ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

РАЗРАБОТКА И ИССЛЕДОВАНИЕ АЛГОРИТМОВ ОБНАРУЖЕНИЯ R – ЗУБЦА ЭЛЕКТРОКАРДИОСИГНАЛА

Выпускная квалификационная работа

обучающегося по направлению подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

очной формы обучения, группы 07001302

Лэ Тхань Хоан

Научный руководитель

к.т.н., доцент

Муромцев В.В.

БЕЛГОРОД 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Развитие компьютерной индустрии в настоящее время затронуло исследования и разработки фактически во всех отраслях современной науки. Помимо этого, компьютеры являются необходимыми и в повседневной жизни людей. Фирмы, производящие компьютеры, с каждым годом выпускают всё более производительное, надежное и качественное оборудование, что, в свою очередь, подталкивает разработчиков электронных приборов в своих нововведениях использовать микропроцессоры. Очевидно, что такое стремительное развитие затронуло и такую отрасль как медицина, ведь без использования современного программного и аппаратного обеспечения высоконаучные исследования невозможны. На сегодняшний день практически на всех этапах медицинского обследования используются компьютеры, как в профилактике заболеваний, так и в их диагностике и терапии.

Ведущей задачей современной медицины является предупреждение заболеваний на ранних стадиях их развития. Заболевания сердца – это одна из самых важных проблем медицины в наши дни. Поэтому для диагностики различных проблем разрабатывается всевозможная аппаратура, которая по своей сути является информационно-измерительной системой.

Наиболее распространенным и быстрым методом проверки сердечно-сосудистой системы человека является электрокардиограмма, или ЭКГ. Она позволяет получить точные сведения о состоянии сердца и его работе во время проведения процедуры. Данный метод проверки может назначаться многократно, так как является совершенно безболезненным и неопасным. Возможность назначения многократных проверок особенно ценна в тех случаях, когда врачу необходимо следить за ходом лечения пациента.

Электрокардиограмма позволяет:

- Определить частоту (пульс) и регулярность сердечных сокращений;
- Увидеть острое или хроническое повреждение миокарда (инфаркт миокарда, ишемия миокарда);
- Выявить нарушения обмена калия, кальция, магнияи других электролитов;
- Выявить нарушения внутрисердечной проводимости (различные блокады) [3].

Кроме того этот метод применяется при ишемической болезни сердца, в том числе и при нагрузочных пробах, дает понятие о физическом состоянии сердца и внесердечных заболеваниях (например тромбоэмболия легочной артерии), позволяет удаленно диагностировать острую сердечную патологию с помощью кардиофона, а так же обязательно применяется при прохождении диспансеризации [3].

Любая ЭКГ состоит из зубцов, сегментов и интервалов. Зубцы ЭКГ обозначают латинскими буквами. Если амплитуда зубца составляет больше 5 мм - такой зубец обозначается заглавной буквой; если меньше 5 мм - строчной. Как видно из рисунка нормальная кардиограмма состоит из следующих участков:

- Зубец P - предсердный комплекс;
- Интервал PQ - время прохождения возбуждения по предсердиям до миокарда желудочков;
- Комплекс QRS - желудочковый комплекс;
- Зубец Q - возбуждение левой половины межжелудочковой перегородки;
- Зубец R - основной зубец ЭКГ, обусловлен возбуждением желудочков;
- Зубец T - регистрируется во время реполяризации желудочков;

- Зубец S - конечное возбуждение основания левого желудочка (непостоянный зубец ЭКГ);
- Сегмент ST - соответствует периоду сердечного цикла, когда оба желудочка охвачены возбуждением;
- Интервал QT - электрическая систола желудочков;
- Зубец U - клиническое происхождение этого зубца точно неизвестно (регистрируется не всегда);
- Сегмент TP - диастола желудочков и предсердий.

Сегодня без использования компьютера исследования сердца обойтись не могут. Компьютерная электрокардиограмма является незаменимым методом контроля динамики заболеваний. Электрические потенциалы могут быть зафиксированы как на бумаге, так и отображаться на экране электрокардиографа. При использовании данного метода диагностики полученные данные становятся известны кардиологу сразу после завершения процедуры.

Наиболее важными преимуществами автоматической обработки данных ЭКГ являются:

- Осуществление обработки по одной схеме;
- Стандартный вид предоставления результатов;
- Использование стандартной терминологии.

Само собой при ручной обработке данных не могут быть получены некоторые параметры, что делает очевидным преимущества компьютеров, не вызывающие сомнений.

Данная выпускная квалификационная работа является актуальной, так как не смотря на то, что сегодня существует невероятное количество программных продуктов, позволяющих работать с данными электрокардиографов, затрачивается большое количество времени на обработку сигналов с их помощью. Помимо этого почти все представленное

программное обеспечение закрыто и недоступно. Исходя из вышесказанного, нужно разработать новое простое и удобное в использовании программное средство для работы с электрокардиограммами, которое будет доступно.

В первой главе произведен обзор существующих алгоритмов выделения R- зубцов, а так же обозначена цель работы, и поставлены задачи, которые необходимо выполнить.

Во второй главе продемонстрирована программная реализация алгоритмов выделения R-зубцов, а также произведено обоснование выбора инструментальных и программных средств и библиотеки обработки кардиосигнала.

В третьей главе произведено тестирование программного обеспечения и анализ разработанного алгоритма.

Выпускная квалификационная работа содержит 48 страниц и 30 рисунков.

ГЛАВА 1. ОБЗОР СУЩЕСТВУЮЩИХ АЛГОРИТМОВ ВЫДЕЛЕНИЯ R-ЗУБЦОВ

При любом анализе и диагностике сердцебиений предметом исследования являются наиболее важные параметры электрокардиограммы – амплитуда и форма QRS-комплекса. Очень широко применяются надежные алгоритмы распознавания комплексов QRS в связи с тем, что компьютерная интерпретация электрокардиограммы в 12 отведениях становится всё популярнее. При лечении коронарной болезни сердца широко используют ЭКГ-мониторы. С помощью устройства холтеровского сканирования, которое включает в себя детектор QRS-комплексов, можно значительно быстрее анализировать электрокардиограммы в режиме реального времени. Сейчас также разрабатываются мониторы для амбулаторных больных , в реальном времени анализирующие ЭКГ. Так, например, такое устройство возможно запрограммировать на мгновенную запись интервала с ненормальной электрокардиограммой при обнаружении аритмии, который впоследствии будет передан на станцию врачу для дальнейшей интерпретации. Поэтому таким мониторам требуется довольно точная локализация пиков. Таким образом, важной составляющей большинства инструментов анализа и обработки электрокардиограмм является точный детектор комплексов QRS.

Способы нахождения QRS-комплексов и R-зубцов в сигналах электрокардиограмм являлись предметом изучения в течение нескольких десятилетий. Множество наиболее ранних алгоритмов основано на полученных сигналах из производных сигнала ЭКГ. Такие методы можно охарактеризовать низкой вычислительной сложностью и не очень хорошими результатами нахождения проблемных сигналов (например, содержащих дрейф изолинии, шум и искажение сигнала, а также изменения в морфологии

QRS) ввиду того, что в них не используются никакие дополнительные правила, которые могли бы позволить сократить нахождение ложных участков.

Современные, использующиеся как пиковые датчики для электрокардиограммы, цифровые фильтры обеспечивают наиболее хорошие результаты нахождения для сигналов с шумом. Для обнаружения R-зубца и QRS применяют большое количество стандартных методов обработки сигнала, например методы линейного и нелинейного фильтрования, вейвлет-преобразования, искусственные нейронные сети, генетические алгоритмы. Эти алгоритмы намного более сложные по сравнению с методами, основанными на производной, и показывают лучшие результаты обнаружения [8].

Обычно на ЭКГ выделяют пять зубцов: P, Q, R, S, T (рисунок 1.1). Также крайне редко можно увидеть и малозаметную волну U.

