Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Е. В. Рогожников
Э. Дмитриев

ПЛИС В СИСТЕМАХ БЕСПРОВОДНОЙ СВЯЗИ

Методические указания для выполнения практических работ
и самостоятельной работы

Томск
2024

УДК 621.376.9
ББК 32.971.9
Р 598

Рецензенты:
Перин А.С., доцент кафедры сверхвысокочастотной и квантовой радиотехники ТУСУРа, кандидат технических наук, доцент;
Крюков Я.В., доцент кафедры телекоммуникаций и основ радиотехники ТУСУРа, кандидат технических наук

Р 598 ПЛИС в системах беспроводной связи: Методические указания для выполнения практических работ и самостоятельной работы / Е. В. Рогожников, Э. Дмитриев – Томск: Томск. гос. ун-т систем управления и радиоэлектроники, 2024. – 170 с.

Настоящие учебно-методическое пособие содержит указания по выполнению практических работ и самостоятельной работе. Основная цель данного практикума заключается в приобретении студентами компетенций в области цифровых методов формирования и обработки сигналов, а также в освоении навыков реализации алгоритмов на программируемых логических интегральных схемах (ПЛИС). Студенты будут ознакомлены с основными принципами функционирования ПЛИС, их программированием и применением в системах беспроводной связи. В пособии представлены как теоретические основы, необходимые для понимания работы алгоритмов формирования и обработки сигналов в системах беспроводной связи, так и практические задания, направленные на закрепление полученных знаний и развитие навыков реализации проектов на ПЛИС. Учебно-методическое пособие рассчитано на студентов технических специальностей, интересующихся областью беспроводной связи и цифровой обработки сигналов, а также на преподавателей, проводящих курсы по данной тематике. Пособие может быть использовано как в рамках учебного процесса в вузах, так и для самостоятельного изучения студентами.

Одобрено на заседании кафедры ТОР, протокол № 1 от 31 августа 2023 г.

УДК 621.376.9
ББК 32.971.9

Оглавление

ВВЕДЕНИЕ

Цель практикума заключается в укреплении и дальнейшем расширении теоретических компетенций студентов в области цифрового формирования и обработки сигналов, а также в приобретении навыков программирования ПЛИС и создания проектов на их основе, включая разработку устройств передачи и приема информации.

Практикум содержит описание следующих работ:

1. Генератор псевдослучайной последовательности на ПЛИС.
2. Формирование спектра OFDM символа. Реализация маппера на ПЛИС.
3. Квадратурный модулятор. Реализация QPSK модулятора.
4. Быстрое преобпазование Фурье. PLL блоки. Нормирование сигнала.
5. Добавление циклического префикса в системе с OFDM.
6. Инверсия спектра. Интерполяции. Реализация цифрового фильтра.
7. Цифровое преобразование частоты. Реализация цифрового смесителя.
8. Создание проекта приёмника OFDM сигнала.
9. Удаление постоянной составляющей. Реализация фильтра скользящего среднего.
10. Цифровое преобразование частоты, перенос спектра сигнала на нулевую частоту.
11. Цифровая фильтрация, удаление зеркальной копии сигнала.
12. Децимация сигнала на ПЛИС.
13. Инверсия спектра на ПЛИС.
14. Кадровая синхронизация. Реализация коррелятора.
15. Удаление циклического префикса в системе с OFDM.
16. Обратное быстрое преобразование Фурье.
17. Эквалайзирование. Оценка канала передачи на ПЛИС.
18. Квадратурная демодуляция. Реализация QPSK демодудятора.

Представленные практические и самостоятельные работы выполняются на базе ПЛИС фирмы Intel, а именно отладочной платы SoCKit Development Kit Cyclone V [1], а также среды разработки Quartus II [2]. Quartus II — это интегрированная среда разработки, предназначенная для проектирования и программирования ПЛИС и комплексных схем на основе ПЛИС, производимая компанией Intel. Эта среда обладает широким спектром функций и инструментов, предназначенных для проектирования и отладки цифровых систем на базе ПЛИС. Среда Quartus II обеспечивает широкие возможности для оптимизации и анализа проектов, включая автоматическую оптимизацию ресурсов, оценку времени и энергопотребления, а также возможности визуализации и отладки сигналов. Кроме того, она предоставляет доступ к библиотекам IP-блоков, что позволяет использовать готовые функциональные блоки для ускорения процесса разработки. Среда разработки Quartus II является одним из ведущих инструментов в области проектирования ПЛИС и широко используется инженерами по всему миру для создания разнообразных цифровых систем, включая системы связи, обработки сигналов, автоматизации и многие другие.

Работа № 1
«Генератор псевдослучайной последовательности»

Целью работы является реализация генератора псевдослучайной последовательности с помощью языка описания аппаратуры Verilog.

Задачи данной работы:

1) Создание проекта в системе автоматизации проектных работ Quartus Prime;

2) Реализация генератора псевдослучайной последовательности на языке описания аппаратуры Verilog;

3) Верификация созданного генератора в среде симуляции ModelSim.

1. Теоретическая часть

Генератор псевдослучайной последовательностей (ПСП) – алгоритм, генерирующий ряд чисел. Каждое число этой последовательности почти не имеет зависимость от других чисел. Полученная последовательность подчиняется определенному распределению (чаще всего равномерному). Генератор ПСП может быть организован на регистре сдвига с линейной обратной связью.

Регистр сдвига с линейной обратной связью (РСЛОС) – регистр сдвига битовых слов, входной бит которого является линейной функцией состояния остальных битов регистра до сдвига.

В РСЛОС выделяют две части: регистр сдвига и функции, вычисляющая значения вдвигаемого бита (обратная связь). Сдвиговый регистр представляет собой последовательность триггеров. Количество триггеров определяется длиной сдвигового регистра. Если длина равна *L* битам, то регистр называется *L*-битовым сдвиговым регистром. Обратная связь представляет собой XOR (сложение по модулю два) некоторых триггеров, перечень этих триггеров называется отводной последовательностью. Схема РСЛОС представлена на рисунке 1.1.

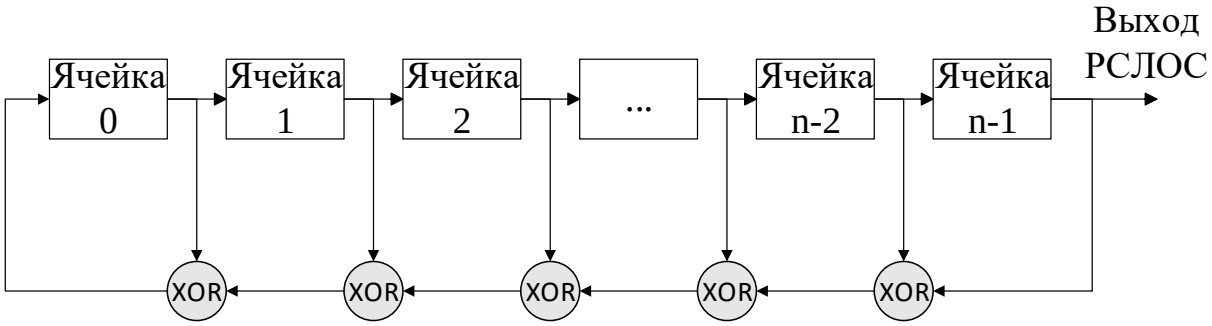


Рисунок 1.1 – Схема РСЛОС

Длина последовательности до начала повторения называется периодом регистра. Максимальный период, для РСЛОС длинной *L*, вычисляется по формуле:

$$T = 2^L - 1. \tag{1.1}$$

Чаще всего последовательность представляется в виде многочлена вида:

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0 x^0 = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0, \tag{1.2}$$

где $a_i = \{0,1\}$ для $i = 1...n$, а x^i – указывает разряд.

Степень многочлена равна длине регистра.

Только при определенных отводных последовательностях РСЛОС пройдет через все $2^L - 1$ внутренних состояний, такой сдвиговый регистр называется РСЛОС с максимальным периодом. Получившийся результат называется М-последовательностью. Многочлен данной последовательности называется примитивным.

Примитивный многочлен p(x) – это неприводимый многочлен степени m, который является примитивным, если n – наименьшее положительное число такое, что p(x) делит x^n+1 и $n=2^m-1$.

Нахождение примитивного многочлена достаточно сложная математическая задачу. Поэтому наиболее простым способом выбора РСЛОС является использование готовых таблиц.

Например, для регистра длинной 32, примитивный многочлен выглядит следующим образом:

$$x^{32}+x^7+x^5+x^3+x^2+x+1$$

Кратко данный многочлен можно записать в виде последовательности степеней: (32, 7, 5, 3, 2, 1, 0).

Существует два вида конфигураций РСЛОС: конфигурация Фибоначчи, представленная на риснуке 1.2, и конфигурация Галуа, представленная на рисунке 1.3.

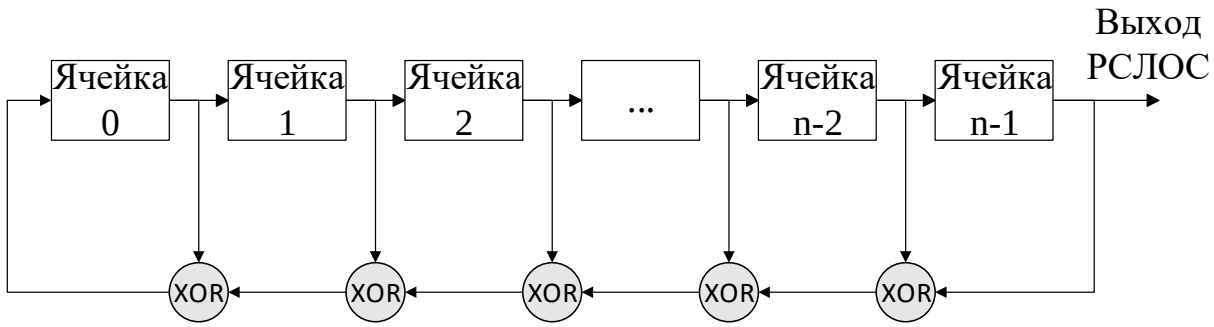


Рисунок 1.2 – РСЛОС конфигурации Фибоначчи

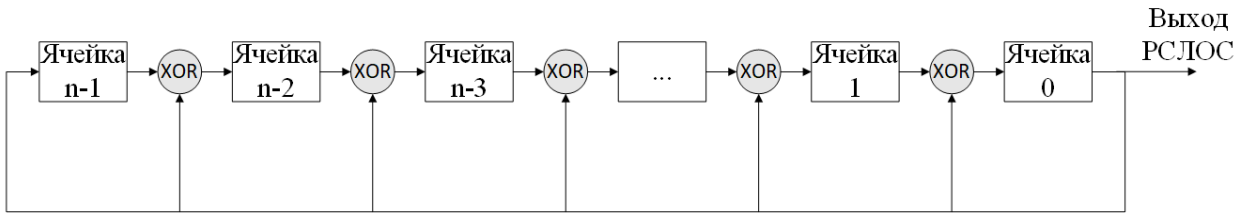


Рисунок 1.3 – РСЛОС конфигурации Галуа

Главное различие заключается в следующем: в конфигурации Фибоначчи обратная связь является функцией от всех ячеек в регистре и результат помещается в первую ячейку, в конфигурации Галуа обратная связь является функцией только от первой ячейки и результат может применятся к каждой ячейке регистра.

2. Практическая часть

2.1 Создание модуля верхнего уровня

Модуль верхнего уровня (top level) предназначен для описания связей между модулями нижнего уровня и для подключения их к портам верхнего уровня, то есть к физическим входам и выходам.

Для создания модуля верхнего уровня:

2.1.1. Создайте папку на рабочем столе. В этой папке будут храниться все файлы проекта. Имя папки не должно содержать русских букв.

2.1.2. Далее необходимо открыть среду проектирования Quartus Prime и создать новый проект, для этого нажмите кнопку «New Project Wizard», выделенную на рисунке 2.1. Далее в открывшемся окне «Introductio», приведенном на рисунке 2.2, нажмите кнопку «Next».

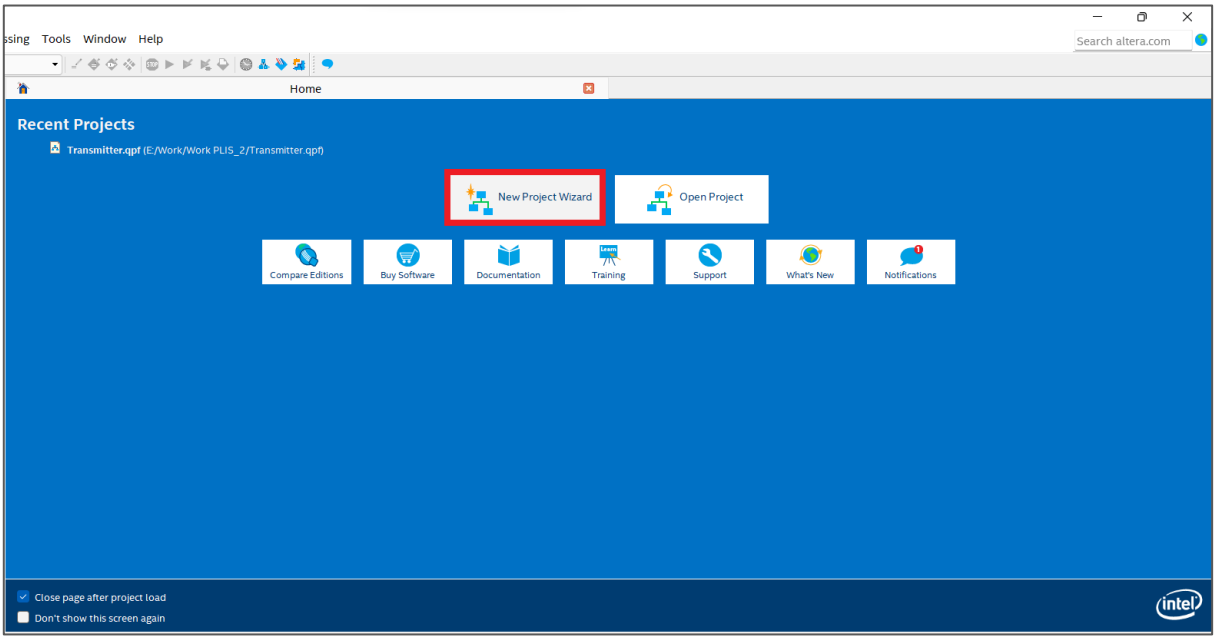


Рисунок 2.1 – Окно программы Quartus Prime

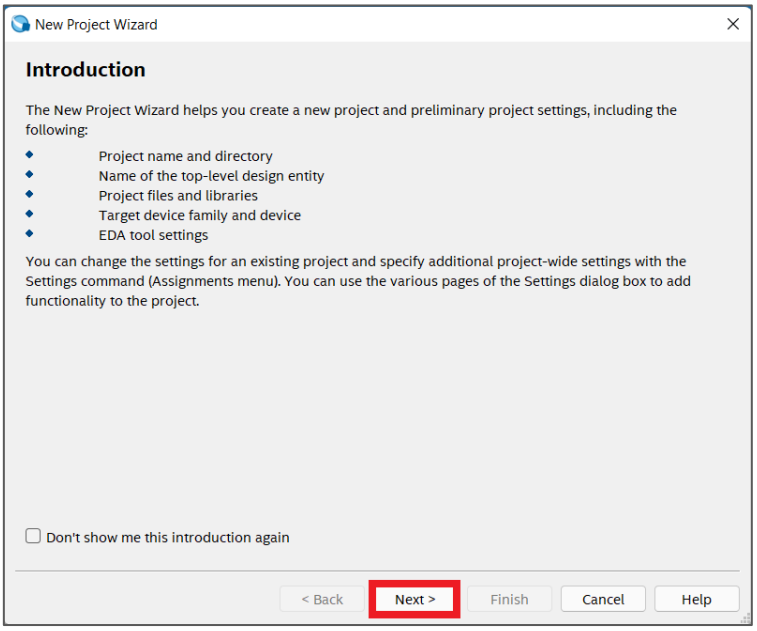


Рисунок 2.2 – Создание нового проекта. Шаг 1

2.1.3. В качестве папки, в которой будет хранится проект, укажите в адресной строке («What is the working directory for this project?»), выделенной на рисунке 2.3, ранее созданную папку. В строке наименования проекта («What is the name of this project»), выделенной на рисунке 2.3, введите название проекта (в данном случае введите «Transmitter»). После чего нажмите кнопку «Next».

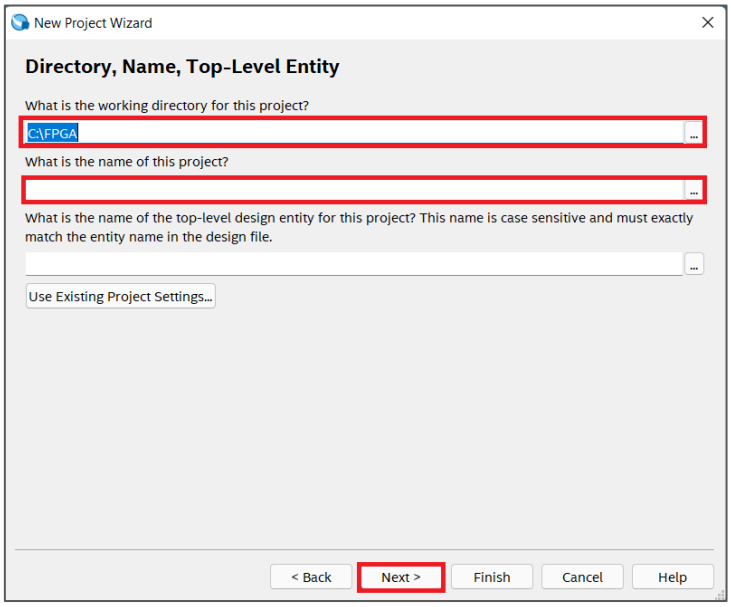


Рисунок 2.3 – Создание нового проекта. Шаг 2

2.1.4. В открывшемся окне «Project Type», приведенном на рисунке 2.4, необходимо поставить маркер напротив «Empty project» (пустой проект) и нажать кнопку «Next».

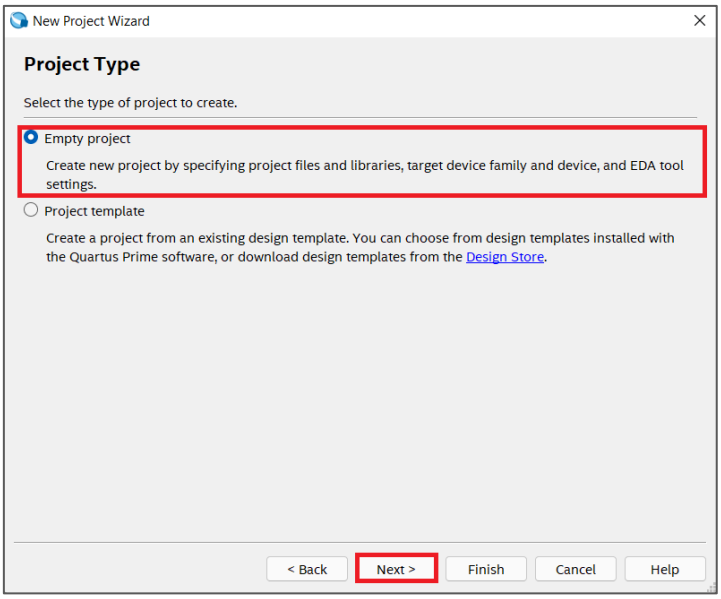


Рисунок 2.4 – Создание нового проекта. Шаг 3

2.1.5. Далее в окне «Add Files», приведенном на рисунке 2.4 просто нажмите кнопку «Next».

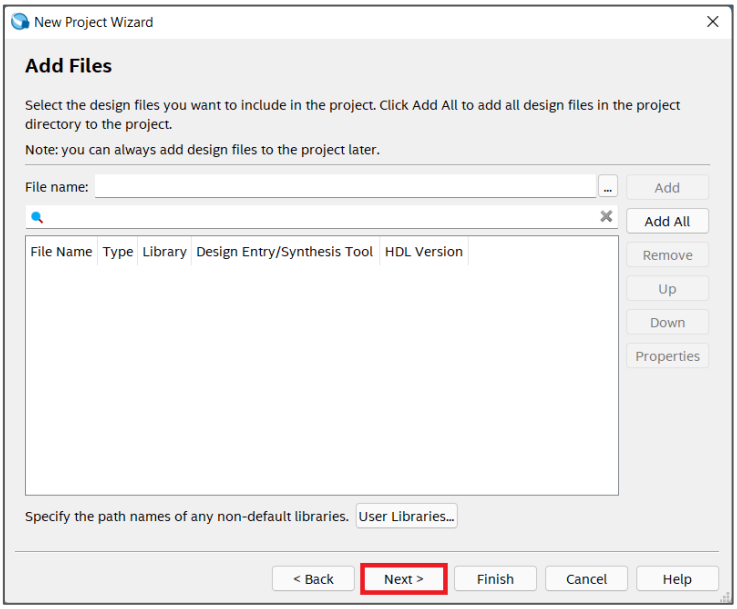


Рисунок 2.5 – Создание нового проекта. Шаг 4

2.1.6. В окне «Family, Device & Board Settings», приведенном на рисунке 2.6, в выплывающем списке «Family» области «Device family» выберете семейство «Cyclone V», в области «Available devices» выберете тип микросхемы «5CSXFC6D6F31C8», после чего нажмите кнопку «Next».

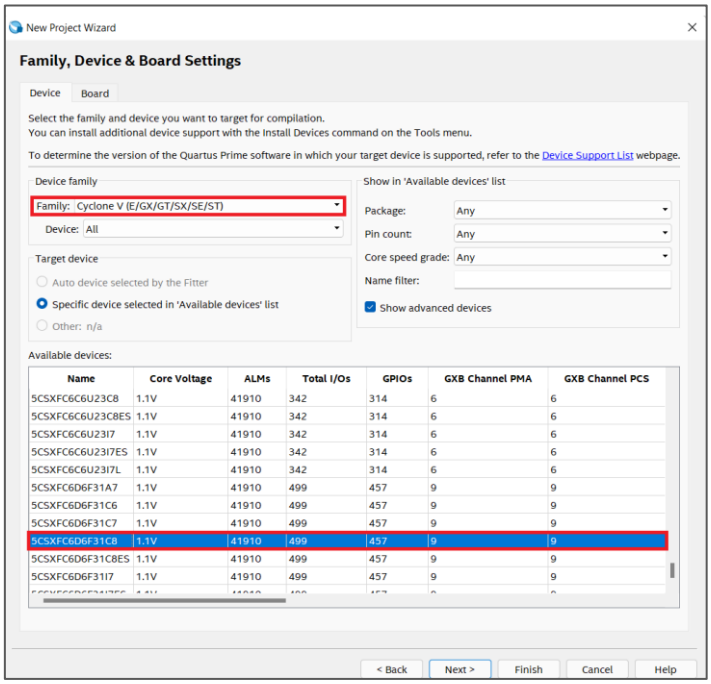


Рисунок 2.6 – Создание нового проекта. Шаг 5

2.1.7. В открывшемся окне «EDA Tool Settings», приведенном на рисунке 2.7 просто нажмите кнопку «Next». Для завершения создания проекта в окне «Summary», приведенном на рисунке 2.8, нажмите кнопку «Finish».

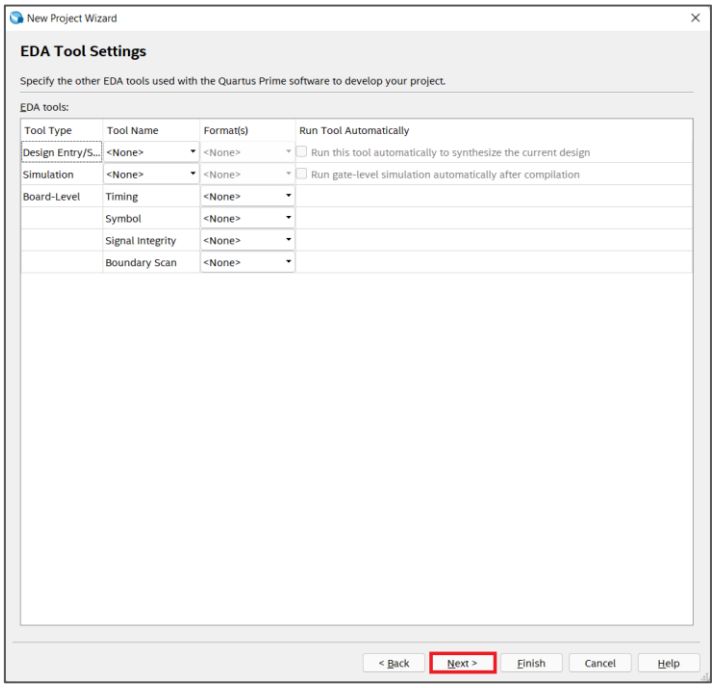


Рисунок 2.7 – Создание нового проекта. Шаг 6

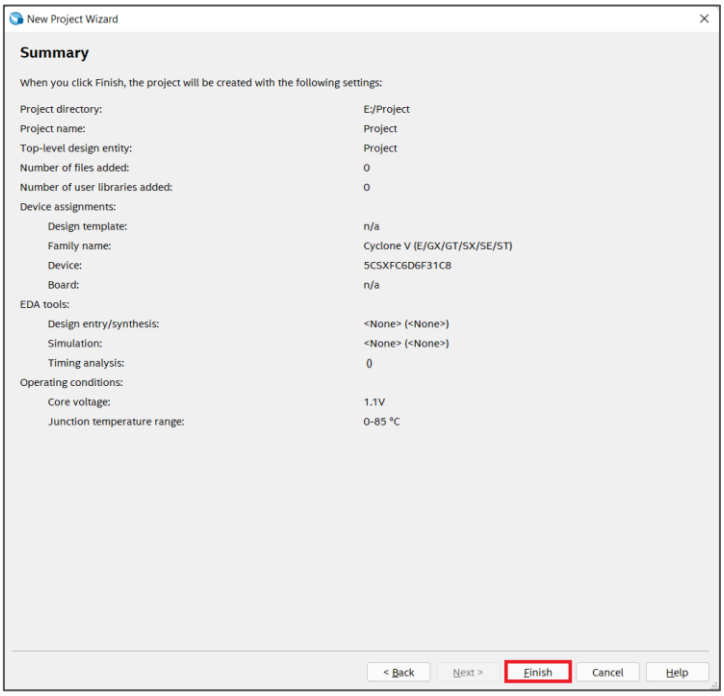


Рисунок 2.8 – Создание нового проекта. Шаг 7

2.1.8. Для добавления в проект файла Verilog HDL нажмите на значок файла в левом верхнем углу окна «Quartus Prime» в соответствии с рисунком 2.9.

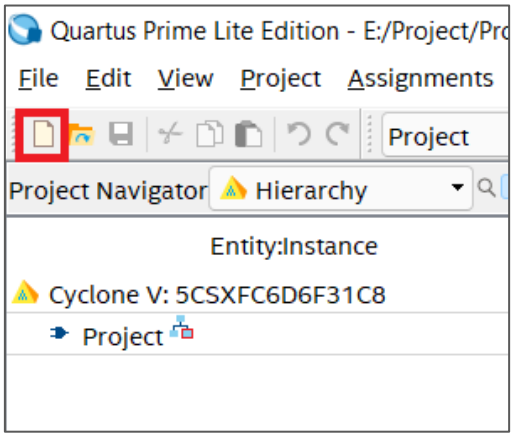


Рисунок 2.9 – Добавление файла Verilog HDL. Шаг 1

2.1.9. В появившемся окне, приведенном на рисунке 2.10, выберите файл «Verilog HDL File», нажав на него левой кнопкой мыши. После чего нажмите кнопку «OK».

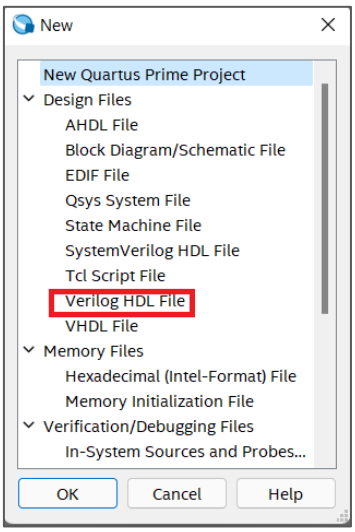


Рисунок 2.10 – Добавление файла Verilog HDL. Шаг 2

2.1.10. В появившемся текстовом редакторе, необходимо описать модуль верхнего уровня на языке SystemVerilog. Описание логики любого модуля на языке SystemVerilog начинается с ключевого слова «module» и последующим его наименованием.

ВНИМАНИЕ

Наименование модуля верхнего уровня и имя проекта всегда должны совпадать!

Модули имеют порты (входы и выходы), которые описываются типом wire. Перечисление портов производится через запятую:

```
module Transmitter(iclk,irst,ireq,ready_in,stop_in,odata_1,odata_2);
```

После чего описываются направление сигналов (input, output):

```
module Transmitter (in_clk, irst, ireq, ready_in,stop_in, odata_1, odata_2);
input in_clk;          // Тактовый сигнал
input irst;                  // Сигнал сброса
input   ireq;                // Сигнал запроса генерации бит
input ready_in;              // Сигнал начала работы
input stop_in;         // Сигнал остановки работы
```

```
output odata_1;          // Шина выходной последовательности 1
output odata_2;          // Шина выходной последовательности 1
```

Заканчиваться описание логики каждого модуля всегда должно ключевым словом «endmodule». Сохраните файл, для этого нажмите File=>Save as, как на рисунке 2.11.

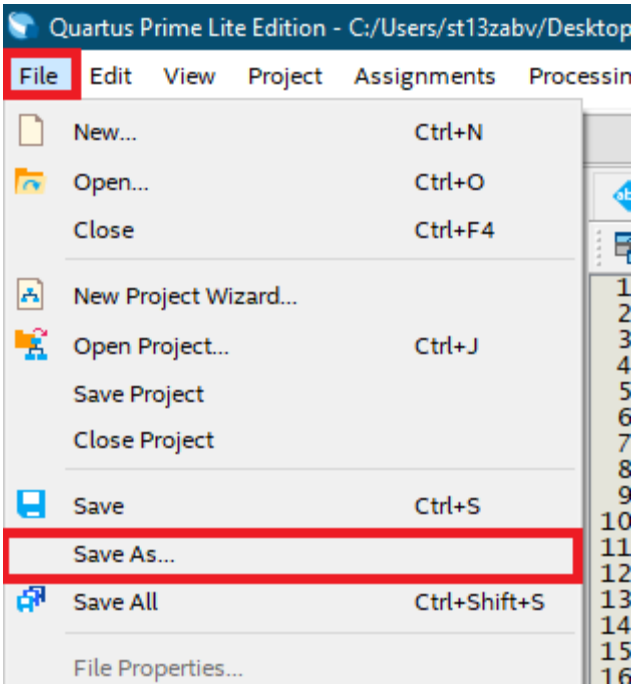


Рисунок 2.11 – Сохранение файла. Шаг 1

Откроется окно сохранения файла. Создайте папку v_file (ПКМ=> Создать=> Папку). Сохраните файл в эту папку (название файла должно совпадать с именем модуля). Все последующие модули сохраняйте в эту папку. (рисунок 2.12).

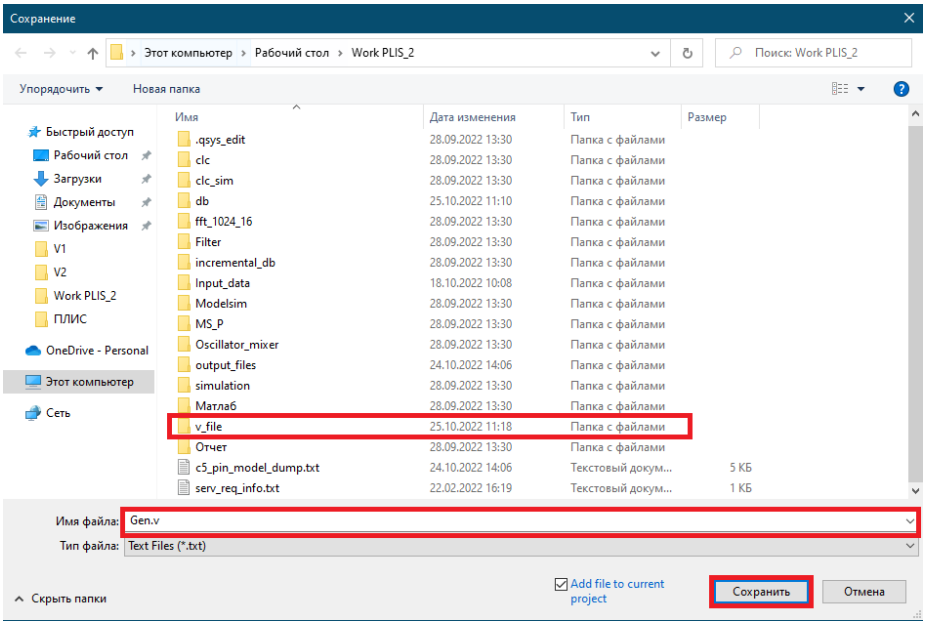


Рисунок 2.12 – Сохранение файла. Шаг 2

2.2 Генератор ПСП

Генератор ПСП будет реализован на РСЛОС конфигурации Фибоначчи с следующими параметрами:

– длина регистра: 27;

– номера отводов: 26, 4, 1, 0.

Графическое представление генератора ПСП представлено на рисунке 2.13.

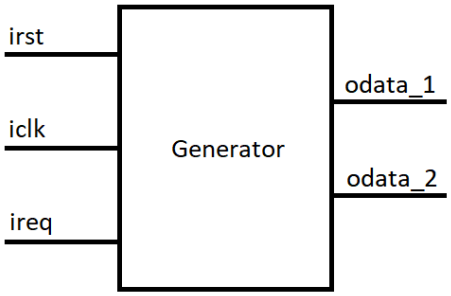


Рисунок 2.13 – Графическое представление генератора ПСП

Описание портов генератора ПСП приведено в таблице 2.1.

Таблица 2.1 – Описание портов генератора ПСП

Порты	Разрядность, бит	Тип	I/0	Описание портов
iclk	1	wire	input	Тактирование 10 МГц
irst	1	wire	input	Сигнал сброса
ireq	1	wire	input	Сигнал разрешения
odata_1	1	wire	output	Первый выходной бит
odata_2	1	wire	output	Второй выходной бит

Для создания модуля генератора ПСП:

2.2.1. Добавьте в проект файл Verilog HDL в соответствии с п. 2.1.8 – 2.1.9.

2.2.2. В появившемся текстовом редакторе опишите модуль генератора ПСП, присвоив ему имя «Gen». Описание логики модулей приведено в п. 2.1.10.

```
module Gen (iclk, irst, ireq, odata_1, odata_2)
```

2.2.3. Объявите порты модуля «Gen» в соответствии с рисунком 2.11 и таблицей 2.1. После чего опишите направление сигналов объявленных портов:

```
input iclk;
input irst;
input ireq;
output odata_1;
output odata_2;
```

2.2.4. Объявите два сдвиговых регистра разрядностью 27 бит и две переменные типа wire для реализации функции обратной связи:

```
reg [26:0]   shift_reg_1;
reg [26:0]   shift_reg_2;
wire         next_bit_1;
wire         next_bit_2;
```

2.2.5. Далее сдвиговым регистрам необходимо присвоить начальные состояния, иначе ячейки сдвигового регистра будут заполнены нулями и при осуществлении сдвига данных на выход сдвигового регистра всегда будет поступать ноль:

```
initial shift_reg_1 <= 27'b111111111111111111111111111;
initial shift_reg_2 <= 27'b001110111111111101111111101;
```

2.2.6. Вычислите новые значения обратной связи с помощью логической операции исключающего ИЛИ (на языке SystemVerilog данная логическая операция обозначается: «^») и с помощью оператора присвоения assign присвойте новые значения их переменным:

```
assign next_bit_1=shift_reg_1[26]^
                  shift_reg_1[ 4]^
                  shift_reg_1[ 1]^
                  shift_reg_1[ 0];

assign next_bit_2=shift_reg_2[26]^
                  shift_reg_2[ 4]^
                  shift_reg_2[ 1]^
                  shift_reg_2[ 0];
```

2.2.7. Так же с помощью оператора присвоения assign присвойте последний разряд сдвигового регистра выходным портам модуля «Gen»:

```
assign odata_1=shift_reg_1[26];
assign odata_2=shift_reg_2[26];
```

2.2.8. Далее с помощью always блока реализуйте сдвиговый регистр с асинхронным сбросом, который будет работать по разрешающему сигналу ireq. Реализацию можно осуществить несколькими способами:

– с помощью цикла for. Необходимо дополнительно объявить переменную i типа integer. Тогда код будет выглядеть следующим образом:

```
always@(posedge iclk or posedge irst)
begin
  if(irst)
      begin
          shift_reg_1 <= 27'b111111111111111111111111111;
          shift_reg_2 <= 27'b001110111111111101111111101;
      end
      else if(ireq)
          begin
          for(i=26;i>=1;i=i-1)
                  begin
                  shift_reg_1[i]<=shift_reg_1[i-1];
                  shift_reg_2[i]<=shift_reg_2[i-1];
                  end
                  shift_reg_1[0]<=next_bit_1;
                  shift_reg_2[0]<=next_bit_2;
          end
      end
```

– прописать цепочку переприсвоения сдвиговых регистров самостоятельно:

```
always@(posedge iclk or posedge irst)
begin
  if(irst)
      begin
          shift_reg_1<= 27'b111111111111111111111111111;
          shift_reg_2 <= 27'b001110111111111101111111101;
      end
      else if(ireq)
          begin
                  shift_reg_1[26]<=shift_reg_1[25];
                  shift_reg_2[26]<=shift_reg_2[25];
                  shift_reg_1[25]<=shift_reg_1[24];
                  shift_reg_2[25]<=shift_reg_2[24];
                  ……..
                  shift_reg_1[1]<=shift_reg_1[0];
                  shift_reg_2[1]<=shift_reg_2[0];
                  shift_reg_1[0]<=next_bit_1;
                  shift_reg_2[0]<=next_bit_2;
          end
      end
```

Не забудьте добавить в конец endmodule и сохранить файл.

2.2.9. После чего необходимо подключить созданный модуль генератора ПСП к проекту, в модуле верхнего уровня. Обобщенная структура подключения выглядит следующим образом:

```
ИМЯ_ПОДКЛЮЧАЕМОГО_МОДУЛЯ    ИМЯ_ВКЛЮЧЕНИЯ(
          .внутренний_порт_1  (внешняя_переменная_1),
          .внутренний_порт _2 (внешняя_переменная _2),
```

```
……..
        .внутренний_порт _n (внешняя_ переменная _n)
);
```

2.2.10. Шины прописываются после входных портов. Обобщённая конструкция выглядит следующим образом:

```
wire [РАЗРЯДНОСТЬ-1:0] ИМЯ_ПРОВОДА;
```

Например, двух разрядная шина Example будет задана следующим образом:

```
wire [1:0] Example;
```

ВНИМАНИЕ

Внутренний порт подключаемого модуля и шина, к которой подключается модуль должны иметь одинаковую разрядность!

Код подключения генератора ПСП в модуле верхнего уровня выглядит так:

```
Gen generator(
        .iclk       (in_clk),
        .irst       (irst),
        .odata_1    (odata_1),
        .odata_2    (odata_2),
        .ireq       (ireq)
);
```

2.2.11. Сохраните все изменения и нажмите кнопку синтеза (рисунок 2.14).

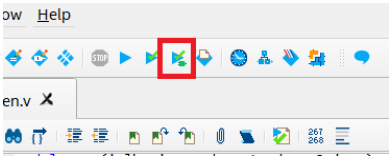


Рисунок 2.14 – Кнопка запуска синтеза

2.2.12. После успешного синтеза, для проверки работоспособности генератора, необходимо произвести симуляцию в ModelSim. Перенесите файл «testbench.v» в ранее созданную папку «v_file»s (данный файл выдается преподавателем к данной работе). Откройте его с помощью блокнота или любого другого текстового редактора.

2.2.13. Описание модуля testbench всегда начинается с объявления временной шкалы, определяющей длительность временных отсчетов:

```
`timescale 1 ps/1 ps
```

2.2.14. Далее пропишите наименование модуля. После чего объявляются внутренние переменные модуля. Объявите четыре регистра и две шины:

```
module testbench;
reg iclk;
reg ireq;
reg ready_in;
```

```
    reg stop_in;
    reg irst;
    wire odata_1;
    wire odata_2;
```

2.2.15. Далее следует сгенерировать сигнал тактовой частоты:

```
    initial iclk=0;
    initial forever iclk=#10 ~iclk;
```

С помощью оператора initial изначально сигналу iclk присваивается начальное состояние равное нулю, далее значение iclk будет меняться на противоположное каждые 10 временных отсчетов имитируя тактовый сигнал. Из п. 2.2.11 видно, что в нашем случае один отсчет равен одной пикосекунде.

2.2.16. Далее с помощью оператора initial и оператора begin……end (если осуществляется более одного присвоения) задайте нулевые начальные состояния входных сигналов:

```
initial
begin
      ireq=0;
      irst=1;
      #20
      irst=0;
     #20
     ireq=1;
     #2000
     ireq=0;
end
```

2.2.17. Далее пропишите подключение пинов модуля верхнего уровня с шинами и регистрами, прописанными в пункте 2.2.3.

```
Transmitter Transmitter(
                .irst(irst),
                .in_clk(iclk),
                ireq(ireq),
                .ready_in(ready_in),
                .stop_in(stop_in),
                .odata_1(odata_1),
                .odata_2(odata_2)
);
```

Сохраните все изменения.

2.2.18. Откройте программу ModelSim. Для этого в поиске в меню Пуск введите: ModelSim. Иконка программы выглядит как на рисунке 2.15.

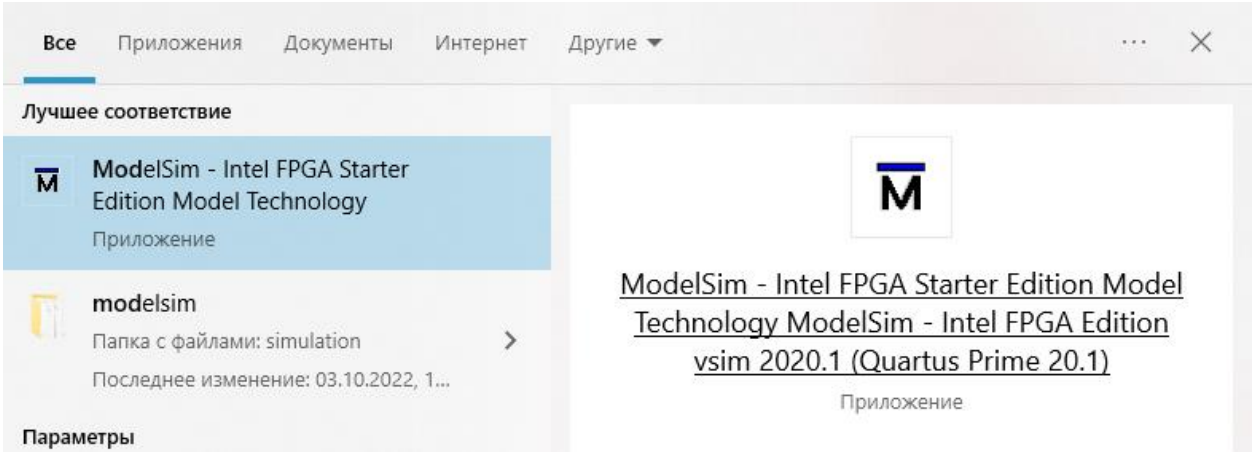


Рисунок 2.15 – Иконка ModelSim

2.2.19. Создайте новый проект, открыв вкладку «Fail» → «New» → «Project», как показано на рисунке 2.16.

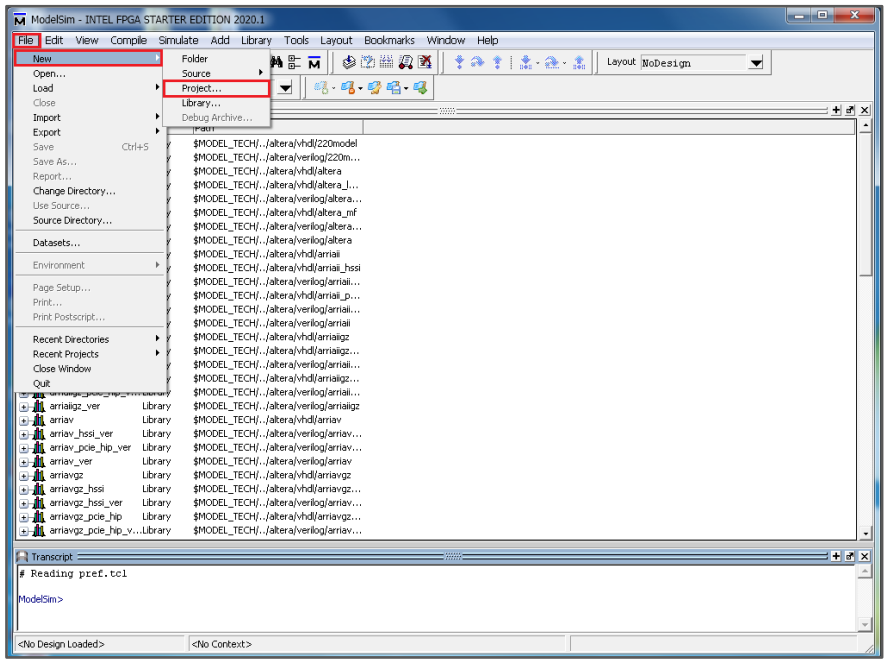


Рисунок 2.16 – Создание нового проекта в среде симуляции ModelSim. Шаг 1

2.2.20. Далее в строке наименования проекта («Project Name»), выделенной на рисунке 2.17, введите название проекта «work». В качестве месторасположения проекта, в адресной строке («Project Location»), выделенной на рисунке 2.17, выберете папку Modelsim, которая расположена в папке основного проекта на рабочем столе (если папка отсутствует, необходимо создать её). После чего нажмите кнопку «ОК».

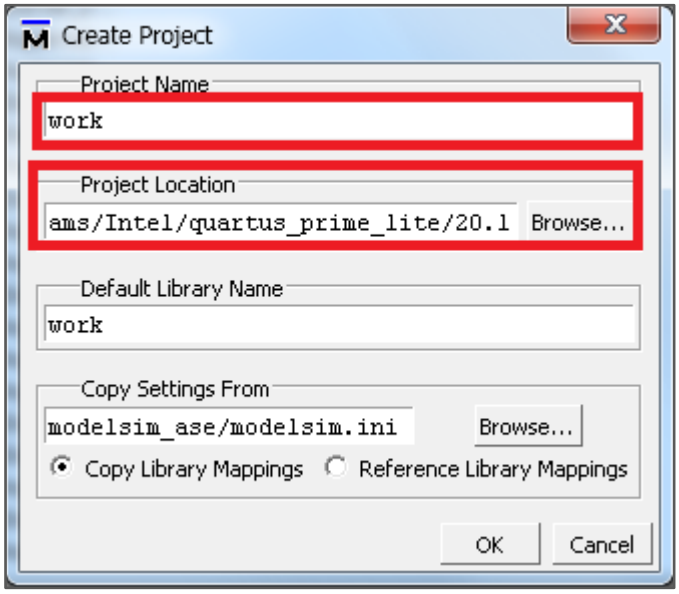


Рисунок 2.17 – Создание нового проекта в среде симуляции ModelSim. Шаг 2

2.2.21. В открывшемся окне, приведенном на рисунке 2.18, для добавления сохраненных файлов нажмите на значок «Add Existing File».

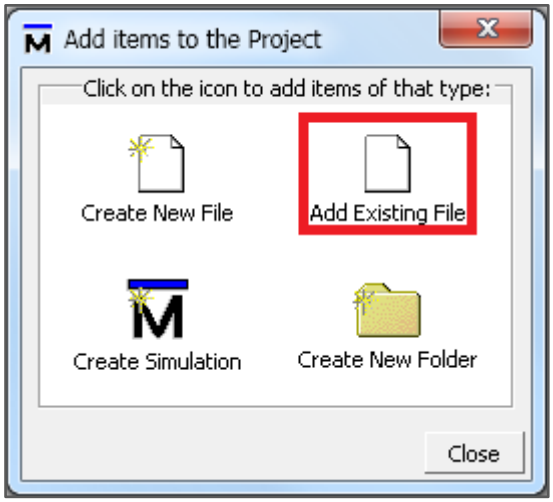


Рисунок 2.18 – Добавление файлов в среде симуляции ModelSim. Шаг 1

2.2.22. Далее в окне, приведенном на рисунке 2.19, нажмите кнопку «Browse» и добавьте файлы, созданные ранее: модуль верхнего уровня, модуль генератора ПСП и модуль testbench.

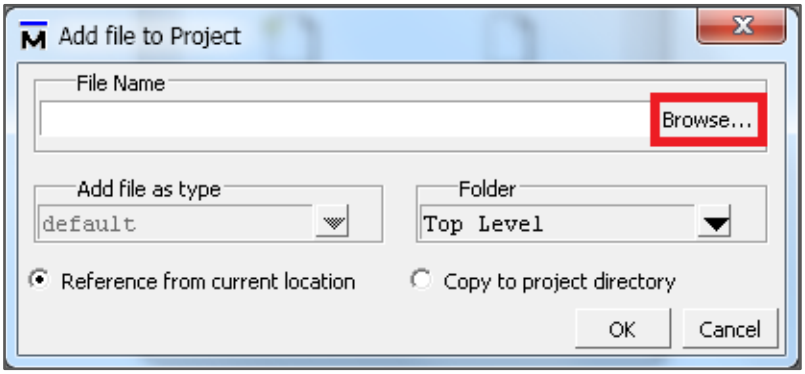


Рисунок 2.19 – Добавление файлов в среде симуляции ModelSim. Шаг 2

2.2.23. Скомпилируйте добавленный файлы, для этого откройте вкладку «Compile» → «Compile all», в соответствии с рисунком 2.20.

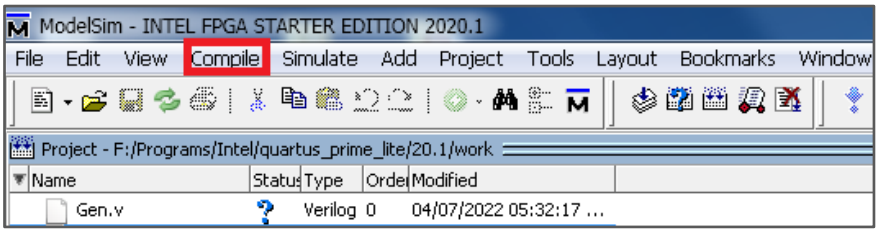


Рисунок 2.20 – Компиляция файлов в среде симуляции ModelSim

Если компиляция пройдет без ошибок напротив файлов у вас появятся зеленые галочки.

2.2.24. Далее перейдите в вкладку «Simulate» → «Start Simulate».

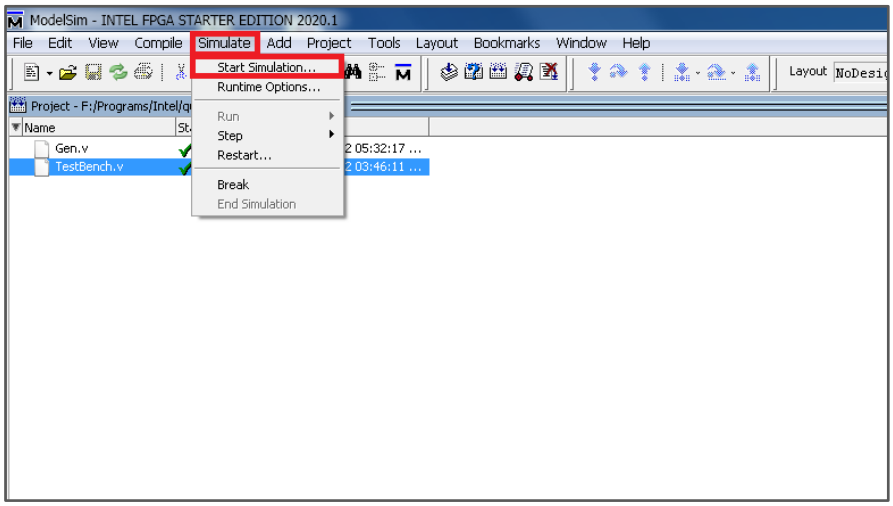


Рисунок 2.21 – Запуск симуляции в среде симуляции ModelSim

2.2.25. В открывшемся окне выберете модуль «Tb» в папке work и нажмите кнопку «OK».

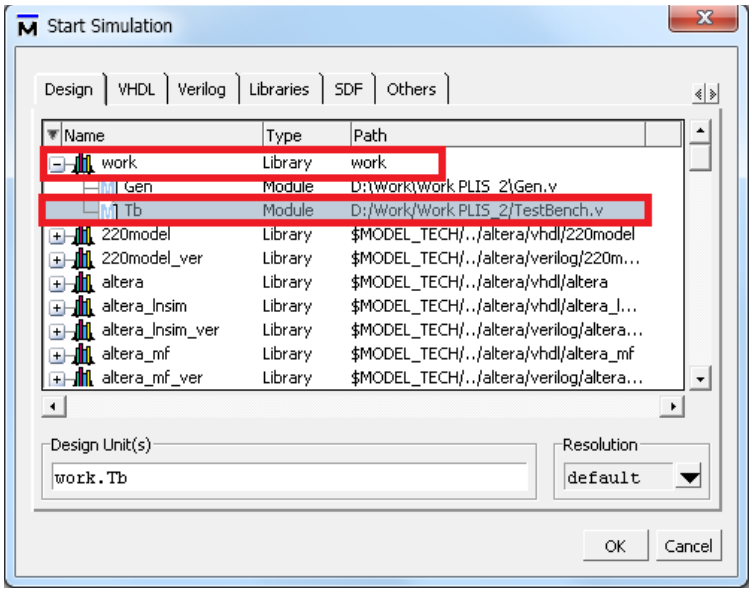


Рисунок 2.21 – Выбор тестбенч в среде симуляции ModelSim

2.2.26. Откройте вкладку «sim» и перетяните из левого окна в правое окно «wave», как показано на рисунке 2.22.

