Akademia Górniczo-Hutnicza
Wydział Informatyki, Elektroniki i Telekomunikacji
Kierunek Elektronika i Telekomunikacja - Systemy Wbudowane

Systemy Dedykowane w Układach Programowalnych

Algorytm Barrel Wheel Transform

Paweł Frączkiewicz, Piotr Duszkiewicz

Kraków, 15 czerwiec 2023

1 Wstęp

Celem projektu było zaimplementowanie algorytmu Barrel Wheel Transform na platformie FPGA (ang. Field Programmable Gate Array). Algorytm ten jest często stosowany w kompresji danych i analizie sekwencji tekstowych. Transformata polega na konwersji wejściowego wyrazu i składa się z trzech kroków: rotacji, sortowaniu oraz ekstrakcji końcowych liter tworząc nowy wyraz. BWT wykorzystywana jest między innymi w kompresji, ponieważ ułatwia skompresowanie ciągu powtarzających się znaków redukując rozmiar danych. Dodatkowo wiele algorytmów kompresji, takich jak bzip2, wykorzystują transformatę BWT do przygotowania danych przed zastosowaniem innych technik kompresji. BWT tworzy sekwencje znaków o dużej skupialności, które są bardziej podatne na skuteczną kompresję. Transformatę BWT można również zastosować w procesie szyfrowania danych. Poprzez odpowiednie przekształcenie tekstu za pomocą BWT i zastosowanie algorytmów szyfrowania, można uzyskać zaszyfrowane dane, które są trudniejsze do odczytania bez odpowiedniego klucza.

Urządzeniem, na którym dokonano implementacji algorytmu jest płyta ewaluacyjna Zedboard Zynq-7000. Posiada ona dwurdzeniowy procesor ARM Cortex A9, 512MB pamięci DDR3 oraz 256MB pamięci flash QSPI. Płyta ewaluacyjną może zostać zastosowana do przetwarzania wideo, obliczeń rekonfigurowalnych czy przyspieszania oprogramowania.

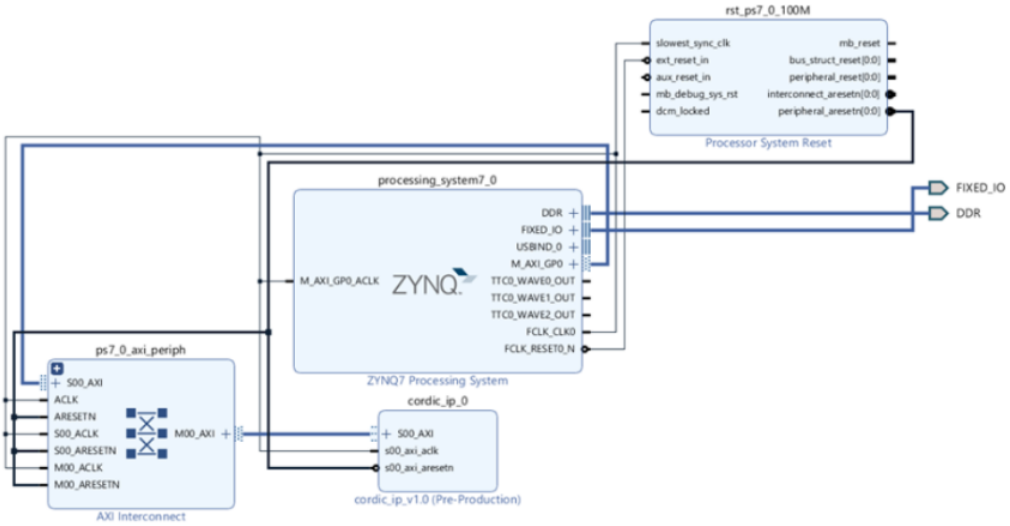
Projekt transformaty wykonano w środowisku Vivado w wersji 2018.3 oraz Visual Code Studio w celu utworzenia aplikacji prezentującej logikę działania BWT w języku Python w wersji 3.11.

2 Struktura projektu

Projekt znajduje się na repozytorium github: link

Struktura repozytorium wygląda następująco:

- BWT – zawiera dwa modele behawioralne implementujące transformatę. Pierwszy model wykorzystuje strukturę sekwencyjnej maszyny stanów symulując działanie klasycznego mikroprocesora, gdzie wszystkie operacje następują jedna po drugiej. Natomiast drugi model wykorzystuje strukturę potokową (ang. pipeline) prezentując zalety wykorzystania układów FPGA jako akceleratorów w wielu dziedzinach.
- `BWT_cordic` – zawiera diagram blokowy integrujący płytę deweloperską Zedboard z utworzonym przez nas akceleratorem (`cordic_ip_0`). W celu połączeniu wyżej wymienionych bloków, wykorzystana została magistrala AXI.



Rysunek 1: Schemat blokowy

- `Ip_repo` – zawiera plik ip, stosowany w diagramie blokowym powyżej. W swojej strukturze zapisany został akcelerator transformaty BWT
- BWT.py – plik prezentujący działanie transformaty w języku Python

3 Opis działania algorytmu

Transformata BWT składa się z trzech poniższych kroków:

a) rotacja - należy utworzyć wszystkie cykliczne rotacje danego wyrazu wejściowego

b) sortowanie - następnym krokiem jest posortowanie rotacji leksykograficznie

c) ekstrakcja - polega na wydobyciu ostatniej litery ze wszystkich wyrazów znajdujących się w utworzonej kolumnie

Poniższy rysunek obrazuję zastosowanie transformaty BWT na przykładowym wyrazie:

Transformation				
Input	All rotations	Sorted into lexical order	Taking last column	Output last column
^BANANA|	^BANANA| |^BANANA A|^BANAN NA|^BANA ANA|^BAN NANA|^BA ANANA|^B BANANA|^	**A**NANA|^B **AN**A|^BAN **A**|^BANAN **B**ANANA|^ **N**ANA|^BA **NA**|^BANA ^BANANA| |^BANANA	ANANA|^**B** ANA|^BA**N** A|^BANA**N** BANANA|^ NANA|^B**A** NA|^BAN**A** ^BANANA| |^BANAN**A**	BNN^AA|A

Rysunek 2: Schemat transformaty BWT

4 Implementacja algorytmu w środowisku Python

W tym podrozdziale zostanie przybliżone działanie algorytmu opracowane w środowisku Python.

Kod został napisany w taki sposób, aby wymusić jak najmniejsze modyfikacje podczas przenoszenia do środowiska Vivado. Wiąże się to również z tym, że nie wykorzystujemy żadnych zewnętrznych bibliotek.

Opis kodu:

1. Na początku użytkownik zostaje poproszony o wprowadzenie wyrazu, który ma zostać poddany transformacji BWT.
2. Następnie, dla podanego wyrazu, generowane są wszystkie możliwe rotacje tego wyrazu. Każda rotacja to przesunięcie znaków wyrazu o jedno miejsce w lewo.
3. Wygenerowane rotacje są przechowywane w liście `new_word`
4. Kolejnym krokiem jest sortowanie rotacji w liście `new_word`. Sortowanie odbywa się zgodnie z regułami algorytmu BWT. Jeśli dwa wyrazy mają ten sam pierwszy znak, porównuje się kolejne znaki aż do znalezienia pierwszego różniącego się znaku. Jeśli pierwszy znak w pierwszej rotacji jest większy

od pierwszego znaku w drugiej rotacji, następuje zamiana miejscami tych rotacji. Jeśli pierwszy znak w pierwszej rotacji jest mniejszy od pierwszego znaku w drugiej rotacji, to nie ma zamiany miejscami.

5. Po posortowaniu, lista `new_word` zawiera rotacje w odpowiedniej kolejności.

6. Następnie, tworzona jest lista `new_new_word`, która przechowuje ostatnie znaki poszczególnych rotacji (znaki ostatniej kolumny w macierzy rotacji).