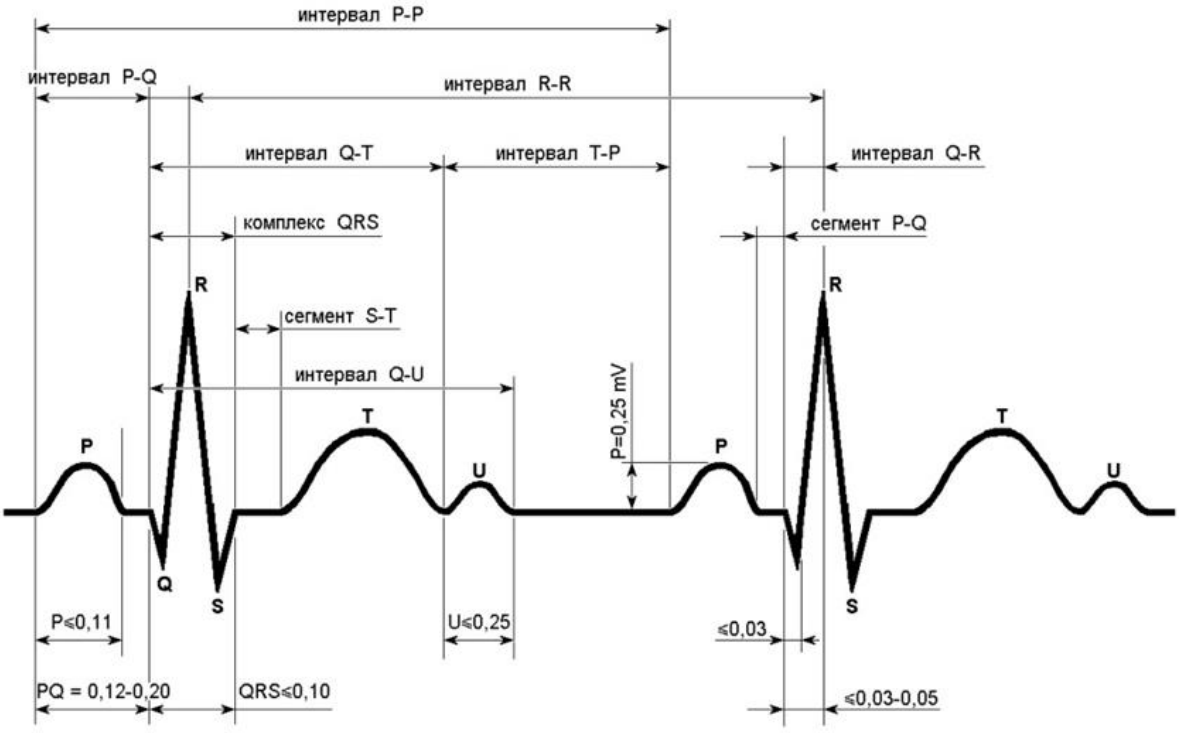


Рис. 1.1. Типичный вид электрокардиограммы

Соответствующие зубцы формируются за счет распространения возбуждения в сердце и отражают этот процесс (см. таблицу 1.1).

Длительность одного сердечного цикла характеризуют временные интервалы между последовательными зубцами P и R.

На рисунке 1.1 показана электрокардиограмма для здорового человека. При всевозможных нарушениях нормального функционирования сердца вид кардиограммы изменяется.

Таблица 1.1

Физиологические процессы, соответствующие различным участкам одного цикла электрокардиограммы

Зубец, сегмент, комплекс	**Физиологический процесс**
P	Охват возбуждением предсердий. Первая половина зубца соответствует распространению возбуждения из синусного узла на правое предсердие, вторая половина – охвату возбуждением левого предсердия.
PQ	Возбуждение распространяется на предсердножелудочковый узел и движется по проводящей системе желудочков. Этот сегмент возникает в результате того, что возбуждение оснований желудочков нарастает медленнее.
QRS	Охват возбуждением желудочков.
Q	Возбуждение верхушки сердца и внутренней поверхности желудочков.
S	Полный охват возбуждением миокарда желудочков.
T	Реполяризация (восстановление нормального мембранного потенциала клеток миокарда).

Основным преимуществом электрокардиограммы является то, что она совершенно безвредна, а так же очень информативна как метод исследования состояния функций сердца.

В данной выпускной квалификационной работе рассмотрены классические методы, использующие аппроксимацию производных сигнала, широко распространенный алгоритм Пана-Томпкинса, новый метод, предложенный Кёлером, подобный методу Такора, и корреляционный алгоритм, который похож алгоритм Чена.

Ниже приведены главные описания четырёх сравниваемых методов с алгоритмами выделения R- зубца и их программной реализации.

1.1 Алгоритмы, основанные на производной

QRS-комплекс характеризуется наибольшей скоростью изменения напряжения в сердечном цикле, то есть наибольшей крутизной наклона сигнала. Операция d/dt – наиболее логичная начальная точка в попытке разработать алгоритм для обнаружения комплекса QRS, потому что скорость изменения задаётся оператором производной, который усиливает комплекс QRS. Однако, сходства с типичным комплексом QRS результирующая волна не имеет никакого. Наиболее высокие значения выходного сигнала наблюдаются на участке, который соответствует комплексу QRS, а относительно медленные зубцы P и T подавляются оператором производной. Исходя из шумового характера сигнала, который получается на выходе операторов, которые основаны на производных, очевидно, что перед тем, как будет выполняться следующая обработка, понадобится существенное сглаживание.

Алгоритм выделения комплекса QRS, основанный на производной, впервые был предложен Бальдом, а позже был исследован и оценен Томпкинсом и Альстромом. Рассмотрим, как работает алгоритм.

Аппроксимируется сглаженная трёхточечная первая производная $y_0(n)$ от входного данного сигнала x(n):

$$y_0(n) = | x(n) - x(n-2) |$$

Следующую формулу использовали чтобы считать второй производной:

$$y_1(n) = | x(n) – 2x(n-2) + x(n-4) |$$

Эти два результата взвешиваются и комбинируются, что даёт:

$$y_2(n) = 1{,}3y_0(n) + 1{,}1y_1(n)$$

Результирующий сигнал $y_2(n)$ сканируется с использованием порога. Если этот порог пересекается с сигналом, следующие отсчеты всегда тоже проверяются с применением этого же порога. При прохождении теста, как минимум, шестью из восьми точек с использованием этого порога, сегмент, который состоит из 8 отсчетов, считают частью комплекса QRS. Импульс, получившийся в результате этой процедуры, имеет ширину, которая пропорциональна ширине комплекса QRS, но этот метод очень чувствителен к шуму.

Алгоритм для нахождения комплекса QRS, который основан на взвешенном и возведенном в квадрат операторе первой производной, а также на фильтре скользящего среднего, предложили Мерфи и Рангараж. Теперь оператор, который основан на производной, определяется так:

$$g_1(n) = \sum_{i=1}^{N} (|x(n-i+1) - x(n-i)^2|(N-i+1))$$

где x(n) – сигнал ЭКГ, N – ширина окна, в пределах которого разность первого порядка вычисляется, возводится в квадрат и взвешивается с использованием коэффициента (N-i+1).

1.2 Алгоритм Пана и Томпкинса

Алгоритм для обнаружения комплексов QRS, который рассчитан на работу в режиме реального времени, предложили Пан и Томпкинс. Этот

алгоритм основан на анализе наклона, амплитуды и ширины QRS-комплексов и состоит из последовательности фильтров и методов: фильтр нижних частот, фильтр верхних частот, оператор производной, возведение в квадрат, интегрирование, адаптивная пороговая процедура и процедура поиска [6].

Рекурсивный ФНЧ, который использован в алгоритме, имеет целые коэффициенты для снижения его вычислительной сложности. Выходной сигнал y(n) связан с входным сигналом x(n) уравнением:

$$\mathrm{y(n)} = 2\mathrm{y(n-1)} - y(n-2) + \frac{1}{32}[x(n) - 2x(n-6) + x(n-12)].$$

Этот фильтр вносит задержку в 5 отсчётов или 25 мс при частоте дискретизации 200 Гц. ФВЧ, который используется в этом алгоритме, реализован как всепропускающий фильтр минус ФНЧ [6]. Выходной сигнал p(n) фильтра верхних частот задаётся разностным уравнением:

$$\mathrm{p(n)} = \mathrm{p(n-1)} - \frac{1}{32}x(n) + x(n-16) - x(n-17) + \frac{1}{32}x(n-32)$$

ФВЧ вносит задержку 80 мс. Использованная Паном-Томпкинсом операция дифференцирования задаётся так:

$$\mathrm{y(n)} = \frac{1}{8}[2\mathrm{x(n)} + \mathrm{x(n-1)} - \mathrm{x(n-3)} - 2\mathrm{x(n-4)}]$$

Эта процедура взятия производной подавляет низкочастотные компоненты зубцов P и T и имеет высокий коэффициент усиления для высокочастотных компонентов, появляющихся из-за крутых склонов QRS-комплекса [6].

