Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Równoległa implementacja szyfrowania AES

Adam Krawczyk
Eryk Wawrzyn
Marcin Skrzypkowski
Numery albumów: 317678,291018, 283419

WARSZAWA 2022

Spis treści

1. Szyfr AES

1.1. Historia

Na początku lat 90. stało się jasne, że potrzebny jest nowy standard w dziedzinie kryptografii. Wynikało to z faktu, że zarówno długość bloku (64 bity) jak i długość klucza (56 bitów) podstawowego algorytmu DES (wynalezionego w latach 70-tych) były zbyt małe dla przyszłych zastosowań (obecnie możliwe jest odzyskanie 56-bitowego klucza DES przy użyciu sieci komputerów lub specjalistycznego sprzętu). W odpowiedzi na ten problem amerykański Narodowy Instytut Norm i Technologii (NIST) zainicjował konkurs na nowy szyfr blokowy, który miał nosić nazwę Advanced Encryption Standard lub AES. W przeciwieństwie do procesu projektowania DES, który był utrzymywany w ścisłej tajemnicy, projekt AES został przeprowadzony publicznie. Wiele grup z całego świata przedstawiło projekty szyfru AES. Ostatecznie wybrano pięć algorytmów, znanych jako finaliści AES, które poddano szczegółowym badaniom.

Były to:

- MARS opracowany przez grupę z IBM,
- RC6 od grupy z RSA Security,
- Twofish od grupy z Counterpane, UC Berkeley i innych,
- Serpent od grupy trzech naukowców z Izraela, Norwegii i Wielkiej Brytanii,
- Rijndael od pary młodych belgijskich kryptografów (Vincent Rijmen i Joan Daemen).

Wreszcie jesienią 2000 roku NIST ogłosił, że ogólnym zwycięzcą AES został wybrany Rijndael.

1.2. Wstęp do zasad działania

DES i wszyscy finaliści AES są przykładami iterowanych szyfrów blokowych. Szyfry blokowe uzyskują swoje bezpieczeństwo poprzez wielokrotne użycie prostej funkcji zaokrąglania. Funkcja zaokrąglająca przyjmuje n-bitowy blok i zwraca n-bitowy blok, gdzie n jest rozmiarem bloku całego szyfru. Liczba rund r może być zmienna lub stała. Ogólną zasadą jest, że zwiększenie liczby rund zwiększa poziom bezpieczeństwa szyfru blokowego.

Aby umożliwić opis, każda runda musi być odwracalna. AES jest algorytmem z kluczem symetrycznym, co oznacza, że ten sam klucz jest używany zarówno do szyfrowania, jak i odszyfrowywania danych. AES ma stały rozmiar bloku 128 bitów i rozmiar klucza 128, 192 lub 256 bitów.

1.3. Definicje

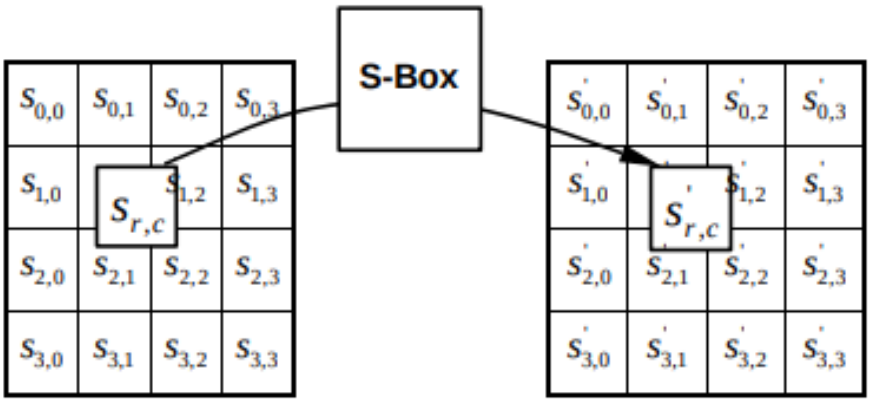
W danym standardzie wykorzystywany jest szereg definicji:

- AES - Advanced Encryption Standard.
- Bit - Cyfra binarna przyjmująca wartość 0 lub 1.
- KeyExpansion - Funkcja używana do generowania ciągu kluczy rund.

- State - Pośredni efekt szyfrowania posiadający cztery wiersze i Nb kolumn.
- Word - Grupa 32 bitów (4 słów ośmiu bitowych).
- Nb - Numer kolumn (32 bitowych słów). W tej implementacji Nb = 4.
- Nk - Liczba słów składających się na klucz szyfru. Nk = 4, 6, 8.
- Nr - Liczba rund szyfrujących. Im więcej rund tym lepsze zabezpieczenie używa się określonej liczby rund do określonej długości klucza. Nr = 10, 12, 14.
- Rcon[] - wartości stałe dla poszczególnych rund.
- S-box - tablica przechowująca stałe dla danego algorytmu [1].

