Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 24.М41-мм

Создание IP-ядра полукогерентного обнаружителя частотно модулированных сигналов с непрерывной фазой

Фаст Никита Михайлович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры информатики, к. ф.-м. н,. Д. В. Луцив

Консультант:
Начальник отдела программирования АО «Концерн ГРАНИТ», А. С. Кривоногов

Санкт-Петербург
2025

Оглавление

Введение

В современном мире технологии беспроводной связи играют важную роль в обеспечении эффективной коммуникации. Радиосигналы, лежащие в основе этих технологий, являются основой для работы мобильных телефонов, радиостанций, навигационных систем и многих других устройств.

Процесс приема радиосигналов затруднен наличием в радиоэфире шумов и помех, неидеальностью приемной и передающей аппаратуры, эффектами многолучевого распространения сигналов. Также в случае использования слотовой передачи данных полезный сигнал может присутствовать в эфире не постоянно, а лишь в определенные промежутки времени. В связи с этим, неотъемлемым компонентом систем беспроводной связи является обнаружитель. Данное устройство устанавливает факт наличия или отсутствия в радиоэфире информационного сигнала, а также определяет параметры искажений, таких как: ошибка по частоте, ошибка по времени, ошибка по фазе и другие.

Процесс обнаружения требует выполнения потоковой обработки данных в режиме реального времени. Для решения этой задачи можно использовать либо сигнальные процессоры, либо интегральные схемы. С учетом серийности производства зачастую экономически оправдано не использовать сигнальные процессоры, а реализовать обнаружитель с использованием программируемых логических интегральных схем (FPGA) или в виде интегральной схемы специального назначения (ASIC).

В рамках работ по разработке беспроводных систем связи, проводимых АО концерн «ГРАНИТ», необходимо разработать IP-ядро полукогерентного обнаружителя частотно модулированных сигналов с непрерывной фазой.

1. Постановка задачи

Целью работы является создание IP-ядра полукогерентного обнаружителя частотно модулированных сигналов с непрерывной фазой. Для достижения данной цели были поставлены следующие задачи.

1. Сформулировать критерии выбора алгоритма обнаружения.
2. Осуществить выбор алгоритма обнаружения путем выполнения обзора существующих алгоритмов обнаружения и их сравнения по выбранным критериям.
3. Спроектировать архитектуру обнаружителя.
4. Реализовать обнаружитель на одном из языков HDL.
5. Разработать тестовое окружение под процессор ARM и с его помощью выполнить тестирование разработанного IP-ядра.

2. Обзор

2.1. Особенности процесса разработки под ПЛИС

Программируемые логические интегральные схемы (ПЛИС) – это особый вид микросхем, логика которых не определяется на этапе производства, а задается инженером путем её программирования. ПЛИС состоит из набора логических блоков и множества программируемых соединений. Логические блоки содержат примитивы, такие как: D-триггеры, таблицы перекодировки (LUT), мультиплексоры, порты ввода-вывода, блоки памяти.

Важно отметить, что процесс программирования ПЛИС отличается от процесса программирования процессора в силу того, что ПЛИС, в отличие от процессора, не исполняет написанный под нее код. В случае ПЛИС этот код конфигурирует соединения между внутренними примитивами, обеспечивая заданную алгоритмику работы.

Для программирования ПЛИС используются предоставляемые её производителем программатор и IDE. Основной задачей IDE является преобразование HDL-кода в набор примитивов, имеющихся на ПЛИС, их коммутация и размещение на ПЛИС. В связи с этим основными компонентами IDE являются: редактор HDL-кода, синтезатор, оптимизатор размещения и маршрутизации. Разработчик может оказывать влияние на синтезатор и оптимизатор при помощи ограничений (constraints).

Результатом разработки на языках HDL являются готовые функциональные блоки, называемые IP-ядрами. IP-ядро может использоваться как обособленно, так и в качестве отдельного блока при проектировании более сложных ядер. На основе IP-ядра может быть получена интегральная схема специального назначения (ASIC), которая впоследствии может быть поставлена на серийное производство.

В настоящее время ПЛИС зачастую комбинируют вместе с процессором на одном общем кристалле, получая так называемый System-on-Chip(SoC). Данный SoC представляет собой комбинирование двух подсистем: процессорной системы (PS) и программируемой логики (PL).

PS содержит процессор ARM, а PL - ПЛИС. Между ними внутри кристалла существуют стандартизированные интерфейсы взаимодействия. Также SoC оснащён разнообразной периферией, которая позволяет организовывать эффективное взаимодействие с таким устройством, например, в целях отладки. Мощностей PS достаточно для развертывания операционной системы Linux[8, 16]. Если же наличие ОС избыточно, то процессорная система может быть использована для развертывания Bare Metal приложений. Наличие процессорной системы расширяет возможности по использованию ПЛИС и существенно упрощает отладку разрабатываемых IP-ядер. В связи с этим распространено использование SoC в качестве платформы для разработки, отладки и апробации IP-ядер.

2.2. Существующие решения

Разработанные IP-ядра зачастую имеют высокую коммерческую ценность, поскольку на их основе могут быть созданы микросхемы. В силу этого количество IP-ядер с открытым исходным кодом крайне ограничено. Дополнительно необходимо учитывать, что IP-ядро может быть реализовано под конкретную ПЛИС или семейство ПЛИС и по этой причине быть несовместимым с ПЛИС от другого производителя. Возможен даже сценарий, при котором IP-ядро будет несовместимо с младшей ПЛИС того же семейства из-за нехватки в ней ресурсов для синтеза данного IP-ядра. Каждый из пунктов, перечисленных выше, может стать причиной, по которой требуется создать собственное IP-ядро. В данной работе рассматривается создание собственного IP-ядра обнаружителя для платформы AMD Zynq™ 7000 SoC[3], в силу того, что готовых решений найдено не было.