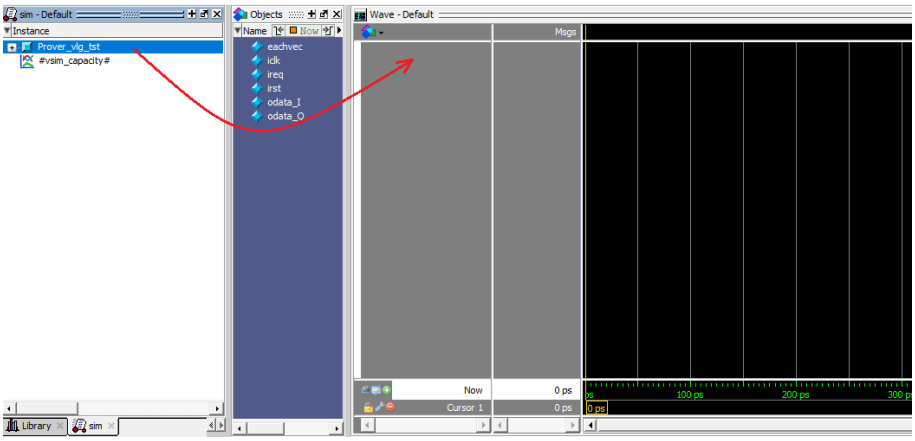


Рисунок 2.22 – Добавление сигналов в среде симуляции ModelSim

2.2.27. Установите время симуляции 2100 ps, в выделенной области на рисунке 2.23. После чего нажмите кнопку «Сброс моделирования ()», далее кнопку «Запуск симуляции на выбранное время ()», выделенные на рисунке 2.23.

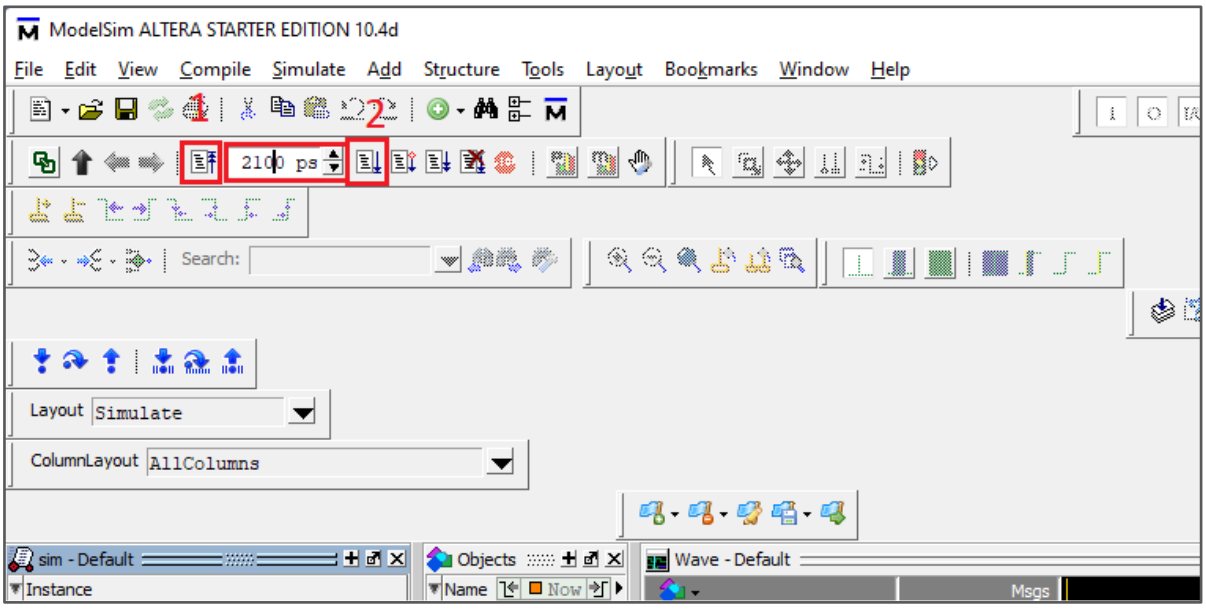


Рисунок 2.23 – Настройка симуляции

2.2.28. В окне «Wave – Default», приведенном на рисунке 2.24, должны отобразиться состояния входных и выходных сигналов.

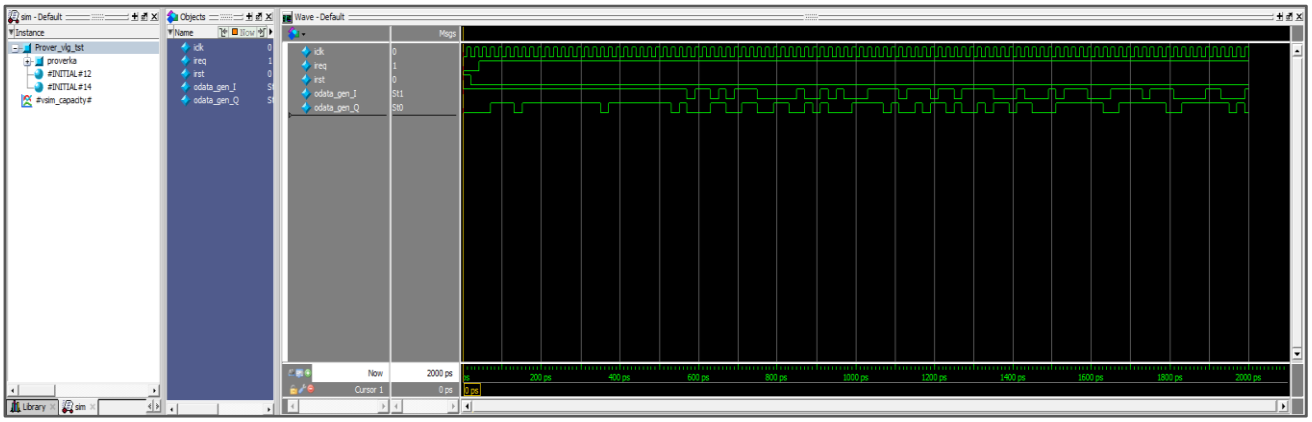


Рисунок 2.24 – Результаты симуляции

Работа № 2

«Формирование спектра OFDM символа»

Целью работы является реализация блока формирования спектра OFDM символа на языке описания аппаратуры Verilog.

Задачами данной работы являются:

1) Реализация блок формирования спектра;
2) Реализация блоков оперативной памяти;
3) Верификация созданных блоков в ModelSim.

1. Теоретическая часть

Блок формирования спектра OFDM симовола (Mapper) используются для расстановки пилотных сигналов, информационных данных и защитных интервалов в Orthogonal frequency-division multiplexing (OFDM) символах.

Пилот-сигнал – это поднесущая, использующаяся для частотной и фазовой синхронизации, а также для оценки характеристики канала связи с целью дальнейшего исправления ее эквалайзером.

Защитные интервалы помогают уменьшить межканальные помехи, возникающие в результате внеполосного излучения. Так же защитные интервалы помогают упростить процедуру фильтрации.

В данном проекте в качестве защитных интервалов используются нули.

В данном проекте будет использоваться следующая конфигурация OFDM сигнала:

– количество точек преобразования Фурье: 1024;

– шаг расстановки пилотов: 10 отсчетов;

– расположение защитных интервалов: 100 отсчетов слева и 99 отсчетов справа;

– расположение поднесущих с данными: от 100 до 512 отсчетов, от 514 до 924 отсчетов (общее количество поднесущих с данными: 824);

– расположение одной поднесущей для несущей частоты на 513 отсчете.

Общая структура расположения поднесущих в симоле OFDM показана на рисунке 1.1.

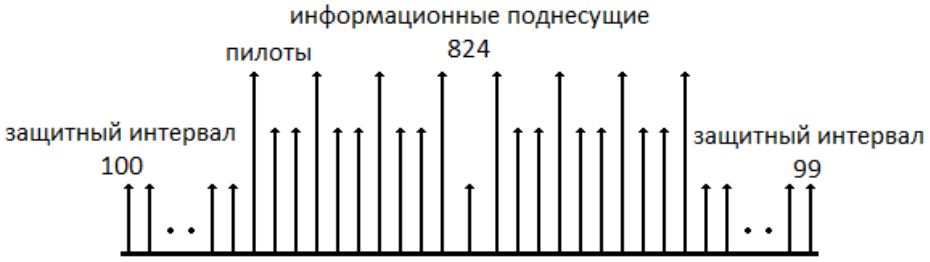


Рисунок 1.1 – Схематичное расположение поднесущих в OFDM символе

2. Практическая часть

Как говорилось ранее, mapper необходим для расстановки защитных интервалов, информационных данных и пилотных поднесущих в спектре сивола OFDM. Расстановка будет выполнятся согласно массиву конфигурации, который будет записан в памяти программируемой логической интегральной схемы (ПЛИС). В зависимости от счетчика всего символа (счетчик будет выступать в роли индекса массива) на выход будут подаваться либо защитный интервалы, либо пилотные поднесущие, либо информационные данные. Тип информационных данных будет зависеть от счетчика преамбулы, если этот счетчик равен нулю, то на место информационных данных, будут помещаться данные из памяти, если же

счетчик отличен от нуля, то информационные данные будут поступать из генератора псевдослучайных последовательностей (ПСП).

Графическое представление маппера представлено на рисунке 2.1.

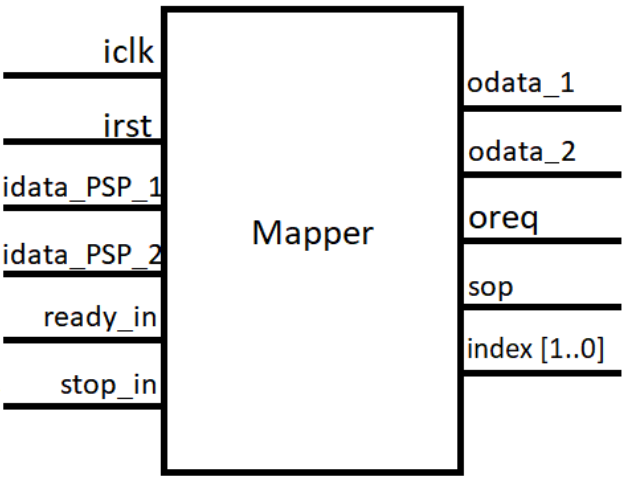


Рисунок 2.1 – Графическое представление маппера

Описание портов маппера приведено в таблице 2.1.

Таблица 2.1 – Описание портов маппера

Порты	Разрядность	Тип	I/O	Описание портов
iclk	1 бит	wire	Input	Тактирование 10 МГц
irst	1 бит	wire	Input	Сигнал сброса
ready_in	1 бит	wire	Input	Сигнал о запуске системы
stop_in	1 бит	wire	Input	Сигнал о остановке системы
idata_PSP_1	1 бит	wire	Input	Входной бит с генератора ПСП для синфазного сигнала($I(t)$)
idata_PSP_2	1 бит	wire	Input	Входной бит с генератора ПСП для квадратурного сигнала ($Q(t)$)
index	[1..0]	wire	Output	Тип выходных данных
odata_1	1 бит	reg	Output	Выходной порт маппера синфазного сигнала ($I(t)$)
odata_2	1 бит	reg	Output	Выходной порт маппера квадратурного сигнала ($Q(t)$)
oreq	1 бит	wire	Output	Сигнал разрешения для генератора ПСП
osop	1 бит	wire	Output	Сигнал начала OFDM символа

Для создания модуля маппера:

2.1 Зайдите в проект, созданный на первой практической работе, и создайте там файл Verilog HDL.

2.2 В появившемся текстовом редакторе опишите модуль маппера, присвоив ему имя «Mapper».

2.3 Объявите порты модуля «Mapper» в соответствии с рисунком 2.1 и таблицей 2.1. После чего опишите направление сигналов объявленных портов.

```
module Mapper(ready_in, stop_in, iclk, irst, idata_PSP_2, idata_PSP_1, index, odata_2, odata_1, oreq,osop);
input                ready_in;
input                stop_in;
input                iclk;
input                irst;
input                idata_PSP_2;
input                idata_PSP_1;
output      [1:0]    index;
output               oreq;
output               osop;
output reg           odata_2;
output reg           odata_1;
```

2.4 Объявите регистры:

– ena – сигнал разрешения;

– count – общий счетчик символа разрядностью 11 бит;

– count_pilot – счетчик пилотов разрядность 6 бит;

– count_preamb – счетчик преамбулы разрядностью 3 бита;

– stop – флаг, содержащий информацию о том, пришел ли сигнал о остановке работы системы.

Пример описание регистра:

```
reg                  [6:0]       count_pilot;
reg                  [10:0] count;
reg                  [2:0]       count_preamb;
reg                              ena;
reg                              stop;
```

2.5 Объявите шины:

– I_rom, Q_rom – ROM-память (ПЗУ – постоянное запоминающее устройство, используется хранения отчетов в памяти устройства) для преамбулы;

– rom_index – шина ROM-памяти для выходного порта index разрядностью 2 бита;

– rom_value – шина ROM-памяти.

Пример описание шины:

```
wire                             I_rom;
wire                             Q_rom;
wire                 [1:0]       rom_index;
wire                             rom_value;
```

2.6 Вам будут даны текстовые файлы index.txt, values.txt, I.txt и Q.txt, скопируйте их в папку с проектом.

2.7 С помощью оператора initial и оператора begin……end (если осуществляется более одного присвоения) задайте нулевые начальные состояния регистров:

```
initial
begin
                count=11'b0;
                count_pilot=7'b0;
                count_preamb=3'b0;
                odata_1=1'b0;
                odata_2=1'b0;
                ena=1'b0;
                stop=1'b0;
end
```

2.8 С помощью оператора присвоения assign реализуйте сигналы sop, index и oreq.

Сигнал sop – сигнал, обозначающий начало OFDM символа.

Сигнал index будет принимать значения, зависящие от типа данных на выходе маппера.

Сигнал oreq – сигнал разрешения работы генератора ПСП.

```
assign osop   =ena==1'b1 && count==11'b0 ? 1'b1:1'b0;
assign index  =rom_index;
assign oreq   = rom_index==2'b10 && count_preamb!=3'b0 ? 1'b1:1'b0;
```

2.9 Сохраните файл.

2.10 Создайте файл Verilog HDL для реализации ROM-памяти для хранения данных с текстовых файлов values.txt и index.txt (предварительно эти файлы необходимо перенести в папку с проектом).

Графическое представление ROM-памяти представлено на рисунке 2.2.

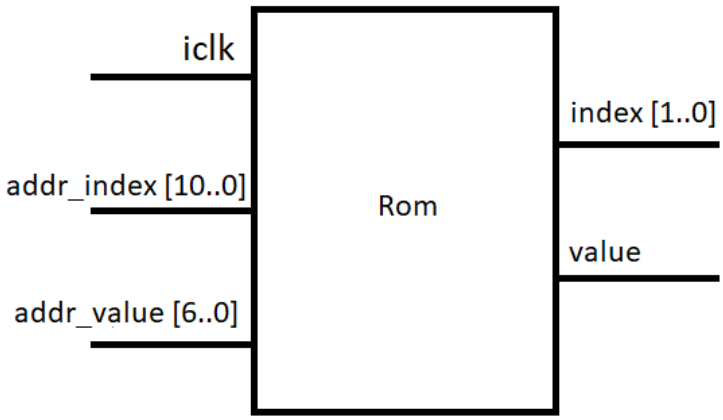


Рисунок 2.2 – Графическое представление ROM-памяти

Описание портов ROM-памяти приведено в таблице 2.2.

Таблица 2.2 – Описание портов ROM-памяти

Порты	Разрядность	Тип	I/O	Описание портов
iclk	1 бит	wire	Input	Тактирование 10 МГц
addr_index	[10..0]	wire	Input	Адрес памяти для index

Окончание таблицы 2.2

addr_value	[6..0]	wire	Input	Адрес памяти для value
index	[1..0]	reg	Output	Тип выходных данных
value	1 бит	reg	Output	Выход данных пилотов

2.11 Объявите порты модуля ROM-памяти в соответствии с рисунком 2.2 и таблицей 2.2. После чего опишите направление сигналов объявленных портов.

```
module Rom(
        input                   iclk,
        input       [10:0] addr_index,
        input       [6:0]       addr_value,
        output reg  [1:0]       index,
        output reg              value);
```

2.12 Объявите два регистра, которые будут выступать в роли ячеек памяти:
– двухразрядный регистр rom_index длинной 1056 бит;
– одноразрядный регистр rom_value длинной 91 бит.

```
reg         [1:0]       rom_index [1055:0];
reg                     rom_value     [90:0];
```

2.13 С помощью оператора initial необходимо считать данные из текстовых файлов в регистры памяти:

```
initial
begin
        $readmemb("Путь\\До\\Файла\\index.txt",rom_index);
        $readmemb("Путь\\До\\Файла \\values.txt",rom_value);
end
```

2.14 В always блоке реализуйте присвоение выходным портам (index и value) значения памяти (rom_index и rom_value) в зависимости от входных индексов (addr_index и addr_value). В конце добавьте endmodule.

```
always @ (posedge iclk)
        begin
                index <= rom_index[addr_index];
                value <= rom_value[addr_value];
                end
endmodule
```

2.15 Пропишите подключение модуля ROM-памяти в модуле «Mapper» по следующей схеме:
– на вход iclk подайте тактовый сигнал;
– на вход addr_index подайте счетчик count;
– на вход addr_value подайте счетчик count_pilot;
– к выходу index подключите память rom_index;

– к выходу value подключите память rom_value.

```
Rom Rom(
        .iclk(iclk),
        .index(rom_index),
        .value(rom_value),
        .addr_value(count_pilot),
        .addr_index(count)
);
```

2.16 Создайте еще один файл Verilog HDL. В этом файле будет реализована ROM-память для преамбулы.

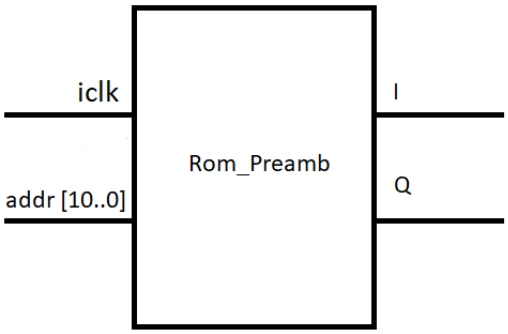


Рисунок 2.3 – Графическое представление модуля «Rom_preamb»

Таблица 2.3 – Описание портов модуля «Rom_preamb»

Порты	Разрядность	Тип	I/O	Описание портов
iclk	1 бит	wire	Input	Тактирование 10 МГц
addr	[10..0]	wire	Input	Адрес памяти для данных преамбулы
I	1 бит	reg	Output	Данные преамбулы реальной части
Q	1 бит	reg	Output	Данные преамбулы мнимой части

2.17 С помощью таблицы 2.3 и рисунка 2.3 задайте входные и выходные порты.

```
module Rom_preamb(
        input               iclk,
        input   [10:0]  addr,
        output          reg     I,
        output          reg     Q
);
```

2.18 Создайте два одноразрядных регистра длинной 1056 бит:

```
reg                         I_rom  [1055:0];
reg                         Q_rom [1055:0];
```

2.19 С помощью оператора initial считайте в созданные регистры данные из текстового файла «I.txt» и «Q.txt»:

```
initial
begin
        $readmemb("Путь\\До\\Файла\\I.txt",I_rom);
        $readmemb("Путь\\До\\Файла\\Q.txt",Q_rom);
end
```

2.20 С помощью always блока реализуйте присвоение выходным регистрам (I и Q) значения памяти (I_rom и Q_rom) в зависимости от номера addr. В конце добавьте endmodule.

```
always @ (posedge iclk)
begin
        I <= I_rom[addr];
        Q <= Q_rom[addr];
end
```

2.21 Пропишите подключение модуля «Rom_preamb» в модуле «Mapper» по следующей схеме:

- на вход iclk подайте тактовый сигнал;
- на вход addr подайте счетчик count;
- к выходу I подключите регистр памяти I_rom;
- к выходу Q подключите регистр памяти Q_rom.

```
Rom_preamb Rom_preamb(
        .iclk(iclk),
        .I(I_rom),
        .Q(Q_rom),
        .addr(count)
);
```

2.22 Далее в модуле «Mapper» с помощью always блока реализуйте три счетчика с асинхронным сбросом:

- count – счетчик символа (считает до 1055);
- count_pilot – счетчик пилотов, увеличивается каждый раз, когда в карте пилотов встречается 01, сбрасывается вместе с счетчиком count;
- count_preamb – счетчик преамбулы (считает до пяти), увеличивается каждый раз, когда счетчик count сбрасывается в нуль.

Счетчики должны работать по разрешающему сигналу ena. Сброс счетчиков должен происходить по сигналу сброса irst. При достижении счетчика count значения 1055 счетчики count и count_pilot сбрасываются в нуль, а счетчик count_preamb увеличивается на одно значение, при достижении значения 4 – сбрасывается.

```
always @(posedge iclk or posedge irst)
begin
        if(irst)
                begin
                        count<=11'b0;
                        count_pilot<=7'b0;
```

```
            end
        else if(count==11'b10000011111)
        begin
            count<=11'b0;
            count_pilot<=7'b0;
            if (count_preamb==3'b100)
            begin
                count_preamb<=3'b0;
            end
            else count_preamb<=count_preamb+1'b1;
        end
        else
            if (ena)
            begin
                count<=count+1'b1;
                if(rom_index==2'b01)
                    count_pilot<=count_pilot+1'b1;
            end
end
```

2.23 Далее необходимо реализовать задержку остановки системы до тех пор, пока счетчик count не досчитает до 1055 для того, чтобы последний символ OFDM полностью сформировался.

Принцип работы заключается в следующем: если на порт ready_in поступает единица и при этом значение регистра ena равно нулю, то регистру ena присваивается единица. Если сигнал stop_in или регистр stop равны единице и счетчик count досчитал до 1055, регистрам ena и stop присваиваются ноль. Если поступил сигнал stop_in, но при этом счетчик count не равен 1055, регистру stop присваивается единица, это значение будет свидетельствовать о том, что поступил сигнал об завершении работы системы.

```
always@(posedge iclk or posedge irst)
begin
        if(irst)
            begin
            ena<=1'b0;
            stop<=1'b0;
            end
        else if (ready_in==1'b1 & ena==1'b0)
            ena<=1'b1;
        else if((stop_in==1'b1 || stop==1'b1) && count==11'b10000011111)
        begin
            ena<=1'b0;
            stop<=1'b0;
        end
        else if(stop_in)
            stop<=1'b1;
end
```

2.24 Далее необходимо реализовать расстановку данных в символе. В зависимости от выходного значения index (таблица 2.4) на выход подаются защитные интервалы, пилот-

сигналы или информационные данные. Если счетчик count_preamb равен нулю, то вместо данных с генератора ПСП, на место информационных данных, берутся данные преамбулы.

Таблица 2.4 – Тип данных в зависимости от значения index

Параметр	Значение			
Значение index	00	01	10	11
Тип данных	Защитный интервал	Пилотные поднесущие	Информационные данные	-

```
always@(posedge iclk)
begin
	if(ena==0)
	begin
		odata_1<=0;
		odata_2<=0;
	end
	else if(ena)
	begin
		if(rom_index==1'b0)
		begin
			odata_2<=0;
			odata_1<=0;
		end
		else if(rom_index==2'b01)
		begin
			odata_1<=rom_value;
			odata_2<=rom_value;
			end
		else if(rom_index==2'b10)
		begin
			if (count_preamb==3'b0)
			begin
				odata_1<=I_rom;
				odata_2<=Q_rom;
			end
			else
			begin
				odata_1<=idata_PSP_1;
				odata_2<=idata_PSP_2;
			end
		end
	end
end
```

Не забывайте, что заканчиваться описание логики каждого модуля всегда должно ключевым словом «endmodule».

2.25 В файле верхнего уровня уберите шину с входного порта ireq (ireq станет шиной (wire ireq;)). Реализуйте подключение созданного модуля «Mapper». Обратите внимание, что теперь к выходным пинам модуля верхнего подключаются пины с выхода модуля «Mapper».

```
Gen generator(
        .iclk(in_clk),
        .irst(~irst),
        .odata_1(odata_gen_1),
        .odata_2(odata_gen_2),
        .ireq(req));
Mapper mapper(
        .iclk(in_clk),
        .irst(~irst),
        .idata_PSP_1(odata_gen_1),
        .idata_PSP_2(odata_gen_2),
        .odata_1(odata_1),
        .odata_2(odata_2),
        .oreq(req),
        .ready_in(~ready_in),
        .stop_in(~stop_in),
        .index(index),
        .osop(sop_mapper));
```

Сохраните изменения. Скомпилируйте проект. Откройте RTL Viewer (Tools => Netlist Viewers => RTL Viewer) и сравните получившуюся схему подключения маппера и генератора со схемой, на рисунке 2.4.

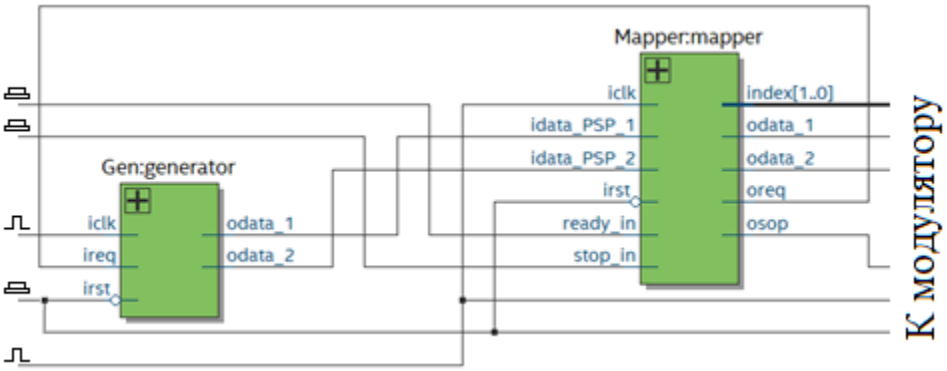


Рисунок 2.4 – Схема подключения модуля «Gen» к модулю «Mapper»

2.26 В модуле «testbench», который был создан в предыдущей работе, найдите часть кода, где происходит подключение файла верхнего уровня, и удалите строку подключения шины к порту ireq. После необходимо немного изменить файл модуля «testbench». В блоке initial замените написанные строки кода на представленные ниже. Это необходимо для того, чтобы создать симуляцию кнопок запуска и остановки.

```
ready_in=0;
stop_in=0;
irst=0;
#20
irst=1;
```

```
#20
irst=0;
#20
ready_in=1;
#20
ready_in=0;
#30000
stop_in=1;
#20
stop_in=0;
```

2.27 Сохраните все изменения и проверьте работу логики реализованного проекта в среде симуляции Modelsim. Для этого откройте проект симуляции, созданный ранее. Добавьте новые файлы (mapper.v, rom.v и rom_preamb.v). Скомпилируйте их. В результате должно сформироваться два OFDM символа, как показано на рисунке 2.5.

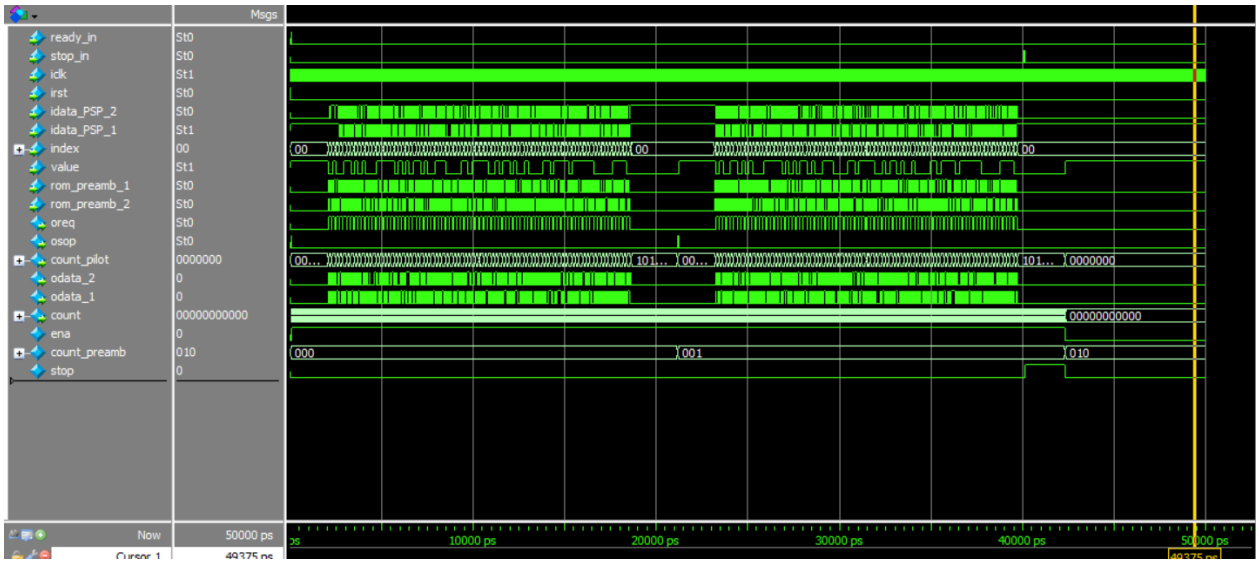


Рисунок 2.5 – Результат симуляции

2.28 По результату симуляции необходимо проверить следующее:

– счетчик count должен считать до 1055 и потом сбрасываться в ноль, и начинать счет заново;

– сигнал osop должен быть в состоянии единица, когда счетчик count в нулевом состоянии;

– сигнал oreq должен быть в состоянии нуля, когда сигнал index будет находиться в состоянии 01;

– сигнал ena должен быть в состоянии единицы от начала первого OFDM символа, до конца второго;

– начало данных на выходе модуля «Mapper» (odata_1 и odata_2) должно совпадать со 102 счетом счётчика count, конец данных должен быть на 925 счете.

Вернитесь в среду проектирования Quartus, перейдите по ссылке «Tools» → «Netlist Viewers» → «RTL Viewer». В открывшемся окне можно увидеть созданную логику соединения модулей. При создании и подключении последующих модулей, с помощью данного инструмента можно проверять прописанные соединения.

Работа № 3

«Квадратурный модулятор»

Целью работы является реализация QPSK модулятора.

Задачи работы:

1) Реализация модулятора;
2) Верификация модулятора в ModelSim.

1. Теоретическая часть

Модуляция – процесс изменения одного или нескольких параметров модулируемого несущего сигнала при помощи модулирующего сигнала.

В представленном передатчике будет использоваться QPSK модуляция (квадратурная фазовая манипуляция).

Квадратурная фазовая модуляция QPSK – один из видов модуляции, используемый в цифровой радиосвязи. Структурная схема QPSK модулятора приведена на рисунке 1.1.

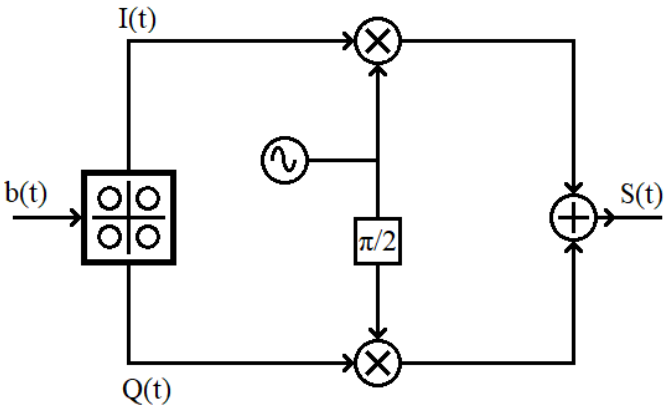


Рисунок 1.1 – Структурная схема QPSK модулятора

На входе модулятора четные биты (с номерами 0, 2, 4 и т.д.) выделяются из потока данных и перемножаются с несущей, получается синфазная составляющая сигнала ($I(t)$). В то же время, нечетные биты (с номерами 1, 3, 5 и т.д.) также выделяются из потока данных и перемножаются с той же несущей, сдвинутой на 90°, формируя квадратурную составляющую сигнала ($Q(t)$). Далее происходит суммирование этих составляющих и на выходе получается модулированный сигнал $S(t)$.

Формирования сигнала *QPSK* описывает формула:

$$E(t) = I(t)\cos(2\pi ft) + Q(t)\sin(2\pi ft). \quad (1.1)$$

Сам процесс QPSK модуляции сводится к выбору I (реальная) и Q (мнимая) компонент в зависимости от поступающей комбинации пары бит. Правило такого отображения можно описать таблицей:

Таблица 1.1 – Правила отображения

Биты (b_0 b_1)	Q	I
00	1	1
01	1	-1
10	-1	1
11	-1	-1

Или с помощью созвездия (рисунок 1.2).

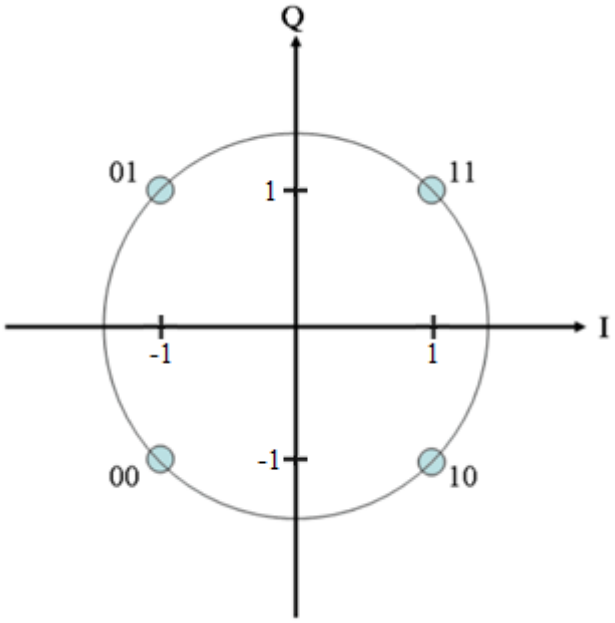


Рисунок 1.2 – Созвездие QPSK

2. Практическая часть

Для модуляции поднесущих OFDM сигнала в данной работе будет реализован QPSK модулятор. Принцип его работы будет заключатся в следующем: в заисимости от значений входного порта index, отвечающего за последовательность карты пилотов, модуляция будет выполняться двумя уровнями: ±26000 для информационных данных и ±32000 для пилотных. Защитные интервалы и положение несущей не модулируются, то есть всегда равны нулю.

На рисунке 2.1 представлена блок схема модулятора. В таблице 2.1 представлено описание входных и выходных портов модулятора.

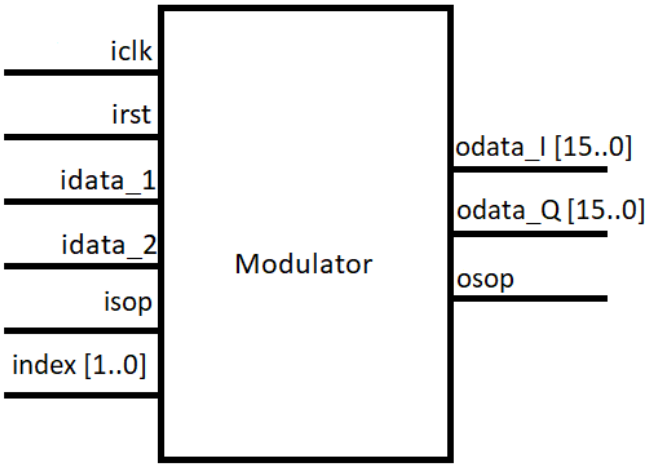


Рисунок 2.1 – Блок схема модулятора

Таблица 2.1 – Описание входов и выходов модулятора

Порты	Разрядность	Тип	Описание портов
irst	-	Input wire	Тактирование 10 МГц
iclk	-	Input wire	Сигнал сброса
idata_1	-	Input wire	Входные данные

Окончание таблицы 2.1

idata_2	-	Input wire	Входные данные
isop	-	Input wire	Индикатор начала символа с выхода маппера
index	[1..0]	Input wire	Тип входных данных
odata_I	[15..0]	Output reg signed	Модулированные данные Реальная часть
odata_Q	[15..0]	Output reg signed	Модулированные данные Мнимая часть
osop		Output reg	Индикатор начала символа с выхода модулятора

1.1. Зайдите в ранее созданный проект и добавьте в проект файл Verilog HDL, назовите его «Modulator».

1.2. Обозначьте следующие входы и выходы согласно таблице 2.1 и блок схеме на рисунке 2.1.

1.3. Объявите 16 разрядные регистры «value_data» и «value_pilot». Регистр «value_data» отвечает за уровень информационных данных, «pilot_data» – уровень пилотов.

```
module Modulator(iclk,irst, idata_1,idata_2, odata_I, odata_Q,isop,osop,index);
        input                           iclk;
        input                           irst;
        input                           idata_1;
        input                           idata_2;
        input           [1:0]           index;
        input                           isop;

        output reg signed   [15:0] odata_I;
        output reg signed   [15:0]      odata_Q;
        output reg                      osop;

        reg signed          [15:0] value_data;
        reg signed          [15:0] value_pilot;
```

1.4. В блоке initial регистрам «value_data», «value_pilot» задайте значения 16’d26000 и 16’d32000 соответственно. Остальным регистрам задайте нулевые значения.

```
initial
begin
        value_data=16'd16000;
        value_pilot=16'd22000;
        odata_I = 16'd0;
        odata_Q = 16'd0;
        osop   = 1'd0;
end
```

1.5. Модуляция производится согласно созвездию на рисунке 2.2.

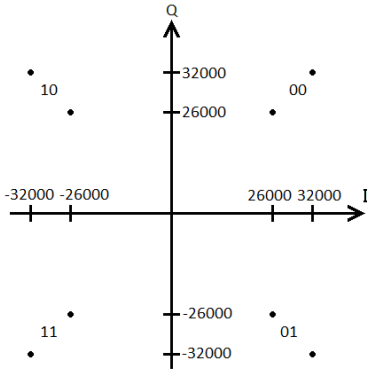


Рисунок 2.2 – Созвездие модулированного сигнала

Ниже приведен код программы.

```
always@(posedge iclk or posedge irst)
begin
	if(irst)
	begin
		odata_I<=0;
		odata_Q<=0;
	end
	else if(index==2'b0)
	begin
		odata_I<=0;
		odata_Q<=0;
	end
	else
	begin
		if(idata_1==0 && idata_2==0)
		begin
			if(index==2'b10)
			begin
				odata_I<=value_data;
				odata_Q<=value_data;
			end
			else
			begin
				odata_I<=value_pilot;
				odata_Q<=value_pilot;
			end
		end
		else if(idata_1==1 && idata_2==0)
			if(index==2'b10)
			begin
				odata_I<=--value_data;
				odata_Q<=value_data;
			end
			else
			begin
```

```
                        odata_I<=--value_pilot;
                        odata_Q<=value_pilot;
                end
        else if(idata_1==0 && idata_2==1)
        begin
                if(index==2'b10)
                begin
                        odata_I<=value_data;
                        odata_Q<=-value_data;
                end
                else
                begin
                        odata_I<=value_pilot;
                        odata_Q<=-value_pilot;
                end
        end
        else if(idata_1==1 && idata_2==1)
        begin
                if(index==2'b10)
                begin
                        odata_I<=-value_data;
                        odata_Q<=-value_data;
                end
                else
                begin
                        odata_I<=-value_pilot;
                        odata_Q<=-value_pilot;
                end
        end
    end
end
```

1.6. Для сигнала «osop» необходимо реализовать задержку на один такт и подать на выход «osop», так как данные на выходе, имеют задержку на один такт, относительно входных. В конце не забудьте добавить «endmodule».

```
always@(posedge iclk)        osop<=isop;
```

1.7. В файле верхнего уровня подключите этот блок к блоку «Mapper».

```
Modulator Modulator(
        .iclk(in_clk),
        .irst(~irst),
        .idata_1(data_I),
        .idata_2(data_Q),
        .odata_I(Modulator_I),
        .odata_Q(Modulator_Q),
        .isop(sop_mapper),
```

```
.osop(sop_modulator),
.index(index));
```

1.8. Выход модулятора («odata_I» и «odata_Q») подключите к выходным пинам модуля верхнего уровня («odata_1» и «odata_2»), перед этим сделав их 16-разрядными. Также в модуле “TestBench” измените разрядность пинов «odata_1» и «odata_2».

1.9. В «RTL Viewer» («Tools»=>«Netlist viewers»=> «RTL Viewer») проверьте соединения блоков. Верная схема подключения представлена на рисунке 2.3

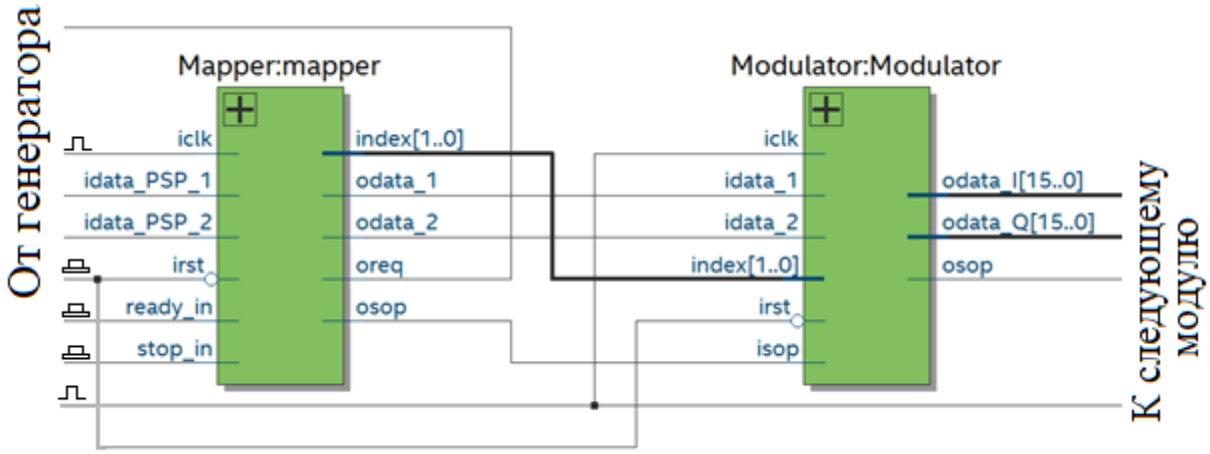


Рисунок 2.3 – Схема подключения

1.10. Откройте ранее созданный проект в Modelsim и добавьте файл с модулятором. Скомпилируйте. Проверьте в ModelSim следующее:

- На месте пилотов должно быть ±32000, на месте информационных ±26000. Определить тип данных определить по значению index: если значени index – 10, то это информационные данные, если 01 – пилотные данные, еслии 00 – защитные интервалы;
- Защитные интервалы и ноль в центре OFDM символа не должны модулироваться, т.е. оставаться 0. Размер OFDM символа с защитными интервалами должен быть равен 1056 (от одного osop до следующего), без защитных интервалов – 824;
- Сигнал osop должен иметь задержку в 1 такт от входящего «isop».

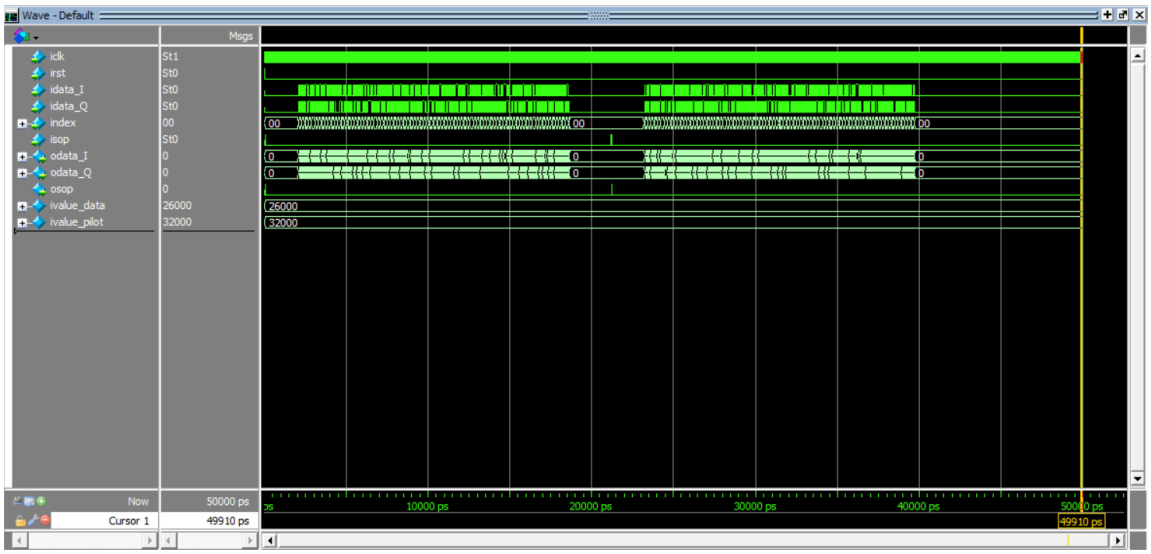


Рисунок 1.4 – Временные диаграммы в *Modelsim*

Работа № 4

«Быстрое Преобпазование Фурье. PLL. Нормирование сигнала»

Целью работы является подключение готовых блоков БПФ и нормировки, а также синтез блока PLL.

Задачи работы:

1) Подключение блока БПФ;
2) Подключение блока нормирования;
3) Синтез блока PLL;
4) Загрузка прошивки на отладочную плату и проверка работоспособности.

1. Теоретическая часть

Phase-Locked Loop

PLL (Phase-Locked Loop) – это специальный генератор со схемой подстройки частоты, управляемый напряжением (VCO – voltage-controlled oscillator). В генераторе реализовано сравнение фаз сигнала входной частоты и сигнала выходной частоты. Измеренная разность фаз этих частот через отрицательную обратную связь как раз и управляет частотой генератора, фиксируя ее на заданном значении. Схема работы PLL представлена на рисунке 1.1.

Рисунок 1.1 – Схема работы PLL блока

PLL широко используются в телекоммуникации, в компьютерах и других электронных устройствах.

На вход PLL подается некая входная частота опорного генератора Fin. Далее на счетчике N-counter опорная частота делится и получается другая частота Fref, которая поступает на фазовый детектор PFD (phase-frequency detector).

Фазовый детектор сравнивает фазы частот Fref и той, что поступает с делителя M-counter. Разность фаз, фильтруется и управляет генератором VCO.

На выходе управляемого генератора - новая частота Fvco. Фазовый детектор подает управляющее воздействие на генератор VCO до тех пор, пока не выполнится условие $F_{ref} = F_{vco} / M$. При этом условии частоты, подаваемые на детектор фаз равны. Таким образом, например, если $M = 2$, то частота генератора VCO должна получиться в 2 раза выше частоты F_{ref}. Последний этап – частота F_{vco} делится на выходном счетчике K-counter.

Подбирая коэффициенты N, M, K можно синтезировать довольно большой диапазон частот. Более сложные компоненты PLL позволяют перезагружать коэффициенты в процессе работы системы.

Преобразование Фурье

Преобразование Фурье – преобразование функции, превращающее её в совокупность частотных составляющих. Более точно, преобразование Фурье – это интегральное преобразование, которое раскладывает исходную функцию по базисным функциям, в качестве которых выступают синусоидальные (или мнимые экспоненты) функции, то есть представляет исходную функцию в виде интеграла синусоид (мнимых экспонент) различной

частоты, амплитуды и фазы. Преобразование названо по имени Жана Фурье.

Преобразование Фурье является основоположником спектрального анализа. Спектральный анализ – это способ обработки сигналов, который позволяет охарактеризовать частотный состав измеряемого сигнала. В зависимости от того, каким образом представлен сигнал, используют разные преобразования Фурье. Различают несколько видов преобразования Фурье:

– Непрерывное преобразование Фурье (в англоязычной литературе Continue Time Fourier Transform – CTFT или, сокращенно, FT);

– Дискретное преобразование Фурье (в англоязычной литературе Discrete Fourier Transform – DFT);

– Быстрое преобразование Фурье (в англоязычной литературе Fast Fourier transform – FFT).

Преобразование Фурье является математическим инструментом, применяемым в различных научных областях. В некоторых случаях его можно использовать как средство решения сложных уравнений, описывающих динамические процессы, которые возникают под воздействием электрической, тепловой или световой энергии. В других случаях оно позволяет выделять регулярные составляющие в сложном колебательном сигнале, благодаря чему можно правильно интерпретировать экспериментальные наблюдения в астрономии, медицине и химии. Непрерывное преобразование фактически является обобщением рядов Фурье при условии, что период разлагаемой функции устремить к бесконечности. Таким образом, классическое преобразование Фурье имеет дело со спектром сигнала, взятым во всем диапазоне существования переменной.

Непрерывное преобразование Фурье используют, как правило, в теории при рассмотрении сигналов, которые изменяются в соответствии с заданными функциями, но на практике обычно имеют дело с результатами измерений, которые представляют собой дискретные данные. Результаты измерений фиксируются через равные промежутки времени с определённой частотой дискретизации, например, 16000 Гц или 22000 Гц. Однако в общем случае дискретные отсчёты могут идти неравномерно, но это усложняет математический аппарат анализа, поэтому на практике обычно не применяется.

Рисунок 1.2 – Исходный аналоговый сигнал и дискретный сигнал

Дискретное преобразование Фурье – это одно из преобразований Фурье, широко применяемых в алгоритмах цифровой обработки сигналов.

Быстрое преобразование Фурье (БПФ, Fast Fourier transform – FFT) представляет собой определенный алгоритм вычисления, который позволяет уменьшить количество производимых действий относительно прямого (по формуле) вычисления ДПФ. В основе алгоритма заложено разбиение заданной последовательности отсчетов дискретного сигнала на несколько промежуточных последовательностей. Следует отметить, что алгоритм БПФ точнее стандартного ДПФ, т.к. при сокращении операций снижаются суммарные ошибки округления.

В настоящее время известны несколько алгоритмов быстрого преобразования Фурье, которые являются частными случаями единого алгоритма, базирующегося на задаче разбиения одного массива чисел на два с последующим рекурсивном вычислении каждого

массива чисел по дискретному преобразованию Фурье и объединении результатов расчетов.

Алгоритм БПФ с прореживанием по времени заключается в разбиении входного сигнала на два массива чисел, составленных соответственно из четных и нечетных отсчетов. После этого для каждого массива чисел рассчитывается ДПФ с образованием двух функций. На заключительном шаге выполняются базовые операции сложения и вычитания с умножением одного из компонентов на экспоненциальный множитель. В результате выполнения данных операций получаем функцию. Граф алгоритма БПФ с прореживанием по времени представлен ниже.

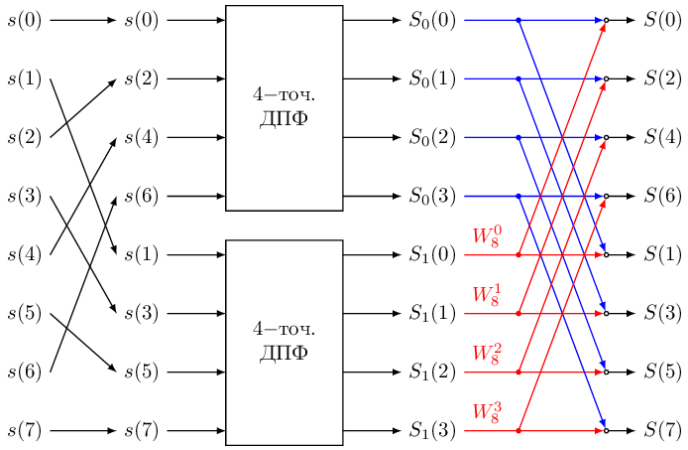


Рисунок 1.3 – Граф алгоритма БПФ с прореживанием по времени при делении исходного сигнала на четные и нечетные отсчеты

На структурной схеме базовую операцию сложения и вычитания с умножением одного из компонентов на экспоненциальный множитель принято показывать сигнальным графом, который в цифровой технике называется «бабочкой».

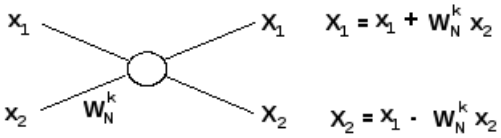


Рисунок 1.4 – Операция «бабочка», используемая при реализации алгоритма БПФ

Алгоритм БПФ с прореживанием по частоте заключается в разбиении входного сигнала пополам с образованием двух массивов чисел. После этого выполняются базовые операции сложения и вычитания с умножением двух компонентов на экспоненциальный множитель. На заключительном шаге рассчитывается ДПФ для четных и нечетных отсчетов с объединением результатов расчета. Граф алгоритма БПФ с прореживанием по частоте представлен ниже.

