

SYCYF Project

-

Team nr 9

Mateusz Dybicz

Mateusz Mięgoć-Kowalski

Jakub Kubiński

Karol Kaproń

Jakub Sulikowski

Warsaw University of Technology, Institute of Telecommunications

10.06.2022

Versioning

Version	Date	Author	Description of changes
1.0	14.03.2022	MD	The first version of the Stage 1 report
1.1	15.03.2022	KK	Justification for choosing Design Thinking
1.2	15.03.2022	KK	Addition of literature
1.3	04.04.2022	KK, JK, MMK	Basic information
1.4	24.04.2022	JS	The first version of the block diagram
1.5	26.04.2022	JK, MMK, JS, MD, KK	Description of the issues of Stage III
1.6	07.05.2022	JS	Addition of a chapter for Stage IV
1.7	12.05.2022	JS	Addition of correlation algorithm diagram.

Spis treści

Versioning	1
1. Introduction	2
2. Work organization	2
2.1. Design Thinking	3
2.2. Project management	3
2.2.1. Methods	3
2.2.2. Tools	3

3. Basic information	3
3.1. DSP	3
3.2. Signal sampling	3
3.3. Localization technique	4
3.4. Signal processing tools	5
4. Concept	6
4.1. Solution Concept (Block Diagram Overview)	7
4.2. Cross-correlation	7
4.3. An example illustrating the operation	7
4.4. Program written in Python	8
4.5. Determining the location of the drone	10
5. Implementation	10
5.1. Intorduction	10
5.2. HDL	10
5.3. System in HDL	11
5.4. Concept	11
5.4.1. Implementation concept: Division	11
5.4.2. Implementation concept: Correlation Algorithm Diagram	12
5.4.3. Implementation concept: Design Technique	12
5.5. Hardware implementation	12
5.5.1. Hardware implementation: Tools used	12
5.5.2. Hardware implementation: RTL Diagram	13
5.5.3. Hardware implementation: DSP module	13
5.5.4. Hardware implementation: ROM	14
5.5.5. Hardware implementation: Multiplier module	15
5.5.6. Hardware implementation: Adder module	16
5.5.7. Hardware implementation: Solver module	16
5.6. Simulation	17
5.6.1. Simulation: Signal Sent and Point A	17
5.6.2. Simulation: Signal Sent and Point B	17
6. Testing	18
6.1. Testbench	18
6.2. Code	18
6.3. Our tests	19
6.4. Modelsim	19
7. Summary	20
7.1. Achieved goals	20
7.2. Involvement of the team during the implementation of stages	20
8. Literature	20

1. Introduction

The project aims to implement the project, which is the creation of a hardware DSP module for computing parameters determining the position of the drone, based on the assumptions agreed with the project coordinator of the SYCYF course implemented as part of the study program in the field of Telecommunications.

2. Work organization

This chapter is a description of the activities carried out by us as part of the StageI. Including:

- Design Thinking approach,
- choice of project management method
- organization of the workshop, selection of tools

2.1. Design Thinking

Design Thinking is a method that allows you to create solutions tailored to the needs of users. One of his most popular models is **Double Diamond**, which is a process of extensive reconnaissance and exploration of a given topic, leading to taking specific, precise actions. This process consists of 4 stages:

- Stage 1 - **Discover**. It consists in collecting information from many sources, acquiring a large amount of knowledge in order to understand the needs of users.
- Stage 2 - **Define**. It consists in drawing conclusions from the collected information.
- Stage 3 - **Develop**. It consists in generating as many ideas as possible for a service that will solve a given problem.
- Stage 4 - **Deliver**. It consists in choosing one, the best idea, creating a prototype and testing with users.

We decided on this method, because by frequent verification of individual elements of the created solution, it will allow us to avoid going in the wrong direction, i.e. undesirable by the user. This approach will make it easier for us to acquire soft skills that are desired by employers.

2.2. Project management

We managed to communicate and determine the leaders for each stage:

- StageI - **Mateusz Dybicz**
- StageII - **Karol Kaproń**
- StageIII - **Mateusz Mięgoć-Kowalski**
- StageIV - **Jakub Kubiński**
- StageV - **Jakub Sulikowski**

2.2.1. Methods

The main method of creating the project will be meetings on the Discord server, where we will solve tasks and problems together.

2.2.2. Tools

The main communication tools in the team will be **Microsoft Teams** and **Discord**. Reports from individual stages and the entire project will be created in the program **Overleaf**. The tool for transferring any files in the group will be **GitLab**. We will use **Python** and **Matlab** to solve the modeling and visualization of a DSP. For DSP module implementation we will use **Intel (Altera) Quartus**, within which we will code the solution in **Verilog** or **VHDL**.

3. Basic information

This chapter and subchapters will be devoted to the description of the executing task under Stage II. We will discuss issues that we believe are useful for the implementation of the project.

3.1. DSP

Digital signal processing (DSP) is the use of digital processing, such as computers or more specialized signal processors, to perform a wide range of signal processing operations. The digital signals thus processed are sequences of numbers that represent samples of a continuous variable in a domain such as time, space, or frequency. In digital technology, a digital signal is represented as a sequence of pulses - zeros and ones.

3.2. Signal sampling

To digitally analyze an analog signal, it needs to be converted to a digital signal using an analog-to-digital converter(ADC). This means that the signal should be divided in equal time intervals and its instantaneous value measured (discretization). Next, divide the value domain into representation levels and assign the instantaneous value of the signal to the nearest representation level. (quantization). Through digital-to-analog conversion (DAC), using the Nyquist-Shannon theorem and taking into account the quantization error, we are able to convert it back to an analog signal.