7. Na koniec, lista `new_word` jest wyświetlana jako wynik transformacji BWT.

Opisany kod znajduje się poniżej:

```
1
2   word = input("podaj wyraz poddany transformacie BWT \n")
3
4   print("word: " + word + "\n")
5   new_word = []
6   for i in range(len(word)):
7       new_word.append(word[i:]+word[:i]) #kolejne rotacje
8   print("tablica rotacji:")
9   print(new_word)
10  print("\n")
11
12
13
14  najmniejsza = True;
15
16  for x in range(len(new_word)):
17      for y in range(len(new_word)-x-1):
18          if new_word[x][0] == new_word[x+y+1][0]:
19
20              for i in range(len(new_word)):
21                  if  new_word[x][i] > new_word[x+y+1][i]:
22                      zmienna = new_word[x]
23                      new_word[x]=new_word[x+y+1]
24                      new_word[x+y+1] = zmienna
25
26                      break
27                  elif  new_word[x][i] < new_word[x+y+1][i]:
28
29                      break
30
31
32          elif new_word[x][0] > new_word[x+y+1][0]:
33              zmienna = new_word[x]
34              new_word[x]=new_word[x+y+1]
35              new_word[x+y+1] = zmienna
36
37
38  print("posortowane:")
39  print(new_word)
40  print("\n")
41
42  new_new_word = []
43
44  for i in range(len(new_word)):
45      new_new_word.append(new_word[i][-1])
46
47  print("wyjście:")
48  print(new_new_word)
49  print("\n")
50
```

Rysunek 3: Kod programu w języku Python

Na poniższym zrzucie widoczny jest działanie przedstawionego kodu:

```
PS D:\AGH_magisterskie\SDUP\Projekt\SDUP_BWT_transformata> python BWT.py
podaj wyraz poddany transformacie BWT
0AB2C1AF
word: 0AB2C1AF

tablica rotacji:
['0AB2C1AF', 'AB2C1AF0', 'B2C1AF0A', '2C1AF0AB', 'C1AF0AB2', '1AF0AB2C', 'AF0AB2C1', 'F0AB2C1A']


posortowane:
['0AB2C1AF', '1AF0AB2C', '2C1AF0AB', 'AB2C1AF0', 'AF0AB2C1', 'B2C1AF0A', 'C1AF0AB2', 'F0AB2C1A']


wyjście:
['F', 'C', 'B', '0', '1', 'A', '2', 'A']


PS D:\AGH_magisterskie\SDUP\Projekt\SDUP_BWT_transformata>
```

Rysunek 4: Wynik działania programu

5 Podstawowa wersja algorytmu i jego symulacje

Kolejnym etapem projektu było zaimplementowanie algorytmu w środowisku Vivado. Pierwsza wersja algorytmu miała proste działanie - tj. nie korzysta z pipeline. Poniżej przedstawiono opis działania poszczególnych stanów modułu ‘‘`BWT_transform`’’ w którym to zaimplementowany został algorytm BWT.

Opis zmiennych oraz parametrów:

Moduł `BWT_transform` przyjmuje sygnały wejściowe clk, rst, start oraz dane wejściowe `data_in`.

Parametr `dl_wyraz` określa szerokość wyrazu (w liczbie bitów) i jest bezpośrednio zależny od szerokości `data_in`.

Parametr `szer_litery` określa szerokość pojedynczej litery w wyrazie (w liczbie bitów).

Parametr `ilosc_liter` oblicza ilość liter w wyrazie na podstawie `dl_wyraz` i `szer_litery`.

Sygnał wyjściowy `data_out` przechowuje dane wyjściowe po transformacji BWT.

Sygnał wyjściowy done jest flagą oznaczającą zakończenie obliczeń.

Zmienna lokalna buffor jest tablicą, na której odbywają się wszystkie operacje rotacji i sortowania.

Zmienna lokalna `data_var` przechowuje wartość bazową podczas wymiany indeksów podczas sortowania.

Zmienna lokalna `dana_wyj` przechowuje ostatnie bity, tworząc tym samym wyraz wyjściowy.

Moduł zawiera parametry S1, S2, ..., S8, które określają kolejne stany automatu.

Zmienne i, x, y i z są licznikami, które kontrolują przebieg procesu transformacji.

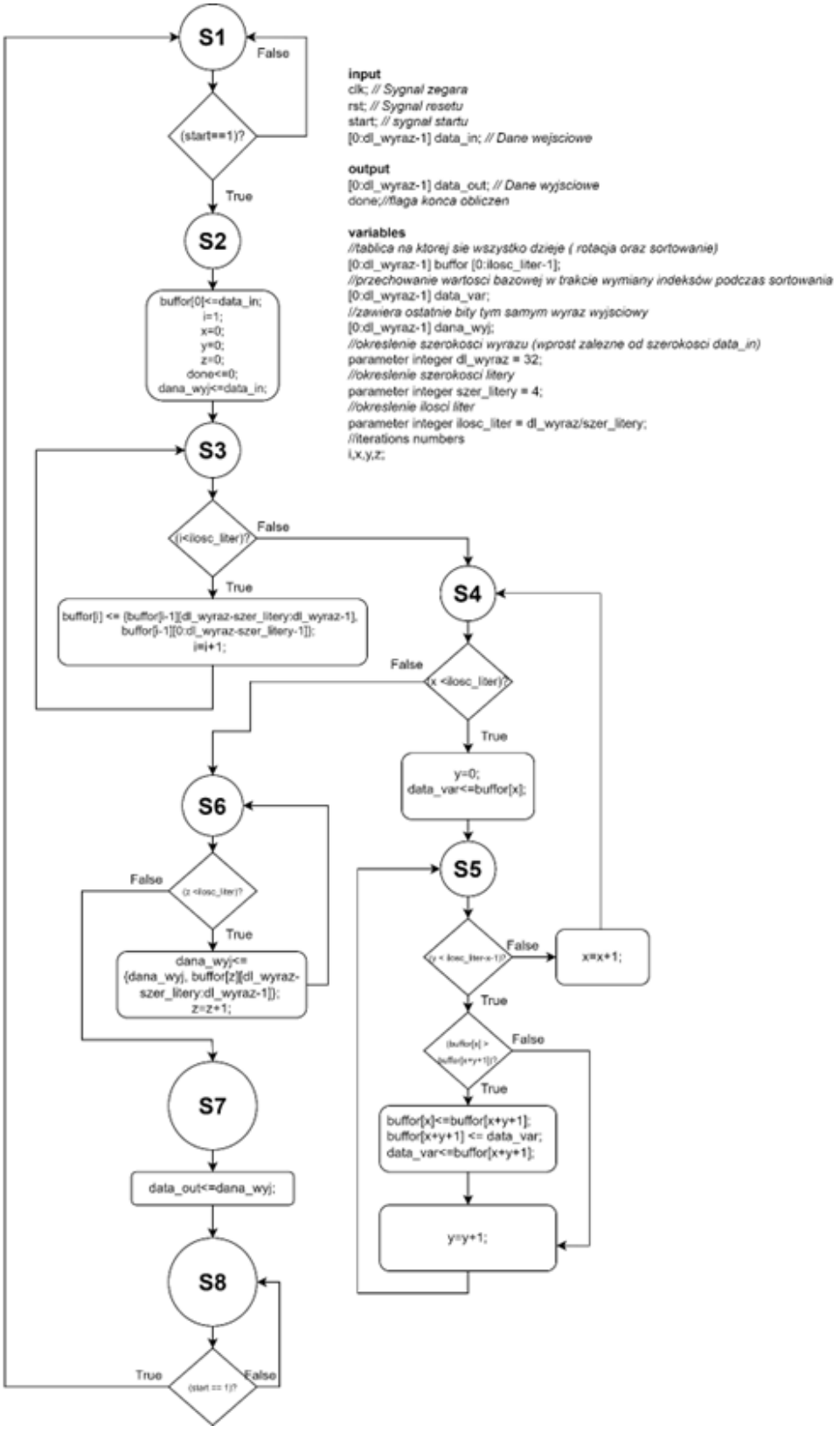
Parametry wejściowe:
clk - Sygnał zegara
rst - Sygnał resetu
start - Sygnał rozpoczęcia obliczeń
`data_in` - Dane wejściowe, reprezentujące wyraz poddawany transformacji BWT

Zmienne i sygnały wyjściowe:
`data_out` - Dane wyjściowe po transformacji BWT
done - Flaga oznaczająca zakończenie obliczeń

