AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Informatyki, Elektroniki i Telekomunikacji

RLC - Run Length Coding

Autor:	Norbert Ligas, Krzysztof Pokora
Kierunek studiów:	Elektronika i Telekomunikacja
Opiekun pracy:	dr hab. inż. Paweł Russek

Kraków, 9 czerwca 2021

OŚWIADCZENIE AUTORA PRACY

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

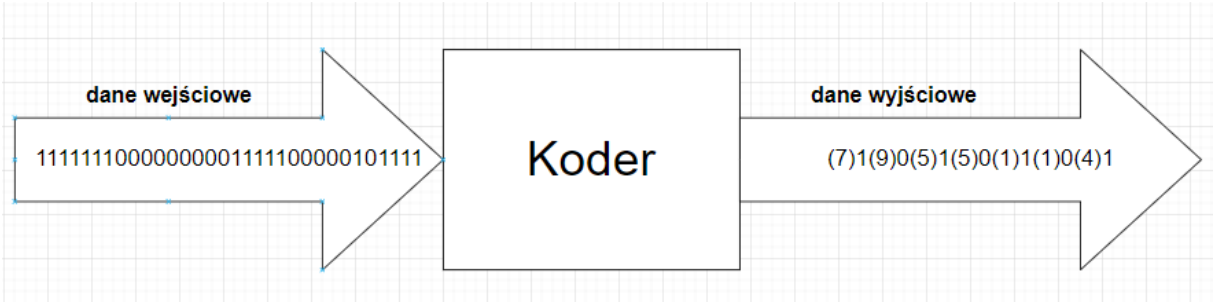
. . . . . .

PODPIS

Spis treści

1. Wstęp.

RLC w rozwinięciu brzmi Run Length Coding i jest znane również pod nazwą Run Length Encoding(RLE). Jest to kompresja polegająca na zastępowaniu długich ciągów znaków znakiem i liczbą jego powtórzeń. Schemat działania przykładowego kodera, można zaobserwować na Rys. 1.1.



Rys. 1.1: Schemat działania kodera.

Dekoder RLC działa w sposób dokładnie odwrotny do kodera, czyli pobierając odpowiednio informacje wstawia odpowiednie ciągi na wyjściu.

2. Założenia.

Projektując koder założyliśmy najpierw, że dane na wejściu oraz wyjściu podawane będą równolegle w odpowiednio dużych paczkach. Jako, że od strony kodera i dekodera im więcej podanych danych, tym teoretycznie lepiej, podjęto decyzję o wykorzystaniu maksymalnej szerokości wektora danych w systemie, który jest 32 bitowy. Dane wewnątrz modułów przetwarzane są sekwencyjnie. Do wyboru była również implementacja równoległa, ale ten sposób mimo, że wykonywał się w pojedynczym takcie zegara, potrzebował więcej czasu podczas pojedynczego taktu, nie licząc ilości zużytych zasobów. Metoda sekwencyjna potrzebowała mniej zasobów i nie wprowadzała problemu związanego z prędkością zegara.

W trakcie projektowania systemu należało zwrócić uwagę na fakt, że zakodowane dane trzeba przesłać w sposób zrozumiały dla dekodera. Zdecydowano, że informacja o znaku zakodowana będzie odpowiednio na dwóch bitach, a informacja o jego ilości na trzech. Znak kodowany był zgodnie z tabelą Tab. 2.1. Jak widać w tabeli, wzięty pod uwagę był przypadek kodowania pojedynczych bitów. W celu oszczędności bitów wyrzucanych potem na wyjście (w końcu oczekiwanym efektem jest kompresja danych) pojedyncze znaki kodowane są w ten sposób, że nie jest potrzebne podawanie informacji o ich ilości.

	1	0
Pojedynczy znak	11	00
Od dwóch do dziewięciu znaków	10	01

Tab. 2.1: Kodowanie znaku za pomocą dwóch bitów.

