Projekt SYCYF

-

Zespół nr 9

Mateusz Dybicz

Mateusz Mięgoć-Kowalski

Jakub Kubiński

Karol Kaproń

Jakub Sulikowski

Politechnika Warszawska, Instytut Telekomunikacji

22 marca 2023

Historia zmian

Wersja	Data	Autor	Opis zmian
1.0	14.03.2022	MD	Pierwsza wersja raportu Etapu 1
1.1	15.03.2022	KK	Uzasadnienie wyboru Design Thinking
1.2	15.03.2022	KK	Dodanie literatury
1.3	04.04.2022	KK, JK, MMK	Informacje podstawowe
1.4	24.04.2022	JS	Pierwsza wersja schematu blokowego
1.5	26.04.2022	JK, MMK, JS, MD, KK	Opisanie zagadnień Etapu III
1.6	07.05.2022	JS	Dodanie rozdziału dla Etapu IV
1.7	12.05.2022	JS	Dodanie diagramu algorytmu korelacji.

Spis treści

1. Wstęp

Projekt ma na celu zrealizowanie przedsięwzięcia, jakim jest stworzenie sprzętowego modułu DSP obliczającego parametry określające położenie drona, na podstawie założeń ustalonych z koordynatorem projektu z przedmiotu SYCYF realizowanego w ramach programu studiów na kierunku Telekomunikacja

2. Organizacja prac

Rozdział ten jest opisem działań wykonanych przez nas w ramach EtapuI. W tym:

— podejście Design Thinking,
— wybór sposobu zarządzania projektem
— organizacja warsztatu pracy, dobór narzędzi

2.1. Design Thinking

Design Thinking to metoda pozwalająca na tworzenie rozwiązań dopasowanych do potrzeb użytkowników. Jednym z najpopularniejszych jego modeli jest **Double Diamond**, który jest procesem szerokiego rekonesansu i eksplorowania danego tematu, prowadzący do podjęcia konkretnych, precyzyjnych działań. Proces ten składa sie z 4 etapów:
— Etap 1 - **Discover**. Polega na zbieraniu informacji z wielu źródeł, pozyskiwaniu dużej ilości wiedzy, w celu zrozumienia potrzeb użytkowników.
— Etap 2 - **Define**. Polega na wyciąganiu wniosków z zebranych informacji.
— Etap 3 - **Develop**. Polega na generowaniu jak największej ilości pomysłów na usługę, która rozwiąże dany problem.
— Etap 4 - **Deliver**. Polega na wybraniu jednego, najlepszego pomysłu, stworzenie prototypu i testowanie z użytkownikami.

Zdecydowaliśmy się na tą metodę, ponieważ przez częste weryfikowanie poszczególnych elementów tworzonego rozwiązania pozwoli to nam na uniknięciu podążania w złym, tzn. niepożądanym przez użytkownika, kierunku. Takie podejście pozwoli nam łatwiej pozyskać umiejętności miękkie, które są pożądane przez pracodawców.

2.2. Zarządzanie projektem

Udało nam się porozumieć i ustalić liderów dla poszczególnych etapów:
— Etap`I` - **Mateusz Dybicz**
— Etap`II` - **Karol Kaproń**
— Etap`III` - **Mateusz Mięgoć-Kowalski**
— Etap`IV` - **Jakub Kubiński**
— Etap`V` - **Jakub Sulikowski**

2.2.1. Metody

Główną metodą tworzenia projektu będą spotkania na serwerze Discord, gdzie będziemy wspolnie rozwiązywać zadania i problemy.

2.2.2. Narzędzia

Głównymi narzędziami do komunikacji w zespole będą **Microsoft Teams** oraz **Discord**. Raporty z poszczególnych etapów oraz całego projektu tworzone będą w programie **Overleaf**. Narzędziem do przekazywania wszelakich plików w grupie będzie **GitLab**. Do rozwiązania zamodelowania i wizualizacji DSP wykorzystamy **Pythona** i **Matlaba**. A do narzędzi HDL wykorzystamy **Intel (Altera) Quartus**, a do kodowanie użyjemy **Verilog** lub **VHDL**.

3. Informacje podstawowe

Ten rozdział i podrozdziały będą poświęcone opisowi zadania realizaującego w ramach Etapu II. Omówimy zagadnienia, które według nas są przydatne do realizacji projektu.

3.1. DSP

Digital signal processing (DSP) to wykorzystanie przetwarzania cyfrowego, takiego jak komputery lub bardziej wyspecjalizowane procesory sygnałowe, do wykonywania szerokiej gamy operacji przetwarzania sygnałów. Przetwarzane w ten sposób sygnały cyfrowe to sekwencje liczb, które reprezentują próbki zmiennej ciągłej w dziedzinie takiej jak czas, przestrzeń lub częstotliwość. W technice cyfrowej sygnał cyfrowy jest reprezentowany jako ciąg impulsów - zer i jedynek.

3.2. Próbkowanie sygnału

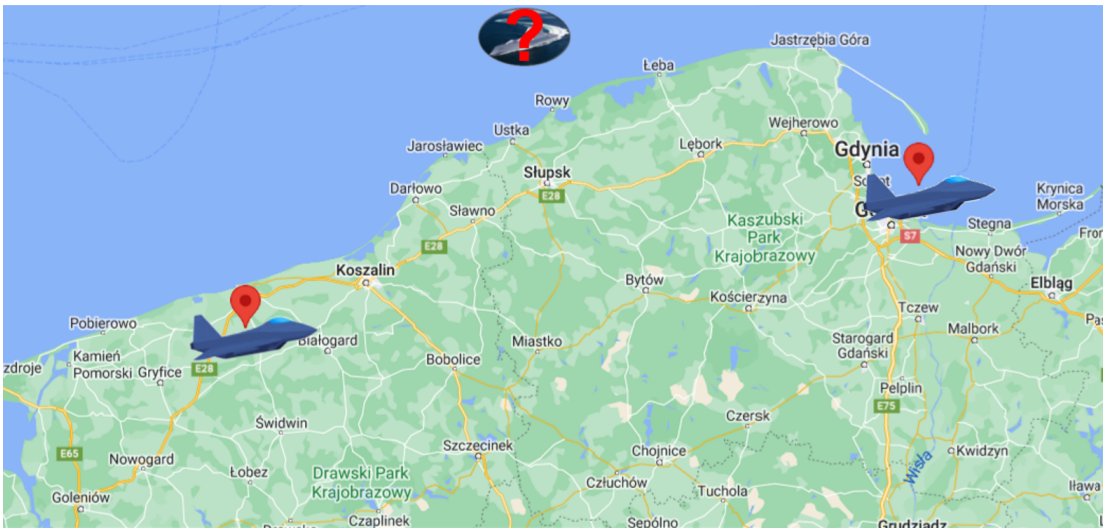
By cyfrowo analizować sygnał analogowy należy przekonerwtować go na sygnał cyfrowy za pomocą konwertera analogowo-cyforwego(ADC).Oznacza to że sygnał należy podzielić w równych odstępach czasu i zmierzyć jego wartość chwilową (dyskretyzacja). Nastepnie należy podzielić dziedzinę wartość na poziomy reprezentacji