Rys. 1. Diagram of the DSP system

3.3. Localization technique

To locate the surface drone, we think it would be a good idea to use the signal sent by the plane at point A and point B with a strictly defined content on a given frequency. The signal echo generated by the drone, which will then be picked up by the aircraft, will enable time measurement. Knowing the location of the aircraft at points A and B and the time of sending the signal will allow us to find the point where the drone is located. An important aspect is also the separation of noise, which will certainly significantly affect the signals sent and received by the drone, but we will focus on solving this problem in the third part of the project.



Rys. 2. Concept visualization

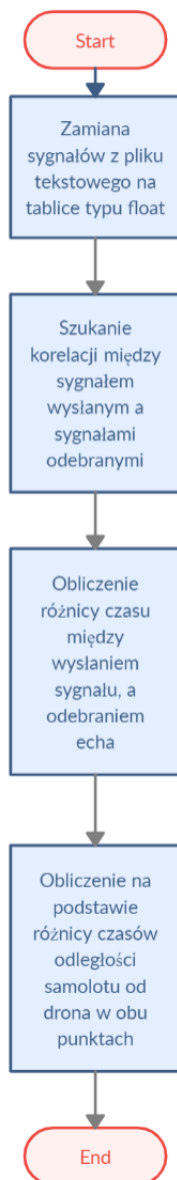
Assuming that our drone is not moving, we can use the trilateration method, which measures the time it takes for a radio signal to reach its position, and when the times from the receiving stations are known, the position of the receiver can be calculated.

3.4. Signal processing tools

We will use **Python** and MatLab programming languages to process the signal and obtain the parameters needed to calculate the location of the surface drone. In Python, we will use the `scipy.signal` module to process the signal, which contains required methods. Using Matlab, we will analyze the signal and simulate the DSP system in a much shorter time than the traditional C or C++ programming language will allow. The calculations of our DSP algorithm will be implemented on specialized FPGA integrated circuits. We will use the VHDL language and the Quartus environment to program such a system.

4. Concept

This chapter and subsections are devoted to presenting the concept and implementation of the reference model in the python environment.



Rys. 3. Solution concept block diagram

4.1. Solution Concept (Block Diagram Overview)

1. First, to be able to perform any operations, you need to load txt files with signals, and then convert them to sample tables.

2. Then, having signals in the form of tables of samples, you can search for the signal sent in the received signals. This can be done using the problem of signal correlation whose function has a maximum in the argument where the signals are best matched. To apply this issue in python, we use the Scipy library, in which this operation is implemented.

3. Having the signal samples that are most correlated with the sent signal, we can calculate the time difference between sending the signal and receiving it, knowing that the signal was recorded with a frequency of 10^6 samples/sec. This value allows us to calculate the distance between the drone and the aircraft.

4. The following formula calculates the distance between the aircraft and the drone:

$$d = c * t / 2$$

Where d - distance, c - speed of light in the air, t - time difference between sending a signal and receiving it. Then we draw two circles where $r = d$ and the intersections show us two points where the drone can be.

4.2. Cross-correlation

Cross-correlation is responsible for showing relationships between variables. In our case, it is needed to determine the similarity of two signals (sent and received). Its calculation works on the principle of time shifts, we shift one of the signals relative to the other, multiply the repeated samples and sum the results. We also use it to determine the delay of two signals relative to each other. To achieve this, we look for the maximum correlation value and subtract the length of the second signal minus 1 from its position. In our code, we use the "scipy.signal.correlate" function for correlation.

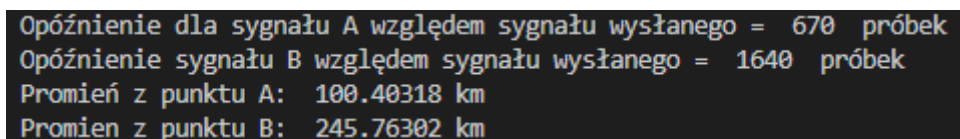
4.3. An example illustrating the operation

The best example that clearly illustrates the operation of the block diagram presented by us is our design task. We will use the signal sent and received from the plane in both locations "A" and "B".

The input in our program are signals in text form:

```
— Point A - received8.txt
— Point B - received8.txt
— sent8.txt
```

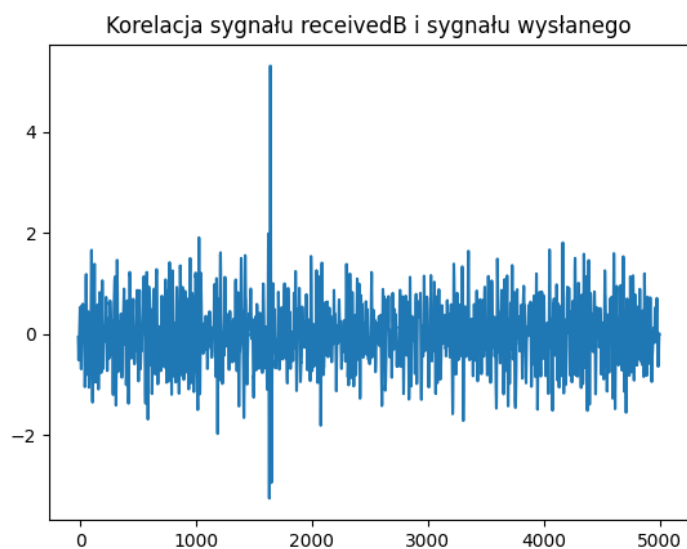
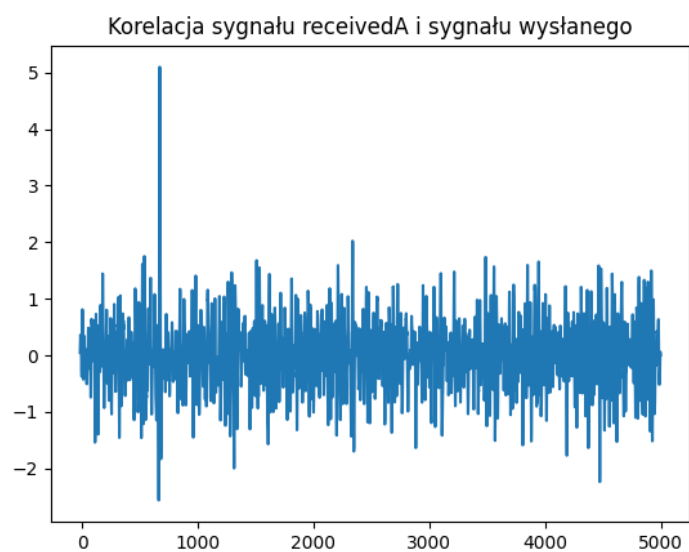
These signals are then processed by our program so that we get the following results:



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

Rys. 4. The results of our program on the provided .txt files

The first two results show the number of samples by which the expected signal was shifted. The next two results are already calculated radii.



The above two plots generated with the matplotlib library show the correlation of the signal sent with the signals received.

4.4. Program written in Python

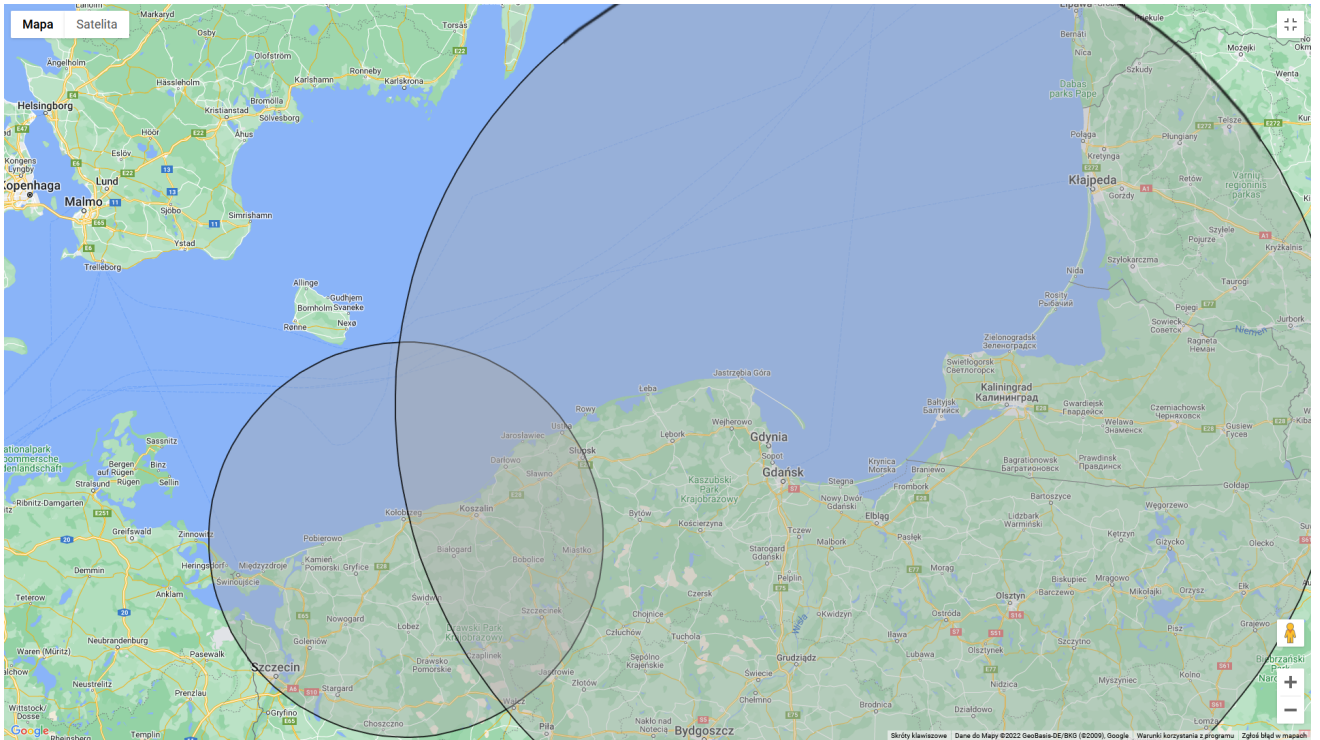
The program written by us in Python, along with comments to it, looks like this:


```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.signal import correlate
4
5 arrReceivedA = []
6 arrReceivedB = []
7 arrSent=[]
8
9 #Pobranie wartości dla próbek sygnału z pliku tekstowego i zapisanie ich w macierzy
10 with open("receivedA.txt",'r') as f:
11     for line in f.readlines():
12         line = line.replace('\n','')
13         line = float(line)
14         arrReceivedA.append(line)
15
16 with open("receivedB.txt",'r') as f:
17     for line in f.readlines():
18         line = line.replace('\n','')
19         line = float(line)
20         arrReceivedB.append(line)
21
22 with open("sent.txt",'r') as f:
23     for line in f.readlines():
24         line = line.replace('\n','')
25         line = float(line)
26         arrSent.append(line)
27
28 arrA = np.array(arrReceivedA)
29 arrB = np.array(arrReceivedB)
30 arrS = np.array(arrSent)
31
32 #Korelacja wzajemna dwóch sygnałów
33 correlateAS = correlate(arrA,arrS)
34 correlateBS = correlate(arrB,arrS)
35
36 #Pobranie indeksu maksymalnej wartości
37 prob1 = np.argmax(correlateAS)
38 prob2 = np.argmax(correlateBS)
39
40 #Okreslenie wartości opóźnienia
41 pointA = prob1 - (len(arrS)-1)
42 pointB = prob2 - (len(arrS)-1)
43
44 #Czas propagacji sygnału do drona wyrażona w sekundach
45 timeA = (pointA/1000000)/2
46 timeB = (pointB/1000000)/2
47
48 #Prędkość światła w powietrzu
49 v = 299711000
50
51 #Droga jaką pokonał sygnał
52 rariousA = v*timeA
53 rariousB = v*timeB
54
55 print("Promień z punktu A: ",round(rariousA/1000,5),"km")
56 print("Promień z punktu B: ",round(rariousB/1000,5),"km")
57
58 plt.figure(1)
59 print("Opóźnienie dla sygnału A względem sygnału wysłanego = ",pointA,"próbek")
60 plt.plot(np.arange(len(correlateAS))-(len(arrS)-1),correlateAS)
61 plt.title("Korelacja sygnału receivedA i sygnału wysłanego")
62
63 plt.figure(2)
64 print("Opóźnienie sygnału B względem sygnału wysłanego = ",pointB,"próbek")
65 plt.plot(np.arange(len(correlateBS))-(len(arrS)-1),correlateBS)
66 plt.title("Korelacja sygnału receivedB i sygnału wysłanego")
67 plt.show()