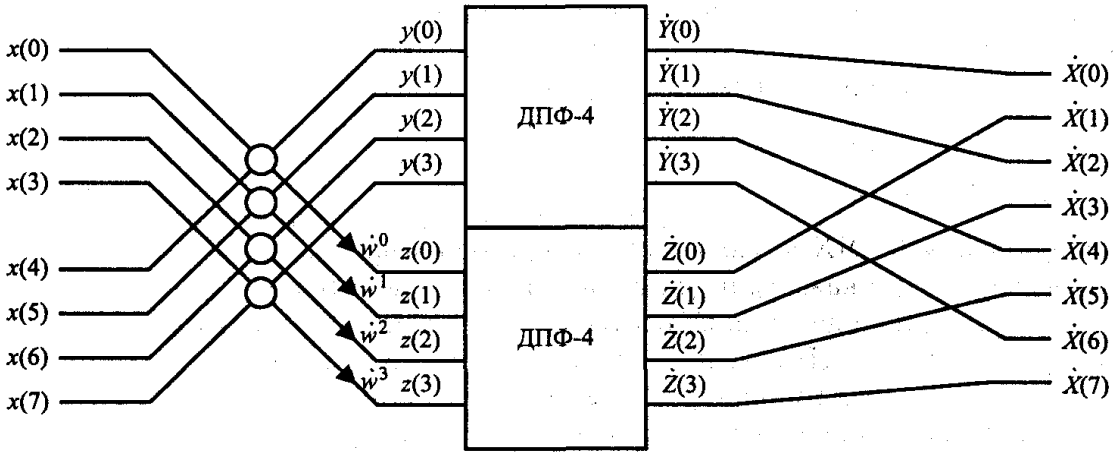


Рисунок 1.5 – Граф алгоритма БПФ с прореживанием по частоте при делении исходного сигнала пополам

2. Практическая часть

2.1 PLL блок

Данный блок будет переобразовывать тактовый сигнал платы, равный 50 МГц, в две частоты: 40 МГц и 10 МГц. Изначально сигнал будет формироваться на частоте 10 МГц, для обеспечения полосы сигнала в 10 МГц. Частота в 40 МГц будет использоваться для интерполяции сигнала в 4 раза.

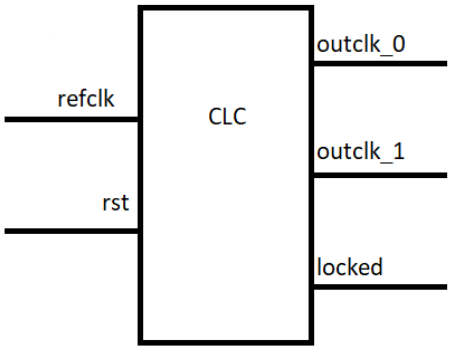


Рисунок 2.1 – Блок схема PLL блока

Таблица 2.1 – Описание входов и выходов PLL

Порты	Разрядность	Тип	I/O	Описание портов
refclk	-	wire	Input	Тактирование 50 МГц
rst	-	wire	Input	Сигнал сброса
outclk_0	-	wire	Output	Тактирование 10 МГц
outclk_1	-	wire	Output	Тактирование 40 МГц
locked	-	wire	Output	Сигнал сброса

Для создания PLL блока следуйте рисункам 2.2 – 2.5.

Для начала откроем IP Catalog (рисунок 2.2). Этот каталог позволяет создавать новые модули из библиотеки Quartus Prime с необходимыми параметрами, например, PLL, ROM, RAM, FIFO и т.д.

Рисунок 2.2 – Создание PLL: шаг 1

В списке библиотек откройте Basic Function => Clocks: PLL and Resets => PLL =>

Altera PLL, как показано на рисунке 2.3.

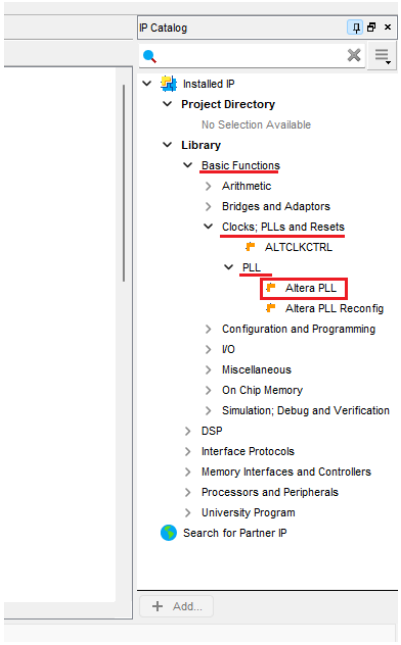


Рисунок 2.3 – Создание PLL: шаг 2

Введите полное имя файл: в качестве папки сохранения укажите папку с проектом, а в качестве имени модуля используйте «CLC».

Рисунок 2.4 – Создание PLL: шаг 3

Далее откроется окно, где указывается необходимые параметры PLL. Установите параметры как на рисунке 2.5.

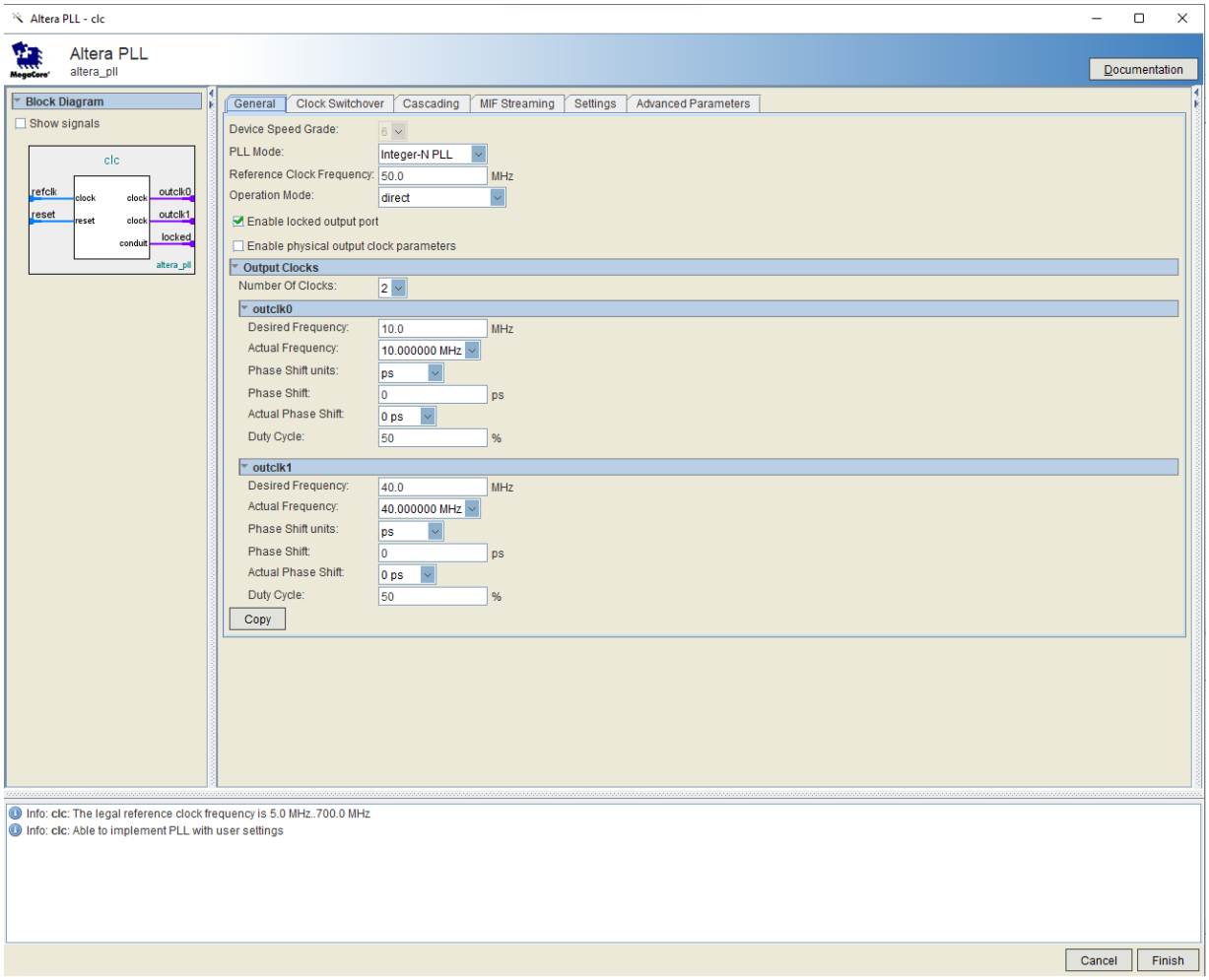


Рисунок 2.5 – Настройка параметров блока PLL

В будущем, при подключении других блоков будьте внимательны, в некоторых необходимо тактирование 10 МГц, в других 40 МГц, это будет прописано в таблице, после блок схемы блока.

После нажатия кнопки Finish появится окно, в котором вам предложат автоматически добавить файлы в ваш проект – нажмите «Yes».

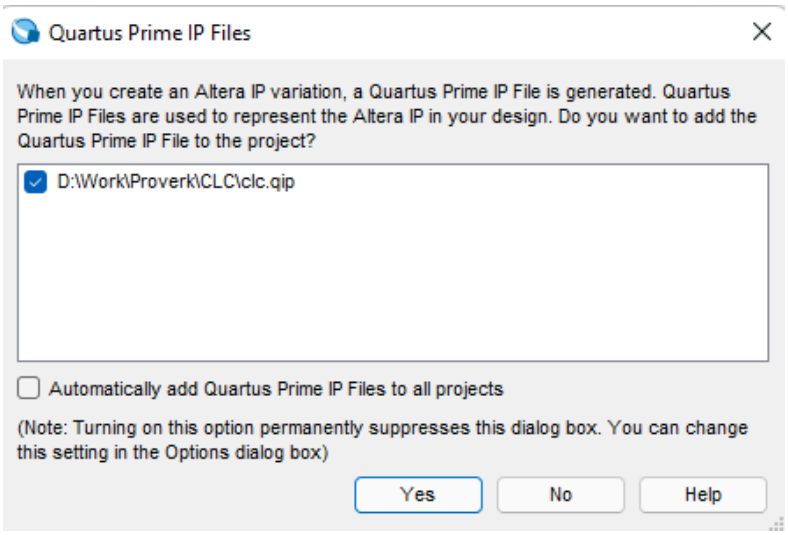


Рисунок 2.6 – Завершение

Пропишите подключение PLL блока в топ-файле. На его вход необходимо подать входный тактовый сигнал (in_clc) частотой 50 МГц. С первого выхода (outclk0) будет подаваться тактовый сигнал 10 МГц, а со второго (outclk1) 40 МГц.

```
clc clc(
        .refclk  (in_clk),
        .rst             (~irst),
        .outclk_0        (clk),
        .outclk_1        (clk4),
        .locked(locked)
);
```

На входной пин iclk всех подключенных блоков (Generator, Mapper, Modulator) подайте сигнал 10 МГц. А на вход irst подайте сигнал ~locked.

2.2 FFT блок

Теперь подключите готовый блок FFT. Скопируйте библиотеки в папку с проектом. Следом откройте окно с настройками settings (Assigment => Settings). В появившемся окне перейдите во вкладку Files. Найдите и откройте папку с FTT. Добавьте в проект два файла: fft_1024_16.vhd и fft_core/lib/fft_pack.vhd.

Перейдите во вкладку Libraries. В окне Project libraries добавьте следующие пути:

1) fft_1024_16/fft_core/lib/
2) fft_1024_16/fft_core/
3) fft_1024_16/

У вас должно получится как на рисунке 2.7.

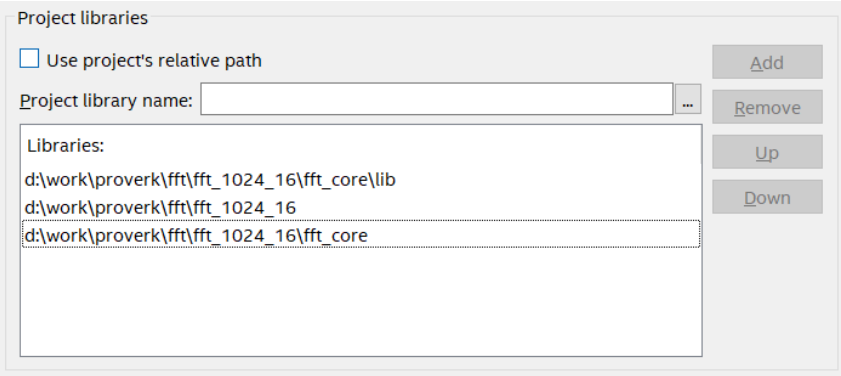


Рисунок 2.7 – Библиотеки FFT

Перед подключением блока FFT необходимо в модуле верхнего уровня создать две 16 разрядные знаковые шины и подключить их к выходу модулятора (odata_I и odata_Q).

```
wire    signed [15:0]       Modulator_I;
wire    signed [15:0]       Modulator_Q;
```

Схема блока представлена на рисунке 2.8.

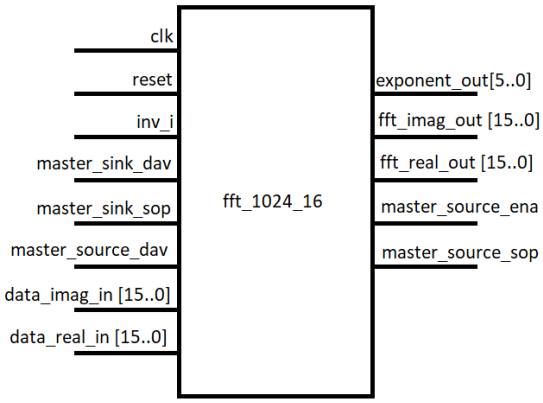


Рисунок 2.8 – Блок схема FFT

Таблица 2.2 – Описание входов и выходов FFT

Порты	Разрядность	Тип	I/O	Описание портов
clk	-	wire	Input	Тактирование 10 МГц
reset	-	wire	Input	Сигнал сброса
inv_i	-	wire	Input	Направление преобразования: 1 – обратное, 0 – прямое
master_sink_dav		wire	Input	Сигнал доступности
master_source_dav	-	wire	Input	Сигнал доступности
data_imag_in	[15..0]	wire signed	Input	Мнимая часть входных данных
data_real_in	[15..0]	wire signed	Input	Реальная часть входных данных
exponent_out	[5..0]	wire signed	Output	Показатель степени блока со знаком
fft_imag_out	[15..0]	wire signed	Output	Мнимая часть выходных данных
fft_real_out	[15..0]	wire signed	Output	Реальная часть выходных данных
master_source_ena	-	wire	Output	Сигнал доступности
master_source_sop	-	wire	Output	Индикатор начала символа

Далее необходимо подключить блок FFT. На вход inv_i, master_source_dav и master_sink_dav подайте логическую единицу.

```
fft_1024_16 fft_submod(
        .clk                (clk),
        .reset              (~locked),
        .inv_i              (1'b1),
        .data_real_in       (Modulator_I),
        .data_imag_in       (Modulator_Q),
        .fft_real_out       (source_real),
        .fft_imag_out       (source_imag),
        .exponent_out       (exp_out),
```

```
.master_sink_dav        (1'b1),
.master_sink_ena        (),
.master_sink_sop        (sop_modulator),
.master_source_sop  (fft_sop_out),
.master_source_eop  (fft_eop_out),
.master_source_ena  (fft_ena_out),
.master_source_dav  (1'b1));
```

Сохраните изменения.

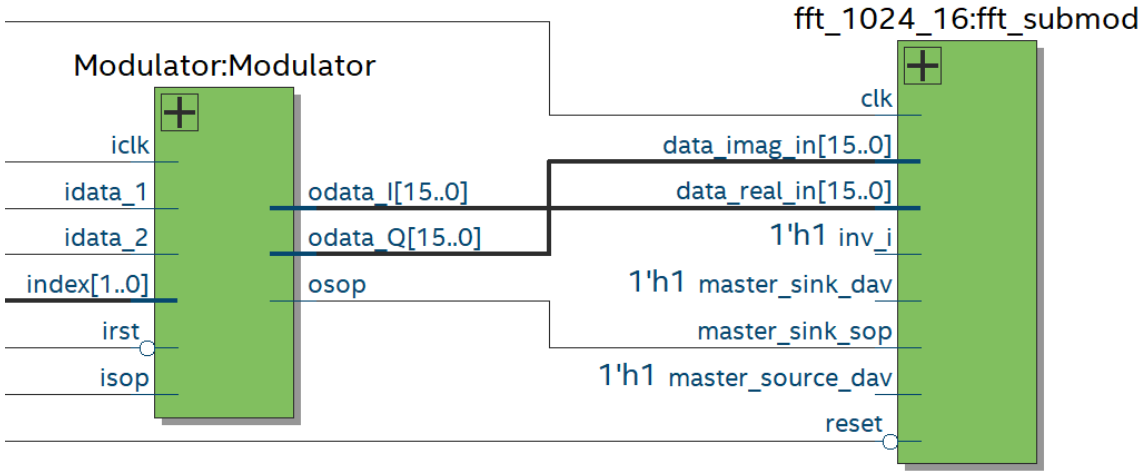


Рисунок 2.9 – Схема подключения блока FFT

Так же в модуле верхенего уровня на каждый блок вместо irst, нужно подать его отрицание (~irst). На маппер подайте отрицание сигналов ready_in и stop_in. Это необходимо из-за принципа работы кнопок на плате: в не нажатом состоянии, выход с кнопки равен 1, а при нажатии, происходит заземление и сигнал становится равен 0.

Теперь скомпилируйте. После компилирования необходимо виртуализировать выход odata_1 и odata_2. В приложении 1 написан процесс виртуализации.

После виртуализации запустите еще раз процесс компиляции и перейдите в Pin planner (Assigments =>Pin Planner).

Откроется окно. Внизу этого окна необходимо задать пины (столбец Location) как на рисунке 2.10.

Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatior	I/O Standard
altera_...ved_tck	Input				PIN_AC5	2.5 V ...fault)
altera_r...rved_tdi	Input				PIN_U8	2.5 V ...fault)
altera_...ved_tdo	Output				PIN_AB9	2.5 V ...fault)
altera_...ved_tms	Input				PIN_V9	2.5 V ...fault)
iclk	Input	PIN_Y26	5B	B5B_N0	PIN_Y26	2.5 V
irst	Input	PIN_AE12	3A	B3A_N0	PIN_AE12	2.5 V
odata_1	Output	PIN_A9	8A	B8A_N0	PIN_A9	2.5 V
odata_2	Output	PIN_A8	8A	B8A_N0	PIN_A8	2.5 V
ready_in	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	2.5 V
stop_in	Input	PIN_AD9	3A	B3A_N0	PIN_AD9	2.5 V
<<new node>>						

Рисунок 2.10 – Таблица подключния пинов

Сохраните изменения и еще раз скомпилируйте проект. Подключите плату для этого: подключите кабель к разъему mini-USB (на плате написано название USB Blaster), второй конец воткните в разъем USB в компьютере. После нажмите кнопку включения.

Теперь перейдите в Tools =>Signal Tap Logic Analyzer (рисунок 2.11).

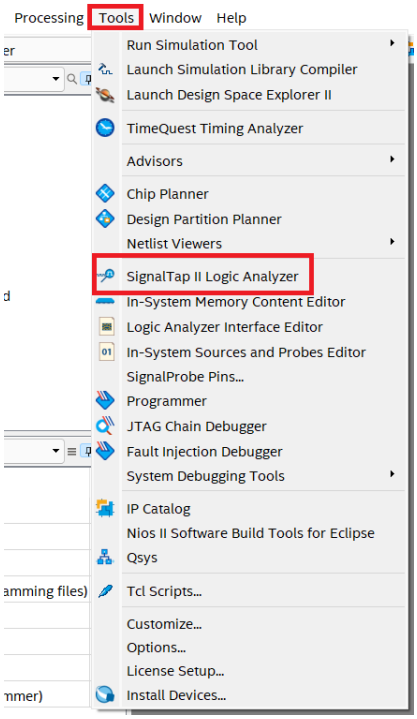


Рисунок 2.11 – Открытие SignalTap

В открывшемся окне в левом нижнем углу перейдите во вкладку Setup, как показано на рисунке 2.12.

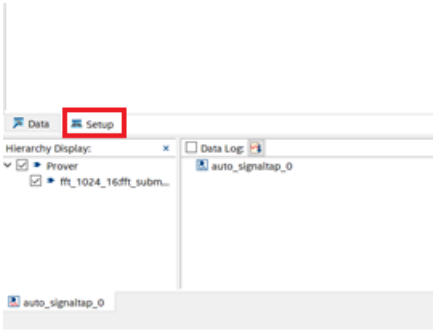


Рисунок 2.12 – Вкладка Setup

Подключите плату к компьютеру и включите её. В правом верхнем углу необходимо выбрать устройство, для это нажмите на кнопку Setup, как показано на рисунке 2.13.

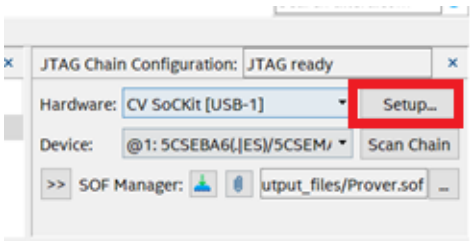


Рисунок 2.13 – Выбор платы: шаг 1

Откроется окно. В нем выберете CV SoCKit и нажмите close, как на рисунке 2.14.

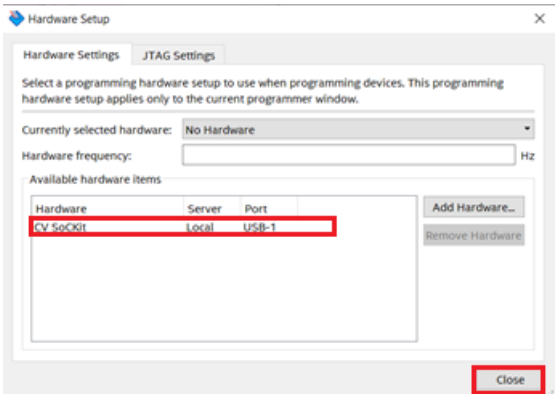


Рисунок 2.14 – Выбор платы: шаг 2

Далее выберете прошивку в папке output_files/*Имя_проекта*.sof (рисунок 2.15).

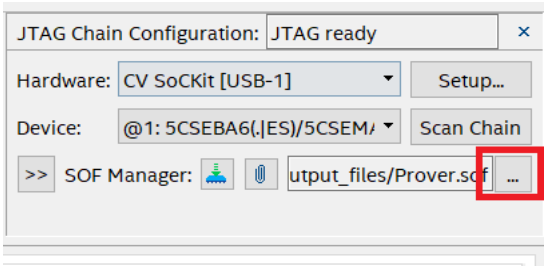


Рисунок 2.15 – Выбор прошивки

Далее необходимо задать параметры для SignalTap.

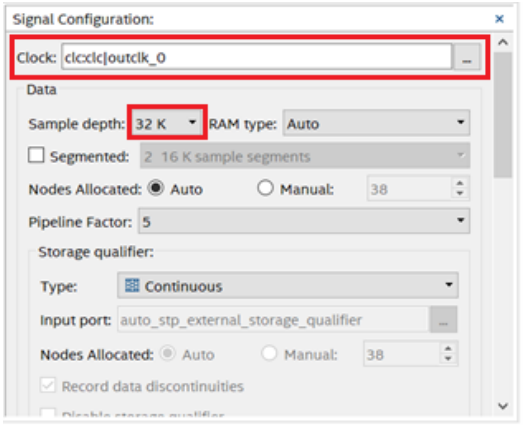


Рисунок 2.16 – Парметры SignalTap

Sample deplth, количество отображаемых бит, выберете 32 K. Clock должен быть такой же частоты, с которой работает блок, с которого вы снимаете данные. Так как блок, с которого будете брать данные работает на частоте 10 МГц, то и тут указывайте тактовый сигнал частотой 10 МГц, т.е. тактовый сигнал outclk_0. Для этого выполните следующее:

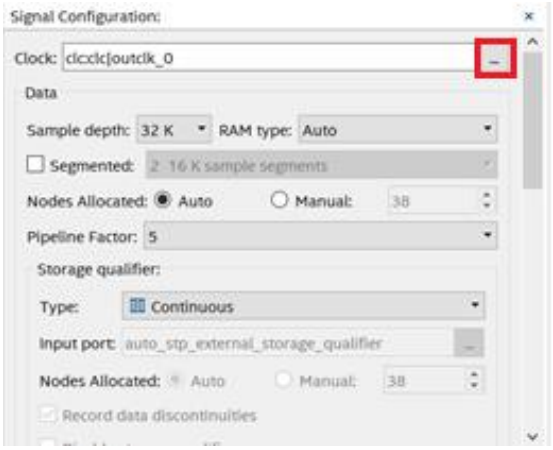


Рисунок 2.17 – Выбор тактового сигнала: шаг 1

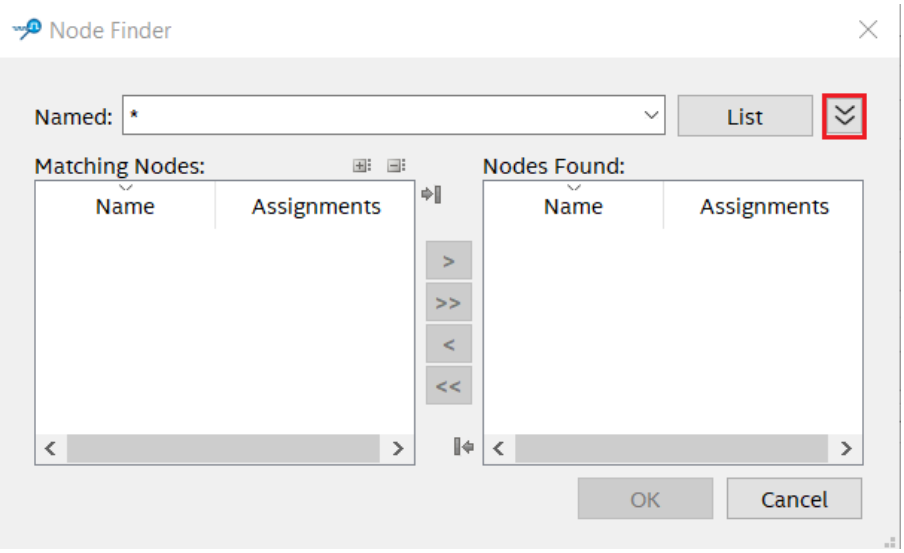


Рисунок 2.18 – Выбор тактового сигнала: шаг 2

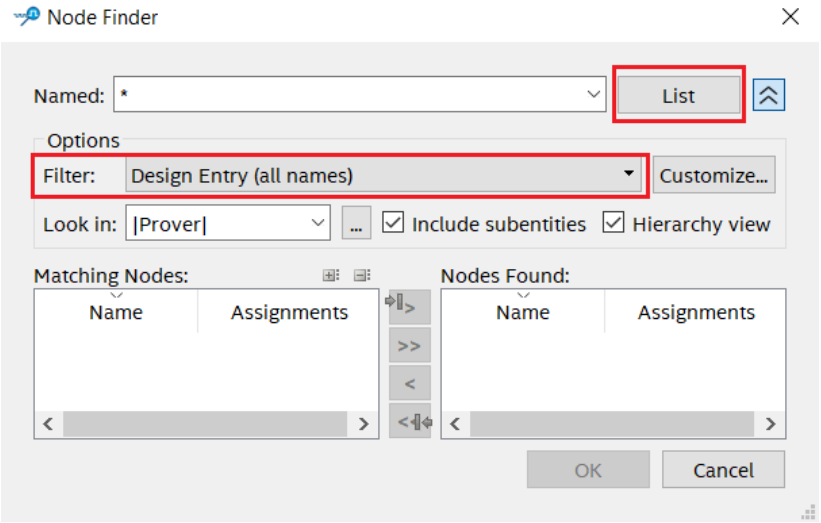


Рисунок 2.19 – Выбор тактового сигнала: шаг 3

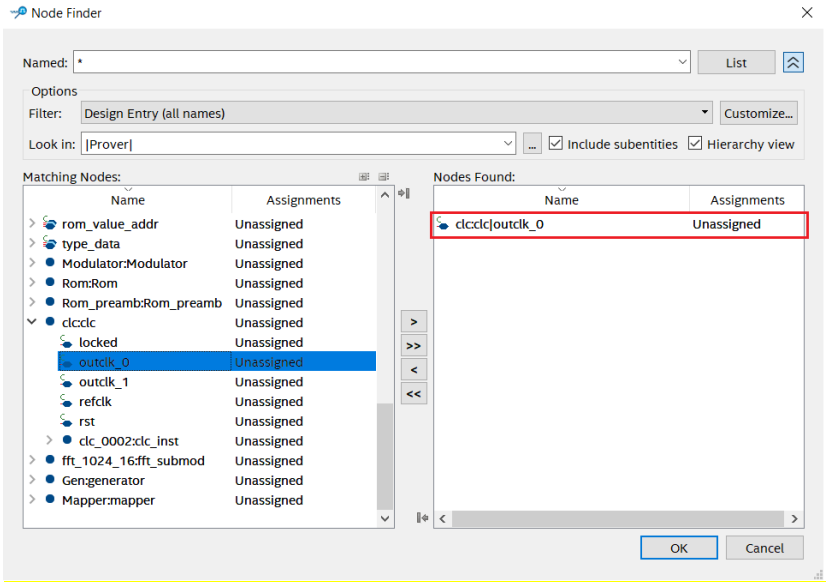


Рисунок 2.20 – Выбор тактового сигнала: шаг 4

Теперь нужно выбрать узлы, с которых нам необходимо отобразить данные в Signal Tap, для этого дважды нажмите ЛКМ по выделенной области (рисунок 2.21)

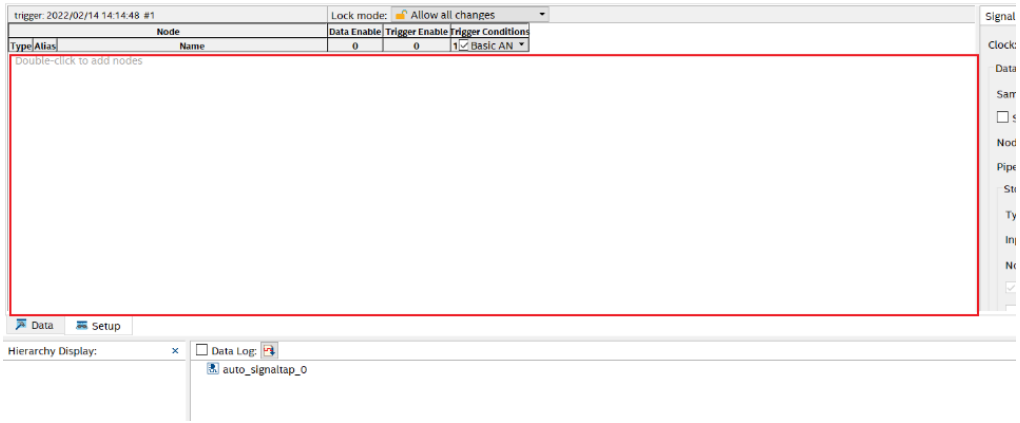


Рисунок 2.21 – Добавление узлов

Из списка найдите блок fft и возьмите его 4 выхода, как это показано на рисунке 2.22.

trigger: 2022/03/05 18:04:55 #1 — Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	39	39	1 Basic AN
		fft_1024_16:fft_submod|exponent_out[5..0]	✓	✓	XXh
		fft_1024_16:fft_submod|fft_imag_out[15..0]	✓	✓	XXXXh
		fft_1024_16:fft_submod|fft_real_out[15..0]	✓	✓	XXXXh
		fft_1024_16:fft_submod|master_source_sop	✓	✓	

Рисунок 2.22 – Добавленные узлы

Сохраните и скомпилируйте еще раз. Далее необходимо загрузить получившуюся прошивку на плату, как показано на рисунке 2.23. После запустите SignalTap, как на рисунке 2.24. Обозначение кнопок Start, Stop, Reset на плате показано на рисунке 2.26.

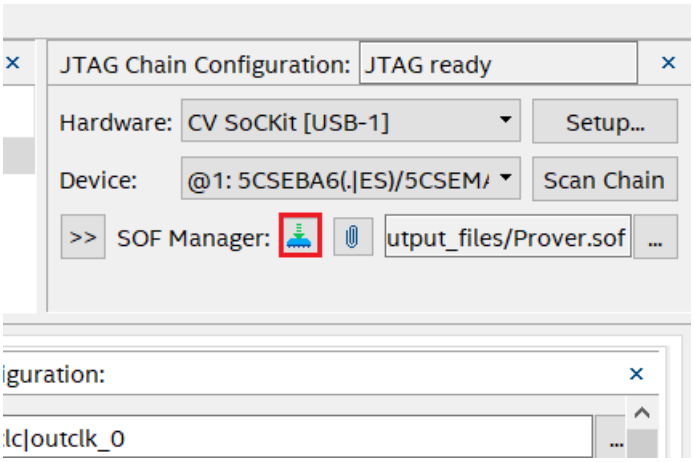


Рисунок 2.23 – Загрузка прошивки на плату

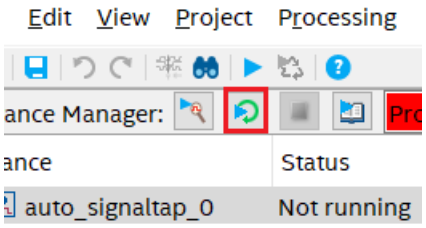


Рисунок 2.24 – Запуск SignalTap

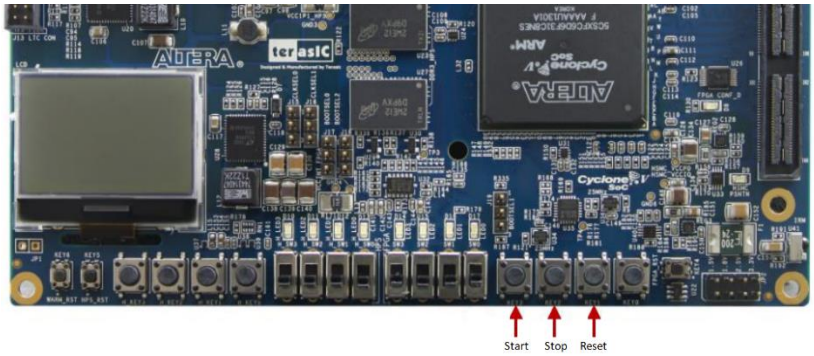


Рисунок 2.25 – Обозначение кнопок на плате

Теперь в Signal Tap вы должны увидеть данные. Данные могут отображаться в шестнадцатеричной системе счисления, чтобы изменить на десятеричную, необходимо нажать ПКМ на название узла, в самом низу найдите Bus Display Format и в подменю выберете Signal Decimal in Two Complement. После этого данные должны отображаться в привычном виде.

После этого необходимо проверить полученные данные в MATLAB. Для этого нажимаем ПКМ в выделенной области, как на рисунке 2.26, и выберете Create SignalTap II List File:

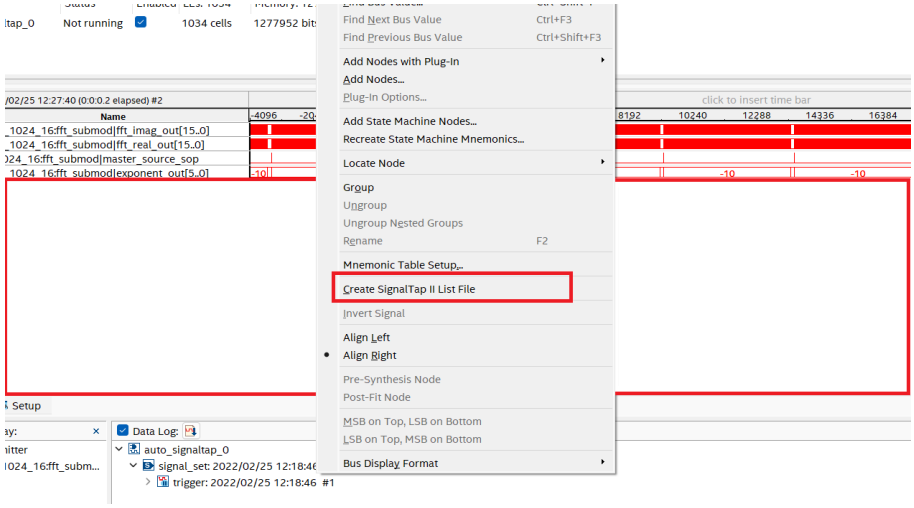


Рисунок 2.26 – Скриншот программы

После появиться окно со значениями. Найдите начало символа по сигналу sop (сигнал sop равен 1 на момент начала символа), выделите 1024 бита и скопируйте их (выделите второй и третий столбец зажав комбинацию SHIFT+ALT) в текстовый файл (нужно скопировать 2 столбца: с реальной и мнимой частью).

Откройте файл Prover.m. Считайте эти данные в MATLAB. Вместо 1.txt, пропишите название файла, в котором хранятся отсчеты (файл с отсчетами должен хранится в папке с файлом Prover.m).

```
Data=dlmread('1.txt');                  % Чтение данных из файла
Sig=(complex(Data(:,1),Data(:,2))); % Обьединение в комплекс
```

Сделайте прямое преобразование Фурье, постройте спектр и созвездие сигнала.

```
Sig_fft=fft(Sig);       %Преобразование Фурье
plot(abs(Sig_fft));     %Построение спектра
scatterplot(Sig_fft);  %Построение созвездия
```

У вас должно получится так же, как на рисунках 2.27 – 2.28.

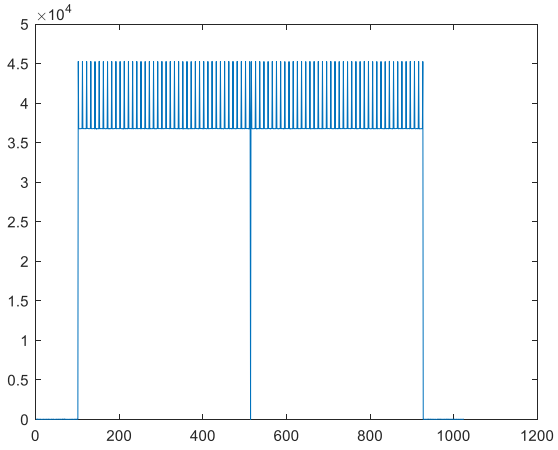


Рисунок 2.27 – Спектр сигнала

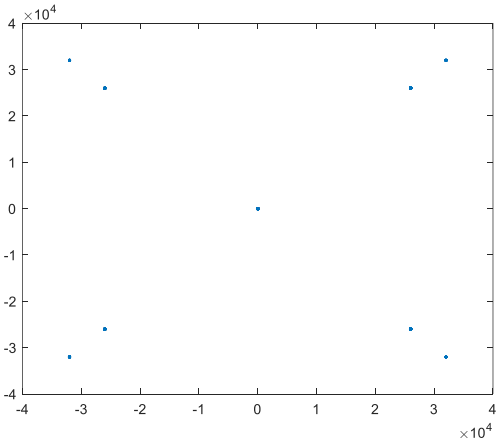


Рисунок 2.28 – Созвездие сигнала

2.3 Блок нормирвоания сигнала

Нормалайзер необходим для того, чтобы данные с выхода блока FTT преобразовать из шестнадцатиразрядных данных в четырнадцатиразрядные и увеличить входные данные в несколько раз по абсолютному значению. Блок схема представлена на рисунке 2.29. В таблице 3 представлено описание входных и выходных портов.

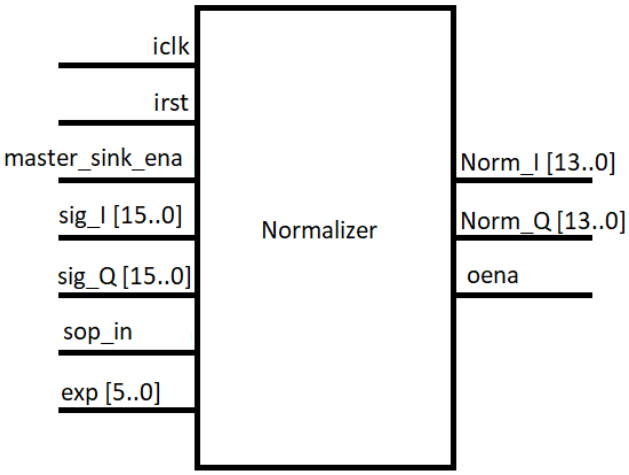


Рисунок 2.29 – Блок схема блока нормирования

Таблица 2.3 – Описание входов и выходов нормалайзера

Порты	Разрядность	Тип	I/O	Описание портов
iclk	-	Wire	Input	Тактирование 10 МГц
irst	-	Wire	Input	Сигнал сброса
master_sink_ena	-	Wire	Input	Сигнал разрешения
sig_I	[15..0]	Wire signed	Input	Мнимая часть входных данных
sig_Q	[15..0]	Wire signed	Input	Реальная часть входных данных
sop_in	-	Wire	Input	Индикатор начала символа
exp	[5..0]	Wire signed	Input	Показатель степени блока со знаком

Окончание таблицы 2.3

Norm_I	[13..0]	Wire signed	Output	Нормированная реальная часть выходных данных
Norm_Q	[13..0]	Wire signed	Output	Нормированная мнимая часть выходных данных
oena	-	Wire signed	Output	Сигнал разрешения

Перед подключение блока нормировки неоходимо создать два шестнадцатиразрядных провода (source_imag и source_real).

```
wire signed     [15:0]      source_real;
wire signed     [15:0]      source_imag;
```

Подключите готовый блок нормирования к выходу FFT блока.

```
Norm_1 Norm_1(
        .sig_I              (source_real),
        .sig_Q              (source_imag),
        .exp                (exp_out),
        .iclk               (iclk),
        .irst               (~irst),
        .sop_in             (fft_sop_out),
        .Norm_I                     (norm_I),
        .Norm_Q                     (norm_Q),
        .master_sink_ena    (fft_ena_out),
        .oena               (norm_ena)
);
```

После подключения скомпилируйте проект и проверьте подключение в RTL Viewer. Верная схема представлена на рисунке 2.30.

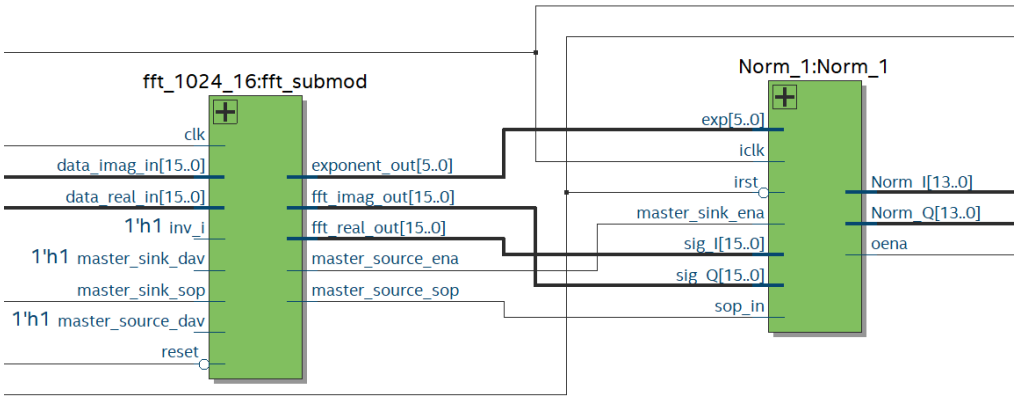


Рисунок 2.30 – Схема подключения нормалайзера

В RTL Viewer (Tools=>Netlist viewers=> RTL Viewer) проверьте соединения блоков.

Работа № 5

«Добавление циклического префикса»

Целью данной работы является реализация блока добавление циклического префикса.
Задачами данной работы являются:
1) Реализация блока добавления циклического префикса;
2) Верификация результатов с помощью Signal Tap.

1. Теоретическая часть

Циклический префикс добавляется в начало каждого OFDM-символа (рисунок 1.1) и представляет собой циклическое повторение окончания символа. Наличие циклического префикса создает временные паузы между отдельными символами, и если длительность защитного интервала превышает максимальное время задержки сигнала в результате многолучевого распространения, то межсимвольной интерференции не возникает.

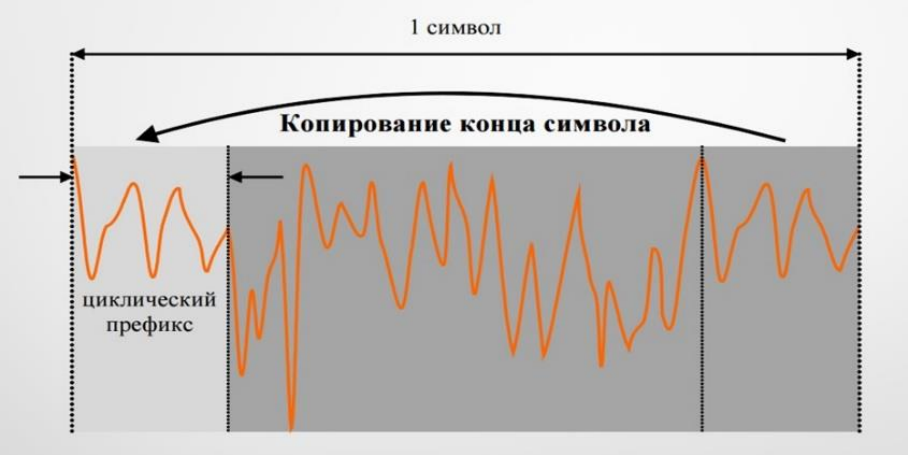


Рисунок 1.1 – Циклический префикс

Многолучевое распространение – это эффект, наблюдаемый при распространении сигналов. Возникает при условии существования в точке приема радиосигнала не только прямого, но и ещё одного или целого ряда отражённых или\и преломлённых лучей. Другими словами, на антенну приёмника приходят не только прямые лучи (непосредственно от самого источника), но и отражённые (от земной поверхности, зданий, строений и прочих объектов).

Межсимвольная интерференция (МСИ) является эффектом наложения в приемнике символов друг на друга. Особенностью многих линий радиосвязи (например, тропосферных, спутниковых, мобильных и т.п.) является многолучевой характер распространения радиосигнала. В точке приёма сигнал является суммой большого числа элементарных сигналов различных амплитуд со случайным временем запаздывания. Отдельные лучи могут запаздывать друг относительно друга на существенную величину (большая разность хода различных лучей в радиоканале), что и вызывает эффект МСИ.

Циклический префикс является избыточной информацией и в этом смысле снижает полезную (информационную) скорость передачи, но именно он служит защитой от возникновения межсимвольной интерференции. Указанная избыточная информация добавляется к передаваемому символу в передатчике и отбрасывается при приеме символа в приемнике.

2. Практическая часть

Задачей этого блока будет копирование последних 32 битов в символе в начало этого символа. Происходить это будет следующим образом: в буффер будет записываться весь символ, с 992 по 1024 отсчета счетчика, данные с входа будут подаваться на выход, начиная с 1025 отсчета, начинается чтение из буффера. Для записи следующего символа, будет использоваться второй буффер.

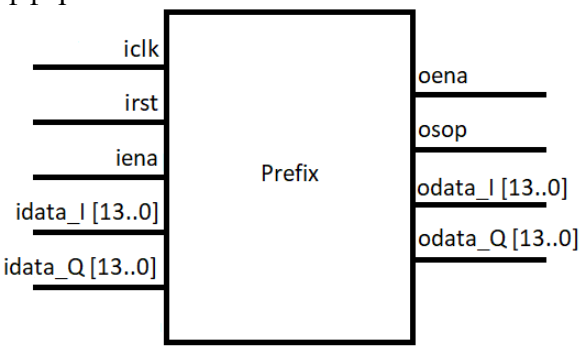


Рисунок 2.1 – Блок схема префикса

Таблица 2.1 – Описание входов и выходов префикса

Порты	Разрядность	Тип	I/O	Описание портов
iclk	-	Wire	Input	Тактирование 10 МГц
irst	-	Wire	Input	Сигнал сброса
iena	-	Wire	Input	Сигнал разрешения
idata_I	[13..0]	Wire signed	Input	Мнимая часть входных данных
idata_Q	[13..0]	Wire signed	Input	Реальная часть входных данных
osop	-	Wire	Output	Индикатор начала символа
oena	-	Reg	Output	Сигнал разрешения
odata_I	[13..0]	Reg signed	Output	Реальная часть выходных данных
odata_Q	[13..0]	Reg signed	Output	Мнимая часть выходных данных

Зайдите в созданный проект и создайте файл Verilog HDL. Назовите его Prefix.

```
module Prefix(iclk,irst,iena,odata_I,odata_Q,oena,osop,idata_I,idata_Q);
```

Объявите входные и выходные порты согласно таблице 2.1 и рисунку 2.1.

```
input                       iclk;
input                       irst;
input                       iena;
input signed        [13:0]  idata_I;
input signed        [13:0]  idata_Q;
output reg signed   [13:0]  odata_I;
output reg signed   [13:0]  odata_Q;
output reg                  oena;
output                  osop;
```

Добавьте три одиннадцати разрядных счетчика: *count, buffer_in* и *buffer_out*.

```
reg             [10:0]      count;
reg             [10:0]      buffer_in;
reg             [10:0]      buffer_out;
```

Далее добавьте четыре четырнадцати разрядных знаковых регистра длинной 1024 бита: buffer1_I, buffer2_I, buffer1_Q и buffer2_Q. Они будут служить для хранения отсчетов.

```
reg     signed [13:0]       buffer1_I       [1023:0];
reg     signed [13:0]       buffer2_I       [1023:0];
reg     signed [13:0]       buffer1_Q       [1023:0];
reg     signed [13:0]       buffer2_Q       [1023:0];
```

Добавьте 5 одноразрядных регистров: read_buff, write_buff, out, num_buff_read, num_buff_write.

```
reg             read_buff, write_buff, out, num_buff_read, buff_write;
```

Эти регистры будут флагами: первый регистр (read_buff) будет равен единице, когда необходимо будет считать буффер, второй (write_buff) – когда необходимо записать в буффер. Третий регистр (out) равен единице, когда необходимо выводить данные на выход со входа, минуя буффер. Четвертый (num_buff_read) и пятый (num_buf_write) регистры отвечают за номер буффера, который будет прочитан или в который будет записываться информация.

В блоке initial задать нулевые значения всех регистров, кроме регистра num_buff_read, ему задайте 1. Это нужно для того, чтобы первый символ, поступивший на вход, начал записываться в буффер.

```
initial
begin
        count<=5'b0;
        read_buff<=1'b0;
        write_buff<=1'b1;
        out<=1'b0;
        buffer_in<=11'b0;
        buffer_out<=11'b0;
        num_buff_write<=1'b0;
        num_buff_read<=1'b1;
        oena<=1'b0;
end
```

Далее необходимо сгенерировать сигнал о начале символа.

```
assign osop= count==11'b01111100001 ? 1'b1:1'b0;
```

Дальше реализуйте счетчик с асинхронным сбросом, счетом до 1056 по разрешающему сигналу oena.

```
always@(posedge iclk or posedge irst)
begin
```

```
        if (irst)
        begin
                count<=11'b0;
        end
        else if(oena)
        begin
                if (count==11'b10000011111)
                        count<=11'b0;
                else
                count<=count+1'b1;
        end
        else
        begin
                count<=11'b0;
        end
end
```

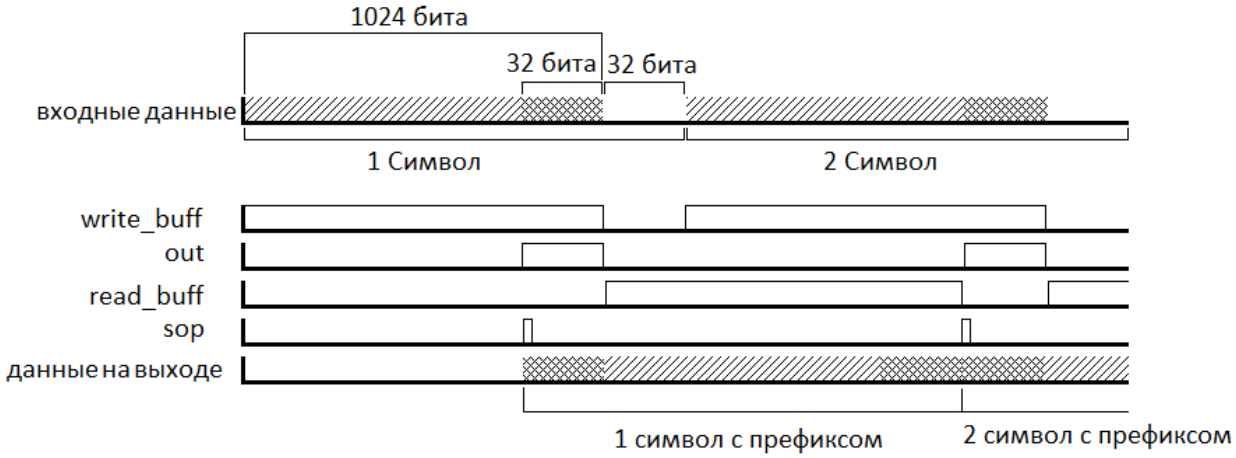


Рисунок 2.2 – Временная схема управляющих сигналов

Для реализации будет использоваться конструкция case. Т.е. в определенные моменты времени – флаги, отвечающие за запись, чтение или прямой вывод со входа минуя буфер, будут равны единице. Такая конструкция имеет задержку в один такт, т.е. если необходимо выполнить действие на 1000 такте счетчика, то в case нужно указывать 999 такт.

Принцип работы следующий: с нуля до 1024 такта будет происходить запись в один из буферов (в зависимости от num_buff_write). С 992 такта по 1024 такт будет вывод данных со входа. С 1024 такта по 992 происходит чтение данных из одного из двух буферов (в зависимости от num_buff_read) и вывод данных.