Do założeń projektowych należało:
- koder oraz dekoder powinny przyjmować po 32 bity równolegle,
- zwracane przez moduły dane powinny być również 32 bitowe podawane równolegle,
- ciągi znaków oznaczone są zgodnie z Tab. 2.1
- zakodowanie ilości znaków za pomocą trzech bitów,
- realizacja kodowania i dekodowania binarnego.

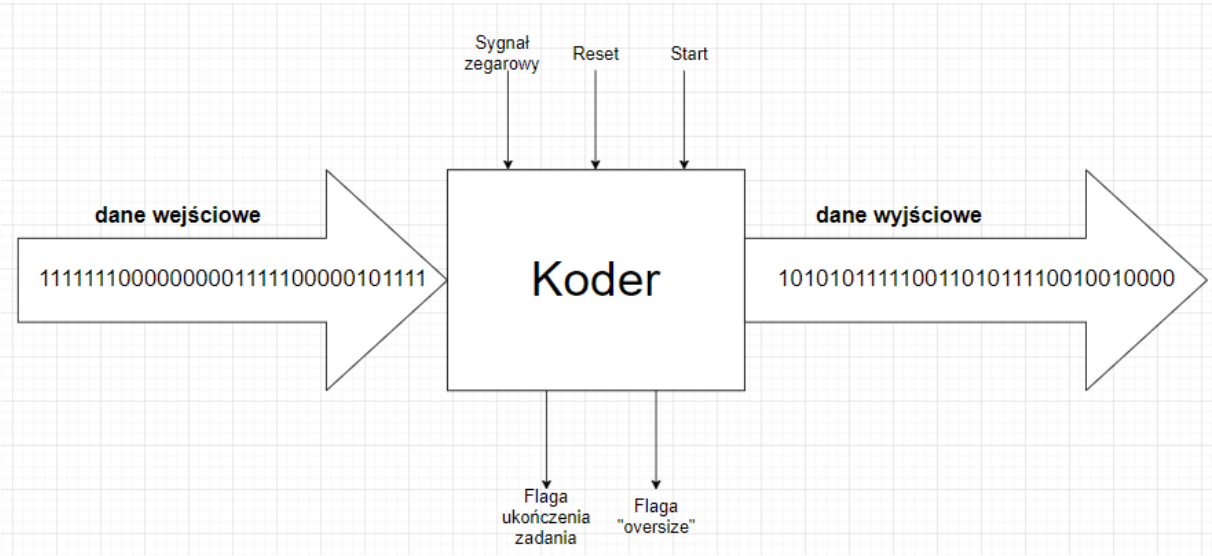
Do głównych założeń związanych ze sposobem realizacji projektu należało:
- zrealizować system na FPGA w języku opisu sprzętu Verilog oraz System Verilog,
- zrealizować koder RLC,
- zrealizować dekoder RLC,
- napisać odpowiednie kody testowe,

- połączyć moduły w spójny system z soft procesorem MicroBlaze,
- napisać kody testowe dla spójnego systemu z MicroBlaze,
- napisać kod na soft procesor MicroBlaze,
- połączyć moduły w spójny system z procesorem ZynQ,
- zaprojektować interfejs użytkownika, pomagający w pracy z systemem po podłączeniu do komputera poprzez USB i komunikację UART.