2.3. Используемые технологии

2.3.1. Платформа

Задачей обнаружителя является обнаружение в эфире наличия радиосигнала и оценка его параметров. Таким образом, для своей работы обнаружителю требуется сигнал. Из этого следует, что для реализации обнаружителя требуется не только ПЛИС, но и микросхема радиоприемника, на вход которой поступает радиосигнал на несущей частоте, а на выходе имеются отсчеты АЦП, которые уже могут быть переданы в ПЛИС для обработки обнаружителем. Соответственно, в качестве платформы требовалось устройство, в составе которого помимо SoC имеются компоненты, необходимые для приема и формирования радиосигналов. Такое устройство называется программно определяемая радиосистема (SDR). Наиболее интересным предложением на рынке по соотношению цена/качество является PlutoSDR+. Главными компонентами данного устройства являются:

- AMD Zynq™ 7000 SoC;
- микросхема приемопередатчика AD9363[11].

2.3.2. Используемые инструменты

При программировании под ПЛИС выбор инструментов разработки определяется вендором ПЛИС. Ранее упоминалось, что код, написанный на языках HDL, не исполняется ПЛИС, а является описанием разрабатываемой микросхемы. Реализация микросхемы получается путем настройки и коммутации примитивов, доступных на ПЛИС. Вывод примитивов, необходимых для реализации микросхемы, на основе ее HDL описания, выполняется инструментом под названием синтезатор. Процесс синтеза требует информации о том, какими примитивами и в каком количестве обладает целевая ПЛИС. Помимо синтеза имеются процессы по размещению синтезированных примитивов на ПЛИС, временному анализу полученного решения и т.д. Все это требует знаний о внутреннем устройстве ПЛИС, которыми может обладать лишь

её производитель. Поэтому каждый вендор ПЛИС, предоставляет собственный набор инструментов разработки.

В данном случае производителем ПЛИС является Xilinx (AMD), поэтому используется Vivado IDE[2]. Поскольку в данной работе используется не просто ПЛИС, а SoC, оснащенный процессором ARM, то для написания кода на C и C++, а также его отладки и заливки на целевое устройство дополнительно используется Vitis IDE[1].

2.3.3. Выбор HDL

Наиболее активно используемыми языками описания аппаратуры являются VHDL[12] и Verilog[13]. Перед началом работ требовалось определиться с тем, какой язык будет использоваться для реализации обнаружителя. К преимуществам VHDL можно отнести строгую систему типов, возможность создавать пользовательские типы данных и детерминированность симуляции. Первое позволяет выявлять значительное количество ошибок уже на этапе компиляции, второе упрощает процесс проектирования, а третье позволяет избежать появления труднообнаружимых ошибок симуляции. Основным преимуществом Verilog является краткость языковых конструкций и наличие готовых операторов, позволяющих ускорить написание кода. Еще одним фактором при выборе HDL является степень поддержки данного языка со стороны инструмента, используемого для синтеза и размещения на кристалле. Однако в данном случае используемая IDE Vivado в равной степени поддерживает оба языка, поэтому этот фактор оказался несущественным. В силу перечисленного выше и того факта, что в организации АО концерн «ГРАНИТ» основным языком описания аппаратуры является VHDL, было принято решение использовать для реализации обнаружителя VHDL.

3. Алгоритм

При обнаружении сигналов широко распространен следующий подход. В передаваемый сигнал добавляется преамбула - заранее известная последовательность данных, обладающая хорошими авто-корреляционными свойствами. Приемник выполняет сравнение полученного сигнала с преамбулой путем расчета взаимнокорреляционной функции. Как известно, корреляция двух сигналов показывает их меру похожести[17].