1.4. Funkcja SubBytes()

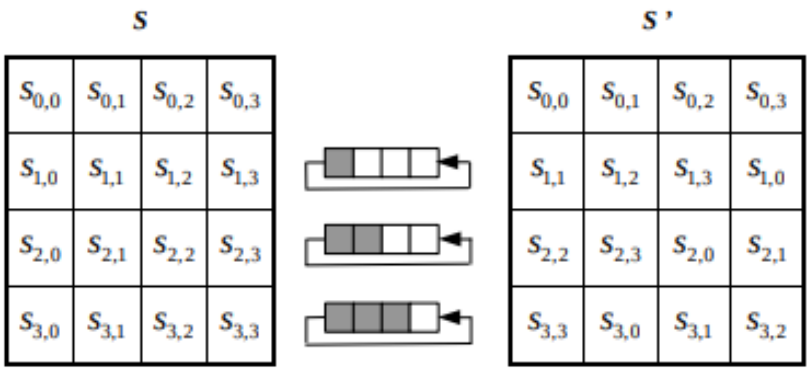
Funkcja odpowiada za przydzielenie wartości z tablicy stałych S-box w zależności od wartości poszczególnych elementów stanu (np. jeżeli któryś ze stanów posiada wartość 1 zostanie mu przyporządkowany pierwszy element tablicy S-box). Na rys. 1.1 przedstawiono uproszczony schemat operacji wykonywanych przez funkcje, s jest tablicą dwu wymiarową stanów w zależności od ich wartości przydzielane są nowe wartości s'.



Rysunek 1.1. Schemat operacji [1]

1.5. Funkcja ShiftRows()

Funkcja ShiftRows() odpowiada za "przesunięcie"w wierszach elementów tablicy stanów. Przesunięcie o jeden element oznacza, że pierwszy element wiersza zostanie przeniesiony na ostatnie miejsce, a na jego miejsce trafi drugi element wiersza. W pierwszym wierszu tablicy stanów nie wykonywane jest żadne przesunięcie. Dla drugiego wiesza wykonywane jest jedno przesunięcie, dla trzeciego wiesza wykonywane są dwa przesunięcia, dla czwartego wiesza wykonywane są trzy przesunięcia. Schemat zamian przedstawiono na rys. 1.2



Rysunek 1.2. Schemat operacji [1]

1.6. Funkcja MixColumns()

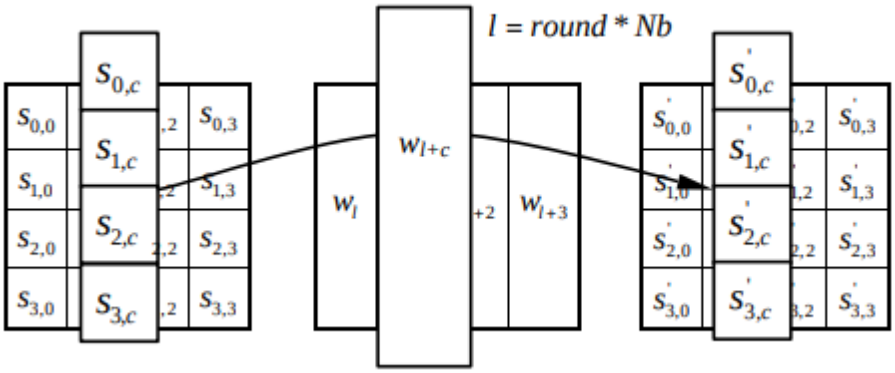
Funkcja MixColumns odpowiada za wykonanie odpowiednich operacji na poszczególnych kolumnach stanów. Pseudokod operacji wykonywanych w ramach danej funkcji dla pojedynczej kolumny tablicy stanów przedstawiono poniżej.

Listing 1. *Pseudokod szyfrowania*

```
1  gmix_column(byte *r)
2  begin
3      byte a[4];
4      byte b[4];
5      byte c;
6      byte h;
7
8      for c = 0 step 1 to 4
9          a[c] = r[c];
10         h = (r[c] >> 7) & 1;
11         b[c] = r[c] << 1;
12         b[c] ^= h * 0x1B;
13     end for
14
15     r[0] = b[0] ^ a[3] ^ a[2] ^ b[1] ^ a[1];
16     r[1] = b[1] ^ a[0] ^ a[3] ^ b[2] ^ a[2];
17     r[2] = b[2] ^ a[1] ^ a[0] ^ b[3] ^ a[3];
18     r[3] = b[3] ^ a[2] ^ a[1] ^ b[0] ^ a[0];
19 end
```

1.7. Funkcja AddRoundKey()

Funkcja AddRoundKey() odpowiada za dołączenie wartości RoundKey do tablicy wartości stanów. Połączanie poszczególnych wartości RoundKey i stanów dokonywane jest poprzez operacji xor. Na rys. 1.3 przedstawiono schemat operacji wartości w to klucze dla poszczególnych rund. Wartości s i w są poddawane operacji xor w wyniku czego powstaje nowa tablica stanów s'.



Rysunek 1.3. Schemat operacji [1]

