Przemysław Jesinowicz

Eryk Wawrzyn

Dokumentacja końcowa PUF

Temat:

Hashowanie danych za pomocą algorytmu SHA-256.

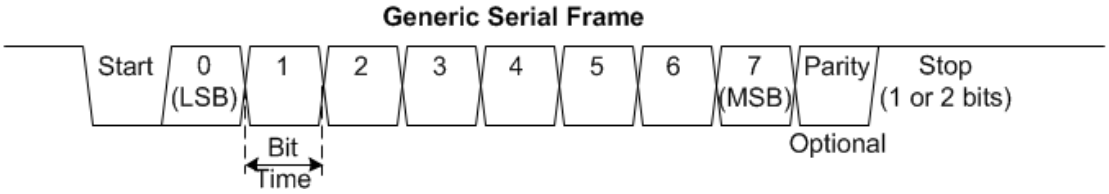
Funkcjonalności:

1. Odbieranie danych do zakodowania poprzez UART.
2. Przetwarzanie otrzymywanych danych.
3. Wysyłanie wyliczonych hash-y poprzez UART.

Analiza techniczna elementów systemu

- **Interfejs komunikacyjny**

Jako interfejs komunikacyjny wybrano UART, jest to asynchroniczny nadajnik (TX) – odbiornik (RX).



Rys. 1 Ramka danych UART

Wysyłana wiadomość zaczyna się od bitu „Startu”, który ma wartość logiczną równą zero (czyli stan niski), następnie wysyłane jest osiem bitów które są nośnikiem wiadomości i na końcu znajduje się „Stop”, który ma wartość logiczną odpowiadającą stanowi wysokiemu, może zajmować „długość” jednego bądź dwóch bitów (do projektu wybrano dwa bity).

Jeżeli nie jest wysyłana żadna wiadomość, to na „TX” pojawia się ciągle stan wysoki.

UART wykorzystuje do transmisji dwie linie:

- RX – linia służąca do odbierania wiadomości
- TX – linia służąca do nadawania wiadomości

Dane mogą być nadawane z różną prędkością transmisji („Baud rate”), najbardziej popularne to:

1200, 2400, 4800, 19200, 38400, 57600 i 115200.

Do projektu wybrana została prędkość: 19200 bps (bits per second).

- **Algorytm funkcji skrótu SHA-256**

Funkcja skrótu (funkcja haszująca) jest wykorzystywana do przyporządkowania dowolnie dużej liczbie klucza (hash-a) o stałym rozmiarze. Hash SHA-256 ma długość 256 bitów.

Realizacja algorytmu składa się z dwóch etapów:

1. Przygotowania.
2. Obliczania hash-a.

Etap 1.

Dopełnienie wiadomości, M, do wielokrotności 512 bitów:

- Dopisanie bitu „1” na końcu wiadomości.
- Dopisanie k bitów „0”, gdzie k jest najmniejszym dodatnim rozwiązaniem równania:

$$l + 1 + k \equiv 448 \, mod \, 512$$
$$\Downarrow$$
$$k = \big(448 - (l + 1)\big) \, mod \, 512$$

- Dopisanie 64-bitowego bloku równemu długości wiadomości, l, w systemie binarnym.

Podział na N 512-bitowych bloków, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$. Ponadto w każdej iteracji algorytmu blok jest dzielony na szesnaście 32-bitowych słów: $M_0^{(i)}, M_1^{(i)}, \dots, M_{15}^{(i)}$.

Przypisanie początkowej wartości hash-owi, $H^{(0)}$. Składa się on z ośmiu 32-bitowych słów uzyskanych jako pierwsze 32 bity części ułamkowej pierwiastka stopnia drugiego z ośmiu kolejnych liczb pierwszych. W zapisie heksadecymalnym:

$$H_0^{(0)} = 6a09e667$$
$$H_1^{(0)} = bb67ae85$$
$$H_2^{(0)} = 3c6ef372$$
$$H_3^{(0)} = a54ff53a$$
$$H_4^{(0)} = 510e527f$$
$$H_5^{(0)} = 9b05688c$$
$$H_6^{(0)} = 1f83d9ab$$
$$H_7^{(0)} = 5be0cd19$$

Etap 2.

