AKADEMIA GÓRNICZO-HUTNICZA

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI
KIERUNEK MIKROELEKTRONIKA W TECHNICE I MEDYCYNIE

SYSTEMY DEDYKOWANE W UKŁADACH PROGRAMOWALNYCH

Projekt zaliczeniowy

Implementacja kodowania długości serii (RLE) w układzie z rodziny Zynq-7000

Agnieszka Kamień i Magdalena Rosół

Kraków, 2021-04-28

Spis treści

1 Historia zmian dokumentu **3**

2 Cel projektu **4**

3 Opis algorytmu **4**

4 Diagram algorytmu **4**

5 Ustalenie architektury modułu **7**

6 Kod w .v/.sv **7**

7 Testy modułu behawioralnego **8**

8 Potokowa wersja modułu **9**

9 Kod w .v/.sv **9**

10 Testy modułu syntezowalnego **9**

11 Inkorporacja modułu do większego systemu **10**

11.1 Magistrala AXI 10

11.2 Sterownik 10

1 Historia zmian dokumentu

Wersja dokumentu	Data	Opis
1.0	22.04.2021	Pierwsza wersja dokumentu
2.0	29.04.2021	Opis pierwszej wersji modułu behawioralnego

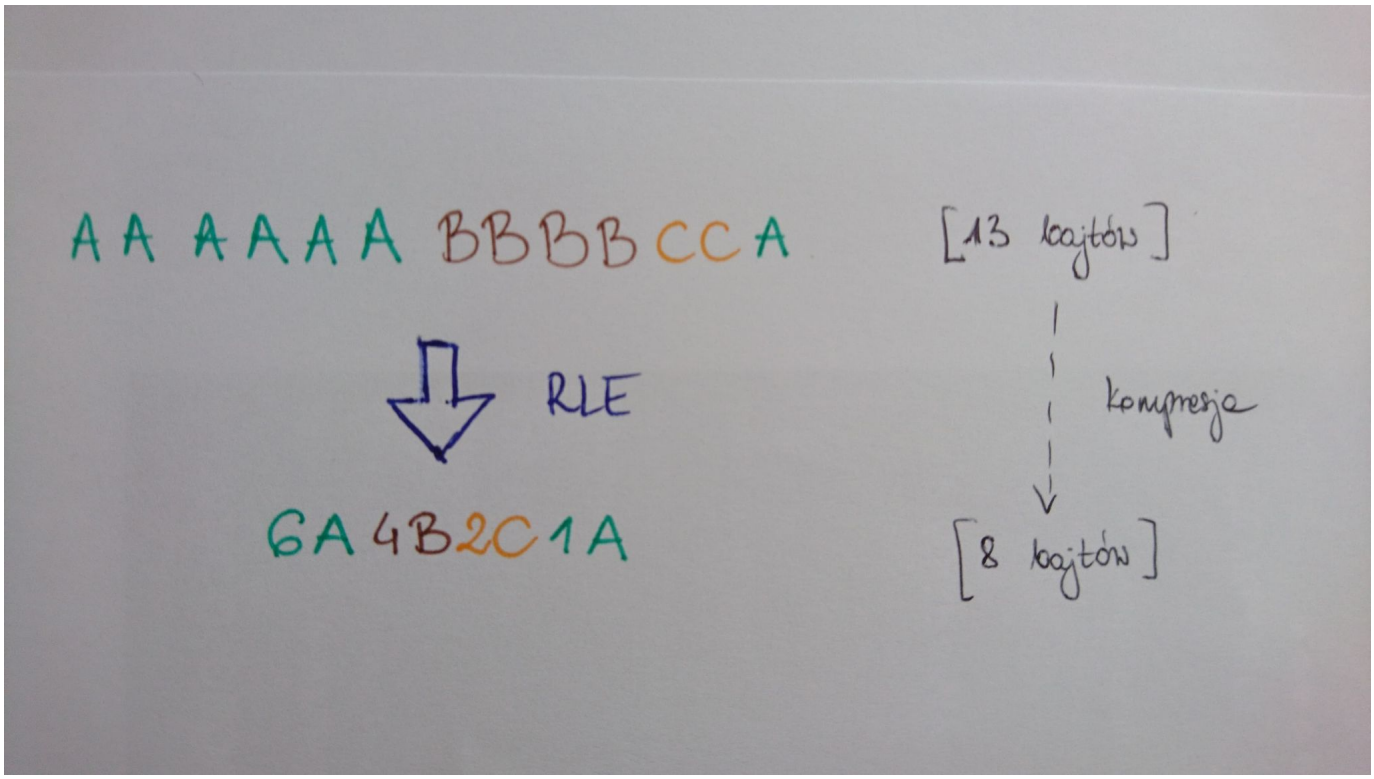
2 Cel projektu