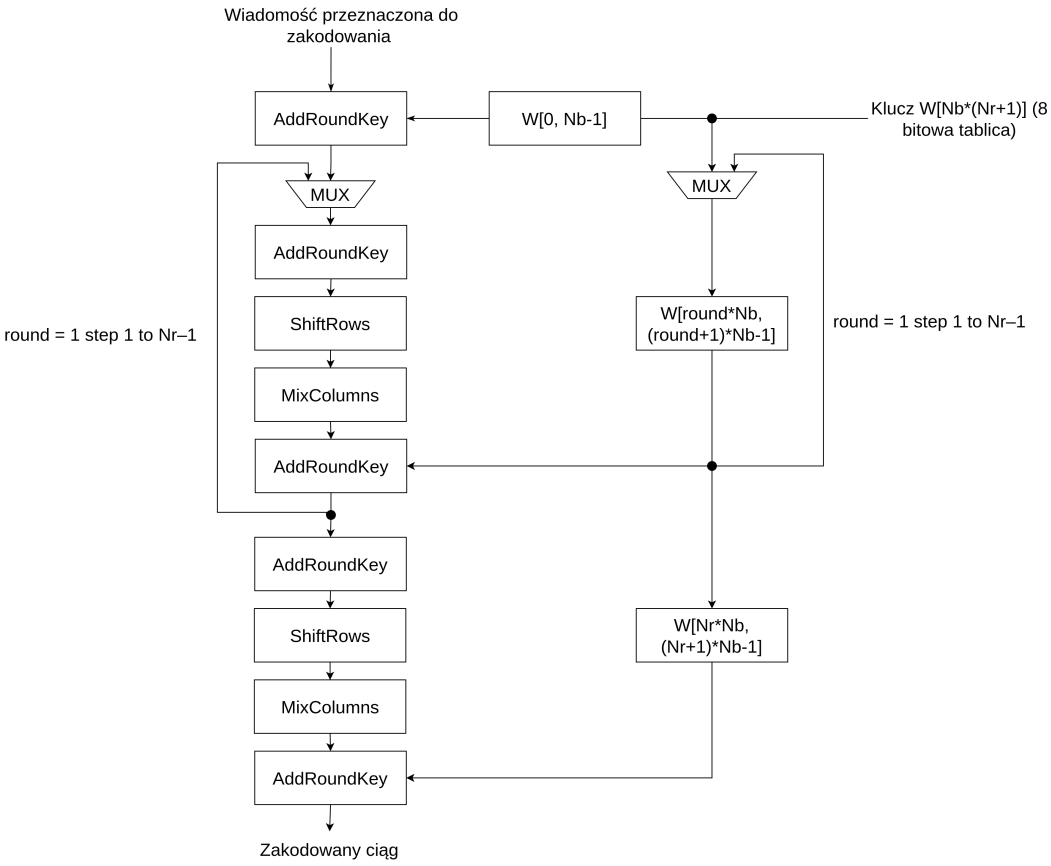
1.8. Cipher

Funkcja cipher łączy powyższe funkcje i odpowiada za kodowanie wartości w standardzie AES. Poniżej przedstawiono schemat blokowy (rys. 1.4) i pseudokod danej funkcji.

Listing 2. *Pseudokod szyfrowania*

```
1  Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2  begin
3      byte state[4,Nb]
4      state = in
5
6      AddRoundKey(state, w[0, Nb-1])
7
8      for round = 1 step 1 to  N r 1
9          SubBytes(state)
10         ShiftRows(state)
11         MixColumns(state)
12         AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
13     end for
14
15     SubBytes(state)
```

```
16        ShiftRows(state)
17        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
18        out = state
19
20    end
```



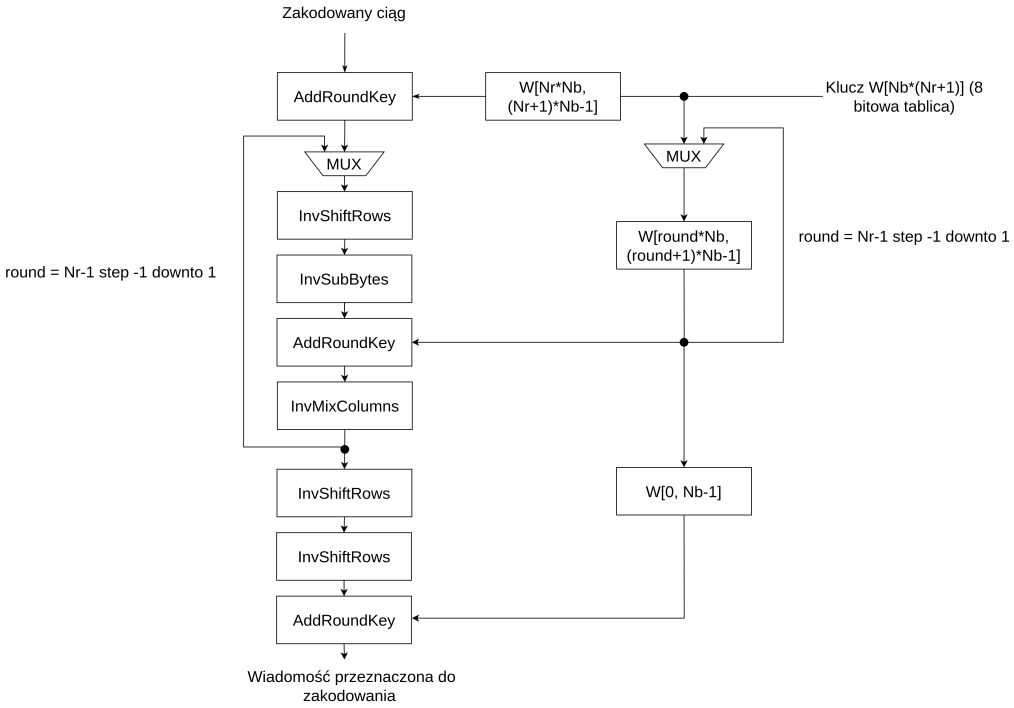
Rysunek 1.4. Schemat blokowy szyfrowania

1.9. InvCipher

Funkcja InvCipher służy do rozkodowania wiadomości zakodowanych w standardzie AES. Poniżej przedstawiono schemat blokowy (rys. 1.5) i pseudokod danej funkcji.