Rys. 1. Schemat systemu DSP

i przypisać wartość chwilową sygnału do najbliższego poziomu reprezentacji (kwantyzacja). Poprzez konwersje cyfrowo-analogową (DAC), przy wykorzystaniu twierdzenia Nyquista–Shannona i uwzględnieniu błędu kwantyzacji, jesteśmy w stanie przekonwertować go spowrotem na sygnał analogowy.

3.3. Technika Lokalizacji

Do lokalizacji drona nawodnego uważamy, że dobrym pomysłem będzie wykorzystanie wysłanego przez samolot sygnału w punkcie A i w punkcie B o ściśle określonej zawartości na danej częstotliwości. Wygenerowane przez drona echo sygnału, które następnie zostanie odebrane przez samolot, umożliwi pomiar czasu. Znajomość położenia samolotu w punkcie A i B oraz czasu wysłania sygnału pozwoli nam na znalezienie punktu w którym znajduje się dron. Istotnym aspektem jest także odseparowanie szumu który na pewno będzie istotnie wpływać na wysyłanych i odbieranych przez drona sygnałach, lecz nad rozwiązaniem tego problemu pochylimy się w III części projektu.



Rys. 2. Zobrazowanie konceptu

Zakładając, że nasz dron nie porusza się możemy skorzystać z metody trilatelacji, która mierzy czas potrzebny do osiągnięcia przez sygnał radiowy swojej pozycji, a gdy znane są czasy z stacji odbiorczych, można obliczyć pozycję odbiornika.

3.4. Narzędzia do przetworzenia sygnału

Do przetworzenia sygnału i uzyskania potrzebnych parametrów do obliczenia lokalizacji drona nawodnego wykorzystamy język programowania **Python** i MatLab. W Pythonie do przetworzenia sygnału wykorzystamy moduł scipy.signal, który zawiera potrzebne metody do obliczenia potrzebnych rzeczy. Za pomocą matlaba przeanalizujemy sygnał i zasymulujemy system DSP w dużo krótszym czasie niż pozwoli na to tradycyjny język programowania C lub C++. Obliczenia naszego algorytmu DSP zostaną zaimplementowane na specjalizowanych układach scalonych FPGA. Do zaprogramowania takiego układu użyjemy języka VHDL i środowiska Quartus.

4. Koncepcja

Ten rozdział i podroździały są poświęcone przedstawieniu koncepcji oraz realizacji modelu referencyjnego w środowisku python.

Rys. 3. Diagram blokowy koncepcji rozwiązania

4.1. Koncepcja rozwiązania (Omówienie diagramu blokowego)

1. Najpierw by móc wykonać jakiekolwiek operacje należy załadować pliki txt z sygnałami, a następnie zamienić je na tablice próbek.

2. Następnie posiadając sygnały w postaci tablic próbek można poszukać sygnału wysłanego w sygnałach odebranych. Można to zrobić korzystając z zagadnienia jakim jest korelacja sygnałów, której funkcja ma maksimum w argumencie, w którym sygnały są najlepiej dopasowane. Żeby zastosować to zagadnienie w pythonie korzystamy z biblioteki Scipy, w której ta operacja jest zaimplementowana.

3. Posiadając próbki sygnałów, które są najbardziej zkorelowane z sygnałem wysłanym możemy obliczyć różnicę czasu między wysłaniem sygnału, a odebraniem wiedząc, że sygnał był rejestrowany z częstotliwością 10^6 próbek/sek. Ta wartość pozwala nam na obliczenie odległości drona od samolotu.

4. Następujacym wzorem liczymy odległość samolotu od drona:

$$d = c * t/2$$

Gdzie d - odległość, c - prędkość światła w powietrzu, t - różnica czasu między wysłaniem sygnału a odebraniem. Następnie rysujemy dwa okręgi gdzie r = d i przecięcia pokazują nam dwa punkty, w których może się znajdować dron.

4.2. Korelacja wzajemna

Korelacja wzajemna jest odpowiedzialna za ukazywanie zależności między zmiennymi. W naszym przypadku jest ona potrzebna w celu określenia podobieństwa dwóch sygnałów (wysłanego i odebranego). Obliczenie jej działa na zasadzie przesunięć czasowych, przesuwamy jeden z sygnału względem drugiego, mnożymy powtarzające się próbki i sumujemy wyniki. Używamy jej również w celu wyznaczenia opóźnienia dwóch sygnałów względem siebie. Aby to osiągnąć poszukujemy maksymalną wartość korelacji i odejmujemy od jej położenia długość drugiego sygnału pomniejszonego o 1. W naszym kodzie korzystamy z korelacji za pomocą funkcji "scipy.signal.correlate".

4.3. Przykład ilustrujący działanie

Najlepszy przykładem, który dobrze zobrazuje działanie przedstawionego przez nas schematu blokowego jest nasze zadanie projektowe. Użyjemy do niego sygnału wysłanego oraz odebranego z samolotu w obydwu lokalizacjach "A" oraz "B".