Celem niniejszego projektu jest opracowanie modułów enkodera i dekodera realizujących kodowanie długości serii. Moduły te mają zostać napisane w języku opisu sprzętu Verilog bądź SystemVerilog, a docelowo powinny zostać uruchomione na płytce ZedBoard Zynq-7000 firmy Xilinx.

3 Opis algorytmu

Kodowanie długości serii (ang. Run-Length Endcoding, RLE) jest formą bezstratnej kompresji danych. Oznacza to, że skompresowane dane mogą zostać z powrotem przekonwertowane do dokładnie takiej samej postaci jak w reprezentacji oryginalnej. Żadne informacje nie są tracone podczas kompresji - proces jest w pełni odwracalny.

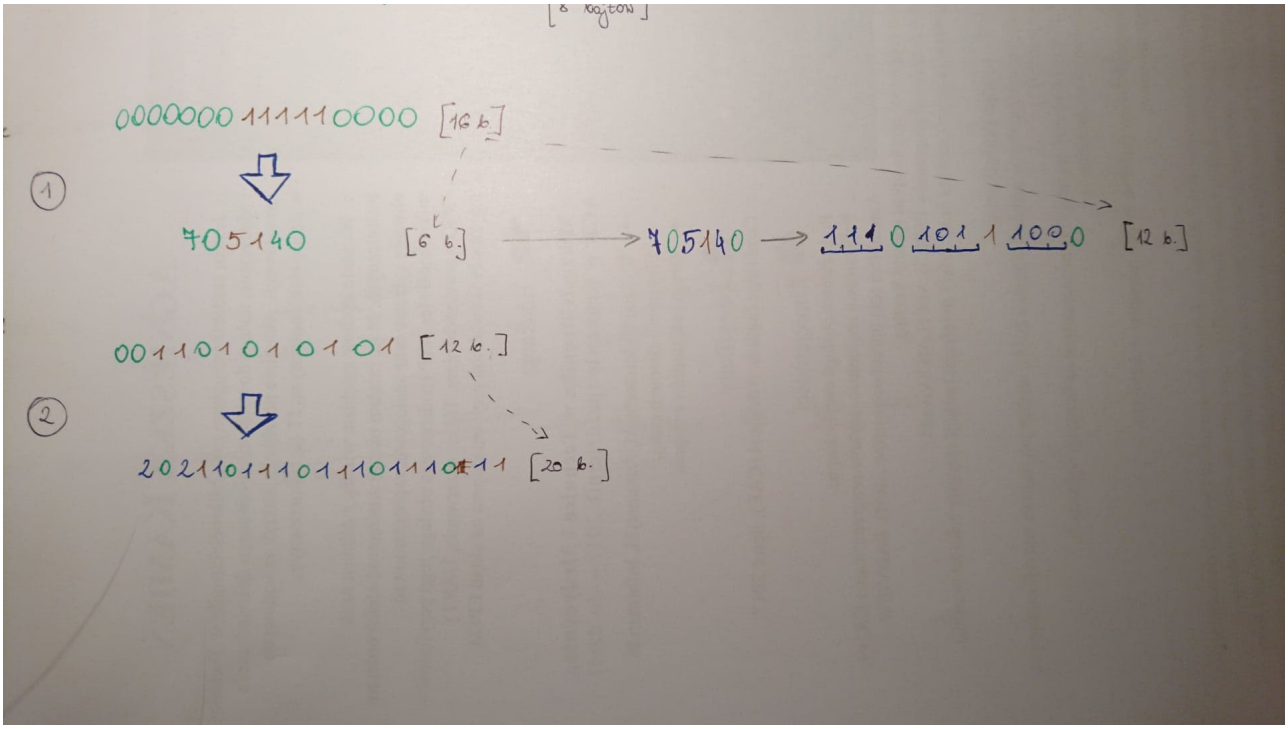
Zdecydowaną zaletą algorytmu jest łatwość jego implementacji oraz to, że nie wymaga dużej ilości zasobów CPU. Polega on na na zastąpieniu ciągu znaków liczbą wystąpień danego znaku oraz jego symbolem (patrz rys. 1). Kompresja jest tym skuteczniejsza, im dane są bardziej powtarzalne - wielokrotnie powtarzający się „kolejno" bajt można zapisać w zaledwie dwóch bajtach. W najgorszym przypadku rozmiar danej może zwiększyć się aż dwukrotnie (np. podczas zapisu ciągu ABCD po kompresji otrzymuje się 1A1B1C1D).