После case необходимо реализовать задержку для последнего символа.

```
always@(posedge iclk or posedge irst)
begin
	if (irst)
	begin
		oena<=1'b0;
		num_buff_write<=1'b0;
		num_buff_read<=1'b1;
		read_buff<=1'b0;
		write_buff<=1'b0;
		out<=1'b0;
	end
	else if(iena)
	begin
		oena<=1'b1;
		case (count)
		11'b10000011111:	begin
						write_buff<=1'b1;
					end
		11'b01111011111:	begin
						read_buff<=1'b0;
						out<=1'b1;
						num_buff_read<=~num_buff_read;
					end
		11'b01111111111:	begin
						write_buff<=1'b0;
						out<=1'b0;
						read_buff<=1'b1;
						num_buff_write<=~num_buff_write;
					end
		endcase
	end
	else if(buffer_out>=11'b00000001000 && buffer_out<=11'b01111111111)
		oena<=1'b1;
	else
	begin
		read_buff<=1'b0;
		oena<=1'b0;
	end
end
```

В следующих двух блоках реализуется запись в буффер и чтение из него, так же вывод данных со входа:

```
always@(posedge iclk)
begin
	if(irst)
		buffer_in<=11'b0;
	else if(write_buff)
	begin
		if(num_buff_write==1'b0)
		begin
			buffer1_I[buffer_in]<=idata_I;
			buffer1_Q[buffer_in]<=idata_Q;
			buffer_in<=buffer_in+1'b1;
		end
		else if(num_buff_write==1'b1)
		begin
			buffer2_I[buffer_in]<=idata_I;
			buffer2_Q[buffer_in]<=idata_Q;
			buffer_in<=buffer_in+1'b1;
		end
	end
	else buffer_in<=11'b0;
end
```

```
always@(posedge iclk)
begin
	if(irst)
		buffer_out<=11'b0;
	else if(read_buff)
	begin
		if(num_buff_read==1'b0)
		begin
			odata_I<=buffer1_I[buffer_out];
			odata_Q<=buffer1_Q[buffer_out];
			buffer_out<=buffer_out+1'b1;
		end
		else if(num_buff_read==1'b1)
		begin
			odata_I<=buffer2_I[buffer_out];
			odata_Q<=buffer2_Q[buffer_out];
			buffer_out<=buffer_out+1'b1;
		end
	end
	else if(out)
	begin
		odata_I<=idata_I;
		odata_Q<=idata_Q;
		buffer_out<=11'b0;
	end
	else
	begin
		odata_I<=14'b0;
		odata_Q<=14'b0;
		buffer_out<=11'b0;
	end
end
```

Добавьте оператор завершения модуля (endmodule).
Перед подключением блока создайте два 14 разрядных провода (norm_I и norm_Q).

```
wire signed [13:0]		norm_I;
wire signed [13:0]		norm_Q;
```

Подключите блок.

```
Prefix Prefix(
	.iclk		(iclk),
	.irst		(~irst),
	.iena		(norm_ena),
	.idata_I	(norm_I),
	.idata_Q	(norm_Q),
	.oena		(pref_ena),
	.osop		(sop_pref),
	.odata_I	(pref_I),
```

```
        .odata_Q        (pref_Q),
);
```

Скомпилируйте проект, откройте RTL Viewer и сравните схему подключения, со схемой на рисунке 2.3.

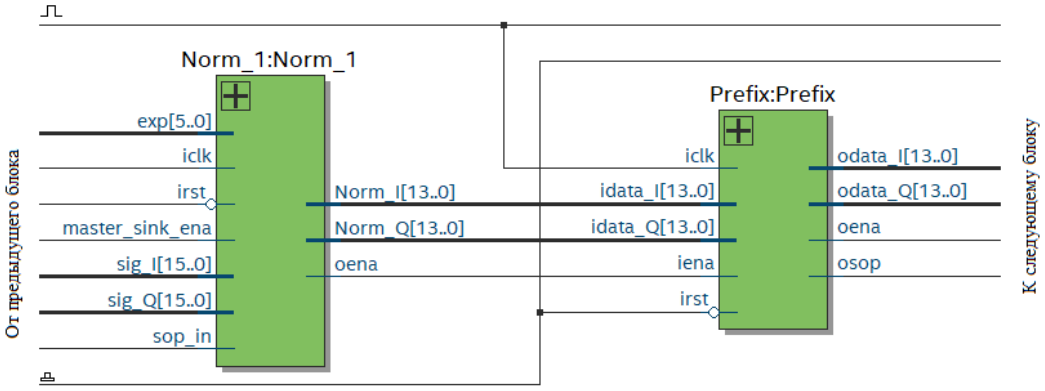


Рисунок 2.3 – Схема подключения

Если схема подключения верная, то загрузите прошивку на плату. Выведите данные с модуля циклического префикса в SignalTap (odata_I, odata_Q, osop, isop, idata_I, idata_Q). Установите первый маркер в момент, когда сигнал osop равен 1, второй маркер установите на расстоянии +1024 отчетов от первого маркера, третий маркер установите -993 отсчета от первого маркера. Расположение маркеров показано на рисунке 2.4.

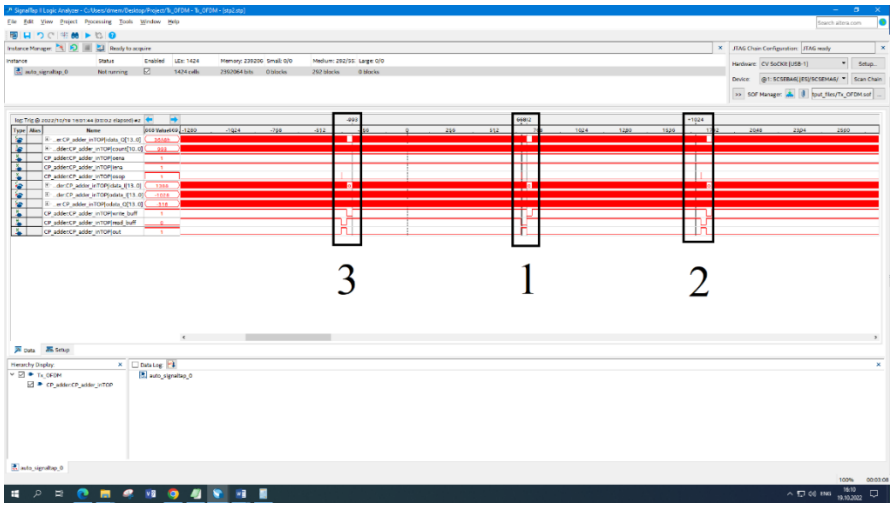


Рисунок 2.4 – Расположение маркеров

Проверьте, что первые 32 отсчета (odata_I и odata_Q), начинающиеся с первого и второго маркера, полностью совпадают. Начиная с 33 отсчета от первого маркера (odata_I и odata_Q) и отсчеты с третьего маркера (idata_I и idata_Q) совпадают.

Работа № 6

«Инверсия спектра. Интерполяции. Фильтр»

Целью работы является реализация блоков разворота спектра, интерполяции и синтез фильтра.

Задачи работы:

1) Реализация блока разворота спектра и его подключение к основному проекту;
2) Реализация блока интерполяции и его подключение к проекту;
3) Синтез фильтра в MATHLAB и его подключение к проекту;
4) Проверка созвездия и спектра сигнала.

1. Теоретическая часть

Интерполяция

Интерполяция – в математической области численного анализа, это тип оценки, метод получения новых точек на основе диапазона дискретного набора известных точек данных.

Существует множество методов интерполяции, отличающиеся такими свойствами, как: точность, стоимость, количество необходимых точек данных и гладкость результирующей интерполяционной функции.

Кусочно-постоянная интерполяция – самый простой метод интерполяции, находится ближайшее значение данных и присваивается ему то же значение.

Линейная интерполяция – один из простейших методов интерполяции, но имеет большую погрешность, которая пропорциональна квадрату расстояния между точками данных. Для данной интерполяции требуется две точки данных, а интерполяция задается как:

$$y = y_1 + (y_2 + y_2)\frac{x - x_1}{x_2 - x_1}. \quad (1.1)$$

где (x_1, y_1) – координаты первой точки данных;

(x_2, y_2) – координаты второй точки данных;

(x, y) – координаты точки, лежащей между первой и второй точкой данных.

Полиномиальная интерполяция – является обобщением линейной интерполяции, но линейная функция заменяется многочленом более высокой степени. Вычисление интерполирующего многочлена требует больших вычислительных затрат, однако имеет меньше ошибку, по сравнению с линейной интерполяцией.

Сплайн-интерполяция использует полиномы низкой степени в каждом из интервалов между точками данных и выбирает части полинома так, чтобы они плавно подходили друг к другу.

В области цифровой обработки сигналов интерполяция относится к процессу преобразования дискретизированного цифрового сигнала в дискретизированный цифровой сигнал с более высокой частотой дискретизации с использованием различных методов фильтрации.

Фильтр

Фильтр – это устройство для выделения желательных компонентов спектра электрического сигнала.

Существует два основных вида фильтра: аналоговый и цифровой.

Главное отличие данных видов в том, что цифровой фильтр представляет из себя систему, которая выполняет математические операции над дискретизированным сигналом. Аналоговый фильтр обычно представляет собой электронную схему, работающую с аналоговыми (непрерывными) сигналами.

Цифровые фильтры имеют следующие параметры:

- Передаточная функция;

- АЧХ и ФЧХ;
- Импульсная характеристика;
- Переходная функция, групповая и фазовая задержки и т.д.;
- Архитектура, эффекты квантования.

С точки зрения частотной избирательности, существует 6 типов фильтров, показанных на рисунке 1.1.

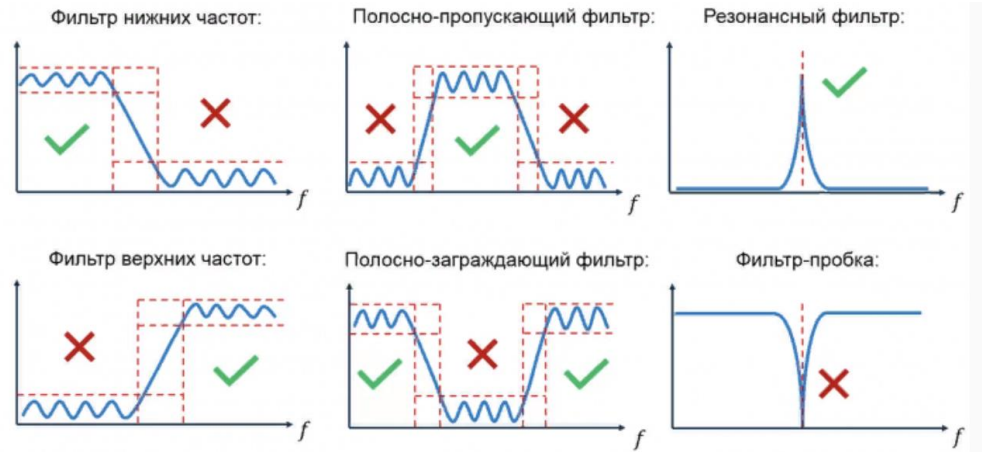


Рисунок 1.1 – Типы частотно-избирательных фильтров

Требования к частотной избирательности фильтра, в зависимости от накладываемой на него задачи, можно записать в виде спецификации к его частотным характеристикам.

У всех фильтров можно выделить три основные полосы: полосу пропускания, полосу перехода, полосу заграждения. На рисунке 1.2 представлена спецификация для фильтра нижних частот (ФНЧ).

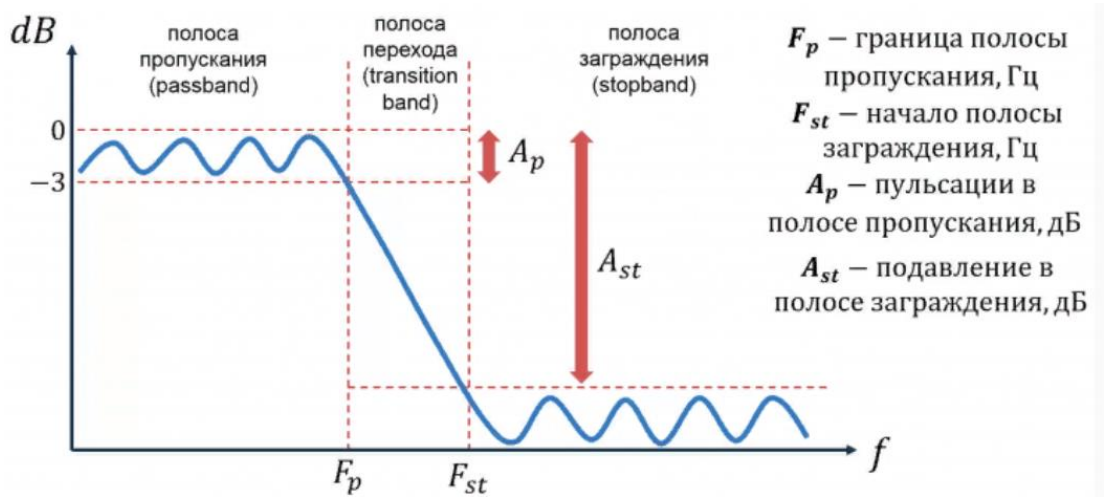


Рисунок 1.2 – Спецификация ФНЧ

Полоса пропускания фильтра – диапазон частот, в котором сигнал проходит без затухания (в идеале). Границей полосы пропускания для ФНЧ является частота F_p. Она отсчитывается по уровню -3 дБ от максимума.

Полоса заграждения – диапазон в частот, в котором происходит ослабление сигнала не менее чем на A_{st} дБ. Начинается полоса заграждения с частоты F_{st}.

Между этими двумя полосами находится переходная полоса, в которой постепенно происходит увеличение затухания. Ширина данной полосы зависит от порядка фильтра – чем больше порядок, тем уже полоса. Порядок фильтра – это количество коэффициентов

фильтров, оно напрямую связано с количеством умножителей, необходимых для реализации.

С точки зрения импульсной характеристики существует 2 типа фильтров: с конечной импульсной характеристикой (КИХ) и с бесконечной импульсной характеристикой (БИХ).

КИХ фильтр – нерекурсивный, это значит, что для вычисления значения на выходе фильтра используется только текущее и задержанные значения входа. Схема такого фильтра не имеет обратных связей (рисунок 1.3).

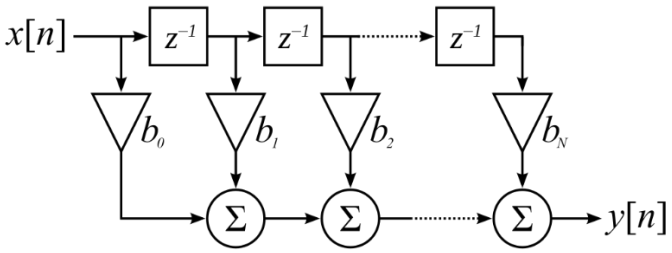


Рисунок 1.3 – Схема КИХ фильтра

Данный тип фильтров имеет следующие преимущества:

- Могут иметь линейную фазу, что сильно упрощает компенсацию фазового набега;
- всегда устойчивы;

Основной их недостаток – КИХ дороже в реализации, чем БИХ фильтры, со схожей АЧХ.

БИХ фильтры – рекурсивные, для вычисления значения на выходе фильтра используются как значения входа, так и задержанные значения выхода. Схема представлена на рисунке 1.4.

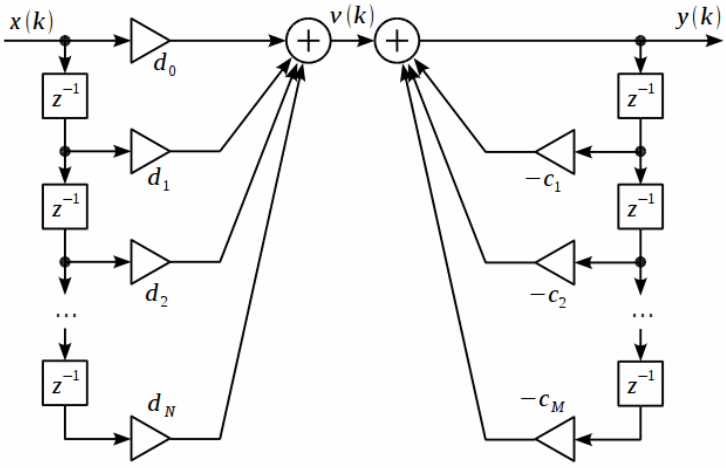


Рисунок 1.4 – Схема БИХ фильтра

К преимуществам данного фильтра относится:

- Относительная простота в реализации, по сравнению с КИХ-фильтрами;
- относительная простота синтеза на основе аналоговых прототипов.

В качестве недостатков:

- Могут быть неустойчивыми;
- не обладают линейной фазой;
- нет возможности сформировать произвольную АЧХ и ФЧХ.

Синтезируются БИХ-фильтры при помощи преобразования непрерывной передаточной характеристики аналогового прототипа в дискретную характеристику цифрового фильтра.

2. Практическая часть

2.1 FFT Shifter

На данном этапе реализовывается блок разворота спектра. На рисунке 2.1 и в таблице 2.1 показаны блок схема модуля и описание входных и выходных портов.

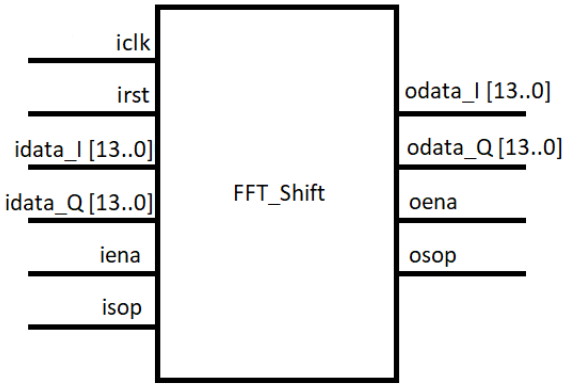


Рисунок 2.1 – Блок схема модуля разворота спектра

2.1.1 Добавьте в проект файл Verilog HDL, назовите его «FFT_Shift».
2.1.2 Обозначьте следующие входы и выходы.

Таблица 2.1 – Описание входов и выходов «FFT Shift»

Порты	Разрядность	Тип	I/O	Описание портов
irst	-	wire	input	Тактирование 10 МГц
iclk	-	wire	input	Сигнал сброса
isop	-	wire	input	Индикатор начала символа
iena	-	wire	input	Сигнал разрешения
idata_I	[13..0]	wire	input	Входные данные
idata_Q	[13..0]	wire	input	Входные данные
odata_I	[13..0]	reg signed	output	Выходные данные
odata_Q	[13..0]	reg signed	output	Выходные данные
oena	-	reg	output	Сигнал разрешения
osop	-	reg	output	Индикатор начала символа

2.1.3 Объявите одноразрядный регистр count для счетчика.

```
reg                                    count;
```

2.1.4 В блоке initial задайте нулевые начальные состояния для выходных портов и регистра count.

```
initial
begin
        oena<=1'b0;
```

```
        count<=1'b0;
        odata_I<=14'b0;
        odata_Q<=14'b0;
end
```

2.1.5 Далее необходимо реализовать счет счетчика в блоке «always».

```
always@(posedge iclk or posedge irst)
begin
        if(irst)
                count<=1'b0;
        else if(oena)
        count<=count+1'b1;
end
```

2.1.6 В следующем блоке «always» необходимо реализовать инвертирование каждого второго входного отсчета.

```
always@(posedge iclk or posedge irst)
begin
if(irst)
        begin
        odata_I<=14'b0;
        odata_Q<=14'b0;
        end
else if(iena)
begin
        if(count==1'b1)
                begin
                odata_I<=-idata_I;
                odata_Q<=-idata_Q;
                end
        else if(count==1'b0)
                begin
                odata_I<=idata_I;
                odata_Q<=idata_Q;
                end
end
else
begin
        odata_I<=14'b0;
        odata_Q<=14'b0;
end
end
```

2.1.7 В последнем «always» блоке необходимо реализовать сигналы «osop» и «oena». По отношению ко входным сигналам «isop» и «iena», сигналы «osop» и «oena» должны иметь задержку в один такт.

```
always@(posedge iclk or posedge irst)
```

```
begin
        if(irst)
        begin
                osop<=1'b0;
                oena<=1'b0;
        end
        else
        begin
                osop<=isop;
                oena<=iena;
        end
end
```

2.1.8 Следующим шагом необходимо подключить блок разворота спектра к основному проекту.

```
FFT_Shift FFT_Shift(
        .iclk(iclk),
        .irst(~locked),
        .idata_I(pref_I),
        .idata_Q(pref_Q),
        .iena(pref_ena),
        .isop(sop_pref),
        .osop(sop_invert),
        .odata_I(invert_I),
        .odata_Q(invert_Q),
        .oena(invert_ena));
```

2.1.9 После проведите компиляцию проекта и проверьте подключение блока в «RTL Viewer». Схема подключения показана на рисунке 2.2.

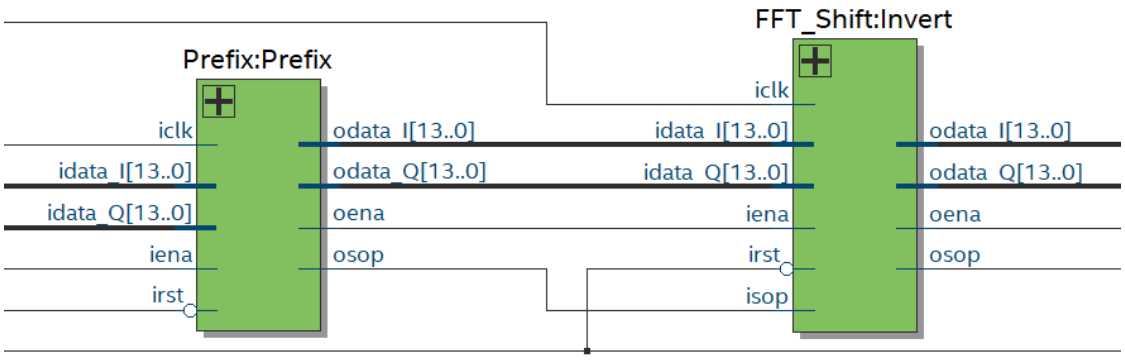


Рисунок 2.2 – Схема подключения

2.1.10 Возьмите данные с выхода блока разворота спектра по аналогии с предыдущей работой, и постройте спектр сигнала в MATLAB. Результат построения изображен на рисунке 2.3.

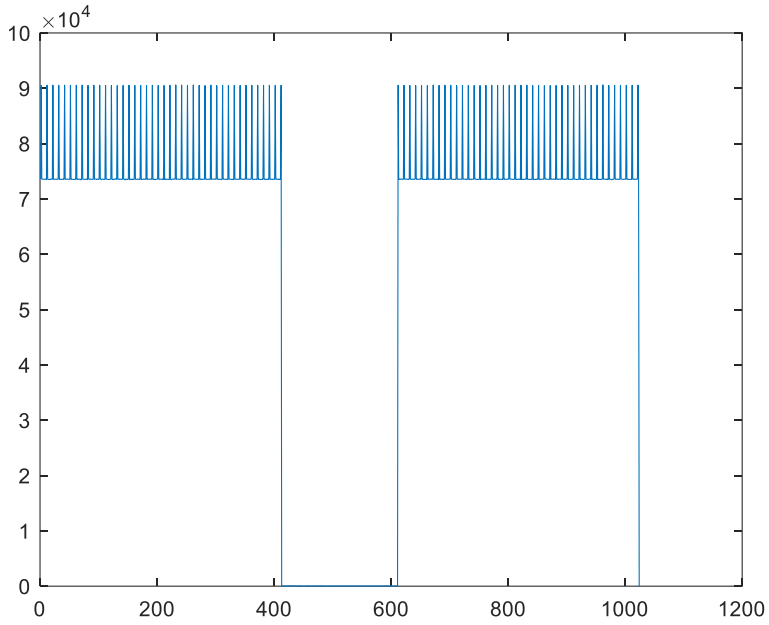


Рисунок 2.3 – Спектр сигнала

2.2 Интерполятор

Следующим этапом необходимо реализовать модуль интерполяции (интерполятор). Блок схема данного модуля и описание портов представлены на рисунках 2.4 и таблице 2.2.

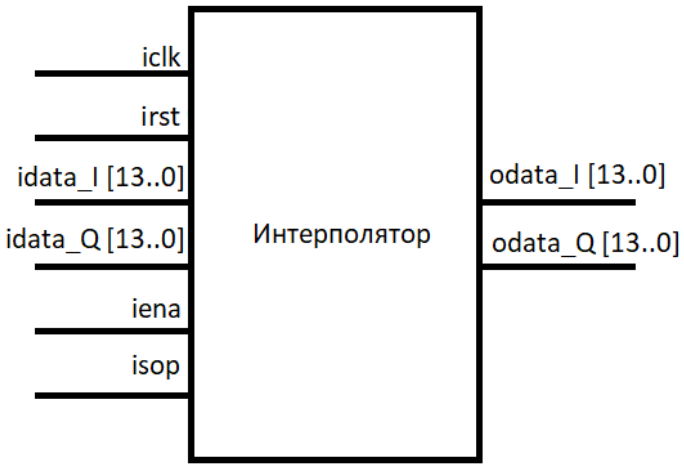


Рисунок 2.4 – Блок схема интерполятора

Таблица 2.2 – Описание входов и выходов интерполятора

Порты	Разрядность	Тип	I/O	Описание портов
irst	-	wire	input	Сигнал сброса
iclk	-	wire	input	Тактирование 40 МГц
isop	-	wire	input	Индикатор начала символа
iena	-	wire	input	Сигнал разрешения
idata_I	[13..0]	wire signed	input	Входные данные

Окончание таблицы 2.2

idata_Q	[13..0]	wire signed	input	Входные данные
osop	-	wire	output	Индикатор начала символа
odata_I	[13..0]	reg signed	output	Выходные данные
odata_Q	[13..0]	reg signed	output	Выходные данные

2.1.11 Объявите входные пины и двухразрядный регистр “*counter”* для счетчика.

```
input iclk;
input irst;
input iena;
input isop;
input signed    [13:0]          idata_I;
input signed    [13:0]          idata_Q;

output reg signed [13:0]        odata_I;
output reg signed [13:0]        odata_Q;
output reg                                      oena;
output reg                                      osop;

reg                             [1:0]           counter;
```

2.1.12 В блоке «initial» задайте нулевое начальное состояние для регистра «counter».

```
initial
begin
        counter<=2'b0;
end
```

2.1.13 Далее в блоке «always» реализуйте счет счетчика по тактовому сигналу «iclk» и разрешающему сигналу «iena» с асинхронным сбросом по сигналу «irst».

```
always @(posedge iclk or posedge irst)
begin
        if(irst)
                counter<=2'b0;
        else if(iena)
                ounter<=counter+1'b1;
end
```

2.1.14 Далее необходимо реализовать передискретизацию сигнала в 4 раза, для этого каждый раз, когда счетчик будет равен 0, значение выходных данных, будет равно значению входных, иначе значение выходных данных будет равно 0. В следующем блоке нули будут сглажены с помощью ФНЧ.

```
always@(posedge iclk or posedge irst)
begin
```

```
        if(irst)
        begin
                odata_I<=14'b0;
                odata_Q<=14'b0;
                osop<=1'b0;
        end
        else if(counter==2'b0 && iena==1'b1)
        begin
                odata_I<=idata_I;
                odata_Q<=idata_Q;
                osop<=isop;
        end
        else
        begin
                odata_I<=14'b0;
                odata_Q<=14'b0;
                osop<=1'b0;
        end
end
```

2.1.15 Допишите в конец файла «endmodule». Обратите внимание, что в этом и во всех последующих блоках нужно подавать тактовый сигнал 40 МГц, т.е. тактовый сигнал, генерирующийся с выхода «outclc_1» модуля «clc».

```
Interpol Interpol(
        .iclk(iclk4),
        .irst(~locked),
        .idata_I(invert_I),
        .idata_Q(invert_Q),
        .iena(invert_ena),
        .isop(sop_invert),
        .odata_I(I_4x),
        .odata_Q(Q_4x)
);
```

2.1.16 После подключения скомпилируйте проект и проверьте подключение блока в «RTL Viewer» со схемой подключения на рисунке 2.5.

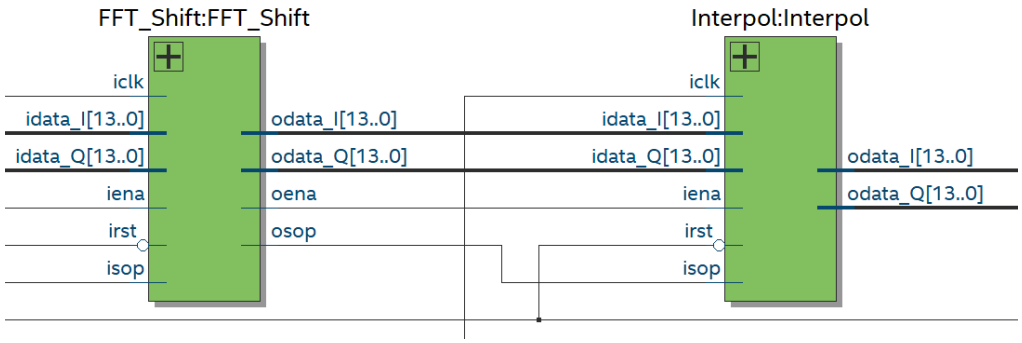


Рисунок 2.5 – Схема подключения интерполятора

2.1.17 Также при снятии данных через «SignalTap» нужно изменить тактовый сигнал с 10 МГц, на 40 МГц. Для этого перейдите во вкладку «Setup» и справа, в поле «Clock», установите «clc:clc|outclk_1».

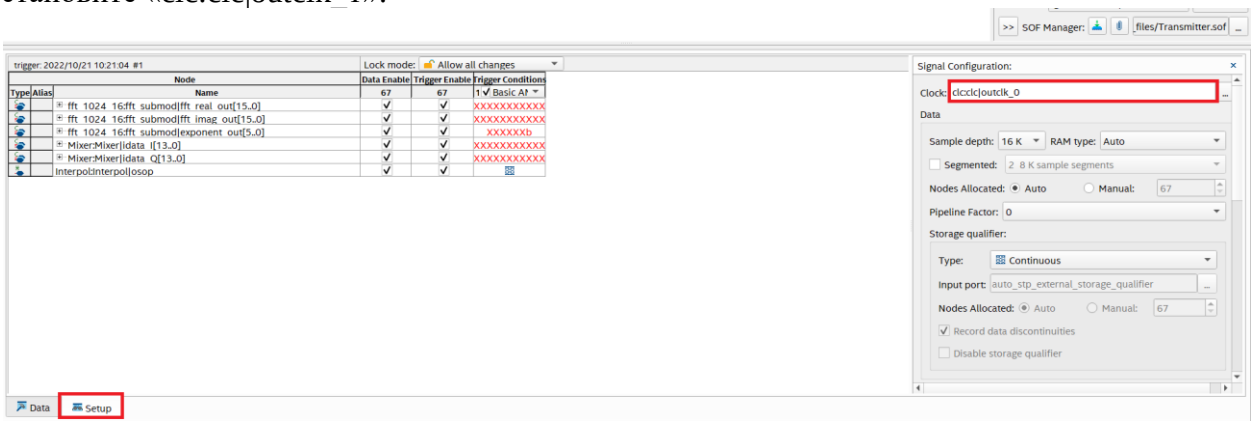


Рисунок 2.6 – Изменение частоты Signal Tap

2.3 Фильтр

Следующем шагом необходимо синтезировать фильтр со следующими параметрами, представленными в таблице 2.3.

Таблица 2.3 – Параметры фильтра

Параметр	Значение
Порядок фильтра	60
Тип фильтра	*FIR Hamming*
Частота сигнала	40 МГц
Частота среза	5 МГц

Блок схема представлена на рисунке 2.7. В таблице 2.4 представлено описание входных и выходных пинов фильтра.

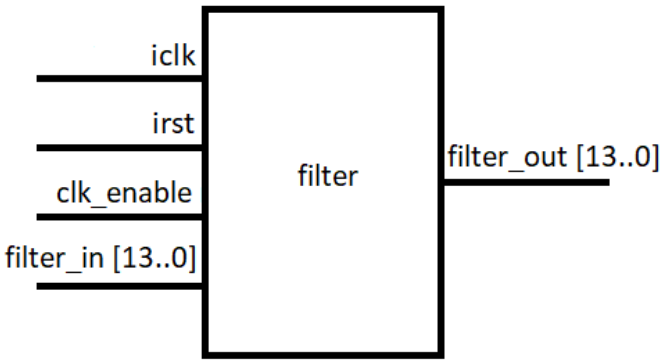


Рисунок 2.7 – Блок схема фильтра

Таблица 2.4 – Описание входов и выходов фильтра

Порты	Разрядность	Тип	I/O	Описание портов
iclk	-	Wire	input	Тактирование 40 МГц
irst	-	Wire	input	Сигнал сброса

Окончание таблицы 2.4

clk_enable	-	Wire	input	Сигнал разрешения
filter_in	[13..0]	Wire signed	input	Входные данные
filter_out	[13..0]	Wire signed	output	Выходные данные

Для создания фильтра будем использовать Filter Design в MATLAB.

2.1.18 Для начала откройте MATLAB, нажмите на кладку «APPS» и в списке приложений найдите «Filter Design».

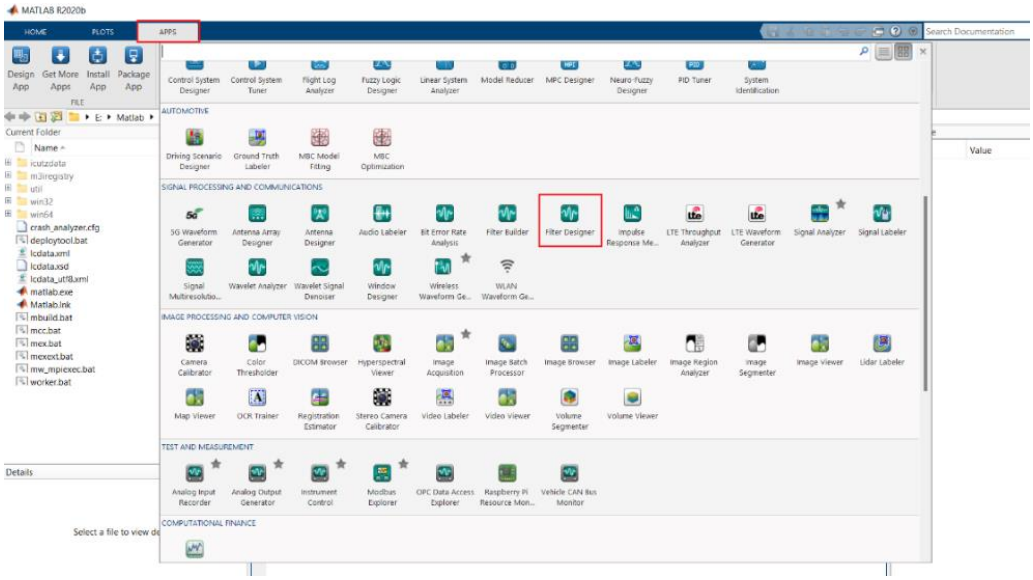


Рисунок 2.8 – Иконка приложения «Filter Designer»

После запуска у вас откроется окно как на рисунке 2.9.

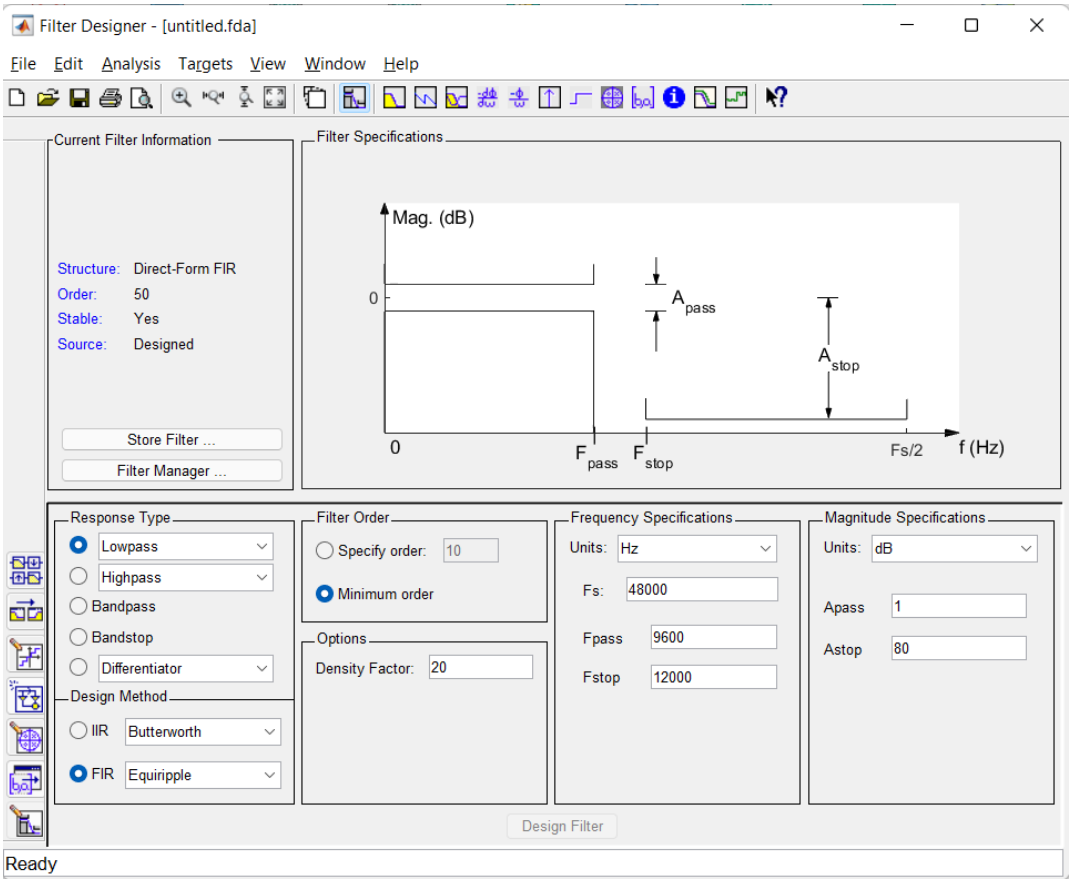


Рисунок 2.9 – Главное окно «Filter Designed»

2.1.19 Установите такие же параметры как на рисунке 2.10 и после нажмите кнопку «Design Filter».

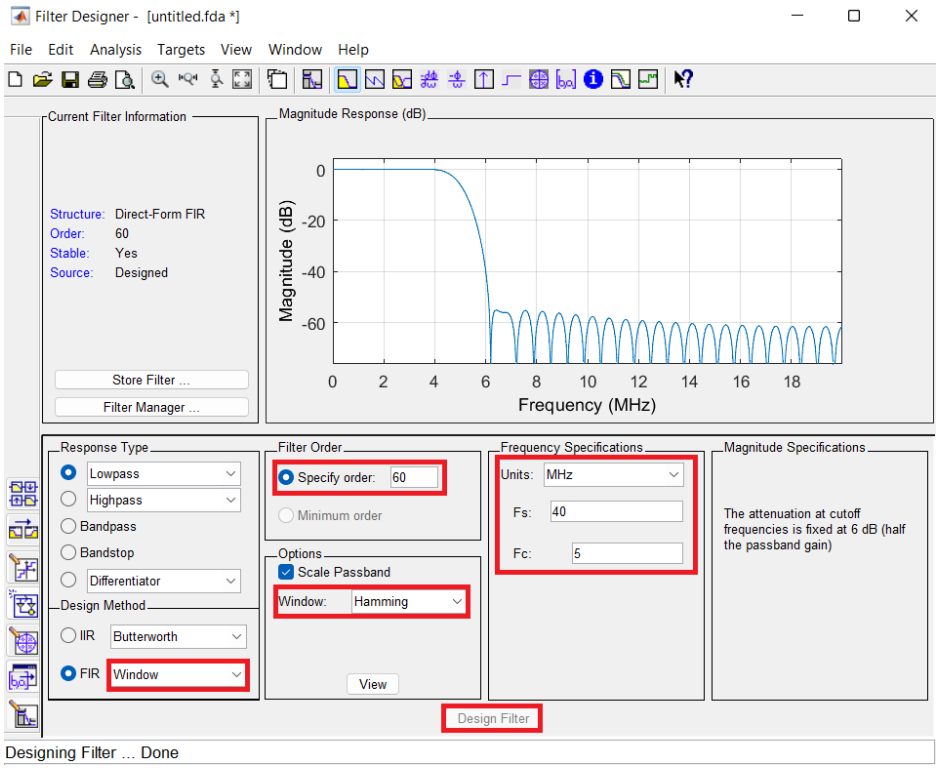


Рисунок 2.10 – Настройка фильтра

2.1.20 Перейдите во вкладку и установите такие же параметры как на рисунке 2.11.

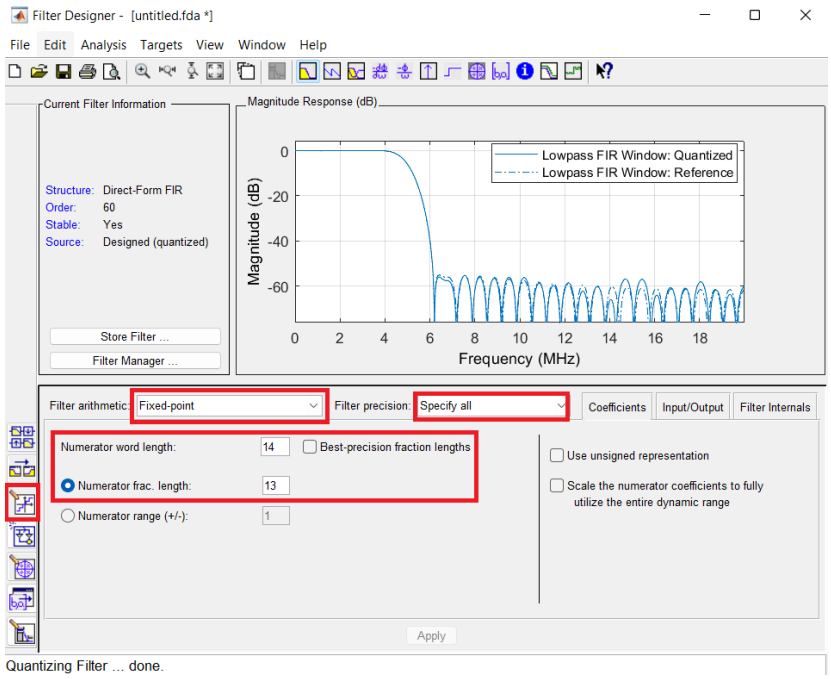


Рисунок 2.11 – Нормировка коэффициентов фильтра

2.1.21 Далее перейдите во вкладку «Input/Output» и установить такие же параметры как на рисунке 2.12. После нажмите кнопку Apply.

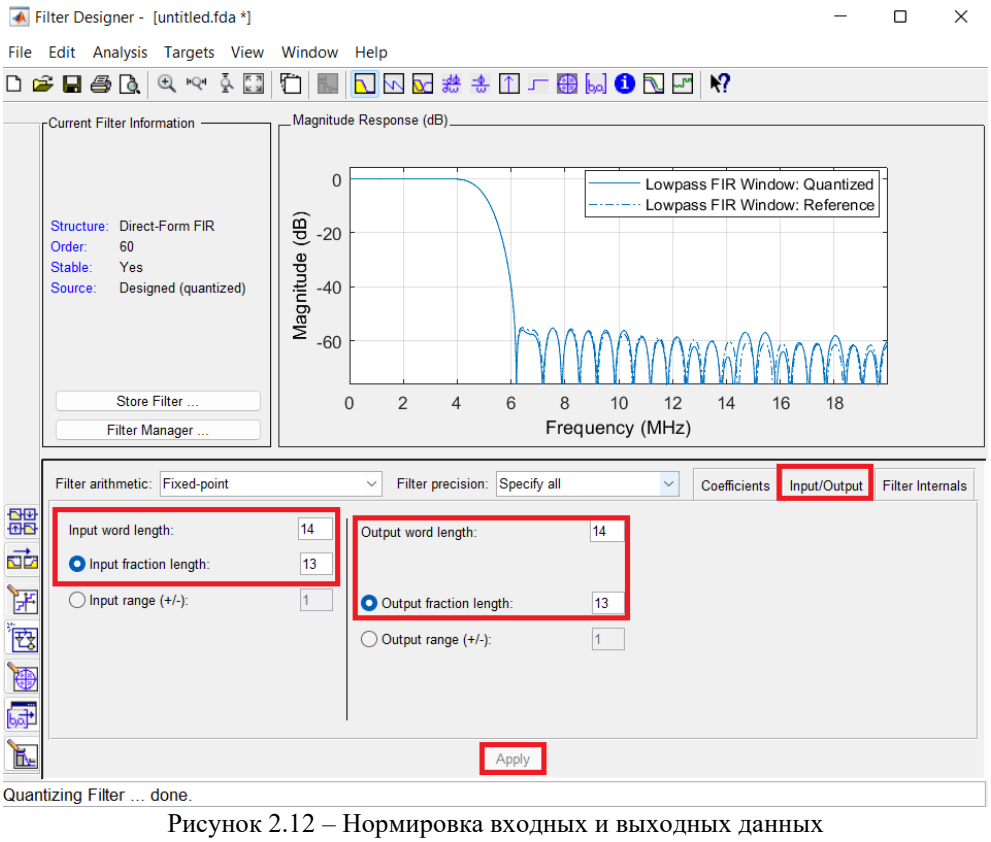


Рисунок 2.12 – Нормировка входных и выходных данных

2.1.22 После необходимо сгенерировать код. Для этого нажмите «Targets» => «Generate HDL», как показано на рисунке 2.13.

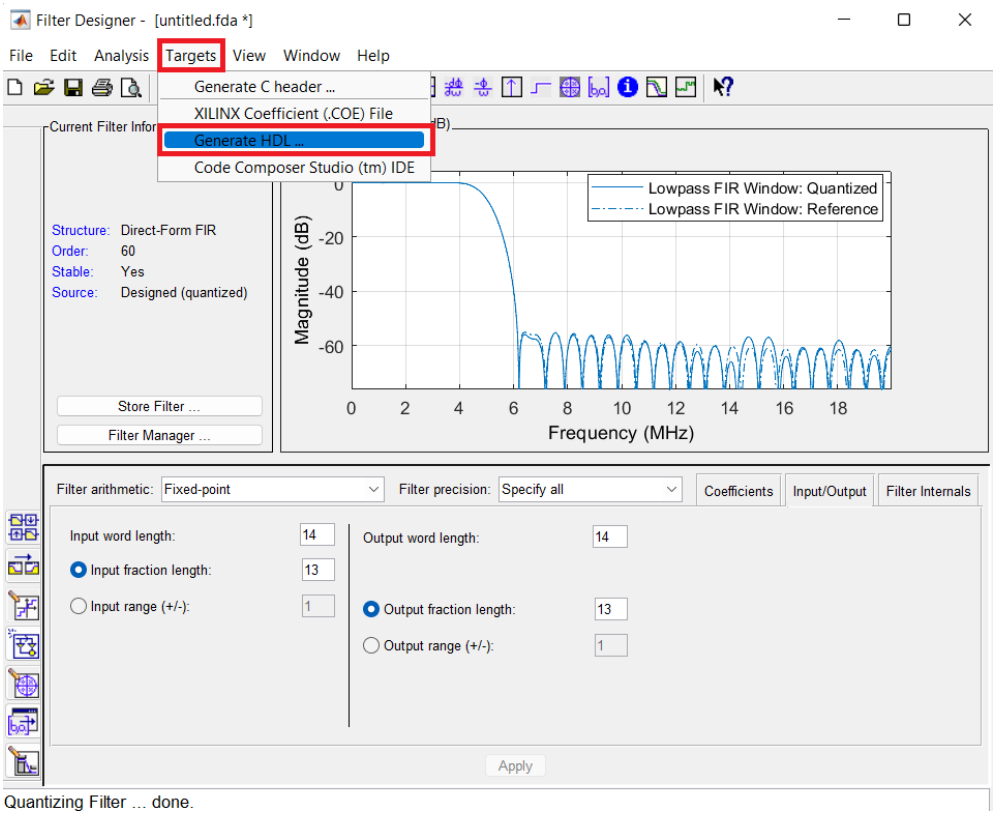


Рисунок 2.13 – Запуск преобразования в HDL

2.1.23 В открывшемся окне установите такие же параметры как на рисунках 2.14 – 2.15, кроме параметра «Folder», здесь укажите папку с вашим проектом после нажмите кнопу Generate.

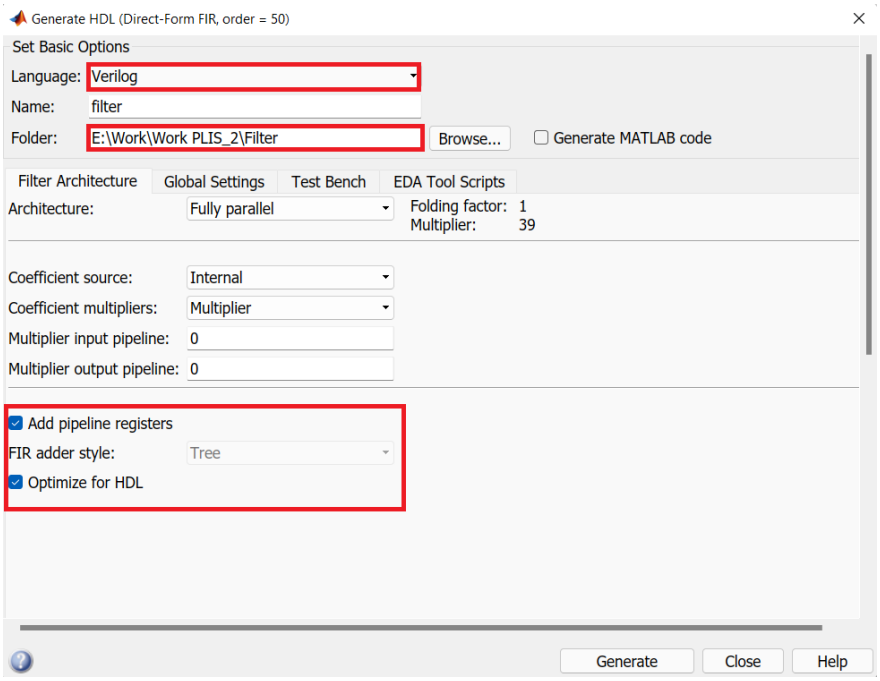


Рисунок 2.14 – Настройка преобразователя HDL

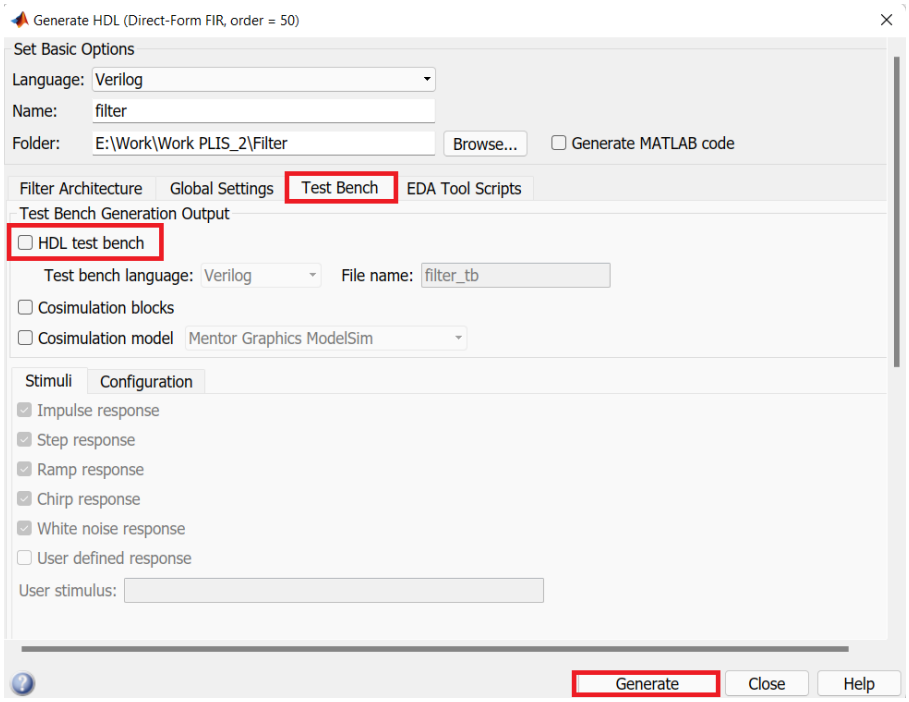


Рисунок 2.15 – Отключение «testbench»

2.1.24 Далее добавьте сгенерированный файл в проект. Следующим шагом необходимо подключить его. Для *I* и *Q* будем использовать один и тот же фильтр, но под разными внутренними названиями. В файле верхнего уровня подключение будет выглядеть следующим образом:

```
filter filter_block_I(
        .clk(iclk4),
        .reset(~locked),
        .clk_enable(1'b1),
        .filter_in(I_4x),
        .filter_out(filter_I)
);
filter filter_block_Q(
        .clk(iclk4),
        .reset(~locked),
        .clk_enable(1'b1),
        .filter_in(Q_4x),
        .filter_out(filter_Q)
);
```

2.1.25 После скомпилируйте проект и в «RTL Veiwer» проверьте верность соединения блоков. На рисунке 4.10 представлена верная схема соединения блоков.