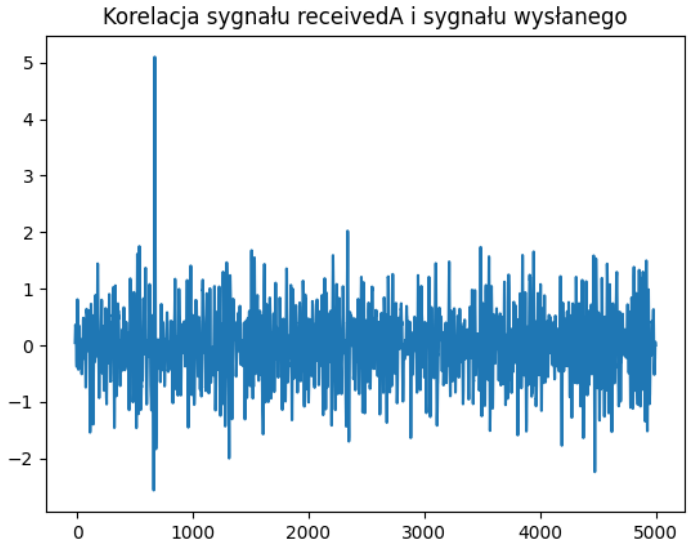
Inputem w naszym programie są sygnały w formie tekstowej:
— `Point A - received8.txt`
— `Point B - received8.txt`
— `sent8.txt`

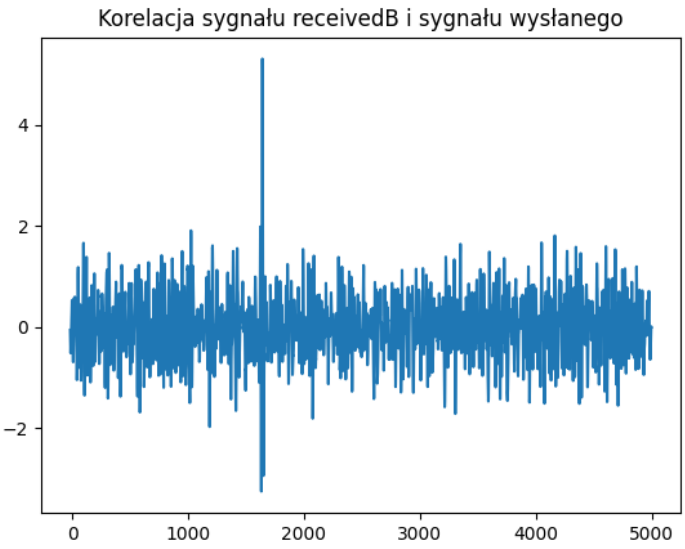
Te sygnały następnie są przetwarzane przez nasz program dzięki czemu otrzymujemy następujące wyniki:

```
Opóźnienie dla sygnału A względem sygnału wysłanego =  670  próbek
Opóźnienie sygnału B względem sygnału wysłanego =  1640  próbek
Promień z punktu A:  100.40318 km
Promien z punktu B:  245.76302 km
```

Rys. 4. Wyniki działania naszego programu na dostarczonych plikach .txt

Pierwsze dwa wyniki prezentują liczbę próbek, o którą oczekiwany sygnał został przesunięty. Kolejne dwa wyniki są już obliczonymi promieniami.





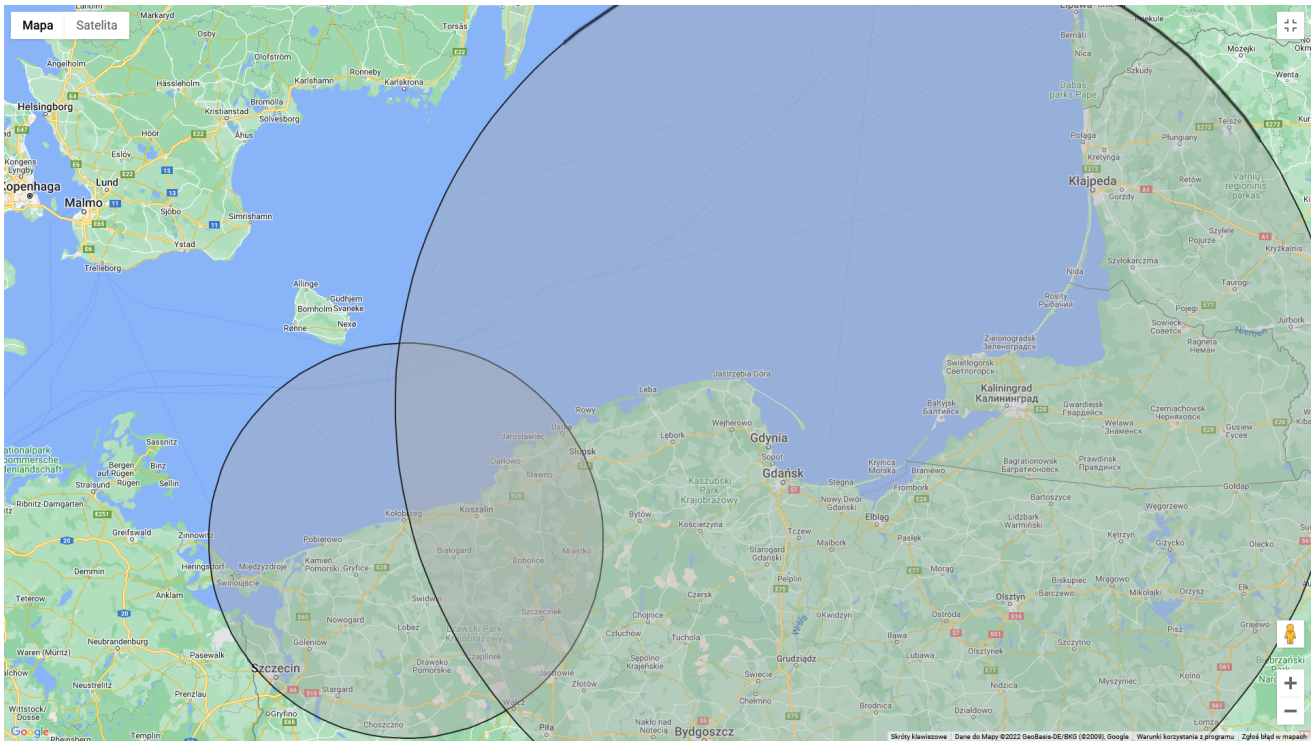
Powyższe dwa wykresy wygenerowane za pomocą biblioteki matplotlib przedstawiają korelację sygnału wysłanego z sygnałami otrzymanymi.