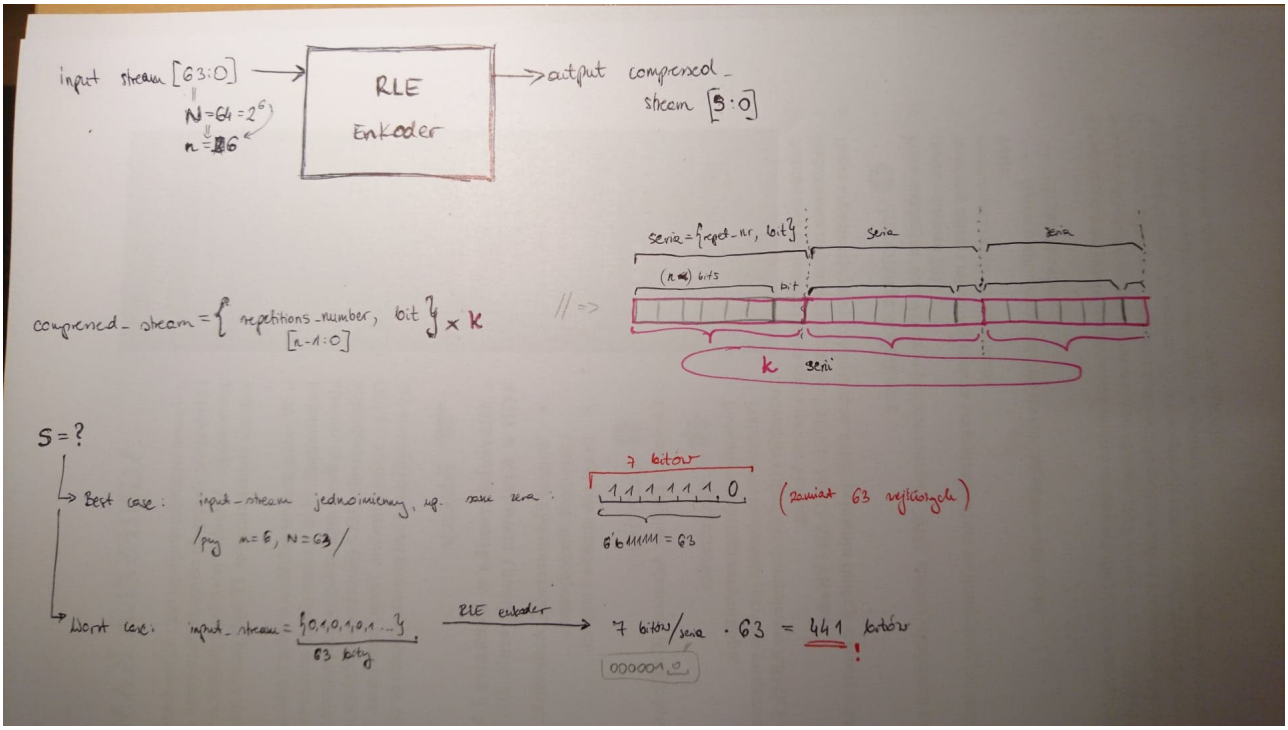
Rysunek 1. Przykład kodowania RLE

4 Diagram algorytmu

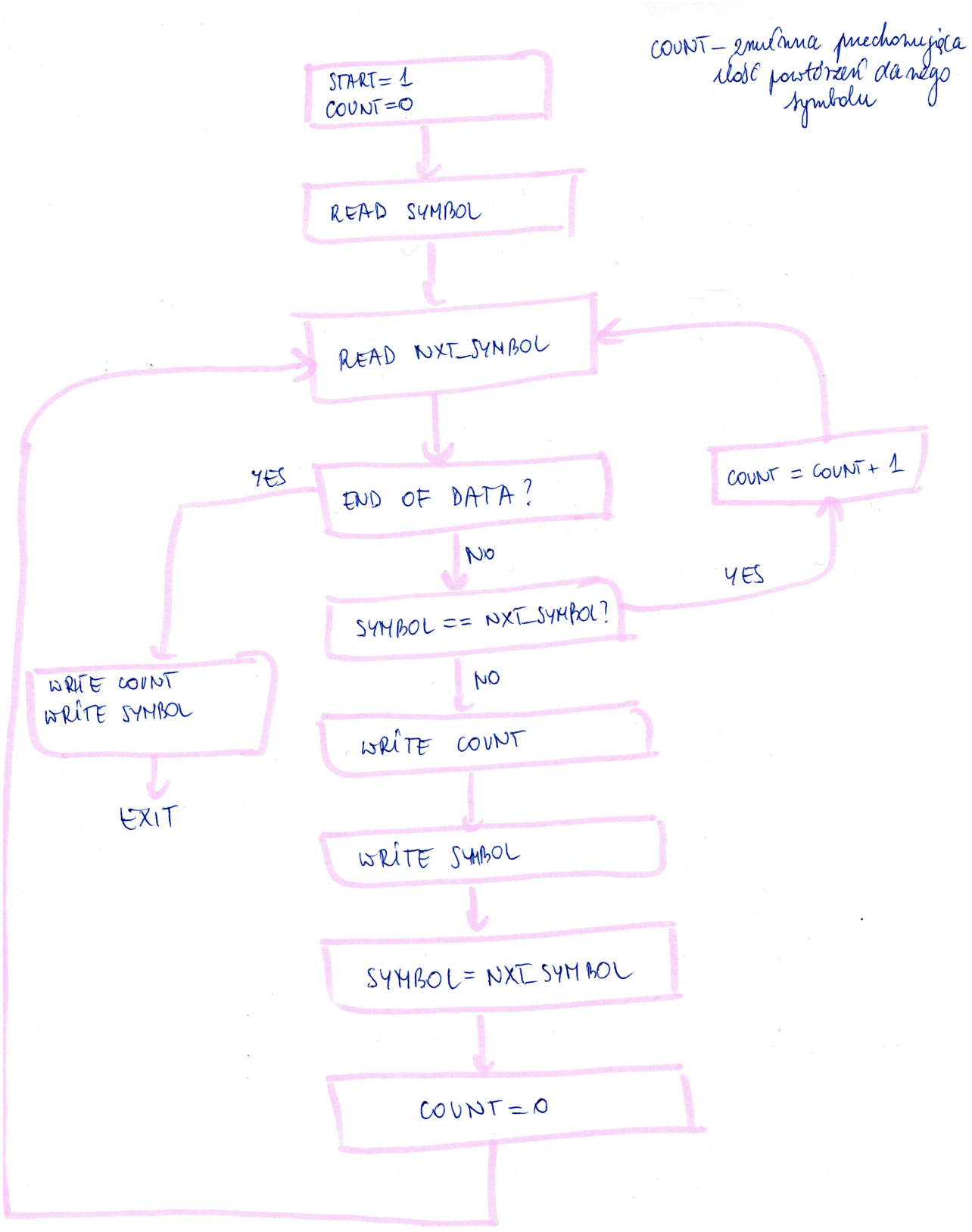
Jeżeli długość serii kodujemy na n bitach, to w sekwencji wejściowej możemy mieć 2^n znaków (zer lub jedynek). Wykorzystamy to, że nigdy nie kodujemy długości serii równej zero. Zatem symbol równy 0, możemy wykorzystać dla serii 16-znakowej.



Rysunek 2. RLE - przykład kodowania strumienia bitów



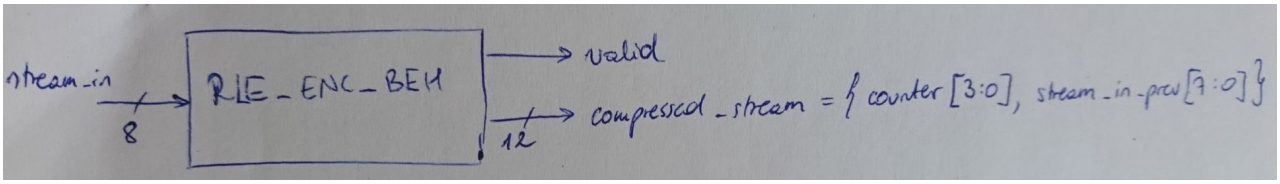
Rysunek 3. RLE Enkoder – pierwszy zamysł