Do obliczeń algorytm wykorzystuje następujące funkcje:

- $Ch(x, y, z) \ = \ (x \ and \ y) \ xor \ ((not \ x) and \ z)$
- $Maj(x, y, z) \ = \ (x \ and \ y) \ xor \ (x \ and \ z) \ xor \ (y \ and \ z)$
- $\sum_0(x) \ = \ ROTR_2(x) \ xor \ ROTR_{13}(x) \ xor \ ROTR_{22}(x)$
- $\sum_1(x) \ = \ ROTR_6(x) \ xor \ ROTR_{11}(x) \ xor \ ROTR_{25}(x)$
- $\sigma_0(x) \ = \ ROTR_7(x) \ xor \ ROTR_{18}(x) \ xor \ SHR_3(x)$
- $\sigma_1(x) \ = \ ROTR_{17}(x) \ xor \ ROTR_{19}(x) \ xor \ SHR_{10}(x)$
- $ROTR_n(x) = \ (x \ \gg \ n) \ or \ (x \ \ll (w - n))$,
  $w - długość \ słowa \ x \ w \ bitach \ (tutaj \ 32 - bity)$
- $SHR_n(x) \ = \ x \ >> \ n$

Każda operacja dodawania (+) przedstawiona w algorytmie jest wykonywana modulo 2^{32}.

Zasoby wymagane przez algorytm w każdej iteracji:

1. Tablica 64 32-bitowych słów: $W_0, W_1, \dots, W_{63}$.
2. Osiem 32-bitowych zmiennych: $a, b, \dots, h$.
3. Hash składający się z 8 32-bitowych słów: $H_0^{(i)}, H_1^{(i)}, \dots, H_7^{(i)}$.
4. Dwie 32-bitowe zmienne: T_1 i T_2.

W trakcie obliczeń wykorzystywane są również stałe, $K_0, K_1, \dots, K_{63}$, w zapisie heksadecymalnym:

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

Uzyskuje się je z części ułamkowej pierwiastka stopnia trzeciego kolejnych liczb pierwszych zachowując pierwsze 32 bity.

Obliczanie hash-a przedstawia się następująco:

Pętla dla każdego bloku wiadomości, $M^{(1)}, M^{(2)}, \dots, M^{(N)}$, od $i = 1$ do N:
{

1. Przygotowanie tablicy słów W_t:

$$W_t = \begin{cases} M_t^{(i)}, & dla\ 0 \le t \le 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}, & dla\ 16 \le t \le 63 \end{cases}$$

2. Zainicjowanie zmiennych:
$a = H_0^{(i-1)}$
$b = H_1^{(i-1)}$
$\vdots$
$h = H_7^{(i-1)}$
3. Pętla od $t = 0$ do 63:

{

$$T_1 = h + \sum_1(e) + Ch(e,f,g) + K_t + W_t$$

$$T_2 = \sum_2(a) + Maj(a,b,c)$$

$$h = g$$
$$g = f$$
$$f = e$$
$$e = d + T_1$$
$$d = c$$
$$c = b$$
$$b = a$$
$$a = T_1 + T_2$$

}

4. Obliczenie i-tej wartości pośredniej hash-a:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$\vdots$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