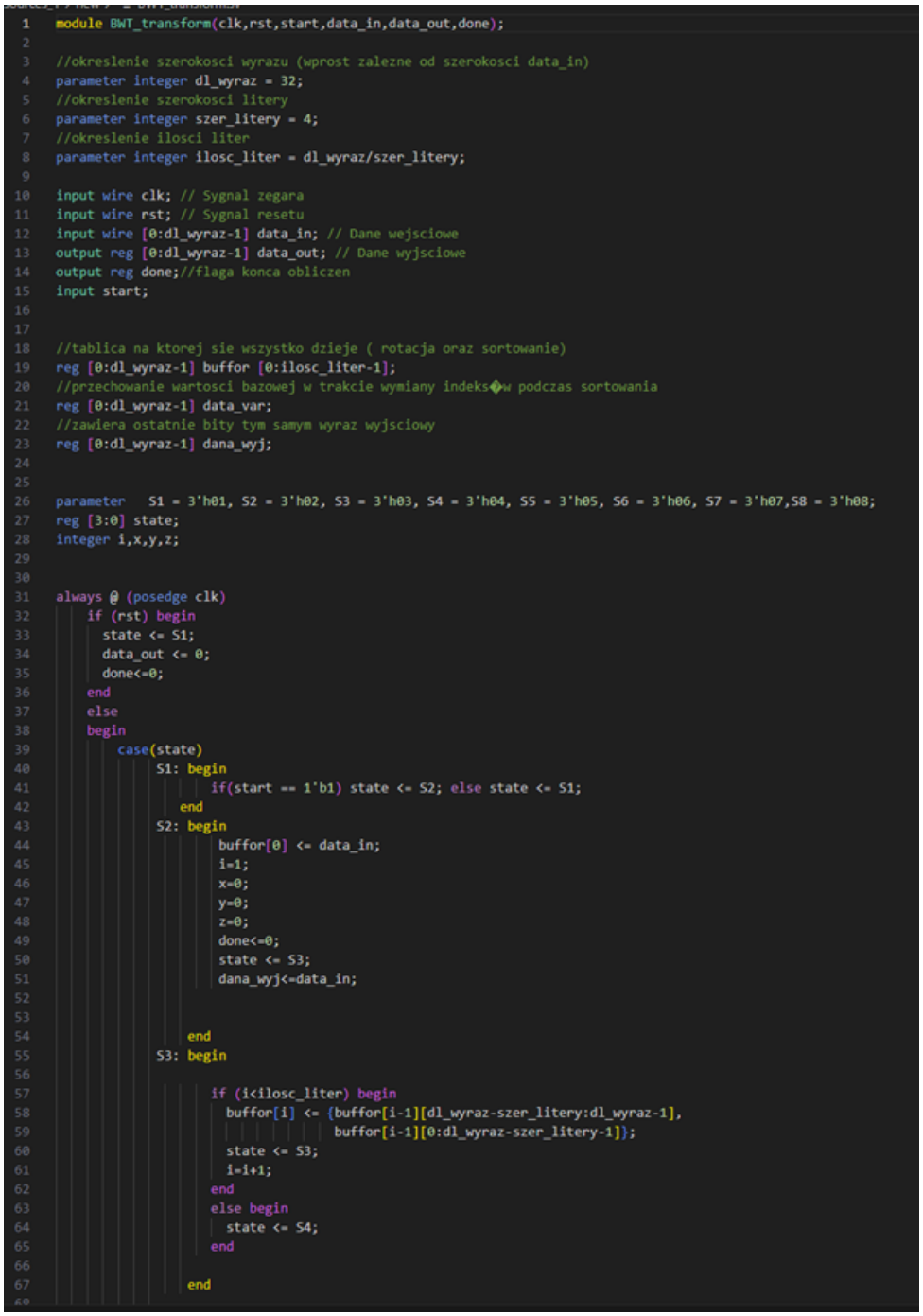
Opis działania kodu:

Po wystąpieniu zbocza narastającego sygnału zegara (posedge clk), w zależności od stanu rst (resetu), następuje przypisanie wartości początkowych dla zmiennych. Wykorzystując konstrukcję case, następuje przejście przez kolejne stany automatu. Stan S1 oczekuje na aktywację sygnału start. Po jego aktywacji, przechodzi do stanu S2. Stan S2 inicjalizuje zmienną buffor[0] wartością `data_in` i ustala wartości początkowe dla zmiennych licznikowych.
Następnie następuje przejście przez kolejne stany S3, S4, S5 i S6, w których wykonuje się generowanie rotacji i sortowanie w tablicy buffor.
W stanie S7 ustawiana jest flaga done, przypisywane są dane wyjściowe do `data_out` i przechodzimy do stanu S8.
W stanie S8 oczekujemy na wyłączenie sygnału start. Po wyłączeniu, przechodzimy ponownie do stanu S1.

Graficzne przedstawienie działania algorytmu:



Rysunek 5: Algorytm programu



```
module BWT_transform(clk,rst,start,data_in,data_out,done);

//okreslenie szerokosci wyrazu (wprost zalezne od szerokosci data_in)
parameter integer dl_wyraz = 32;
//okreslenie szerokosci litery
parameter integer szer_litery = 4;
//okreslenie ilosci liter
parameter integer ilosc_liter = dl_wyraz/szer_litery;

input wire clk; // Sygnal zegara
input wire rst; // Sygnal resetu
input wire [0:dl_wyraz-1] data_in; // Dane wejsciowe
output reg [0:dl_wyraz-1] data_out; // Dane wyjsciowe
output reg done;//flaga konca obliczen
input start;


//tablica na ktorej sie wszystko dzieje ( rotacja oraz sortowanie)
reg [0:dl_wyraz-1] buffor [0:ilosc_liter-1];
//przechowanie wartosci bazowej w trakcie wymiany indeks?w podczas sortowania
reg [0:dl_wyraz-1] data_var;
//zawiera ostatnie bity tym samym wyraz wyjsciowy
reg [0:dl_wyraz-1] dana_wyj;


parameter   S1 = 3'h01, S2 = 3'h02, S3 = 3'h03, S4 = 3'h04, S5 = 3'h05, S6 = 3'h06, S7 = 3'h07,S8 = 3'h08;
reg [3:0] state;
integer i,x,y,z;


always @ (posedge clk)
    if (rst) begin
      state <= S1;
      data_out <= 0;
      done<=0;
    end
    else
    begin
        case(state)
            S1: begin
                  if(start == 1'b1) state <= S2; else state <= S1;
                end
            S2: begin
                  buffor[0] <= data_in;
                  i=1;
                  x=0;
                  y=0;
                  z=0;
                  done<=0;
                  state <= S3;
                  dana_wyj<=data_in;


                end
            S3: begin

                  if (i<ilosc_liter) begin
                    buffor[i] <= {buffor[i-1][dl_wyraz-szer_litery:dl_wyraz-1],
                                  buffor[i-1][0:dl_wyraz-szer_litery-1]};
                    state <= S3;
                    i=i+1;
                  end
                  else begin
                    state <= S4;
                  end

                end
```

Rysunek 6: Moduł BWT Transform

```
69              S4: begin//X
70                   if (x <ilosc_liter) begin
71                     y=0;
72                     data_var<=buffor[x];
73                     state <= S5;
74
75                   end
76                   else begin
77                     state <= S6;
78                   end
79                 end
80              S5:begin//y
81                   if (y < ilosc_liter-x-1) begin
82
83                     if(buffor[x] > buffor[x+y+1]) begin
84                         buffor[x]<=buffor[x+y+1];
85                         buffor[x+y+1] <= data_var;
86                         data_var<=buffor[x+y+1];
87                     end
88                     y=y+1;
89                     state <= S5;
90                   end
91                   else begin
92                     x=x+1;
93                     state <= S4;
94                   end
95                 end
96              S6:begin
97                  if (z <ilosc_liter) begin
98                     dana_wyj<= {dana_wyj,
99                                 buffor[z][dl_wyraz-szer_litery:dl_wyraz-1]};
100                    z=z+1;
101                    state <= S6;
102                  end
103                  else begin
104                    state <= S7;
105                  end
106                end
107             S7:begin
108                done<=1;
109                data_out<=dana_wyj;
110                state <= S8;
111                end
112             S8:begin
113                if(start == 1'b0) state <= S1; else state <= S8;
114               end
115
116         endcase
117     end
118 endmodule
```