4.4. Program napisany w języku Python

Program napisany przez nas w Pythonie, wraz z komentarzami do niego, wygląda w następujący sposób:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import correlate

arrReceivedA = []
arrReceivedB = []
arrSent=[]

#Pobranie wartości dla próbek sygnału z pliku tekstowego i zapisanie ich w macierzy
with open("receivedA.txt",'r') as f:
    for line in f.readlines():
        line = line.replace('\n','')
        line = float(line)
        arrReceivedA.append(line)

with open("receivedB.txt",'r') as f:
    for line in f.readlines():
        line = line.replace('\n','')
        line = float(line)
        arrReceivedB.append(line)

with open("sent.txt",'r') as f:
    for line in f.readlines():
        line = line.replace('\n','')
        line = float(line)
        arrSent.append(line)

arrA = np.array(arrReceivedA)
arrB = np.array(arrReceivedB)
arrS = np.array(arrSent)

#Korelacja wzajemna dwóch sygnałów
correlateAS = correlate(arrA,arrS)
correlateBS = correlate(arrB,arrS)

#Pobranie indeksu maksymalnej wartości
prob1 = np.argmax(correlateAS)
prob2 = np.argmax(correlateBS)

#Okreslenie wartości opóźnienia
pointA = prob1 - (len(arrS)-1)
pointB = prob2 - (len(arrS)-1)

#Czas propagacji sygnału do drona wyrażona w sekundach
timeA = (pointA/1000000)/2
timeB = (pointB/1000000)/2

#Prędkość światła w powietrzu
v = 299711000

#Droga jaką pokonał sygnał
radiousA = v*timeA
radiousB = v*timeB

print("Promień z punktu A: ",round(radiousA/1000,5),"km")
print("Promien z punktu B: ",round(radiousB/1000,5),"km")

plt.figure(1)
print("Opóźnienie dla sygnału A względem sygnału wysłanego = ",pointA,"próbek")
plt.plot(np.arange(len(correlateAS))-(len(arrS)-1),correlateAS)
plt.title("Korelacja sygnału receivedA i sygnału wysłanego")

plt.figure(2)
print("Opóźnienie sygnału B względem sygnału wysłanego = ",pointB,"próbek")
plt.plot(np.arange(len(correlateBS))-(len(arrS)-1),correlateBS)
plt.title("Korelacja sygnału receivedB i sygnału wysłanego")
plt.show()
```

4.5. Wyznaczanie lokalizacji drona



Lokalizację wyznaczyliśmy odczytując współrzędne, punktu przecięcia dwóch okręgów, na mapie. Okręgi mają promienie równe odległościom drona od samolotu (tj. w przybliżeniu 100.4 km i 245.8 km), a ich środki to współrzędne samolotu w punktach: A (54°04'02.2"N 15°35'55.8"E) i B (54°27'30.1"N 18°46'25.0"E). Wykorzystując wiedzę o dronie wiemy, że jest to dron nawodny, więc za możliwe jego położenie uznaliśmy punkt, który znajduje się na wodzie. Lokalizacja drona ma współrzędne: 55.001544 N,15.551075 E. Ewentualny błąd w współrzędnych drona może wynikać z metody ich wyznaczenia.

5. Implementacja

5.1. Wstęp

W tym rozdziale i jego podrozdziałach zostanie opisana realizacja naszego systemu służącego do znalezienia położenia drona nawodnego.

5.2. HDL

W inżynierii komputerowej język opisu sprzętu (HDL) jest specjalistycznym językiem komputerowym używanym do opisywania struktury i zachowania układów elektronicznych, najczęściej cyfrowych układów logicznych.

Język opisu sprzętu wygląda bardzo podobnie do języka programowania, takiego jak C ; jest to opis tekstowy składający się z wyrażeń, instrukcji i struktur kontrolnych. Jedną z ważnych różnic między większością języków programowania a HDL-ami jest to, że HDL-y zawierają pojęcie czasu.

HDL stanowią integralną część systemów automatyzacji projektowania elektronicznego (EDA), zwłaszcza w przypadku złożonych układów, takich jak układy scalone specyficzne dla aplikacji, mikroprocesory i programowalne urządzenia logiczne.

5.3. System w HDL

Zaznajomiliśmy się z modelowaniem elementów systemu HDL aby zaprojektować układ i opisać jego działanie. Dzięki systemowi HDL mamy również możliwość przetestowania i zasymulowania działania układu zanim zostanie on zaimplementowany w struturze krzemu.

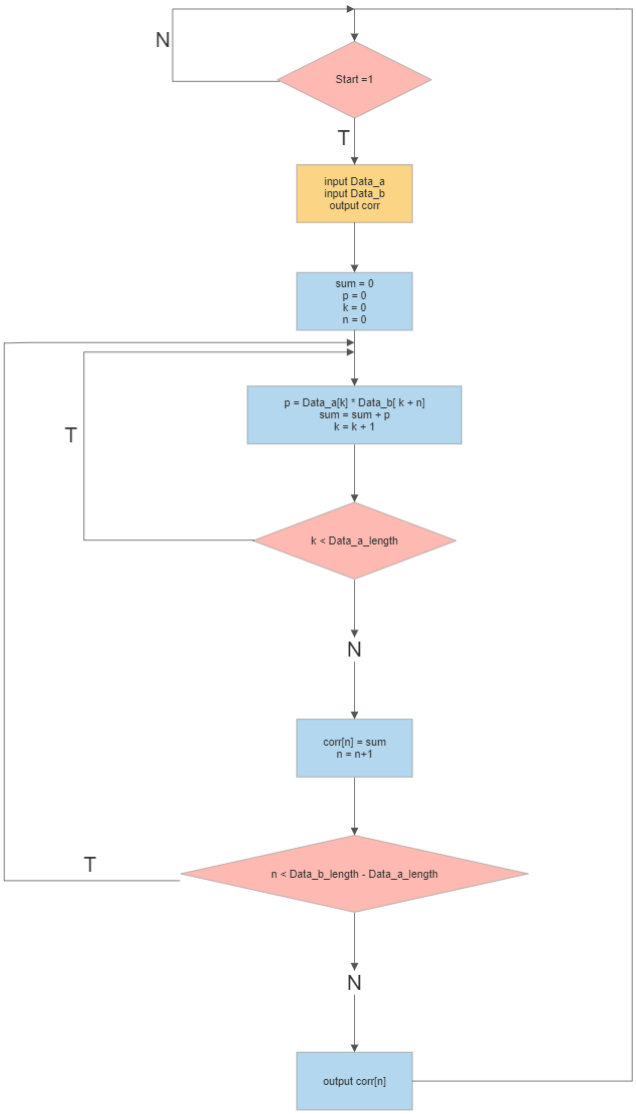
5.4. Koncepcja

5.4.1. Koncepcja realizacji: Podział

Problem, który mieliśmy rozwiązać czyli znalezienie drona nawodnego podzieliliśmy na różne sposoby realizacji. Poniżej znajdują się główne z nich.

— moduł konwersji próbek sygnału do bitów - realizacja programowa
— moduł DSP - realizacja sprzętowa
— moduł lokalizacji - realizacja programowa

5.4.2. Koncepcja realizacji: Diagram Algorytmu Korelacji



5.4.3. Koncepcja realizacji: Technika Projektowa

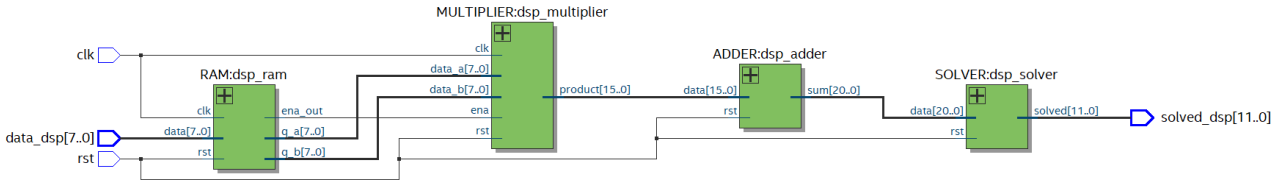
Moduł DSP w verilogu zaprojektowaliśmy techniką projektową RTL. RTL (Register Transfer Level) jest technika projektowa, w której projektuje się na poziomie rejestrów. Jest to wysoki poziom abstrakcji, który podczas procesu designowania upraszcza pogląd na projektowany system i przyśpiesza powyższy proces. System pobiera sygnał referencyjny i sygnał porównywany po konwersjach na NKB. Następnie oblicza korelację tych sygnałów dla danej próbki czasu i przekazuje do modułu porównawczego. Tam obliczona korelacja zostaje przyrównana do największej dotychczasowej korelacji

5.5. Realizacja sprzętowa

5.5.1. Ralizacja sprzętowa: Użyte narzędzia

Do zaprojektowania sprzętowego modułu DSP użyliśmy języka Verilog Hdl, który jest stworzony właśnie do takich zadań. Syntezę i kompilację przeprowadzaliśmy w środowisku Quartus Prime Lite w wersji 19.1. Symulację modułu przeprowadziliśmy w programie ModelSim - INTEL FPGA STARTER EDITION.

5.5.2. Realizacja sprzętowa: Diagram RTL



Nasz moduł DSP jest zbudowany z 4 mniejszych modułów. Pamięci RAM, modułu mnożnika, modułu sumatora oraz modułu rozwiązania. Połączenia między nimi są przedstawione na powyższym rysunku.

5.5.3. Realizacja sprzętowa: Moduł DSP

