УДК 621.396.96: 681.32

ОТЛАДКА АЛГОРИТМОВ ОБРАБОТКИ РАДИОЛОКАЦИОННЫХ СИГНАЛОВ В СИСТЕМЕ GNU RADIO

Липатников В.С., *инженер научно-технического отдела АО «Ордена Трудового Красного Знамени Всероссийский научно-исследовательский институт радиоаппаратуры» (АО «ВНИИРА»), г. Санкт-Петербург, e-mail: meteo@vniira.ru;*
Чинёнов Д.А., *инженер-программист научно-технического отдела АО «ВНИИРА» г. Санкт-Петербург, e-mail: meteo@vniira.ru/*

RADAR SIGNAL PROCESSING ALGORITHMS DEBUG IN GNU RADIO SYSTEM

Lipatnikov V.S., Chinenov D.A.

The radar signal processing device model is described. The free and open-source SDR-system GNU Radio used as the modeling system. The short review of GNU Radio modeling main opportunities was given. Every fragment of radar signal processing device simple model is described in detail.

Key words: radar signal, phase shift keying, modeling, GNU Radio.

Ключевые слова: радиолокационный сигнал, фазовая манипуляция, моделирование, GNU Radio.

Описана модель устройства обработки радиолокационного сигнала. В качестве системы моделирования использована свободно распространяемая SDR-система с открытыми исходными кодами GNU Radio. Проведен краткий обзор ее основных возможностей по моделированию и подробно описан каждый фрагмент упрощенной модели устройства обработки радиолокационного сигнала.

Введение

В настоящее время при проектировании устройств цифровой обработки радиолокационных сигналов (ЦОС) большое внимание уделяется компьютерному моделированию. При отладке алгоритмов работы устройств ЦОС моделируют не только отлаживаемые устройства, но и обрабатываемые сигналы.

При работе с конкретной элементной базой отработку алгоритмов производят, как правило, в интегрированных средах разработки, возможно с привлечением дополнительных инструментов. Например, проектирование на базе программируемых логических интегральных схем (ПЛИС) фирма Altera производят с помощью интегрированной среды Quartus, а разработку программ для цифровых процессоров обработки сигналов (ЦПОС) фирма Texas Instruments осуществляют в среде Code Composer Studio.

Для работы с этими средами в качестве удобного инструмента разработки прототипов устройств и алгоритмов обработки, а также визуализации массивов данных, активно используется широко распространенный пакет MATLAB/Simulink фирма MathWorks. Его тесная интеграция со средами разработки позволяет отлаживать не только прототипы устройств создаваемые с его помощью, но и реализованные на базе элементов, предоставляемых используемой средой проектирования. Так, например, взаимодействуют друг с другом среда Quartus и пакет Simulink. Также, из разработанного с помощью MATLAB/Simulink прототипа устройств можно получить автоматически сгенерированный код для встраивания его в проектируемое устройство. Однако, для достижения оптимального результата, алгоритмы, отработанные в MATLAB/Simulink, вручную переводят на язык описания аппаратуры (для ПЛИС) или язык программирования (для ЦПОС). Тем не менее, первоначальная разработка и отладка алгоритмов ЦОС в пакете MATLAB – довольно распространенное явление, на ограничение использования которого может повлиять только его высокая цена. Модули интеграции со средами разработки могут распространяться по отдельным дополнительным лицензиям, что может повысить суммарную стоимость владения средствами разработки.

Поиск среди свободно распространяемого программного обеспечения показал, что при разработке устройств обработки сигналов могут использоваться не только коммерческие продукты. Например, в [1] упомянуты свободно распространяемые программные продукты, используемые при разработке и отладке прототипов устройств на основе ПЛИС. Использование пакета Icarus Verilog позволит смоделировать работу разрабатываемого на языке описания аппаратных схем Verilog или SystemVerilog устройства, а генерация входных и визуализация выходных данных может быть осуществлена с помощью языка Python и его библиотек numpy и matplotlib.

При разработке модулей обработки сигналов радиолокационных устройств удобно компоновать их из блоков, обеспечивающих базовые функции ЦОС, таких как, например, накопление отсчетов, фильтрация или выполнение преобразования Фурье. Для моделирования устройств, состоящих из таких базовых блоков, с использованием свободно распространяемого программного обеспечения предложен набор инструментов GNU Radio [2]. Попробуем смоделировать с его помощью основные устройства обработки сигналов, которые могут

применяться в радиолокационных станциях.

Обзор особенностей GNU Radio

GNU Radio представляет собой набор программных инструментов (Software Development Tool), включающих в себя библиотеку компонентов, реализованных на языке программирования C++ с предоставлением интерфейса прикладного программирования (API) на языках C++ и Python, и ряд вспомогательных утилит. Компоненты библиотеки представляют собой множество классов, в которых реализована функциональность различных блоков обработки сигналов. Среди блоков встречаются источники сигналов, приемники сигналов и, собственно, блоки обработки. Блоки обработки могут быть синхронными (количество отсчетов входных сигналов равно количеству отсчетов в выходных сигналах), децимирующими (с понижением количества выходных отсчетов) и интерполирующими (с повышением количества выходных отсчетов).

