rozkładanie na mniejsze czynniki tej operacji nie dało już lepszych wyników.

Bibliografia:

1. Toshiba Corporation – Specification on a Block Cipher: Hierocrypt –3, 09.2001r.
2. Toshiba Corporation – Self Evaluation: Hierocrypt –3, 10.2001r.
3. NESSIE (B.Preneel, B.Van Rompay, L.Granboulan, G.Martinet, S.Murphy, R.Shipsey, J.White, M.Dichtl, P. Serf, M.Schaftheutle, E.Biham, O.Dunkelman, M.Ciet, J-J.Quisquarter, F.Sica, L.Knudsen, H.Raddum) – Phase I: Selection of primitives.-23.09.2001r.
4. P. Bora, T. Czajka – „Implementation of the Serpent alghoritm using Altera FPGA devices" – 13.05.2000r.
5. P. Mroczkowski - Implementation of block cipher Rijndael using Altera FPGA – 10.05.2000 r.
6. Janusz Szmidt, Michał Misztal – "Wstęp do Kryptologii" –2001r.
7. B. Schneier – Kryptografia dla praktyków,WNT Warszawa 1995r.
8. Y. Braziler – Statistical Evaluation of the NESSIE Submission Hierocrypt-3, Technion, Izrael, 28. 11. 2001r.
9. M. Misztal – Cykl wykładów "Projektowanie szyfrów blokowych", Warszawa, 2003r.
10. T. Łuba, M.A. Markowski, B.Zbierzychowski - „Komputerowe projektowanie układów cyfrowych w strukturach PLD" – Warszawa, 1993r.
11. ALTERA Corporation - „Device Data Book", 1999r.