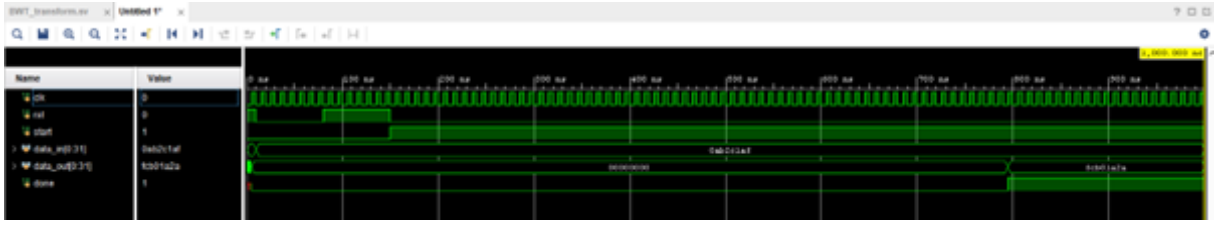
Rysunek 7: Moduł BWT Transform

Kolejnym etapem było stworzenie testbench'u w celu weryfikacji poprawności działania modułu. Dane wejściowe symulacji zostały podane takie same jak w przypadku implementacji w środowisku Python w celu możliwie łatwego zweryfikowania i zestawienia wyników.

```
module BWT_transform_tb;

  // Sygnadź˙y testowe
  logic clk;
  logic rst;
  logic start;
  logic [0:31] data_in;
  logic [0:31] data_out;
  logic done;

  BWT_transform dut (
    .clk(clk),
    .rst(rst),
    .start(start),
    .data_in(data_in),
    .done(done),
    .data_out(data_out)
  );

  // Generacja sygnadź˙u zegara
  always #5 clk = ~clk;

  // Inicjalizacja wejdź˙dź˙
  initial begin
    clk = 0;
    rst = 1;
    start=0;
    data_in = '0;


    #10 rst = 0;
    data_in = 'h0AB2C1AF;

    //hFCB01a2a

    #70 rst = 1;
     data_in = 'h0AB2C1AF;
  ;


    #70 rst = 0;
     start=1;
     data_in = 'h0AB2C1AF;

    #300;

  end

  // Wypisanie wynik
  always @(posedge clk) begin
    $display("data_in = %h, data_out = %h", data_in, data_out);
  end

endmodule
```

Rysunek 8: Testbench modułu

Na poniższym zrzucie widać przebieg symulacji oraz otrzymaną wartość wyjściową. Jak można zaobserwować wynik wyjściowy pokrywa się z wynikiem uzyskanym w przypadku implementacji w środowisku Python.



Rysunek 9: Uzyskany wynik symulacji

6 Dodanie potoku do algorytmu

W celu zrealizowania potoku utworzone zostały dodatkowe dwa moduły ‘‘BWT_last_letter‘‘ oraz ‘‘BWT_step".

Opis działania modułu ‘‘BWT_step’’:

1. Moduł BWT_step przyjmuje sygnały wejściowe ‘clk’, ‘start oraz dane wejściowe ‘wejscie’.
2. Parametr ‘moved’ określa liczbę przesunięć, czyli ile liter jest przesuwanych w jednym kroku transformacji BWT.
3. Parametr ‘dl_wyraz’ określa szerokość wyrazu (w liczbie bitów) i jest bezpośrednio zależny od szerokości ‘wejscie’.
4. Parametr ‘szer_litery’ określa szerokość pojedynczej litery w wyrazie (w liczbie bitów).
5. Sygnał wyjściowy ‘wyjscie’ przechowuje dane wyjściowe po wykonaniu jednego kroku transformacji BWT.
6. W bloku ‘always @(posedge clk)’ następuje reakcja na zbocze narastające sygnału zegara.
7. Jeśli sygnał ‘start’ jest aktywny (‘start == 1’b1’), wykonuje się przesunięcie liter w wyrazie zgodnie z wartością parametru ’moved’.
8. Do sygnału ‘wyjscie’ przypisywane są przesunięte wartości z wyrazu wejscie.
9. W przeciwnym razie, gdy sygnał start jest nieaktywny, sygnał ‘wyjscie’ jest wyzerowany (wyjscie <= 0).

Opis działania modułu ‘‘BWT_last_letter’’:

1. Moduł ‘‘BWT_last_letter’’ przyjmuje sygnały wejściowe clk, start oraz dane wejściowe ‘wejscie’.
2. Parametr ‘moved’ jest bezużyteczny w tym module, ponieważ określa liczbę przesunięć, które nie są stosowane w ostatnim kroku transformacji.
3. Parametr ‘dl_wyraz’ określa szerokość wyrazu (w liczbie bitów) i jest bezpośrednio zależny od szerokości ‘wejscie’.
4. Parametr ‘szer_litery’ określa szerokość pojedynczej litery w wyrazie (w liczbie bitów).
5. Sygnał wyjściowy ‘wyjscie’ przechowuje dane wyjściowe po wykonaniu ostatniego kroku transformacji BWT.
6. W bloku ’always @(posedge clk)’ następuje reakcja na zbocze narastające sygnału zegara.
7. Jeśli sygnał start jest aktywny (‘start == 1’b1’), wartość sygnału ‘wejscie’ przypisywana jest bezpośrednio do sygnału ‘wyjscie’.
8. Do sygnału ‘wyjscie’ przypisywane są tylko ostatnie ‘szer_litery’ bitów z wyrazu ‘wejscie’.
9. W przeciwnym razie, gdy sygnał start jest nieaktywny, sygnał wyjscie jest wyzerowany (wyjscie <= 0).

Moduł ‘BWT_transform_with_Pipeline‘ jest implementacją transformacji BWT (Burrows-Wheeler Transform) w języku Verilog z zastosowaniem pipelinowania dla lepszej wydajności. Poniżej przedstawiam opis działania kodu oraz parametry wejściowe i wyjściowe:

Parametry wejściowe:
clk - sygnał zegara
rst - sygnał resetu
start - sygnał rozpoczęcia transformacji BWT
data_in - dane wejściowe

Parametry wyjściowe:
`data_out` - dane wyjściowe po wykonaniu transformacji BWT
done - flaga końca obliczeń

Zmienne:
state - zmienna reprezentująca aktualny stan procesu transformacji BWT
x - zmienna pomocnicza używana w pętli do iteracji po literach wyrazu
y - zmienna pomocnicza używana w pętli do iteracji po literach wyrazu.
zacznij - zmienna sterująca, wskazująca, czy należy rozpocząć przepisywanie wartości z `'buffor_rotacji'` do 'buffor' (w przypadku pierwszej iteracji).
scal - zmienna pomocnicza używana do skalowania indeksów podczas przypisywania danych wyjściowych.

Parametry:
`dl_wyraz` - parametr określający szerokość wyrazu (liczba bitów) oraz liczność tablic `data_in`, `data_out` i 'buffor'.
`szer_litery` - parametr określający szerokość pojedynczej litery w wyrazie (liczba bitów).
`ilosc_liter` - parametr określający liczbę liter w wyrazie, czyli ilość przesunięć, które zostaną wykonane w ramach transformacji BWT.
'S1', 'S2', 'S3', 'S4', 'S5', 'S6' - parametry reprezentujące stany procesu transformacji BWT.

Parametry te umożliwiają konfigurację i dostosowanie działania modułu `'BWT_transform_with_Pipeline'` do konkretnych wymagań aplikacji. Przykładowo, można zmienić szerokość wyrazu `'dl_wyraz'` na inną wartość, taką jak 64 lub 128 bitów, dostosować szerokość pojedynczej litery `'szer_litery'` do określonego formatu danych wejściowych oraz zmienić liczbę liter `'ilosc_liter'` w zależności od oczekiwanej wydajności i ilości przesunięć, które chcemy wykonać podczas transformacji BWT.

Przez modyfikację tych zmiennych i parametrów można dostosować moduł `'BWT_transform_with_Pipeline'` do różnych aplikacji, umożliwiając wykonywanie transformacji BWT na różnych typach danych wejściowych.

Opis działania kodu:

1. Moduł `'BWT_transform_with_Pipeline'` przyjmuje sygnały wejściowe 'clk', 'rst', 'start' oraz dane wejściowe `'data_in'`.
2. Parametr `'dl_wyraz'` określa szerokość wyrazu (w liczbie bitów) i jest bezpośrednio zależny od szerokości `'data_in'`.
3. Parametr `'szer_litery'` określa szerokość pojedynczej litery w wyrazie (w liczbie bitów).
4. Parametr `'ilosc_liter'` określa ilość liter, czyli ilość przesunięć, które zostaną wykonane w ramach transformacji BWT.
5. Sygnał wyjściowy `'data_out'` przechowuje dane wyjściowe po wykonaniu transformacji BWT.
6. Sygnał 'done' jest flagą końca obliczeń.
7. Sygnał 'start' inicjuje proces transformacji BWT.
8. Tablica 'buffor' jest używana do przechowywania danych, na których odbywa się rotacja i sortowanie.
9. Tablica `'buffor_rotacji'` przechowuje wynik rotacji z poprzedniego kroku transformacji BWT.
10. Sygnał `'data_var'` przechowuje wartość bazową podczas wymiany indeksów podczas sortowania.
11. W bloku 'generate' tworzone są instancje modułów `'BWT_step'` i `'BWT_last_letter'` w celu zastosowania pipelinowania w transformacji BWT.
12. W bloku 'always @(posedge clk)' następuje reakcja na zbocze narastające sygnału zegara.
13. Gdy sygnał 'rst' (reset) jest aktywny, inicjalizowane są początkowe wartości zmiennych, a stan 'state' ustawiany jest na 'S1'.