```
1  //Główny moduł
2  module DSP (
3
4     //Sygnały wejścia i wyjścia
5     input clk,
6     input rst,
7     input ena,
8     output [11:0] solved_dsp
9
10 );
11 //Lokalne zmienne
12 wire [7:0] q_a_dsp;
13 wire [7:0] q_b_dsp;
14 wire ena_ram;
15 wire [15:0] product_multi;
16 wire [20:0] sum_dsp;
17
18 RAM dsp_ram(
19   .rst    (rst),
20   .clk    (clk),
21   .ena_in (ena),
22   .ena_out(ena_ram),
23   .q_a    (q_a_dsp),
24   .q_b    (q_b_dsp)
25 );
26
27 MULTIPLIER dsp_multiplier(
28   .data_a    (q_a_dsp),
29   .data_b    (q_b_dsp),
30   .clk       (clk),
31   .rst       (rst),
32   .ena       (ena_ram),
33   .product   (product_multi)
34 );
35
36 ADDER dsp_adder(
37   .rst    (rst),
38   .clk    (clk),
39   .data   (product_multi),
40   .sum    (sum_dsp),
41   .next_add  (next)
42 );
43
44 SOLVER dsp_solver(
45   .rst    (rst),
46   .data   (sum_dsp),
47   .solved (solved_dsp)
48 );
49
50 endmodule
```

5.5.4. Realizacja sprzętowa: ROM

```
module RAM
#(
    //Parametry ilości próbek w sygnałach
    parameter signalA_samples = 20,
    parameter corr_samples = 5000,
    parameter signalB_samples = 5000
)(

    //Wejścia i wyjścia
    input rst,
    input clk,
    input ena_in,
    output ena_out,
    output [7:0] q_a,
    output [7:0] q_b

);
initial begin
    // Skrypt ładujący dane z pliku do pamięci ram
    // synthesis translate_off
    $readmemb("D:/REPO/SYF_PROJ/SignalSBin.txt",memory_a);
    $readmemb("D:/REPO/SYF_PROJ/SignalABin.txt",memory_b);
    // synthesis translate_on

end
//Obszary pamięci RAM oraz inne lokalne zmienne
reg [7:0] memory_a [0:signalA_samples];
reg [7:0] memory_b [0:signalB_samples];
reg [7:0] out_a;
reg [7:0] out_b;
integer i = 0;
integer c = 0;
reg rdy = 1'b0;



always@(posedge clk)
    begin
    //Stan resetu
    if (rst) begin
        i = 0;
        c = 0;
        ri = 0;
        rc = 0;
        rdy = 1'b0;

    end else
    begin
    //Odczytywanie pamięci i podawanie dalej kolejnych wartości zgodnie z algorytmem korelacji
          if (ena_in) begin

             if ( c <= (signalB_samples - signalA_samples)) begin

                if(i < signalA_samples) begin
                   out_a = memory_a [i];
                   out_b = memory_b [i + c];
```

```
    //Wejścia i wyjścia
    input rst,
    input clk,
    input ena_in,
    output ena_out,
    output [7:0] q_a,
    output [7:0] q_b