```

4.5. Determining the location of the drone



We determined the location by reading the coordinates of the intersection of two circles on the map. The circles have radii equal to the distances of the drone from the aircraft (i.e. approximately 100.4 km and 245.8 km), and their centers are the coordinates of the aircraft at points: A ($54^{\circ}04'02.2''\text{N}$ $15^{\circ}35'55.8''\text{E}$) and B ($54^{\circ}27'30.1''\text{N}$ $18^{\circ}46'25.0''\text{E}$). Using the knowledge about the drone, we know that it is a surface drone, so we considered a point on the water as its possible location. The location of the drone has the following coordinates: 55.001544 N , 15.551075 E . A possible error in the drone's coordinates may result from the method of determining them.

5. Implementation

5.1. Introduction

This chapter and its subchapters will describe the implementation of our system for finding the location of a surface drone.

5.2. HDL

In computer engineering, a hardware description language (HDL) is a specialized computer language used to describe the structure and behavior of electronic circuits, most commonly digital logic circuits.

The hardware description language looks very similar to a programming language like C; it is a textual description consisting of expressions, statements and control structures. One important difference between most programming languages and HDLs is that HDLs contain the concept of time.

HDLs are an integral part of electronic design automation (EDA) systems, especially for complex circuits such as application-specific integrated circuits, microprocessors, and programmable logic devices.

5.3. System in HDL

We familiarized ourselves with modeling the elements of the HDL system to design the system and describe its operation. Thanks to the HDL system, we also have the opportunity to test and simulate the operation of the system before it is implemented in the silicon structure.

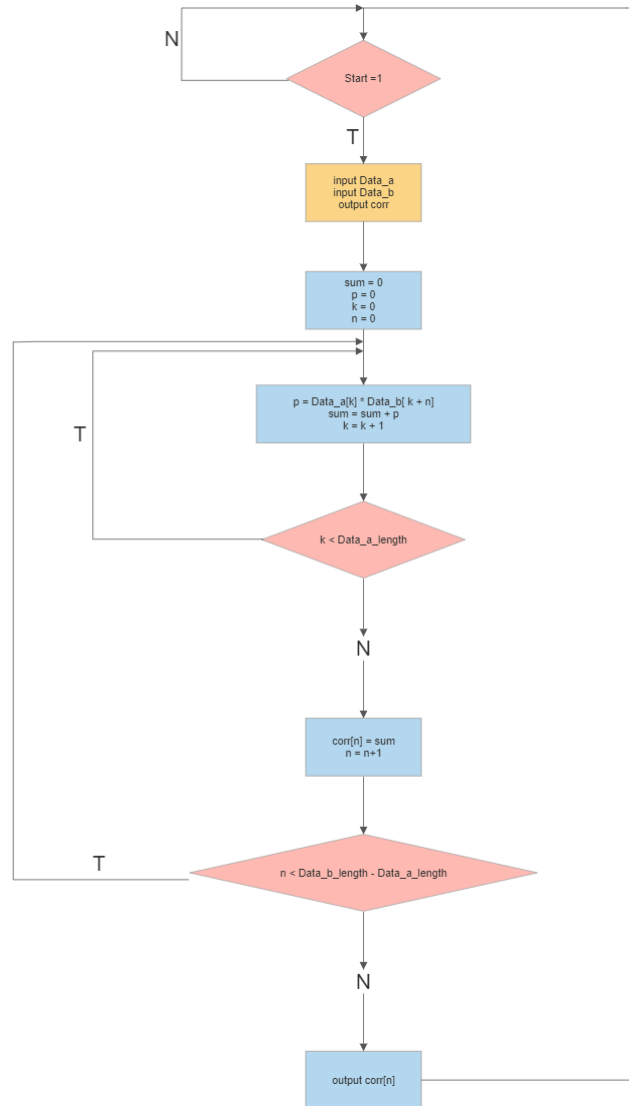
5.4. Concept

5.4.1. Implementation concept: Division

The problem we were supposed to solve, i.e. finding a surface drone, was divided into different ways of implementation. Below are the main ones.