Listing 3. *Pseudokod deszyfrowania*

```
1  InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
2  begin
3      byte state[4,Nb]
4      state = in
5      AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
6      for round = Nr-1 step -1 downto 1
7          InvShiftRows(state)
8          InvSubBytes(state)
9          AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
10         InvMixColumns(state)
11     end for
12
13 InvShiftRows(state)
14 InvSubBytes(state)
15 AddRoundKey(state, w[0, Nb-1])
16 out = state
17
18 end
```



Rysunek 1.5. Schemat blokowy deszyfrowania

1.10. KeyExpansion

Funkcja KeyExpansion odpowiada za rozszerzenie klucza wejściowego, tworzone są klucze osobne dla każdej rundy algorytmu (każdej iteracji pętli funkcji Cipher). Poniżej przedstawiono pseudokod danej funkcji.

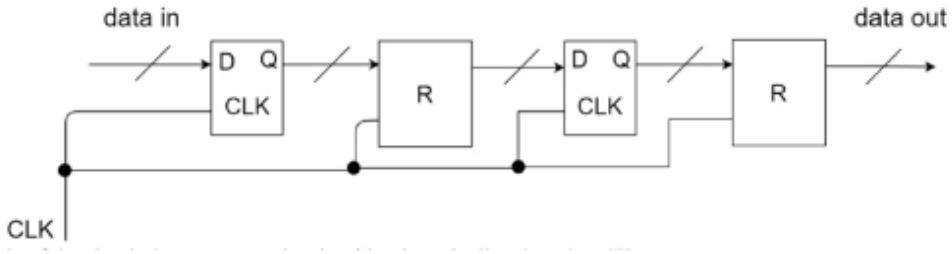
Listing 4. *Pseudokod rozwijania klucza*

```
1  KeyExpansion(byte key[4 * Nk], word w[Nb * (Nr + 1)], Nk)
2  begin
3
4  word temp
5  i = 0
6
7  while (i < Nk)
8      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
9      i = i+1
10
11 end while
12
13 i = Nk
14
15 while (i < Nb * (Nr+1)]
16
```

```
17      temp = w[i-1]
18      if (i mod Nk = 0)
19          temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
20      else if (Nk > 6 and i mod Nk = 4)
21          temp = SubWord(temp)
22      end if
23
24      w[i] = w[i-Nk] xor temp
25      i = i + 1
26  end while
27
28  end
```

2. Implementacja algorytmu AES na układ FPGA

W ramach pracy została zaimplementowana potokowa wersja algorytmu w technice HLS FPGA (umożliwiającej implementacje na platformę FPGA w językach C i C++). Do każdej funkcji algorytmu i pętli zostanie dodana dyrektywa *#pragma pipeline rewind*. Technika potokowa pozwala uniknąć zależności danych i zwiększyć przepustowość algorytmu. Tworzony obwód podzielony jest na łańcuch niezależnych etapów. Wszystkie stopnie w łańcuchu przebiegają szeregowo w tym samym cyklu sygnału zegarowego. Każdy etap w obliczeniach otrzymuje wartości danych z wyniku obliczonego przez poprzedni etap, podczas poprzedniego cyklu sygnału zegarowego. Implantacje z pełnym wypełnieniem potoku umożliwią uzyskanie dużej wydajności. Układy przetwarzające algorytm w sposób potokowy wykorzystują dużą liczbę zasobów, dlatego są one wdrażane w rozwiązaniach, gdzie wymagana jest duża wydajność realizowanej funkcji. Implementacja algorytmu w sposób potokowy możliwa jest, gdy nie występują zależności pomiędzy danymi w poszczególnych rundach algorytmu. Schemat układu przetwarzającego algorytm potokowo przedstawiono na rys. 2.1.



Rysunek 2.1. Schemat blokowy przeważania potokowego **FPGA**

W danym projekcie został wykorzystany interfejs komunikacyjny AXI4 Stream. Interfejs AXI4 Stream umożliwia przesyłanie w każdym cyklu sygnału zegarowego danych pomiędzy modułami (master - slave). Jest bezpośrednim kanałem komunikacji „punk-punkt" (schemat transmisji strumieniowej przedstawiono na rys. 2.2)