);
initial begin
    // Skrypt ładujący dane z pliku do pamięci ram
    // synthesis translate_off
    $readmemb("D:/REPO/SYF_PROJ/SignalSBin.txt",memory_a);
    $readmemb("D:/REPO/SYF_PROJ/SignalABin.txt",memory_b);
    // synthesis translate_on

end
 //Obszary pamięci RAM oraz inne lokalne zmienne
 reg [7:0] memory_a [0:signalA_samples];
 reg [7:0] memory_b [0:signalB_samples];
 reg [7:0] out_a;
 reg [7:0] out_b;
 integer i = 0;
 integer c = 0;
 reg rdy = 1'b0;

 always@(posedge clk)
    begin
    //Stan resetu
    if (rst) begin
        i = 0;
        c = 0;
        rdy = 1'b0;
    end else
    begin
    //Odczytywanie pamięci i podawanie dalej kolejnych wartości zgodnie z algorytmem korelacji
          if (ena_in) begin
             if ( c <= (signalB_samples - signalA_samples)) begin
                if(i < signalA_samples) begin
                   out_a = memory_a [i];
                   out_b = memory_b [i + c];
                   i = i + 1;
                   rdy = 1;
                end else begin
                   c = c + 1;
                   i = 0;
                   rdy = 0;
                end
             end
          end
       end
    end

    assign ena_out = rdy;
    assign q_a = out_a;
    assign q_b = out_b;

endmodule
```

5.5.5. Realizacja sprzętowa: Moduł Multiplier

```
module MULTIPLIER (


    //Sygnały wejścia i wyjścia
    input rst,
    input [7:0] data_a,
    input [7:0] data_b,
    input clk,
    input ena,
    output [15:0] product

);
 //Lokalne zmienne
 reg [15:0] prod;
 reg [2:0] gate_local;


always@(posedge clk)begin
    //Stan resetu
    if(rst) begin
        prod = 0;
    end else
    //Mnożenie dwóch podanych wartości oraz przekazanie ich dalej
    begin
        if (ena) begin
            prod = data_a * data_b;

    end
end
end


    assign product = prod;


endmodule
```

5.5.6. Realizacja sprzętowa: Moduł Adder

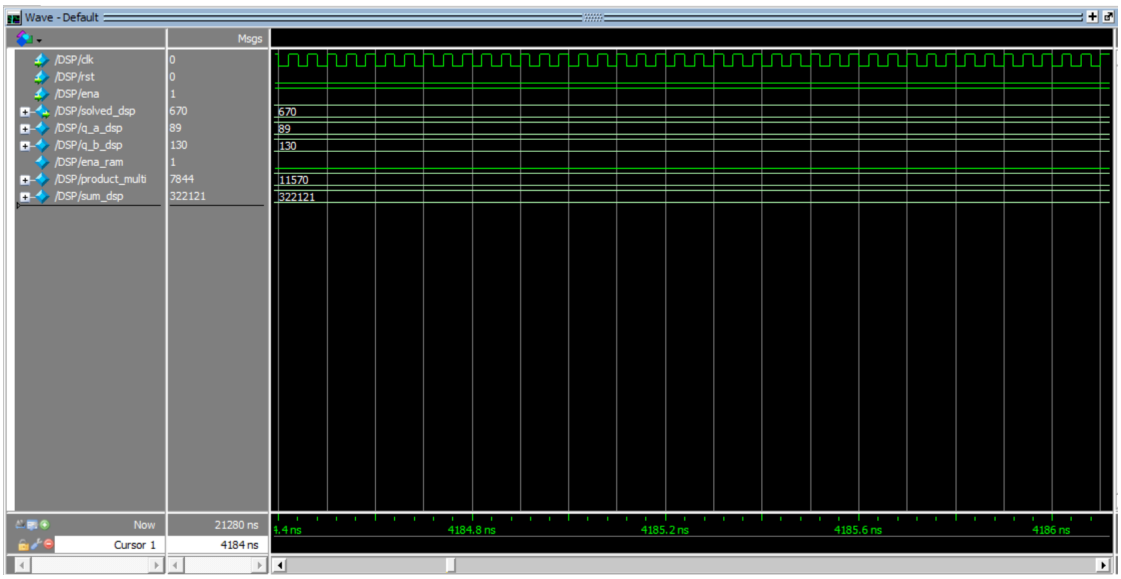
```
module MULTIPLIER (

    //Sygnały wejścia i wyjścia
    input rst,
    input [7:0] data_a,
    input [7:0] data_b,
    input clk,
    input ena,
    output [15:0] product

);

//Lokalne zmienne
reg [15:0] prod;
reg [2:0] gate_local;


always@(posedge clk)begin
   //Stan resetu
   if(rst) begin
      prod = 0;
   end else
   //Mnożenie dwóch podanych wartości oraz przekazanie ich dalej
   begin
      if (ena) begin
         prod = data_a * data_b;

   end
end
end


    assign product = prod;


endmodule
```

5.5.7. Realizacja sprzętowa: Moduł Solver

```
module SOLVER
#(
   //Parametr ilości argumentów funkcji wyjściowej korelacji
   parameter corr = 4980

)(
   //Sygnały wejścia i wyjścia
   input rst,
   input [20:0] data,
   output [11:0] solved

);

//Lokalne zmienne
reg [11:0] solved_local;
integer cnt;
integer saved_cnt;
reg [20:0] saved = 0;


always@(*) begin
   //Stan resetu
      if(rst)begin
         solved_local = 0;
         cnt = -1;
         saved_cnt = 0;
         saved = 0;
      /*Znajdowanie największej wartości z podanych przez sumator sum poprzez porównywanie kolejnych wartości do
      siebie i zapamiętywanie największej z nich */
      end else begin
      if (cnt < corr) begin
         if (data > saved) begin


            saved = data;
            saved_cnt = cnt;

         end
         cnt = cnt + 1;
      end
      if (cnt == corr) begin

            solved_local = saved_cnt;

      end
   end
   end


assign solved = solved_local;