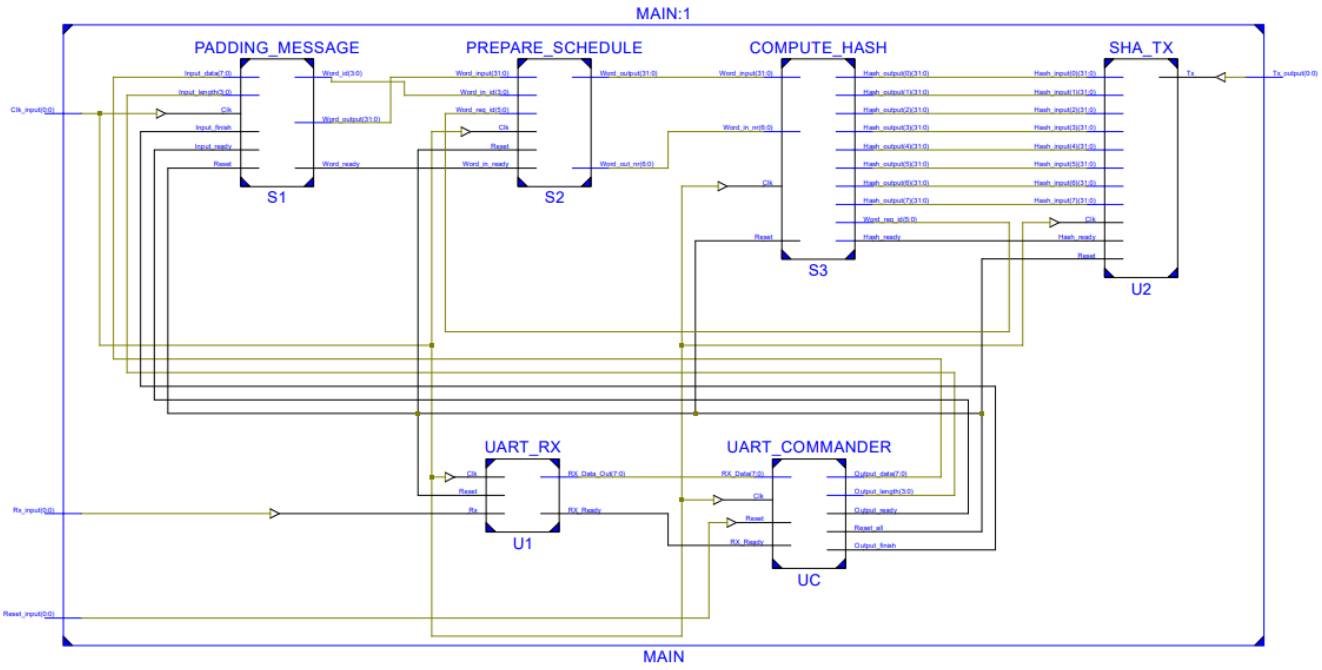
}

Po wykonaniu wszystkich iteracji N razy, otrzymany hash prezentuje się następująco:

$$H_0^{(N)}, H_1^{(N)}, H_2^{(N)}, H_3^{(N)}, H_4^{(N)}, H_5^{(N)}, H_6^{(N)}, H_7^{(N)}$$

Przedstawiono algorytm na podstawie: FIPS PUB 180-4

Uzyskane schematy RTL



Rys. 2 Uzyskany schemat RTL całego układu

Schematy pozostałych bloków

Opis sygnałów i portów Dokumentacja końcowa - kodu.pdf

Opis plików i bloków

- MAIN: Główny blok łączący moduły i przesyłający między nimi sygnały.

- UART_RX: Odbiornik UART wczytujący dane z zewnątrz.

- UART_COMMANDER: Nadzoruje blok UART_RX oraz interpretuje przychodzące dane. Rozróżnia komendy i przesyła dane do kolejnych bloków.

- SHA_PADDING MESSAGE: Pobiera dane, składa je w 32-bitowe słowa i przekazuje dalej. Na końcu danych dodaje odpowiednie ciągi bitów wyrównujące treść do 512 bitów.

- SHA_PREPARE_SCHEDULE: Indeksuje przychodzące słowa, podaje je do wewnętrznego modułu WORD_T i odbiera od niego wyliczone słowa. Obsługuje moduł SHA_COMPUTE_HASH podając mu słowa o wcześniej zadanym indeksie.