14. Gdy sygnał ‘start‘ jest po zakończeniu sortowania, w stanie ‘S5‘ ustawiana jest flaga ‘done‘ na wartość ‘1‘, a dane wyjściowe są przechowywane w tablicy `‘data_out‘`.t aktywny (‘start == 1’b1‘), następuje rozpoczęcie transformacji BWT.

15. W stanie ‘S2‘ następuje przypisanie danych wejściowych do tablicy ‘buffor[0]‘, ustawienie zmiennych pomocniczych i flag, oraz przejście do stanu ‘S3‘.

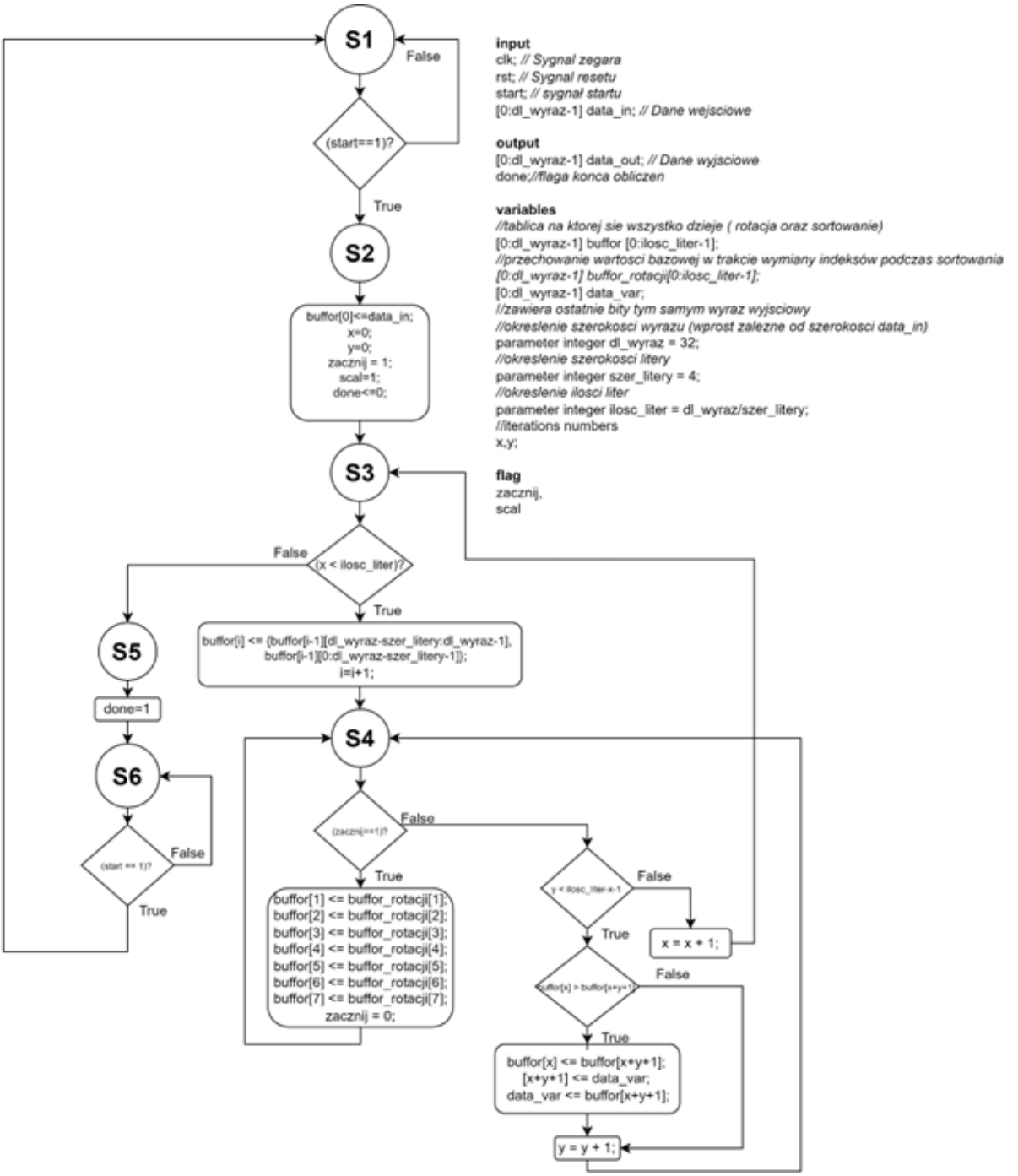
16. W stanach ‘S3‘ i ‘S4‘ następuje sortowanie danych w tablicy ‘buffor‘ za pomocą algorytmu Bubble Sort.

17. Po zakończeniu sortowania, w stanie ‘S5‘ ustawiana jest flaga ‘done‘ na wartość ‘1‘, a dane wyjściowe są przechowywane w tablicy `‘data_out‘`.

18. W stanie ‘S6‘ następuje sprawdzenie, czy sygnał ‘start’ jest aktywny. Jeśli nie, następuje powrót do stanu ‘S1‘.

19. Proces transformacji BWT jest powtarzany, dopóki sygnał ‘start‘ jest aktywny.

Graficzne przedstawienie działania algorytmu:



Rysunek 10: Algorytm transformaty BWT przy zastosowaniu potoku

Opis stanów:
S1: Stan początkowy. Oczekuje na sygnał start. Jeśli otrzyma sygnał start, przechodzi do stanu S2; w przeciwnym razie pozostaje w stanie S1.
S2: Inicjalizacja bufora zerowego, iteratorów i zmiennych pomocniczych. Ustawienie flag scal i zacznij. Przechodzi do stanu S3.
S3: Wykonywanie rotacji dla kolejnych liter w buforze. Przechodzi do stanu S4, gdy wszystkie rotacje są zakończone.
S4: Sortowanie bufora. Jeśli zmienna zacznij jest równa 1, przypisuje wartości `bufor_rotacji` do odpowiednich buforów. W przeciwnym razie porównuje i wymienia elementy bufora. Przechodzi do stanu S3, aby kontynuować sortowanie.

S5: Końcowy stan sortowania. Ustawia flagę "done"na 1. Przechodzi do stanu S6.
S6: Oczekiwanie na sygnał "start"równy 0. Jeśli otrzyma sygnał "start"równy 0, przechodzi do stanu S1; w przeciwnym razie pozostaje w stanie S6.

Opisany kod znajduje się poniżej:

```
module BWT_transform_with_Pipeline(clk, rst, start, data_in, data_out, done);

  // Okre?lenie szeroko?ci wyrazu (wprost zale?ne od szeroko?ci data_in)
  parameter integer dl_wyraz = 32;
  // Okre?lenie szeroko?ci litery
  parameter integer szer_litery = 4;
  // Okre?lenie ilo?ci liter
  parameter integer ilosc_liter = dl_wyraz / szer_litery;

  input wire clk;              // Sygna? zegara
  input wire rst;              // Sygna? resetu
  input wire [0:dl_wyraz-1] data_in;    // Dane wej?ciowe
  output wire [0:dl_wyraz-1] data_out;   // Dane wyj?ciowe
  output reg done;             // Flaga ko?ca oblicze?
  input start;

  // Tablica, na kt?rej si? wszystko dzieje (rotacja oraz sortowanie)
  reg [0:dl_wyraz-1] buffor[0:ilosc_liter-1];
  wire [0:dl_wyraz-1] buffor_rotacji[0:ilosc_liter-1];
  // Przechowanie warto?ci bazowej w trakcie wymiany indeks?w podczas sortowania
  reg [0:dl_wyraz-1] data_var;
  // Zawiera ostatnie bity tym samym wyraz wyj?ciowy
//wire [0:dl_wyraz-1] dana_wyj;

  parameter S1 = 3'h01, S2 = 3'h02, S3 = 3'h03, S4 = 3'h04, S5 = 3'h05, S6 = 3'h06, S7 = 3'h07;
  reg [3:0] state;
  integer x, y;
  integer wypelnij_macierz;
  integer scal;


  generate
    genvar numb;
    for (numb = 1; numb < ilosc_liter; numb = numb + 1) begin : BWT_step_loop
      BWT_step #(numb, dl_wyraz, szer_litery) BWT_step_part(clk, wypelnij_macierz, buffor[0], buffor_rotacji[numb]);
    end

    genvar numb2;
    for (numb2 = 0; numb2 < ilosc_liter; numb2 = numb2 + 1) begin : BWT_last_letter_loop
      BWT_last_letter #(numb2, dl_wyraz, szer_litery) BWT_last_letter_part(clk, scal, buffor[numb2], data_out[numb2*szer_litery : (szer_litery*(numb2+1)-1)]);
    end
  endgenerate

  always @(posedge clk)
  begin
    if (rst) begin
      state <= S1;
      //data_out <= 0;
      done <= 0;
      scal=0;
      wypelnij_macierz = 0;
    end
    else begin
      case(state)
        S1: begin
          if (start == 1'b1) state <= S2;
          else state <= S1;
        end
        S2: begin

          buffor[0] <= data_in;
          x = 0;
          y = 0;
          wypelnij_macierz = 1;
          scal=1;
          done = 0;
```