- module for converting signal samples to bits - software implementation
- DSP module - hardware implementation
- localization module - program implementation

5.4.2. Implementation concept: Correlation Algorithm Diagram



5.4.3. Implementation concept: Design Technique

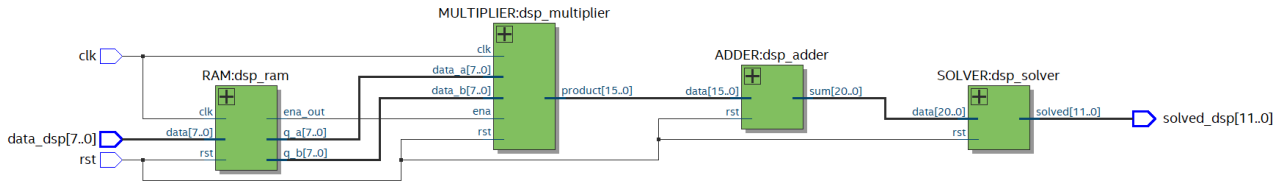
We designed the DSP module in verilog using the RTL design technique. RTL (Register Transfer Level) is a design technique where you design at the register level. It is a high level of abstraction, which simplifies the view of the designed system during the design process and speeds up the above process. The system takes the reference signal and the signal compared after conversions to NKB. It then calculates the correlation of these signals for a given time sample and transmits it to the comparison module. There, the calculated correlation is compared to the highest correlation so far.

5.5. Hardware implementation

5.5.1. Hardware implementation: Tools used

To design the hardware DSP module, we used the Verilog Hdl language, which is designed for such tasks. Synthesis and compilation were carried out in the Quartus Prime Lite environment, version 19.1. We simulated the module in ModelSim - INTEL FPGA STARTER EDITION.

5.5.2. Hardware implementation: RTL Diagram



Our DSP module is made up of 4 smaller modules. RAM memory, multiplier module, adder module and solution module. The connections between them are shown in the figure above.

5.5.3. Hardware implementation: DSP module

```

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. Hardware implementation: ROM

```
1 module RAM
2 begin
3 //Parametry ilości próbek w sygnałach
4 parameter signalA_samples = 20,
5 parameter corr_samples = 5000,
6 parameter signalB_samples = 5000
7 end
8
9 //Wejścia i wyjścia
10 input rst,
11 input clk,
12 input ena_in,
13 output ena_out,
14 output [7:0] q_a,
15 output [7:0] q_b
16
17 end;
18
19 initial begin
20 // Skrypt ładujący dane z pliku do pamięci ram
21 // synthesis translate_off
22 $readmemb("D:/REPO/SYF_PROJ/SignalSBin.txt",memory_a);
23 $readmemb("D:/REPO/SYF_PROJ/SignalABin.txt",memory_b);
24 // synthesis translate_on
25
26 end
27
28 //Obszary pamięci RAM oraz inne lokalne zmienne
29 reg [7:0] memory_a [0:signalA_samples];
30 reg [7:0] memory_b [0:signalB_samples];
31 reg [7:0] out_a;
32 reg [7:0] out_b;
33 integer i = 0;
34 integer c = 0;
35 reg rdy = 1'b0;
36
37
38
39
40 always@(posedge clk)
41 begin
42 //Stan resetu
43 if (rst) begin
44 i = 0;
45 c = 0;
46 ri = 0;
47 rc = 0;
48 rdy = 1'b0;
49
50 end else
51 begin
52 //Odczytywanie pamięci i podawanie dalej kolejnych wartości zgodnie z algorytmem korelacji
53 if (ena_in) begin
54
55 if (c <= (signalB_samples - signalA_samples)) begin
56
57 if(i < signalA_samples) begin
58 out_a = memory_a [i];
59 out_b = memory_b [i + c];
60
```

```

9      //wejścia i wyjścia
10     input rst,
11     input clk,
12     input ena_in,
13     output ena_out,
14     output [7:0] q_a,
15     output [7:0] q_b
16
17 );
18
19 initial begin
20     // Skrypt ładujący dane z pliku do pamięci ram
21     // synthesis translate_off
22     $readmemb("D:/REPO/SYF_PROJ/SignalSbin.txt",memory_a);
23     $readmemb("D:/REPO/SYF_PROJ/SignalABin.txt",memory_b);
24     // synthesis translate_on
25
26 end
27
28 //Obszary pamięci RAM oraz inne lokalne zmienne
29 reg [7:0] memory_a [0:signalA_samples];
30 reg [7:0] memory_b [0:signalB_samples];
31 reg [7:0] out_a;
32 reg [7:0] out_b;
33 integer i = 0;
34 integer c = 0;
35 reg rdy = 1'b0;
36
37 always@(posedge clk)
38 begin
39     //Stan resetu
40     if (rst) begin
41         i = 0;
42         c = 0;
43         rdy = 1'b0;
44     end else
45     begin
46         //Odczytywanie pamięci i podawanie dalej kolejnych wartości zgodnie z algorytmem korelacji
47         if (ena_in) begin
48             if (c <= (signalB_samples - signalA_samples)) begin
49                 if (i < signalA_samples) begin
50                     out_a = memory_a [i];
51                     out_b = memory_b [i + c];
52                     i = i + 1;
53                     rdy = 1;
54                 end else begin
55                     c = c + 1;
56                     i = 0;
57                     rdy = 0;
58                 end
59             end
60         end
61     end
62 end
63
64 assign ena_out = rdy;
65 assign q_a = out_a;
66 assign q_b = out_b;
67
68 endmodule

```

5.5.5. Hardware implementation: Multiplier module