3. RTL.

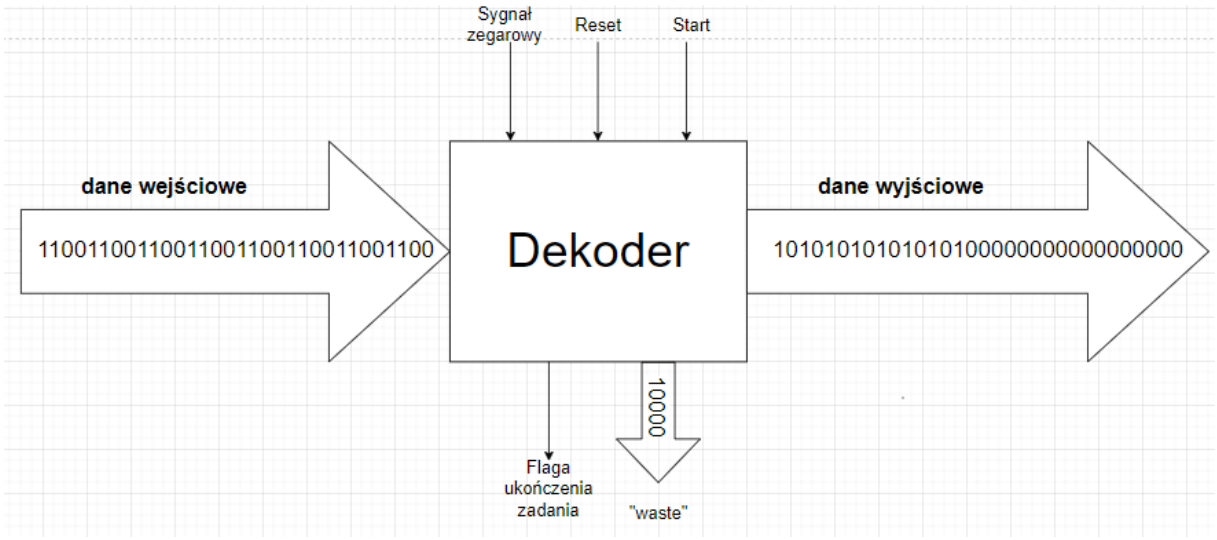
Pracę rozpoczęliśmy od napisania modułu przeznaczonego do kodowania danych. Za pomocą parametrów określone zostały szerokość danych, ilość bitów przeznaczona na zakodowanie znaku oraz jego ilości. Wstępnie przyjęte wartości to 32 bity dla danych wejściowych i wyjściowych, dwa przeznaczone do zakodowania znaku i trzy do zakodowania jego ilości.

W trakcie projektowania systemu natknęliśmy się na problem nieopłacalności kodowania w niektórych przypadkach. Chodzi o fakt, że celem do osiągnięcia jest otrzymanie skompresowanych danych. Problem został rozwiązany poprzez zwracanie flagi ”oversize”, która oznacza, że dane po kompresji zajmują więcej miejsca binarnie niż przed. Zwracane dane na wyjściu są wtedy niekompletne. Kompresja powinna być uważana za nieważną. Schemat kodera oraz dekodera można przedstawić za pomocą Rys. 3.1. Koder przyjmuje dane wejściowe, a następnie sekwencyjnie je przetwarza zliczając ilość danego znaku i odpowiednio kodując wyjście. W przypadku, gdy kompresja nie jest opłacalna i wyjście nie zmieści się na 32 bitach kodowanie jest przerywane i wystawiana jest flaga ”oversize” oraz flaga ukończenia zadania. Koder można również zresetować, natomiast po podaniu danych na rejestr należy go włączyć za pomocą flagi ”Start”. Poprawna kompresja sygnalizowana jest gotowością wyjścia poprzez flagę ukończenia zadania oraz wyzerowaną flagę oversize.



Rys. 3.1: Schemat działania kodera.

Dekoder został zrealizowany zgodnie ze schematem przedstawionym na Rys. 3.2. Podobnie jak koder posiada flagę startu, resetu oraz potrzebuje sygnału zegarowego do sekwencyjnej realizacji dekodowania. Przyjmowane dane są 32 bitowe, podobnie jak wyjście i podawane są równolegle. Ukończenie zadania sygnalizowane jest flagą ukończenia zadania. Dodatkowo w odróżnieniu do kodera, dekoder posiada pięcio-bitowy wektor danych podawany na wyjście, które informują ile z bitów na końcu podanego wektora zostały dołożone w celu dopełnienia wektora 32 bitowego.



Rys. 3.2: Schemat działania dekodera.