Rysunek 11: Moduł transformaty BWT wykorzystujący pipeline

```
189         end
190         S3: begin // X
191           if (x < ilosc_liter) begin
192             y = 0;
193             data_var <= buffor[x];
194             state <= S4;
195           end
196           else begin
197             state <= S5;
198           end
199         end
200         S4: begin // Y
201 
202           if( wypelnij_macierz==1) begin
203               buffor[1] <= buffor_rotacji[1];
204               buffor[2] <= buffor_rotacji[2];
205               buffor[3] <= buffor_rotacji[3];
206               buffor[4] <= buffor_rotacji[4];
207               buffor[5] <= buffor_rotacji[5];
208               buffor[6] <= buffor_rotacji[6];
209               buffor[7] <= buffor_rotacji[7];
210               wypelnij_macierz = 0;
211 
212               //buffor[1:ilosc_liter-1]<=buffor_rotacji[1:ilosc_liter-1];
213             end
214             else begin
215                   if (y < ilosc_liter-x-1) begin
216                     if (buffor[x] > buffor[x+y+1]) begin
217                       buffor[x] <= buffor[x+y+1];
218                       buffor[x+y+1] <= data_var;
219                       data_var <= buffor[x+y+1];
220                     end
221                     y = y + 1;
222                     state <= S4;
223                   end
224                   else begin
225                     x = x + 1;
226                     state <= S3;
227                   end
228              end
229             end
230         S5: begin
231           //scal=1;
232           done = 1;
233           //data_out <= dana_wyj;
234           state <= S6;
235         end
236         S6: begin
237           if (start == 1'b0) state <= S1;
238           else state <= S6;
239         end
240       endcase
241     end
242   end
243 endmodule
244 
```

Rysunek 12: Moduł transformaty BWT wykorzystujący pipeline

Moduł do wszystkich rotacji:

```
`timescale 1ns / 1ps

module BWT_step #(parameter integer moved = 1, parameter integer dl_wyraz = 32, parameter integer szer_litery = 4)
                 (input clk, input start, input  [0:dl_wyraz-1] wejscie, output reg  [0:dl_wyraz-1] wyjscie);

  always @(posedge clk) begin
    if (start == 1'b1) begin
        wyjscie <= {wejscie[dl_wyraz - szer_litery * moved : dl_wyraz - 1],
                    wejscie[0 : dl_wyraz - szer_litery * moved - 1]};
    end
    else begin
        wyjscie<=0;
    end
  end

endmodule

```

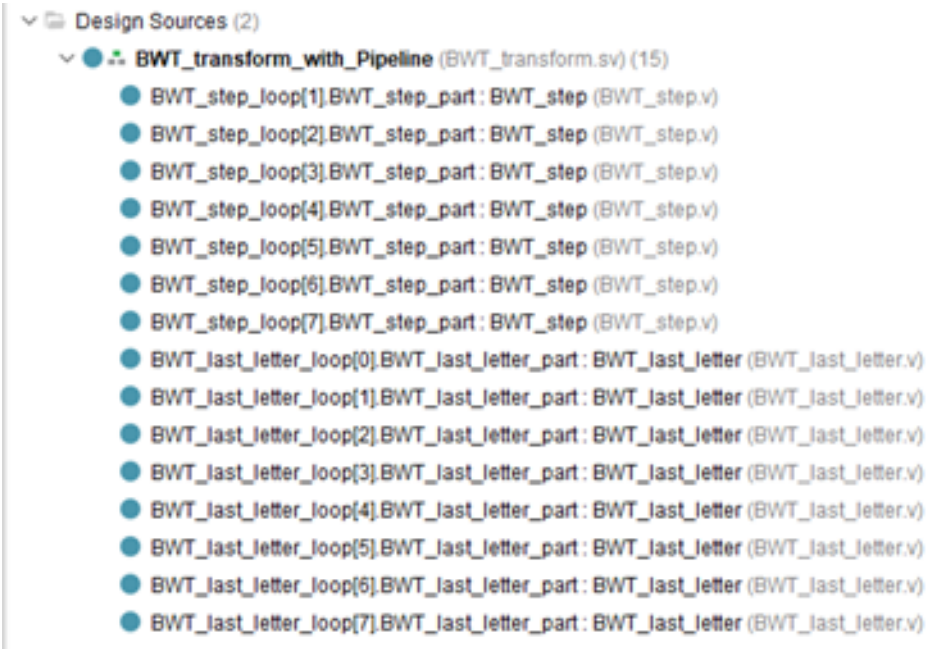
Rysunek 13: Moduł rotacji

Moduł do zebrania ostatniej litery w słowach:

```
`timescale 1ns / 1ps

module BWT_last_letter #(parameter integer moved = 0, parameter integer dl_wyraz = 32, parameter integer szer_litery = 4)
                        (input clk, input start, input  [0:dl_wyraz-1] wejscie, output reg  [0:dl_wyraz-1] wyjscie);

  always @(posedge clk) begin
    if (start == 1'b1) begin
      wyjscie <= wejscie[dl_wyraz - szer_litery : dl_wyraz - 1];

    end
    else begin
      wyjscie<=0;
    end
  end
endmodule
```

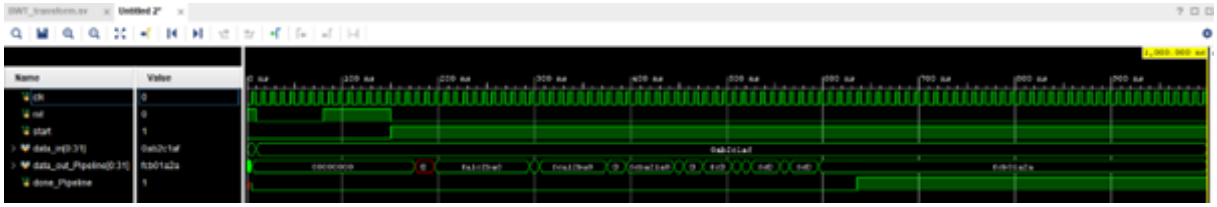
Rysunek 14: Moduł ekstrakcji

Poniżej przedstawiono strukturę projektu:



Rysunek 15: Struktura projektu

Na poniższym zrzucie widać przebieg symulacji oraz otrzymaną wartość wyjściową. Jak można zaobserwować wynik wyjściowy pokrywa się z wynikiem uzyskanym w przypadku implementacji w środowisku Python.



Rysunek 16: Wynik symulacji przy wykorzystaniu potoku

7 Zestawienie symulacji podstawowego algorytmu z algorytmem wykorzystującym pipeline

W celu łatwego zestawienia symulacji zmodyfikowany został testbench tak, aby obie wersje algorytmu działały równolegle.

BWT > BWT.srcs > sources_1 > new > test_bwt_transform.v

```
module BWT_transform_tb;

  // Sygna?y testowe
  logic clk;
  logic rst;
  logic start;
  logic [0:31] data_in;
  logic [0:31] data_out;
  logic done;



  logic [0:31] data_out_Pipeline;
  logic done_Pipeline;
  // DUT
  BWT_transform dut (
    .clk(clk),
    .rst(rst),
    .start(start),
    .data_in(data_in),
    .done(done),
    .data_out(data_out)
  );

  BWT_transform_with_Pipeline dut2 (
    .clk(clk),
    .rst(rst),
    .start(start),
    .data_in(data_in),
    .done(done_Pipeline),
    .data_out(data_out_Pipeline)
  );



  // Generacja sygna?u zegara
  always #5 clk = ~clk;

  // Inicjalizacja wej??
  initial begin
    clk = 0;
    rst = 1;
    start=0;
    data_in = '0;


    #10 rst = 0;
    data_in = 'h0AB2C1AF;

    //hFCB01a2a

    #70 rst = 1;
     data_in = 'h0AB2C1AF;
;



    #70 rst = 0;
     start=1;
     data_in = 'h0AB2C1AF;

    #300;

  end

  // Wypisanie wynik
  always @(posedge clk) begin
    $display("data_in = %h, data_out = %h", data_in, data_out);
  end

endmodule
```

Rysunek 17: Testbench do przeprowadzenia symulacji wydajności algorytmu

Na poniższych przebiegać można porównać czasy wykonania algorytmów:

Rysunek 18: Wyniki symulacji porównawczej

W tym przypadku zastosowanie potoku pozwoliło skrócić czas symulacji o 25% !!!