endmodule
```

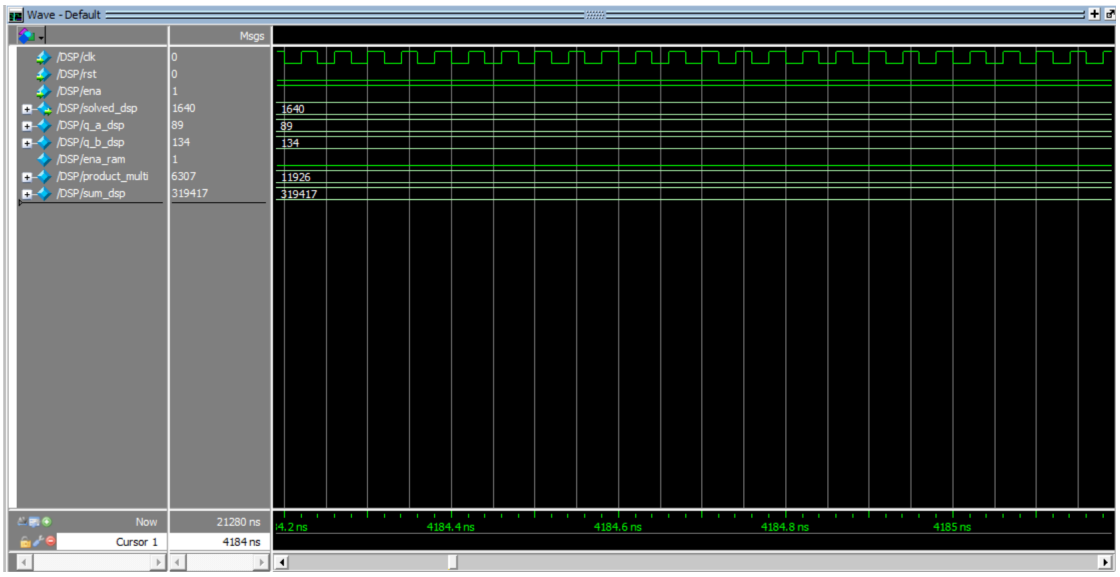
5.6. Symulacja

5.6.1. Symulacja: Sygnał Sent i Point A



W powyższej symulacji użyto wcześniej przekonwertowanych plików z danymi w postaci bajtów danych. Wynik symulacji, w postaci opóźnienia w postaci próbek, jest zgodny z otrzymanym przy tworzeniu modelu w Pythonie. Symulacja trwała 4184ns.

5.6.2. Symulacja: Sygnał Sent i Point B



Tak samo jak wcześniej użyto danych z plików w tej samej postaci. W tej symulacji ponownie wynik, w postaci opóźnienia w postaci próbek, jest zgodny z otrzymanym przy tworzeniu modelu w Pythonie. Symulacja trwała 4184ns.

6. Testowanie

6.1. Testbench

Podstawowe zastosowaniem języka HDL jest symulacja, gdzie bada się działanie układu dla zweryfikowania poprawności projektu. Symulację można porównać do badania fizycznego układu, którego wejścia podłączamy do stymulatora (generatora testów) i obserwujemy wyjścia. Symulacja kodu HDL to wykonywanie wirtualnego eksperymentu, w którym fizyczny układ zastąpiono opisem HDL. Dodatkowo można wykorzystać HDL do opisu generatora testów i modułu kolekcjonującego odpowiedzi układu i porównującego je ze wzorcem.

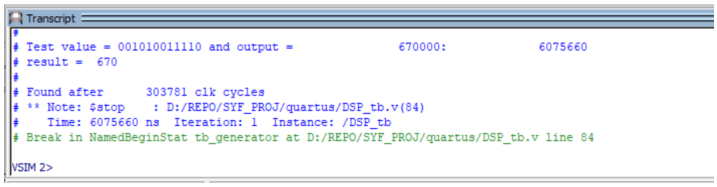
6.2. Kod

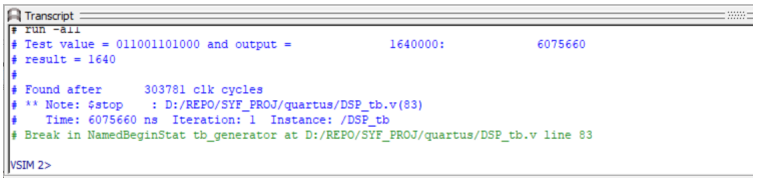
```
// synthesis translate_off
`timescale 1 ns/1 ps
`define RTL_SIMULATION
module DSP_tb();

localparam T = 20;
localparam TD = 3;
localparam sTime = 1000 * T;




reg clk;
reg rst;
reg ena;
wire [11:0] output_rsp;


integer file_uns;


DSP UUT(
 .ena (ena),
 .clk (clk),
 .rst (rst),
 .solved_dsp   (output_rsp)
);

initial begin : clock_generation
    clk = 0;
    #(T/2);
    forever begin
        #(T/2) clk = ~clk;
    end

end

initial begin : reset
    ena = 0;
    rst = 1;
    @(posedge clk);
    @(negedge clk);
    rst = 0;
```

```
end

reg [11:0] test_value = 670;
reg waiting = 0;
time first,next;
integer diff;
initial
begin: tb_generator
integer i;



   wait (rst == 1);
   wait (rst == 0);
   $display("%t: Zerowanie zakonczone\n",$time);
   file_uns = $fopen("results_unsigned.txt","w");

   @(posedge clk);
   ena = 1;
   first = $time;

   @(posedge clk);
   while (!waiting)begin
      @(posedge clk);
      if(output_rsp == test_value)begin
          waiting = 1;
      end
   end

   next = $time;
   diff = next - first;
   $fwrite(file_uns,"%d\n",$unsigned(output_rsp));
   $display("Test value = %b and output = %t:",test_value, output_rsp, $time);
   $display("result = %d\n",output_rsp);
   $display("Found after %d clk cycles", diff/T);
   $stop;