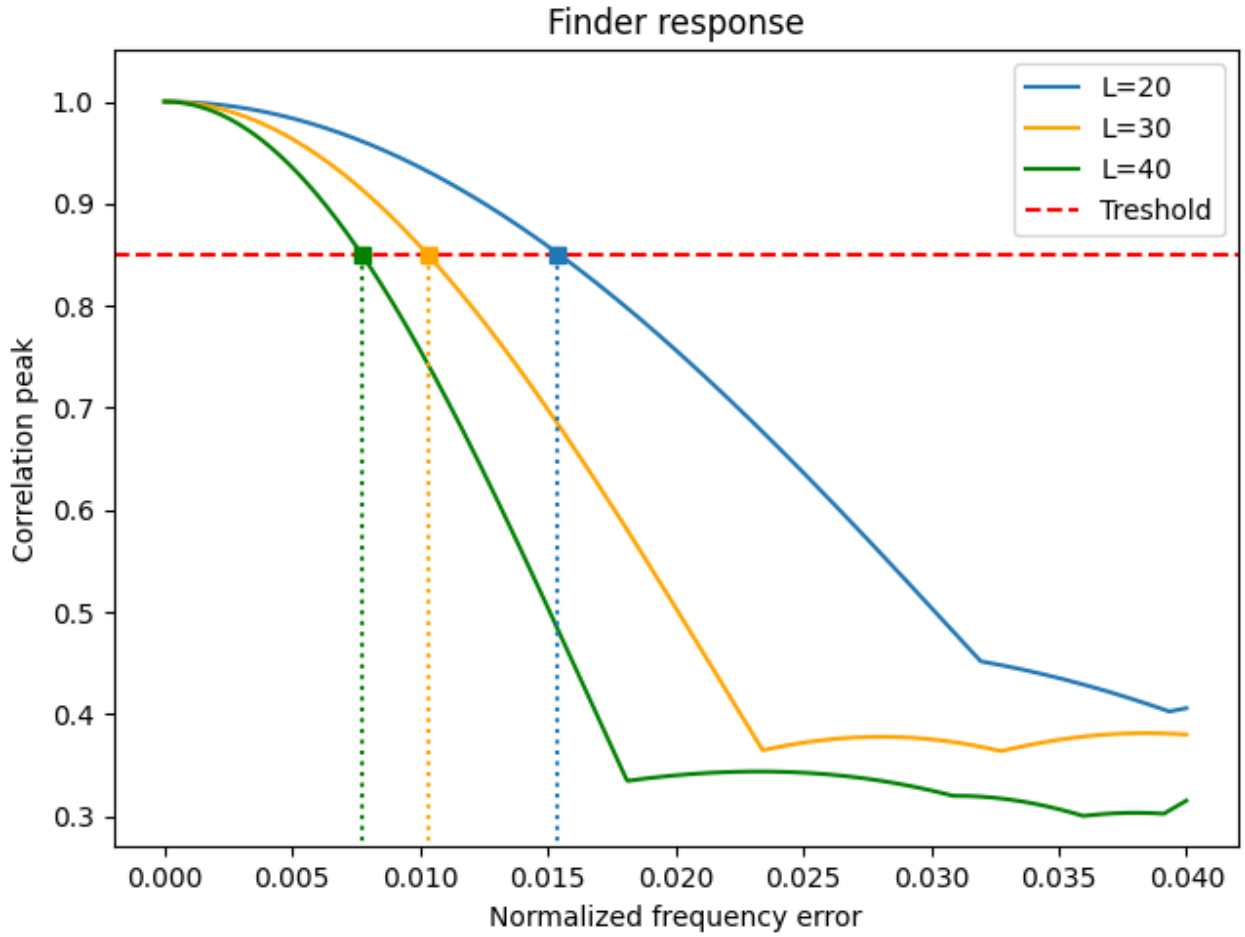
Если взаимнокорреляционная функция рассчитывается между принятой из радиоэфира преамбулой и опорной преамбулой, то по максимальному значению этой функции можно судить о наличии сигнала, его уровне и степени искажений. В случае отсутствия сигнала, взаимная корреляция между шумом или любым другим сигналом и опорой будет давать максимальные значения заведомо ниже тех, что были бы в случае наличия сигнала. Поэтому можно выбрать порог, позволяющий отличить две данные ситуации. Если отклик превышает заранее выбранный порог, то сигнал обнаружен, если не превышает, то сигнал отсутствует. Среди всех алгоритмов обнаружения корреляционный вариант является оптимальным с точки зрения энергетики для каналов с аддитивным белым гауссовским шумом[18]. По этой причине в работе рассматривается разработка именно корреляционного обнаружителя.

Отметим, что принятый радиосигнал не является в точности таким, каким он был на передатчике. Сигнал, принятый из радиоэфира, будет иметь искажения, обусловленные шумами приемного тракта, помехами из радиоэфира, эффектами многолучевого распространения, нестабильностью опорного генератора на приемной и передающих сторонах.

Так как корреляционный обнаружитель основан на расчете взаимнокорреляционной функции, которая значительно деградирует при наличии частотного рассогласования между опорой и сигналом, то необходимо выполнить частотную коррекцию принятого сигнала. Оптимальным подходом с точки зрения энергетики является перебор всех возможных частотных сдвигов и расчет взаимнокорреляционной функции при каждом из них. Однако такой подход вычислительно нереализуем, поэтому

перебор частотных сдвигов производится с некоторым шагом. Величина шага связана с размером преамбулы. Чем длиннее преамбула, тем меньше частотная ошибка, приводящая к снижению пика взаимнокорреляционной функции на фиксированную величину (рис. 1). Поэтому, если шаг будет слишком большим, то остаточной частотной ошибки, имеющейся в сигнале после частотной коррекции, будет достаточно для снижения корреляции ниже уровня порога, из-за чего сигнал не будет обнаружен. Перебор частотных сдвигов с меньшим шагом потребует большого числа итераций и, как следствие, больших вычислительных ресурсов для устранения частотной ошибки, значение которой неизвестно, но для конкретной системы связи находится в известных пределах.

Рис. 1: Влияние частотной ошибки на отклик корреляционного обнаружителя в зависимости от длины преамбулы.



Одновременно с этим увеличение длины преамбулы прямо пропорционально увеличивает ее энергию, что при фиксированной мощности

передатчика позволяет обнаруживать сигналы на большем расстоянии. Соответственно, уменьшение размеров преамбулы приводит к ухудшению энергетических свойств системы.