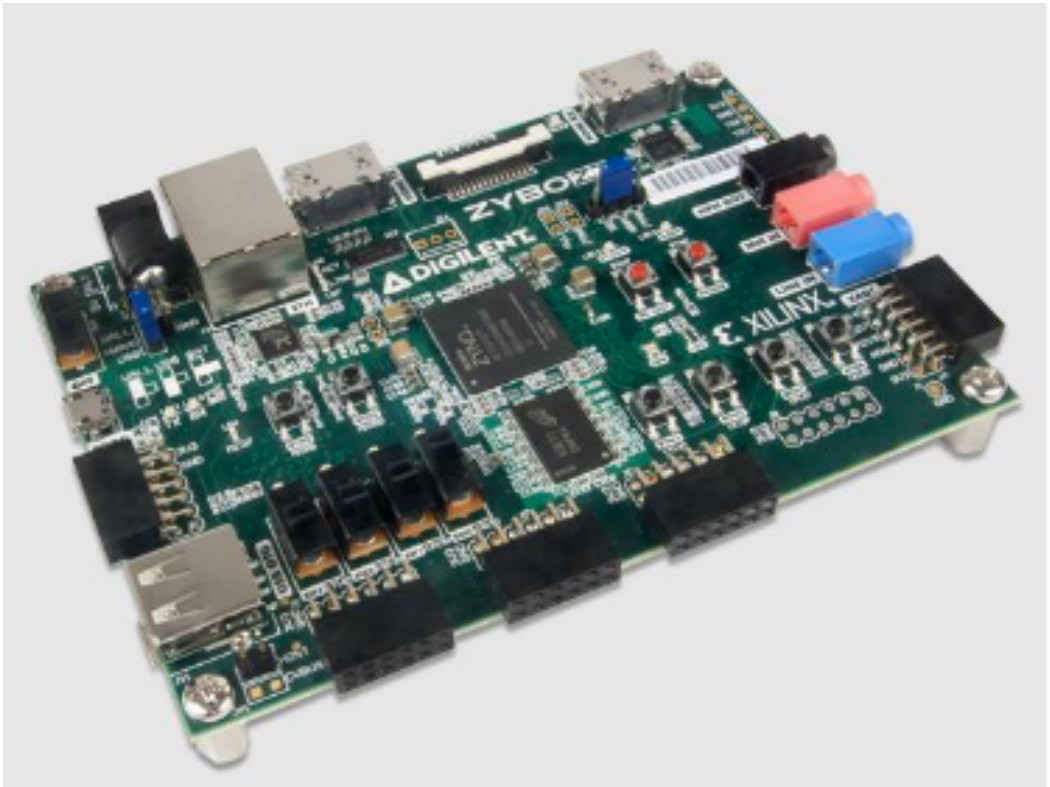
AXI Stream składa się głównie z trzech sygnałów:

- Tdata – dane docelowo transmitowane;
- Tvalid – sygnał wysyłany z jednostki master do slave określa,czy nowe dane są gotowe do transmisji;
- Tready –sygnał wysyłany z slave dojednostki master wskazuje,czy slave może odbierać dane.

Rysunek 2.2. Schemat blokowy szyfrowania

2.1. Wyniki syntezy

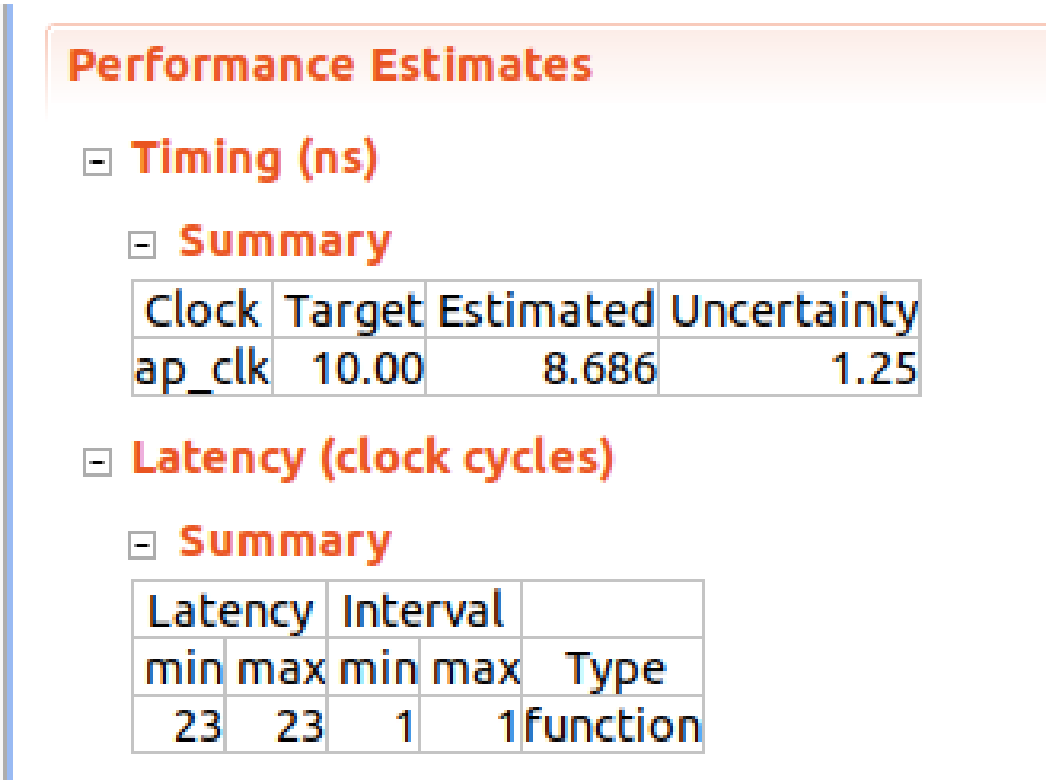
W ramach projektu została przeprowadzona synteza bloków do szyfrowania i deszyfrowania wiadomości za pomocą algorytmu AES w środowisku Vivado HLS 2019 na układy FPGA. W celu wykonania testów zaimplementowanych bloków w technice HLS została wybrana płytka deweloperska „ZYBO Z7-20" (rys. 2.3). Wyposażona jest w układ programowalny i procesor ARM Zynq-7000 (model: XC7Z020-1CLG400C).Rodzina układów Zynq jest oparta na architekturze „Xilinx All Programmable System-on-Chip", która integruje procesor ARM Cortex-A9 z FPGA z serii Xilinx 7.