Из блоков-источников сигналов часто используются встроенный программный генератор сигнала синусоидальной, пилообразной или прямоугольной формы, генератор шума, источник постоянного сигнала, файл с набором данных в двоичном формате, звуковой файл в формате WAV, аудиоплата ПК или подключенный совместимый аппаратный ВЧ-модуль, для которого в библиотеке имеется соответствующий модуль сопряжения. Блоками-приемниками могут быть также ВЧ-модуль, аудиоплата, звуковой или произвольный двоичный файл. Блоки-приемники, позволяющие визуализировать принимаемый сигнал во временной или частотной области, подобно тому, как он отображается на экране осциллографа или анализатора спектра, очень удобны для моделирования и отладки работы устройств ЦОС.

Среди блоков обработки могут встречаться как простые, например, задержка сигнала на заданное количество отсчетов или выполнение элементарной арифметической операции над каждым входным отсчетом, так и более сложные, например, фильтры или блоки, выполняющие преобразование Фурье. Каждый блок обработки принимает один или несколько входных потоков данных, обрабатывает их и генерирует один или несколько выходных потоков данных (количество которых может отличаться от количества входных потоков) [2]. Некоторые блоки обработки принимают на вход или передают на выход цифровые отсчеты только в виде выборок фиксированной длины (векторов), как, например, в блоке, выполняющем преобразование Фурье. В цепочке соединенных друг с другом блоков тип выходных данных одного блока, должен совпадать с типом входных данных следующего. При возникновении необходимости ввода данных одного типа в блок со входами, принимающими данные другого типа, можно использовать блоки-преобразователи типов.

При разработке приложений с использованием GNU Radio на языках C++ или Python используется следующая концепция. Создается объект класса блока верхнего уровня top_block или класса, унаследованного от него. Создаются объекты блоков-компонентов проектируемого устройства, блоков генерации и визуализации и соединяются друг с другом с помощью метода connect() объекта класса top_block. Схему таких соединений блоков принято называть потоковым графом (flowgraph). После создания всех соединений производится запуск схемы на выполнение с помощью метода start(). Если присутствуют блоки, использующие графический интерфейс пользователя (блоки визуализации), то требуется обеспечить необходимую программную обвязку для сопряжения с используемой соответствующими блоками визуализации графической библиотекой.

Проектирование может сопровождаться множественным использованием повторяющихся фрагментов схем. Для повышения удобства использования таких фрагментов GNU Radio позволяет создавать пользовательские блоки, состоящие из компонентов, входящих в комплект поставки библиотеки. Такие пользовательские блоки называются иерархическими. Возможно создание пользовательских блоков, не использующих другие блоки, а выполняющих определенные пользователем функции. Эта особенность является очень полезной во время разработки и реализации подпрограмм ЦОС на языках C/C++, поскольку разработанная подпрограмма, протестированная в виде пользовательского блока с помощью GNU Radio, может быть перенесена с минимальными изменениями в рабочий проект для процессора общего назначения или специализированного процессора ЦОС, среда разработки для которого поддерживает проектирование на языках C/C++. Для разработки пользовательских блоков предоставлена утилита gr_modtool, упрощающая их сопряжение с остальной частью библиотеки.

Поскольку разработка моделей на языках программирования C++ и Python с использованием библиотеки GNU Radio требует определенной подготовки и более глубоких знаний API библиотеки, может быть использовано средство визуального проектирования GNU Radio Companion (GRC). С его помощью схема соединений блоков (потоковый граф) может быть представлена в удобной для восприятия форме.

Программа GNU Radio Companion позволяет создавать как схемы верхнего уровня, так и иерархические пользовательские блоки. Разработанные в GRC потоковые графы конвертируются в сценарии на языке Python, сохраняемые в текущем каталоге, а для иерархических блоков – в каталоге с настройками GRC, откуда они считываются при последующем запуске программы и могут использоваться наряду со стандартными блоками GNU Radio.

При разработке модели части тракта обработки радиолокационного сигнала, описанной ниже, некоторые ее фрагменты реализованы в виде пользовательских блоков на языке C++, другие – в виде иерархических блоков, выполненных в GRC. Для отладки фрагментов разработано несколько схем верхнего уровня, в некоторых контрольных точках которых подключены соответствующие блоки визуализации.

Рассмотрим в следующем разделе модель тракта обработки радиолокационного сигнала.

Моделирование устройств обработки сигналов

Пример тракта цифровой обработки сигналов когерентной доплеровской радиолокационной станции приведен в [3]. Рассмотрим для реализации с помощью GNU Radio похожую общую модель, примерная функциональная схема которой приведена на рис. 1.

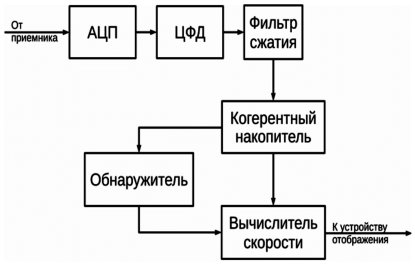

Рис. 1. Общая функциональная схема тракта цифровой обработки РЛС

Типичный тракт цифровой обработки начинается с аналого-цифрового преобразователя (АЦП), на который сигнал на промежуточной частоте (ПЧ) поступает от приемника. После АЦП цифровые отсчеты сигнала поступают на цифровой фазовый детектор (ЦФД), который может быть выполнен в виде отдельной микросхемы или реализован на основе ПЛИС. Отсчеты синфазной (I) и квадратурной (Q) составляющих на пониженной частоте дискретизации с выхода ЦФД поступают на вход фильтра сжатия. Фильтр сжатия может быть реализован программно в цифровом процессоре обработки сигналов (ЦПОС), а при выборе ПЛИС с достаточным количеством ресурсов целесообразнее реализовать его аппаратно. Далее отсчеты сжатого сигнала накапливаются в когерентном накопителе, после чего производится обработка накопленных отсчетов сигналов, отраженных от искомых целей для решения задачи их обнаружения и вычисления доплеровской скорости. Эти процедуры выделены на схеме, приведенной на рис. 1, в отдельные функциональные блоки обнаружителя и вычислителя скорости. Проведем моделирование каждого блока и проверку его работы, используя набор инструментов GNU Radio.