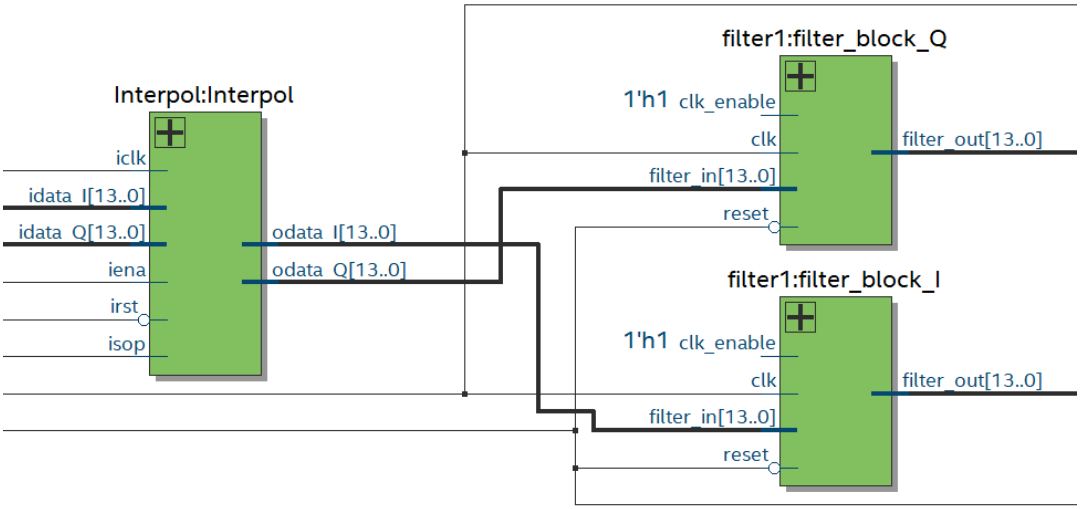


Рисунок 2.16 – Схема подключения фильтров.

В «RTL Viewer» («Tools»=> «Netlist viewers»=> «RTL Viewer») проверьте соединения блоков.

Теперь загрузите прошивку на плату и снимите данные для проверки. Чтобы определить начало символа используйте sop сигнал с выхода интерполятора.

Далее с помощью MATLAB децимируйте данные в 4 раза, проверьте созвездие и спектр сигнал. Код для выполнения этих операций приведен ниже.

```
clc
clear all
close all

Data=dlmread('1.txt');
I=Data(1:1:length(Data),1);
Q=Data(1:1:length(Data),2);
Sig=(complex(I,Q));

for i=1:170
        plot(real(fft(Sig(i:4:i+4095))),imag(fft(Sig(i:4:i+4095))),'.');
        pause(0.4)
end
figure

plot(abs(fft(Sig(i:4:i+4095))));
```

В результате должен построиться развернутый спектр сигнала, как на риснуке 2.17, и его созвездие, как на рисунке 2.18.

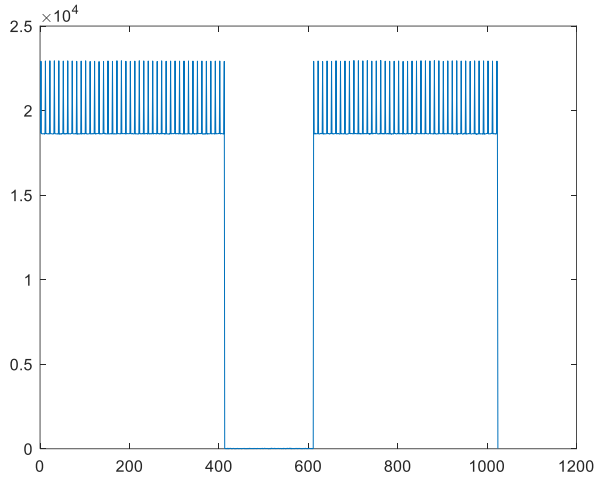


Рисунок 2.17 – Созвездие сигнала

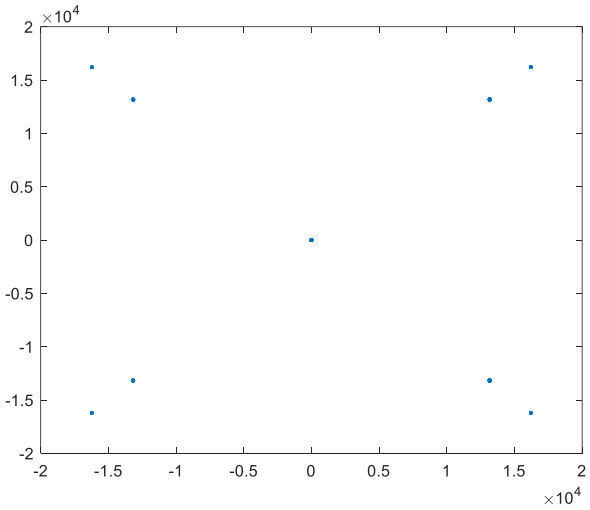


Рисунок 2.18 – Спектр сигнала

Работа № 7

«Преобразование частоты»

Целью работы является реализация смесителя сигнала и проверка спектра генерируемого сигнала с помощью осциллографа.

Задачи работы:

1) Реализация смесителя сигнала;
2) Подключение АЦ/ЦА преобразователя;
3) Проверка спектра на осциллографе.

1. Теоретическая часть

Квадратурный смеситель

Смеситель позволяет объединить I и Q составляющие сигнала в один, посредством умножения на синус и синус сдвинутый по фазе на 90 градусов. На рисунке 1.1 представлена структурная схема квадратурного смесителя.

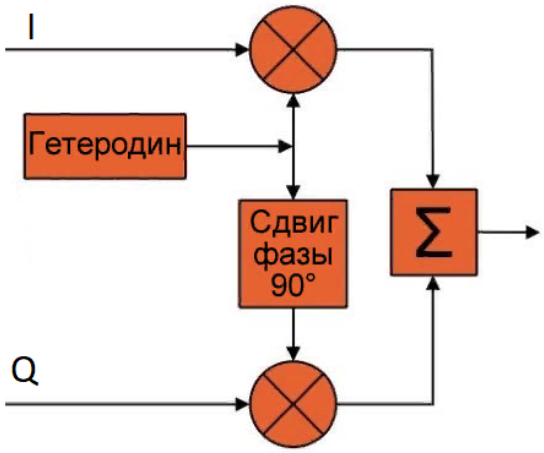


Рисунок 1.1 – Схема квадратурного смесителя

На вход данный смеситель имеет два сигнала – синфазный (I) и квадратурный (Q). Гетеродин генерирует несущую синусоиду. Сам сигнал гетеродина становится несущей I, а для создания несущей Q применяется фазовый сдвиг на 90°. Несущие I и Q умножаются на потоки данных I и Q, и два сигнала, полученные в результате этих умножений, суммируются для получения сигнала.

ЦАП

ЦАП преобразует поток бинарного кода в напряжение или ток. Можно описать это преобразование как функцию напряжения, которое определяется кодом. Чем выше или ниже значение в коде, тем выше или ниже получаем на выходе напряжение. Зависимость уровня напряжения или тока от кода в литературе называется монотонностью.

Но все немного сложнее. Так как ЦАП широко распространенное в электронике устройство, оно обладает множеством характеристик. Основные, значимые для нас, это: разрядность – количество бит во входном коде; частота дискретизации – максимальная частота, на которой ЦАП может работать корректно.

2. Практическая часть

Первым этапом необходимо реализовать смеситель квадратур. Блок схема представлена на рисунке 2.1. В таблице 2.1 представлено описание входных и выходных портов.

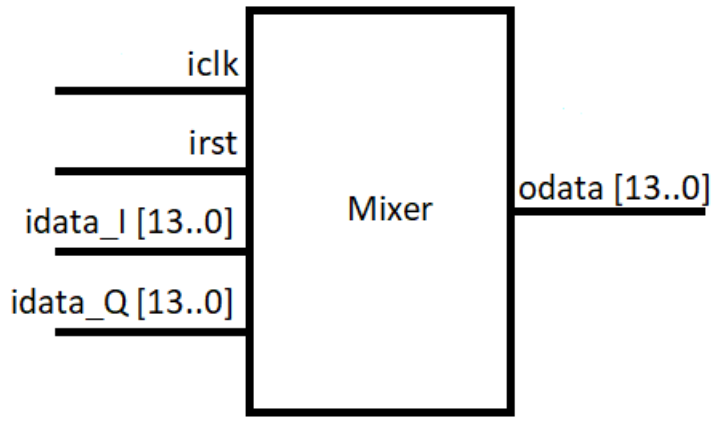


Рисунок 2.1 – Блок схема смесителя квадратур

Таблица 2.1 – Описание входов и выходов смесителя

Порты	Разрядность	Тип	I/O	Описание портов
iclk	-	Wire	Input	Тактирование 40 МГц
irst	-	Wire	Input	Сигнал сброса
idata_I	[13..0]	Wire signde	Input	Входные данные
idata_Q	[13..0]	Wire signed	Input	Входные данные
odata	[13..0]	Reg signed	Output	Выходные данные

Зайдите в проект, создайте файл Verilog HDL и назовите его «Mixer». Объявите 5 регистров: 2 знаковых 28 разрядных регистра (buff_I и buff_Q), 2 знаковых 14 разрядных регистра с длинной равной 4 («koef_cosinus» и «koef_sinus») и один двухразрядный регистр («count»).

```
input irst;
input iclk;
input signed [13:0]    idata_I;
input signed [13:0]    idata_Q;
output reg signed[13:0]        odata;
reg signed [28:0]              buff_I;
reg signed [28:0]              buff_Q;
reg signed [13:0]      koef_cosinus [3:0];
reg signed [13:0]      koef_sinus [3:0];
reg [1:0]       count;
```

Первые два регистра будут сложить буфером для результата умножения входных данных и отсчетов гармонического сигнала, которые будут записаны в регистры. Третий и четвертые регистры также будут выступать в качестве буфера и буду хранить в себе по 4 значения синуса и косинуса. Последний регистр будет выступать в роли счетчика.

В блоке «initial» задайте значения для косинуса и синуса, а также начальное нулевое состояние счетчика:

```
initial
begin
        koef_cosinus[0]=14'd101;
```

```
        koef_cosinus[1]=-14'd16;
        koef_cosinus[2]=-14'd101;
        koef_cosinus[3]=14'd16;

        koef_sinus[0]=14'd16;
        koef_sinus[1]=14'd101;
        koef_sinus[2]=-14'd16;
        koef_sinus[3]=-14'd101;

        count<=2'b0;
    end
```

В блоке «always» необходимо реализовать перемножение входных данных с отчетами гармонического сигнала, счет счетчика и суммирование результатов после перемножения.

```
    always@(posedge iclk or posedge irst)
    begin
        if(irst)
        begin
            buff_I<=29'b0;
            buff_Q<=29'b0;
            odata<=14'b0;
            count<=2'b0;
        end
        else
        begin
            buff_I=(idata_I*koef_cosinus[counter]);
            buff_Q=(idata_Q*koef_sinus[counter]);
            odata={buff_I[28], buff_I[19:7]}+{buff_Q[28], buff_Q[19:7]};
            count<=count+1'b1;
        end
    end
```

В конце добавьте «endmodule» и сохраните. В модуле верхнего уровня пропишите подключение модуля.

```
    Mixer Mixer(
        .irst(~locked),
        .iclk(iclk4),
        .idata_I(filter_I),
        .idata_Q(filter_Q),
        .odata(Output_signal)
    );
```

После проведите компиляцию проекта и проверьте подключение в RTL Viewer со схемой на рисунке 2.2.

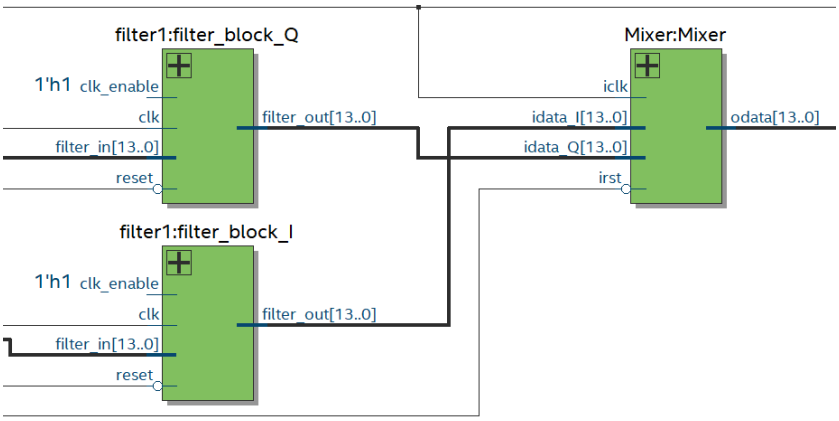


Рисунок 2.2 – Схема подключения

Следующим шагом необходимо получившийся сигнал подать на АЦ/ЦА преобразователь.

Для этого в файле верхнего уровня создайте выходной 14 разрядный шину «out_A», («Wire») через который мы будем подавать сигнал.

```
output wire signed [13:0]          out_A;
```

Далее через «assign» присвоим ему значение с выхода мультиплексора, но перед этим необходимо инвертировать старший разряд и выполнить конкатенацию с оставшейся частью выходной шины.

```
assign out_A={~Output_signal[13], Output_signal[12:0]};
```

Далее для работы ЦАПа нам необходимо создать еще 2 выходных сигнала: тактовый («oclk_A») и разрешающий («DA_WRTA»). Тактовый и разрешающий сигнал – копии тактового сигнала 40 МГц, поэтому через присвоение нужно подать им тактовый сигнал 40 МГц с выхода *PLL* блока.

```
output wire                        DA_WRTA;
output wire                        oclk_A;
assign oclk_A=iclk4;
assign DA_WRTA=iclk4;
```

В файле верхнего уровня создайте еще один выходной регистр («mod») и просто присвойте ему 1. Этот регистр отвечает за тип работы ЦАПа.

Выполните компиляцию после чего зайдите в «Pin planner».

Рисунок 2.3 – Иконка «Pin planner»

Внизу открывшегося окна будет таблица с входными и выходными портами. Заполните столбик «Location» согласно рисунку 2.4 для платы SoCKit, и согласно рисунку 2.5 для платы DE10.

Node Name	Direction	Location	I/O Bank	VREF Group
DA_WRTA	Output	PIN_C3	8A	B8A_N0
iclk_input	Input	PIN_Y26	5B	B5B_N0
irst	Input	PIN_AE12	3A	B3A_N0
mod	Output	PIN_B3	8A	B8A_N0
oclk_A	Output	PIN_E6	8A	B8A_N0
out_A[13]	Output	PIN_D4	8A	B8A_N0
out_A[12]	Output	PIN_E4	8A	B8A_N0
out_A[11]	Output	PIN_E2	8A	B8A_N0
out_A[10]	Output	PIN_E3	8A	B8A_N0
out_A[9]	Output	PIN_C4	8A	B8A_N0
out_A[8]	Output	PIN_D5	8A	B8A_N0
out_A[7]	Output	PIN_C5	8A	B8A_N0
out_A[6]	Output	PIN_D6	8A	B8A_N0
out_A[5]	Output	PIN_F6	8A	B8A_N0
out_A[4]	Output	PIN_G7	8A	B8A_N0
out_A[3]	Output	PIN_D7	8A	B8A_N0
out_A[2]	Output	PIN_E8	8A	B8A_N0
out_A[1]	Output	PIN_A8	8A	B8A_N0
out_A[0]	Output	PIN_A9	8A	B8A_N0

Рисунок 2.4 – Таблица пинов для платы SoCKit

iclk1	Output	PIN_E6	8A	B8A_N0
iclk2	Output	PIN_C3	8A	B8A_N0
iclk_input	Input	PIN_AF14	3B	B3B_N0
irst	Input	PIN_AK4	3B	B3B_N0
mod	Output	PIN_B3	8A	B8A_N0
out_A[13]	Output	PIN_D4	8A	B8A_N0
out_A[12]	Output	PIN_E4	8A	B8A_N0
out_A[11]	Output	PIN_E2	8A	B8A_N0
out_A[10]	Output	PIN_E3	8A	B8A_N0
out_A[9]	Output	PIN_C4	8A	B8A_N0
out_A[8]	Output	PIN_D5	8A	B8A_N0
out_A[7]	Output	PIN_C5	8A	B8A_N0
out_A[6]	Output	PIN_D6	8A	B8A_N0
out_A[5]	Output	PIN_F6	8A	B8A_N0
out_A[4]	Output	PIN_G7	8A	B8A_N0
out_A[3]	Output	PIN_D7	8A	B8A_N0
out_A[2]	Output	PIN_E8	8A	B8A_N0
out_A[1]	Output	PIN_A8	8A	B8A_N0
out_A[0]	Output	PIN_A9	8A	B8A_N0
ready_in	Input	PIN_AA15	3B	B3B_N0
stop_in	Input	PIN_AA14	3B	B3B_N0

Рисунок 2.5 – Таблица пинов для платы DE10

Сохраните все изменения, проведите компиляцию. Зайдите в «RTL Viewier», проверьте общую получившуюся схему соединений и сравните со схемой в приложении 2.

Подключите ЦАП к плате (плата должны быть выключена). Включите плату. Загрузите прошивку на плату.

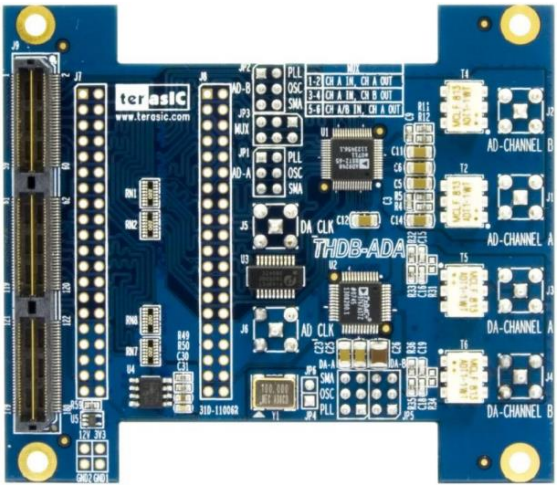


Рисунок 2.6 – ЦАП

Подключите коаксиальный кабель к разъёму «DA-CHANNEL A».

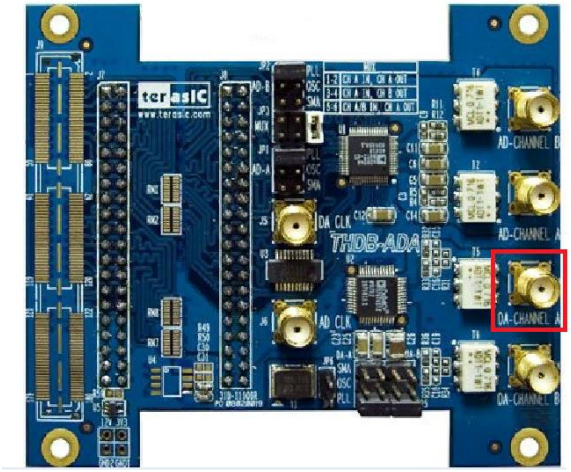


Рисунок 2.7 – ЦАП

К входу осциллографа подключите второй конец кабеля. На плате нажмите кнопку старта. На экране осциллографа должен появится сигнал.

Теперь постройте спектр сигнала с центральной частотой 10 МГц и полосой 10 МГц. Для этого нажмите кнопку *FFT* и установите: «Span» – 10 МГц, «Center» – 10 МГц. Развертку по времени установите 50 микросекунд. Должен получится спектр *OFDM*.

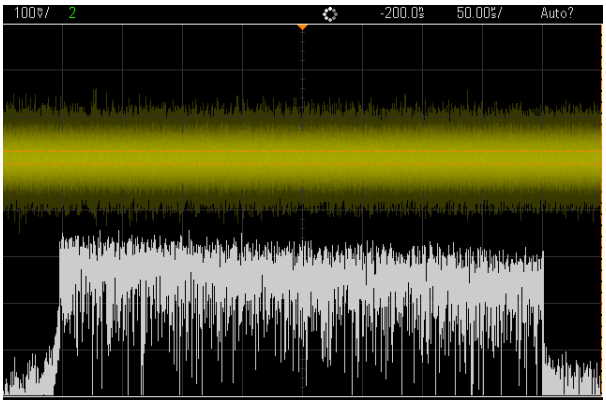


Рисунок 2.8 – Сигнал на экране осциллографа.

Работа № 8

«Создание проекта приёмника OFDM сигнала»

Цель работы: создание проекта в Quartus II под знакомство со структурой приёмника OFDM сигнала на ПЛИС.

Задачи работы:

1) Создание репозитория по проект;
2) Создание топ-файла проекта приёмника OFDM.

1. Практическая часть

2) Первым этапом работы откройте папку с проектом SoCKit_ADA. Папка расположена в «ПУТЬ». На рисунке 1.1 показаны файлы и папки, содержащиеся внутри.

Имя	Дата изменения	Тип	Размер
db	12.09.2023 8:40	Папка с файлами	
demo_batch	12.09.2023 8:21	Папка с файлами	
Input Data	12.09.2023 8:30	Папка с файлами	
v	12.09.2023 8:38	Папка с файлами	
c5_pin_model_dump.txt	11.09.2023 18:40	Текстовый докум...	5 КБ
pll.xml	12.10.2015 19:14	Документ XML	11 КБ
PLLJ_PLLSPE_INFO.txt	12.10.2015 19:14	Текстовый докум...	1 КБ
SoCKit_ADA.done	11.09.2023 18:42	Файл "DONE"	1 КБ
SoCKit_ADA.fit.smsg	11.09.2023 18:40	Файл "SMSG"	1 КБ
SoCKit_ADA.fit.summary	11.09.2023 18:40	Файл "SUMMARY"	1 КБ
SoCKit_ADA.htm	12.10.2015 19:14	Microsoft Edge H...	37 КБ
SoCKit_ADA.jdi	11.09.2023 18:40	Файл "JDI"	7 КБ
SoCKit_ADA.map.summary	11.09.2023 18:32	Файл "SUMMARY"	1 КБ
SoCKit_ADA.pin	11.09.2023 18:40	Файл "PIN"	103 КБ
SoCKit_ADA.qpf	12.10.2015 19:14	Файл "QPF"	1 КБ
SoCKit_ADA.qsf	12.09.2023 8:40	Файл "QSF"	141 КБ
SoCKit_ADA.qws	12.09.2023 8:40	Файл "QWS"	1 КБ
SoCKit_ADA.sdc	12.10.2015 19:14	Файл "SDC"	3 КБ
SoCKit_ADA.sld	11.09.2023 18:40	Файл "SLD"	1 КБ
SoCKit_ADA.sof	11.09.2023 18:41	Файл "SOF"	6 560 КБ
SoCKit_ADA.sta.summary	11.09.2023 18:42	Файл "SUMMARY"	7 КБ
SoCKit_ADA.v	16.06.2023 16:07	Файл "V"	6 КБ
SoCKit_ADA_assignment_defaults.qdf	12.10.2015 19:14	Файл "QDF"	49 КБ
stp1.stp	11.09.2023 18:29	Файл "STP"	3 732 КБ
stp1_auto_signaltap_0.txt	11.09.2023 17:51	Текстовый докум...	1 650 КБ

Рисунок 1.1 – Содержание папки проекта

3) В папке «V» расположены несколько папок и файл верхнего уровня, как показано на рисунке 1.2.

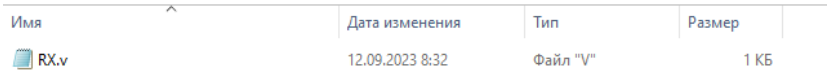
Имя	Дата изменения	Тип	Размер
.qsys_edit	12.09.2023 8:21	Папка с файлами	
pll_10_40	12.09.2023 8:21	Папка с файлами	
RX	12.09.2023 8:38	Папка с файлами	
TX	12.09.2023 8:39	Папка с файлами	
C5S_ADDA.v	12.09.2023 8:32	Файл "V"	3 КБ

Рисунок 1.2 – Содержание папки «V»

В папках TX и RX содержатся файлы модулей приемника и передатчика соответственно. C5S_ADDA.v – файл верхнего уровня (главный файл проекта). Папка pll_10_40 содержит блок преобразования частоты тактового генератора платы 50 МГц в

необходимые нам частоты (10 и 40 МГц).

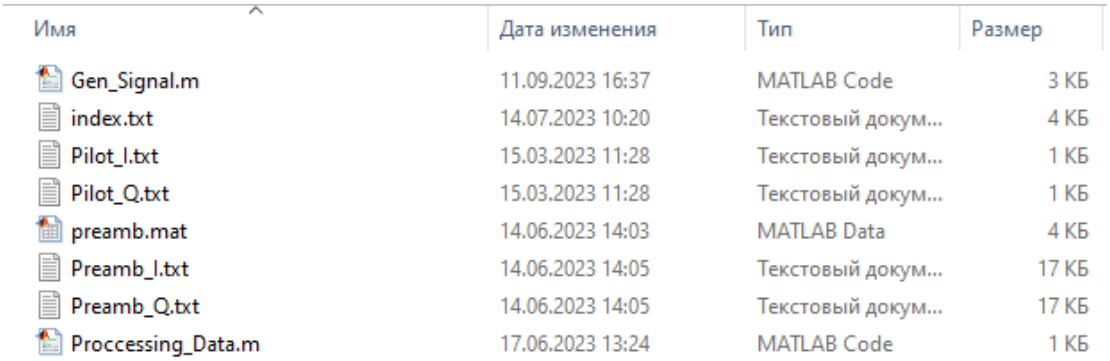
Все файлы созданных модулей необходимо сохранять в папку «RX». В данной папке уже находится файл «RX.v», как показано на рисунке 1.3. Данный файл является главным файлом приемной части всего проекта, в нем необходимо будет прописывать подключения между созданными вами модулями.



Имя	Дата изменения	Тип	Размер
RX.v	12.09.2023 8:32	Файл "V"	1 КБ

Рисунок 1.3 – Содержание папки «RX»

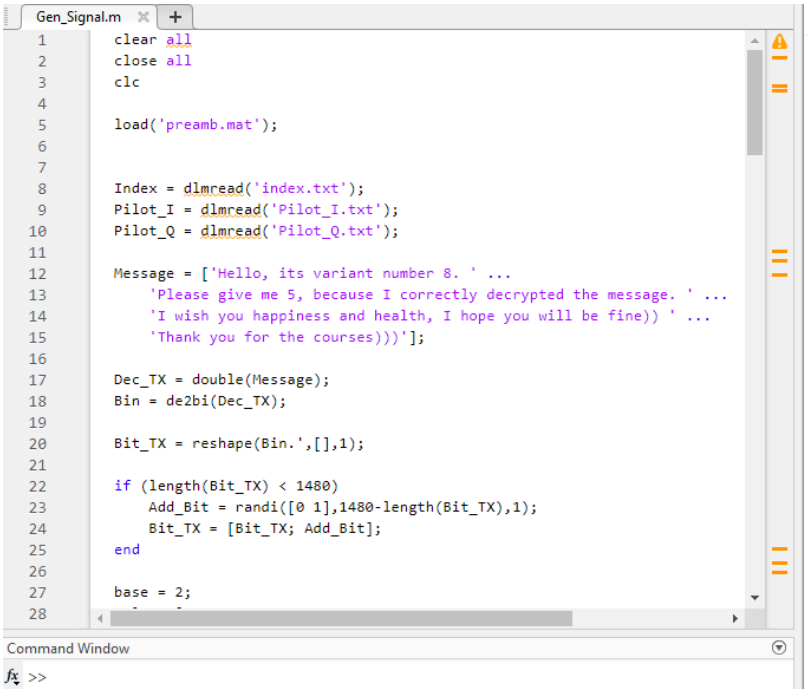
4) Вернитесь в папку SoCKit_ADA и перейдите в папку Input Data. В ней содержится несколько файлов, как показано на рисунке 1.4.



Имя	Дата изменения	Тип	Размер
Gen_Signal.m	11.09.2023 16:37	MATLAB Code	3 КБ
index.txt	14.07.2023 10:20	Текстовый докум...	4 КБ
Pilot_I.txt	15.03.2023 11:28	Текстовый докум...	1 КБ
Pilot_Q.txt	15.03.2023 11:28	Текстовый докум...	1 КБ
preamb.mat	14.06.2023 14:03	MATLAB Data	4 КБ
Preamb_I.txt	14.06.2023 14:05	Текстовый докум...	17 КБ
Preamb_Q.txt	14.06.2023 14:05	Текстовый докум...	17 КБ
Proccessing_Data.m	17.06.2023 13:24	MATLAB Code	1 КБ

Рисунок 1.4 – Содержание папки “Input Data”

Откройте файл «Gen_Signal.m». Данный файл генерирует сигнал для передачи. В сигнале содержится сообщение, которое необходимо будет получить в результате работы. На рисунке 1.5 представлена часть кода генерации сигнала.



Gen_Signal.m

```
clear all
close all
clc

load('preamb.mat');


Index = dlmread('index.txt');
Pilot_I = dlmread('Pilot_I.txt');
Pilot_Q = dlmread('Pilot_Q.txt');

Message = ['Hello, its variant number 8. ' ...
    'Please give me 5, because I correctly decrypted the message. ' ...
    'I wish you happiness and health, I hope you will be fine)) ' ...
    'Thank you for the courses)))'];

Dec_TX = double(Message);
Bin = de2bi(Dec_TX);

Bit_TX = reshape(Bin.',[],1);

if (length(Bit_TX) < 1480)
    Add_Bit = randi([0 1],1480-length(Bit_TX),1);
    Bit_TX = [Bit_TX; Add_Bit];
end

base = 2;
```

Command Window

>>

Рисунок 1.5 – Часть кода генерации сигнала

Запустите код нажатием клавиши F5, ы результате чего сгенерируется 3 файла:

«Sig_I.txt», «Sig_Q.txt» и «Pilot.txt».

5) Вернитесь в папку SoCKit_ADA и откройте проект двойным нажатием клавиши мыши на файл «SoCKit_ADA.qpf». После запуска появится окно Quartus Prime, как на рисунке 1.6.

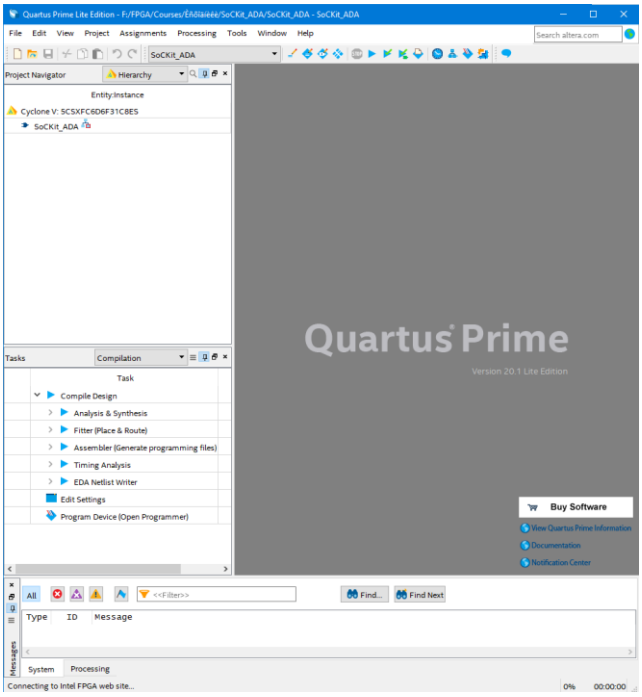


Рисунок 1.6 – Окно программы Quartus Prime

Во вкладке «Project Navigator» измените тип навигации по проекту с «Hierachy» на «Files», как показано на рисунке 1.7.

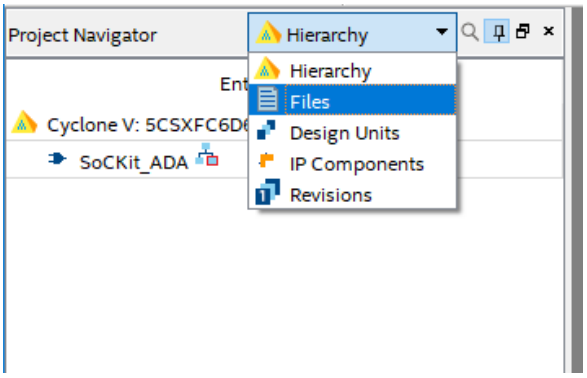


Рисунок 1.7 – Изменение типа навигации по проекту

В списке файлов найдите файл «Sig_Fr_File.v» и откройте его. На рисунке 1.8 показан списко файлов.

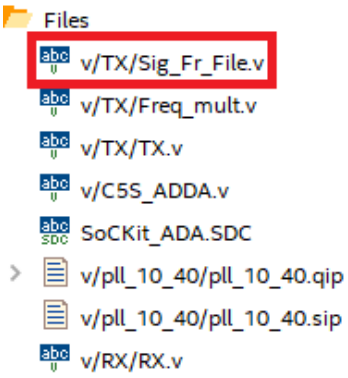


Рисунок 1.8 – Список фалов в проекте

После запуска файла модуля необходимо в строчках, выделенных на рисунке 1.9, вместо «Path to folder» указать путь до папки с проектом.

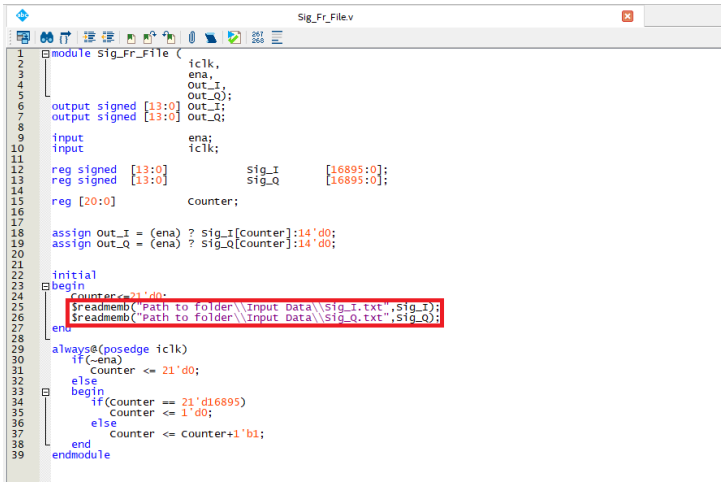


Рисунок 1.9 – Код загрузки файла

Сохраните изменения.

6) Работа состоит из 10 основных блоков: блока удаления постоянной составляющей (1), блока преобразования частоты (2), фильтра (3), блока децимации (4), коррелятора (5), блока разворота спектра (6), блока удаления циклического префикса (7), блока преобразования Фурье (8), эквалайзера (9) и блока демодуляции (10). На рисунке 1.10 представлена схема готового проекта.

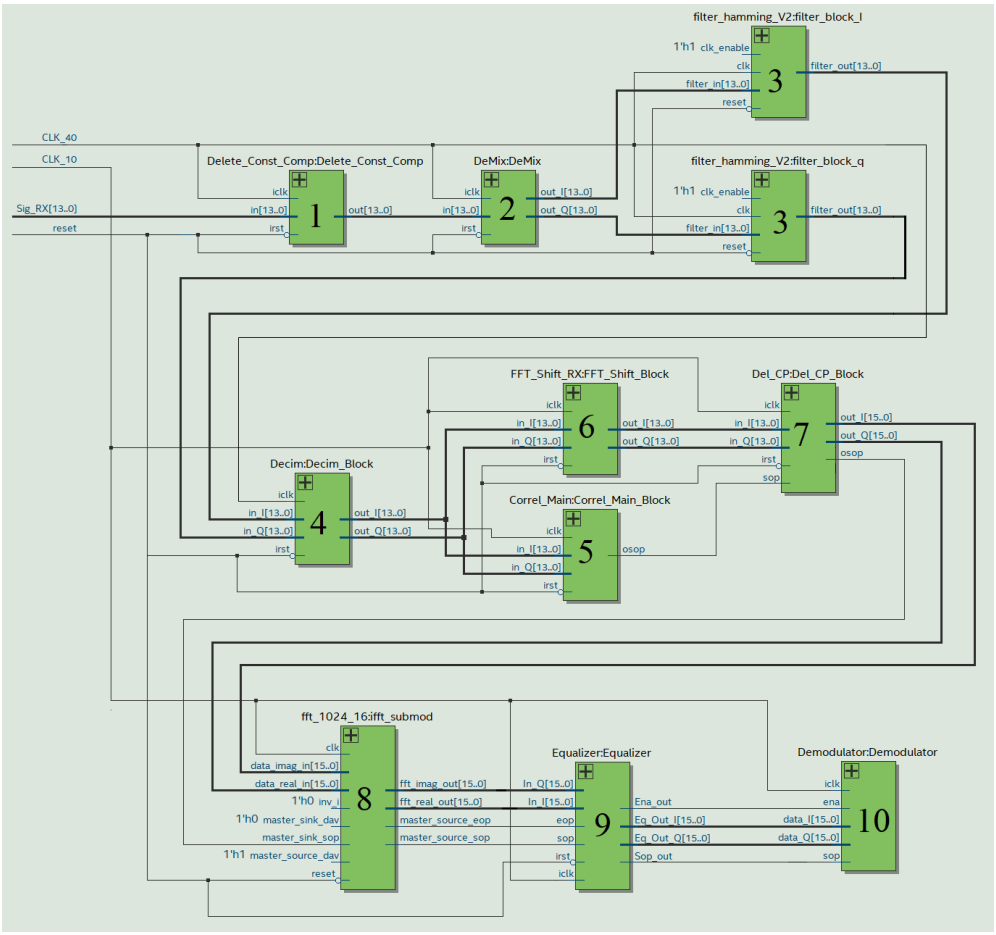


Рисунок 1.10 – Схема проекта

Работа № 9

«Удаление постоянной составляющей»

Цель работы: реализовать модуль удаления постояной составляющей на ПЛИС
Задачи работы:
1) Реализовать сдвиговый регистр;
2) Реализовать древовидный сумматор;
3) Подключение платы расширения АЦП-ЦАП.

1. Теоретическая часть

Постоянная составляющая сигнала – это часть сигнала, которая не изменяется со временем, и представляет собой постоянное значение. Данная часть является базовой составляющей сигнала, которая не имеет колебаний, волновых периодов или изменений амплитуды.

Значение постоянной составляющей сигнала – это среднее значение этого сигнала на рассматриваемом промежутке времени. Для аналогового сигнала постоянная составляющая вычисляется интегральным выражением:

$$X_0 = \frac{1}{T}\int_0^T x(t)dt\,, \tag{1.1}$$

где T – интервал времени, стремящийся к бесконечности.

Постоянная составляющая представлена на рисунке 1.1, где показан сигнал во временной области, значение постоянной составляющей показано красной линией.

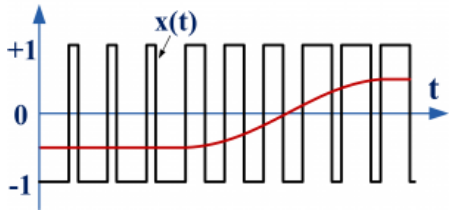


Рисунок 1.1 – Сигнал и его постоянная составляющая

Для цифрового сигнала оценка постоянной составляющей вычисляется как среднее арифметическое выборки из N отсчетов.

Так как с помощью сумматора возможно суммирование лишь двух чисел, то для суммирования большого количества чисел необходима реализация древовидного сумматора. На рисунке 1.2 показана схема древовидного сумматора для 8 чисел.

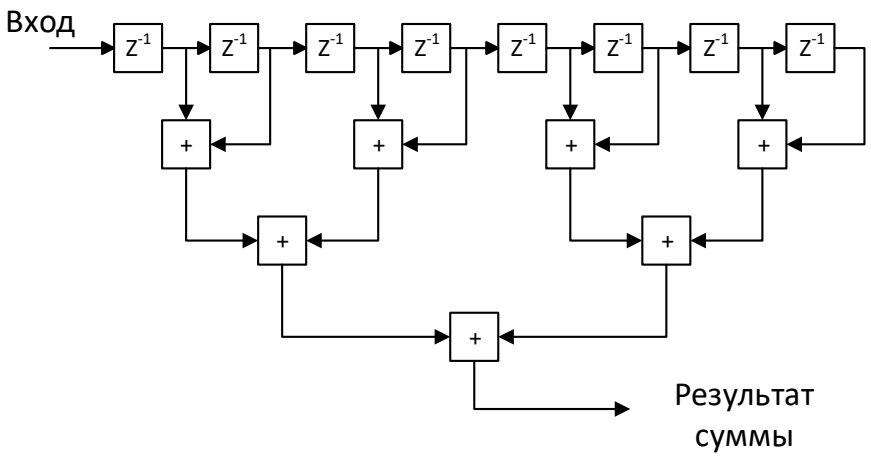


Рисунок 1.2 – Схема древовидного сумматора

Древовидный сумматор состоит из двух основных частей: сдвигового регистра и древа сумматоров. Сдвиговый регистр необходим для накопления чисел, которые необходимо суммировать. Древо сумматора производит суммирование содержимого регистра.

Для нахождения среднего значения, необходимо результат суммирования поделить на количество чисел суммирования. Размер окна должен быть степенью числа два. Это необходимо для удобства реализации сумматора и деления.

2. Практическая часть

1) Блок удаления постоянной составляющей работает следующим образом: входной сигнал записывается в первую ячейку сдвигового регистра. Размер сдвигового регистра равен окну, в котором расчитывается среднее значение. Значения каждой ячейки регистра суммируются при помощи древовидного сумматора. Результат суммирования делится на размер окна, результатом деления будет среднее значение чисел в сдвиговом регистре. Для компенсации постоянной составляющей необходимо вычесть из значения в последней ячейке сдвигового регистра вычисленное среднее значение. Результат разницы поступает на выходной порт.

2) Откройте проект SoCKit_ADA, с которым вы работали в вводной лабораторной работе, и создайте новый файл Verilog HDL File. В начале файла необходимо инициализировать модуль Delete_Const_Comp, параметры и порты ввода/вывода. Описание портов и параметров представлены в таблице 2.1.

Таблица 2.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
in	input signed	[13:0]	Входной порт данных
iclk	input		Входной порт тактирования
irst	input		Входной порт сброса
out	output wire signed	[13:0]	Выходной порт данных
Window	parameter		Параметр, отвечающий за размерность окна
Numb_Layer	parameter		Параметр, отвечающий за количество слоев древовидного сумматора

Для возможности управления параметрами из файла модуля более высокого уровня используется следующая конструкция: после указания названия модуля ставится значок решетки (#) и в скобках после него указываются параметры, в следующих скобках указываются внешние порты. Описание параметров и портов данного модуля будет выглядеть следующим образом:

```
module Delete_Const_Comp#(     parameter Window = 64,
                               parameter Numb_Layer = $clog2(Window))
                           (   iclk,
                               irst,
                               in,
                               out);
```

Параметр Window отвечает за размер окна, при этом его значение обязательно должно быть степенью 2. В данном случае размер окна равен 64. Параметр Numb_Layer отвечает за

количество слоев древовидного сумматора и вычисляется как логарифм от размера окна по основанию 2.

3) Следующим этапом необходимо описать переменные, порты, внутренние регистры и шины. Список всех шин и регистров представлен в таблице 2.2.

Таблица 2.2 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
Buffer	reg signed	[13:0]	[Window-1:0]	Сдвиговый регистр для накопления входного сигнала
Sum_Wire	wire signed	[13:0]	[Numb_Layer-1:0] [Window-1:0]	Шины соединяющий отдельные блоки суммирования в один древовидный сумматор
Mean	wire signed	[13:0]		Шина, соединяющая выход последнего сумматора
k	integer	-	-	Переменная необходимая для реализации сдвига в регистре

Ниже представлен код для инициализации переменных.

```
input                                   iclk;
input                                   irst;
input          signed          [13:0]   in;
output wire    signed          [13:0]   out;
reg            signed          [13:0]   Buffer      [Window-1:0];
wire           signed          [13:0]   Sum_Wire    [Numb_Layer-1:1][Window-1:0];
wire           signed          [13:0]   Mean;
integer                                 k;
```

4) Следующим этапом реализуйте сдвиг данных в сдвиговом регистре при помощи цикла for, где каждой последующей ячейке присваивается значение предыдущей. При сигнале irst сдвиговый регистр Buffer заполняется нулями.

В цикле переменная k изменяется от Window-1 до 1. В ячейку с адресом 0 записывается значение со входного пина.

```
always@(posedge iclk)
begin
        // Сдвиговый регистр заполняется нулями
        if(irst)
                for(k = 0; k < Window; k = k+1)
                begin
                        Buffer[k] <= 14'd0;
                end
        else
        // Производится сдвиг регистра и запись принятого значения в 0 ячейку.
        begin
                for(k = Window -1; k > 0; k = k-1)
```

```
                begin
                        Buffer[k] <= Buffer[k-1];
                end

                Buffer[0] <= in;
        end
end
```

5) Далее реализуйте древовидный сумматор, используя конструкцию generate. Данная конструкция позволяет создавать множество блоков из одного экземпляра. Использование конструкции начинается с ключевого слова generate и заканчивается endgenerate. С помощью ключевого слова genvar задаются переменные, используемые в создании блоков. В данном случае необходимы две переменные для двух циклов for. Далее прописывается первый цикл for, отвечающий за номер слоя в сумматоре. Данный цикл изменяется с шагом один до значения Numb_Layer-1. После пишется ключевое слово begin и к нему через двоеточие приписывается название блоков, которые будут генерироваться, в данном случае Summ_B1. После прописывается второй цикл for, изменяющийся от 0 до Window-1 с шагом 2^(i+1), где i – номер слоя. Данный цикл выстраивает сумматоры в каждом слое и соединяет предыдущий слой с последующим. В цикле стоит условие if для соединения первого слоя со сдвиговым регистром. В теле условия прописывается подключение сумматора. Код для реализации древовидного сумматора представлен ниже:

```
generate
genvar i,j;
        for(i = 0;i<Numb_Layer-1;i=i+1)

        begin: Summ_B1
                for(j = 0;j<Window;j=j+2**(i+1))
                begin: Summ_B2
                        if(i == 0)
                        begin
                                Summ #(14) Summ_Bl_Means(
                                                .Summ_1         (Buffer[j]),
                                                .Summ_2         (Buffer[j+2**(i)]),
                                                .out
(Sum_Wire[i+1][j]));
                        end
                        else
                        begin
                                Summ #(14) Summ_Bl_Means(
                                                .Summ_1         (Sum_Wire[i][j]),
                                                .Summ_2
(Sum_Wire[i][j+2**(i)]),
                                                .out
(Sum_Wire[i+1][j]));
                        end
                end
        end
endgenerate
```

Последний блок сумматора необходимо подключить без использования структуры generate. Это необходимо для того, чтобы была возможность подключения другой шины. Код для подключения представлен ниже.

```
Summ #(14) Summ_lst_Means(
                .Summ_1         (Sum_Wire[Numb_Layer-1][0]),
                .Summ_2         (Sum_Wire[Numb_Layer-1][2**(Numb_Layer-
1)]),
                .out            (Mean));
```

После необходимо прописать подключение входного порта к выходному с компенсацией. Для компенсации необходимо вычесть из входного сигнала вычисленную сумму деленую на размер окна (параметр Window).

```
assign out = in - Mean/Wind_Size;
```

В конце пропишите окончание модуля endmodule. Сохраните все изменения.

```
endmodule
```

Откройте файл RX.v. Пропишите подключение модуля Delete_Const_Comp, предварительно создав 14 разрядную шину Delete_Const_DeMix_Data. После, используя символ решетки (#), можно установить значения параметров. Так как второй параметр напрямую зависит от первого, то необходимо указать одно число, отвечающее за размерность окна, это число в обязательном порядке должно быть степенью числа 2. На входной порт in подключите шину Sig_RX, на iclk – CLK_40, irst – reset, out – Delete_Const_DeMix_Data. Код подключения представлен ниже.

```
//////////////////Delete_Const_Comp/////////////
wire signed [13:0] Delete_Const_DeMix_Data;

Delete_Const_Comp Delete_Const_Comp #(32)(
                        .iclk   (CLK_40),
                        .irst   (~reset),
                        .in     (Sig_RX),
                        .out    (Delete_Const_DeMix_Data));
```

6) После создайте блок суммирования (Summ), который использовался в древовидном сумматоре. Создайте новый файл Verilog HDL File. Описание портов и параметров представлено в таблице 2.3.

Таблица 2.3 – Описание портов и параметров

Название	Тип	Разрядность	Описание
N_Bit	parameter		Данный параметр отвечает за разрядность входных чисел
Summ_1	input signed	[0:0]	Входной порт для первого числа
Summ_2	input signed	[N_Bit-1:0]	Входной порт для второго числа
out	input signed	[N_Bit-1:0]	Сумма чисел на входных портах

Код инициализации модуля и портов представлено ниже:

```
module Summ #(parameter N_Bit = 14)(     input   signed [N_Bit-1:0] Summ_1,
                                         input   signed [N_Bit-1:0] Summ_2,
                                         output signed  [N_Bit-1:0] out);
```

Далее с помощью конструкции assigned присвойте выходной шине *out* сумму входных портов Summ_1 и Summ_2. После добавьте строчку окончания модуля и сохраните все изменения.

```
assign out = Summ_1+Summ_2;
endmodule
```

7) Скомпилируйте проект, нажав на кнопку на панели инструментов, как показано на риснуке 2.3.

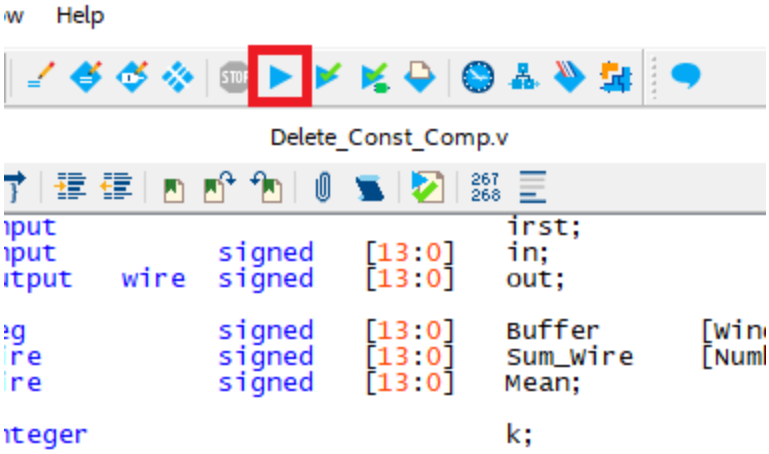


Рисунок 2.3 – Запуск компиляции проекта

После успешной компиляции проекта необходимо подключить плату к компьютеру. Подключите питание к плате в разъем 2 и кабель в разъем micro-USB (1) (Рисунок 2.4). К разъему номер 4 подключите плату расширения, общий вид которой предстален на рисунке 2.5. Для включения платы нажмите на кнопку 3.

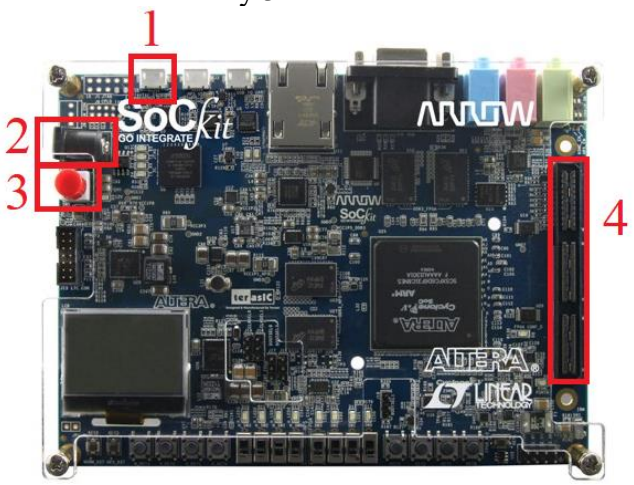


Рисунок 2.4 – Общий вид платы

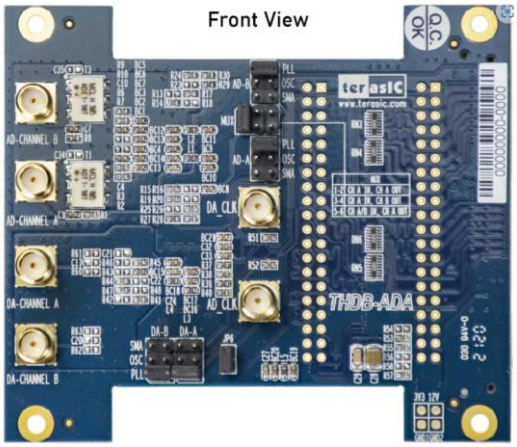


Рисунок 2.5 – Карта расширения

Плата расширения на рисунке 2.5 имеет два канала на передачу и два канала на прием. Для передачи используется оба канала (DA-CHANNEL A/B), для приема используется только канал A (AD-CHANNEL A). Для получения сигнала соедините один из каналов передачи с приемным каналом при помощи коаксиального кабеля.

8) Для верификации созданного модуля откройте Signal Tap Logic Analyzer в панели инструментов во вкладке Tools, как показано на рисунке 2.6. Данный инструмент позволяет отслеживать изменения сигналов в ПЛИС.

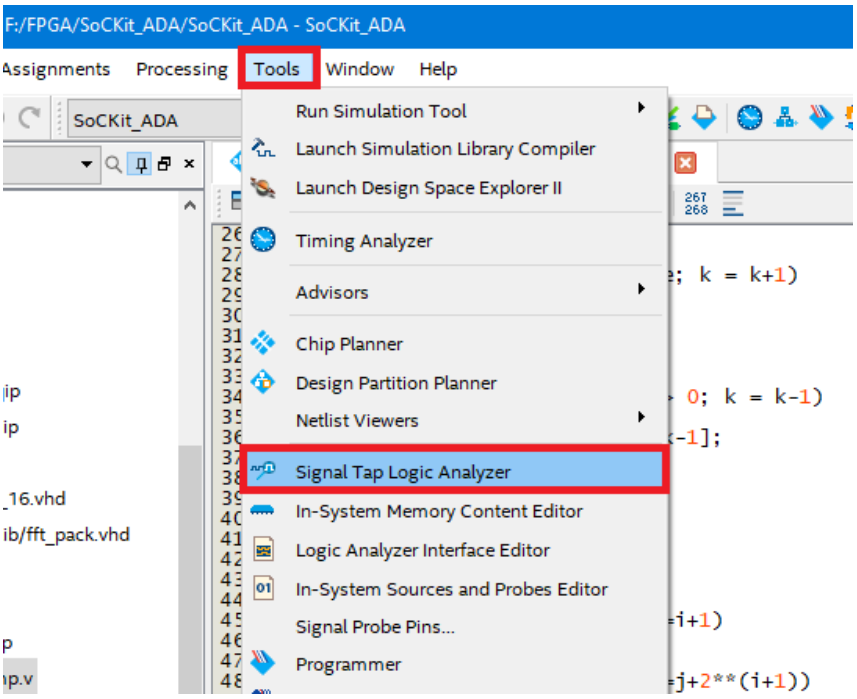


Рисунок 2.6 – Открытие Signal Tap Logic Analyzer

После чего откроется окно как на рисунке 2.7.