Выходной сигнал операции, которая основана на производной, демонстрирует многочисленные пики в пределах длительности отдельного комплекса QRS. Алгоритм Пана-Томпкинса выполняет сглаживание

выходного сигнала предыдущих операций, используя интегрирующий фильтр типа скользящего окна, который задается следующим уравнением:

$$y(n) = \frac{1}{N}[x(n-(N-1)) + x(n-(N-2)) + \cdots + x(n)]$$

1.3 Корреляционный алгоритм

На первом этапе работы алгоритма находятся значения функции взаимной корреляции R(t), функции сравнения B(t+τ) и отрезков исходной функции A(t) той же длительности, что и функция сравнения:

$$R(t) = \frac{1}{T}\sum_{t=0}^{T-1} B(t+\tau) * A(\tau)$$

Появляется возможность построить взаимно корреляционную функцию на всем протяжении кардиосигнала, но результаты вычисления R(t) зависят от значений исходных данных и не могут быть нормально интерпретированы. Поэтому на втором этапе функция взаимной корреляции масштабируется для приведения к диапазону значений [-1,1]:

$$p(t) = \frac{R(t)}{\frac{1}{N}\sqrt{\sum_{\tau=0}^{T-1} A^2(\tau) \cdot \sum_{\tau=0}^{T-1} B^2(t+\tau)}}$$

Полученную функцию уже можно использовать для принятия решения о том, найдена точка, на которую настроен алгоритм, или нет. Решение принимается по превышению коэффициентом корреляции (КК) определенного порога A1. Этот выбор определяет чувствительность алгоритма.

Так как данный алгоритм при принятии решения о положении искомой точки синхронизации не опирается на абсолютные значения исходного

сигнала, он имеет более высокую устойчивость к помехам и изменчивости кардиосигнала по сравнению с амплитудным пороговым детектором и может легко находить R-зубцы даже при его весьма сильном искажении шумами [9].

Недостатком данного алгоритма является большой объем вычислений, необходимых для его реализации, что приводит к увеличению времени анализа ЭКГ большой длительности. Кроме того, на конечный результат влияет выбор той или иной функции сравнения, поскольку она определяет конечный вид функции КК.

В [9] в качестве функции сравнения используется отрезок исходного сигнала, который содержит один комплекс QRS. Такой подход даёт наиболее близкие к 1 значения КК в районе комплекса QRS. Но у этого подхода также есть и минусы. Всегда перед началом анализа новой электрокардиограммы нужно выделять область сигнала, принимаемую за эталонную. Поэтому результаты одного вычисления КК могут не совпадать с результатами другого, так как функции сравнения были разные. Кроме того, в этом случае полученная функция КК будет сдвигаться относительно исходного сигнала в зависимости от характеристик функции сравнения, что требует введения компенсационного сдвига.

1.4 Метод, основанный на подсчете числа пересечений нуля

В статье [11] предложен новый метод для обнаружения комплексов QRS в электрокардиографических сигналах, который основан на функции, полученной при подсчете числа пересечений нуля на ЭКГ-сегмент. Он обеспечивает высокую степень обнаружения даже в тех случаях, когда кардиосигналы очень зашумлены. Кроме того он обеспечивает в вычислительном отношении эффективное решение проблемы обнаружения

QRS-комплексов из-за простоты обнаружения и подсчета нулевых пересечений.

Из-за спектральных характеристик компонентов электрокардиограммы нужно отфильтровать кардиосинал для того, чтобы уменьшить среднее значение P-и T-волн, и шум высоких частот. Так как фильтруемый сигнал будет использоваться для временной локализации R-волны, используют полосовой фильтр с линейной фазовой характеристикой. Иначе точная локализация R-волны была бы невозможна [11].

Отфильтрованный сигнал колеблется вокруг нуля, при этом в области комплекса QRS он имеет высокую амплитуду, а в остальных интервалах его амплитуда является низкой. Добавление высокочастотной последовательности к отфильтрованному сигналу позволяет получить сигнал, у которого много нулевых пересечений вне QRS-сегментов и только небольшое количество нулевых пересечений в области комплекса QRS. Последовательность высокой частоты вычисляется так:

$$b(n) = (-1)^2 K(n)$$

где *K(n)* – амплитуда, которая изменяется со временем. Улучшение сигнала получается путем нелинейного преобразования сигнала.

$$y(n) = sign\left(x_f(n)\right) {x_f}^2(n),$$

где $x_f(n)$ – отфильтрованный сигнал и *y(n)* – нелинейно преобразованный сигнал. Сигнал *y(n)* используется для определения временного расположения R-волны.

Из-за приложения полосового фильтра к сигналу *y(n)* высокочастотные колебания ослаблены. Следовательно, необходимо добавить высокочастотную последовательность к сигналу:

$$z(n) = y(n) + b(n)$$

чтобы увеличить число нулевых пересечений вне QRS-сегментов.

В реализации [11] K(n) определяется так:

$$K(n) = \lambda_K K(n-1) + (1-\lambda_K)|y(n)|c,$$

где $\lambda_K \in (0;1)$ – фактор упущения, и параметр *c* обозначает постоянное усиление, например, *c* = 4. Обнаружение и подсчет пересечений нуля:

$$d(n) = \left[\frac{sign\,[\,z\,(n)] - sign\,[\,z\,(n-1)]}{2}\right].$$

Число нулевых пересечений на сегмент:

$$D(n) = \sum_{i=0}^{N-1} d\,(n-\,i)\ .$$

Обнаружение событий выполняется с использованием адаптивного порога Θ:

$$\Theta(n) \ = \ \lambda_\Theta \Theta(n-1) \ + \ (1 \ - \lambda_\Theta) D(n),$$

где $\lambda\,\Theta \in (0;1)$, представляет фактор упущения. Для обнаружения события порог $\Theta(n)$ сравнивают с сигналом *D(n)*: как только *D(n)* меньше - событие обнаружено.

Временная локализация события обеспечивает границы для поискового интервала, используемого для временной локализации R-волны. В сигнале *y(n)* выполняется объединенный поиск максимума/минимума: если величина минимума намного больше, чем величина максимума, временное расположение минимума берут в качестве времени R-волны, в других случаях расположение R-волны определяет максимальная позиция. Для установления фактической позиции R-волны должна быть учтена групповая задержка полосового фильтра [11].

1.5 Постановка задачи

Целью работы является анализ существующих алгоритмов обнаружения R-зубца электрокардиосигнала, а также разработка собственного алгоритма.

Для достижения поставленной цели необходимо решить следующие задачи:

1) Произвести обзор существующих алгоритмов выделения R-зубцов;

2) Обосновать выбор инструментальных и программных средств;

3) Обосновать выбор исходных данных;

4) Разработать функциональную схему и панель инструментов;

5) Выполнить программную реализацию алгоритмов выделения R-зубцов.

ГЛАВА 2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМОВ ВЫДЕЛЕНИЯ R-ЗУБЦОВ

2.1 Обоснование выбора инструментальных и программных средств

В кардиологии всё чаще используется анализ электрокардиограмм с помощью компьютера. Компьютерный анализ ЭКГ можно разделить на три этапа:

1) Получение данных ЭКГ;

2) Распознавание участков ЭКГ, которые имеют диагностическое значение;

3) Ввтоматизированный анализ результатов, полученных на втором этапе и постановка диагноза.

На первом этапе осуществляется сбор информации о работе сердца с последующим ее внесением в компьютер, то есть рабочее место врача. Для реализации данного этапа можно использовать большое количество средств: от электрокардиографа, связанного с компьютером врача, до сложных сетевых систем, позволяющих собирать информацию с удаленных электрокардиографов. Обычно такие системы имеют модульную структуру и настроены для необходимого уровня сложности.

Для реализации второго этапа анализа электрокардиограмм с помощью компьютера существует большое количество алгоритмов распознавания образов. Однако остается нужным создать эффективный алгоритм, который позволил бы определять особые точки сигналов электрокардиограмм для выявления отклонений от нормы.

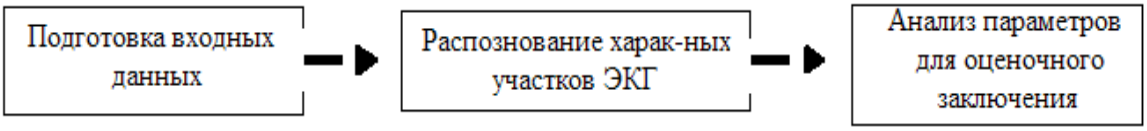


Рис. 2.1. Алгоритм автоматического анализа ЭКГ

На рисунке 2.1 представлен традиционный алгоритм автоматического анализа ЭКГ.

Третий этап компьютерного анализа ЭКГ может быть решен только в автоматизированном режиме. Компьютер здесь используется для внесения и хранения информации, а также для автоматизации ее поиска, так как окончательно поставить диагноз может только врач кардиолог.

Для разработки программного обеспечения для выделения R-зубцов была выбрана графическая среда программирования LabVIEW.