Модель цифрового фазового детектора

Потоковый граф моделируемой схемы цифрового фазового детектора (ЦФД) тракта обработки радиолокационного сигнала построенный в программе GNU Radio Companion приведен на рис. 2.

После точки входа в схему блок «Rail» ограничивает входной сигнал, отсекая значения, не входящие в указанный пользователем диапазон, и присваивая им максимальное или минимальное значение соответственно при выходе за верхнюю или нижнюю границу (имитация перегрузки АЦП). Блок «Float to Short» конвертирует входной поток действительных значений в поток целых 16-разрядных значений с коэффициентом масштабирования 8192 (2^{13}). Таким образом, блоки «Rail» с параметрами среза [−1;+1] и «Float to Short» с коэффициентом масштабирования 8192 имитируют работу 14-битного АЦП.

Далее сигнал идет в блок квадратурного демодулятора «Quadrature demodulator», где раскладывается на I и Q составляющие. Пользовательский блок «Quadrature demodulator» реализован на языке C++. Поскольку промежуточная частота в данном случае равна трем четвертям частоты дискретизации, его реализация может быть сильно упрощена за счет отсутствия умножителей и генератора колебаний, как описано в [4].

После разложения сигнала на квадратуры, каждая из них подвергается обработке в каскадном интегрально-гребенчатом фильтре (Cascaded integral-comb filter – CIC-filter). Такие фильтры являются фильтрами с бесконечной импульсной характеристикой, передаточная функция которых имеет вид [5]:

$$H_K(z) = \left[\frac{1-z^{-D}}{1-z^{-1}}\right]^K, \qquad (1)$$

где D – количество тактов задержки в гребенчатой части, а K – количество каскадов фильтра.

GNU Radio предоставляет для использования стандартный блок «IIR Filter» (рис. 3) и для реализации CIC-фильтра порядка K = 1 с задержкой D = 4 можно указать ряды значений (1, 0, 0, 0, -1) и (1, 1) в качестве параметров «Feed-forward Taps» и «Feedback Taps» соответственно.

Однако, этот блок работает только с действительными данными, поэтому блок «CIC Filter» реализован в виде пользовательского блока на языке C++ и вставлен в схему ЦФД. Потоковый граф с реализацией схемы CIC-фильтра стандартными блоками GNU Radio в программе GNU Radio Companion, приведенный на рисунке 4, подробно описан в [2].

Далее блок «Keep 1 in N» с параметром «N» равным «2» отбрасывает одно значение из двух, осуществляя таким образом децимацию сигнала. После первой деци-