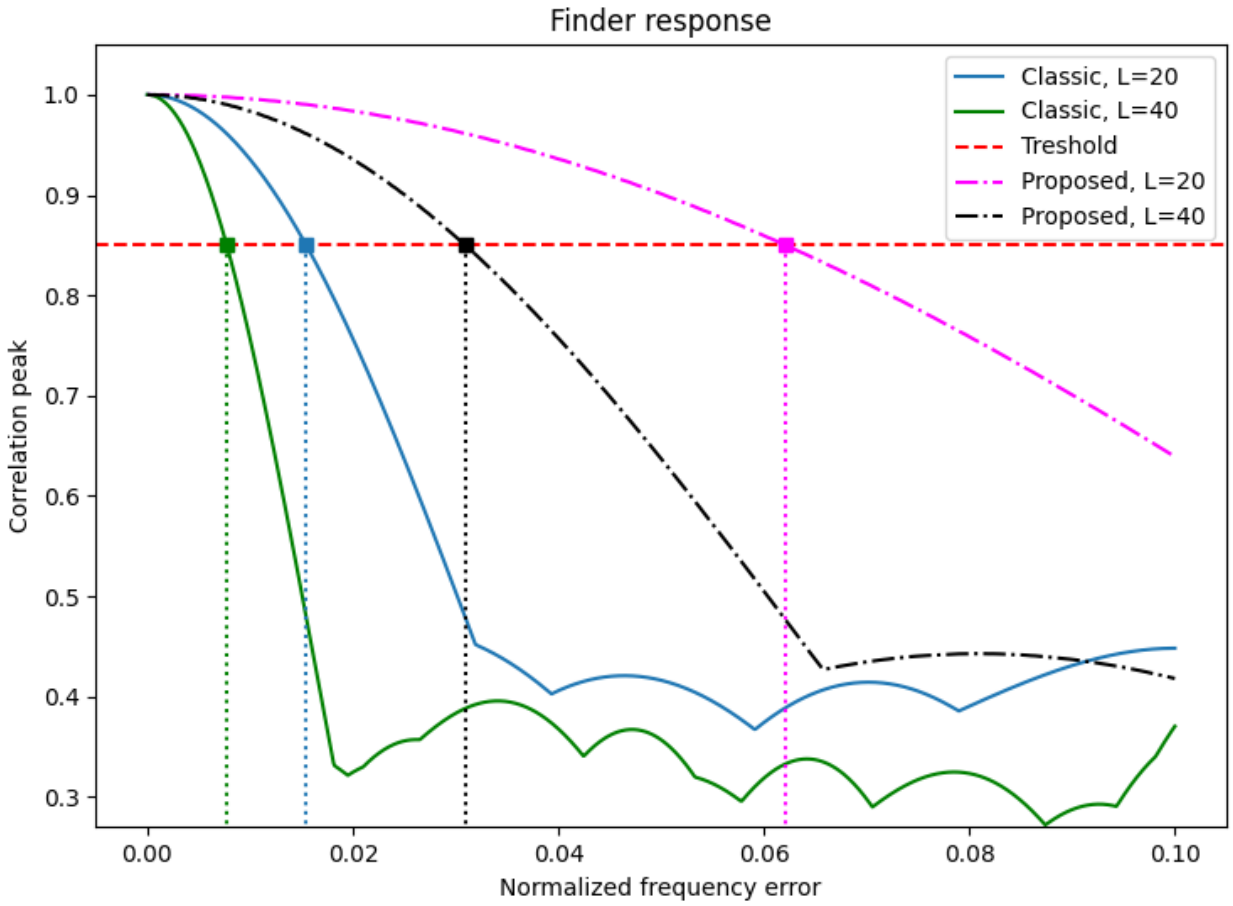
Таким образом, выбор длины преамбулы является компромиссом между энергетическими характеристиками обнаружителя и ресурсоемкостью его реализации. Для борьбы с данной проблемой предлагается следующий модифицированный алгоритм корреляционного обнаружения.

3.1. Описание алгоритма

Преамбула разделяется на сегменты равной длины, называемые под-преамбулами. Для определения и устранения частотной ошибки принятого сигнала осуществляется перебор частотных сдвигов с фиксированным шагом. При каждом конкретном значении выполняется сдвиг сигнала по частоте на данную величину, а также производится расчет взаимнокорреляционной функции между полученным сигналом и каждой под-преамбулой. Поскольку длина сегмента много меньше длины всей преамбулы, взаимнокорреляционная функция способна «пережить» значительно большую остаточную ошибку по частоте, чем в случае коррелирования с цельной преамбулой (рис. 2). Поэтому перебор частотных сдвигов осуществляется со значительно большим шагом, чем в классическом алгоритме, что позволяет сократить объем ресурсов, необходимых для реализации.

Значение вычисленной корреляционной функции в каждый конкретный момент времени (корреляция) является комплексным числом, которое по определению из Менгали [ССЫЛКА] представимо как сумма отклика корреляторa и шумовой компоненты. Далее выполняется сложение корреляций с каждой из под-преамбул друг с другом. Поскольку рассматривается канал с АБГШ, то при сложении корреляций среднее значение шума не изменяется, но возрастает его дисперсия. При этом происходит суммирование энергий откликов корреляторов, что улучшает энергетические свойства алгоритма. При суммировании корреляций

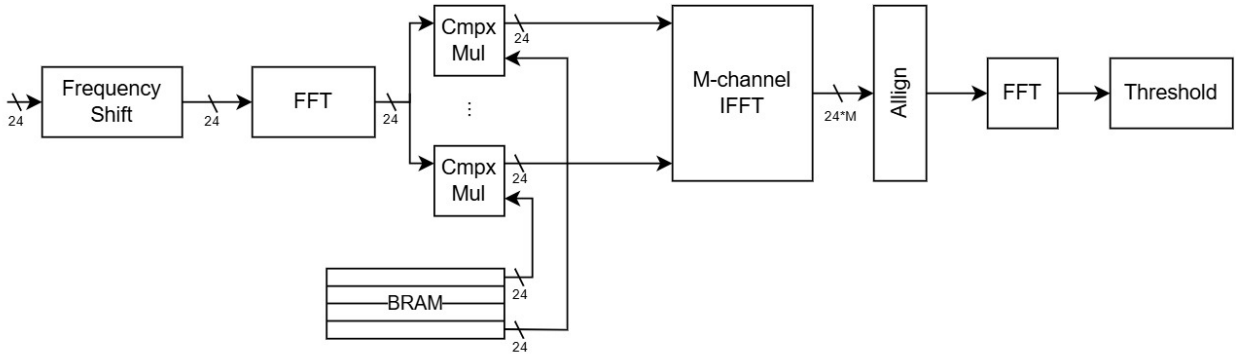
Рис. 2: Влияние частотной ошибки на отклики классического и предлагаемого обнаружителя в зависимости от длины преамбулы.