Stworzone moduły należało odpowiednio przetestować na różnego rodzaju danych wejściowych. W tym celu napisane zostały testy, które uruchamiały moduły kilkakrotnie. Ilość przeprowadzonych testów to parametr, tak samo jak i wektory testowe, które można łatwo dołożyć do testów. Na ten moment wektorami testowymi dla kodera i dekodera oraz spodziewanymi odpowiedziami były wektory zgodnie z Tab. 3.1 oraz Tab. 3.2.

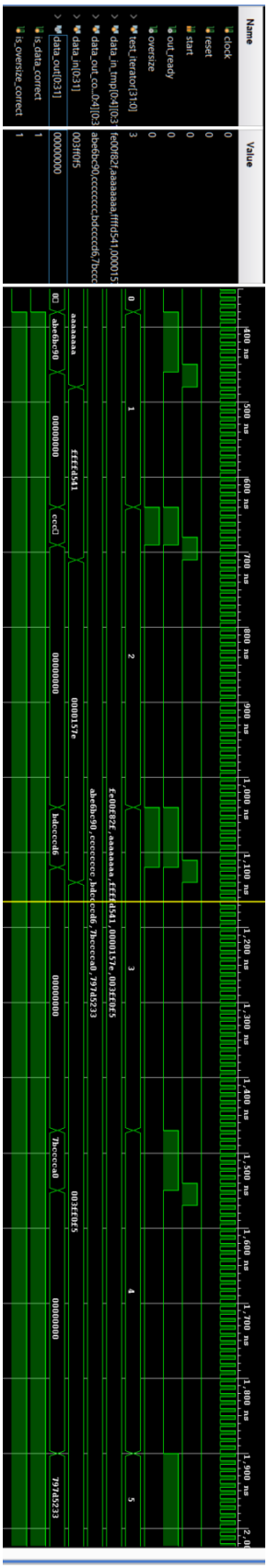
It	dane wejściowe	dane wyjściowe	“oversize”
1	11111110000000001111100000101111	10101011111001101011110010010000	0
2	1010_1010_1010_1010_1010_1010_1010_1010	1100_1100_1100_1100_1100_1100_1100_1100	1
3	11111111_11111111_0101_0101_00000_1	10111_10111_00_11_00_11_00_11_00_11_01011_0	1
4	000000000_000000000_0101010_111111_0	01111_01111_00_11_00_11_00_11_00_10100_00_0	0
5	000000000_0_111111111_1_0000_1111_0101	01111_00_10111_11_01010_10010_00_11_00_11	0

Tab. 3.1: Wektory testowe i spodziewane odpowiedzi testów kodera.

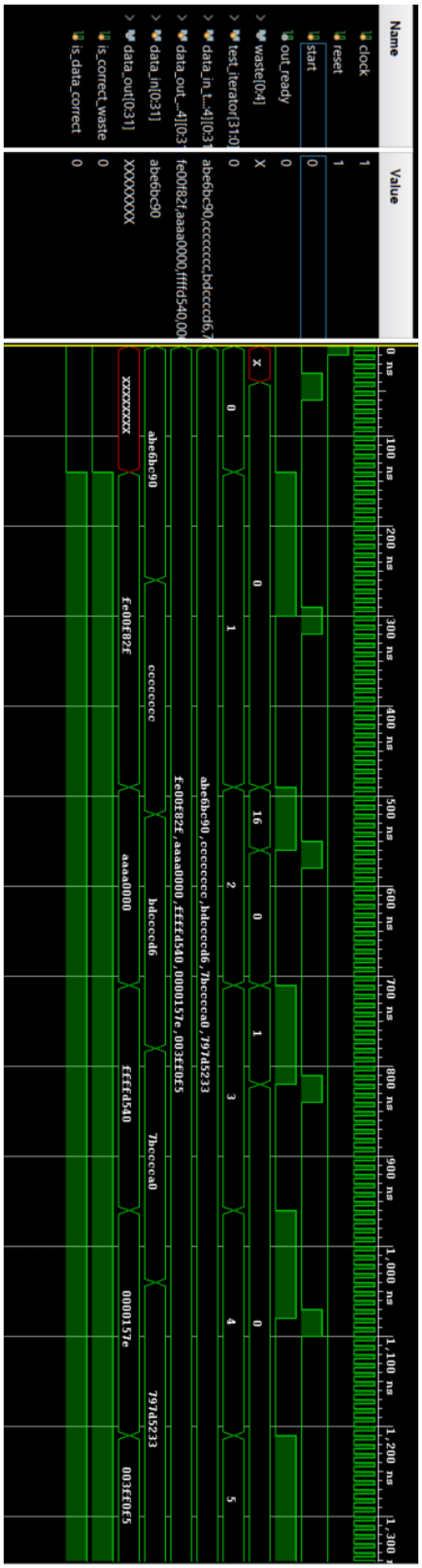
Wyniki przeprowadzonych symulacji zamieszczone są na Rys. 3.3 oraz Rys. 3.4.