```

1  module MULTIPLIER (
2
3      //Sygnały wejścia i wyjścia
4      input rst,
5      input [7:0] data_a,
6      input [7:0] data_b,
7      input clk,
8      input ena,
9      output [15:0] product
10
11 );
12
13 //Lokalne zmienne
14 reg [15:0] prod;
15 reg [2:0] gate_local;
16
17
18
19 always@(posedge clk)begin
20     //Stan resetu
21     if(rst) begin
22         prod = 0;
23     end else
24     //Mnożenie dwóch podanych wartości oraz przekazanie ich dalej
25     begin
26         if (ena) begin
27             prod = data_a * data_b;
28         end
29     end
30 end
31
32
33 assign product = prod;
34
35
36
37
38 endmodule

```

5.5.6. Hardware implementation: Adder module

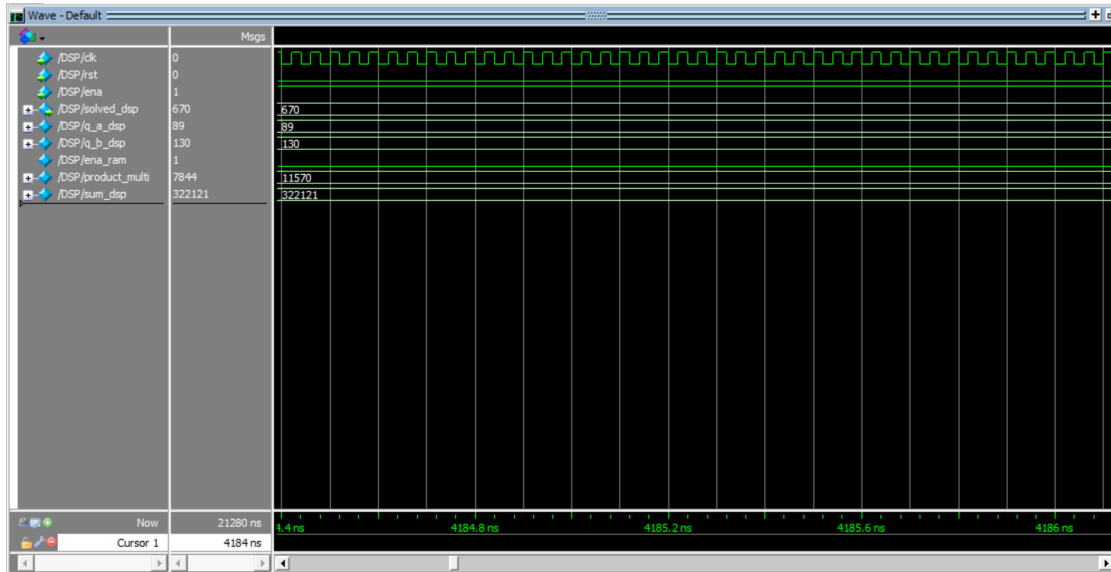
```
1 module MULTIPLIER (  
2  
3  
4 //Sygnały wejścia i wyjścia  
5 input rst,  
6 input [7:0] data_a,  
7 input [7:0] data_b,  
8 input clk,  
9 input ena,  
10 output [15:0] product  
11  
12 );  
13  
14 //Lokalne zmienne  
15 reg [15:0] prod;  
16 reg [2:0] gate_local;  
17  
18 always@(posedge clk)begin  
19 //Stan resetu  
20 if(rst) begin  
21 prod = 0;  
22 end else  
23 //Mnożenie dwóch podanych wartości oraz przekazanie ich dalej  
24 begin  
25 if (ena) begin  
26 prod = data_a * data_b;  
27 end  
28 end  
29  
30 end  
31  
32  
33  
34 assign product = prod;  
35  
36  
37  
38 endmodule
```

5.5.7. Hardware implementation: Solver module