LabVIEW (англ. Laboratory Virtual Instrumentation Engineering Workbenh) – это среда разработки и платформа для выполнения программ, созданных на графическом языке программирования «G» фирмы National Instruments (США). Первая версия LabVIEW была выпущена в 1986 году для Apple Macintosh, в настоящее время существуют версии для UNIX, Linux, Mac OS и пр., а наиболее развитыми и популярными являются версии для Microsoft Windows [4].

Компания, создавшая LabVIEW (National Instruments), была основана Джеймсом Тручардом (James Truchard), Джеффом Кодоски (Jeff Kodosky) и Биллом Новлиным (Bill Nowlin) в 1976 году в Остине, штат Техас. Компания специализировалась на автоматизации производства и инструментальных средствах для измерений.

Первая версия LabVIEW была выпущена в 1986 году для Apple Macintosh. Инженеры NI создали графическую среду для разработки, тем самым бросив вызов традиционным языкам программирования.

Первой же кроссплатформенной версией LabVIEW стала третья версия, которая была выпущена в 1993 году.

Головной офис компании до сих пор находится в Остине, а остальные офисы находятся почти в 40 странах, в том числе и в России.

Наиболее полно возможности LabVIEW раскрываются при создании приборов и систем для измерений физических величин в научных экспериментах, лабораторных и промышленных установках. Наиболее важным достоинством LabVIEW можно назвать возможность управления процессом измерения в автоматическом или интерактивном режиме. Взаимодействие с исследователем или оператором осуществляется с помощью продуманного и простого в программировании графического интерфейса. Для обработки и анализа данных используется огромный набор как функциональных библиотек общего назначения, так и специализированных библиотек [10].

В LabVIEW программы составляются в виде графических диаграмм, которые похожи на обычные блок-схемы. Этим LabVIEW отличается от текстовых языков программирования, таких как C, Pascal, Java и других, где программы составляются в виде строк текста.

Пользователю и разработчику доступны функционально одинаковые системы программирования для различных ОС, например для Linux, Microsoft Windows и MacOS, что несомненно является достоинством LabVIEW. Так, программа, разработанная для MacOS, почти без изменений будет работать на компьютере с Windows.

Программирование в LabVIEW очень близко к понятию алгоритм. После продумывания алгоритма программы остается только нарисовать блок-схему алгоритма с помощью графического языка программирования «G». Это дает возможность не задумываться о различных атрибутах системного программирования.

В процессе создания программ в LabVIEW программист формирует пользовательский интерфейс программы. Для этого нужно мышкой выбирать нужные элементы (графики, кнопки, регуляторы) из наглядных палитр-меню и перемещать их в рабочее поле программы. Аналогично создается и

алгоритм. Если в процессе программирования будет допущена ошибка, то почти всегда LabVIEW сразу обратит на это внимание программиста. Программа готова к работе сразу после того, как алгоритм будет нарисован.

Также в системе программирования LabVIEW содержится встроенный механизм отладки приложений. В процессе отладки можно выполнять программу «по шагам», назначать точки останова программы, визуализировать процесс исполнения программы и контролировать любые данные в любом месте программы [10].

Кроме того разработчики могут устанавливать пароли для доступа к их приложениям или полностью удалить исходный код из работающего приложения, что позволяет защитить программы от просмотра исходного кода или несанкционированного изменения.

Исходя из вышесказанного, достоинствами LabVIEW являются:

1) Интуитивно понятный процесс графического программирования;

2) Полноценный язык программирования;

3) Совместимость с такими операционными системами, как Windows2000/NT/XP, Mac OS X, Linux и др.;

4) Огромные возможности сбора, обработки и анализа данных, управления приборами, генерации отчетов и обмена данных через сетевые интерфейсы;

5) Большое количество примеров и шаблонов приложений;

6) Драйверная поддержка более 2000 приборов;

7) Высокая скорость выполнения откомпилированных программ;

8) Возможности интерактивной генерации кода.

Помимо всего прочего, LabVIEW не только поддерживает большое количество оборудования разных производителей, но и имеет в своём составе (либо позволяет добавлять к базовому пакету) многочисленные библиотеки компонентов. Отсюда следует вывод, что LabVIEW является очень эффективной средой программирования на графическом языке, которая

предназначена для реализации масштабируемых и функционально гибких приложений.

2.1 Обоснование выбора исходных данных

Для тестирования алгоритмов был использован банк данных электрокардиограмм PhysioBank, потому что он довольно популярен и находится в открытом доступе.

В левом верхнем углу портала PhysioNet находится карта сайта, сделанная как выпадающее меню. Это сделано для удобства перемещения по сайту с возможностью быстро вернуться на главную страницу сайта. На карте сайта видно, что портал состоит из шести разделов с различным количеством подразделов в каждом.

Раздел «What's New?» («Что нового?») включает в себя 5 следующих подразделов:

1) News from PhysioNet (новости PhysioNet);

2) PhysioNet in news (PhysioNet в новостях);

3) Events (мероприятия);

4) Sponsorship opportunities (возможности для спонсорства);

5) Training opportunities (возможности обучения).

На рис. 2.2 представлен раздел «Что нового?» («What's New?»).

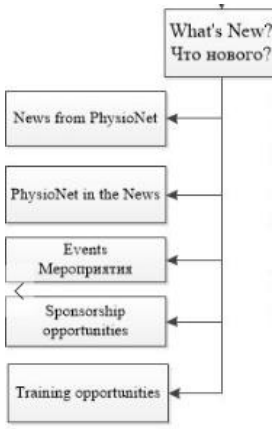


Рис. 2.2 Струкрурная схема раздела сайта PhysioNet «Что нового?» («What's New?»)

Раздел «PhysioNetLibrary» («Библиотека PhysioNet») состоит из 7 подразделов (рис. 2.3):

1) Getting started (приступая к работе);

2) Callenges (сложные задачи);

3) Contibutors (авторы);

4) Publications (публикации);

5) Tutorials (учебники);

6) About PhysioNet (о PhysioNet);

7) External resources (внешние ресурсы).

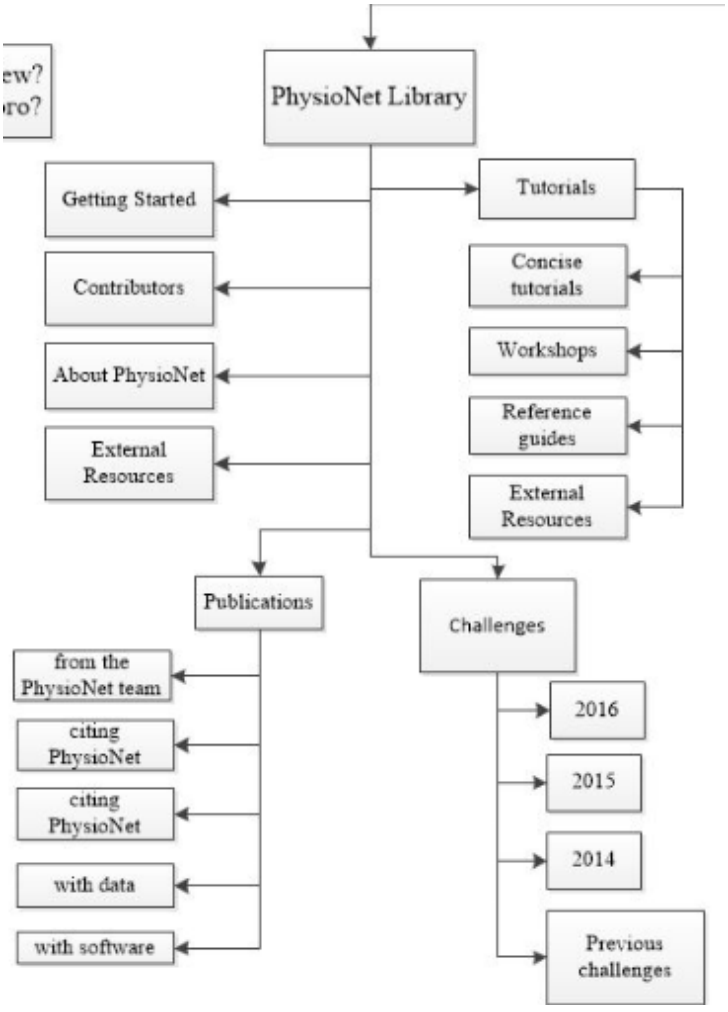


Рис. 2.3 Структурная схема раздела сайта Physionet «Библиотека Physionet» («PhysioNetLibrary»)

В разделе «PhysioBank» находится 8 подразделов (рис. 2.4):

1) Getting started (приступая к работе);

2) Signal archives (архивы сигналов). Здесь названы все имеющиеся в настоящее время базы данных в архивах PhysioBank;

3) LightWAVE (средство для просмотра аннотаций и редактор);

4) PhysioBank ATM (объект для самостоятельного изучения PhysioBank);

5) PhysioBank search (поиск записи PhysioBank);

6) Call for Data;

7) About PhysioBank (о PhysioBank);

8) External resources (внешние ресурсы).