- WORD_T: Tworzy tablicę słów z przychodzących danych i wylicza na ich podstawie kolejne. Informuje o liczbie wyliczonych i gotowych do odczytania słów.

- SHA_COMPUTE_HASH: Pobiera słowa o zadanym indeksie od poprzedniego modułu oraz realizuje główną część algorytmu, który oblicza funkcje skrótu (hash-e). Na koniec informuje czy obliczony hash jest gotowy do wysłania

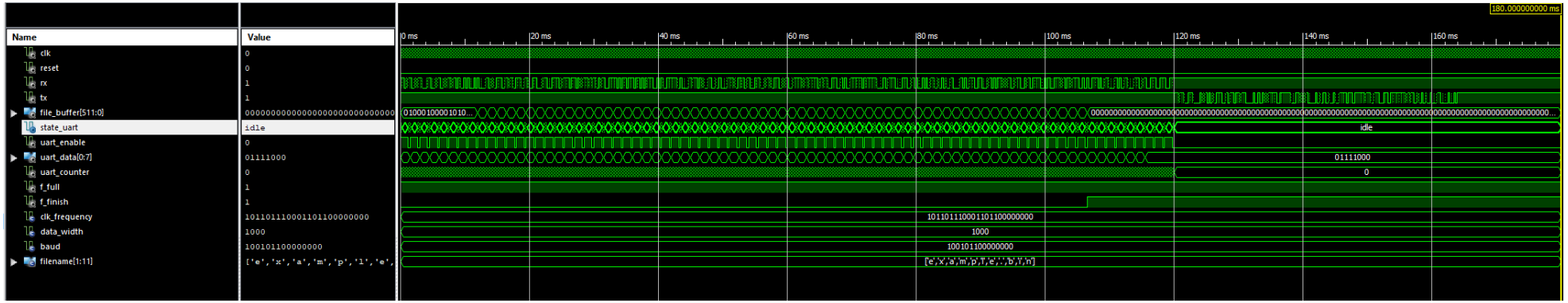
- SHA_TX: Blok odpowiadający za podział hash-a (zajmującego 256 bitów) po osiem bitów w celu przygotowania do wysłania przez UART (wykonano)

- UART_TX: Nadajnik (TX) UART (zgonie z opisem znajdującym się w zakładce „Interfejs komunikacyjny”) (wykonano)

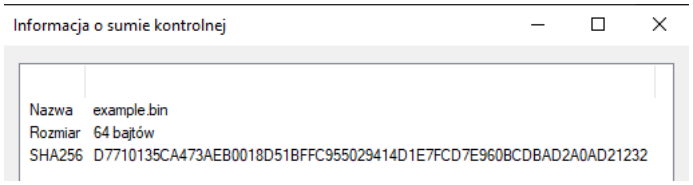
- sha_function: Funkcje wykorzystywane przez koder SHA-256 znajdujące się w pliku package

- constants: Deklaracja stałych wykorzystywanych przez koder SHA-256, się w pliku package