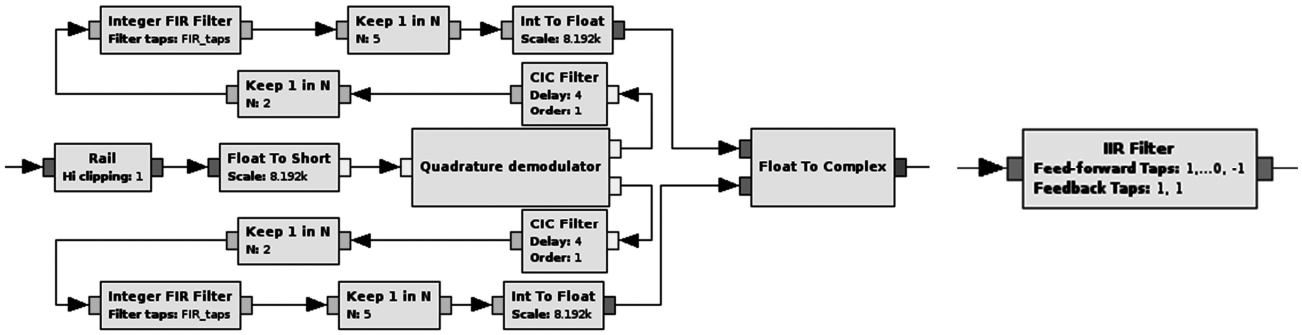

Рис. 2. Потоковый граф модели ЦФД в программе GNU Radio Companion

Рис.3. Блок «IIR Filter» в GRC

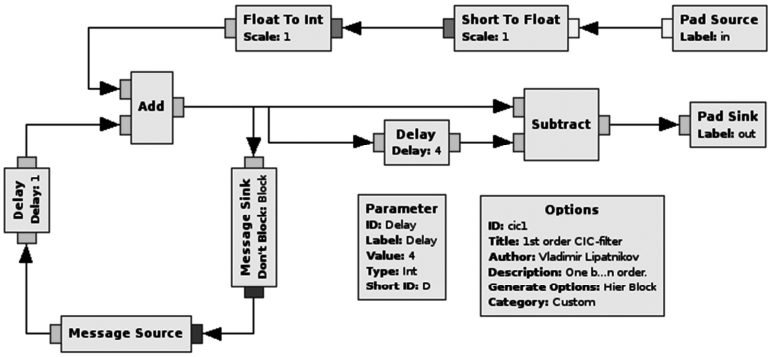

Рис. 4. Потоковый граф пользовательской реализации CIC-фильтра

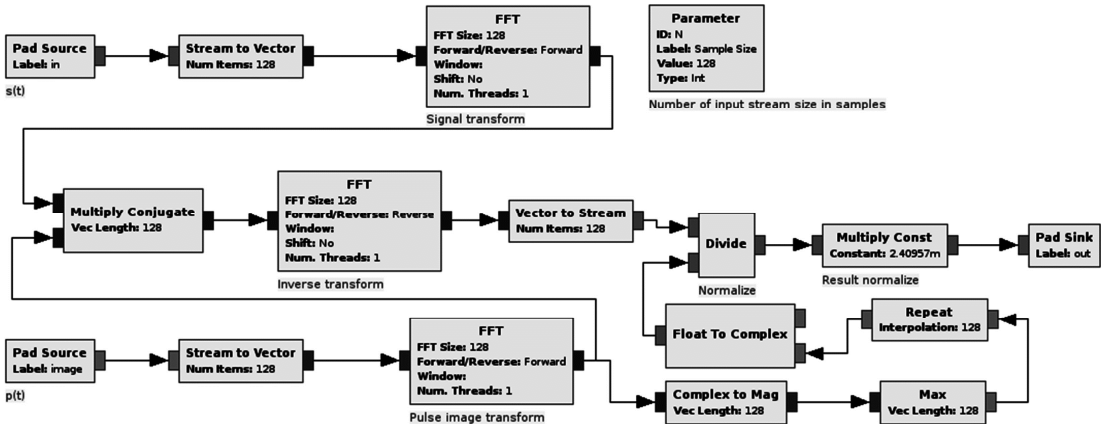

Рис. 5. Потоковый граф фильтра сжатия

мации сигнал подается на фильтр нижних частот с конечной импульсной характеристикой (КИХ), реализованный в виде пользовательского блока «Integer FIR Filter» и включенного после него блока «Keep 1 in N» с параметром «5». В данном случае снова использован пользовательский блок вместо стандартного блока «Decimating FIR Filter», поскольку стандартный блок использует арифметику с плавающей запятой, а нам необходимо проверить алгоритм работы с целочисленной арифметикой.

Далее, согласно схеме, синфазная и квадратурная составляющие из целочисленных значений конвертируются в действительные и объединяются в блоке «Float to Complex» в качестве действительной и мнимой частей комплексного сигнала, следующего на выход из данного пользовательского блока для дальнейшей обработки в устройствах, позволяющих использовать арифметику с плавающей запятой.

Модель фильтра сжатия

Предположим, что фильтр сжатия будет реализован на микропроцессорном устройстве, имеющем арифметику с плавающей запятой, поэтому для его моделирования поначалу могут быть использованы блоки GNU Radio, входящие в комплект его поставки.

Рассмотрим потоковый граф простого фильтра сжатия, который может быть реализован через блоки быстрого преобразования Фурье (БПФ), приведенный на рис. 5.

У фильтра имеются два входа: первый – для отсчетов принятого сигнала $\boldsymbol{p}=\{p_0, p_1, \ldots, p_{N-1}\}$, второй – для выборки зондирующего импульса $\boldsymbol{s}=\{s_0, s_1, \ldots, s_{N-1}\}$. Выходной сигнал фильтра должен быть равен

$$\boldsymbol{s}_{сж.} = \frac{4\pi^2}{N} \cdot \frac{F^{-1}\{\boldsymbol{P}^* \cdot \boldsymbol{S}\}}{max\{|\boldsymbol{P}|\}}, \qquad (2)$$

где $\boldsymbol{P}=\{\boldsymbol{p}\}$, $\boldsymbol{S}=\{\boldsymbol{s}\}$, $F\{\}$ – оператор дискретного преобразования Фурье (ДПФ), $F^{-1}\{\}$ – оператор обратного дискретного преобразования Фурье (ОДПФ), * – комплексное сопряжение, N – количество отсчетов принятого сигнала.

На схеме, приведенной на рис. 5, выборка цифровых отсчетов зондирующего импульса имеет действительный тип данных, а выборка отсчетов принятого сигнала после ЦФД – комплексный. Параметр N (блок «Parameter») пользовательского блока, реализующего фильтр сжатия, определяет количество отсчетов в выборке сигнала. Поскольку блоки «FFT» GNU Radio работают не с единичными отсчетами, а с векторами данных, сначала входные сигналы накапливаются в буферах и преобразовываются в векторы по N отсчетов в блоках «Stream to Vector». Далее в блоке «Multiply Conjugate» вектор результатов ДПФ принятого сигнала умножается на комплексно-сопряженный вектор Фурье-образа цифровых отсчетов зондирующего сигнала, и выполняется обратное преобразование Фурье в следующем блоке «FFT». Далее блоком «Vector to Stream» выполняется преобразование вектора выходных данных блока «FFT» в поток отсчетов. Выходной сигнал блока нормируется с использованием следующих блоков GNU Radio:

1. «Complex to Mag» – вычисление амплитуды комплексного результата;

2. «Max» – вычисление значения максимальной амплитуды комплексного результата (одно выходное значение при входном векторе длительностью N отсчетов);

3. «Repeat» – повтор выходного значения блока «Max» столько раз, сколько отсчетов во входном векторе блока «Max»;

4. «Float to Complex» – преобразование действительного типа данных в комплексный, так как последующее деление в блоке «Divide» осуществляется над комплексными отсчетами.