It	dane wejściowe	dane wyjściowe	“waste”
1	10101_01111_10011_01011_11_00_10010_000	1111111_000000000_11111_00000_1_0_1111	00000
2	1100_1100_1100_1100_1100_1100_1100_1100	1010_1010_1010_1010_0000_0000_0000_0000	10000
3	10111_10111_00_11_00_11_00_11_00_11_01011_0	111111111_111111111_0101_0101_00000_0	00001
4	01111_01111_00_11_00_11_00_11_00_10100_00_0	000000000_000000000_0101010_111111_0	00000
5	01111_00_10111_11_01010_10010_00_11_00_11	000000000_0_111111111_1_0000_1111_0101	00000

Tab. 3.2: Wektory testowe i spodziewane odpowiedzi testów dekodera.



Rys. 3.3: Wyniki symulacji kodera.



Rys. 3.4: Wyniki symulacji dekodera.

4. MicroBlaze.

Po zrealizowaniu odpowiednich modułów przystąpiliśmy do połączenia ich z softprocesorem MicroBlaze w jeden system. W tym celu należało najpierw przypisać porty skonstruowanych modułów do rejestrów używanych w pamięci. Przypisanie portów jest opisane odpowiednio dla kodera i dekodera za pomocą tabel Tab. 4.1 oraz Tab. 4.2.

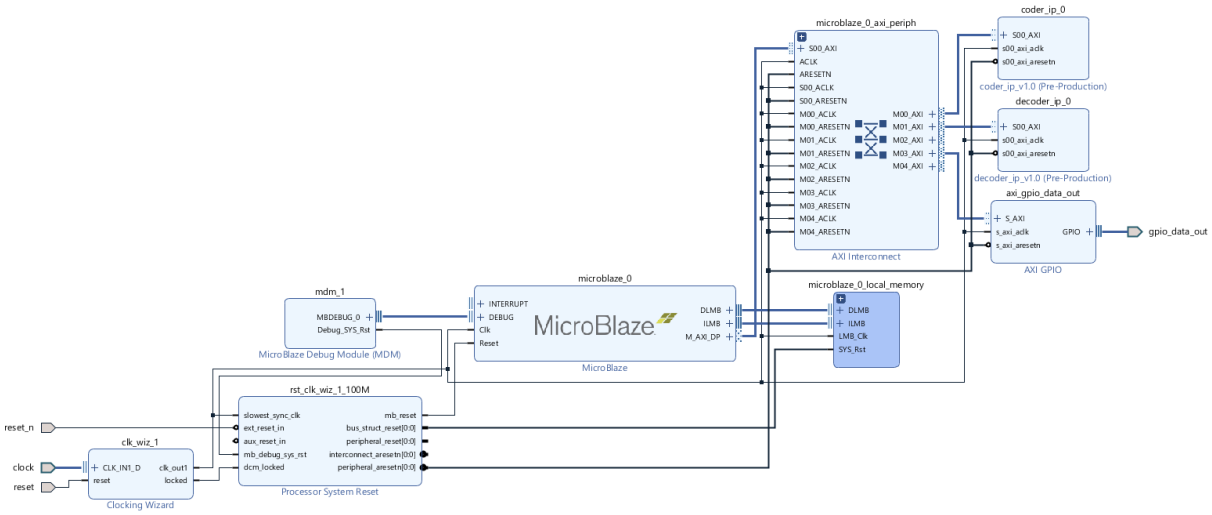
Porty	Rejestry	Kierunek danych	Szerokość portu
start	slv_reg0[0]	input	1
out_ready	slv_reg2[0]	output	1
oversize	slv_reg2[1]	output	1
datain	slv_reg1	input	32
dataout	slv_reg3	output	32

Tab. 4.1: Mapowanie portów kodera na rejestry w systemie.