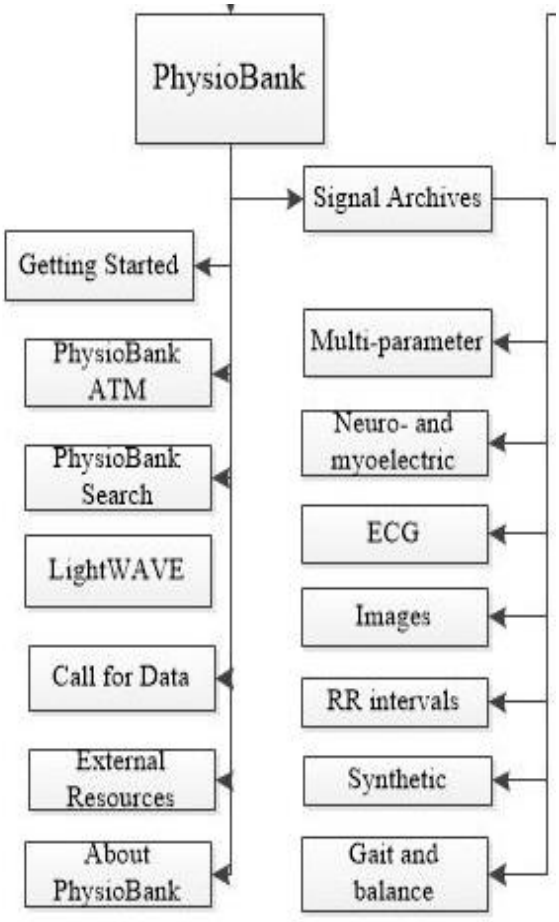


Рис. 2.4 Структурная схема раздела сайта Physionet«PhysioBank»

Раздел под названием «PhysioToolkit», состоит из 6 подразделов (рис. 2.5):

1) Getting started (приступая к работе);

2) PhysioToolkit Software Index;

3) Matlab software (программное обеспечение для Matlab и Octave);

4) WFDB software (пакет программного обеспечения WFDB);

5) Manuals (руководства);

6) External resources.

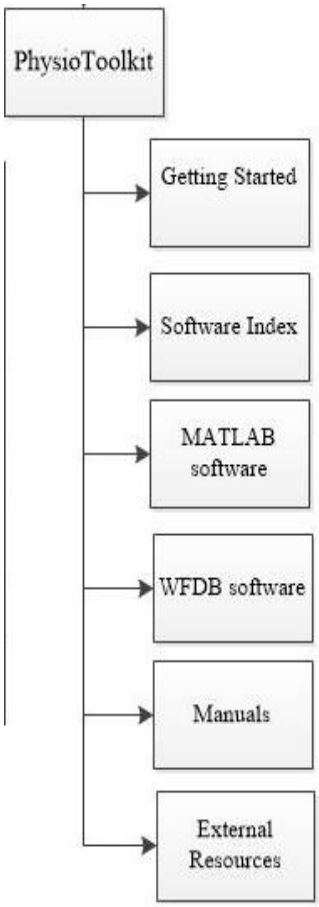


Рис. 2.5 Структурная схема разделов сайта Physionet «PhysioToolkit»

Виртуальная лаборатория PhysioNetWorks находится в одноименном разделе «PhysioNetWorks», который включает в себя 6 подразделов (рис. 2.6):

1) User home page (домашняя страница пользователя);

2) Login/create an account (войти/создать учетную запись);

3) Works in progress (работы над улучшением);

4) PhysioNetWorks project guide (руководство по проекту PhysioNetWorks);

5) Data sharing (обмен данными);

6) Introduction (введение в PhysioNetWorks).

В разделе «Help» («Помощь») (рис. 2.6) можно найти ответы на часто задаваемые вопросы, информацию о содействии, конфиденциальности и копировании, карту портала PhysioNet и ссылки на учебники, полезные для пользователей.

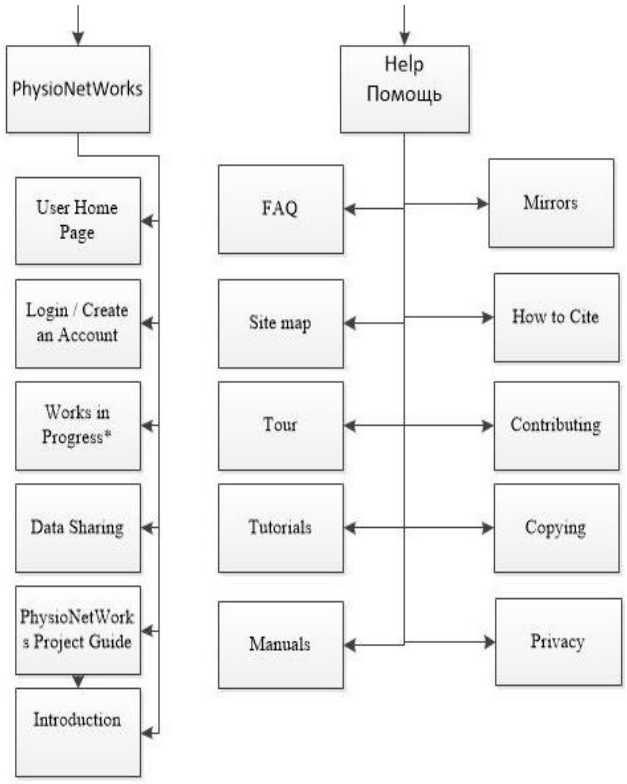


Рис. 2.6 Структурная схема разделов сайта Physionet «PhysioNetWorks» и «Помощь»

Полная структурная схема портала Physionet представлена в приложении 1.

2.3 Разработка функциональной схемы и панели инструментов

При запуске LabVIEW (см. рис. 2.7) на экране появляется начальное окно, которое используется, если нужно создать виртуальный инструмент, открыть файл LabVIEW, который был ранее уже создан, найти примеры или обратиться к справке. Кроме того, здесь можно увидеть дополнительные ресурсы и информацию.

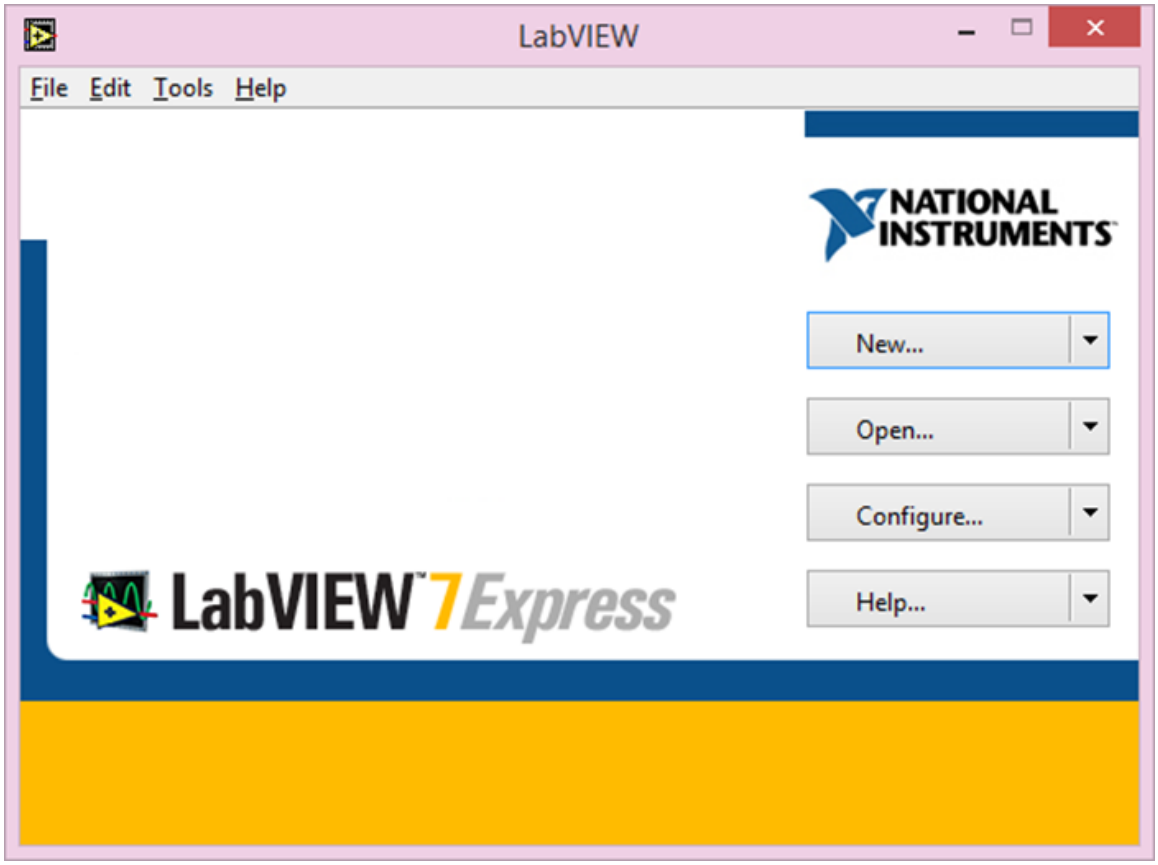


Рис. 2.7 Стартовое диалоговое окно.