Для уменьшения уровня боковых лепестков выходного сигнала фильтра сжатия, он может быть построен с использованием метода обратных пульсаций [6]. Выходной сигнал такого фильтра определяется как

$$\boldsymbol{s}_{сж.} = \frac{4\pi^2}{N} \cdot \frac{F^{-1}\{|\boldsymbol{Q}|^2 \cdot \boldsymbol{S} / \boldsymbol{P}\}}{max\{|\boldsymbol{Q}|^2 / |\boldsymbol{P}|\}}, \quad (3)$$

где $\boldsymbol{Q}=\{\boldsymbol{q}\}$, а $\boldsymbol{q}=\{q_0, q_1, \ldots, q_{N-1}\}$ – элементарный импульс кодовой последовательности, использованной для генерации зондирующего сигнала. Потоковый граф модели такого фильтра, построенный из стандартных блоков GNU Radio, приведен на рис. 6. В отличие от потокового графа фильтра реализованного в виде пользовательского иерархического блока, приведенного на рис. 5, потоковый граф данного фильтра имеет еще один параметр l – количество отсчетов в одном элементе кодовой последовательности зондирующего сигнала (второй блок «Parameter»). Этот параметр использован для фор-мирования вектора заданных отсчетов $\boldsymbol{Q}^2$ в блоке-источнике «Vector Source», принимающего в качестве параметра выражение на языке Python с подстановкой переменных Q и q, присутствующих на потоковом графе в блоках «Variable», тоже принимающих выражения на языке Python в качестве своих параметров. Вектор выходных отсчетов блока «Vector Source» делится на вектор Фурье-образа зондирующего импульса. Полученный результат перемножается с Фурье-образом принятого сигнала. Нормировка выходных результатов фильтра выполняется так же, как и в потоковом графе простого фильтра сжатия.

Для проверки и сравнения моделей оптимального и подоптимального фильтров сжатия подготовлен потоковый граф, приведенный на рис. 7-9. Для удобства визуального восприятия, разделения функциональных групп потокового графа и исключения его загромождения соединительными линиями использованы блоки «Virtual Sink» и «Virtual Source».