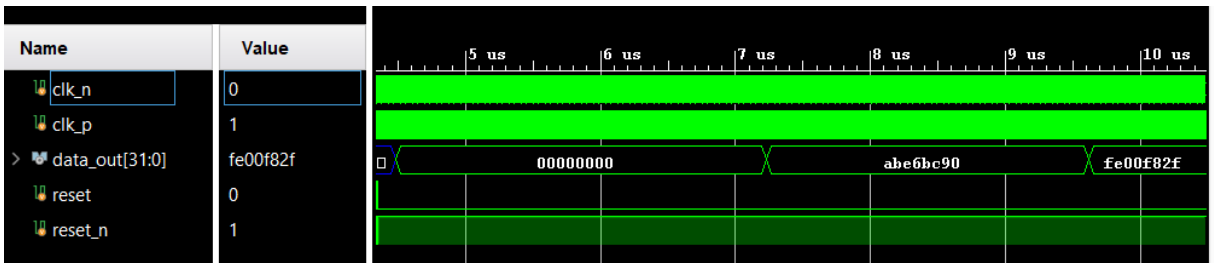
Porty	Rejestry	Kierunek danych	Szerokość portu
start	slv_reg0[0]	input	1
out_ready	slv_reg2[0]	output	1
waste	slv_reg2[5:1]	output	5
datain	slv_reg1	input	32
dataout	slv_reg3	output	32

Tab. 4.2: Mapowanie portów dekodera na rejestry w systemie.

Po odpowiednim skonfigurowaniu nowo powstałego IP, dołożone zostały pozostałe elementy systemu m. in. MicroBlaze. Schemat całego systemu wyszedł jak na Rys. 4.1. Oprócz skonstruowanych IP i Microblaze oraz dodanych automatycznie elementów w systemie znalazł się też pojedynczy IP nazwany GPIO umożliwiający odczyt danych w testach na całym systemie. Symulacje wyszły jak na Rys. 4.2.



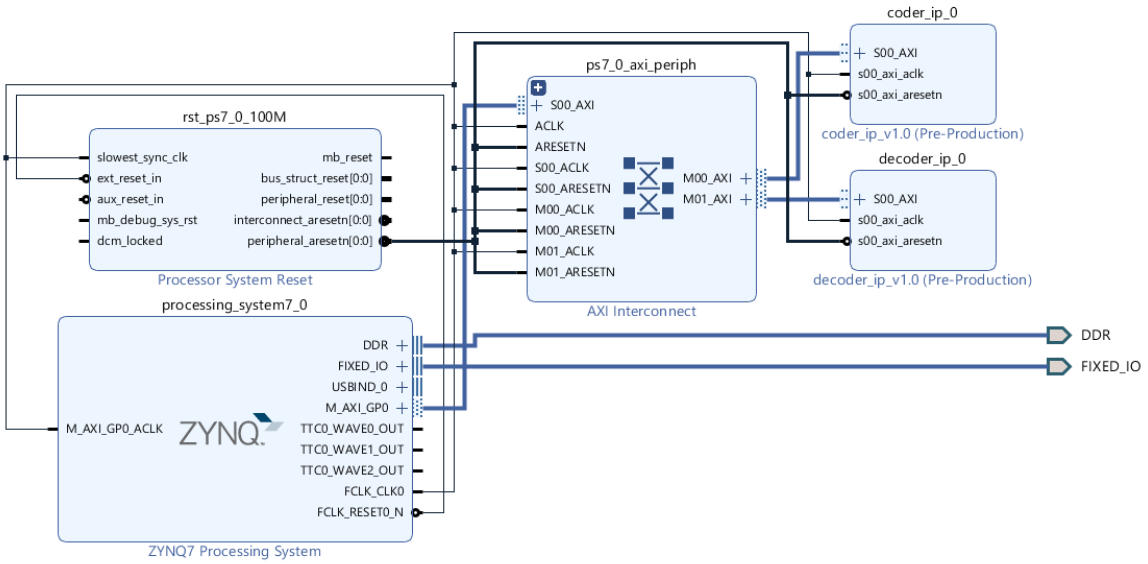
Rys. 4.1: Schemat systemu z procesorem ZynQ.