После выбора из меню стартового диалогового окна строки меню New - Blank VI создаются программные модули, называющиеся Виртуальные инструменты («Virtual Instruments»). Виртуальные приборы состоят из двух частей:

- Передняя панель (FrontPanel) – интерфейс программы;
- Блок-диаграмма (BlockDiagram). Содержит визуально-графическое представление программного кода.

Окна виртуальных инструментов содержат одинаковые панели меню: File, Edit, Operate, Tools, Browse, Windows и Help, которые встречаются чаще всего. Сразу под меню находится полоса панели инструментов, которая дает возможность запускать и редактировать программу. Кроме того в окне BlockDiagram есть дополнительные кнопки для отладки. На рис 2.8 показаны окна FrontPanel и Block Diagram.

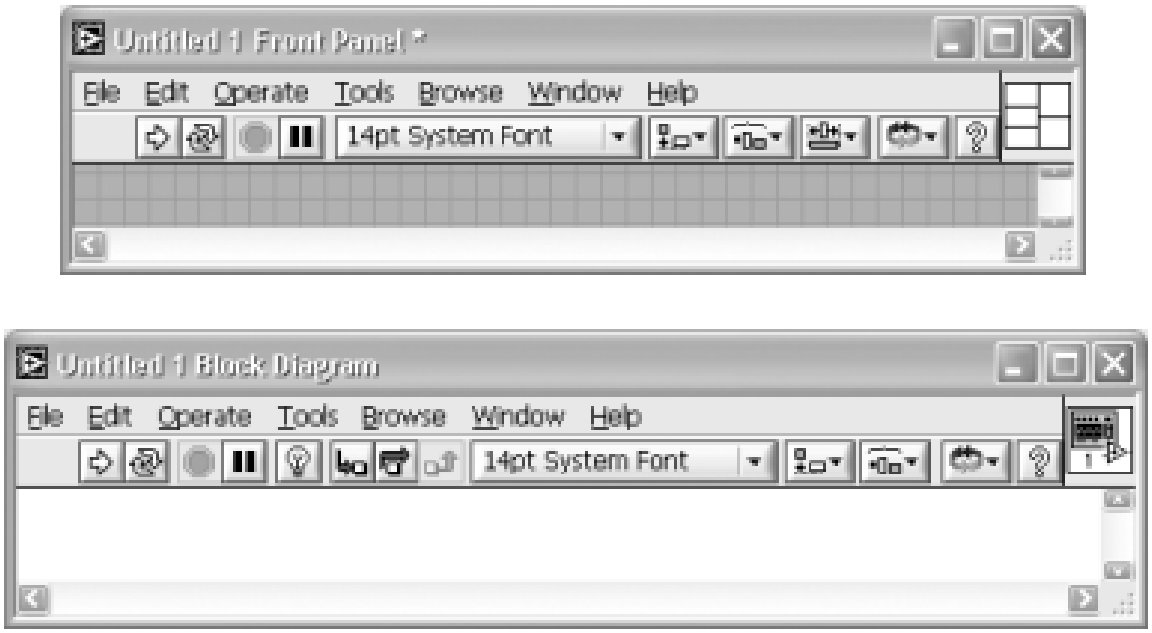


Рис. 2.8 Вид лицевой панели (Front Panel) и блок-диаграммы (Block Diagram)

Палитру элементов можно вывести на лицевую панель при помощи меню Window строки ShowControlsPalette. Для вывода на окно Block Diagram палитры функций нужно воспользоваться строкой ShowFunctionsPalette этого же меню.

Для переноса любого объекта из палитры необходимо перемещать мышку по разделам палитр. Выбранный объект берется из палитры благодаря щелчку левой кнопкой мышки и перемещается в заданную область соответствующей панели.

На рис. 2.9 изображена инструментальная панель, на которой размещаются управляющие кнопки:

- Кнопка для начала выполнения программы, которая при необходимости также компилирует. Если стрелка непрерывная, то программу можно запустить;
- Кнопка для запуска программы сразу же после завершения;
- Кнопка для прерывания выполнения программы;

• Кнопка для временной остановки программы.

Рис. 2.9 Полоса инструментальной панели на лицевой панели

На рис. 2.10 изображена инструментальная панель Block Diagram, на которой размещаются дополнительные кнопки управления:

• Кнопка для включения на схеме видимости выполнения программы;

• Кнопка для пошагового выполнения программы;

• Кнопка для остановки перед следующим шагом;

• Кнопка для выполнения текущего действия с последующим завершением пошагового выполнения.

Рис. 2.10 Полоса инструментальной панели блок-диаграммы

ToolsPalette, то есть палитра Инструменты, изображенная на рисунке 2.11, предоставляет возможность выполнять необходимые операции по отладке виртуальных приборов, а также их созданию и редактированию. На палитре можно увидеть такие инструменты, как:

• Инструмент для вызова контекстного меню определенного объекта по щелчку;

• Инструмент для ввода текста или изменения значения элементов управления;

• Инструмент для ввода текста или же его редактирования, а также для создания свободных меток;

- Инструмент для перемещения или изменения размера текста;
- Инструмент для соединения объектов на блок-диаграмме;
- Инструмент для быстрой прокрутки окна без обращения к полосам прокрутки;
- Для удаления или размещения контрольных точек.

Рис. 2.11 Палитра Инструменты

Для Отображения ЭКГ используется библиотека ECGAnnotationC++ Library. Для этого в среде LabVIEW были созданы виртуальные приборы. Иерархия приборов представлена на рис. 2.12.

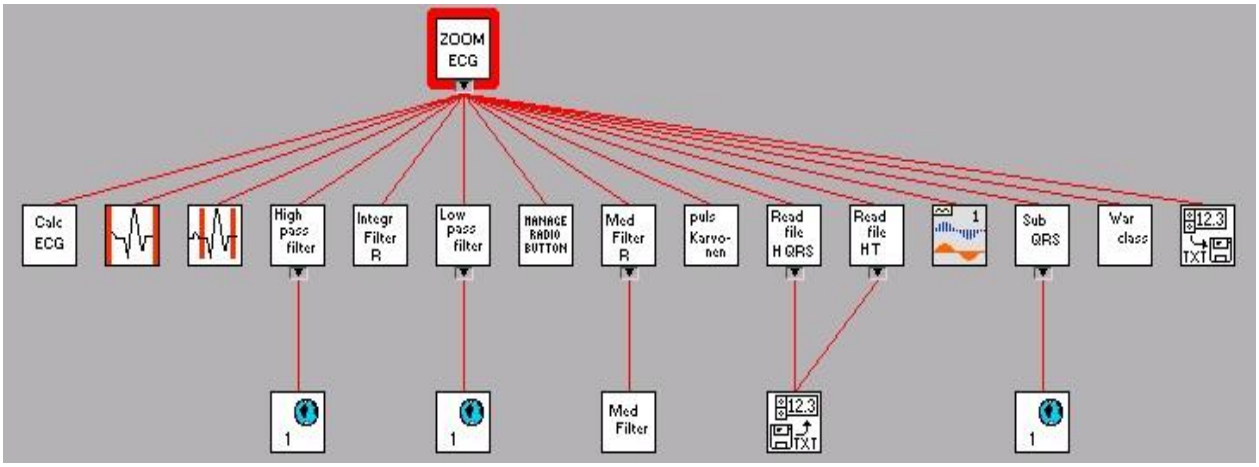


Рис. 2.12 Иерархия виртуальных приборов

Наиболее значимым является первый прибор, то есть прибор CalcECG. Остальные же являются вспомогательными.

Функциональная схема прибора CalcECG изображена на рис. 2.13.