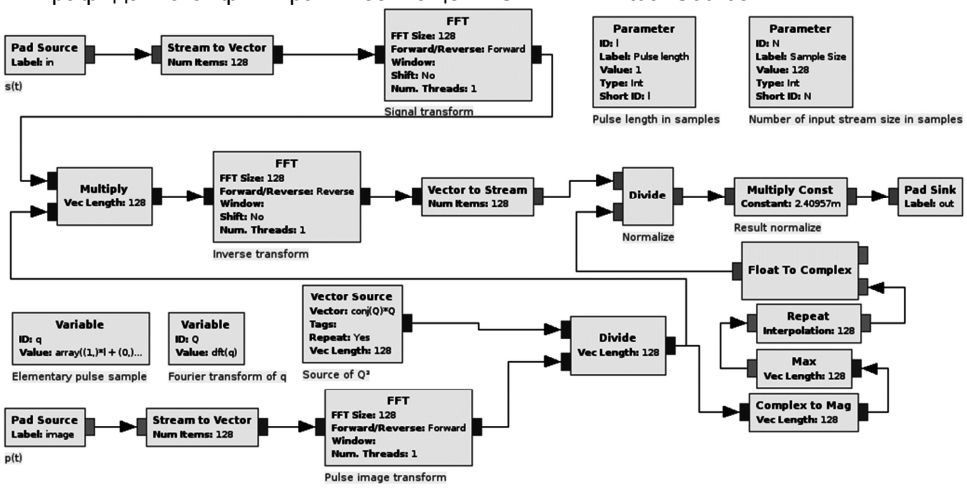

Рис. 6. Потоковый граф подоптимального фильтра сжатия

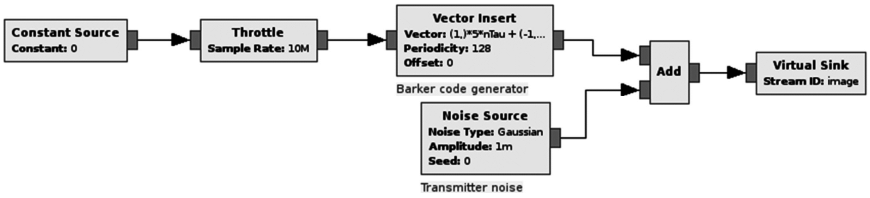

Рис. 7. Потоковый граф проверки фильтров сжатия: генерация импульсов

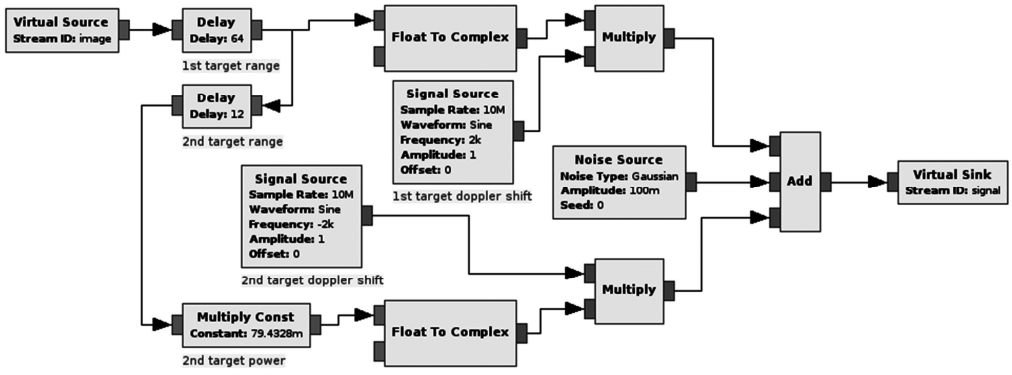

Рис. 8. Потоковый граф проверки фильтров сжатия: имитация двух целей

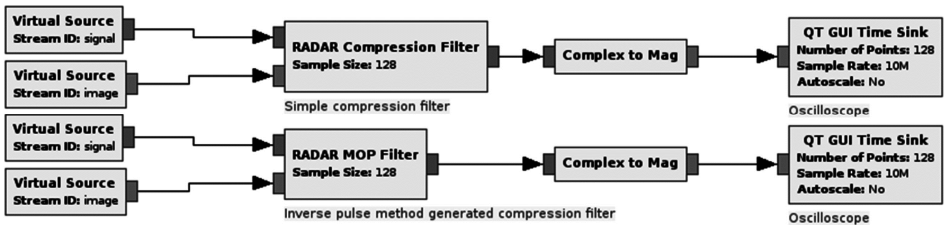

Рис. 9. Потоковый граф проверки фильтров сжатия: схема включения фильтров

Последовательность блоков на рис. 7 формирует 13-символьную кодовую последовательность Баркера. Блок «Throttle» нужен для ограничения времени работы процессора в случае, если его производительности более чем достаточно для работы на заданной частоте дискретизации. Блок «Vector Insert» вставляет в поток отсчетов, равных нулю, отсчеты, соответствующие кодовой последовательности. В блоке «Noise Source» имитируются шумы передатчика, которые складываются с исходной кодовой последовательностью в блоке «Add». С помощью блока «Virtual Sink» с идентификатором «image» осуществляется соединение со следующим фрагментом без загромождения схемы. Модель сигнала выполнена в квадратурах на нулевой частоте.

На рис. 8 показан потоковый граф генератора имитационного эхо-сигнала от двух целей с разными скоростями (доплеровскими частотами) и разной ЭПР (мощностью сигнала).

Блок «Virtual Source» с идентификатором «image» продолжает схему, приведенную на рис. 7, окончившуюся блоком «Virtual Sink» с таким же идентификатором. Два блока «Delay» задерживают сигнал на разное количество отсчетов: второй блок задерживает сигнал на такое количество отсчетов, чтобы второй сигнал после сжатия наложился на боковой лепесток первого сжатого сигнала. Амплитуда второго сигнала уменьшена на 22 дБ с помощью блока «Multiply Const». Формирование доплеровских сдвигов осуществляется в блоках «Multiply» в комплексном виде подмешиванием синусоидальных сигналов, сгенерированных блоками «Signal Source». Сигналы складываются в блоке «Add» с добавлением белого шума заданной мощности, сгенерированного блоком «Noise Source». Получившийся сигнал может быть передан в другую часть схемы с помощью блока «Virtual Sink» с идентификатором «signal».

На фрагменте потокового графа, приведенном на рис. 9, показано подключение сигналов от блоков «Virtual Sink» с идентификаторами «signal» и «image» к пользовательским блокам «RADAR Compression Filter» и «RADAR MOP Filter» с разными реализациями фильтров сжатия, описанными выше.

В блоках «Complex to Mag» вычисляются амплитуды выходных комплексных сигналов фильтров и, затем, визуализируются с помощью блоков «GUI Time Sink». Окно с визуализацией результатов работы фильтров сжатия блоками «GUI Time Sink» после запуска потокового графа на выполнение показано на рис. 10. Верхняя осциллограмма отражает результат оптимального фильтра сжатия, нижняя – подоптимального.

Как видно из рисунка, оптимальная фильтрация приводит к высокому уровню боковых лепестков, в которых теряется сигнал от второй имитируемой цели, а в сжатом подоптимальным фильтром сигнале можно четко разделить две составляющие.

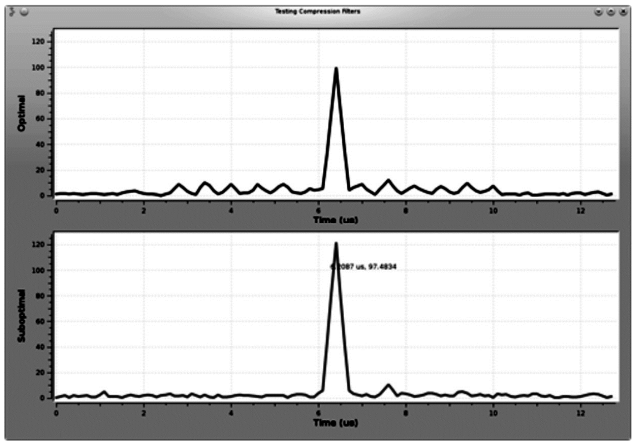

Рис. 10. Осциллограмма результатов сжатия

Таким образом, собрав несложную схему проверки в программе GNU Radio Companion, мы провели наглядное сравнение двух алгоритмов фильтрации.