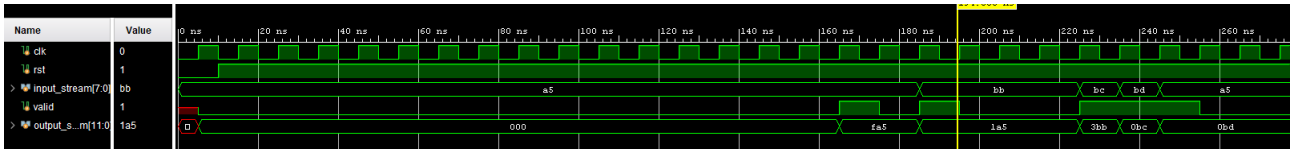
Rysunek 4. Schemat blokowy działania algorytmu

5 Ustalenie architektury modułu

W pierwszym podejściu zaimplementowano moduł enkodera RLE bez parametrów. Przyjmuje on na wejście 8-bitowy strumień danych *stream_in*. Dopóki w kolejnych taktach zegara na wejściu enkodera pojawia się taki sam strumień, wewnątrz modułu inkrementowany jest licznik, a wyjście *valid* znajduje się w stanie niskim. Gdy licznik się przepełni lub *stream_in* ulegnie zmianie, sygnał *valid* ustawiany jest w stan wysoki, a na wyjściu *compressed_stream* pojawiają się dane zakodowane długością serii. W tej wersji modułu zastosowano licznik 4-bitowy, zatem jego zakres mieści się od wartości 0 do wartości 15. Po szesnastokrotnym wystąpieniu serii na wyjściu pojawia się informacja o wystąpieniu takiej sekwencji, a licznik zaczyna na nowo liczyć od 0. Przykład takiego przebiegu zaprezentowano na rys. 6. Warto przypomnieć o przyjętej konwencji zliczania – by wykorzystać maksymalnie zakres licznika, wszystkie wartości, które może on przyjąć, są wykorzystywane. Jednak wartości te są pomniejszone o jeden względem rzeczywistej liczby wystąpień serii. Dla zobrazowania: jednokrotne pojawienie się na wejściu enkodera strumienia 0xBC, poskutkuje wyjściem 0x**0**BC, a nie – jak można by się spodziewać – 0x**1**BC. Z kolei szesnastokrotne (d'16=h'F) wystąpienie serii 0xCC, znajdzie odzwierciedlenie na wyjściu postaci 0xFCC.



Rysunek 5. Prototyp enkodera RLE



Rysunek 6. Przykładowy wynik symulacji modułu behawioralnego enkodera RLE

Kolejnym krokiem była parametryzacja modułu enkodera. Dla parametrów dobranych jak w wyżej opisanym module (bez parametryzacji) efekt symulacji był taki jak na rys. 6. Kod sparametryzowanego modułu umieszczono w podrozdziale 6

6 Kod w .v/.sv

```
module rle_encoder_beh
(
    input clk,
    input rst,
    input [DATA_WIDTH - 1: 0] stream_in,
    output reg [DATA_WIDTH + CTR_WIDTH - 1 :0] compressed_stream,
    output reg valid
);

parameter DATA_WIDTH=8,
          CTR_WIDTH=4;

localparam CTR_MAX = (1<<CTR_WIDTH) - 1;
```

```
14
15 reg [DATA_WIDTH-1:0] stream_in_prev;
16 reg [CTR_WIDTH:0] seq_counter = 0;
17
18 always @(posedge clk)  begin
19     stream_in_prev <= stream_in;
20     if (!rst) begin
21         valid <= 1'b0;
22         compressed_stream <= 0;
23     end else begin
24         if (stream_in == stream_in_prev) begin
25             if (seq_counter != CTR_MAX) begin
26                 valid <= 1'b0;
27                 seq_counter <= seq_counter + 1;
28             end else begin
29                 valid <= 1'b1;
30                 compressed_stream <= {seq_counter[CTR_WIDTH-1:0], stream_in_prev};
31                 seq_counter <= 0;
32             end
33         end else begin
34             valid <= 1'b1;
35             compressed_stream <= {seq_counter[CTR_WIDTH-1:0], stream_in_prev};
36             seq_counter <= 0;
37         end
38     end
39 end
40
41 endmodule
```

Listing 1. RLE encoder

7 Testy modułu behawioralnego

8 Potokowa wersja modułu

9 Kod w .v/.sv

10 Testy modułu syntezowalnego

11 Inkorporacja modułu do większego systemu

11.1 Magistrala AXI

11.2 Sterownik

Symulacja