```
1 module SOLVER  
2 #(  
3 //Parametr ilości argumentów funkcji wyjściowej korelacji  
4 parameter corr = 4980  
5 )(  
6  
7 //Sygnały wejścia i wyjścia  
8 input rst,  
9 input [20:0] data,  
10 output [11:0] solved  
11  
12 );  
13  
14 //Lokalne zmienne  
15 reg [11:0] solved_local;  
16 integer cnt;  
17 integer saved_cnt;  
18 reg [20:0] saved = 0;  
19  
20  
21  
22 always@(*) begin  
23 //Stan resetu  
24 if(rst)begin  
25 solved_local = 0;  
26 cnt = -1;  
27 saved_cnt = 0;  
28 saved = 0;  
29 //Znajdowanie największej wartości z podanych przez sumator sum poprzez porównywanie kolejnych wartości do  
30 siebie i zapamiętywanie największej z nich */  
31 end else begin  
32 if (cnt < corr) begin  
33 if (data > saved) begin  
34  
35 saved = data;  
36 saved_cnt = cnt;  
37  
38 end  
39 cnt = cnt + 1;  
40 end  
41 if (cnt == corr) begin  
42 solved_local = saved_cnt;  
43 end  
44 end  
45  
46 end  
47  
48 end  
49  
50  
51  
52 assign solved = solved_local;  
53  
54  
55 endmodule
```

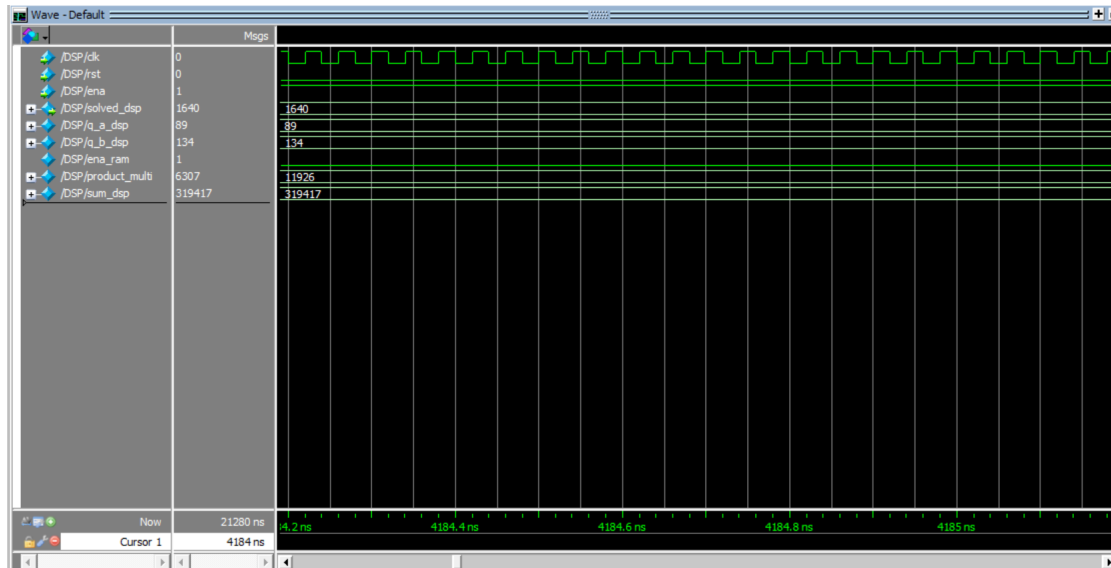

5.6. Simulation

5.6.1. Simulation: Signal Sent and Point A



In the above simulation, previously converted data files in the form of data bytes were used. The simulation result, in the form of delay in the form of samples, is consistent with the one obtained when creating the model in Python. The simulation lasted 4184ns.

5.6.2. Simulation: Signal Sent and Point B



As before, data from files in the same form was used. In this simulation, again, the latency result in the form of samples is consistent with that obtained when creating the Python model. The simulation lasted 4184ns.

6. Testing

6.1. Testbench

Basic use of the language HDL is a simulation where performance is tested system to verify correctness project. A simulation can be compared to a study physical system whose inputs connected to the stimulator (generator tests) and observe the outputs. HDL code simulation is execution a virtual experiment in which the physical the layout was replaced with the HDL description. In addition, HDL can be used to describe the test generator and collector response of the system and comparing them with pattern.

6.2. Code

```
1 // synthesis translate_off
2 timescale 1 ns/1 ps
3 define RTL_SIMULATION
4 module DSP_tb();
5
6 localparam T = 20;
7 localparam TD = 3;
8 localparam sTime = 1000 * T;
9
10
11
12
13 reg clk;
14 reg rst;
15 reg ena;
16 wire [11:0] output_rsp;
17
18 integer file_uns;
19
20
21
22 DSP UUT(
23     .ena (ena),
24     .clk (clk),
25     .rst (rst),
26     .solved_dsp (output_rsp)
27 );
28
29
30 initial begin : clock_generation
31     clk = 0;
32     #(T/2);
33     forever begin
34         #(T/2) clk = ~clk;
35     end
36
37 end
38
39
40
41 initial begin : reset
42     ena = 0;
43     rst = 1;
44     @(posedge clk);
45     @(negedge clk);
46     rst = 0;
```