Wyniki symulacji



Rys. 3 Wynik symulacji układu

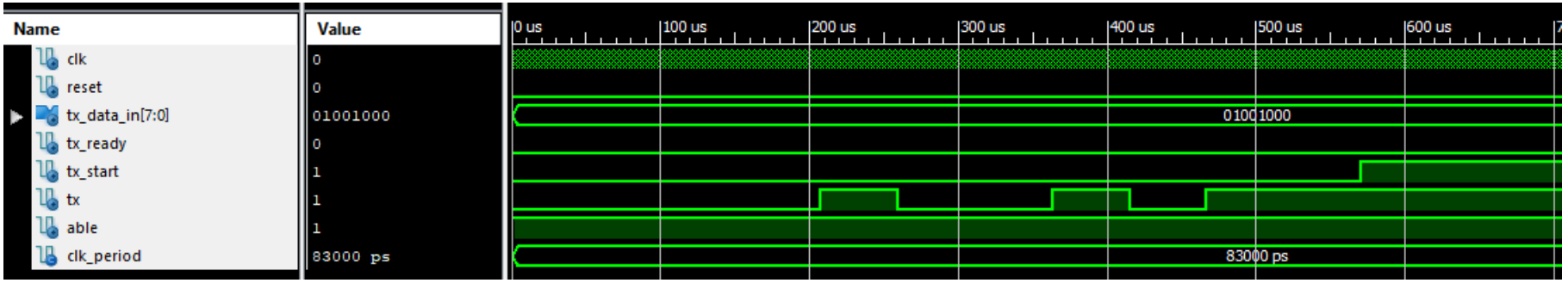


Rys. 4 Prawidłowy wynik kodowani

Z przeprowadzonego badania wynika, że układ działa prawidłowo „RX” poprawnie wczytuje informacje, a na „TX” uzyskujemy zgodny z oczekiwaniami wynik kodowania. Obliczanie hashu sha256 przez układu jest bardzo szybkę, największym „buforem” jest przesyłanie i wprowadzanie wiadomości przez powolny UART.

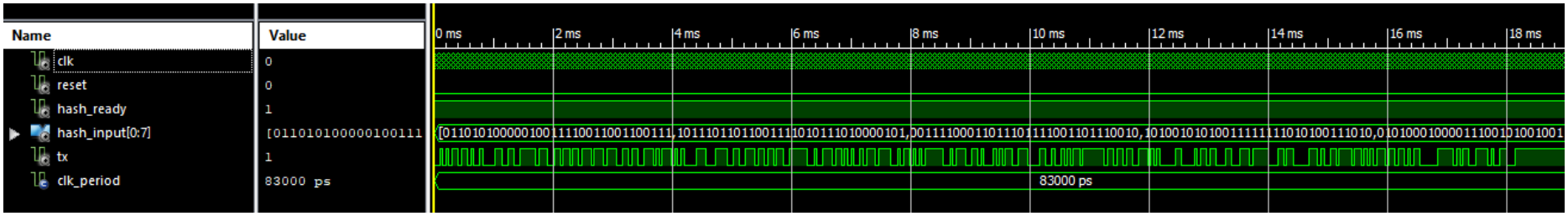
Nazwy sygnałów są zgodne z opisem znajdującym się w Dokumentacja końcowa - kodu.pdf.

Dodatkowe kryteria



Rys. 5 Symulacja UART TX

Na magistrali UART szeregowo jest transmitowana wartość 18 (tx_data_in 00010010), na magistrali równoległej poprawnie otrzymano wysłaną wartość (tx). Czas nadawania jednego bitu wynosi 52 µs, więc osiągnięto zakładany Baud rate 19200.