Модель когерентного накопителя и обнаружителя

Пример потоковых графов блоков, моделирующих когерентный накопитель, пороговый обнаружитель и вычислитель доплеровского сдвига для восьми интервалов по дальности приведен на рис. 11 – 13. Как видно из рис. 11, в GRC схема выглядит довольно громоздкой даже для восьми отсчетов по дальности. Поскольку при когерентном накоплении сигнала вычисления производятся в каждой дискрете по дальности, которых может быть довольно много, использование программы GNU Radio Companion для моделирования дальнейших устройств обработки нецелесообразно. Для этих целей лучше реализовать на языке C++ пользовательский блок когерентного накопителя с вычислением преобразования Фурье, мощности сигнала, пороговой обработкой и оценкой доплеровской скорости в каждой дискрете по дальности в виде иерархического блока GNU Radio, который потом может быть включен в общий потоковый граф модели радиолокатора. Фрагмент создания блоков «Stream to Streams» и «Stream Mux» для иерархического пользовательского блока «Doppler Accumulator» и соединений между ними на языке Python приведен на рис. 14.

Пример вида экрана индикатора скорость-дальность (номер доплеровского канала – время), полученного с помощью блока «GUI Time Sink», приведенного на рис. 13, показан на рис. 15. На данном рисунке цели обнаружены на разных дальностях в 12-м и 4-м доплеровских каналах из 16.

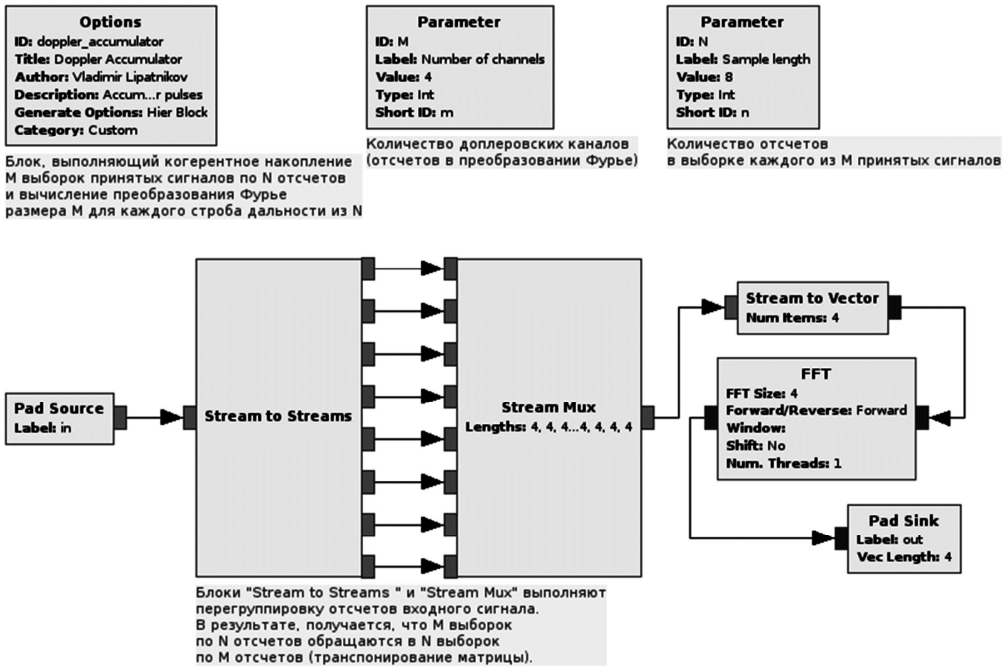

Рис. 11. Потоковый граф когерентного накопителя

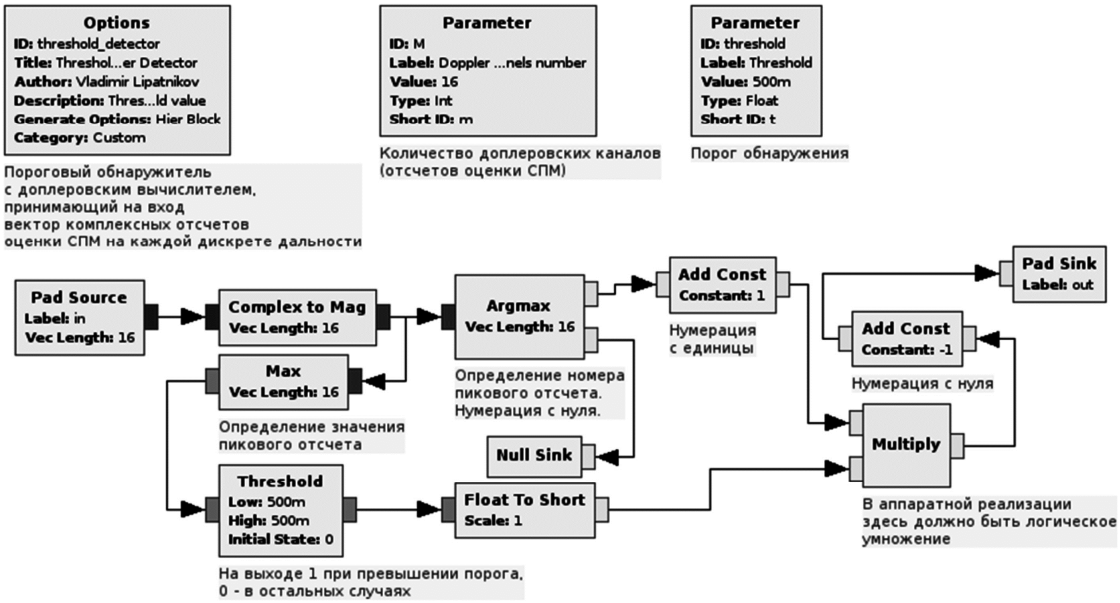

Рис. 12. Потоковый граф вычислителя доплеровской скорости с пороговым обнаружителем