Rysunek 2.3. Układ programowalny Zybo z7-20

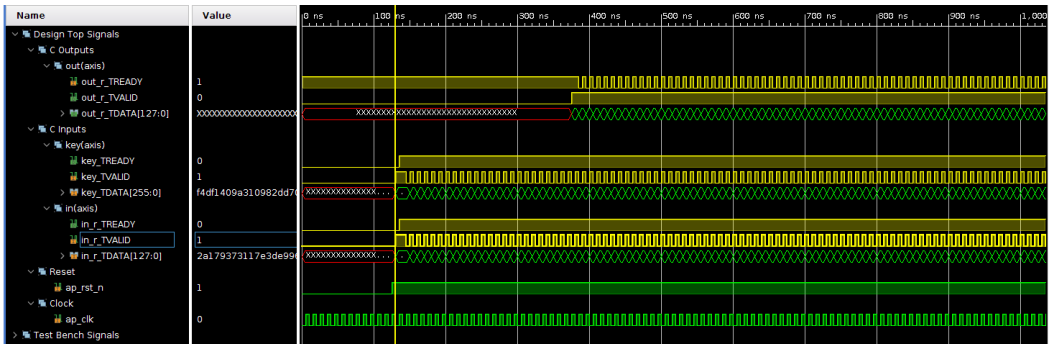
2.1.1. Blok szyfrujący wiadomość

Z raportu syntezy wygenerowanego przez środowisko Vivado HLS 2019 (rys. 2.4) wynika, że dla danego bloku uzyskano latencje (opóźnię po jakim dla danej wejściowej zostanie obliczony wynik) równą 23 cykle sygnału zegarowego. Natomiast interwał danego układu to 1 cykl sygnału zegarowego, co dowodzi, że pełen potok został osiągnięty. Interwał definiuje z jakim opóźnieniem dane na mogą się pojawiać się na wejściu i wyjściu. Wydajność obliczeniowa jest definiowana przez interwał ponieważ po napełnieniu potoku (23 cykle sygnału zegarowego) dane mogą pojawiać się na wyjściu co jeden cykl sygnału zegarowego.



Rysunek 2.4. Wyniki syntezy dla bloku FPGA odpowiedzialnego za szyfrowanie wiadomości

Na rys. 2.5 przedstawiono wyniki symulacji rtl dla danego bloku. Układ działa zgodnie z oczekiwaniami po 23 cyklach sygnału zegarowego od pojawienia się pierwszej danej wejściowej zwraca wynik obliczeń. Dane pojawiają się co jeden cykl sygnału zegarowego na wejściu i wyjściu układu, co dowodzi temu, że pełen potok układu został osiągnięty.



Rysunek 2.5. Wyniki symulacji dla bloku FPGA odpowiedzialnego za szyfrowanie wiadomości

2.1.2. Blok odszyfrowujący wiadomość

Z raportu syntezy wygenerowanego przez środowisko Vivado HLS 2019 (rys. 2.6) wynika, że dla danego bloku uzyskano latencje (opóźnię po jakim dla danej wejściowej zostanie obliczony wynik) równą 50 cykle sygnału zegarowego. Natomiast interwał danego układu to 1 cykl sygnału zegarowego, co dowodzi, że pełen potok został osiągnięty. Interwał definiuje z jakim opóźnieniem dane na mogą się pojawiać się na wejściu i wyjściu. Wydajność obliczeniowa jest definiowana przez interwał ponieważ po napełnieniu potoku (50 cykle sygnału zegarowego) dane mogą pojawiać się na wyjściu co jeden cykl sygnału zegarowego.

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.488	1.25

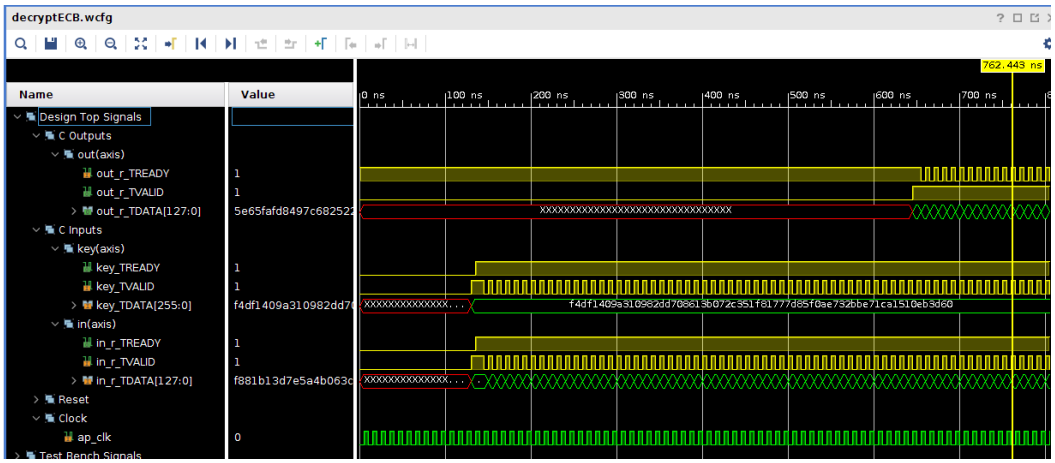
Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
50	50	1	1	function

Rysunek 2.6. Wyniki syntezy dla bloku FPGA odpowiedzialnego za deszyfrowanie wiadomości

Na rys. 2.7 przedstawiono wyniki symulacji rtl dla danego bloku. Układ działa zgodnie z oczekiwaniami po 50 cyklach sygnału zegarowego od pojawienia się pierwszej danej wejściowej zwraca wynik obliczeń. Dane pojawiają się co jeden cykl sygnału zegarowego na wejściu i wyjściu układu, co dowodzi temu, że pełen potok układu został osiągnięty.