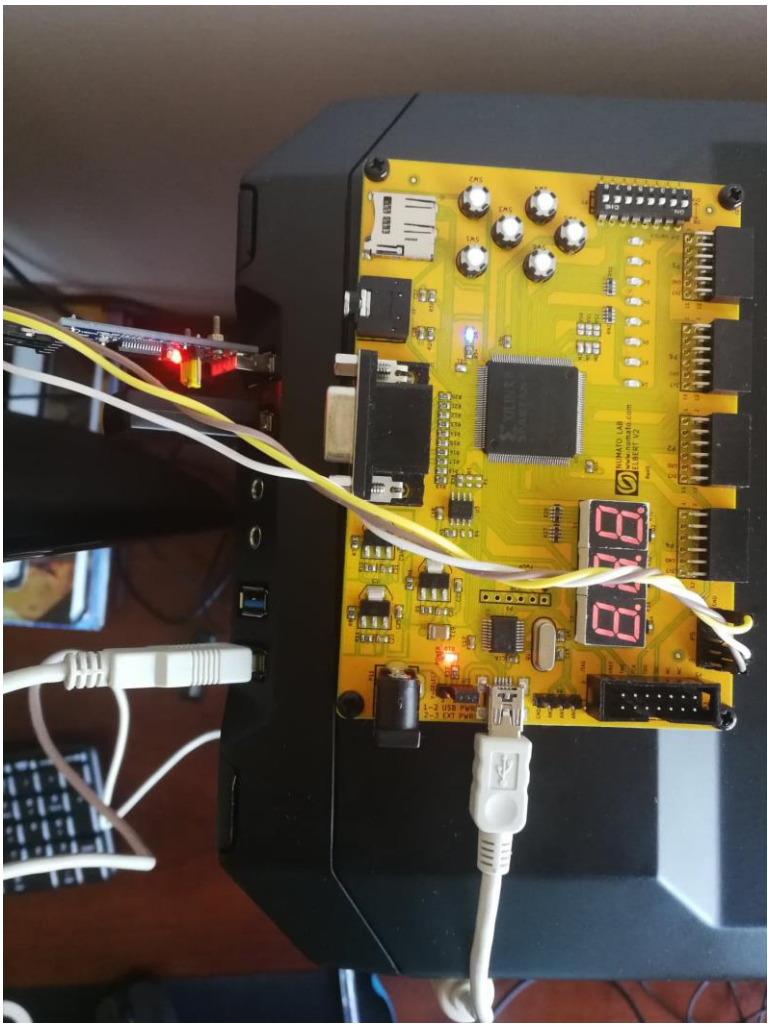
Rys. 6 Symulacja SHA TX

Z przeprowadzonej symulacji wynika, że blok działa prawidłowo. Dane zostały prawidłowo podzielone po osiem bitów, blok „wprowadza” dane do UART_TX dopiero gdy TX prześle poprzednią wiadomość.

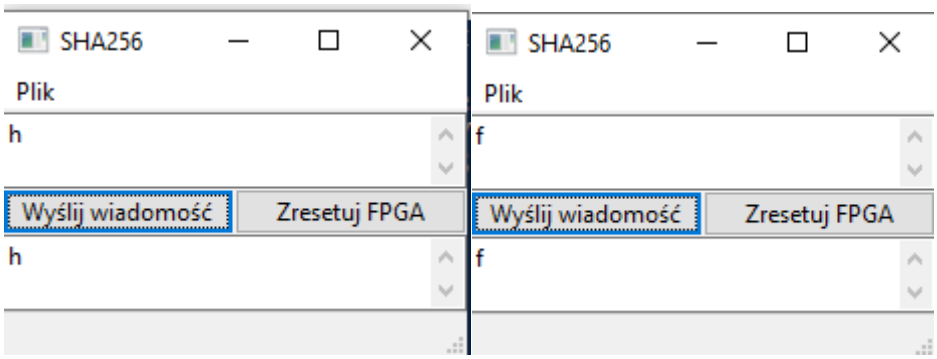
Nazwy sygnałów są zgodne z opisem znajdującym się w Dokumentacja końcowa - kodu.pdf.

Echo UART

Podstawowy układ (układ „funkcji skrótu sha256”) okazał się zbyt wymagający dla płytki rozwojowej „FPGA Elbert v2 - Spartan 3A”. Aby zademonstrować działanie płytki rozwojowej FPGA, skonfigurowano dodatkowy układ który pobiera dane a następnie zwraca odrzyna wiadomość (przez UART).



Rys. 7 Badana płytka rozwojowa FPGA podłączona do komputera



Rys. 6 Przykładowe uzyskane wyniki, górne pole tekstowe - wiadomości przeznaczonej do wysłania, dolne pole tekstowe – odszywana wiadomość zwrotna

Z przeprowadzonego badania wynika, że układ działa prawidłowo. Dane są transmitowane z odpowiednią częstotliwością, odpowiedzi zwrotną otrzymano prawidłową.

Dokumentacja kodu powyższego układu - Dokumentacja.pdf