end

endmodule
// synthesis translate_on
```

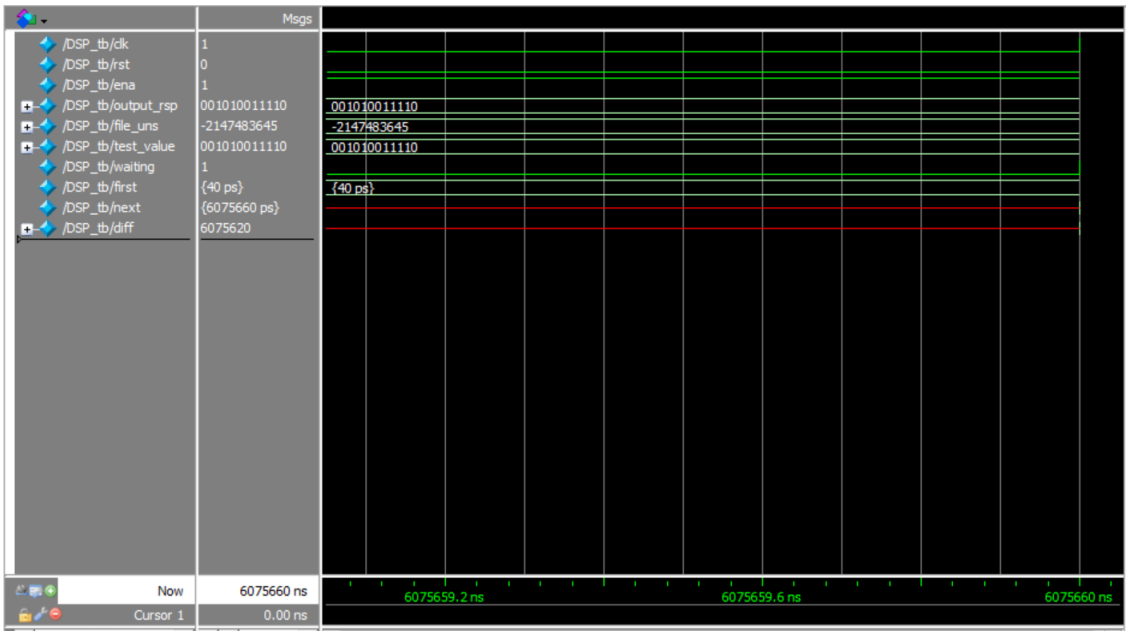
6.3. Nasze testy

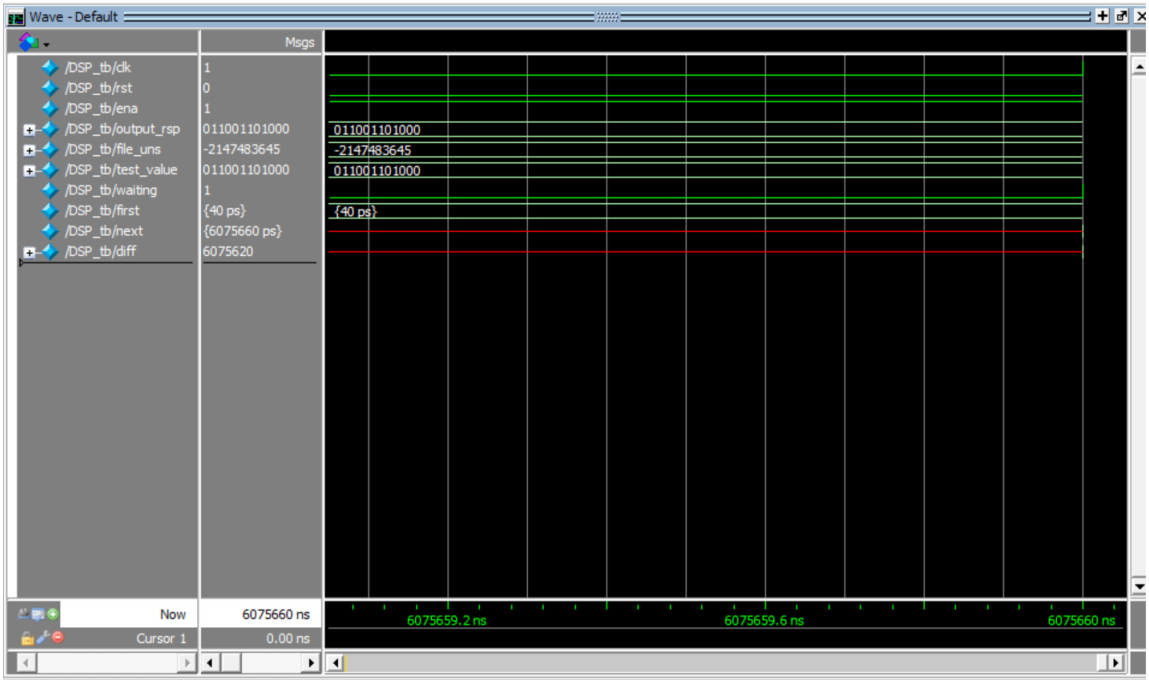
W naszym testbench'u używamy próbek pozyskanych z wav które są liczbami całkowitymi. Inicjuje moduł DSP i pobudzając go, w tym wypadku tylko sygnałami resetu, ena i zegara otrzymujemy wynik zwrotny. Jak na poniższych screenach. Na koniec w testbench'u tworzymy plik .txt z zapisanym wynikiem. Dodatkowo, udało nam się zsynchronizować działanie bloków programu.





6.4. Modelsim





7. Podsumowanie

Na tym etapie omówione zostały zrealizowane przez nas założenia projektowe oraz opis przebiegu projektu. Program działa poprawie, lecz doprowadzenie go do perfekcji wymagałoby dalszej pracy.

7.1. Zrealizowane cele

— organizacja pracy
— zaproponowanie wielu metod rozwiązań
— stworzenie prototypu działania w Pythonie
— implementacja w Verilogu
— testowanie kodu

7.2. Zaangażowanie zespołu podczas realizacji etapów

Zaangażowanie wszystkich członków zespołu w etapach I-IV wynosiło po 20%, natomiast w realizacji etapu V podział pracy wygląda następująco:
— Jakub Sulikowski - 40%
— Karol Kaproń - 15%
— Jakub Kubiński - 15%
— Magteusz Mięgoć-Kowalski - 15%
— Mateusz Dybicz - 15%

8. Literatura

— https://www.synergylab.pl/design-thinking-czyli-myslenie-projektowe/#software-house-efekty-stosowania-design-thinking
— https://gazeta.sgh.waw.pl/po-prostu-ekonomia/co-jest-design-thinking-i-dlaczego-jest-tak-popularne
— https://en.wikipedia.org/wiki/Digital_signal_processing
— https://en.wikipedia.org/wiki/Discrete_Fourier_transform
— https://docs.python.org/3/library/signal.html#example
— https://warszawa24.ovh/trilateracja-vs-triangulacja-poznaj-metody-radiolokacji

— https://sound.eti.pg.gda.pl/~greg/dsp/04-Splot.html
— https://www.mapdevelopers.com/draw-circle-tool.php