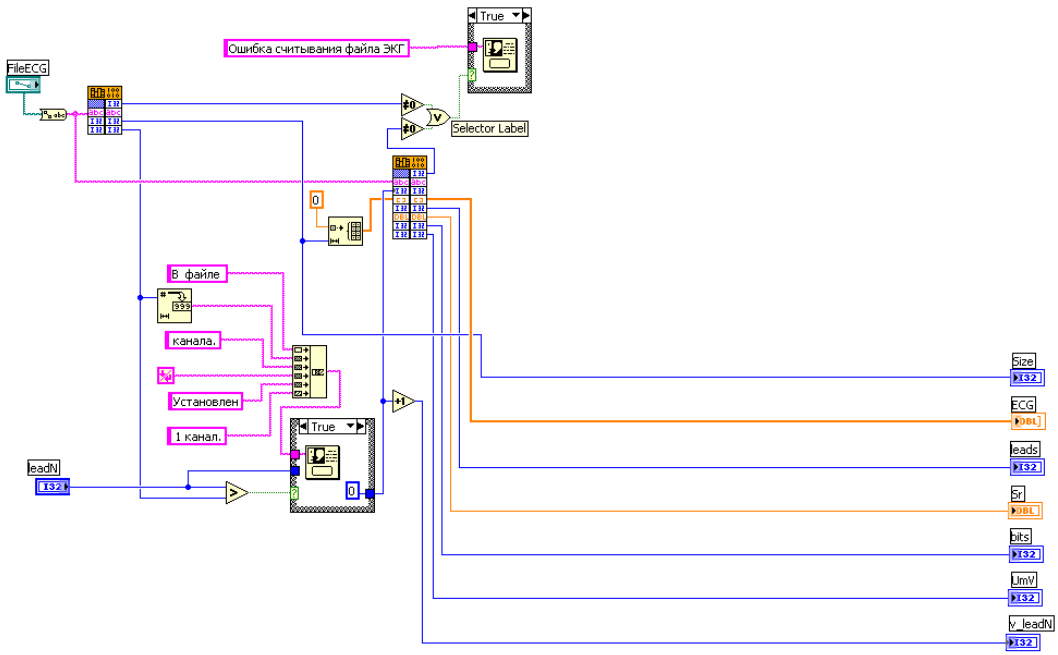


Рис. 2.13 Функциональная схема прибора CalcECG

Виртуальный прибор CalcECG использует внешние функции ReadSignalSize. ReadSignal, AnnECG и RRS_ECG, которые написаны на языке C++, с использованием элементов кода ECGAnnotationC++ Library, и оформлены в виде динамически подключаемой библиотеки.

При загрузке файла электрокардиограммы вызывается функция ReadSignalSize, предназначенная для определения размера сигнала и числа каналов ЭКГ, который записан в заданном файле. Функция имеет вид:

Листинг 2.1 Функция ReadSignalSize

```
int ReadSignalSize(const char* nameFileEcg, int*
SignalSize, int* leads)
{
    wchar_t nameFile[_MAX_PATH] = L"";
    ConvNameFile(nameFileEcg,nameFile);
```

```
    class Signal signal;
    if (signal.ReadFile(nameFile)) {
       *(SignalSize)=signal.GetLength();
       *(leads)=signal.GetLeadsNum();
      return 0;  //Ошибок нет
    } else {
      return -1; //Ошибка считывания файла ЭКГ
    }
}
```

ЭКГ по заданному каналу из заданного файла электрокардиограммы сигнал считывает функция ReadSignal, которая имеет вид:

Листинг 2.2 Функция ReadSignal

```
int ReadSignal (const char* nameFileEcg, int leadNumber,
double* data, int* leads, double* sr, int* bits, int* UmV)
{
    wchar_t nameFile[_MAX_PATH] = L"";
    ConvNameFile(nameFileEcg,nameFile);
//  MessageBox(NULL,nameFile,L"3",MB_OK   |MB_DEFBUTTON1   |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
    class Signal signal;
    if (signal.ReadFile(nameFile)) {
      int SignalSize=signal.GetLength();
      *(leads)= signal.GetLeadsNum();
      *(sr) = signal.GetSR();
      *(bits)=signal.GetBits();
      *(UmV)=signal.GetUmV();
// --------------------------обойтись без pData
      double*  pData = signal.GetData(leadNumber);
      for (int i=0; i<SignalSize; i++) data[i]=pData[i];
      return 0;  //Ошибок нет
    } else {
      return -2; //Ошибка считывания файла ЭКГ
    }
```

Аннотирует электрокардиограмму функция AnnECG, имеющая вид:

Листинг 2.3 Функция AnnECG

```
int AnnECG(const char* nameDirFilters, double* data, int SignalSize, double sr, int* beats, int* ANN_num, int* ANN_smpl, int* ANN_type)
{
wchar_t nameDirF[_MAX_PATH] = L"";
ConvNameFile(nameDirFilters,nameDirF);
//   MessageBox(NULL,nameDirF,L"3",MB_OK   |MB_DEFBUTTON1   | MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
  class EcgAnnotation ann;  //default annotation params
                            //or add your custom ECG params to annotation class from lib.h
                           // ANNHDR hdr;
                           //  hdr.minbpm = 30;
                            //  etc...
                           // class EcgAnnotation ann( &hdr );
  class Signal signal;
  //  tic();
   int**   qrsAnn   =   ann.GetQRS(data,   SignalSize,   sr, nameDirF);         //get QRS complexes
    *(beats)=0;
  if (qrsAnn) {
     *(beats)=ann.GetQrsNumber();
      ann.GetEctopics(qrsAnn,    ann.GetQrsNumber(),    sr); //label Ectopic beats
     int annNum = 0;
     int** ANN = ann.GetPTU(data, SignalSize, sr, nameDirF, qrsAnn, ann.GetQrsNumber());     //find P,T waves
     if (ANN) {
        annNum = ann.GetEcgAnnotationSize();
     } else {
        ANN = qrsAnn;
        annNum = 2 * ann.GetQrsNumber();
```

```
    }
    //printing out annotation
    for (int i = 0; i < annNum; i++) {
       ANN_smpl[i]=ANN[i][0];
    ANN_type[i]=ANN[i][1];
 }
 *(ANN_num)=annNum;
  return 0;
}
```

Для выделения RRS используется функция RRS_ECG:

Листинг 2.4 Функция RRS_ECG

```
int RRS_ECG(double sr, int ANN_num, int* ANN_smpl, int*
ANN_type,int* RRS_num, int* RRS_smpl, double* RRS, double*
mean_heart_rate)
{
  class EcgAnnotation ann;  //default annotation params
//or add your custom ECG params to annotation class from
lib.h
                          // ANNHDR hdr;
                          //  hdr.minbpm = 30;
                           //  etc...
                          // class EcgAnnotation ann( &hdr
);
  class Signal signal;
   int **ANN;
   ANN = new int* [ANN_num];
   for (int i = 0; i < ANN_num; i++) {
      ANN[i] = new int[3];
      ANN[i][0]=ANN_smpl[i];
      ANN[i][1]=ANN_type[i];
   }
   //saving RR seq
   vector<double> rrs;
```

```
   vector<int> rrsPos;
   if (ann.GetRRseq(ANN, ANN_num, sr, &rrs, &rrsPos)) {
     for (int i = 0; i < (int)rrs.size(); i++) {
       RRS_smpl[i]=rrsPos[i];
       RRS[i]=rrs[i];
     }
     *(RRS_num)=(int)rrs.size();
     *(mean_heart_rate)=signal.Mean(&rrs[0],
(int)rrs.size());
  } else {
  return -4; //
  }
  return 0;
}
```

С остальными функциями можно ознкомиться в приложении 2.

Разработанное программное обеспечение работает по следующей схеме (см. рис. 2.14):

1) Считывание;

2) Выбор участка сигнала;

3) Передискретизация;

4) Нормирование;

5) Фильтр низких частот;

6) Фильтр высоких частот;

7) Производная;

8) Производная в квадрате;

9) Интегральный и медианный фильтры;

10) Выделение комплексов QRS;

11) Отображение результатов.