```
47 end
48
49 reg [11:0] test_value = 670;
50 reg waiting = 0;
51 time first,next;
52 integer diff;
53 initial
54 begin: tb_generator
55     integer i;
56
57
58
59
60
61 wait (rst == 1);
62 wait (rst == 0);
63 $display("%t: Zerowanie zakonczzone\n", $time);
64 file_uns = $fopen("results_unsigned.txt", "w");
65
66 @(posedge clk);
67 ena = 1;
68 first = $time;
69
70 @(posedge clk);
71 while (!waiting) begin
72     @(posedge clk);
73     if (output_rsp == test_value) begin
74         waiting = 1;
75     end
76 end
77
78 next = $time;
79 diff = next - first;
80 $fwrite(file_uns, "%d\n", $unsigned(output_rsp));
81 $display("Test value = %b and output = %t:", test_value, output_rsp, $time);
82 $display("result = %d\n", output_rsp);
83 $display("Found after %d clk cycles", diff/T);
84 $stop;
85
86
87 end
88
89 endmodule
90 // synthesis translate_on
91
```

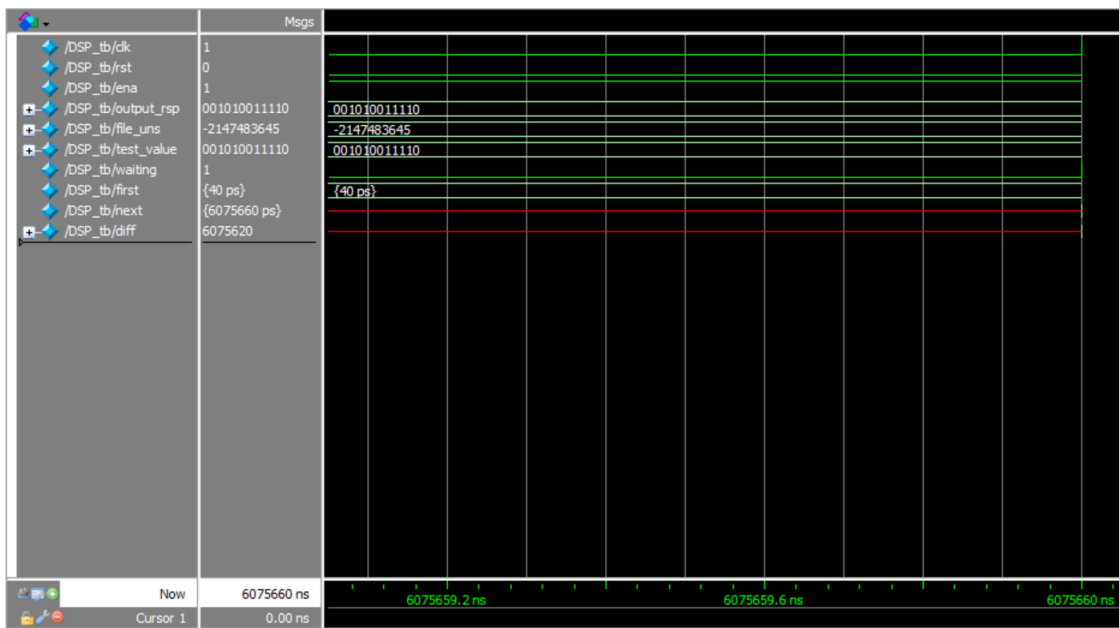
6.3. Our tests

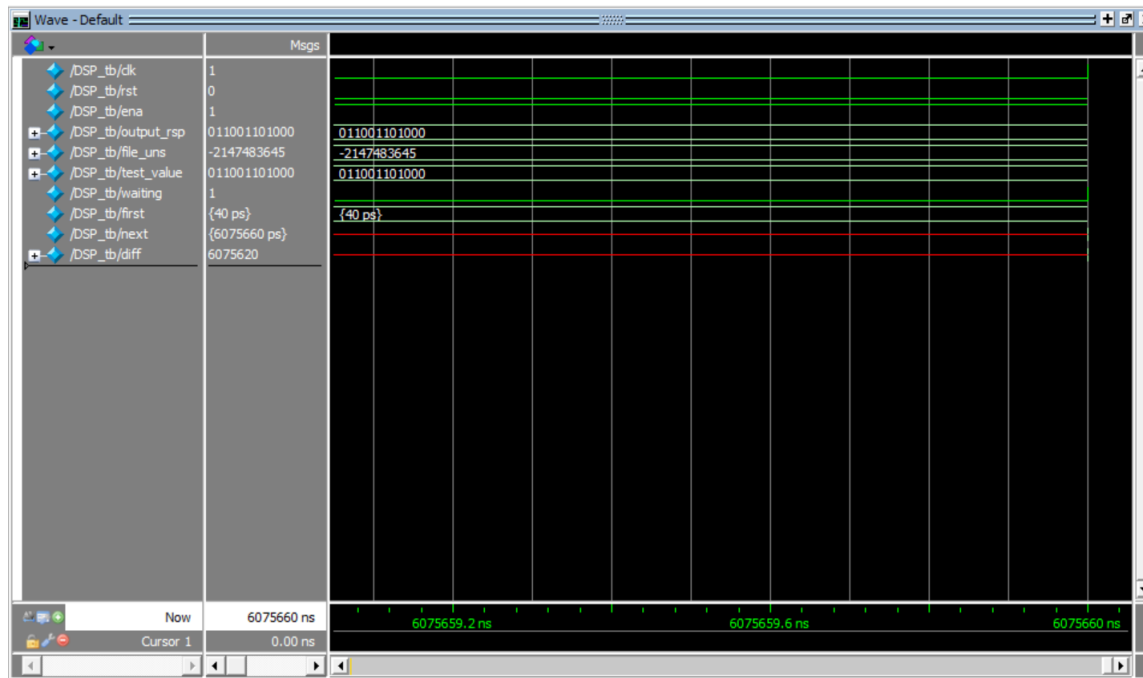
In our testbench we use wav samples which are integers. It initializes the DSP module and by stimulating it, in this case only with reset, ena and clock signals, we get a return result. As in the screenshots below. Finally, in the testbench we create a .txt file with the saved result. In addition, we managed to synchronize the operation of program blocks.

```
Transcript
#
# Test value = 001010011110 and output =          670000:          6075660
# result = 670
#
# Found after 303781 clk cycles
# ** Note: $stop : D:/REPO/SYF_PROJ/quartus/DSP_tb.v(84)
# Time: 6075660 ns Iteration: 1 Instance: /DSP_tb
# Break in NamedBeginStat tb_generator at D:/REPO/SYF_PROJ/quartus/DSP_tb.v line 84
V$IM 2>
```

```
Transcript
# Run -all
# Test value = 011001101000 and output =          1640000:          6075660
# result = 1640
#
# Found after 303781 clk cycles
# ** Note: $stop : D:/REPO/SYF_PROJ/quartus/DSP_tb.v(83)
# Time: 6075660 ns Iteration: 1 Instance: /DSP_tb
# Break in NamedBeginStat tb_generator at D:/REPO/SYF_PROJ/quartus/DSP_tb.v line 83
V$IM 2>
```

6.4. Modelsim





7. Summary

At this stage, the project assumptions implemented by us and the description of the project were discussed. The program is working better, but it would take more work to perfect it.

7.1. Achieved goals

- organization of work
- proposing multiple solutions
- creating a prototype of an action in Python
- implementation in Verilog
- code testing

7.2. Involvement of the team during the implementation of stages

The involvement of all team members in stages I-IV was 20

- Jakub Sulikowski - 40%
- Karol Kaproń - 15%
- Jakub Kubiński - 15%
- Magteusz Mięgoć-Kowalski - 15%
- Mateusz Dybicz - 15%

8. Literature

- <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-triangularcja-poznaj-metody-radiolokacji>
- <https://sound.eti.pg.gda.pl/~greg/dsp/04-Splot.html>

— <https://www.mapdevelopers.com/draw-circle-tool.php>