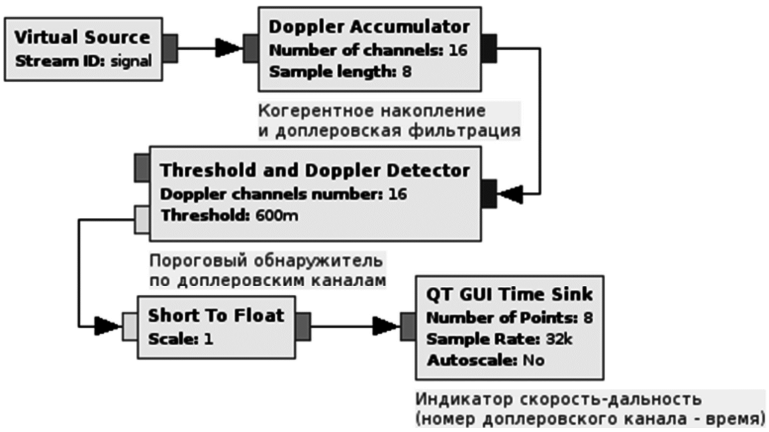

Рис. 13. Потоковый граф подключения индикатора скорость-дальность

```
#...Blocks
self.blocks_stream_to_streams_0 =
    blocks.stream_to_streams(gr.sizeof_gr_complex*1, N)
self.blocks_stream_mux_0 =
    blocks.stream_mux(gr.sizeof_gr_complex*1, ((M,)*N))
#...Connections
for i in range(self.N):
    self.connect((self.blocks_stream_to_streams_0, i),
                 (self.blocks_stream_mux_0, i))
```

Рис. 14. Фрагмент листинга реализации блока «Doppler Accumulator» на языке Python

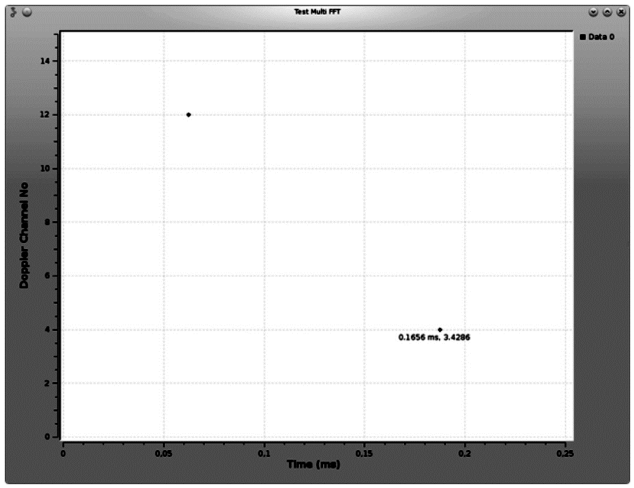

Рис. 15. Вид индикатора скорость-дальность (номер канала – время)

Заключение

Рассмотренная модель тракта обработки радиолокационного сигнала является сильно упрощенной, но вполне достаточной для демонстрации некоторых возможностей отладки алгоритмов ЦОС в системе GNU Radio. Постепенный переход от фрагментов модели, выполненных в виде иерархических блоков GRC, к пользовательским блокам, реализованным на языке C++, позволит приблизить модель к созданию реального уст-ройства. Но при разработке первоначальных моделей в GRC лучше не увлекаться чрезмерным загромождением потоковых графов, поскольку это приводит не только к ухудшению восприятия, но и к замедлению работы самой программы визуального проектирования. Это связано с тем, что она разработана на языке Python и, по всей видимости, текущая версия (на момент написания статьи – это 3.7.8) использует графические возможности компьютера далеко не самым наилучшим образом. Тем не менее, режим выполнения самих моделей близок к режиму реального времени при использовании разумных частот дискретизации (в нашем случае 10 МГц) и простых моделей отраженных сигналов. Использование более сложных моделей сигналов, отраженных, например, от распределенных объектов, потребует перехода от глобальной модели, выполняющейся бесконечно, пока пользователь не прервет ее выполнение, к модели, выполняющейся определенное число раз с выводом данных не в блок визуализации GNU Radio, а на график с помощью библиотеки matplotlib. В любом случае, система GNU Radio позволит выполнить только отладку алгоритмов: проверить их работу и/или выявить функциональные ошибки. Профилирование и оптимизация подпрограмм обработки должны выполняться с помощью средств разработки для целевой платформы, на базе которого проектируется устройство.

Литература

1. Максфилд К. Проектирование на ПЛИС. Курс молодого бойца. – М.: Издательский дом «Додэка-XXI», 2007. – 408 с.

2. Липатников В.С. Моделирование устройств цифровой обработки сигналов с помощью программного инструмента GNU Radio // 17-я Международная конференция «Цифровая обработка сигналов и ее применение – DSPA-2015». – Т. 2. – М., 2015. – С. 641–645.

3. Соловьев А.Г. Тракт цифровой обработки сигналов когерентной импульсно-доплеровской РЛС // Цифровая обработка сигналов. – 2000. – №2. – С. 2-5.

4. Справочник по радиолокации / Под ред. М.И. Сколника. – М.: Техносфера, 2015. – Т. 2. – 680 с.

5. Справочник по радиолокации / Под ред. М.И. Сколника. – М.: Техносфера, 2015. – Т. 1. – 672 с.

6. Синицын Е.А., Чиненов Д.А. Особенности обработки сложных фазоманипулированных радиолокационных сигналов методом обратных пульсаций // Успехи современной радиоэлектроники. – 2015. – №3. – С. 119–121.