необходимо помнить о наличии остаточной частотной ошибки. В случае простого суммирования всех откликов частотная ошибка не устраняется, что приводит к потерям в энергетике. Поэтому в рамках данного алгоритма при помощи быстрого преобразования Фурье выполняется оценка остаточной ошибки по частоте, ее устранение и суммирование откликов корреляторов. От получившегося комплексного числа необходимо взять модуль, поскольку начальная фаза сигнала неизвестна. В результате полученное действительное число сравнивается с порогом и устанавливается факт наличия или отсутствия сигнала.

Блок-схема предлагаемого алгоритма изображена на рис. 3. Рассмотрим реализацию блоков подробнее.

Рис. 3: Блок схема алгоритма.



3.1.1. Частотный сдвиг

Принятый сигнал имеет неизвестную ошибку по частоте, значение которой однако находится в заведомо известных пределах. В рамках алгоритма осуществляется перебор частотных сдвигов с некоторым фиксированным шагом с последующей частотной коррекцией сигнала на данную величину. После такой процедуры сигнал все еще может содержать ошибку по частоте, однако ее значение будет не более чем половина от величины шага перебора. Для сдвига сигнала в частной области используется умножение на комплексную экспоненту[17]. В соответствии с формулой Эйлера, комплексная экспонента определяется как:

$$e^{jx} = cosx + jsinx$$

.

Для формирования комплексной экспоненты в ПЛИС используется Xilinx IP-ядро под названием DDS Compiler[6]. В данном случае IP-ядро настраивается для формирования 24-битной комплексной экспоненты с частотой, соответствующей рассматриваемому частотному сдвигу. Умножение сигнала на комплексную экспоненту производится с помощью IP-ядра Complex Multiplier[5]. Оба входных сигнала являются комплексными и имеют разрядность в 24-бита. Важной настройкой комплексного умножителя является выбор механики переполнения[1].

[1]Расчет значений в ПЛИС происходит в арифметике с фиксированной точкой, при этом как операнды, так и результат будут иметь одинаковую разрядность, что приводит к наличию переполнений, которые в отсутствии контрмер, приведут к неработоспособности алгоритма. Поэтому в дальнейшем

3.1.2. Расчет взаимнокорреляционной функции между сигналом под-преамбулами

Корреляция двух дискретных сигналов f и g, где f это сигнал, а g - опора, рассчитывается по формуле

$$(f \star g)_i = \sum_j f_j g^*_{i+j}.$$

При таком подходе для расчета взаимной корреляционной функции необходимо выполнить

$$L^2(N - L + 1)$$

операций, где L это длина опоры, а N - длина сигнала. Такой подход будет вычислительно неэффективен для сигналов и опор умеренной и большой длительности. Существует вычислительно эффективный метод расчета, основанный на Теореме о свертке. Данная теорема гласит, что спектр свертки двух сигналов равен произведению спектров сворачиваемых сигналов[17]. Поскольку корреляция двух сигналов это свертка первого сигнала с комплексно-сопряженной и взятой в обратном порядке копией второго сигнала, то для расчета корреляции может быть использована следующая процедура.

1. рассчитать спектр опоры при помощи БПФ;
2. рассчитать спектр сигнала при помощи БПФ[2];
3. поэлементно перемножить рассчитанные спектры;
4. взять обратное БПФ от произведения спектров.

Данная процедура используется в рамках алгоритма для расчета взаимнокорреляционной функции между сигналом и сегментами преамбулы. Поскольку сегменты преамбулы известны заранее, то их спектр может быть вычислен и сохранен в памяти на ПЛИС, внутри блоков

необходимо уделить пристальное внимание алгоритмике переполнения.

[2]При расчете с помощью БПФ получается циклическая свертка, однако дополнив опору и сигнал нулями, можно выполнить расчет требуемой линейной свертки с помощью расчета циклической.

BRAM[4]. Для расчетов прямого и обртаного преобразований Фурье используется FFT IP-ядро, разработанное Xilinx[7].

3.1.3. Суммирование откликов

Значение вычисленной корреляционной функции в каждый конкретный момент времени (корреляция) является комплексным числом, которое, на основании работы У. Менгали[9], может быть представлено как сумма отклика коррелятора и шумовой компоненты. Далее выполняется сложение корреляций с каждой из под-преамбул друг с другом. Поскольку рассматривается канал с АБГШ, то при сложении корреляций среднее значение шума не изменяется, но возрастает его дисперсия. При этом происходит суммирование энергий откликов корреляторов, что улучшает энергетические свойства алгоритма. При суммировании корреляций необходимо помнить о наличии остаточной частотной ошибки. В случае простого суммирования всех откликов частотная ошибка не устраняется, что приводит к потерям в энергетике. Поэтому в рамках данного алгоритма при помощи быстрого преобразования Фурье выполняется оценка остаточной ошибки по частоте[3], ее устранение и суммирование откликов корреляторов. От получившегося комплексного числа необходимо взять модуль, поскольку начальная фаза сигнала неизвестна. В результате полученное действительное число сравнивается с порогом и устанавливается факт наличия или отсутствия сигнала.