Rys. 4.2: Symulacje systemu.

5. ZynQ.

Po realizacji systemu na softprocesorze, należało skonfigurować podobny system, ale już na płytce z procesorem ZynQ. Moduły wraz z przypisanymi do portów rejestrami pozostały te same. W systemie znikło jedynie GPIO, a softprocesor MicroBlaze został zamieniony na procesor ZynQ. Schemat systemu przedstawiony jest na Rys. 5.1.



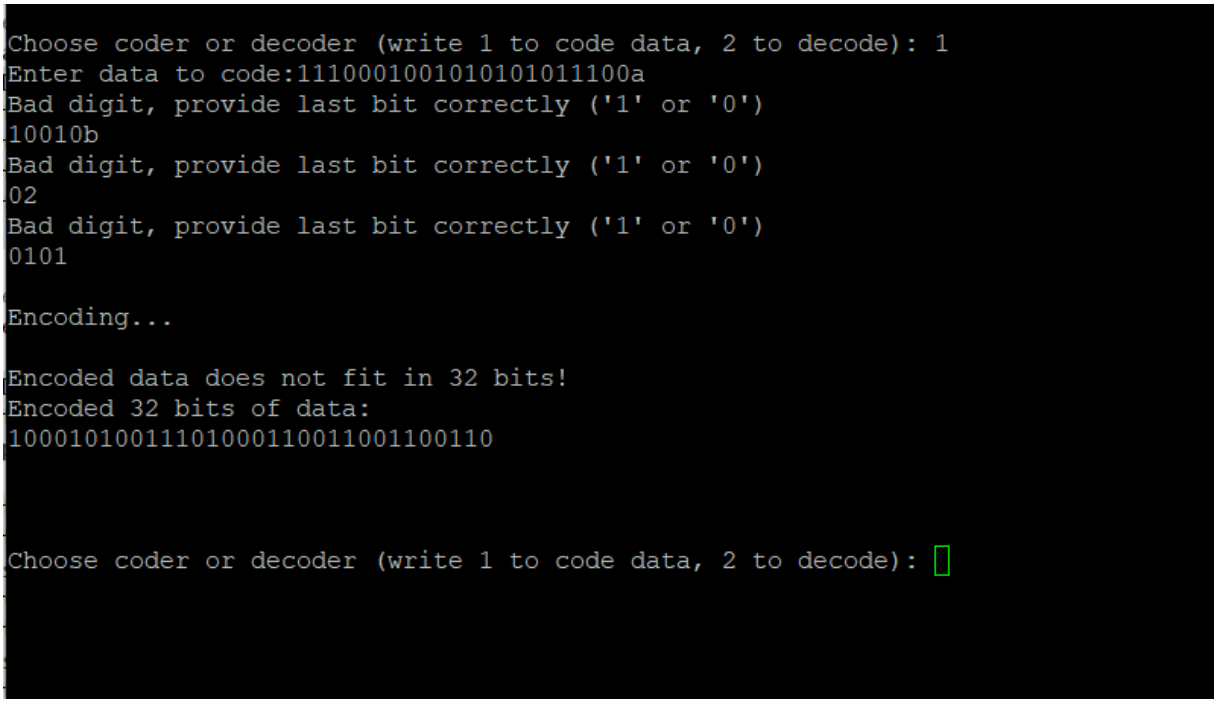
Rys. 5.1: Schemat systemu z procesorem ZynQ.