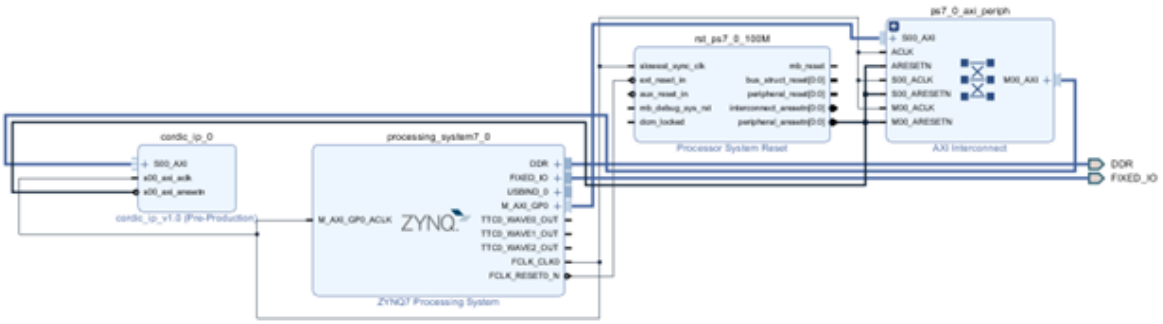
8 Przygotowanie modułu na platformie FPGA

Poniżej przedstawiony został moduł cordic `Ip_v1_0_S00_AXI`:

```
wire ARESET;
assign ARESET = ~S_AXI_ARESETN;
//Code to transfer output data from cordic processor to output registers
wire [C_S_AXI_DATA_WIDTH-1:0] slv_wire2;
wire [C_S_AXI_DATA_WIDTH-1:0] slv_wire3;
always @( posedge S_AXI_ACLK )
begin
 slv_reg2 <= slv_wire2;
 slv_reg3 <= slv_wire3;
end
//Assign zeros to unused bits
assign slv_wire2[31:1] = 31'b0;
BWT_transform BWT_transform_inst( S_AXI_ACLK, //clock,
                        ARESET, //reset,
                        slv_reg0[0], //start
                        slv_reg1[31:0], //angle_in
                        slv_wire3[31:0],//sin_out
                        slv_wire2[0] //ready_out,
                        );
// User logic ends

    endmodule

module BWT_transform(clk, rst, start, data_in, data_out, done);

  // Okre□lenie szeroko□ci wyrazu (wprost zale□ne od szeroko□ci data_in)
  parameter integer dl_wyraz = 32;
  // Okre□lenie szeroko□ci litery
  parameter integer szer_litery = 4;
  // Okre□lenie ilo□ci liter
  parameter integer ilosc_liter = dl_wyraz / szer_litery;

  input wire clk;              // Sygna□ zegara
  input wire rst;              // Sygna□ resetu
  input wire [0:dl_wyraz-1] data_in;    // Dane wej□ciowe
  output wire [0:dl_wyraz-1] data_out;   // Dane wyj□ciowe
  output reg done;             // Flaga ko□ca oblicze□
  input start;

  // Tablica, na kt□rej si□ wszystko dzieje (rotacja oraz sortowanie)
  reg [0:dl_wyraz-1] buffor[0:ilosc_liter-1];
  wire [0:dl_wyraz-1] buffor_rotacji[0:ilosc_liter-1];
  // Przechowanie warto□ci bazowej w trakcie wymiany indeks□w podczas sortowania
  reg [0:dl_wyraz-1] data_var;
  // Zawiera ostatnie bity tym samym wyraz wyj□ciowy
//wire [0:dl_wyraz-1] dana_wyj;

  parameter S1 = 3'h01, S2 = 3'h02, S3 = 3'h03, S4 = 3'h04, S5 = 3'h05, S6 = 3'h06, S7 = 3'h07;
  reg [3:0] state;
  integer x, y;
  integer zacznij;
  integer scal;


  generate
    genvar numb;
    for (numb = 1; numb < ilosc_liter; numb = numb + 1) begin : BWT_step_loop
      BWT_step #(numb, dl_wyraz, szer_litery) BWT_step_part(clk, zacznij, buffor[0], buffor_rotacji[numb]);
    end

    genvar numb2;
    for (numb2 = 0; numb2 < ilosc_liter; numb2 = numb2 + 1) begin : BWT_last_letter_loop
      BWT_last_letter #(numb2, dl_wyraz, szer_litery) BWT_last_letter_part(clk, scal, buffor[numb2],
                        data_out[numb2*szer_litery : (szer_litery*(numb2+1)-1)]);
    end
  endgenerate

  always @(posedge clk)
  begin
    if (rst) begin
      state <= S1;
      //data_out <= 0;
      done <= 0;
      scal=0;
      zacznij = 0;
    end
    else begin
      case(state)
        S1: begin
          if (start == 1'b1) state <= S2;
          else state <= S1;
        end
        S2: begin

          buffor[0] <= data_in;
          x = 0;
```

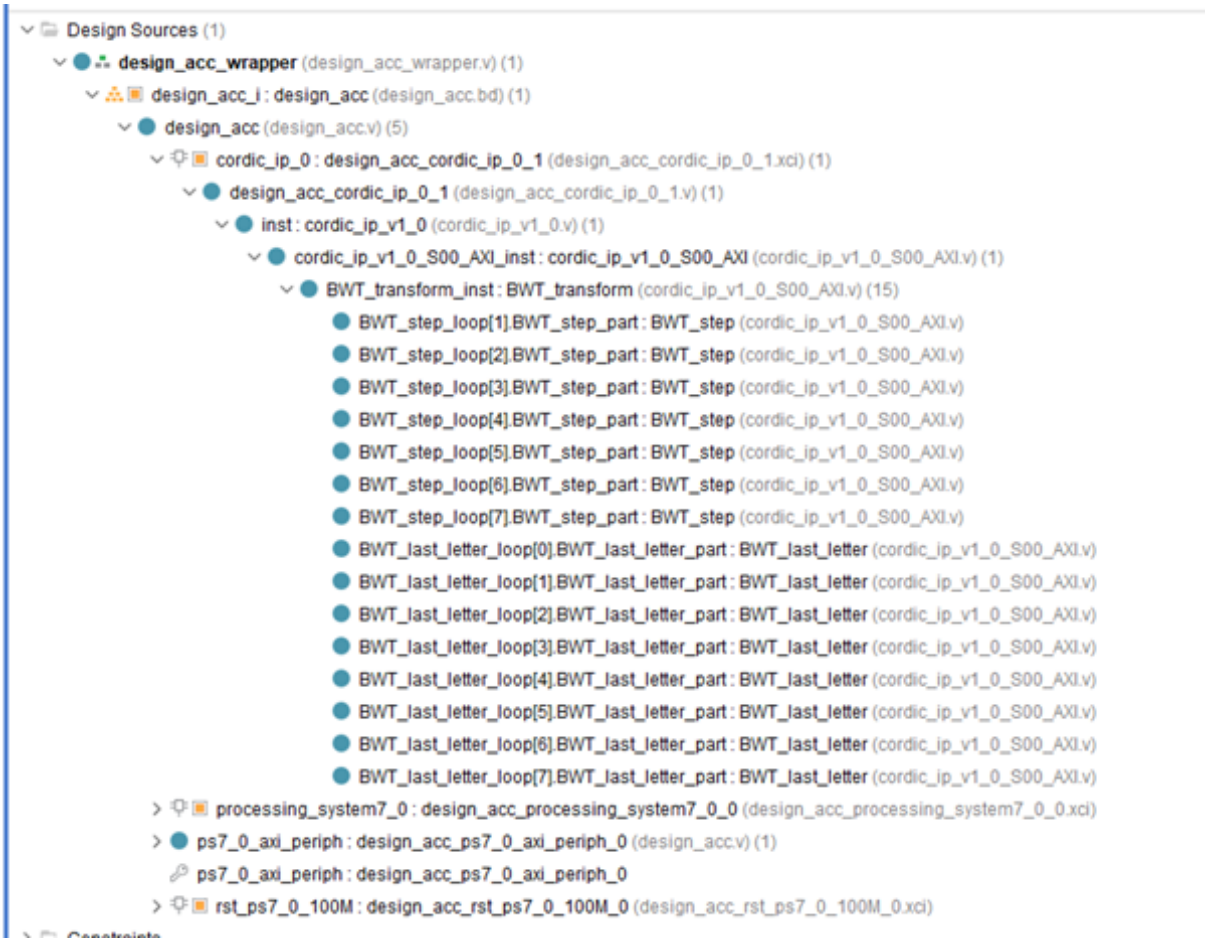
Rysunek 19: Cordic IP transformaty

Pniżej przedstawiony został przygotowany diagram blokowy Microblaze



Rysunek 20: Schemat blokowy

Struktura projektu:



Rysunek 21: Struktura projektu

Wyeksportowanie do platformy sprzętowej do SDK:

```
/***************************** Include Files *********************************/
#include "xil_io.h"
#include "xparameters.h"
#include "cordic_ip.h"

/************************** user definitions *****************************/

//Cordic processor base addres redefinition
#define CORDIC_BASE_ADDR      XPAR_CORDIC_IP_0_S00_AXI_BASEADDR
//Cordic processor registers' offset redefinition
#define CONTROL_REG_OFFSET    CORDIC_IP_S00_AXI_SLV_REG0_OFFSET
#define ANGLE_REG_OFFSET      CORDIC_IP_S00_AXI_SLV_REG1_OFFSET
#define STATUS_REG_OFFSET     CORDIC_IP_S00_AXI_SLV_REG2_OFFSET
#define RESULT_REG_OFFSET    CORDIC_IP_S00_AXI_SLV_REG3_OFFSET
//Cordic processor bits masks
#define CONTROL_REG_START_MASK (u32)(0x01)
#define STATUS_REG_READY_MASK (u32)(0x01)



/***************************** calculateCordicVal function ********************

*/

int calculateBWT(u32 input_string, s32* result)
{
u32 data  = input_string;

//Debug
//  result = CORDIC_IP_mReadReg(CORDIC_BASE_ADDR, RESULT_REG_OFFSET);


//Send data to data register of cordic processor
    CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, ANGLE_REG_OFFSET, data);
//Start cordic processor - pulse start bit in control register
    CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, CONTROL_REG_OFFSET, CONTROL_REG_START_MASK);
    CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, CONTROL_REG_OFFSET, 0);
//Wait for ready bit in status register
    while( (CORDIC_IP_mReadReg(CORDIC_BASE_ADDR, STATUS_REG_OFFSET) & STATUS_REG_READY_MASK) == 0);
//Get results
    *result = CORDIC_IP_mReadReg(CORDIC_BASE_ADDR, RESULT_REG_OFFSET);

    return 1;
}
```

Rysunek 22: Zapis rejestrów funkcji calculateBWT

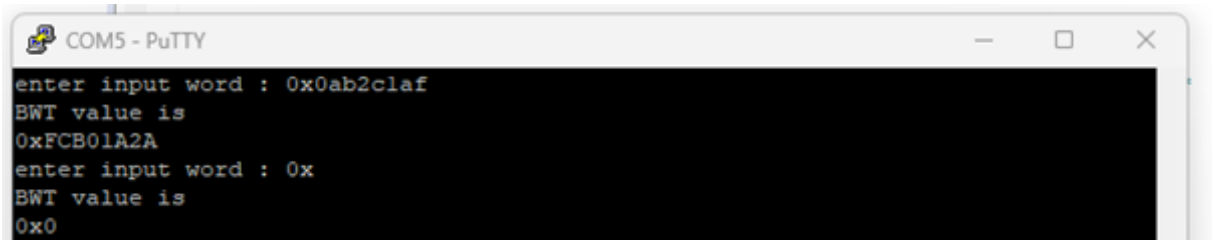
Poniżej przedstawiony został plik "main.c":

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"

//Define PI in fxp(12:10)
#define PI 3215

teBWT(u32 input_string, s32* result);

u32 readHexVal() {
    u32 ret = 0;
    char8 c;

    for (int i = 0; i < 32; i++) {
        outbyte(c = inbyte());

        if (c >= '0' && c <= '9') {
            ret = (ret << 4) + (c - '0');
        } else if (c >= 'A' && c <= 'F') {
            ret = (ret << 4) + (c - 'A' + 10);
        } else if (c >= 'a' && c <= 'f') {
            ret = (ret << 4) + (c - 'a' + 10);
        } else {
            break;
        }
    }

    return ret;
}


int main()
{
u32 input_string = 0;
s32 result;

    init_platform();

    while(1){
        print("enter input word : 0x");
        input_string = readHexVal();
        print("\n\r");

        calculateBWT(input_string, &result);

        print("BWT value is ");
        print("\n\r");
        xil_printf("0x%X", result);
        print("\n\r");
    }

}
```

Rysunek 23: Plik główny

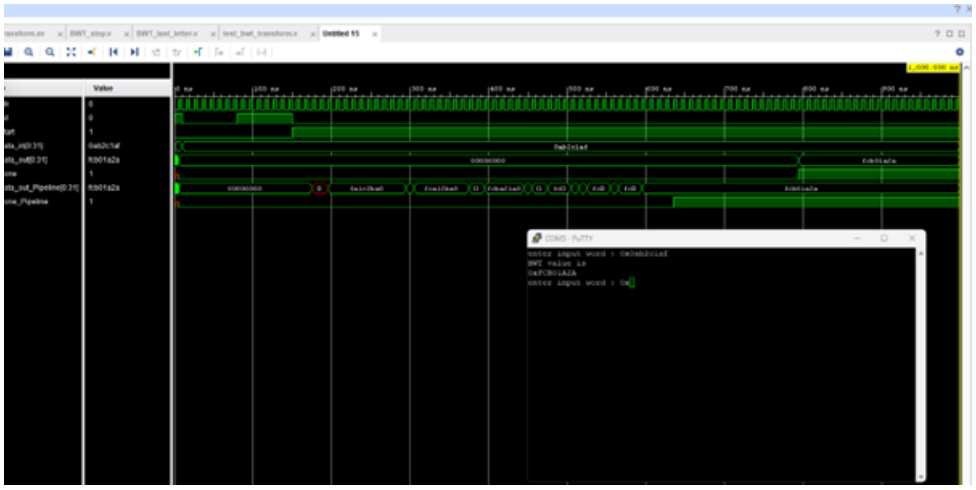
9 Uruchomienie programu oraz zestawienie z pozostałymi wynikami

Uruchomienie algorytm na platformie Zedboard :

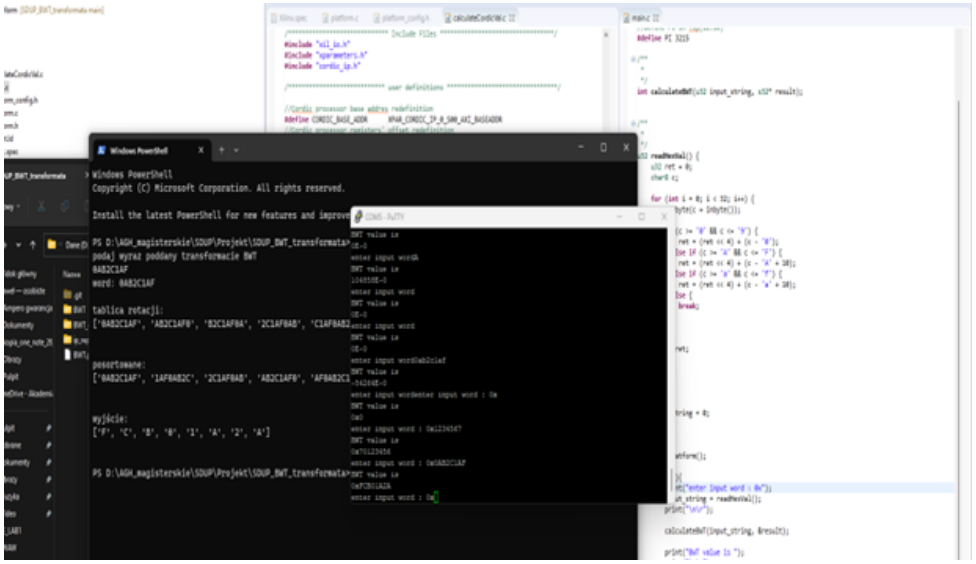


Rysunek 24: Okno programu na platformie Zedboard

Poniższe ilustracje zestawiają wyniki otrzymane w poszczególnych etapach projektu:



Rysunek 25: Otrzymane wyniki programu



Rysunek 26: Otrzymane wyniki programu