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. 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.

Процесс обнаружения требует выполнения потоковой обработки данных в режиме реального времени. При реализации обнаружителя на процессоре общего назначения зачастую возникают проблемы с производительностью, вызванные тем, что процесс обнаружения требует выполнения большого числа однотипных операций, а процессор не является оптимальным вычислителем для решения такой задачи. По этой причине обнаружитель реализуется в виде отдельной микросхемы, прототип которой разрабатывается с использованием ПЛИС (программируемой логической интегральной схемы).

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. Вместо этого код, написанный под ПЛИС на языках описания аппаратуры (HDL), используется для описания структуры и поведения разрабатываемой микросхемы.

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 общего назначения на одном общем кристалле, получая так называемый System-on-Chip(SoC). Помимо процессорной системы (PS) и

программируемой логики (PL) SoC оснащён разнообразной периферией, позволяющей организовывать эффективное взаимодействие с таким устройством, например, в целях отладки. Мощностей PS достаточно для развертывания операционной системы Linux[4, 11]. 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. 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[6].

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[7] и Verilog[8]. 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. 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[5]. 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.

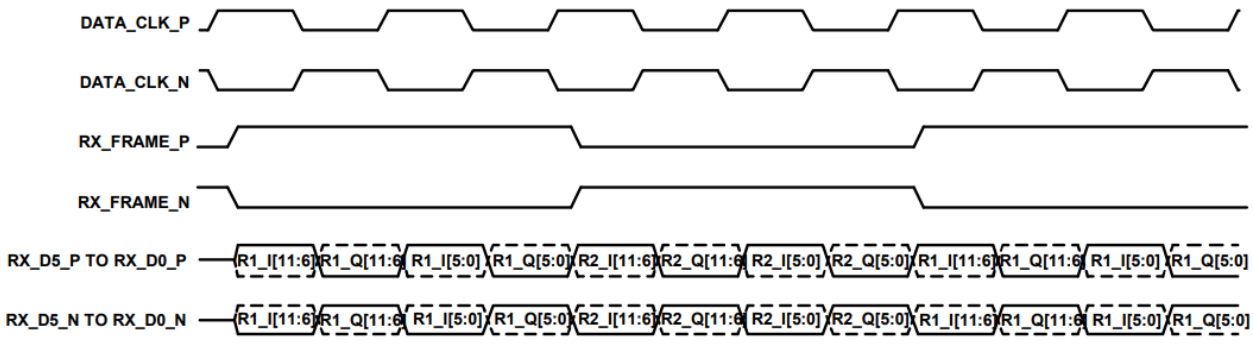
Поэтому необходимо использовать VHDL-драйвера для SPI и BBP. Однако в репозиториях Analog Devices удалось найти лишь реализацию драйвера BBP на языке Verilog. Найденная реализация очень объемна из-за высокой степени универсальности, она подходит для нескольких семейств ПЛИС, позволяет работать BBP в 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.

3.1. 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.

3.2. Baseband Processor

Рис. 1: 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;
```

4. Тестирование

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

4.1. 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. behaveariol 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.

4.2. ILA

Для проверки работоспособности обнаружителя размещенного в ПЛИС, следует использовать готовое IP-ядро под названием Integrated Logic Analyzer (ILA)[10]. 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.

4.3. 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.

4.3.1. 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[9]. 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 опреде-

ленную 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.

4.3.2. Обмен данными между SDR и HOST-ПК

При отладке обнаружителя может возникнуть необходимость в построении графиков корреляции и спектра сигнала. Подобные задачи легко решить на Python с использованием numpy и matplotlib. Однако сделать то же самое, написав код на C++ под процессор ARM, крайне затруднительно. Причиной этого является использование Xilinx собственной урезанной версии стандартной библиотеки языка C++. Для решения данной задачи был разработан механизм передачи данных между PlutoSDR+ и Host-ПК. 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 на Host-ПК 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 Ethernet.

Заключение

В ходе работы за осенний семестр были достигнуты следующие результаты.

1. Создана инфраструктура для разработки и отладки IP-ядра обнаружителя.
   (a) Разработаны VHDL-драйвера для приемопередатчика AD9363, позволяющие подать на вход обнаружителя целевой сигнал в оцифрованном виде.
   (b) Реализованы средства обмена данными между ПЛИС, процессорной системой и HOST-ПК, позволяющие осуществлять промежуточную отладку разрабатываемого IP-ядра.

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

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

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

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

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

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

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

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