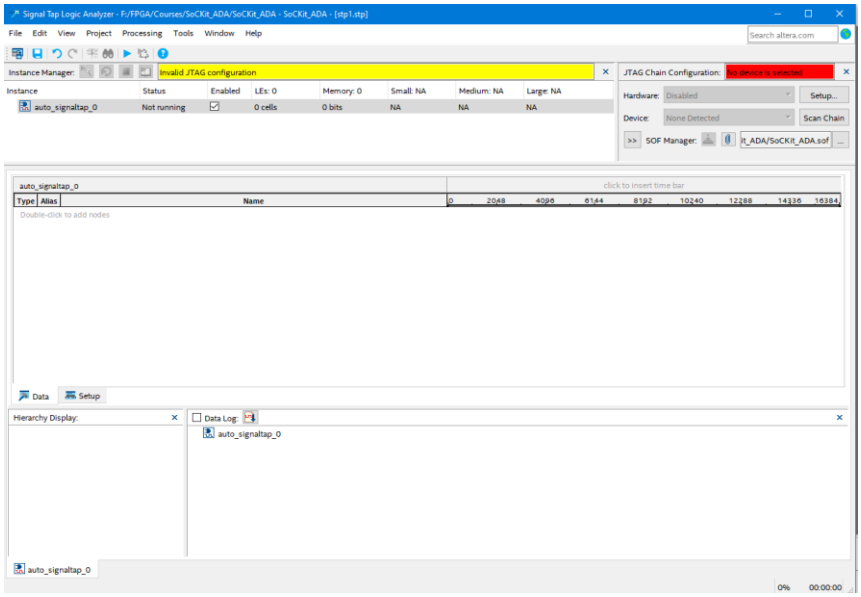


Рисунок 2.7 – Окно Signal Tap Logic Analyzer

Перейдите во вкладку Setup, как показано на рисунке 2.8.

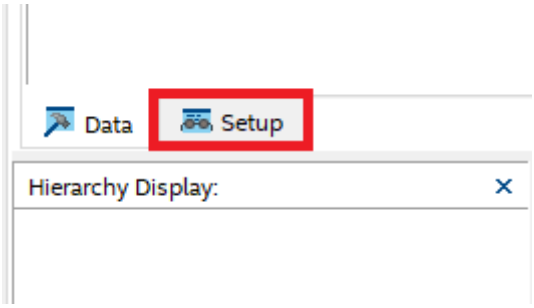


Рисунок 2.8 – Вкладка Setup

После чего необходимо добавить сигналы, для вывода их на экран. Для этого нажмите двойным кликом левой кнопкой мыши по выделенной области, показанной на рисунке 2.9.

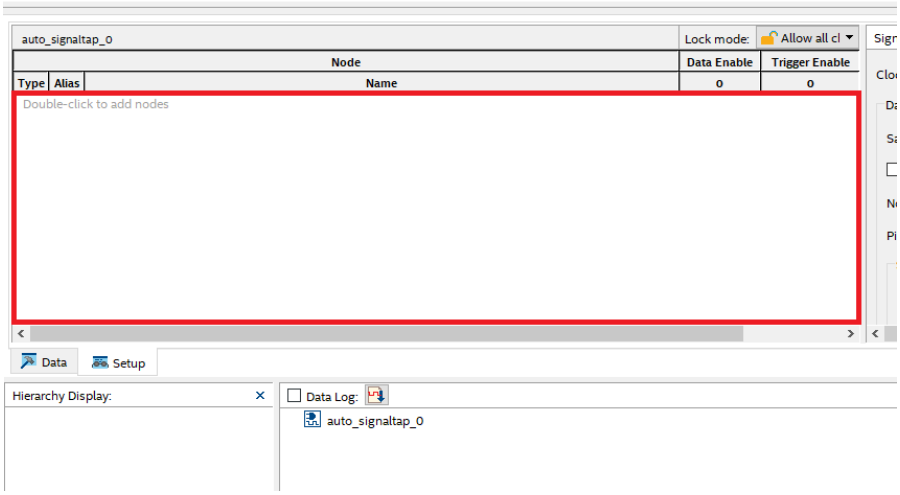


Рисунок 2.9 – Добавление сигналов

После чего откроется дополнительное окно, как на рисунке 2.10.

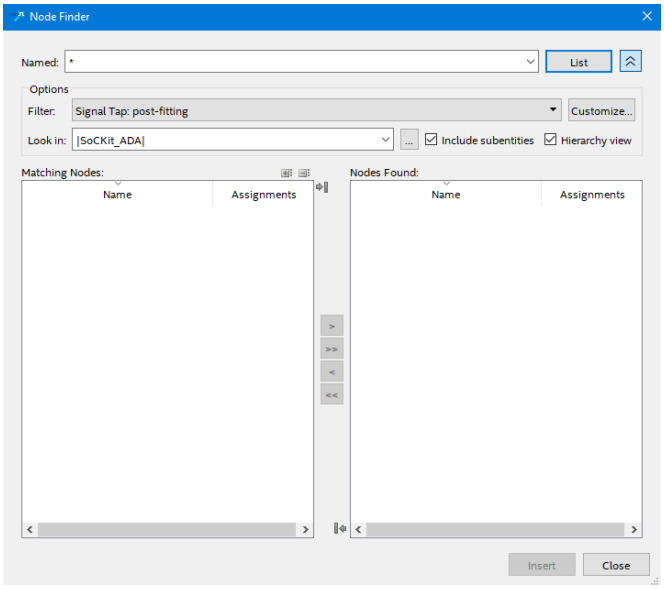


Рисунок 2.10 – Окно с сигналами и переменными

Установите параметр Filter, как показано на рисунке 2.11, и после нажмите кнопку List.

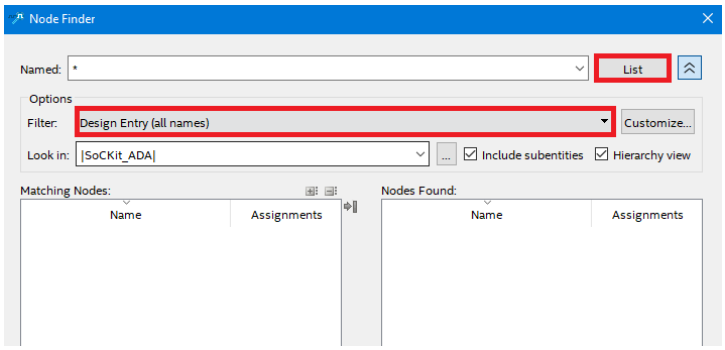


Рисунок 2.11 – Установка фильтра сигналов

После чего в левом нижнем поле появится список всех возможных сигналов. Пролистните в самый низ списка и найдите C5S_ADDA:C5S_ADDA_ins, разверните этот вложенный список. Пролистните в самый конец и найдите RX:RX, разверните этот список. Выделите сигнал Sig_RX, как показано на рисунке 2.12. После чего нажмите на стрелочку между двумя полями. Сигналы перенесутся в правое поле.

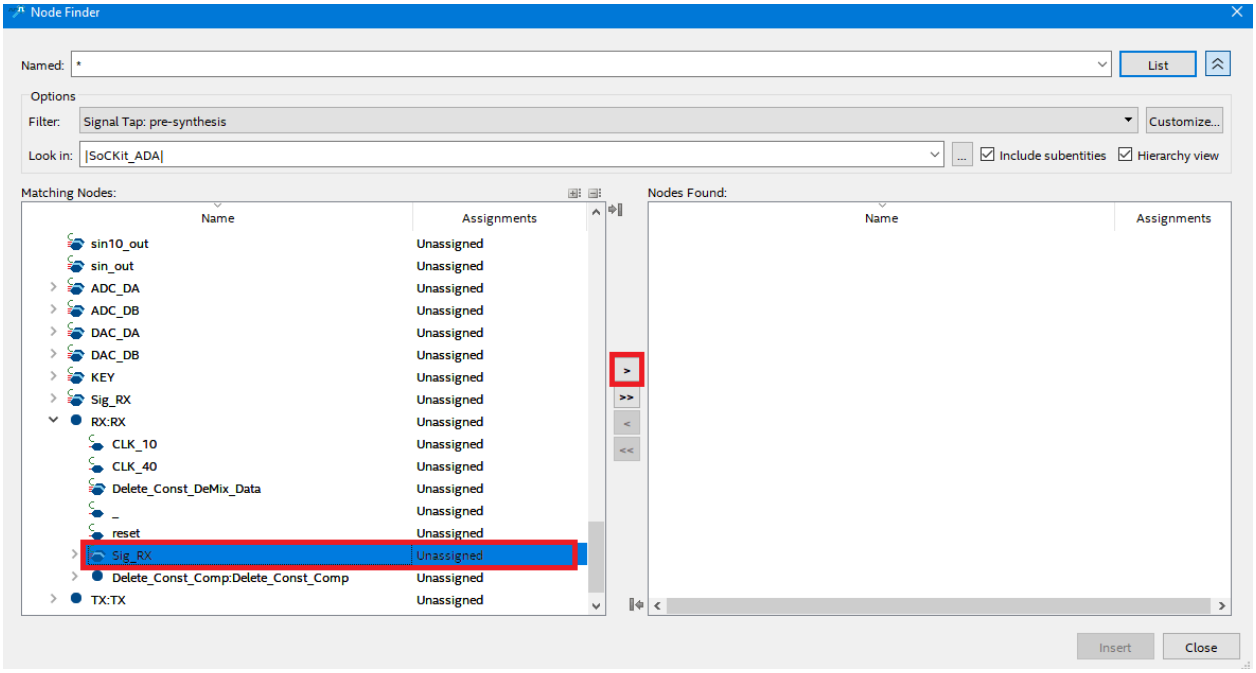


Рисунок 2.12 – Выбор сигналов

Далее разверните список Delete_Const_Comp:Delete_Const_Comp и в появившемся списке найдите порт out, выделите его и нажмите на стрелочку для переноса в правое поле. После этих действий нажмите на кнопку Insert в правом нижнем углу, как показано на риснуке 2.13, и закройте это окно.

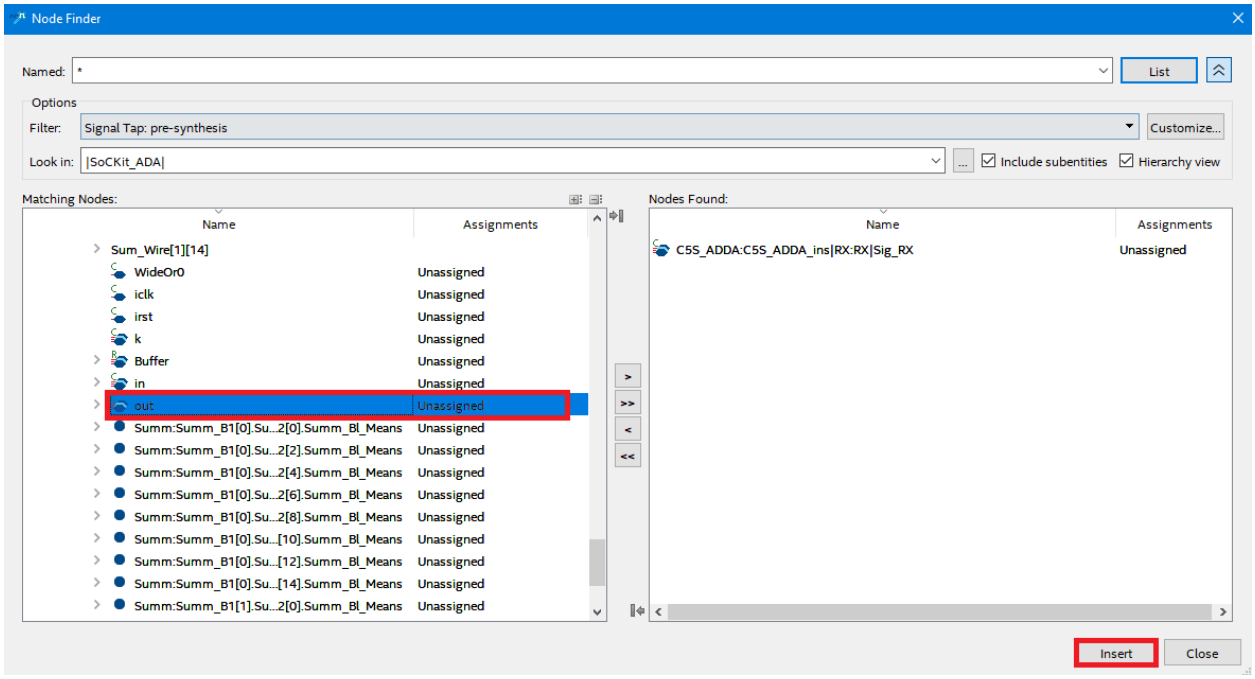


Рисунок 2.13 – Выбор сигналов и загрузка их в Signal Tap

После выполненных действий в списке в Signal Tap появятся ваши выбранные сигналы, как на рисунке 2.14.

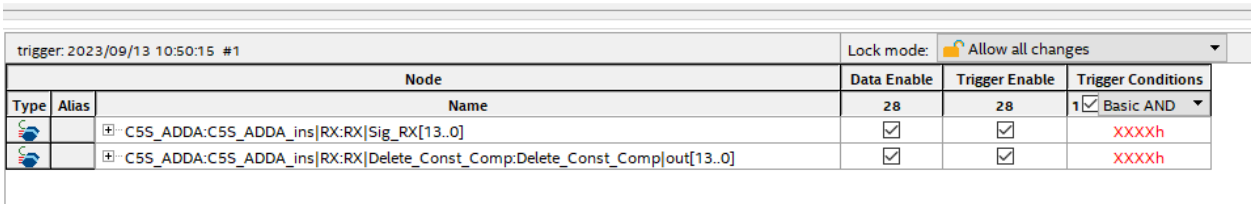


Рисунок 2.14 – Список сигналов в Signal Tap

Далее необходимо выбрать сигнал с тактовым сигналом частотой 40 МГц, для этого в правой части окна в области Signal Configuration нажмите на кнопку, выделенную на рисунке 2.15.

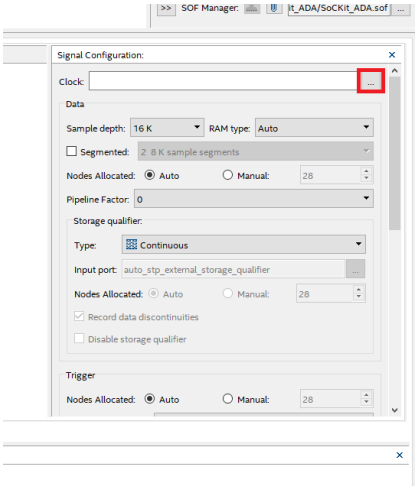


Рисунок 2.15 – Выбор тактового сигнала

Установите те же параметры как и на рисунке 11, и нажмите кнопку List. После чего разверните список C5S_ADDA:C5S_ADDA_ins, после него pll_10_40:pll_0. В списке выберете outclk_1, с помощью стрелочки перенесите его в правую часть и нажмите кнопку OK, как показано на рисунке 2.16.

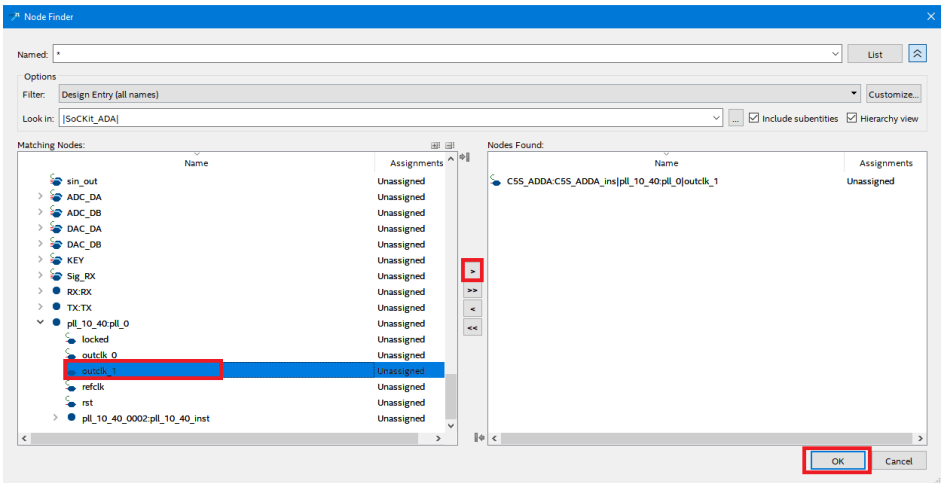


Рисунок 2.16 – Выбор тактового сигнала 40 МГц

Так же ниже установки тактового сигнала, установите Sample depth 16K.

После выполненных манипуляций, запустите компиляцию проекта еще раз, если выйдет окно о предложении сохранить изменения нажмите Yes.

После завершения компиляции необходимо загрузить прошивку на плату. Для этого в правом верхнем углу нажмите кнопку Setup, как показано на рисунке 2.17.

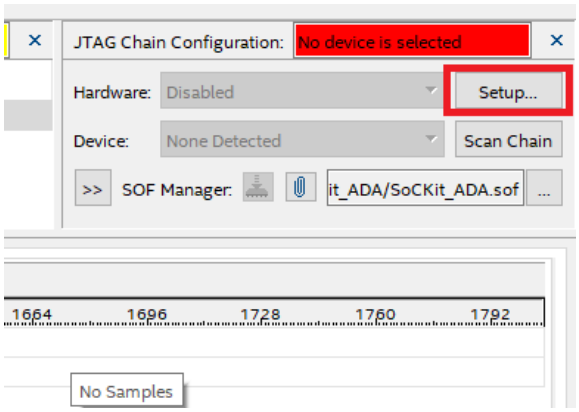


Рисунок 2.17 – Выбор платы

Откроется окно со списком подключенных плат, двойным кликом выберете плату и нажмите кнопку Close, как показано на рисунке 2.18.

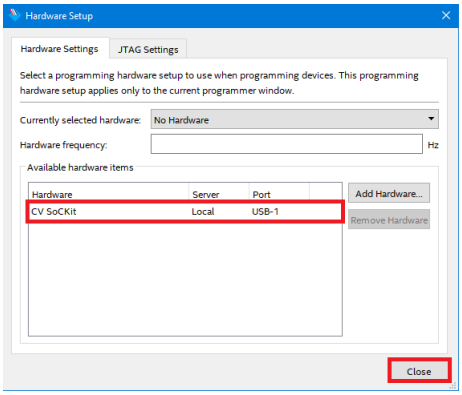


Рисунок 2.18 – Список подключенных плат

После чего нажмите кнопку загрузки прошивки в плату, как показано на рисунке 2.19.

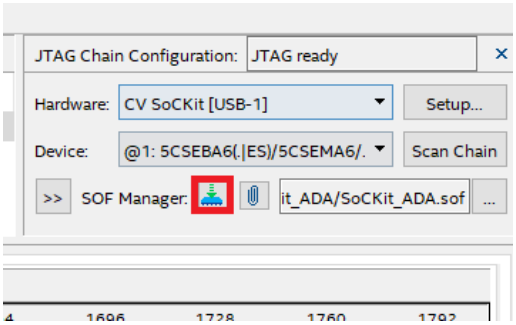


Рисунок 2.19 – Загрузка прошивки в плату

Далее запустите выгрузку отсчетов при помощи кнопки на панели инструментов, как показано на рисунке 2.20.

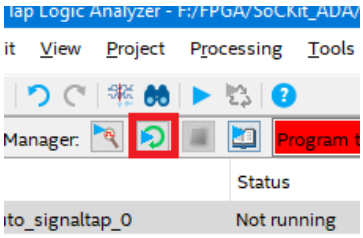


Рисунок 2.20 – Выгрузка отсчетов

После чего напротив выбранных сигналов появится отсчеты. По умолчанию отсчеты отображаются в шестнадцатеричной системе счисления. Для отображения отсчетов в десятичной системе необходимо сделать следующее: нажать правой кнопкой мыши на сигнал, в появившемся меню выберите пункт Bus Display Format, далее выберите Signed Decimal in Two Complement, как показано на рисунке 2.21.

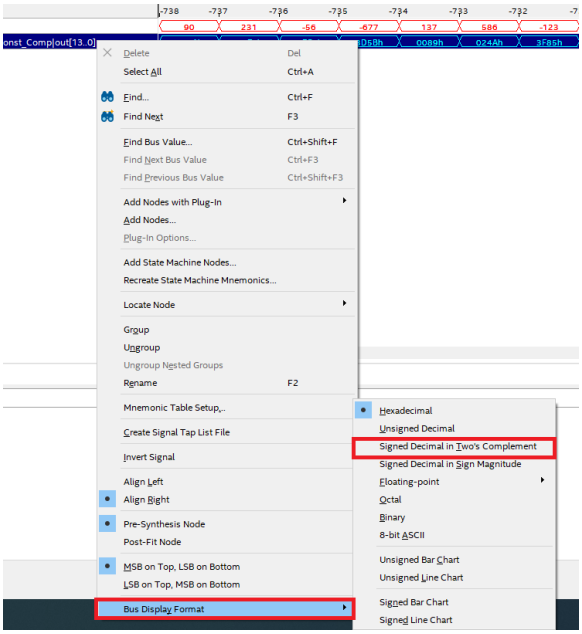


Рисунок 2.21 – Изменение системы счисления

Сравните значения первого и второго сигналов. Если сигналы имеют небольшое различие, не превышающее 10 по абсолютному значению, значит все работает верно. На рисунке 2.22 показан пример значений.

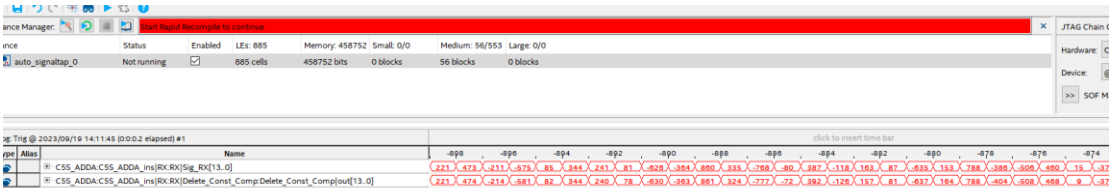


Рисунок 2.22 – Пример временной диаграммы

Работа № 10

«Преобразование частоты, перенос спектра сигнала на нулевую частоту»

Цель работы: реализовать модуля преобразования частоты на ПЛИС.
Задачи работы:
1) Реализовать модуля переноса спектра принятого сигнала на нулевую частоту;
2) Построение созвездия и спектра принимаемого сигнала.

1. Теоретическая часть

Блок преобразования частоты позволяет преобразовать вещественный сигнал в комплексный, т.е. разделяет сигнал на квадратуры. На рисунке 1.1 показана структурная схема квадратурного демодулятора.

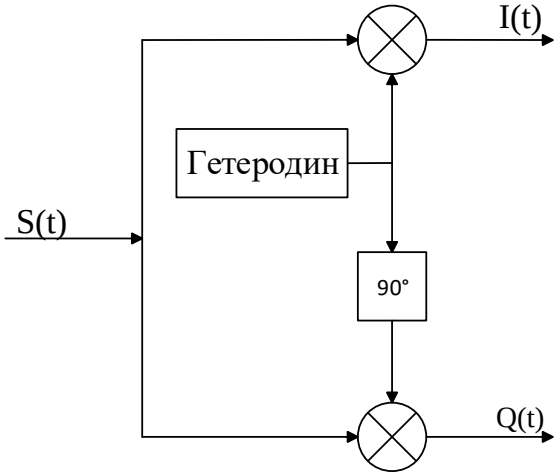


Рисунок 1.1 – Структурная схема демодулятора

На вход блока поступает вещественный сигнал S(t) и проходит через 2 умножителя. Гетеродин генерирует несущий сигнал, который поступает на два умножителя: в первом случае сдвиг несущего сигнала равен 90 градусов по фазе, а во втором ноль. После перемножения входного сигнала и со сдвинутым на 90 градусов несущим сигналом образуется мнимая часть сигнала, а с несущей без сдвига – реальная часть.

2. Практическая часть

1) Откройте проект, с которым вы работали в предыдущей работе, и создайте новый файл Verilog HDL File. В начале файла инициализируйте модуль DeMix и порты ввода/вывода. Описание портов представлено в таблице 2.1.

Таблица 2.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
in	input signed	[13:0]	Входной порт данных
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
out_I	output wire signed	[13:0]	Выходной порт реальной части сигнала
out_Q	output wire signed	[13:0]	Выходной порт мнимой части сигнала

Код инициализация модуля и портов ввода/вывода представлен ниже.

```
module DeMix(
        irst,
        iclk,
        in,
        out_I,
        out_Q);
```

2) Следующим этапом опишите внешние порты, внутренние регистры и шины, использующиеся в модуле. Описание внутренних шин и регистров представлено в таблице 2.

Таблица 2.2 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
counter	reg	[1:0]		Регистр, выступающий в роли счетчика.
koef_cosinus	reg signed	[7:0]	[3:0]	Данный регистр содержит коэффициенты косинуса для выделения реальной части из реального сигнала
koef_sinus	reg signed	[7:0]	[3:0]	Данный регистр содержит коэффициенты синуса для выделения мнимой части из реального сигнала
buf_I	reg signed	[20:0]		Промежуточный буфер, содержащий результат перемножения коэффициента косинуса с входным сигналом
buf_Q	reg signed	[20:0]		Промежуточный буфер, содержащий результат перемножения коэффициента синуса с входным сигналом

Код для описания внешних портов, внутренних регистров и шин представлен ниже.

```
input                   irst;
input                   iclk;
input   signed  [13:0]  in;
output  signed  [13:0]  out_I;
output  signed  [13:0]  out_Q;
reg     signed  [20:0]  buf_I;
reg     signed  [20:0]  buf_Q;
reg     signed  [7:0]   koef_cosinus    [3:0];
reg     signed  [7:0]   koef_sinus      [3:0];
reg             [1:0]   counter;
```

3) Далее при помощи оператора initial пропишите в регистры koed_cosinus и koef_sinus значения синуса и косинуса, а также задайте начальное нулевое состояние счетчику counter. Код данных операций представлен ниже.

```
initial
```

```
begin
        koef_cosinus[0] =  14'd25;
        koef_cosinus[1] =  14'd125;
        koef_cosinus[2] = -14'd25;
        koef_cosinus[3] = -14'd125;

        koef_sinus[0] =  14'd125;
        koef_sinus[1] = -14'd25;
        koef_sinus[2] = -14'd125;
        koef_sinus[3] =  14'd25;

        counter <= 2'b0;
end
```

4) Следующим этапом реализуйте операцию умножения коэффициентов синуса и косинуса на входной сигнал. Значение коэффициентов выбираются в зависимости от значения счетчика counter. Вычисленные значения записываются в промежуточные буферы buf_I и buf_Q с делением значений на 32 для предотвращения переполнения в результате умножения двух больших чисел. После к значению счетчика прибавляется единица. Так же необходимо при значении irst производить сброс промежуточных буферов и счетчика. Код представлен ниже.

```
always@(posedge iclk or posedge irst)
begin
        // Операция обнуления счетчика и буфера
        if(irst)
        begin
                buf_I  <= 21'b0;
                buf_Q  <= 21'b0;
                counter <= 2'b0;
        end
        // Операции умножения входного сигнала на коэффициенты и счет счетчика
        else
        begin
                buf_I <= (in * koef_cosinus[counter])/32;
                buf_Q <= (in * koef_sinus[counter])/32;
                counter <= counter+1'b1;
        end
end
```

После пропишите подключение выходных портов (out_I и out_Q) к промежуточным буферам buf_I и buf_Q при помощи структуры assign.

```
assign out_I = {buf_I[20], buf_I[12:0]};
assign out_Q = {buf_Q[20], buf_Q[12:0]};
```

В конце добавьте строчку окончания модуля endmodule.

```
endmodule
```

5) Сохраните файл как DeMix.v. После откройте файл RX.v и пропишите подключение созданного модуля. Предварительно создайте две 14 разрядных шины DeMix_Filter_Data_I и DeMix_Filter_Data_Q. К порту irst подключите шину reset, к iclk – CLK_40, in - Delete_Const_DeMix_Data, а к портам out_I и out_Q - DeMix_Filter_Data_I и DeMix_Filter_Data_Q соответственно. Код для подключения представлен ниже.

```
////////////////// DEMIXER //////////////////////

wire signed [13:0] DeMix_Filter_Data_I;
wire signed [13:0] DeMix_Filter_Data_Q;

DeMix DeMix(
            .irst       (~reset),
            .iclk       (CLK_40),
            .in         (Delete_Const_DeMix_Data),
            .out_I      (DeMix_Filter_Data_I),
            .out_Q      (DeMix_Filter_Data_Q));
```

6) Скомпилируйте проект. После откройте Signal Tap и добавьте выходные порты Out_I и Out_Q из блока DeMix, как на рисунке 2.1.

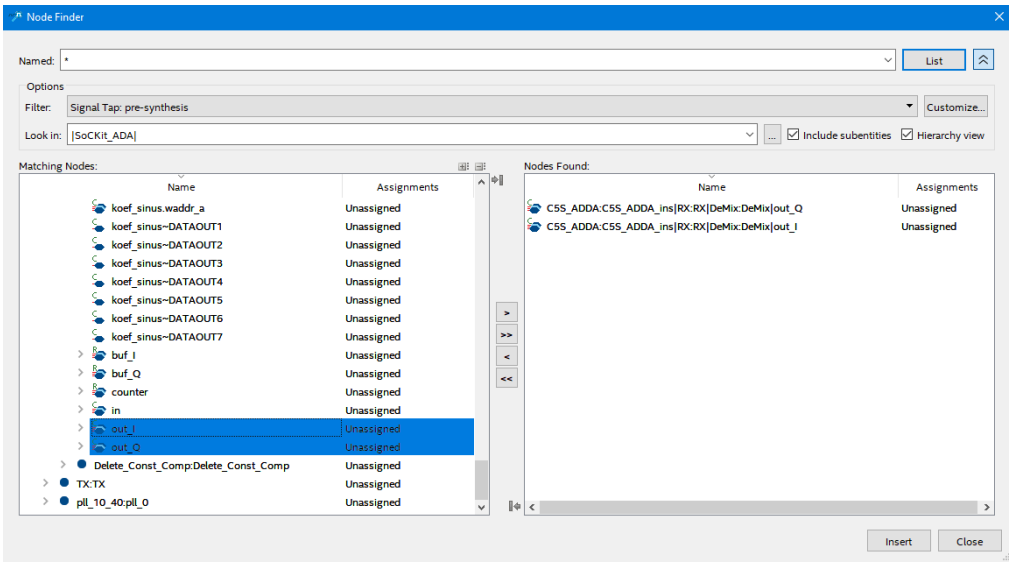


Рисунок 2.1 – Выходные порты блока DeMix

После того, как вы добавите порты, скомпилируйте проект и загрузите прошивку в плату. Запустите считывание отсчетов и далее поставьте на паузу с помощью кнопки, показанной на рисунке 2.2.

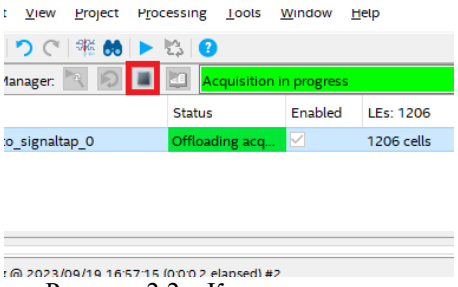


Рисунок 2.2 – Кнопка для паузы

После остановки на паузу необходимо перенести данные отсчеты в MATLAB для

верификации. Для этого нажмите правой кнопкой мыши по пустой области и выберите пункт Create Signal Tap List File.

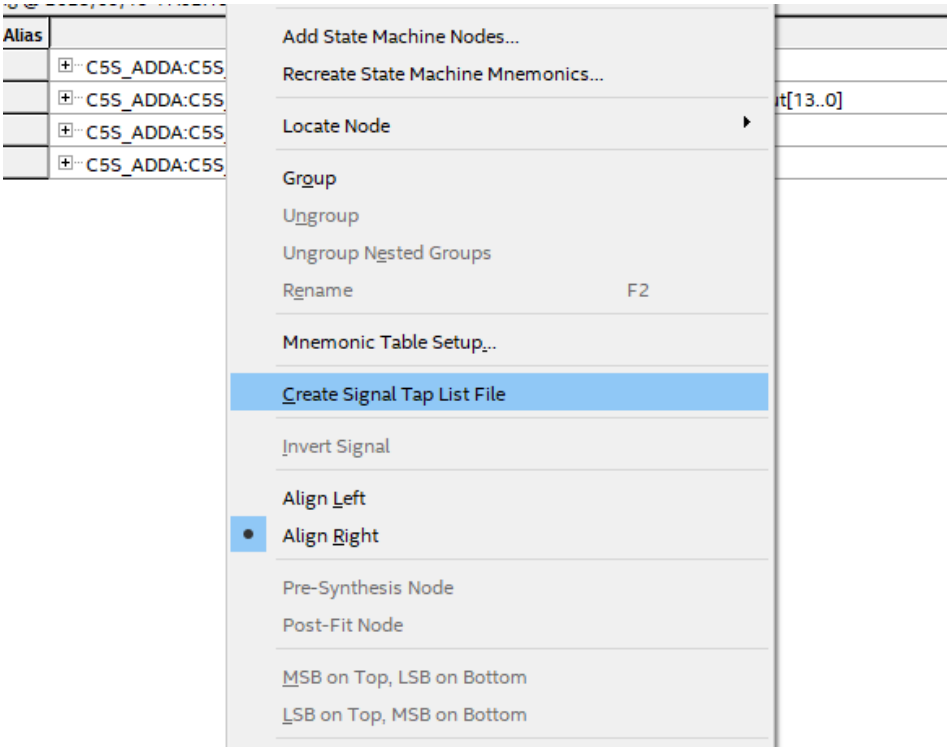


Рисунок 2.3 – Создание файла с отсчетами

Откроется файл, в котором будут отсчеты всех сигналов. Скопируйте два столбца с отсчетами out_Q и out_I. Для выделения столбцов нажмите и удерживайте комбинацию клавиш Shift + Alt.

После откройте папку с вашим проектом и откройте папку Verification. Затем откройте файл 2.txt и скопируйте туда отсчеты. В этой же папке лежит файл FoundScatter.m – откройте его. Код, содержащийся в этом файле, представлен на рисунке 2.4

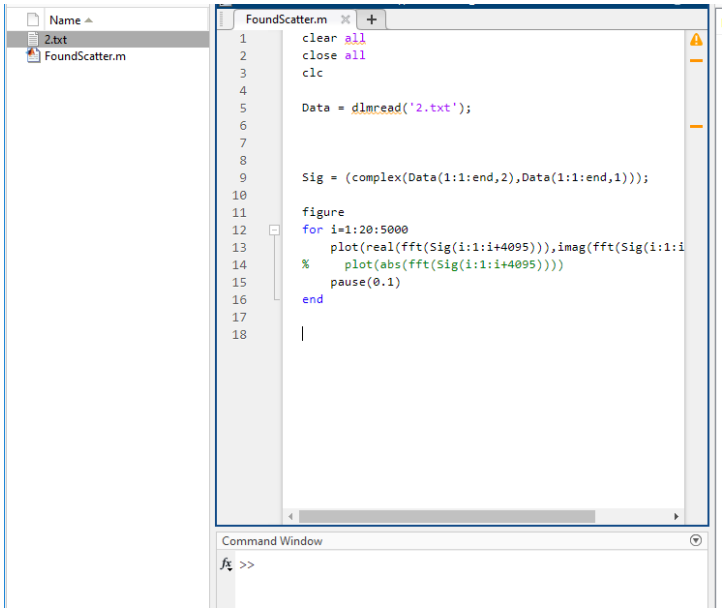


Рисунок 2.4 – Код программы для проверки принятого сигнала

Запустите программу нажатием клавиши F5. После запуска программы построится

созвездие, которое со временем будет изменяться. Если ваше созвездие подобно созвездию на рисунке 2.5, значит сигнал присутствует и произведен верный перенос на нулевую частоту.

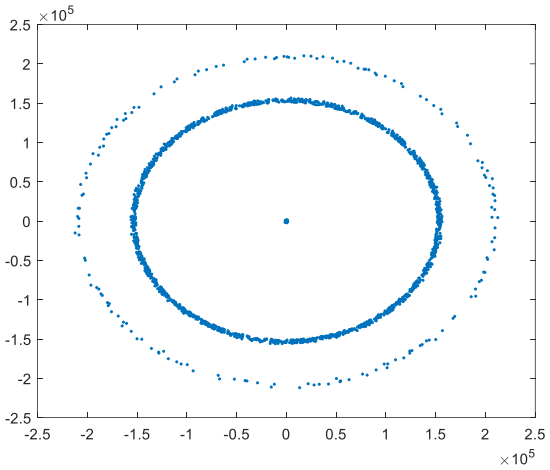


Рисунок 2.5 – Принятое созвездие без временной синхронизации

Созвездие на рисунке 2.5 выглядит в виде кольца из-за отсутствия временной синхронизации. С временной синхронизацией созвездие на рисунке 7 преобразуется в созвездие на рисунке 2.6.

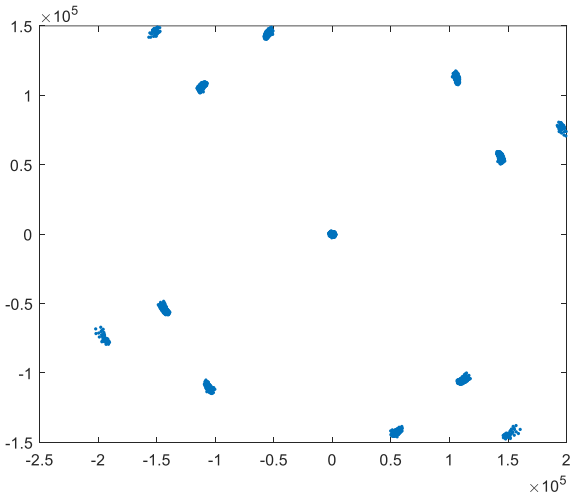


Рисунок 2.6 – Принятое созвездие с временной синхронизацией

Чтобы построить спектр такого сигнала, уберите знак процент (%) в начале 14 строчки и поставьте его в начале 13 строчки, тогда код будет выглядеть как на рисунке 2.7.

```
FoundScatter.m
1   clear all
2   close all
3   clc
4
5   Data = dlmread('2.txt');
6
7
8
9   Sig = (complex(Data(1:1:end,2),Data(1:1:end,1)));
10
11  figure
12  for i=3307:1:3307
13  %    plot(real(fft(Sig(i:1:i+4095))),imag(fft(Sig(i:1:
14       plot(abs(fft(Sig(i:1:i+4095))))
15      pause(0.1)
16  end
```

Рисунок 2.7 – Код для построения спектра сигнала

Спектр принятого сигнала при наличии и отсутствии временной синхронизации мало различен и выглядит как на рисунке 2.8.

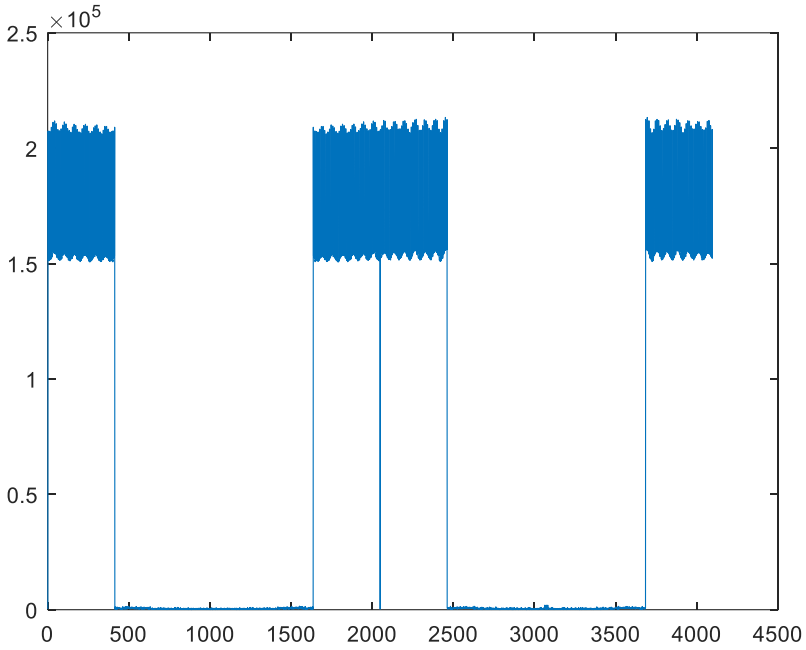


Рисунок 2.8 – Спектр принятого сигнала

Работа № 11

«Фильтрация, удаление зеркальной копии сигнала»

Цель работы: реализация модуля фильтрации для удаления зеркальной копии сигнала.
Задачи работы:
1) Генерация HDL кода КИХ-фильтра в MATLAB;
2) Подключение сгенерированого кода в проект;
3) Выполнить верификацию модуля.

1. Теоретическая часть

Фильтр – это устройство для выделения желательных компонентов спектра электрического сигнала.

Существует два основных вида фильтра: аналоговый и цифровой.

Главное отличие данных видов в том, что цифровой фильтр представляет из себя систему, которая выполняет математические операции над дискретизированным сигналом. Аналоговый фильтр обычно представляет собой электронную схему, работающую с аналоговыми (непрерывными) сигналами.

Цифровые фильтры имеют следующие параметры:
- Передаточная функция;
- АЧХ и ФЧХ;
- Импульсная характеристика;
- Переходная функция, групповая и фазовая задержки и т.д.;
- Архитектура, эффекты квантования.

С точки зрения частотной избирательности существует 6 типов фильтров, показанных на рисунке 1.1.

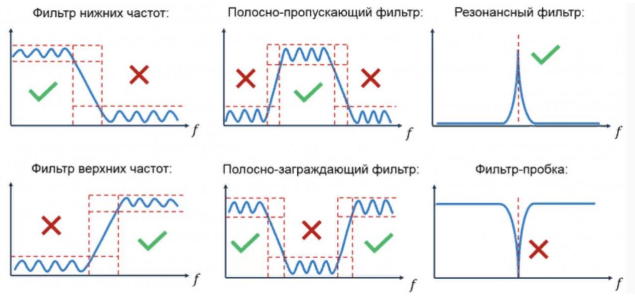


Рисунок 1.1 – Типы частотно-избирательных фильтров

Требования к частотной избирательности фильтра, в зависимости от накладываемой на него задачи, можно записать в виде спецификации к его частотным характеристикам.

У всех фильтров можно выделить три основные полосы: полосу пропускания, полосу перехода, полосу заграждения. На рисунке 1.2 представлена спецификация для фильтра нижних частот (ФНЧ).

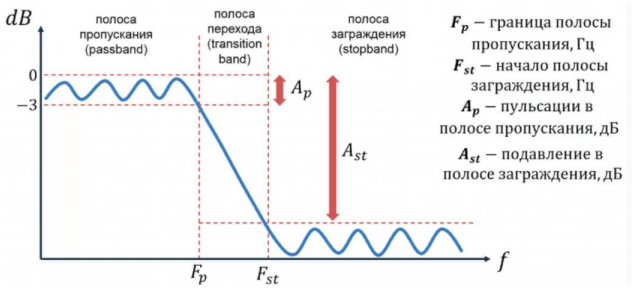


Рисунок 1.2 – Спецификация ФНЧ

Полоса пропускания фильтра – диапазон частот, в котором сигнал проходит без затухания (в идеале). Границей полосы пропускания для ФНЧ является частота F_p, которая отсчитывается по уровню -3 дБ от максимума.

Полоса заграждения – диапазон частот, в котором происходит ослабление сигнала не менее чем на A_{st} дБ. Начинается полоса заграждения с частоты F_{st}.

Между этими двумя полосами находится переходная полоса, в которой постепенно происходит увеличение затухания. Ширина данной полосы зависит от порядка фильтра – чем больше порядок, тем уже полоса. Порядок фильтра – это количество коэффициентов фильтров, оно напрямую связано с количеством умножителей, необходимых для реализации.

С точки зрения импульсной характеристики существует 2 типа фильтров: с конечной импульсной характеристикой (КИХ) и с бесконечной импульсной характеристикой (БИХ).

КИХ фильтр – нерекурсивный, это значит, что для вычисления значения на выходе фильтра используется только текущее и задержанные значения входа. Схема такого фильтра не имеет обратных связей (рисунок 1.3).

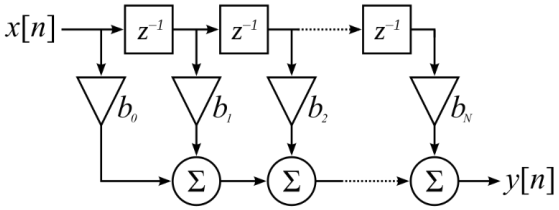


Рисунок 1.3 – Схема КИХ фильтра

Данный тип фильтров имеет следующие преимущества:

- Могут иметь линейную фазу, что сильно упрощает компенсацию фазового набега;
- всегда устойчивы;

Основной их недостаток – КИХ дороже в реализации, чем БИХ фильтры, со схожей АЧХ.

БИХ фильтры – рекурсивные, для вычисления значения на выходе фильтра используются как значения входа, так и задержанные значения выхода. Схема фильтра представлена на рисунке 1.4.

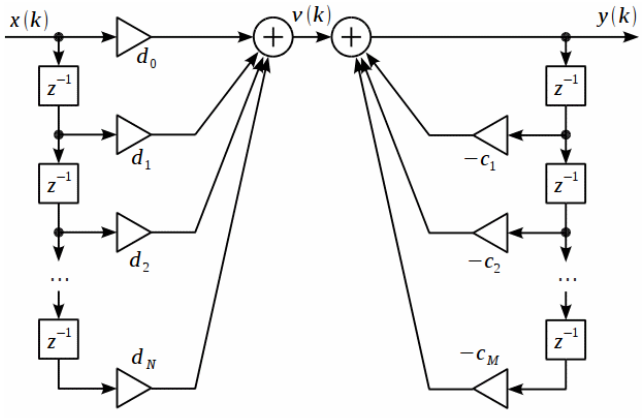


Рисунок 1.4 – Схема БИХ фильтра

К преимуществам данного фильтра относится:

- относительная простота в реализации, по сравнению с КИХ-фильтрами;
- относительная простота синтеза на основе аналоговых прототипов.

В качестве недостатков:

- могут быть неустойчивыми;

- не обладают линейной фазой;
- нет возможности сформировать произвольную АЧХ и ФЧХ.

Синтезируются БИХ-фильтры при помощи преобразования непрерывной передаточной характеристики аналогового прототипа в дискретную характеристику цифрового фильтра.

2. Практическая часть

1) Запустите MATLAB и перейдите во вкладку APPS. В списке приложений найдите Filter Designer и запустите его. Откроется окно как на рисунке 2.1

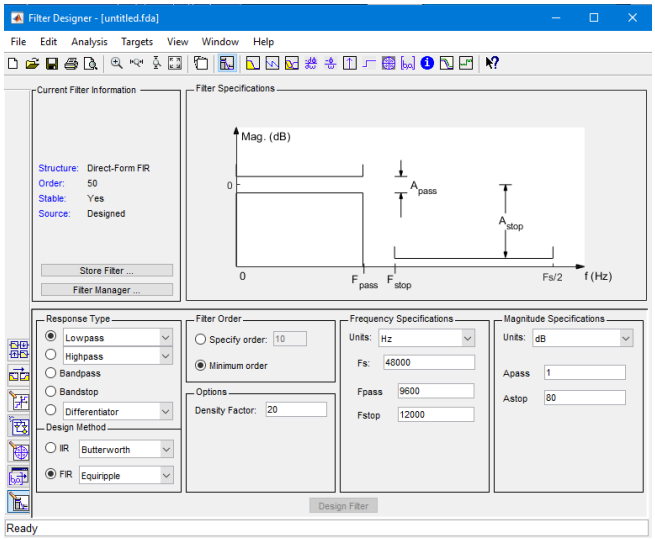


Рисунок 2.1 – Окно приложения Filter Designer

2) Установите параметры фильтра, представленные в таблице 2.1.

Таблица 2.1 – Параметры фильтра

Название параметра	Значение
Типа фильтра	Оконный ФНЧ
Тип окна	Хемминга
Порядок фильтра	47
Частота дискретизации	40 МГц
Частота среза	5,5 МГц

После установки параметров нажмите кнопку Design Filter, расположенную в нижней части окна. На рисунке 2.2 представлено окно с сгенерированным фильтром.

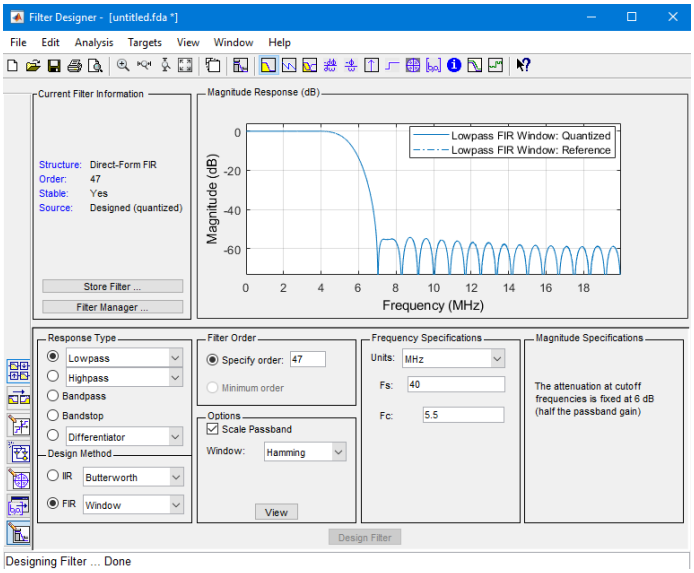


Рисунок 2.2 – Сгенерированный фильтр

3) Третьим этапом настройте параметры квантования. Для этого перейдите во вкладку Set quantization parameters, как показано на рисунке 2.3.

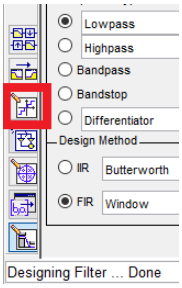


Рисунок 2.3 – Расположение вкладки Set quantization parameters

В открывшейся вкладке параметр Filter arithmetic выставите Fixed-point, после чего появятся множество дополнительных параметров, параметр Filter precision выставите Specify all. Следующим шагом перейдите во вкладку Input/Output, как на рисунке 2.4.

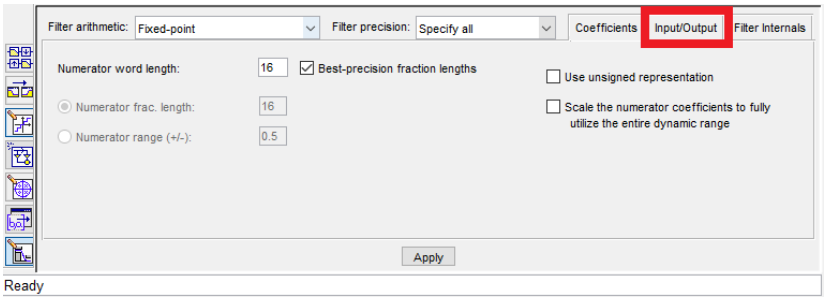


Рисунок 2.4 – Расположение вкладки Input/Output

В данной вкладке настраивается разрядность входных и выходных значений. Разрядность входных и выходных чисел равна 14, один разряд отводится под знак. Выставите параметрам Input word length и Output word length значение 14, Input fraction length и Output fraction length значение 13, параметрам Input range и Output range установите значение 1. После установки данных параметров нажмите кнопку Apply в нижней части окна, как показано на рисунке 2.5.

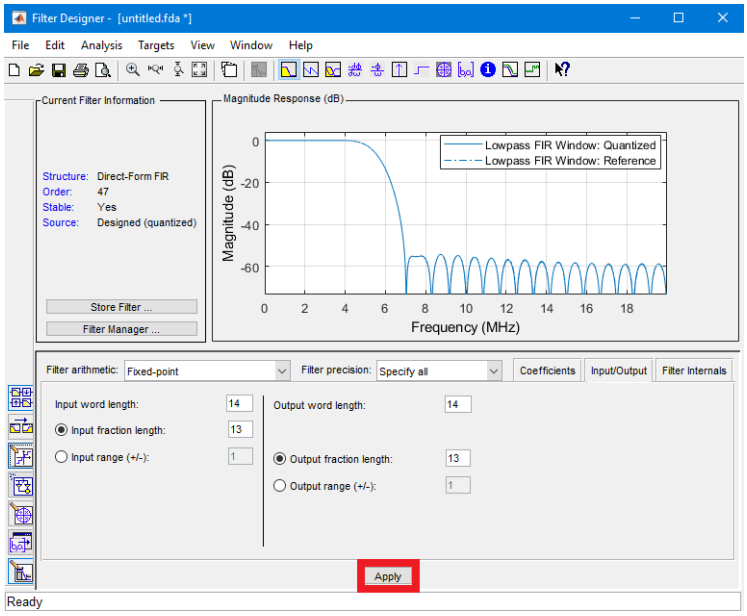


Рисунок 2.5 – Вкладка разрядности внешних портов

4) Далее сгенерированный фильтр экспортируйте в HDL. Для этого в верхней части окна во вкладке targets выберите Generate HDL. Откроется новое окно с параметрами генерации HDL. Выставите следующие параметры: Language – Verilog, Folder – Выберите папку «папка с вашим проектом/v/RX». Остальные параметры выставите как на рисунке 2.6.