[todo порядок fft определяет размер бина и точность оценки частотной ошибки, формула ДПФ, формула из Менгали]

3.1.4. Сравнение с порогом

Отлик, получившийся в результате суммирования, также является коплексным числом. Поскольку начальная фаза сигнала неизвестна, то от него необходимо взять модуль. Полученное действительное число сравнивается с порогом, устанавливая факт наличия или отсутствия сигнала. Также важно отметить, что позиция максимума указывает на

[3]Порядок fft определяет размер бина и точность оценки остаточной частотной ошибки.

начало цельной преамбулы, что позволяет осуществить временную синхронизацию.

4. Архитектура

Архитектура решения основывается на устройстве используемой платформы, а точнее PLutoSDR+, которая состоит из:

- SoC (процессор ARM и FPGA);
- приемопередатчик AD9363;
- периферия.

Обнаружитель реализуется на ПЛИС, однако он не может функционировать сам по себе. Для своей работы обнаружителю требуется получать оцифрованный входной сигнал. Для этого необходимо корректно настроить часть микросхемы AD9363, отвечающую за прием радиосигнала, а также доставить данные с АЦП приемника до обнаружителя, попутно выполнив преобразование данных в удобный для обработки формат.

Управление и настройка микросхемы AD9363 производится путем изменения значений её регистров при помощи протокола SPI. Поэтому для конфигурирования микросхемы требуется драйвер, реализующий SPI. Чтение и форматирование данных АЦП осуществляет Baseband Processor (BBP). Данная сущность получает поток данных от АЦП приемника, преобразует дифференциальные сигналы в CMOS-сигналы, выделяет из общего потока данных 24-битные комплексные отсчёты сигнала для каждого из двух доступных RX-каналов. Таким образом, для получения оцифрованного входного сигнала и его доставки до обнаружителя требуются драйвера, реализующие SPI и BBP.

Для своих микросхем у Analog Devices имеется репозиторий с драйверами, написанными на языке C. Также эти драйвера можно использовать в Python-коде при помощи модуля pyadi-iio[10]. Для своей работы данные драйвера на самом нижнем уровне используют VHDL-драйвера.

Поскольку обнаружитель реализуется на VHDL, то драйвера, написанные на C и Python не подходят, поскольку их использование будет сопряжено с ненужной передачей данных из PL в PS и обратно в PL.

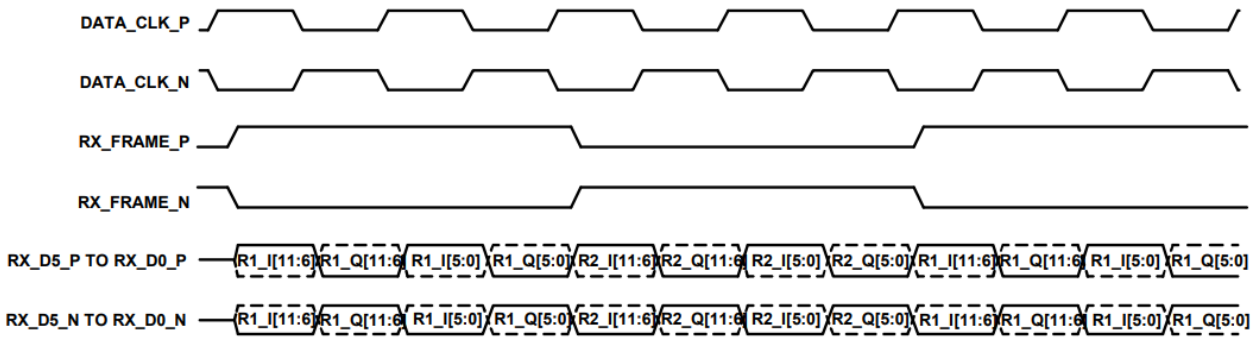
Поэтому необходимо использовать VHDL-драйвера для SPI и BBP. Однако в репозиториях Analog Devices удалось найти лишь реализацию драйвера BBP на языке Verilog. Найденная реализация очень объемна из-за высокой степени универсальности, она подходит для нескольких семейств ПЛИС, позволяет работать BBP в режимах CMOS и LVDS. Такая реализация избыточна, поскольку используется конкретная платформа, а из-за способа распайки AD9363 на данной платформе, работа BBP возможна лишь в режиме LVDS. В силу вышесказанного, автором был самостоятельно реализован VHDL драйвер SPI, а также выполнена упрощенная реализация VHDL драйвера BBP, достаточная для получения отсчетов АЦП приемника.

4.1. SPI

Реализация SPI состоит из двух сущностей. Первая формирует набор сигналов, в соответствии с SPI-протоколом AD9363, представляющий чтение или запись одного байта по указанному адресу. Вторая сущность предназначена для управления SPI-ем с помощью программного кода, исполняемого процессором ARM, для чего используется небольшой register map. Для отображения данных регистров в адресном пространстве процессора используется AXI Memory mapped порт.

4.2. Baseband Processor

Рис. 4: Временная диаграмма приемника.



Для передачи данных приемник использует 3 диффиренциальных

сигнала.

- data_clk – тактовый сигнал;
- rx_frame – сигнал, указывающий на начало отсчета;
- rx_d – 6-битная шина, по которой передаются данные АЦП.

Отсчет сигнала представлен в виде 24-битного комплексного числа, где на действительную и мнимую часть выделено по 12 бит. Для получения одного отсчета необходимо за 2 такта принять 4 порции данных. На рисунке с временной диаграммой приемника один отсчет это группа данных, соответствующая одинаковому значению сигнала rx_frame.

Приемник осуществляет передачу данных в формате edge-aligned source synchronous ddr. Это означает следующее:

- приемник передает как данные, так и тактовый сигнал (clock);
- данные обновляются дважды за период тактового сигнала;
- момент обновления данных совпадает с фронтами тактового сигнала.