5.1. Oprogramowanie.

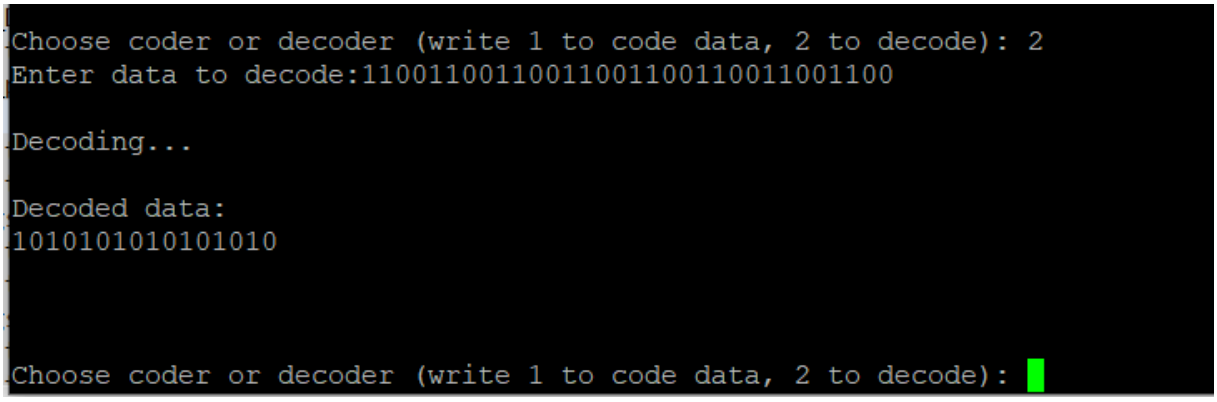
Do obsługi systemu wykorzystano komunikację UART. Układ podłączony do komputera z pomocą programu pokroju PuTTY wyświetla dane użytkownikowi. Schemat działania programu jest bardzo prosty. Najpierw użytkownik dostaje pytanie czy chce skompresować dane, czy odkodować.

Następnie w obu przypadkach użytkownik proszony jest o podanie danych wejściowych. Jeżeli podczas wpisywania pojawi się dana niebinarna użytkownik zostanie o tym poinformowany, a znak nie będzie brany pod uwagę. Po wprowadzeniu danych wybrany moduł od razu zaczyna swoje działanie. W odpowiedzi użytkownik otrzymuje przetworzone dane. W przypadku kodera i podania danych nieopłacalnych do kompresji użytkownik zostaje

poinformowany o zajściu, a następnie wypisywana jest część danych, która zmieściła się na 32 bitach. W przypadku dekodera, korzystając z informacji o bitach dopełniających dane do 32 bitów, wyświetlana jest ilość jedynie tych, które przenoszą informację. Na Rys.5.2 przedstawiony jest interfejs konsolowy z wykonywaniem działań na koderze, a na Rys. 5.3 na dekoderze.



Rys. 5.2: Interfejs konsolowy z wykonywaniem działań na koderze.



Rys. 5.3: Interfejs konsolowy z wykonywaniem działań na dekoderze.

6. Podsumowanie.

Projekt spełnia założenia, czyli realizuje kompresję i dekompresję RLC. System udało się tak zaprojektować, żeby dało się go łatwo rozbudowywać, również moduły. Przy stosunkowo niewielkim nakładzie pracy, można na jego postawie stworzyć system mogący kompresować nawet całe pliki. Dodatkowo całość ma łatwe do rozbudowania testy, które mogą być bardzo przydatne przy ewentualnych modyfikacjach.

Przy pracy nad projektem udało nam się poznać między innymi plusy i minusy realizacji systemów równoległego oraz sekwencyjnego. Dodatkowo rozwinęliśmy swoje umiejętności kodowania w języku opisu sprzętu Verilog oraz SystemVerilog oraz poznaliśmy lepiej platforme Vivado.