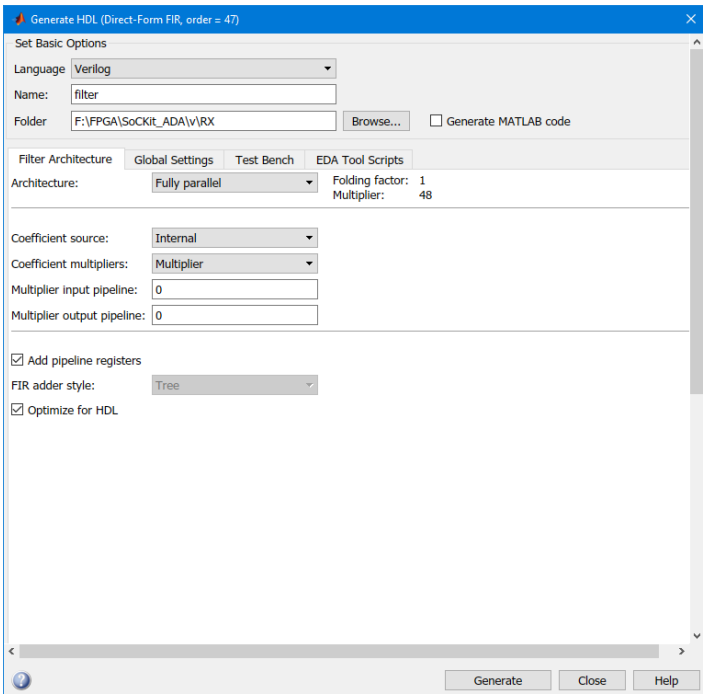


Рисунок 2.6 – Настройка генерации HDL

Так же необходимо отключить генерацию дополнительных файлов, для этого во вкладках Test Bench и EDA Tool Scripts уберите галочку напротив HDL test bench и Generate EDA scripts.

После того как вы установили все параметры, нажмите кнопку Generate в нижнем правом углу окна.

После генерации фильтра в командном окне появится сообщение об успешной генерации фильтра, как показано на рисунке 2.7.

```
### Starting Verilog code generation process for filter: filter_Hamming
### Generating: F:\FPGA\SoCKit_ADA\v\RX\filter_Hamming.v
### Starting generation of filter_Hamming Verilog module
### Starting generation of filter_Hamming Verilog module body
### Successful completion of Verilog code generation process for filter: filter_Hamming
### HDL latency is 8 samples
```

Рисунок 2.7 – Сообщение успешной генерации фильтра

5) Пятым шагом необходимо добавить сгенерированный файл в ваш проект. Для этого вернитесь в Quartus Prime. В верхней части окна во вкладке Project выберете Add/Remove File in Project. В открывшемся окне необходимо выбрать сгенерированный файл, для этого напротив File name после поля ввода нажмите на кнопку, выделенную на рисунке 2.8, для выбора файла.

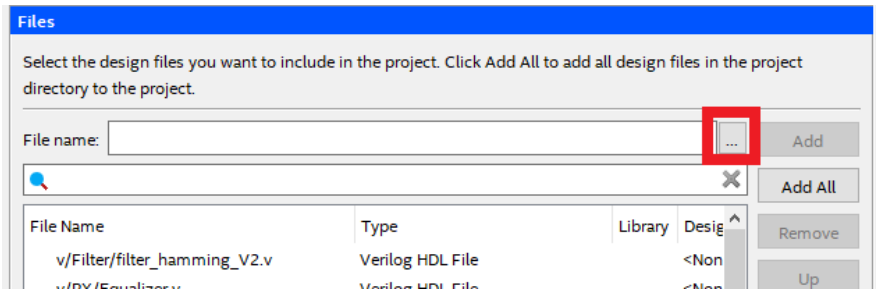


Рисунок 2.8 – Выбор файла

После найдите сгенерированный файл «filter.v», выделите его и нажмите кнопку открыть в нижнем правом углу окна.

После выбранный файл добавиться в список файлов. Нажмите кнопку Apply в нижнем правом углу окна для добавления файла в проект. Закройте окно добавление файла и вернитесь в главное окно. В навигации проекта должен появиться ваш сгенерированный файл, как на рисунке 2.9.

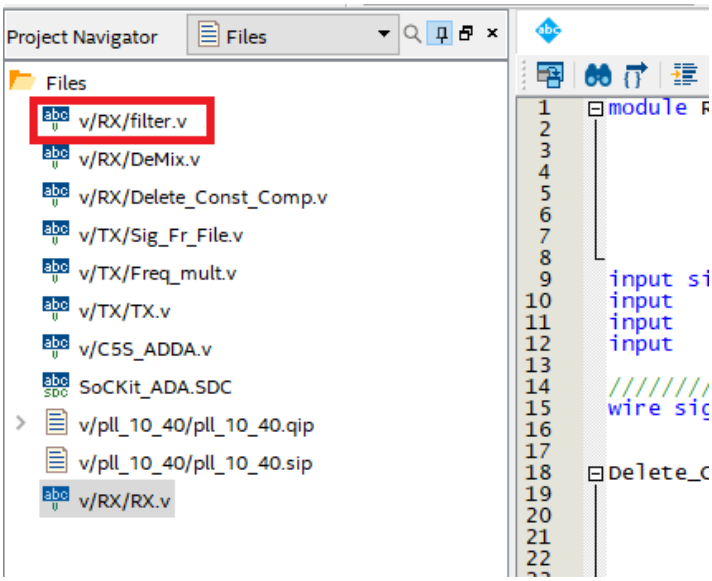


Рисунок 2.9 – Модуль в навигации проекта

6) Следующим этапом пропишите подключение сгенерированного фильтра к предыдущему модулю. Для этого откройте файл RX.v и создайте две 14-разрядные шины Filter_Decimator_Data_I и Filter_Decimator_Data_Q.

```
wire signed [13:0] Filter_Decimator_Data_I;
```

```
wire signed      [13:0]          Filter_Decimator_Data_Q;
```

Внешние порты сгенерированного фильтра представлены в таблице 2.2.

Таблица 2.2 – Описание портов и параметров

Название	Тип	Разрядность	Описание
clk	input	[0:0]	Входной порт тактирования
reset	input	[0:0]	Входной порт сброса
clk_enable	input	[0:0]	Входной порт активации, для работы необходимо единичный уровень на входе
filter_in	output wire signed	[13:0]	Входной порт данных
filter_out	output wire signed	[13:0]	Выходной порт данных

Для реальной и мнимой части сигнала необходимо использовать два различных фильтра из одного экземпляра. Код подключения прописан ниже.

```
filter filter_block_I(
                        .clk            (CLK_40),
                        .reset          (~reset),
                        .clk_enable     (1),
                        .filter_in      (DeMix_Filter_Data_I),
                        .filter_out     (Filter_Decimator_Data_I));

filter filter_block_Q(
                        .clk            (CLK_40),
                        .reset          (~reset),
                        .clk_enable     (1),
                        .filter_in      (DeMix_Filter_Data_Q),
                        .filter_out     (Filter_Decimator_Data_Q));
```

7) Скомпилируйте проект. Откройте Signal Tap и добавьте сигнал filter_out с модулей filter_block_I и filter_block_Q (Рисунок 2.10).

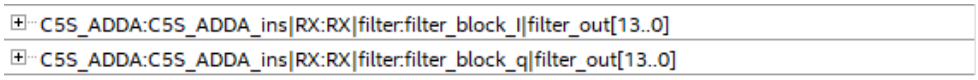


Рисунок 2.10 – Названия сигналов в Signal Tap

8) Скомпилируйте проект и загрузите прошивку в плату. После запустите signal tap и остановите для сохранения полученных отсчетов, сгенерируйте файл с отчетами (ПКМ => Create Signal Tap List File). На рисунке 2.11 показан сгенерированный файл с отсчетами.

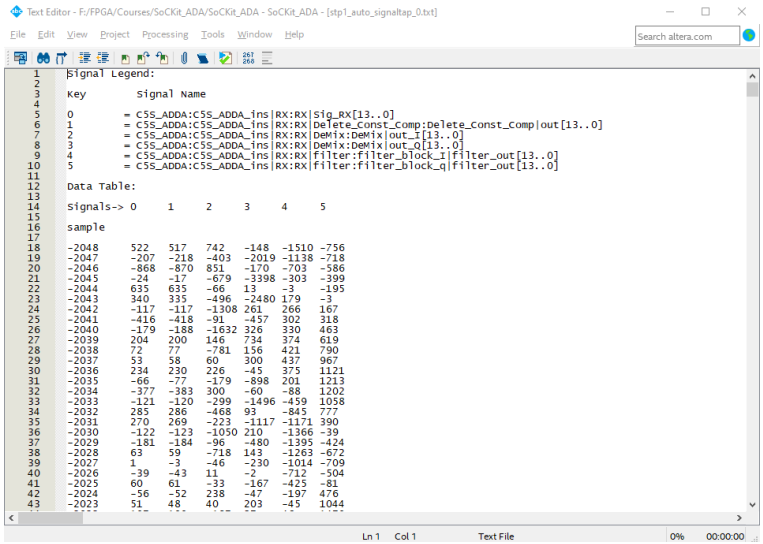


Рисунок 2.11 – Файл со сохраненными отсчетами

9) Откройте папку Verification, которая располагается в папке вашего проекта. Очистите содержимое файла 2.txt и скопируйте в файл отсчеты с выходов фильтра. Откройте файл «FoundScatter.m» в MATLAB. Уберите знак процента (%) с 13 строчки, и поставьте его в 14, как показано на рисунке 2.12.

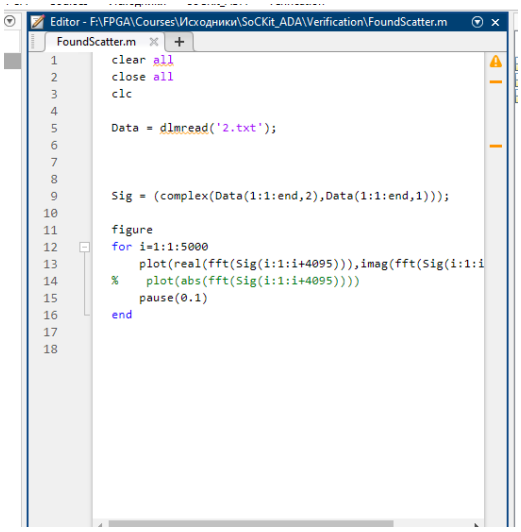


Рисунок 2.12 – Измененный код

Запустите код нажатием клавиши F5. Если созвездие, построенное на основе ваших отсчетов, выглядит как на рисунке 2.13, то фильтр сформирован верно и без ошибок.

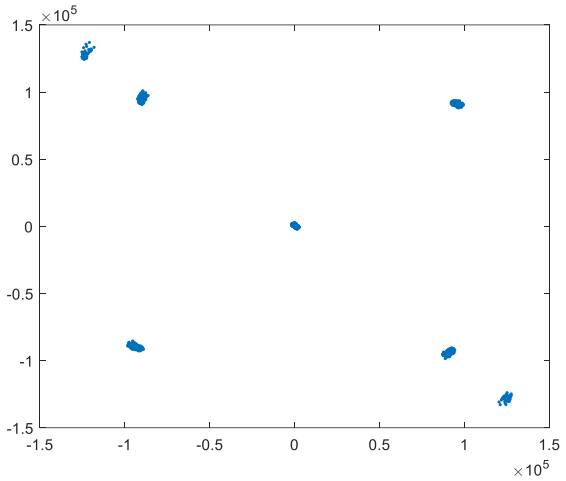


Рисунок 2.13 – Созвездие сигнала после фильтра

Постройте спектр сигнала. Спектр сигнала после фильтра должен выглядеть как на рисунке 2.14.

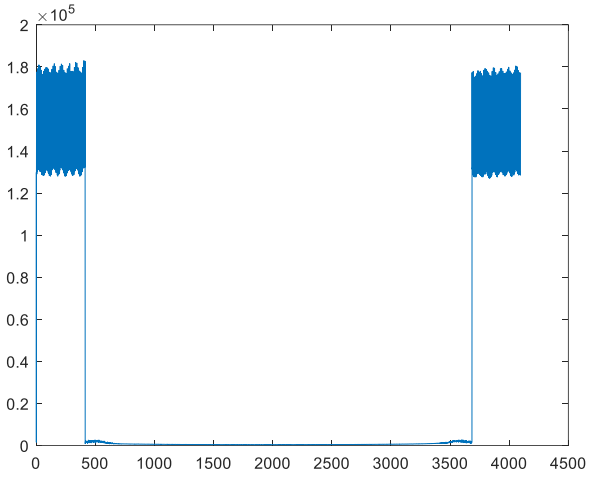


Рисунок 2.14 – Спектр сигнала после фильтра

Работа № 12

«Децимация сигнала»

Цель работы: реализация модуля децимации принимаемого сигнала.
Задачи работы:
1) Уменьшить частоту дискретизации принимаемого сигнала в N раз;
2) Выполнить верификацию модуля.

1. Теоретическая часть

Децимация – это уменьшение частоты дискретизации сигнала во временной области путем прореживания его отсчетом с определенным шагом. Прореживание отсчетов происходит путем выборочного удаления из исходного массива или на основе частичного суммирования в фиксированных временных интервалах – стробах.

При децимации удалением отсчетов сигнала из исходной последовательности отсчетов a_0, a_1, a_2 …, берется каждый N-й отсчет, где N – целое число. В итоге исходная последовательность преобразуется в a_0, a_N, a_{2N} …

На рисунке 1.1 представлен пример сигнала до и после децимаци.

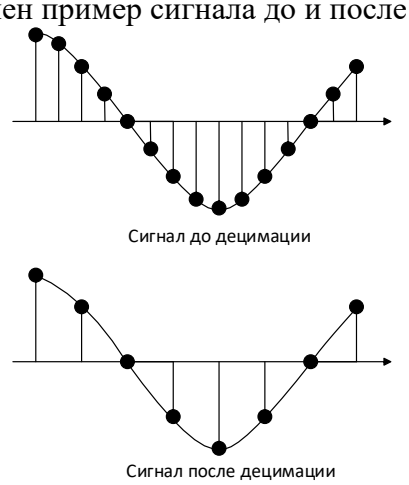


Рисунок 1.1 – Сигнал до и после децимации

2. Практическая часть

1) Откройте проект и создайте новый файл *Verilog HDL File*. Первой строчкой в файле необходимо задать название модуля (Decim) и внешние порты. Описание портов представлено в таблице 2.1.

Таблица 2.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
in_I	input signed	[13:0]	Входной порт реальной части данных
in_Q	input signed	[13:0]	Входной порт мнимой части данных
out_I	output wire signed	[13:0]	Выходной порт реальной части сигнала
out_Q	output wire signed	[13:0]	Выходной порт мнимой части сигнала

Код инициализации модуля и портов ввода/вывода представлен ниже.

```
module Decim(
		iclk,
		irst,
		in_I,
		in_Q,
		out_I,
		out_Q);
```

2) Следующим шагом опишите внешние порты, внутренние регистры и шины. В таблице 2 представлено описание внутренних регистров и шин.

Таблица 2.2 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
counter	reg	[1:0]		Регистр, выступающий в роли счетчика.

Код для описания внешних портов, внутренних регистров и шин представлен ниже.

```
input			iclk;
input			irst;
input		[13:0]	in_I;
input		[13:0]	in_Q;
output reg	[13:0]	out_I;
output reg	[13:0]	out_Q;

reg		[ 1:0]	Counter;
```

3) Третьим этапом задайте нулевое начальное состояние счетчику, выходным регистрам (out_I, out_Q) и реализуйте счет счетчика с синхронным сбросом по сигналу irst и с синхронной работой по положительному фронту сигнала iclk. Код данных операций представлен ниже.

```
initial
begin
	Counter		<=  2'd0;
	out_I		<= 14'd0;
	out_Q		<= 14'd0;
end

always@(posedge iclk)
begin
	if(irst)
		Counter <= 2'd0;
	else
		Counter <= Counter+1'b1;
end
```

4) Четвертым этапом реализуйте операцию децимации. Коэффициент децимации равен 4, это значит, что значение на выходе меняется раз в 4 такта относительно входного. Выбор фазы децимации производится за счет значения счетчика. Опорное значение счетчика выставите 2’b10. Код данной операции представлен ниже.

```
always@(posedge iclk)
begin
        if(irst)
        begin
                out_I           <= 14'd0;
                out_Q           <= 14'd0;
        end
        else
                if(Counter == 2'b10)
                begin
                        out_I <= in_I;
                        out_Q <= in_Q;
                end
end
```

В конце пропишите строчку, обозначающую конец модуля.

```
endmodule
```

5) Сохраните файл как Decim.v. После откройте файл RX.v, создайте две 14-разрядные шины (Decimator_Correlator_Shifter_Data_I и Decimator_Correlator_Shifter_Data_Q) и пропишите подключение данного блока после блока преобразования частоты. К порту irst подключите шину reset, к iclk – CLK_40, in_I - Filter_Decimator_Data_I, in_Q - Filter_Decimator_Data_Q, а к портам out_I и out_Q - Decimator_Correlator_Shifter_Data_I и Decimator_Correlator_Shifter_Data_Q соответственно. Код для подключения представлен ниже.

```
////////////////// DECIMATOR //////////////////
wire signed [13:0] Decimator_Correlator_Shifter_Data_I;
wire signed [13:0] Decimator_Correlator_Shifter_Data_Q;

Decim Decim_Block(
                        .iclk           (CLK_40),
                        .irst           (~reset),
                        .in_I           (Filter_Decimator_Data_I),
                        .in_Q           (Filter_Decimator_Data_Q),
                        .out_I          (Decimator_Correlator_Shifter_Data_I),
                        .out_Q          (Decimator_Correlator_Shifter_Data_Q));
```

6) Сохраните все изменения и запустите компиляцию. После успешного завершения компиляции запустите Signal Tap. Добавьте сигналы out_I и out_Q с выхода блока децимации. На рисунке 2.1 показаны названия сигналов, которые необходимо добавить.

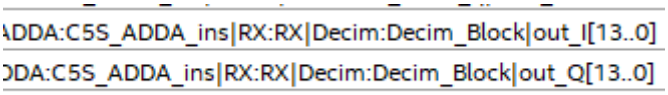


Рисунок 2.1 – Сигналы с выхода модуля децимации

7) Сравните сигналы с выхода фильтра и с дециматора. Отсчеты на выходе фильтра должны изменятся в 4 раза чаще, чем на выходе дециматора. Так же каждое четвертое значение на выходе фильтра должно совпадать со значением на выходе дециматора в следующий такт. На рисунке 2.2 показано сравнение сигналов с выхода фильтра и с выхода дециматора.

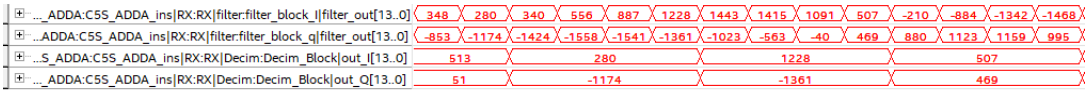


Рисунок 2.2 – Сравнение сигналов с фильтра и дециматора

8) Если все верно, то теперь необходимо изменить тактовый сигнал в Signal Tap, с 40 МГц на 10, так как после дециматора все блоки работают с частотой 10 МГц. Для этого откройте вкладку Setup (Рисунок 2.3).

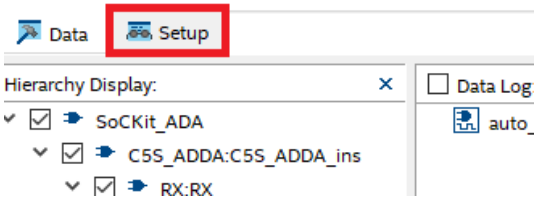


Рисунок 2.3 – Вкладка Setup

В настройке сигналов, напротив надписи Clock, измените сигнал со значения «…|outclk_1”ъ» на «…|outclk_0». В результате полное название сигнала будет выглядеть как на рисунке 2.4.

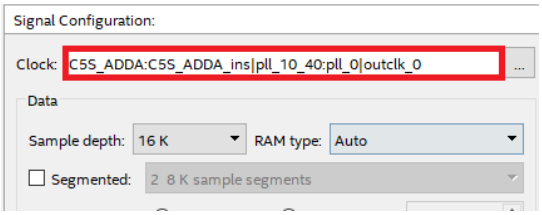


Рисунок 2.4 – Полное название тактового сигнала

9) Сохраните изменения и запустите компиляцию проекта. После компиляции загрузите прошивку в плату и сравните сигналы: сигналы с выхода фильтра и дециматора должны изменяться с одной частотой, однако значения их должны быть разными (Рисунок 2.5).

Рисунок 2.5 – Сравнение сигналов после изменения тактового сигнала

10) Сгенерируйте текстовый файл (ПКМ => Create Signal Tap List File). Скопируйте столбцы с сигналами с выхода блока децимации. После откройте файл «2.txt»,

расположенный в «папка вашего проектом/Verification/2.txt», удалите все содержимое файла и вставьте туда скопированные отсчеты.

11) После чего откройте файл «FoundScatter.m». Закомментируйте строчку 14 и удалите знак комментария вначале 13 строчки. После чего измените все числа 4095 на числа 1023, так как после децимации сигнала в 4 раза, сигнал имеет размерность не 4096 отсчетов, а 1024. Цикл for после изменения будет выглядеть как на рисунке 2.6.

```
figure
for i=1:1:5000
    plot(real(fft(Sig(i:1:i+1023))),imag(fft(Sig(i:1:i+1023))),'.')
%     plot(abs(fft(Sig(i:1:i+1023))))
    pause(0.1)
end
```

Рисунок 2.6 – Измененный цикл

12) Запустите код нажатием клавиши F5. При верной децимации сигнала его созвездие будет выглядеть как на рисунке 2.7

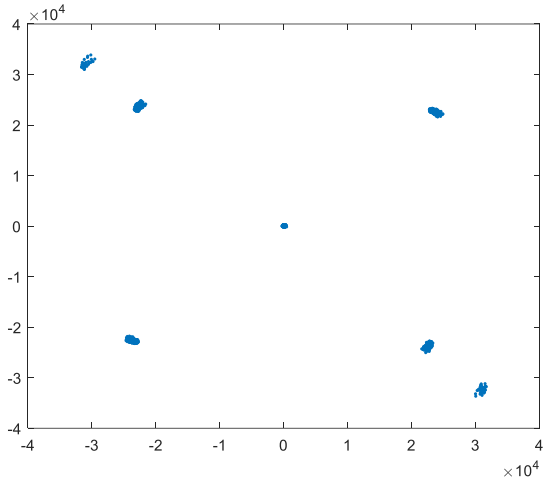


Рисунок 2.7 – Созвездие принятого сигнала при верной децимации

В случае если ваше созвездие представляет из себя несколько колец, как на рисунке 2.8, то по желанию можете выполнить шаги, описанные ниже.

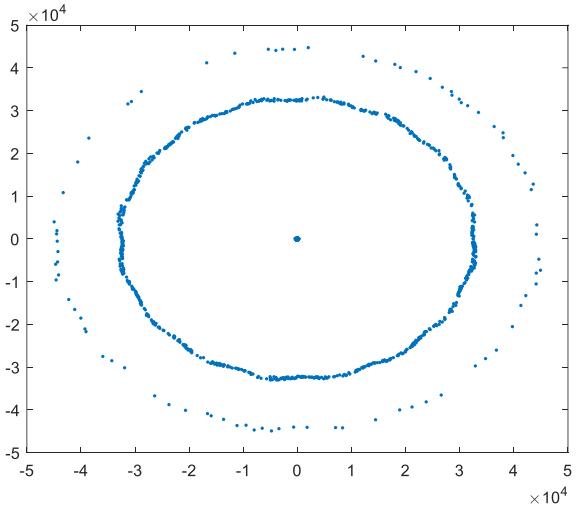
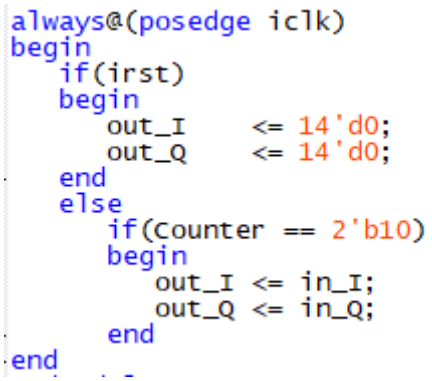


Рисунок 2.8 – Созвездие сигнала при ошибочной децимации

13) Для начала вернитесь в Quartus Prime и откройте файл с модулем дециматора. Найдите отрезок кода, отвечающий за децимацию сигнала (Рисунок 2.9).



```
always@(posedge iclk)
begin
    if(irst)
    begin
        out_I     <= 14'd0;
        out_Q     <= 14'd0;
    end
    else
        if(Counter == 2'b10)
        begin
            out_I <= in_I;
            out_Q <= in_Q;
        end
end
```

Рисунок 2.9 – Часть кода, отвечающая за децимацию

14) Для изменения фазы децимации необходимо изменить опорное значение, с которым сравнивается значение счетчика Counter. На рисунке 10 данное значение равно 2’b10. Выберите одно из 3х других значений (2’b00, 2’b01 или 2’b11) и пропишите его. Повторите шаги 9, 10 и 12.

Работа № 13

«Инверсия спектра»

Цель работы: реализая модуля инверсии спектра принимаемого сигнала.

Задачи работы:

1) Выполнить циклический сдвиг спектра принимаемого сигнала во временной области;

2) Выполнить верификацию модуля.

1. Практическая часть

1) Откройте проект и создайте новый файл Verilog HDL File. Первой строчкой в файле необходимо задать название модуля (FFT_Shift) и внешние порты. Описание портов представлено в таблице 1.1.

Таблица 1.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
in_I	input signed	[13:0]	Входной порт реальной части данных
in_Q	input signed	[13:0]	Входной порт мнимой части данных
out_I	output signed	[13:0]	Выходной порт реальной части сигнала
out_Q	output signed	[13:0]	Выходной порт мнимой части сигнала

Код инициализации модуля и внешних портов представлен ниже.

```
module FFT_Shift(
                iclk,
                irst,
                in_I,
                in_Q,
                out_I,
                out_Q);
```

2) Следующим шагом необходимо описать внешние порты, внутренние регистры и шины. В таблице 1.2 представлено описание внутренних регистров и шин.

Таблица 1.2 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
counter	reg	[0:0]		Регистр, выступающий в роли счетчика.
buf_I	reg signed	[13:0]	[1:0]	Регистр задержки реальной части сигнала, необходимый для синхронизации с коррелятором

Окончание таблицы 1.2

buf_Q	reg signed	[13:0]	[1:0]	Регистр задержки мнимой части сигнала, необходимый для синхронизации с коррелятором

Код для описания внешних портов, внутренних регистров и шин представлен ниже.

```
input                           iclk;
input                           irst;
input       signed      [13:0]  in_I;
input       signed      [13:0]  in_Q;
output      signed      [13:0]  out_I;
output      signed      [13:0]  out_Q;
reg         signed      [13:0]  buf_I       [1:0];
reg         signed      [13:0]  buf_Q       [1:0];
reg                             counter;
```

3) Третьим шагом задайте нулевые начальные состояния во внутренние регистры и реализуйте счетчик с синхронным сбросом по сигналу irst. Код представлен ниже.

```
initial
begin
        counter   <=  1'b0;
        buf_I[0] <= 14'd0;
        buf_Q[0] <= 14'd0;
        buf_I[1] <= 14'd0;
        buf_Q[1] <= 14'd0;
end

always@(posedge iclk or posedge irst)
begin
        if(irst)
                counter <= 1'b0;
        else
        counter <= ~counter;
end
```

4) Далее необходимо реализовать сдвиг данных в регистрах задержки (buf_I и buf_Q) и подсоединить выходные шины (out_I и out_Q) к крайней ячейки регистров с чередованием знаков в зависимости от значения счетчика (counter) при помощи assign. Код представлен ниже.

```
always@(posedge iclk)
begin
        buf_I[1] <= buf_I[0];
        buf_Q[1] <= buf_Q[0];

        buf_I[0] <= in_I;
        buf_Q[0] <= in_Q;
```

```
end

assign out_I = (counter) ? -buf_I[1]: buf_I[1];
assign out_Q = (counter) ? -buf_Q[1]: buf_Q[1];
```

В конце файла пропишите строчку окончания файла.

```
endmodule
```

5) Сохраните файл как FFT_Shift.v и откройте RX.v. Следующим шагом необходимо прописать подключение данного блока к основному проекту, для этого сначала создайте две 14 разрядные шины: Shifter_DeleteCP_Data_I и Shifter_DeleteCP_Data_Q. Далее после блока корреляции пропишите подключение шин к блоку в следующем порядке: к порту iclk подключите шину CLK_10, к irst – reset, in_I и in_Q - Decimator_Correlator_Shifter_Data_I и Decimator_Correlator_Shifter_Data_Q соответственно, out_I и out_Q - Shifter_DeleteCP_Data_I и Shifter_DeleteCP_Data_Q. Код подключения представлен ниже.

```
////////////////// FFT SHIFT //////////////////

wire signed [13:0] Shifter_DeleteCP_Data_I;
wire signed [13:0] Shifter_DeleteCP_Data_Q;

FFT_Shift FFT_Shift_Block(
                                    .iclk       (CLK_10),
                                    .irst       (~reset),
                                    .in_I
(Decimator_Correlator_Shifter_Data_I),
                                    .in_Q
(Decimator_Correlator_Shifter_Data_Q),
                                    .out_I      (Shifter_DeleteCP_Data_I),
                                    .out_Q      (Shifter_DeleteCP_Data_Q));
```

6) Сохраните изменения и скомпилируйте проект. После компиляции откройте Signal Tap Logic Analyzer и удалите все сигналы кроме модуля децимации. Добавьте сигналы out_I и out_Q с выхода модуля разворота спектра. Все сигналы, присутствующие в Signal Tap, представлены на рисунке 1.1.

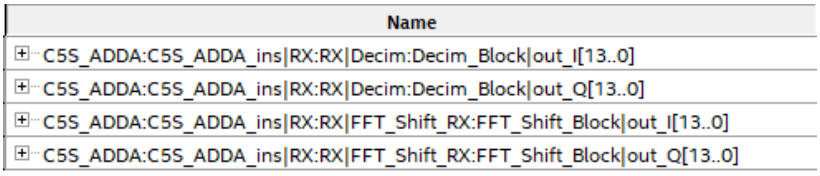


Рисунок 1.1 – Список сигналов в Signal Tap

Скомпилируйте проект и загрузите прошивку в плату. Сгенерируйте Signal Tap List. Скопируйте отсчеты в файл «2.txt», предварительно удалив все содержимое файла. Откройте файл «Found Scatter.m». Уберите комментарий в начале 14 строчки и поставьте в начало 13 строчки. Измененный цикл представлен на рисунке 1.2.

```
figure
for i=1:1:5000
%    plot(real(fft(Sig(i:1:i+1023))),imag(fft(Sig(i:1:i+1023))),'.')
    plot(abs(fft(Sig(i:1:i+1023))))
    pause(0.1)
end
```

Рисунок 1.2 – Измененный цикл

Спектр сигнала должен выглядеть как на рисунке 1.3.

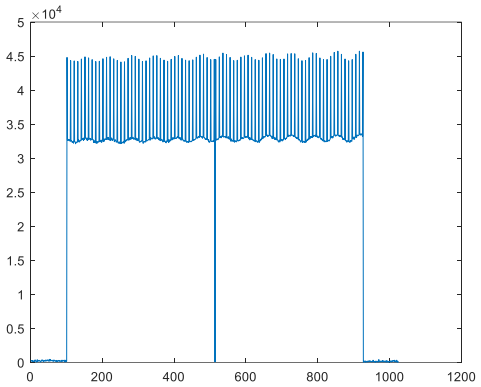


Рисунок 1.3 – Спектр принятого сигнала после операции разворота

Работа № 14

«Кадровая синхронизация. Коррелятор»

Цель работы: реализовать модуль кадровой синхронизации сигнала.

Задачи работы:

1) Реализовать расчёт коэффициента взаимной корреляции на ПЛИС;

2) Выполнить верификацию модуля.

1. Теоретическая часть

Коррелятор – устройство для автоматического вычисления взаимнокорреляционной функции.

Взаимнокорреляционной функция – метод оценки степени корреляции двух последовательностей.

Корреляция – статическая взаимосвязь двух или более случайных величин. Математической мерой корреляции двух случайных величин служит корреляционное соотношение или коэффициент корреляции.

В общем случае для непрерывных функций f(t) и g(t) взаимная корреляция определяется по формуле 1.1:

$$(f * g)_i \overset{def}{=} \sum_j f_j^* g_{i+j}, \quad (1.1)$$

где i – сдвиг между последовательностями относительно друг друга;

* – комплексное сопряжение.

2. Практическая часть

1) Откройте проект и создайте новый файл *Verilog HDL File*. Первой строчкой в файле необходимо прописать название модуля (Correl_Main) и внешние порты. Описание портов представлено в таблице 2.1.

Таблица 2.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
in_I	input signed	[13:0]	Входной порт реальной части данных
in_Q	input signed	[13:0]	Входной порт мнимой части данных
osop	output	[0:0]	Выходной порт начала информационных символов

Код названия модуля и инициализации внешних портов представлен ниже.

```
module Correl_Main (
                    in_I,
                    in_Q,
                    iclk,
                    irst,
                    osop);
```

2) На втором этапе пропишите две внутренние 23 разрядные шины Correlator_Result_I и Correlator_Result_Q. В таблице 2.2 представлено подробное описание внутренних шин.

Таблица 2.2 – Описание внутренних шин и регистров

Название	Тип	Разрядность	Описание
Correlator_Result_I	signed wire	[22:0]	Внутренняя шина с результатом корреляции реальной части
Correlator_Result_Q	signed wire	[22:0]	Внутренняя шина с результатом корреляции мнимой части.

Код для описания внешних портов, внутренних регистров и шин представлен ниже.

```
    input                   iclk;
    input                   irst;
    input signed    [13:0]  in_I;
    input signed    [13:0]  in_Q;
    output                  osop;
    wire signed     [22:0]  Correlator_Result_I;
    wire signed     [22:0]  Correlator_Result_Q;
```

3) Далее необходимо прописать подключение трёх модулей из двух прототипов: Correl_I и Correl_Q на основе Correl, и Threshold на основе Threshold. Блоки Correl_I и Correl_Q служат для вычисления коэффициента корреляции между принятым сигналом и преамбулой в памяти приемника. Третий блок суммирует результаты корреляции и проверяет, превысил ли результат суммирования порог, если результат суммирования больше порога, то на выходе данного сигнала появляется sop сигнал, сигнализирующий о конце преамбулы. В таблицах 2.3 и 2.4 представлены порты прототипов Correl и Threshold соответственно.

Таблица 2.3 – Описание портов и параметров прототипа Correl

Название	Тип	Разрядность	Описание
iclk	input		Входной порт сигнала тактирования
irst	input		Входной порт сигнала сброса
in	input signed	[13:0]	Входной порт принятого сигнала
result	output	[22:0]	Вычисленное значение коэффициента корреляции
N_Samp	parameter		Длина опорного сигнала для корреляции, в данном случае 1024
N_Layer	parameter		Количество слоев в древовидном сумматоре. Равно логарифму от длины опорного сигнала по основанию 2.

Таблица 2.4 – Описание портов и параметров прототипа Threshold

Название	Тип	Разрядность	Описание
iclk	input		Входной порт сигнала тактирования

Окончание таблицы 2.4

Res1	input signed	[22:0]	Вычисленное значение коэффициента корреляции
Res2	input signed	[22:0]	Вычисленное значение коэффициента корреляции
sop	output reg		Выходной порт начала информационных символов

Код подключения представлен ниже. После подключения пропишите окончание модуля.

```
Correl Correl_I(
                .iclk       (iclk),
                .in         (in_I),
                .irst       (irst),
                .result     (Correlator_Result_I) );
Correl Correl_Q(
                .iclk       (iclk),
                .in         (in_Q),
                .irst       (irst),
                .result     (Correlator_Result_Q));
Threshold Threshold(
                .iclk   (iclk),
                .Res1   (Correlator_Result_I),
                .Res2   (Correlator_Result_Q),
                .sop    (osop));
```

4) Сохраните файл как Correl_Main.v.

5) Четвертым этапом реализуйте прототип Correl. В данном блоке производится расчет коэффициента корреляции одной из составляющей входного сигнала с реальной частью преамбулы.

a. Создайте файл Verilog HDL File и инициализируйте входные порты и параметры, указанные в таблице 2.3. Код инициализации представлен ниже.

```
module Correl#(parameter N_Samp   = 1024,
               parameter N_Layer = $clog2(N_Samp))
               (
               iclk,
               irst,
               in,
               result);
```

b. Далее опишите порты, внутренние шины и регистры. Список внутренних регистров и шин представлен в таблице 2.5. Код для описания портов, внутренних шин и регистров представлен после таблицы.

Таблица 2.5 – Описание внутренних шин и регистров

Название	Тип	Разрядность	Размер	Описание
Connect_wire	signed wire	[22:0]	[N_2-1:1] [N-1:0]	Внутренняя шина, соединяющая блоки суммирования в один древовидный сумматор
result_signed	signed wire	[22:0]		Вычисленный коэффициент корреляции
Bufer_Preamb	reg signed	[15:0]	[N-1:0]	Регистр, содержащий преамбулу
Bufer_Sig_Input	reg signed	[13:0]	[N-1:0]	Сдвиговый регистр, содержащий 1024 отсчета входного сигнала
Bufer_Sum	reg signed	[15:0]	[N-1:0]	Буфер, содержащий отсчеты преамбулы с измененными знаками
k,l	integer			Переменные необходимые для генерации двух сдвига в регистрах

```
        input                           iclk;
        input                           irst;
        input       signed      [13:0]  in;
        output                  [22:0]  result;
        wire        signed      [22:0]  Connect_wire        [N_Layer-1:1][N_Samp-
1:0];
        wire        signed      [22:0]  result_signed;
        reg         signed      [15:0]  Bufer_Preamb        [N_Samp -1:0];
        reg         signed      [15:0]  Bufer_Sig_Input     [N_Samp -1:0];
        reg         signed      [22:0]  Bufer_Summ          [N_Samp -1:0];
        integer     k,l;
```

c. Следующим шагом реализуйте запись отсчетов реальной части преамбулы из файла в регистр памяти. Для этого воспользуйтесь командой $readmemb(Address, Memory), где Address – полное имя файла, указанное в кавычках, Memory – название регистра памяти. В качестве файла с отчетами преамбулы используйте Preamb_I.txt, расположенный в «Папка с проектом\input_files\». В качестве регистра памяти используйте регистр Bufer_Preamb.

```
        initial
        begin
        $readmemb(“Путь_до_проекта\\SoCKit_ADA\\InputData\\Preamb_I.txt",Bufer_Preamb
);
        end
```

d. Далее реализуйте сдвиг регистра Bufer_Sig_Input, используя цикл for для инициализации данной операции.

```
        always@(posedge iclk)
        begin
```

```
        for(k=0;k<N_Samp-1;k=k+1)
        begin
            Bufer_Sig_Input[k]=Bufer_Sig_Input[k+1];
        end
        Bufer_Sig_Input[N_Samp-1]<=in;
    end
```

e. Следующим шагом выполните проверку знака каждого отсчета входного сигнала в Bufer_Sig_Input, если отсчет имеет отрицательный знак, то необходимо взять ячейку с таким же адресом, хранящую преамбулу, и записать её в третий буфер (Bufer_Sum) с инвертированием знака. Если знак у принятого отсчета положительный, то отсчет преамбулы записывается без изменения знака. Необходимо расширить разрядность преамбулы до 23 символов, скопировав 15 бит несколько раз. Реализация данной процедуры представлена ниже.

```
    always@(posedge iclk)
    begin
        for(l=0;l<N_Samp;l=l+1)
        begin
            if (Bufer_Sig_Input[l]<0)
            begin
                Bufer_Summ[l]<=-
({Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer
_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l]});
            end
            else
            begin
                Bufer_Summ[l]<=
({Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer
_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l][15],Bufer_Preamb[l]});
            end
        end
    end
```

f. Далее реализуйте древовидный сумматор по такому же принципу, как и в блоке удаления постоянной составляющей. На вход блоков суммы, расположенных в первом слое, подключите регистр Buffer_Summ. Последний блок так же пропишите вручную. Код генерации сумматора представлен ниже.

```
    generate
    genvar i,j;
        for(i = 0;i<=N_Layer-2;i=i+1)
        begin: Summ_B1
            for(j = 0;j<N_Samp;j=j+2**(i+1))
            begin: Summ_B2
            if(i == 0)
            begin
            Summ #(23) Summ_Bl(
                        .Summ_1     (Bufer_Summ[j]),
                        .Summ_2     (Bufer_Summ[j+2**(i)]),
```

```
                        .out            (Connect_wire[i+1][j]));
                end
                else
                begin
                Summ #(23) Summ_Bl(
                        .Summ_1         (Connect_wire[i][j]),
                        .Summ_2         (Connect_wire[i][j+2**(i)]),
                        .out            (Connect_wire[i+1][j]));
                end
                end
        end
endgenerate

Summ #(23) Summ_lst(
                .Summ_1         (Connect_wire[N_Layer-1][0]),
                .Summ_2         (Connect_wire[N_Layer-1][2**(N_Layer-1)]),
                .out            (result_signed)
                );
```

6) После при помощи конструкции assign соедините выходной порт result с модулем сигнала result_signed.

```
assign result = (result_signed < 0) ? -result_signed:result_signed;
```

7) В конце пропишите окончание модуля и сохраните файл как Correl.v.

```
endmodule
```

8) Создайте файл Verilog HDL File. Согласно таблице 2.4 инициализируйте модуль (Treshold) и порты ввода-вывода. После добавьте описание к портам.

```
module Threshold(       Res1,
                        Res2,
                        sop,
                        iclk);
input [22:0]    Res1;
input [22:0]    Res2;
input           iclk;
output reg      sop;
```

После с помощью конструкции always реализуйте генерацию сигнала sop, когда сумма портов Res1 и Res2 будет больше 300000. В конце файла добавьте окончание модуля endmodule.

```
always@(posedge iclk)
        if (Res1 + Res2 > 300000)
                sop <= 1'b1;
        else
                sop <= 1'b0;
endmodule
```

9) Сохраните файл как Treshold.v. Откройте файл RX.v и пропишите подключения блока корреляции после блока разворота спектра. Предварительно необходимо создать одноразрядную шину Correlator_Sop. Код подключения блока представлен ниже.

```
////////////////// CORRELATOR //////////////////
wire                                    Correlator_Sop;

Correl_Main Correl_Main_Block (
                                        .in_I
(Decimator_Correlator_Shifter_Data_I),
                                        .in_Q
(Decimator_Correlator_Shifter_Data_Q),                                  .iclk
(CLK_10),
                                        .irst           (~reset),
                                        .osop           (Correlator_Sop));
```

10) Сохраните изменения и скомпилируйте проект. Откройте Signal Tap и добавьте туда сигналы с внутреннего блока коррелятора Treshold: Res1, Res2 и sop. Список сигналов, которые необходимо добавить, представлен на рисунке 2.1.

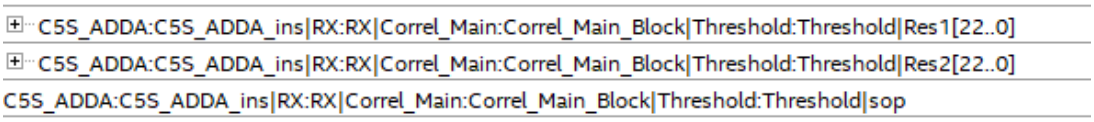


Рисунок 2.1 – Список добавленных сигналов

Сохраните изменения и скомпилируйте проект еще раз. Загрузите прошивку в плату и запустите анализатор. Сигнал sop должен повторятся раз с шагом 4224 отсчета. Сигналы из Signal Tap Analyzer представлены на рисунке 2.2.

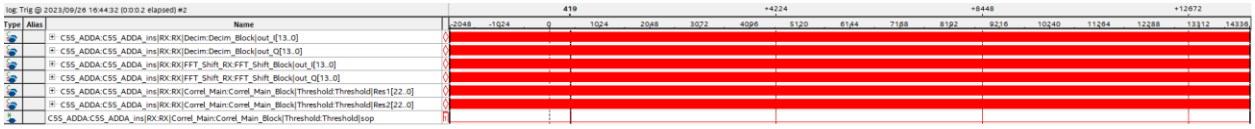


Рисунок 2.2 – Сигналы в Signal Tap Analyzer

11) Далее сгенерируйте Signal Tap List и скопируйте отсчеты. Откройте файл «3.txt», удалите все содержимое файла и вставьте туда скопированные отсчеты. После чего откройте файл «Sum.m», запустите его нажатием клавиши F5. После запуска построится график корреляции. На рисунке 2.3 представлен пример корреляции.

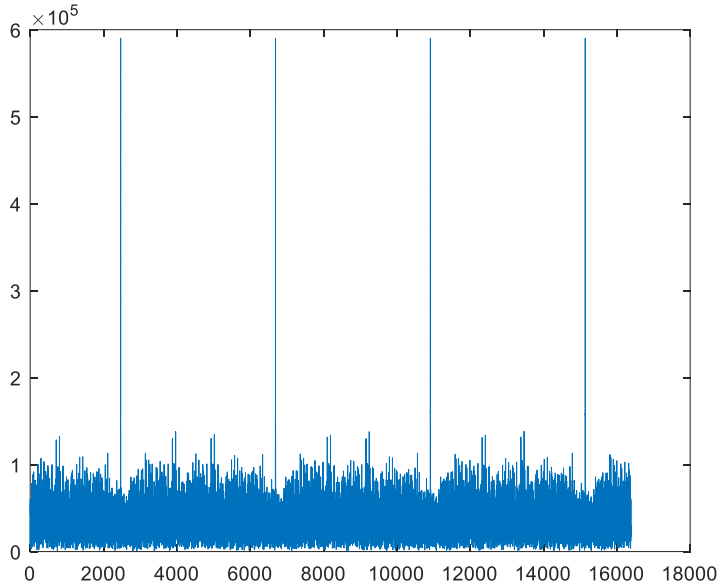


Рисунок 2.3 – График корреляции

Работа № 15

«Удаление циклического префикса»

Цель работы: реализовать модуль удаления циклического префикса OFDM сигнала.

Задачи работы:

1) Реализовать счётчик отсчётов OFDM символа и сгенерировать сигнал начала символа OFDM;

2) Удалить символы преамбулы в принимаемом кадре.

3) Выполнить верификацию модуля

1. Практическая часть

1) Откройте проект и создайте новый файл Verilog HDL File. Первой строчкой в файле необходимо задать название модуля и внешние порты. Описание портов представлено в таблице 1.1.

Таблица 1.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
in_I	input signed	[13:0]	Входной порт реальной части данных
in_Q	input signed	[13:0]	Входной порт мнимой части данных
sop	input		Входной сигнал, означающий начало кадра, состоящего из преамбулы и трех информационных символов
out_I	output reg signed	[15:0]	Выходной порт реальной части сигнала
out_Q	output reg signed	[15:0]	Выходной порт мнимой части сигнала
osop	output reg		Выходной сигнал, означающий начало символа

Код инициализации модуля и внешних портов представлен ниже.

```
module Del_CP(
                iclk,
                irst,
                in_I,
                in_Q,
                sop,
                out_I,
                out_Q,
                osop);
```

2) Следующим шагом необходимо описать внешние порты, внутренние регистры и шины. В таблице 1.2 представлено описание внутренних регистров и шин.

Таблица 1.2 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
counter	reg	[13:0]		Регистр, выступающий в роли счетчика

Код для описания внешних портов, внутренних регистров и шин представлен ниже.

```
input                       sop;
input                       irst;
input                       iclk;
input        signed [13:0]  in_I;
input        signed [13:0]  in_Q;
output reg   signed [15:0]  out_I;
output reg   signed [15:0]  out_Q;
output reg                  osop;
reg                  [13:0] counter;
```

3) Третьим шагом необходимо реализовать счетчик. Принятый кадр состоит из четырех символов, один из которых – преамбула, а три оставшихся – информационные. Также сигнал sop с коррелятора совпадает с окончанием преамбулы, необходимо чтобы счетчик производил сброс либо при достижении числа 4223 (4 символа по 1056 отсчетов), либо при сигнале sop, либо при сигнале сброса irst. Код представлен ниже.

```
always@(posedge iclk)
begin
        if(counter == 14'd4223 || sop || irst)
                counter<=14'd0;
        else
                counter<=counter+1'b1;
end
```

4) Далее необходимо удалить циклический префикс и преамбулу из принятого сигнала. Так как опорный счетчик counter производит сброс при окончании преамбулы, то его начало счета совпадает с началом циклического префикса первого информационного символа. Длина циклического префикса равна 32 отсчетам, длина информационного символа равна 1024 отсчетам. Так как необходимо заменить все отсчеты, которые не относятся к информационным символам, нулями и, опираясь на длины циклического префикса и символа, можно вычислить номера отсчетов (относительного sop сигнала с выход коррелятора), относящиеся к информационным символам: с 31 по 1054 отсчет – первый информационный символ, с 1087 по 2110 – второй информационный символ, и с 2141 по 3166 отсчет – третий информационный символ. В остальные временные рамки необходимо выводить нули. Код данной операции представлен ниже.

```
always@(posedge iclk)
begin
        if(counter >= 14'd31 && counter <= 14'd1054)
        begin
                out_I<={in_I[13],in_I[13],in_I};
                out_Q<={in_Q[13],in_Q[13],in_Q};
```

```
        end
        else if(counter >= 14'd1087 && counter<= 14'd2110)
        begin
                out_I<={in_I[13],in_I[13],in_I};
                out_Q<={in_Q[13],in_Q[13],in_Q};
        end
        else if(counter >= 14'd2143 && counter <= 14'd3166)
        begin
                out_I<={in_I[13],in_I[13],in_I};
                out_Q<={in_Q[13],in_Q[13],in_Q};
        end
        else
        begin
                out_I<=14'd0;
                out_Q<=14'd0;
        end
end
```

5) Сигнал osop должен быть равен единице в первый отсчет информационного символа, т.е. в момент, когда опорный счетчик counter равен 31, 1087 и 2143.

```
always@(posedge iclk)
begin
        if(counter == 14'd31)
                osop<=1'b1;

        else if (counter == 14'd1087)
                osop<=1'b1;

        else if (counter == 14'd2143)
                osop<=1'b1;
        else
                osop<=1'b0;
end
```

В конце файла пропишите строчку окончания файла.

```
endmodule
```

6) Сохраните файл как Delete_CP.v и откройте RX.v. Следующим шагом необходимо прописать подключение данного блока к основному проекту, для этого сначала создайте две 14 разрядные шины: Shifter_DeleteCP_Data_I и Shifter_DeleteCP_Data_Q. Далее после блока корреляции пропишите подключение шин к блоку в следующем порядке: к порту iclk подлкючите шину CLK_10, к irst – reset, in_I и in_Q - Decimator_Correlator_Shifter_Data_I и Decimator_Correlator_Shifter_Data_Q соответственно, out_I и out_Q - Shifter_DeleteCP_Data_I и Shifter_DeleteCP_Data_Q. Код подключения представлен ниже.

```
////////////////// DELETE CYCLIC PREFIX ////////
wire signed [15:0] DeleteCP_FFT_Data_I;
wire signed [15:0] DeleteCP_FFT_Data_Q;

Del_CP Del_CP_Block(
                    .iclk       (CLK_10),
                    .irst       (~reset),
                    .in_I       (Shifter_DeleteCP_Data_I),
                    .in_Q       (Shifter_DeleteCP_Data_Q),
                    .out_I      (DeleteCP_FFT_Data_I),
                    .out_Q      (DeleteCP_FFT_Data_Q),
                    .sop        (Correlator_Sop),
                    .osop       (Del_CP_Sop));
```

7) Скомпилируйте проект. Откройте Signal Tap и добавьте сигналы с выхода модуля удаления циклического префикса: out_I, out_Q и osop. Список сигналов представлен на рисунке 1.1.

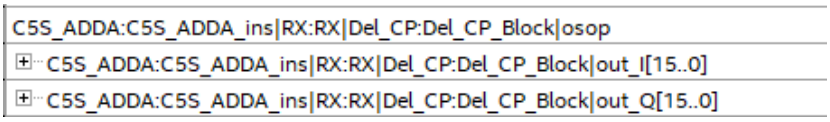


Рисунок 1.1 – Список сигналов в Signal Tap

Сохраните все изменения и скомпилируйте проект. Загрузите прошивку в плату. После чего запустите анализ. На рисунке 1.2 представлена правильная зависимость сигналов.

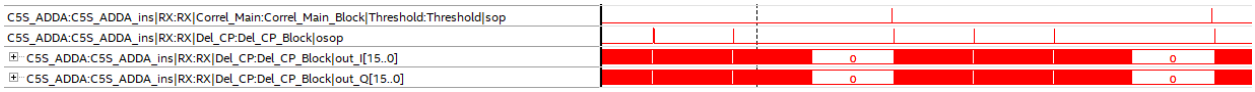


Рисунок 1.2 – Сигналы в Signal Tap

Весь кадр состоит из 4 символов: преамублы и трех информационных символов. Так как sop сигнал с выхода коррелятора совпадает с окончанием преамбулы, значит в следующий такт начинается информационный сигнал. В начале каждого информационного символа присутствует циклический префикс длиной 32 отсчета, который был заменен нулями. Длина каждого информационного символа составляет 1024 отсчета. В начале каждого информационного сигнала присутствует флаг osop, обозначающий начало символа.