В совокупности с тем, что частота тактового сигнала может достигать значения в 245.76 Mhz, становится понятным, что к реализации ВВР предъявляются строгие временные ограничения (timing constraints).

Для выполнения временных ограничений в реализации ВВР используются ряд примитивов, доступных в ПЛИС Xilinx 7-й серии[ссылка на ug471].

- ibufds – преобразователь LVDS-сигнала в LVCMOS-сигнал;
- idelaye2 – примитив, позволяющий вносить задержку до нескольких наносекунд;
- iddr – штатный механизм приема ddr сигналов.

Все перечисленные выше примитивы физически расположены на ПЛИС рядом друг с другом, что позволяет снизить до минимума задержки распространения сигналов в реализации, выполненной на основе данных компонентов.

В реализации примитивы соединены в следующей последовательности: ibufds –> idelaye2 –> iddr. Примитив ibufds выполняет преобразование дифференциального сигнала в обычный, который уже может быть обработан ПЛИС. Следующий за ним блок задержки idelaye2 позволяет внести задержку и за счет этого выровнять фронты сигналов. Затем iddr выделяет отдельно синфазную (I) и квадратурную (Q) части сигнала. Через такую последовательность примитивов проходят не только rx_d, но и rx_frame, что в дальнейшем упрощает синхронизацию. Заметим, что сигнал data_clk проходит лишь через примитив ibufds, а затем, проходя через специальный буфер тактовых сигналов ibufg, выполняет тактирование всех остальных примитивов. Данные с выходов iddr задерживаются на 1 такт с помощью d-триггеров с целью накопления старших и младших бит отсчета. Затем в соответствии с листингом 1 происходит сохранение данных в 48-битный регистр. В зависимости от текущего и задержанного на 1 такт значений сигнала rx_frame в регистр сохраняется отсчет 1-го либо 2-го канала. В результате регистр будет содержать 24-битный отсчет сигнала для обоих RX-каналов.

Листинг 1: Процесс сохранения данных с iddr в регистр

```
reg_adc_data_p : process (clk) begin
      case rx_frame_s & rx_frame is
        when "1111" =>
          adc_data[23:12] <= rx_data_1 & rx_data_1_s;
          adc_data[11: 0] <= rx_data_0 & rx_data_0_s;
        when "0000" =>
          adc_data[47:36] <= rx_data_1 & rx_data_1_s;
          adc_data[35:24] <= rx_data_0 & rx_data_0_s;
        when others =>
            adc_error <= 1'b1;
      end case;
end process;
```

5. Тестирование

Для проверки корректности работы разработанного IP-ядра Vivado IDE предоставлет набор штатных средств, таких как симуляция и интегрированный логический анализатор (IP-ядро ILA).

5.1. Симуляция

Первым механизмом проверки IP-ядра является симуляция. Vivado IDE позволяет выполнить 5 различных видов симуляций. Отличием симуляций является формат представления тестируемого объекта и совокупность учитываемых задержек распространения сигнала. Главным образом используются следующие два вида симуляций.

1. behavеariol simulation – объект представлен в виде HDL-кода, задержки не учитываются.

2. post-implementation timimng simulation – объект представлен в виде размещенного набора синтезированных примитивов, учитываются задержки распространения сигнала в соединениях, а также задержки самих примитивов.

Первая симуляция является наиболее простой, а последнея – наиболее строгой. Симуляции наиболее активно используются при разработке IP-ядра и его промежуточной отладке.

5.2. ILA

Для проверки работоспособности обнаружителя размещенного в ПЛИС, следует использовать готовое IP-ядро под названием Integrated Logic Analyzer (ILA)[15]. Данный блок можно воспринимать как программный осциллограф, который при срабатывании триггера выполняет отписывание набора сигналов и передачу собранных данных на ПК. Для использования этого IP-ядра, его необходимо добавить в VHDL-код отлаживаемой системы и подключить к его входам интересующие линии

данных. Полученную схему необходимо повторно синтезировать и разместить на ПЛИС, поэтому перед использованием ILA рекомендуется прописать ограничения на размещение синтезированных примитивов (placement constraints). В противном случае добавление ILA может повлиять на расположение прочих синтезированных примитивов и нарушить спроектированный до этого VHDL-дизайн.

5.3. Тестовое окружение

Разработанный обнаружитель должен удовлетворять предъявляемым к нему требованиям надежности, что означает наличие пропусков сигнала или ложных обнаружений сигнала с частотами не превышающими заданные. Выполнение таких проверок осуществляется на временных интервалах продолжительностью до нескольких дней. Выполнение столь длительных проверок на одной лишь ПЛИС затруднительно. Однако PLutoSDR+ помимо ПЛИС имеет еще и процессор ARM, поэтому для тестирования разработанного обнаружителя будет разумно дополнительно задействовать процессор. PS будет формировать массивы данных и отправлять их в PL на обработку, результаты работы обнаружителя будут возвращены в PS и проанализированы, а ошибки, в случае их наличия, будут зафиксированы.

5.3.1. Обмен данными между PS и PL

Для реализации рассмотренного сценария требуется механизм обмена данными между PS и PL. Такой механизм может быть реализован на основе DMA или AXI memory-mapped портов.

Первым вариантом реализации такого механизма, подходящим для потоковой передачи данных, является использование готового IP-ядра AXI DMA[14]. Данное IP-ядро позволяет как читать данные из PL в PS, так и записывать данные из PS в PL, для чего используются отдельные каналы на чтение и на запись. За счет AXI-портов регистры этого IP-ядра отображаются в адресном пространстве процессора. Для инициирования DMA операций пользователю необходимо выполнить

определенную последовательность записей в регистры AXI DMA. На основе информации о регистровой модели[ug на AXI DMA] данного IP-ядра автором был реализован драйвер на языке C, позволяющий выполнять обмен числовыми массивами между PS и PL.