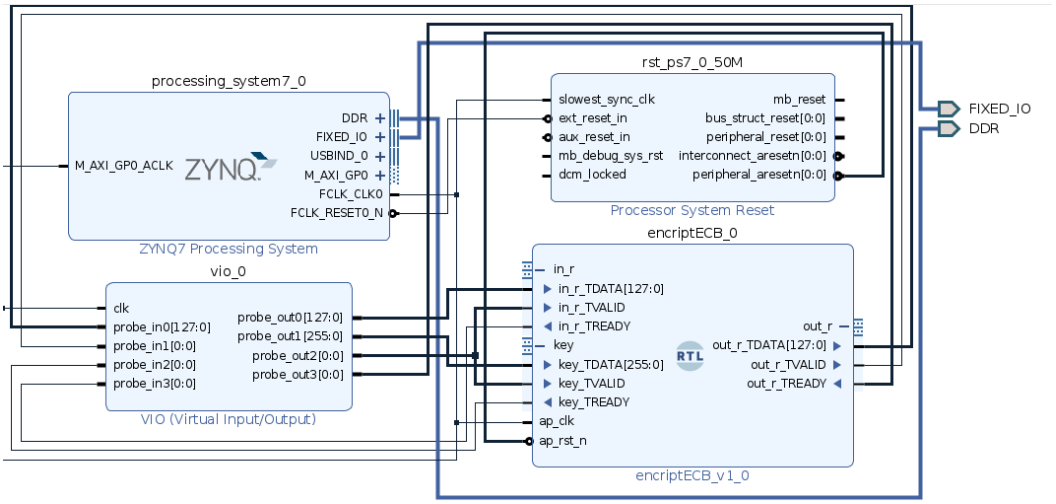
Rysunek 2.7. Wyniki symulacji dla bloku FPGA odpowiedzialnego za deszyfrowanie wiadomości

2.2. Wyniki implementacji

W celu przetestowania danych bloków na układzie FPGA wykonano implementacje układy w środowisku Vivado 2021. Na podstawie implementacji odczytano uzyskaną częstotliwość sygnału zegarowego oraz zajętość zasobów.

2.2.1. Blok szyfrujący wiadomość

W ramach pracy została przygotowana infrastruktura (rys. 2.10), która umożliwiała zbadanie poprawności obliczeń bloków (komponent vio), ustalenie maksymalnej częstotliwości sygnału zegarowego i zajętość zasobów.



Rysunek 2.8. Infrastruktura zaimplementowana w środowisku Vivado

Opis bloków:

- ZYNQ – procesor użyty jako źródło sygnału zegarowego;
- proc_sys_reset – blok obsługujący sygnały resetu;
- VIO – blok umożliwiający wysyłanie i odbieranie danych;

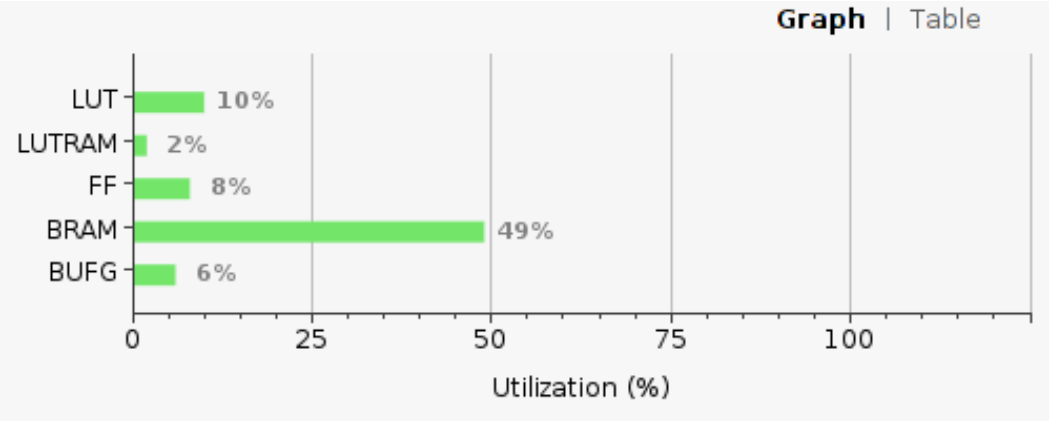
- encriptECB – blok odpowiedzialny za szyfrowanie wiadomości zgodnie z standardem AES.

Na rys. 2.9 przedstawiono zajętość zasobów danego układu, dany blok zużywa dużo elementów BRAM 49%, przez co na danym układzie istnieje możliwość implementacji około 2 bloków do szyfrowania (w optymistycznym przypadku). Dla danego układu uzyskano częstotliwość sygnału zegarowego równą 125 MHz.