Далее сгенерируйте Signal Tap List и скопируйте один символ в файл «2.txt». После чего откройте файл «FoundScatter.m». Так как в текстовом файле находится отсчеты одного символа, то необходимо изменить рамки цикла for: от первого отсчета до первого с шагом один. Так же раскомментируйте построение созвездие сигнала (строчка 13) и закомментируйте построение спектра (строчка 14). Код измененного цикла представлен на рисунке 1.3.

```
for i=1:1:1
    plot(real(fft(Sig(i:1:i+1023))),imag(fft(Sig(i:1:i+1023))),'.')
%     plot(abs(fft(Sig(i:1:i+1023))))
    pause(0.1)
end
```

Рисунок 1.3 – Измененный цикл

График созвездия, которое получится, представлено на рисунке 1.4.

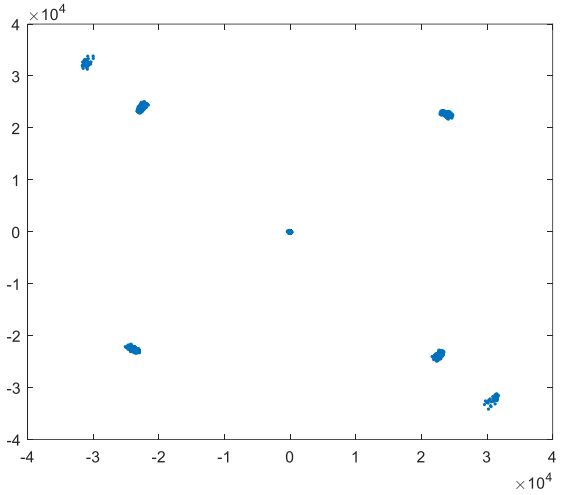


Рисунок 1.4 – Созвездие сигнала после удаления циклического префикса

Работа № 16

«Преобразование Фурье»

Цель работы: подключение IP ядра обратного быстрого преобразования Фурье.
Задачи работы:
1) Подключить IP ядро ОБПФ в проект;
2) Выполнить верфикацию.

1. Теоретическая часть

Преобразование Фурье – преобразование функции, превращающее её в совокупность частотных составляющих. Более точно, преобразование Фурье – это интегральное преобразование, которое раскладывает исходную функцию по базисным функциям, в качестве которых выступают синусоидальные (или мнимые экспоненты) функции, то есть представляет исходную функцию в виде интеграла синусоид (мнимых экспонент) различной частоты, амплитуды и фазы. Преобразование названо в честь Жана Фурье.

Преобразование Фурье является основоположником спектрального анализа. Спектральный анализ – это способ обработки сигналов, который позволяет охарактеризовать частотный состав измеряемого сигнала. В зависимости от того, каким образом представлен сигнал, используют разные преобразования Фурье. Различают несколько видов преобразования Фурье:

– Непрерывное преобразование Фурье (в англоязычной литературе Continue Time Fourier Transform – CTFT или, сокращенно, FT);

– Дискретное преобразование Фурье (в англоязычной литературе Discrete Fourier Transform – DFT);

– Быстрое преобразование Фурье (в англоязычной литературе Fast Fourier transform – FFT).

Преобразование Фурье является математическим инструментом, применяемым в различных научных областях. В некоторых случаях его можно использовать как средство решения сложных уравнений, описывающих динамические процессы, которые возникают под воздействием электрической, тепловой или световой энергии. В других случаях оно позволяет выделять регулярные составляющие в сложном колебательном сигнале, благодаря чему можно правильно интерпретировать экспериментальные наблюдения в астрономии, медицине и химии. Непрерывное преобразование фактически является обобщением рядов Фурье при условии, что период разлагаемой функции устремить к бесконечности. Таким образом, классическое преобразование Фурье имеет дело со спектром сигнала, взятым во всем диапазоне существования переменной.

Непрерывное преобразование Фурье используют, как правило, в теории при рассмотрении сигналов, которые изменяются в соответствии с заданными функциями, но на практике обычно имеют дело с результатами измерений, которые представляют собой дискретные данные. Результаты измерений фиксируются через равные промежутки времени с определённой частотой дискретизации, например, 16000 Гц или 22000 Гц. Однако в общем случае дискретные отсчёты могут идти неравномерно, но это усложняет математический аппарат анализа, поэтому на практике обычно не применяется.

Рисунок 1.1 – Исходный аналоговый сигнал и дискретный сигнал

Дискретное преобразование Фурье – это одно из преобразований Фурье, широко применяемых в алгоритмах цифровой обработки сигналов.

Быстрое преобразование Фурье (БПФ, Fast Fourier transform – *FFT*) представляет собой определенный алгоритм вычисления, который позволяет уменьшить количество производимых действий относительно прямого (по формуле) вычисления ДПФ. В основе алгоритма заложено разбиение заданной последовательности отсчетов дискретного сигнала на несколько промежуточных последовательностей. Следует отметить, что алгоритм БПФ точнее стандартного ДПФ, т.к. при сокращении операций снижаются суммарные ошибки округления.

В настоящее время известны несколько алгоритмов быстрого преобразования Фурье, которые являются частными случаями единого алгоритма, базирующегося на задаче разбиения одного массива чисел на два с последующим рекурсивном вычислении каждого массива чисел по дискретному преобразованию Фурье и объединении результатов расчетов.

Алгоритм БПФ с прореживанием по времени **за**ключается в разбиении входного сигнала на два массива чисел, составленных соответственно из четных и нечетных отсчетов. После этого для каждого массива чисел рассчитывается ДПФ с образованием двух функций. На заключительном шаге выполняются базовые операции сложения и вычитания с умножением одного из компонентов на экспоненциальный множитель. В результате выполнения данных операций получаем функцию. Граф алгоритма БПФ с прореживанием по времени представлен ниже.

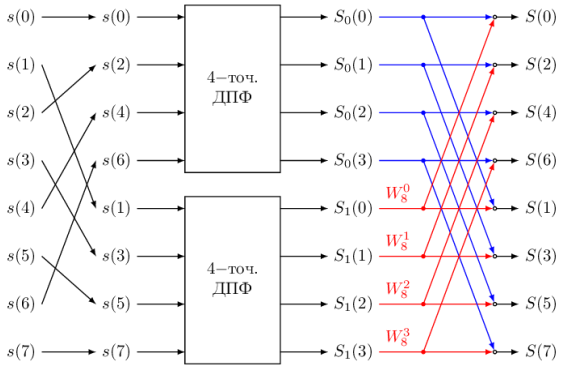


Рисунок 1.2 – Граф алгоритма БПФ с прореживанием по времени при делении исходного сигнала на четные и нечетные отсчеты

На структурной схеме базовую операцию сложения и вычитания с умножением одного из компонентов на экспоненциальный множитель принято показывать сигнальным графом, который в цифровой технике называется «бабочкой».

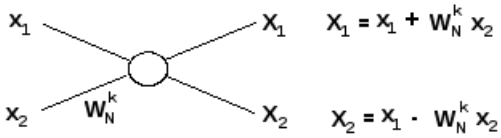


Рисунок 1.3 – Операция «бабочка», используемая при реализации алгоритма БПФ

Алгоритм БПФ с прореживанием по частоте заключается в разбиении входного сигнала пополам с образованием двух массивов чисел. После этого выполняются базовые операции сложения и вычитания с умножением двух компонентов на экспоненциальный множитель. На заключительном шаге рассчитывается ДПФ для четных и нечетных отсчетов с объединением результатов расчета. Граф алгоритма БПФ с прореживанием по частоте представлен ниже.

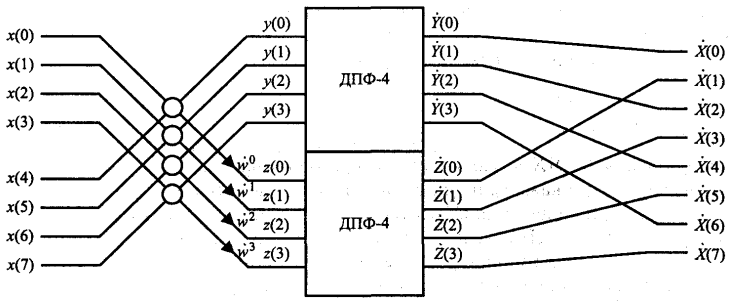


Рисунок 1.4 – Граф алгоритма БПФ с прореживанием по частоте при делении исходного сигнала пополам

2. Практическая часть

1. Для подключения блока преобразования Фурье необходимо открыть проект и во вкладке Project выбрать Add/Remove Files in Project, как показано на рисунке 2.1.

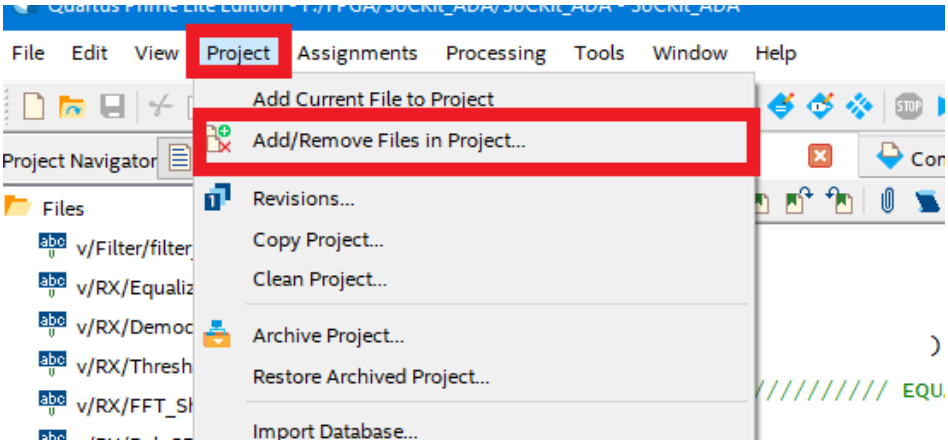


Рисунок 2.1 – Добавление библиотеки

2. В открывшемся окне перейдите в категорию Libraries. В секции Project libraries добавьте путь до следующих папок: fft_1024_16, fft_1024_16\fft_core, fft_1024_16\fft_core\lib. Папка fft_1024_16 расположена в «ваш_проект/libraries». Список библиотек представлен на рисунке 2.2. После добавления папок нажмите на Apply и OK.

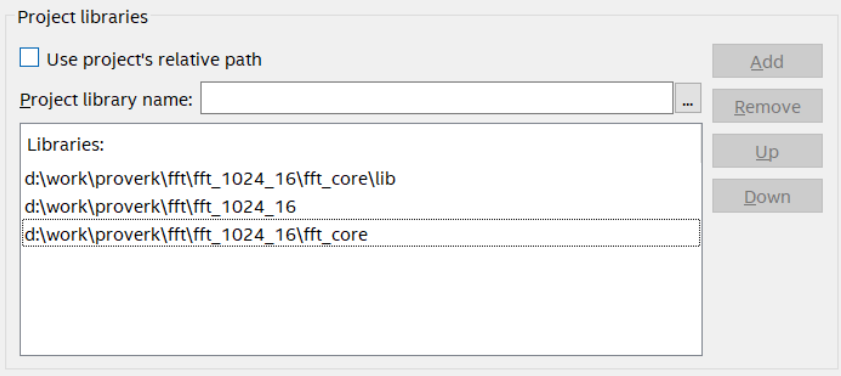


Рисунок 2.2 – Библиотеки FFT

3. Далее необходимо в проект добавить 2 файла: fft_1024_16.vhd и fft_pack.vhd. Первый файл расположен в папке «fft_1024_16». Второй файл расположен в «fft_1024_16/fft_core/lib/». Файлы в навигаторе проекта показаны на рисунке 2.3.

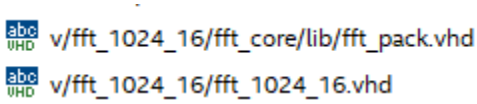


Рисунок 2.3 – Список фалов преобразования Фурье

4. Теперь необходимо подключить блок Фурье к проекту. Для этого откройте файл «RX.v» и после блока удаления циклического префикса пропишите подключение. Список портов представлен в таблице 2.1.

Таблица 2.1 – Описание портов

Название	Тип	Разрядность	Описание
clk	input	[0:0]	Входной порт тактирования
reset	input	[0:0]	Входной порт сброса
inv_i	input	[0:0]	Входной порт направление преобразования: 1 – обратное, 0 – прямое.
master_sink_dav	input	[0:0]	Входной порт разрешения
master_sink_sop	input	[0:0]	Входной порт флага начала символа
master_source_dav	input	[0:0]	
data_real_in	input	[15:0]	Входной порт реальной части данных
data_imag_in	input	[15:0]	Входной порт мнимой части данных
master_source_sop	output	[0:0]	Выходной порт флага начала символа
master_source_eop	output	[0:0]	Выходной порт флага конца символа
master_source_ena	output	[0:0]	
master_sink_ena	output	[0:0]	
fft_real_out	output	[15:0]	Выходной порт реальной части данных
fft_imag_out	output	[15:0]	Выходной порт мнимой части данных
exponent_out	output	[5:0]	Выходной порт показателя степени

Предварительно необходимо инициализировать две 16 разрядные шины (FFT_Equalizer_Data_I и FFT_Equalizer_Data_Q). Код инициализации и описания подключения представлен ниже.

```
//////////////////// IFFT //////////////////////

        wire    signed          [15:0]          FFT_Equalizer_Data_I;
        wire    signed          [15:0]          FFT_Equalizer_Data_Q;

fft_1024_16 ifft_submod(
                                        .clk                    (CLK_10),
                                        .reset                  (~reset),
                                        .inv_i                  (1'b0),
                                        .data_real_in           (DeleteCP_FFT_Data_I),
                                        .data_imag_in           (DeleteCP_FFT_Data_Q),
                                        .fft_real_out           (FFT_Equalizer_Data_I),
                                        .fft_imag_out           (FFT_Equalizer_Data_Q),
                                        .exponent_out           (exp_out),
                                        .master_sink_dav        (1'b0),
                                        .master_sink_ena        (),
                                        .master_sink_sop        (Del_CP_Sop),
                                        .master_source_sop      (fft_sop_out),
                                        .master_source_eop      (fft_eop_out),
                                        .master_source_ena      (fft_ena_out),
                                        .master_source_dav      (1'b1));
```

5. Скомпилируйте проект. Добавьте следующие сигналы, представленные на риснуке 2.4, с выхода блока преобразования фурье: fft_real_out, fft_imag_out и exponent_out.

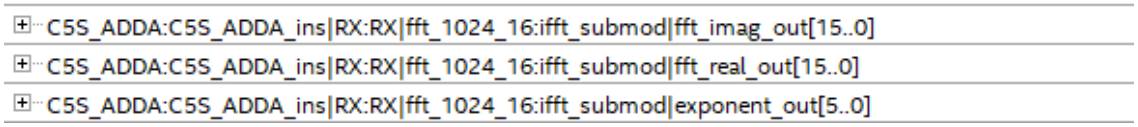


Рисунок 2.4 – Сигналы в Signal Tap

6. Сохраните изменения и скомпилируйте проект. Загрузите прошивку в плату и запустите анализ. На рисунке 2.5 показаны сигналы с выхода модулей удаления циклического префикса и преобразования Фурье.

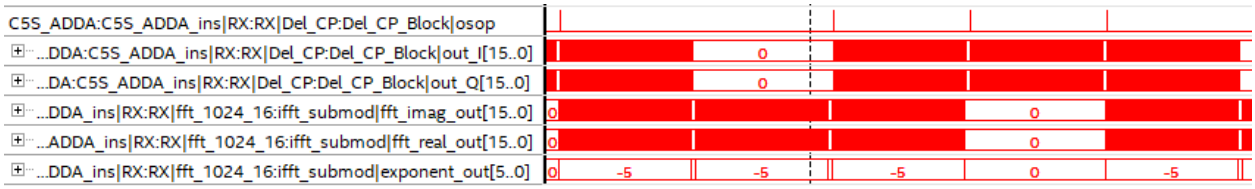


Рисунок 2.5 – Сравнение сигналов с выхода модулей удаление циклического префикса и преобразования Фурье

Как видно из рисунка 2.5, сигналы на выходе модуля Фурье, имеют задержку в 2 символа. В моменты, когда на выходных портах данных модуля преобразования Фурье не нулевое состояние, значение порта exponent_out должно быть не более -5.

7. Сгенерируйте Signal Tap List и скопируйте один символ с выхода модуля преобразования Фурье. Откройте папку Verification и откройте файл «1.txt». Удалите содержимое файла и вставьте скопированные отсчеты. Откройте файл «Plotting.m» и запустите программу нажатием клавиши F5. После запуска программы должно построится 2 графика: спектр и созвездие сигнала после модуля преобразования Фурье, показанные на рисунках 2.6 и 2.7 соотвественно.

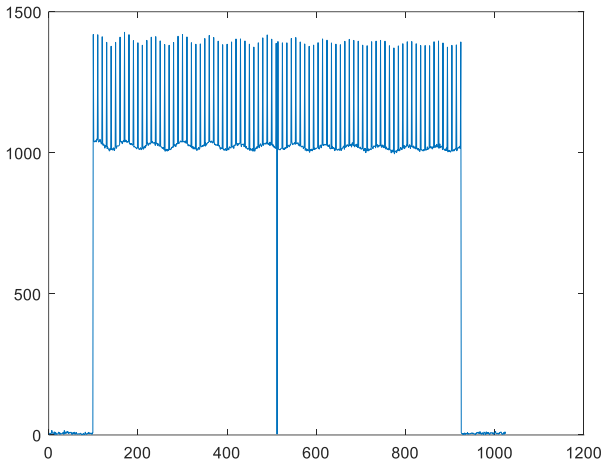


Рисунок 2.6 – Спектр сигнала

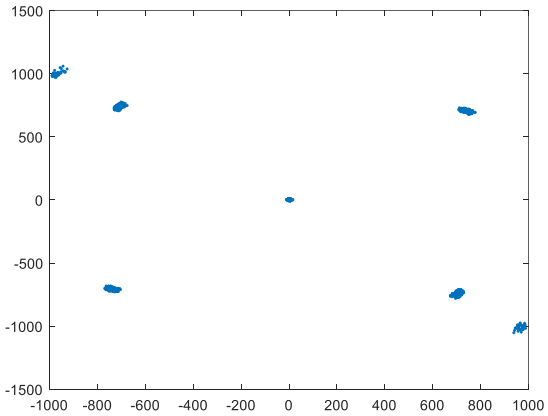


Рисунок 2.7 – Созвездие сигнала

Работа № 17

«Эквалайзирование»

Цель работы: реализация модуля эквалайзирования принимаемого сигнала с OFDM.

Задачи работы:

1) Реализовать оценку канала по пилотным поднесущим;

2) Реализовать востановления спектра принимаемого сигнала с помощью выполненой оценки;

3) Выполнить верификацию модуля.

1. Теоретическая часть

При распространении сигнала по каналу связи в результате воздействия шума, переотражения от различных объектов на пути распространения волны и других эффектов неизбежно возникают помехи. Спектр принятого сигнала может сильно искажаться, что влечёт за собой появление ошибок в принимаемом сообщении. По этой причине восстановление спектра является одной из важных задач приёмного устройства.

В спектр каждого символа OFDM добавляются специальные опорные поднесущие, так называемые пилот-сигналы. Приемник, зная расположение пилотных поднесущих может оценить искажения, вносимые каналом. По принятым значениям пилотных поднесущих определяется передаточная функция канала, зная которую восстанавливается спектр отправленного сигнала (эта процедура называется эквалайзирование). Точность оценки передаточной функции канала распространения радиоволн зависит от количества пилотных поднесущих в OFDM символе, а также от используемого алгоритма эквалайзирования.

На рисунке 1.1 иллюстрируется метод эквалайзирования: а – спектр передаваемого сигнала, б – спектр принятого сигнала, в – оценка АЧХ канала по значениям пилот-сигналов с линейной интерполяцией, г – восстановленный спектр полученного сигнала.

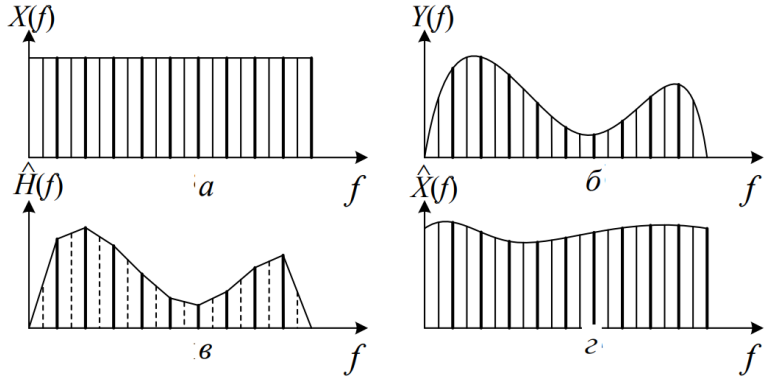


Рисунок 1.1 – Иллюстрация эквалайзирования

2. Практическая часть

1) Откройте проект и создайте новый файл Verilog HDL File. Первой строчкой в файле задайте название модуля (Equalizer) и внешние порты. Описание портов представлено в таблице 2.1.

Таблица 2.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса

Окончание таблицы 2.1

sop	input	[0:0]	Входной порт флага начала пакета
eop	input	[0:0]	Входной порт флага окончания пакета
In_I	input	[15:0]	Входной порт реальной части данных
In_Q	input	[15:0]	Входной порт мнимой части данных
Eq_Out_I	output	[15:0]	Выходной порт реальной части данных
Eq_Out_Q	output	[15:0]	Выходной порт мнимой части данных
Sop_out	output	[0:0]	Выходной порт флага начала пакета
Ena_out	output	[0:0]	Выходной порт разрешения

Код инициализации портов представлен ниже.

```
module Equalizer(   irst,
                    iclk,
                    sop,
                    eop,
                    In_I,
                    In_Q,
                    Eq_Out_I,
                    Eq_Out_Q,
                    Sop_out,
                    Ena_out);
```

2) Следующим шагом опишите порты, внутренние регистры и шины. Описание внутренних шин и регистров представлено в таблице 2.2.

Таблица 2.2 – Описание переменных

Название	Тип	Разрядность	Описание
Data_I	wire signed	[24:0]	Внутренняя шина реальной части данных
Data_Q	wire signed	[24:0]	Внутренняя шина мнимой части данных
Average_Out_I	wire signed	[9:0]	Внутренняя шина с оценкой эквалайзера реальной части входного сигнала
Average_Out_Q	wire signed	[9:0]	Внутренняя шина с оценкой эквалайзера мнимой части входного сигнала
Numerator_I	wire signed	[24:0]	Внутренняя шина входного сигнала
Numerator_Q	wire signed	[24:0]	Внутренняя шина входного сигнала
Pilot	wire signed	[15:0]	Внутренняя шина пилотных сигналов
Index_Out	wire	[1:0]	Внутренняя шина с картой несущих в символе

Окончание таблицы 2.2

Inter_sop	wire		Внутренняя шина с флагом начала пакета

Код описания портов, внутренних регистров и шин представлен ниже.

```
input                   iclk;
input                   irst;
input                   sop;
input                   eop;
input  signed    [15:0] In_I;
input  signed    [15:0] In_Q;

output signed    [15:0] Eq_Out_I;
output signed    [15:0] Eq_Out_Q;
output                  Sop_out;
output                  Ena_out;


wire signed      [24:0] Data_I;
wire signed      [24:0] Data_Q;

wire signed      [9:0]  Average_Out_I;
wire signed      [9:0]  Average_Out_Q;

wire signed      [24:0] Numerator_I;
wire signed      [24:0] Numerator_Q;

wire signed      [15:0] Pilot;
wire             [1:0]  Index_Out;
wire                    Inter_sop;
```

3) Эквалайзер состоит из четырех внутренних блоков: блока памяти, блока оценки и двух блоков деления. В первом блоке производится загрузка пилотов и карты несущих в символе из памяти, синхронизация с входным сигналом по флагам начала и окончания символа. Во втором блоке производится вычисление оценки канала по пилотным поднесущим. В третьем и четвертом блоках производится применение оценки к реальной и мнимой частям входного сигнала посредством деления их на оценку.

a. Создайте новый файл Verilog HDL File. Инициализируйте модуль (Sub_Eq_1) и входные порты. Описание входных портов первого блока представлено в таблице 2.3.

Таблица 2.3 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
sop	input	[0:0]	Входной порт флага начала пакета
eop	input	[0:0]	Входной порт флага окончания пакета

Окончание таблицы 2.3

In_I	input	[24:0]	Входной порт реальной части данных с умножением на 2048
In_Q	input	[24:0]	Входной порт мнимой части данных с умножением на 2048
Out_I	output	[24:0]	Выходной порт с реальной частью данных
Out_Q	output	[24:0]	Выходной порт с мнимой частью данных
Pilot_Out	output	[15:0]	Выходной порт с опорным пилотным сигналом
osop	output		Выходной порт начала символа
Index_Out	output	[1:0]	Выходной порт карты пилотов символа

Код инициализации портов представлен ниже.

```
module Sub_Eq_1 ( irst,
                  iclk,
                  sop,
                  eop,
                  In_I,
                  In_Q,
                  Out_I,
                  Out_Q,
                  Pilot_Out,
                  osop,
                  Index_Out);
```

b. Следующим шагом необходимо описать внутренние регистры и шины. Внутренние регистры и шины описаны в таблице 2.4.

Таблица 2.4 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
Pilot	reg signed	[15:0]	[90:0]	Регистр памяти с пилотными поднесущими
Index	reg	[1:0]	[1023:0]	Регистр памяти с картой символа
Counter	reg	[9:0]		Счетчик для памяти с картой символа
Counter_2	reg	[6:0]		Счетчик для памяти с пилотными поднесущими

Код описания портов, внутренних регистров и шин представлен ниже.

```
input                   iclk;
input                   irst;
input                   sop;
```

```
input                           eop;
input       signed      [24:0]  In_I;
input       signed      [24:0]  In_Q;

output reg  signed      [24:0]  Out_I;
output reg  signed      [24:0]  Out_Q;
output reg              [1:0]   Index_Out;
output      signed      [15:0]  Pilot_Out;
output reg                      osop;

reg         signed      [15:0]  Pilot       [90:0];
reg                     [1:0]   Index       [1023:0];
reg                     [9:0]   Counter;
reg                     [6:0]   Counter_2;
```

c. Далее загрузите карту символа и пилотные поднесущие из файлов в регистры памяти и задайте начальные состояния счетчиков. Код представлен ниже.

```
initial
begin
        $readmemb("Путь До Проекта\\SoCKit_ADA\\Input Data\\index.txt",Index);
        $readmemb("Путь До Проекта \\SoCKit_ADA\\Input Data\\Pilot.txt",Pilot);
        Counter     <=  10'd0;
        Counter_2<=  7'd127;
end
```

d. Следующим этапом инициализируйте счетчика Counter по сигналу sop и сброс по eop или isrt.

```
always@(posedge iclk)
begin
        if(eop || irst)
        begin
                Counter <= 10'd0;
        end
        else if(sop || Counter != 10'd0)
                Counter <= Counter+1'b1;
end
```

e. Далее инициализируйте счетчика Counter_2, когда значение регистра Index с адресом Counter равно 01, и возврат в начальное состояние, когда счетчик Counter равен 0. Код данной операции представлен ниже.

```
always@(posedge iclk)
begin
        if(Index[Counter] == 2'b01)
        begin
                Counter_2 <= Counter_2+1'b1;
        end
        else if(Counter == 10'd0)
```

```
            begin
                Counter_2 <= 7'd127;
            end
        end
```

f. Следующим шагом необходимо присвоить выходным портам Out_I, Out_Q, Index_Out и osop при помощи конструкции always значения входных и внутренних регистров In_I, In_Q, Index[Count] и sop. После с помощью assign присвойте выходному порту Pilot_Out значение ячейки регистра Pilot с адресом Counter_2.

```
always@(posedge iclk)
begin
        Out_I           <= In_I;
        Out_Q                   <= In_Q;
        Index_Out       <= Index[Counter];
        osop                    <= sop;
end
assign Pilot_Out = Pilot[Counter_2];
```

g. В конце файла пропишите строчку окончания модуля endmodule и сохраните файл как Sub_Eq_1.v.

4) Во втором внутреннем блоке эквалайзера происходит вычисление оценки канала по пилотным поднесущим, т.е. производится деление пилотов в принятом сигнале на опорные пилоты из памяти. Между двумя соседними оценками вычисляется среднее значение

a. Создайте новый файл Verilog HDL File. Для начала инициализируйте модуль (Sub_Eq_2) и внешние порты. Список внешних портов представлен в таблице 2.5.

Таблица 2.5 – Описание портов и параметров

Название	Тип	Разрядность	Описание
iclk	input	[0:0]	Входной порт тактирования
sop	input	[0:0]	Входной порт флага начала пакета
In_I	input	[24:0]	Входной порт реальной части данных с умножением на 2048
In_Q	input	[24:0]	Входной порт мнимой части данных с умножением на 2048
Pilot_In	input	[15:0]	Входной порт с опорным пилотным сигналом
Average_Out_I	output signed	[9:0]	Среднее значение между соседними оценками пилотных сигналов
Average_Out_Q	output signed	[9:0]	Среднее значение между соседними оценками пилотных сигналов
Out_I	output	[24:0]	Выходной порт с реальной частью данных
Out_Q	output	[24:0]	Выходной порт с мнимой частью данных
osop	output		Выходной порт начала символа

Код инициализации портов и модуля представлен ниже.

```
module Sub_Eq_2 (    iclk,
                                    In_I,
                                    In_Q,
                                    Pilot_In,
                                    sop,
                                    Index_In,
                                    Average_Out_I,
                                    Average_Out_Q,
                                    Out_I,
                                    Out_Q,
                                    osop,
                                    oena);
```

b. Следующим шагом опишите внешние порты, внутренние шины и регистры. Список внутренних шин и регистров представлен в таблице 2.6.

Таблица 2.6 – Описание переменных

Название	Тип	Разрядность	Размерность	Описание
Ozen_1_I	reg signed	[9:0]		Значение оценки принятого пилота реальной части
Ozen_2_I	reg signed	[9:0]		Значение оценки предыдущего принятого пилота реальной части
Ozen_1_Q	reg signed	[9:0]		Значение оценки принятого пилота мнимой части
Ozen_2_Q	reg signed	[9:0]		Значение оценки предыдущего принятого пилота мнимой части
Average_I	reg signed	[9:0]		Среднее значение между текущим и предыдущим значением оценки пилотных сигналов реальной части сигнала
Average_Q	reg signed	[9:0]		Среднее значение между текущим и предыдущим значением оценки пилотных сигналов мнимой части сигнала
Delay_Line_I	reg signed	[24:0]	[11:0]	Линия задержки для реальной части входного сигнала
Delay_Line_Q	reg signed	[24:0]	[11:0]	Линия задержки для мнимой части входного сигнала
Delay_Line_Index	reg signed	[1:0]	[11:0]	Линия задержки для карты символа
Delay_Sop	reg signed		[11:0]	Линия задержки для флага начала символа

Окончание таблицы 2.6

i	integer			Переменная для инициализации сдвига в линиях задержки.

Код описания внешних портов, внутренних шин и регистров представлен ниже.

```
input                       sop;
input                       iclk;
input          [1:0]        Index_In;
input signed   [15:0]       Pilot_In;
input signed   [24:0]       In_I;
input signed   [24:0]       In_Q;

output signed [9:0]         Average_Out_I;
output signed [9:0]         Average_Out_Q;
output signed [24:0]        Out_I;
output signed [24:0]        Out_Q;
output                      osop;
output                      oena;

reg     signed [9:0]        Ozen_1_I;
reg     signed [9:0]        Ozen_2_I;
reg     signed [9:0]        Ozen_1_Q;
reg     signed [9:0]        Ozen_2_Q;

reg     signed [9:0]        Average_I;
reg     signed [9:0]        Average_Q;

reg     signed [24:0]       Delay_Line_I        [11:0];
reg     signed [24:0]       Delay_Line_Q        [11:0];
reg     signed [1:0]        Delay_Line_Index    [11:0];
reg                         Delay_Sop           [11:0];

integer i;
```

с. Следующим этапом необходимо предыдущую оценку пилота (Ozen_1_I/Ozen_1_Q) присвоить в регистр Ozen_2_I/Ozen_2_Q. В регистр Ozen_1_I/ Ozen_1_Q записать новую оценку, вычисленную как отношение входного пилота и пилота из памяти. Месторасположение пилота определяется по карте символа Index_in. В регистрах Average_I и Average_Q записывается среднее значение между двумя подряд идущими пилотными сигналами реальной и мнимой части. Код представлен ниже.

```
always@(posedge iclk)
begin
        if(Index_In == 2'b01)
        begin
                Ozen_2_I <= Ozen_1_I;

                Ozen_1_I <= In_I/Pilot_In;
```

```
        end
    end

    always@(posedge iclk)
    begin
        Average_I <= (Ozen_2_I + Ozen_1_I)/2;
    end

    always@(posedge iclk)
    begin
        if(Index_In == 2'b01)
        begin
            Ozen_2_Q <= Ozen_1_Q;

            Ozen_1_Q <= In_Q/Pilot_In;
        end
    end

    always@(posedge iclk)
    begin
        Average_Q <= (Ozen_2_Q + Ozen_1_Q)/2;
    end
```

d. В следующем always блоке инициализируйте сдвиг регистров Delay_Line_I, Delay_Line_Q, Delay_Line_Index, Delay_Sop. В конце блока добавьте строчку окончания модуля и сохраните файл как Sub_Eq_2.v.

```
    always@(posedge iclk)
    begin
        for(i=1; i<=11; i=i+1)
        begin
            Delay_Line_I[i]     <= Delay_Line_I[i-1];
            Delay_Line_Q[i]     <= Delay_Line_Q[i-1];
            Delay_Line_Index[i] <= Delay_Line_Index[i-1];
            Delay_Sop[i]        <= Delay_Sop[i-1];
        end
        Delay_Line_I[0]     <= In_I;
        Delay_Line_Q[0]     <= In_Q;
        Delay_Line_Index[0] <= Index_In;
        Delay_Sop[0]        <= sop;
    end
    endmodule
```

e. Далее соедините выходные порты с регистрами задержки.

```
    assign Average_Out_I = (Delay_Line_Index[10] == 2'b10) ? Ozen_1_I: 10'd0;
    assign Average_Out_Q = (Delay_Line_Index[10] == 2'b10) ? Ozen_1_Q: 10'd0;
    assign Out_I         = (Delay_Line_Index[10] == 2'b10) ? Delay_Line_I[10]:
10'd0;
    assign Out_Q         = (Delay_Line_Index[10] == 2'b10) ? Delay_Line_Q[10]:
```

```
10'd0;
        assign osop                 = Delay_Sop[10];
        assign oena                 = (Delay_Line_Index[10] == 2'b10) ? 1'b1:1'b0;
```

5) В третьем модуле происходит эквалайзирование: деление входного сигнала на полученную оценку. Для реализации деления воспользуйтесь готовыми блоками деления. Для этого нажмите сочетание клавиш «Alt + 7» для открытия каталога готовых IP блоков. В секции Library откройте Basic Function => Arithmetic => LPM_DIVIDE, как показано на рисунке 2.1.

Рисунок 2.1 – Библиотека готовых IP блоков

При открытии блока укажите полное имя блока «Путь_до_вашего_проекта / Divide_Block.v». В первой вкладке укажите разрядность numerator равной 24 бита, denominator 8 бит. Оба параметра выберете Signed. Для перехода на следующие вкладки нажмите клавишу Next в нижней правой части окна. Все вкладки, кроме последней, оставьте без изменения, на последней вкладке уберите все галочки кроме Devide_Block.v. После

нажмите Finish.

6) Шестым шагом пропишите подключение внутренних блоков в Equalizer.v. Первым блоком необходимо подключить Sub_Eq_1.v. Ко входным пинам блока In_I и In_Q подключите входные шины In_I и In_Q умноженные на 2048, к пинам irst, iclk, sop, eop подключите одноименные шины (irst, iclk, sop, eop), к пинам Out_I, Out_Q, Pilot_Out, osop, Index_Out подключите внутренние шины Data_I, Data_Q, Pilot, Inter_Sop, Index_Out соответственно. Код данного подключения представлен ниже.

```
Sub_Eq_1 Sub_Eq_1(.irst        (irst),
                  .iclk        (iclk),
                  .sop         (sop),
                  .eop         (eop),
                  .In_I        (In_I*2048),
                  .In_Q        (In_Q*2048),
                  .Out_I       (Data_I),
                  .Out_Q       (Data_Q),
                  .Pilot_Out   (Pilot),
                  .osop        (Inter_sop),
                  .Index_Out   (Index_Out));
```

Следующим подключите блок Sub_Eq_2. К пину iclk подключите шину iclk, к пинам In_I и In_Q подключите внутренние шины Data_I и Data_Q соответственно, у Pilot_In и Index_In подключите шины Pilot и Index_Out, к Average_Out_I и Average_Out_Q подключите шины Average_Out_I и Average_Out_Q, к Out_I и Out_Q шины Numerator_I и Numerator_Q, к пинам osop и oena выходные шины Sop_Out и Ena_Out. Код подключения представлен ниже.

```
Sub_Eq_2 Sub_Eq_2 (.iclk       (iclk),
                   .sop        (Inter_sop),
                   .In_I       (Data_I),
                   .In_Q       (Data_Q),
                   .Pilot_In   (Pilot),
                   .Index_In   (Index_Out),
                   .Average_Out_I  (Average_Out_I),
                   .Average_Out_Q  (Average_Out_Q),
                   .Out_I      (Numerator_I),
                   .Out_Q      (Numerator_Q),
                   .osop       (Sop_out),
                   .oena       (Ena_out));
```

Далее подключите блок деления Divide_Block. Для реальной и мнимой частей используйте два блока с разными названиями (Divide_Block_I и Divide_Block_Q) по одному экземпляру (Divide_Block). К пину denom подключите Average_Out_ (I или Q), к пину numer подкючите Numerator_, к пину quotient подключите выходную шины Eq_Out_. Код подключения представлен ниже.

```
Divide_Block Divide_Block_I(  .denom      (Average_Out_I),
                              .numer      (Numerator_I),
                              .quotient   (Eq_Out_I),
                              .remain     ());
```

```
Divide_Block Divide_Block_Q(    .denom      (Average_Out_Q),
                                .numer      (Numerator_Q),
                                .quotient   (Eq_Out_Q),
                                .remain     ());
```

После пропишите окончания модуля.

```
endmodule
```

7) Сохраните все изменения. В блоке RX.v пропишите подключение эквалайзера после блока преобразования Фурье. Предварительно создайте 4 шины: две 16-разрядные шины Quotient_I и Quotient_Q, и две однорязрядные шины Eq_osop и Eq_ena. Код создания шин представлен ниже.

```
wire signed [15:0]  Quotient_I;
wire signed [15:0]  Quotient_Q;
wire                Eq_osop;
wire                Eq_ena;
```

Далее пропишите подключение блока. К пинам iclk и irst шины CLK_10 и ~reset соответственно, к пинам sop, In_I, In_Q и eop подключите fft_sop_out, FFT_Equalizer_Data_I, FFT_Equalizer_Data_Q и fft_eop_out. К пинам Eq_Out_I, Eq_Out_Q, Sop_out и Ena_out подключите созданные шины Quotient_I, Quotient_Q, Eq_osop и Eq_ena. Код подключения представлен ниже.

```
Equalizer Equalizer(    .irst       (~reset),
                        .iclk       (CLK_10),
                        .sop        (fft_sop_out),
                        .In_I       (FFT_Equalizer_Data_I),
                        .In_Q       (FFT_Equalizer_Data_Q),
                        .eop        (fft_eop_out),
                        .Eq_Out_I   (Quotient_I),
                        .Eq_Out_Q   (Quotient_Q),
                        .Sop_out    (Eq_osop),
                        .Ena_out    (Eq_ena));
```

8) Скомпилируйте проект и откройте Signal Tap. Удалите сигналы, идущие с модулей, расположенные до блока преобразования Фурье. Добавьте следующие:

- Сигналы с выхода блока эквалайзирования: Sop_out, Eq_Out_I и Eq_Out_Q;
- Сигналы с выхода модуля Sub_Eq_1: Out_I, Out_Q и Pilot_Out;
- Сигналы с выхода модуля Sub_Eq_2: Out_I, Out_Q, Average_Out_I и Average_Out_Q.

Список всех сигналов в Signal Tap представлен на рисунке 2.2.

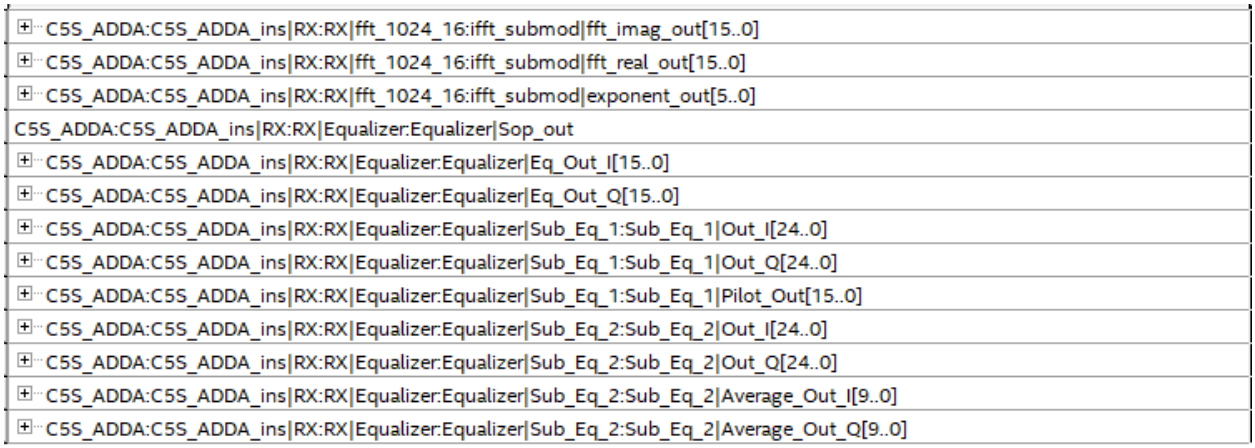


Рисунок 2.2 – Список сигналов в Signal tap

Сохраните все изменения и скомпилируйте проект. Загрузите прошивку в плату и запустите анализ. Сравните значения сигналов в Average_Out_I и Average_Out_Q. Знак каждого из отсчетов не должен меняться, как на рисунке 2.3.

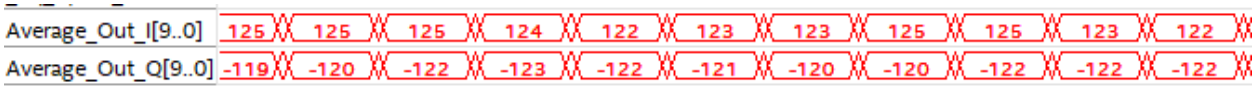


Рисунок 2.3 – Пример значений Average_Out_I и Average_Out_Q

Далее сгенерируйте Signal Tap List и скопируйте один символ с портов Eq_Out_I и Eq_Out_Q в файл «1.txt». После чего откройте файл «Plotting.m». Запустите программу нажатием клавиши «F5», после чего построится созвездие и спектр сигнала, как на рисунках 2.4 и 2.5 соотвественно.

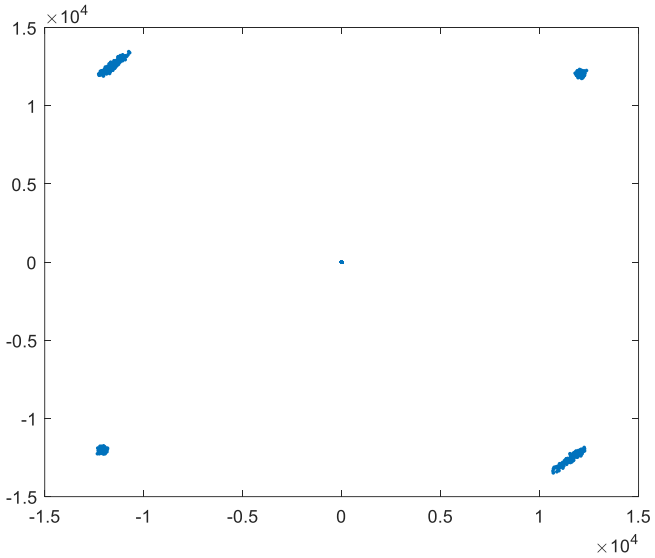


Рисунок 2.4 – Созвездие сигнала после эквалайзера

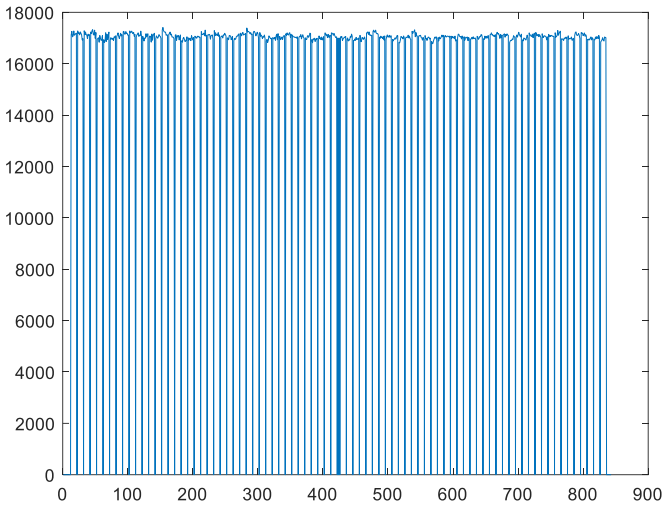


Рисунок 2.5 – Спектр сигнала после эквалайзера

Работа № 18

«QPSK демодудятор»

Цель работы: реализовать модуль QPSK демодуляции поднесущих.

Задачи работы:

1) Реализовать QPSK демодуляцию принятых значений сигнала;

2) Выполнить верефикацию модуля.

1. Теоретическая часть

Демодулятор – радиотехническое устройство, предназначенное для выделения информационного сигнала из модулированного.

В данном приемнике используется QPSK модуляция (квадратурная фазовая манипуляция).

Квадратурная фазовая модуляция QPSK – один из видов модуляции, используемый в цифровой радиосвязи. Структурная схема QPSK демодулятора приведена на рисунке 1.1.

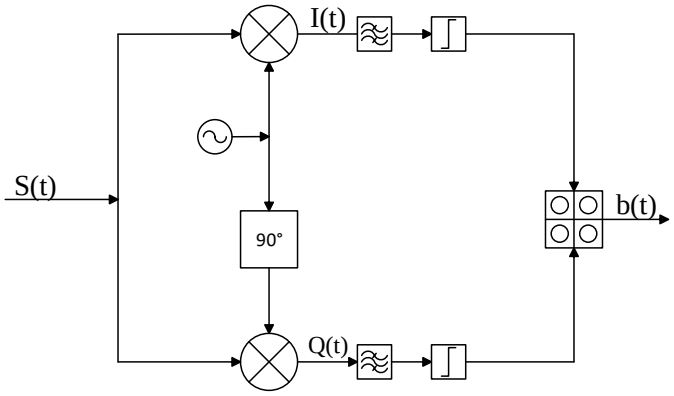


Рисунок 1.1 – Схема демодулятора

На вход демодулятора поступает модулированный сигнал. Первым этапом сигнал копируется на два потока и перемножается с несущим сигналом: в первом случае без сдвига фазы, в втором сдвиг фазы составляет 90 градусов. Сигнал, перемноженный с несущим сигналом без сдвига фазы, образует синфазную составляющую сигнала *(I(t))*, а второе перемножение дает квадратурную составляющую сигнала *(Q(t))*. После чего полученные составляющие сигнала поступают на вход ФНЧ.

Сам процесс QPSK модуляции сводится к выбору *I* (реальная) и *Q* (мнимая) компонент в зависимости от поступающей комбинации пары бит. Правило такого отображения можно описать таблицей 1.1.

Таблица 1.1 – Таблица соответствия

Биты (b_0 b_1)	Q	I
00	1	1
01	1	-1
10	-1	1
11	-1	-1

Или с помощью созвездия, представленного на рисунке 1.2.

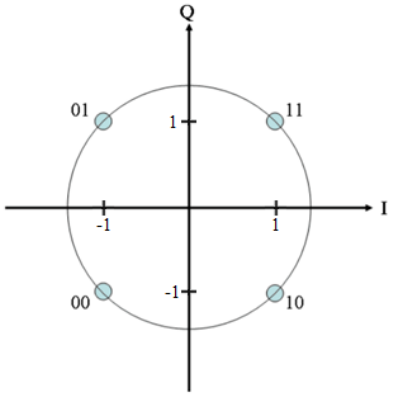


Рисунок 1.2 – Созвездие QPSK

2. Практическая часть

1) Откройте проект и создайте новый файл Verilog HDL File. Первой строчкой в файле необходимо задать название модуля (Demodulator) и внешние порты. Описание портов представлено в таблице 2.1.

Таблица 2.1 – Описание портов и параметров

Название	Тип	Разрядность	Описание
data_I	input signed	[15:0]	Входной порт реальной части данных
data_Q	input signed	[15:0]	Входной порт мнимой части данных
iclk	input	[0:0]	Входной порт тактирования
irst	input	[0:0]	Входной порт сброса
sop	input	[0:0]	Входной порт
oena	output	[0:0]	Выходной порт
bit_I	output	[0:0]	Выходной порт реальной части сигнала
bit_Q	output	[0:0]	Выходной порт мнимой части сигнала

Код для инициализации модуля и внешних портов представлен ниже.

```
module Demodulator (iclk,
                    ena,
                    sop,
                    data_I,
                    data_Q,
                    oena,
                    bit_I,
                    bit_Q);
```

2) Вторым этапом опишите внешние порты. Код представлен ниже.

```
input                   iclk;
input                   ena;
input                   sop;
input signed [15:0]     data_I;
input signed [15:0]     data_Q;

output                  oena;
```

```
output          bit_I;
output          bit_Q;
```

3) Третьим этапом реализуйте процесс модуляции. Для этого числа на входе сравниваются с нулем: если число меньше нуля, значит, модулирован ноль, если число равно 0, то модулирована единица. Так как в двоичной системе за знак числа отвечает последний бит, и если этот бит равен 1, то число отрицательное, а если 0 – то положительное. Процесс демодуляции можно упростить, взяв бит отвечающий за знак и инвертировать его. Код данной операции представлен ниже.

```
assign bit_I = ~data_I[15];
assign bit_Q = ~data_Q[15];
assign oena      = ena;
```

В конце укажите окончание блока и сохраните файл как “Demodulator.v”.

```
endmodule
```

4) Откройте файл «RX.v» и пропишите подключение созданного модуля, предварительно создав три одноразрядные шины demod_bit_I, demod_bit_Q и demod_ena.

```
wire demod_bit_I;
wire demod_bit_Q;
wire demod_ena;

Demodulator Demodulator (
                .iclk       (CLK_10),
                .ena        (Eq_ena),
                .sop        (Eq_ena),
                .data_I     (Quotient_I),
                .data_Q     (Quotient_Q),
                .oena       (demod_ena),
                .bit_I      (demod_bit_I),
                .bit_Q      (demod_bit_q));
```

Сохраните все изменения и запустите компиляцию проекта. После откройте Signal Tap и добавьте туда сигналы с выходов: bit_I, bit_Q и oena. Важно расположить сигналы именно в таком порядке!!! На рисунке 2.1 показано правильное расположение сигналов.

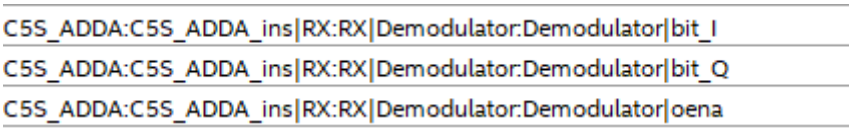


Рисунок 2.1 – Верное расположение сигналов

5) Сохраните Signal Tap и запустите компиляцию. После загрузите прошивку в плату и запустите анализ. Сгенерируйте Signal Tap List и скопируйте один пакет бит, показанный на рисунке 2.2.

Рисунок 2.2 – Один пакет бит

Вставьте скопированные отсчеты в файл «Bit_RX.txt» и откройте файл «Bit_RX.m». Запустите программу нажатием клавиши «F5». Если в командном окне выведется текст, который вы вписывали в самой первой лабораторной работе, то все выполнено верно.

ЗАКЛЮЧЕНИЕ

В данном учебно-методическом пособии были рассмотрены основные аспекты использования программируемых логических интегральных схем (ПЛИС) в системах беспроводной связи. Основной целью представленых работ является не только предоставить студентам необходимые теоретические знания, но и сосредоточиться на практических навыках, необходимых для успешного применения этих знаний в реальных проектах.

В пособии освещены основные принципы цифрового формирования и обработки сигналов, а также подробно описаны их реализация на ПЛИС. Кроме того, были представлены практические задания, направленные на закрепление полученных знаний и развитие навыков работы с современными инструментами разработки, такими как Quartus II.

В процессе изучения данного пособия студенты получат не только теоретические знания, но и практические навыки, которые пригодятся им в будущей профессиональной деятельности.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. SoCKit - the Development Kit for New SoC Device : сайт / terasic. – URL: https://www.terasic.com.tw/cgi-bin/page/archive.pl. – Режим доступа: свободный (дата обращения: 15.08.2023).

2. Intel® Quartus® II Web Edition Design Software Version 13.1 for Windows : сайт / intel. – URL: https://www.intel.com/content/www/us/en/software-kit/666221/intel-quartus-ii-web. – Режим доступа: свободный (дата обращения: 07.08.2023).