Вторым вариантом реализации механизма обмена данных между PS и PL является использование AXI memory-mapped портов. Данный подход используется при создании регистровых моделей и позволяет реализовать собственные протоколы управления для разработанных IP-ядер.

5.3.2. Обмен данными между SDR и инструментальной системой

При отладке обнаружителя может возникнуть необходимость в построении графиков корреляции и спектра сигнала. Подобные задачи легко решить на Python с использованием numpy и matplotlib[4]. Однако сделать то же самое, написав код на C++ под процессор ARM, крайне затруднительно. Для решения данной задачи был разработан механизм передачи данных между PlutoSDR+ и инструментальной системой. Pluto оснащена USB-C разъемами, которые используются для её подключения к ПК и через которые PLuto может передавать данные по протоколу UART. Таким образом, Pluto может отправлять из PS данные по UART на последовательный порт. В это же время Python-программа, работающая на инструментальной системе, может вычитывать данные из этого же порта. Принятые данные можно обрабатывать и визуализировать требуемым образом. К недостаткам данного подхода можно отнести низкую скорость UART, которая составляет 115200 бод. Однако данной скорости достаточно для передачи численных массивов длиной до нескольких тысяч элементов. При необходимости достижения больших скоростей может быть рассмотрен переход с UART на

[4]В репозиториях организации имеется большое количество готового python-кода, предназначенного для отладки алгоритмов ЦОС. В тоже время запуск python-кода на SDR хоть и возможен (проект PYNQ), однако затруднителен и неэффективен т.к. вычислительные мощности инструментальной системы много больше. Поэтому кажется привлекательным отправить данные из SDR на компьтер и проводить отладку там.

Ethernet.

Заключение

В ходе работы за весенний семестр были достигнуты следующие результаты.

1. Реализована математическая модель алгоритма обнаружения цифровых сигналов на языке Python.

2. На основе результатов моделирования осуществлен сбор предельных характеристик обнаружителя, при которых выполняется успешное обнаружение сигналов.

3. На основе собранных эмпирические данных осуществлен вывод математических выражений, являющихся аппроксимацией параметров алгоритма.

4. Осуществлена сборка и синтез прошивки ПЛИС на базе драйверов фирмы Analog Devices, обеспечивающую интерфейс управления микросхемой ad9361, а также интерфейс получения отсчетов оцифрованного сигнала.

5. На базе IP-ядер фирмы Xilinx реализовано IP-ядро переноса по частоте.

6. На базе TCL-скриптов реализована возможность получения оцифрованных данных от АЦП на ПК.

Список литературы

[1] AMD Vitis IDE. — URL: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis.html (дата обращения: 6 января 2025 г.).

[2] AMD Vivado IDE. — URL: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html (дата обращения: 6 января 2025 г.).

[3] AMD Zynq 7000. — URL: https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html (дата обращения: 6 января 2025 г.).

[4] Block Memory Generator v8.4 Product Guide (PG058). — URL: https://docs.amd.com/v/u/en-US/pg058-blk-mem-gen (дата обращения: 18 июня 2025 г.).

[5] Complex Multiplier LogiCORE IP Product Guide (PG104). — URL: https://docs.amd.com/r/en-US/pg104-cmpy (дата обращения: 18 июня 2025 г.).

[6] DDS Compiler LogiCORE IP Product Guide (PG141). — URL: https://docs.amd.com/r/en-US/pg141-dds-compiler (дата обращения: 18 июня 2025 г.).

[7] Fast Fourier Transform v9.1 LogiCORE IP Product Guide. — URL: https://www.xilinx.com/support/documents/ip_documentation/xfft/v9_1/pg109-xfft.pdf (дата обращения: 18 июня 2025 г.).

[8] Linaro Linux. — URL: https://old.linaro.org/core-technologies/linux-kernel/ (дата обращения: 6 января 2025 г.).

[9] Mengali Umberto. Synchronization techniques for digital receivers. — Ney York : Plenum Press, 1997. — ISBN: 0-306-45725-3.

[10] Python interfaces for ADI hardware with IIO drivers. — URL: https://github.com/analogdevicesinc/pyadi-iio (дата обращения: 6 января 2025 г.).

[11] RF transceiver AD9363. — URL: https://www.analog.com/en/products/ad9363.html (дата обращения: 6 января 2025 г.).

[12] VHDL 2019 standard. — URL: https://ieeexplore.ieee.org/document/8938196 (дата обращения: 6 января 2025 г.).

[13] Verilog 2005 standard. — URL: https://ieeexplore.ieee.org/document/1620780 (дата обращения: 6 января 2025 г.).

[14] Xilinx AXI DMA. — URL: https://users.ece.utexas.edu/~mcdermot/arch/articles/Zynq/pg021_axi_dma.pdf (дата обращения: 6 января 2025 г.).

[15] Xilinx ILA. — URL: https://www.xilinx.com/products/intellectual-property/ila.html (дата обращения: 6 января 2025 г.).

[16] Xilinx Linux Kernel. — URL: https://github.com/Xilinx/linux-xlnx (дата обращения: 6 января 2025 г.).

[17] Сергиенко А. Б. Цифровая обработка сигналов. — Санкт-Петербург : Питер, 2002. — ISBN: 5-318-00666-3.

[18] Ю. П. Гришин В. П. Ипатов. Радиотехнические системы. — Москва : Высшая школа, 1990. — ISBN: 5-06-000687-5.