Na podstawie uzyskanych danych została policzona wydajność danego układu, określona jako liczba zakodowanych wiadomości jaką może dany blok policzyć w ciągu jednej sekundy(H/s). Wydajność danego bloku to 125 MH/s.

$$W = 1/(T * L)[H/s]$$

W – wydajność
T –okres sygnału zegarowego
L –liczba cykli zegara pomiędzy nowymi danymi wejściowymi



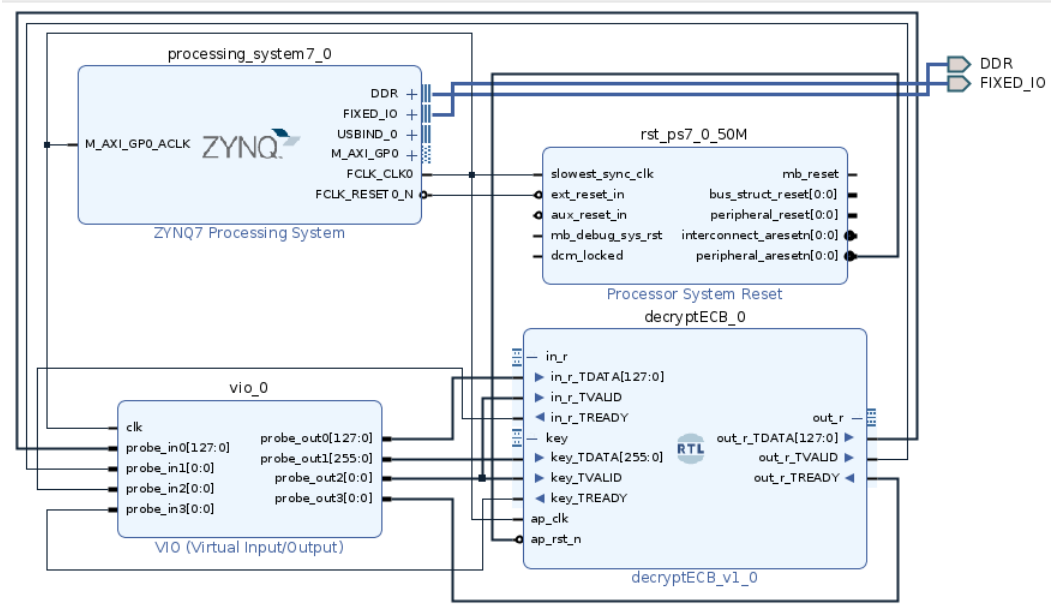
Rysunek 2.9. Zajętość zasobów infrastruktury

2.2.2. Blok deszyfrujący wiadomość

W ramach pracy została przygotowana infrastruktura (rys. **??**), która umożliwiała zbadanie poprawności obliczeń bloków (komponent vio), ustalenie maksymalnej częstotliwości sygnału zegarowego i zajętość zasobów.

Opis bloków:

- ZYNQ – procesor użyty jako źródło sygnału zegarowego;
- proc_sys_reset – blok obsługujący sygnały resetu;
- VIO – blok umożliwiający wysyłanie i odbieranie danych;
- dencriptECB – blok odpowiedzialny za deszyfrowanie wiadomości zgodnie z standardem AES.



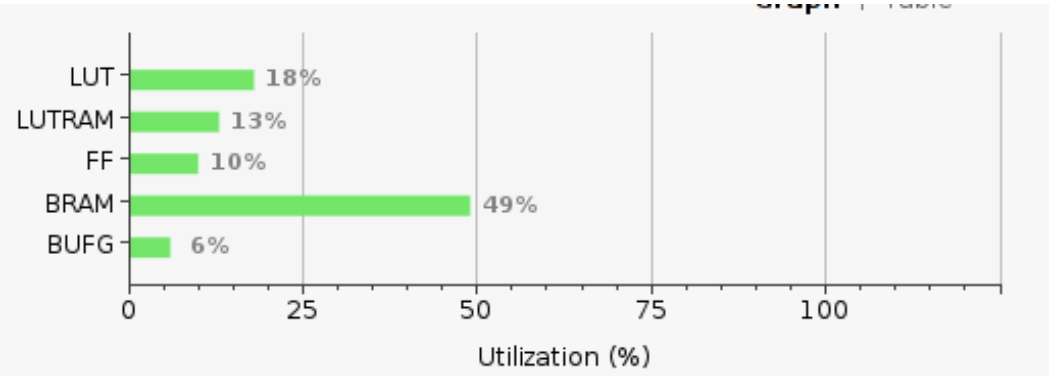
Rysunek 2.10. Infrastruktura zaimplementowana w środowisku Vivado

Na rys. 2.11 przedstawiono zajętość zasobów danego układu, dany blok zużywa dużo elementów BRAM 49%, przez co na danym układzie istnieje możliwość implementacji około 2 bloków do szyfrowania (w optymistycznym przypadku). Dla danego układu uzyskano częstotliwość sygnału zegarowego równą 125 MHz.

Na podstawie uzyskanych danych została policzona wydajność danego układu, określona jako liczba zakodowanych wiadomości jaką może dany blok policzyć w ciągu jednej sekundy(H/s). Wydajność danego bloku to 125 MH/s.

$$W = 1/(T * L)[H/s]$$

W – wydajność
T –okres sygnału zegarowego
L –liczba cykli zegara pomiędzy nowymi danymi wejściowymi



Rysunek 2.11. Zajętość zasobów infrastruktury

Bibliografia

[1] NIST, “ADVANCED ENCRYPTION STANDARD (AES)”, https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf, 2001.

Wykaz symboli i skrótów

EiTI – Wydział Elektroniki i Technik Informacyjnych
PW – Politechnika Warszawska
WEIRD – ang. *Western, Educated, Industrialized, Rich and Democratic*

Spis rysunków

Spis tabel

Spis załączników