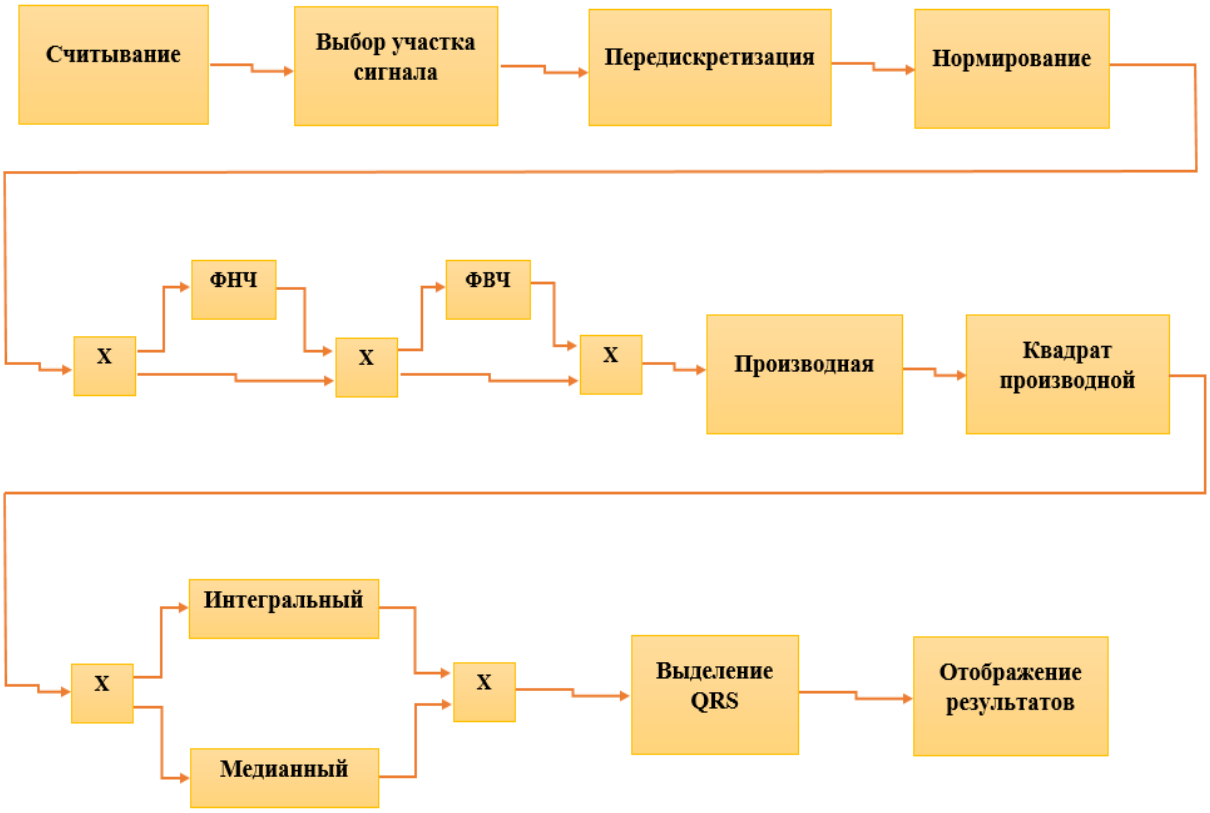


Рис. 2.14 Схема работы программы

ГЛАВА 3. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

После сбора информации о различных существующих программных продуктах, позволяющих работать с данными электрокардиограмм, их изучения и анализа была разработана программа ZOOM ECG. На рис. 3.1 – 3.4 продемонстрирован интерфейс программного обеспечения ZOOM ECG.

На рис. 3.1 можно увидеть поле для выбора файла, содержащего исходный сигнал электрокардиограммы, а также выбора файла из каталога шаблонов QRS. Кроме того, на рисунке изображены поля с номером канала, числом канала, выбранным каналом, временем ЭКГ, числом отсчетов, частотой и разрядностью. Также ниже можно увидеть текущее окно исходного сигнала и кнопки для запоминания текущего окна и перемещения по окнам вперед и назад.

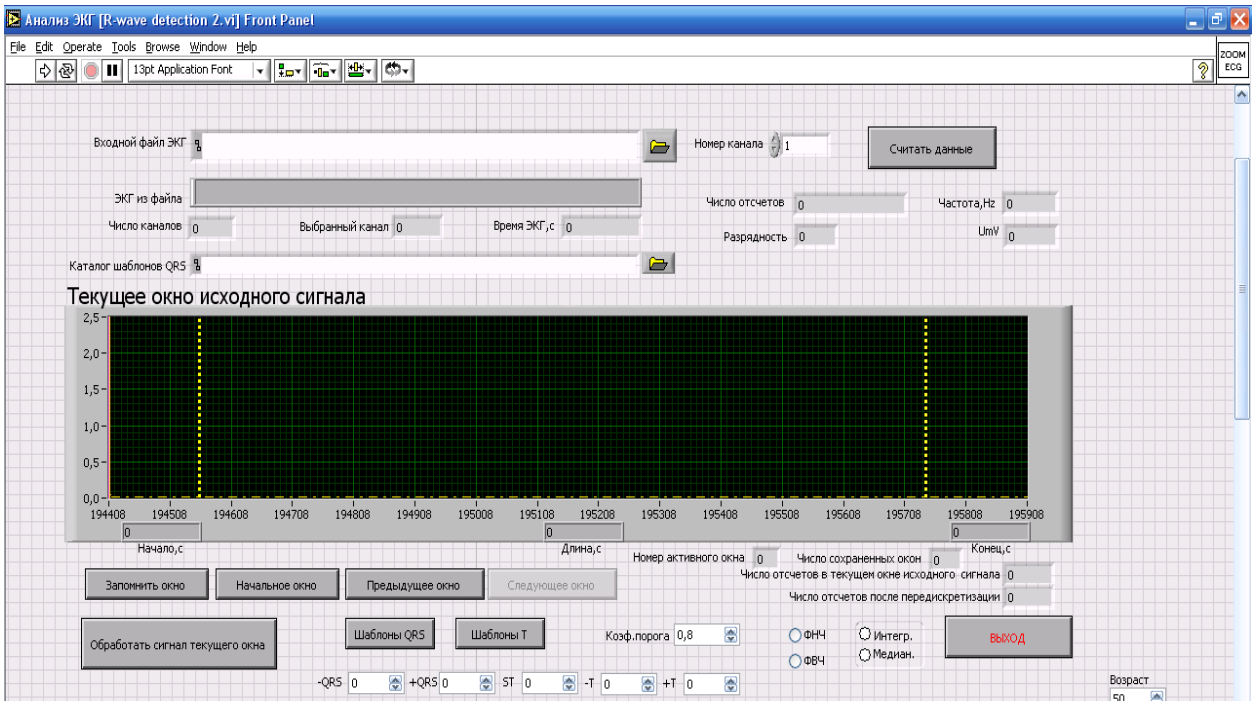


Рис. 3.1 Интерфейс программы ZOOM ECG

На рис. 3.2, который представлен ниже, изображены окна, в которых можно будет увидеть передискретизированный сигнал текущего окна, X и комплексы QRS.

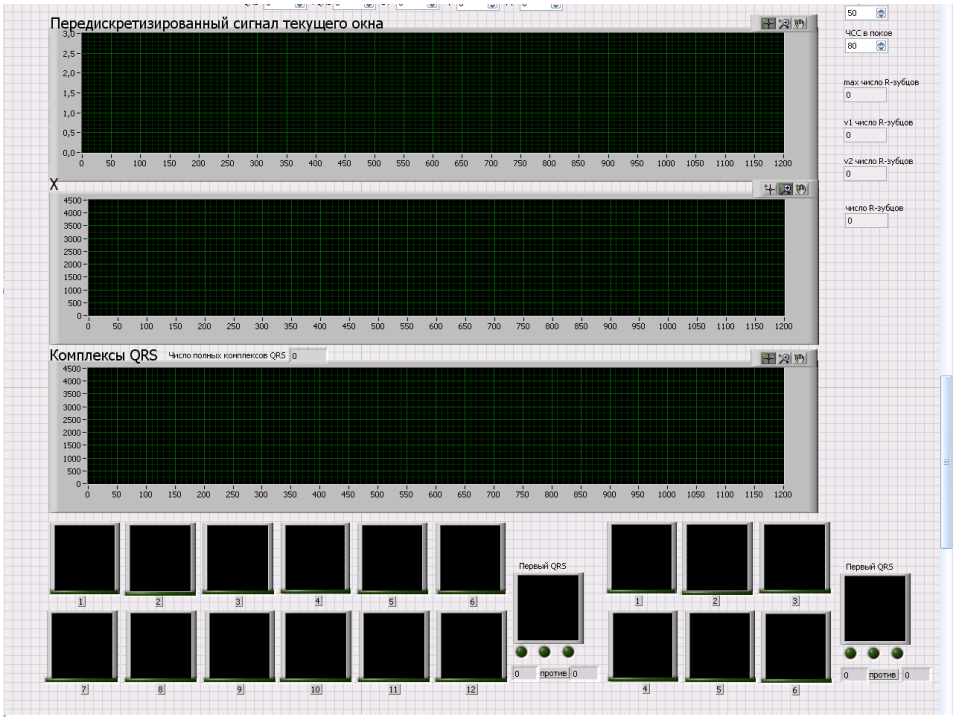


Рис. 3.2 Интерфейс программы ZOOM ECG

На рис. 3.3 показаны окна для отображения сигнала после первой ступени фильтрации, сигнала после второй ступени фильтрации (G), производной от сигнала G.

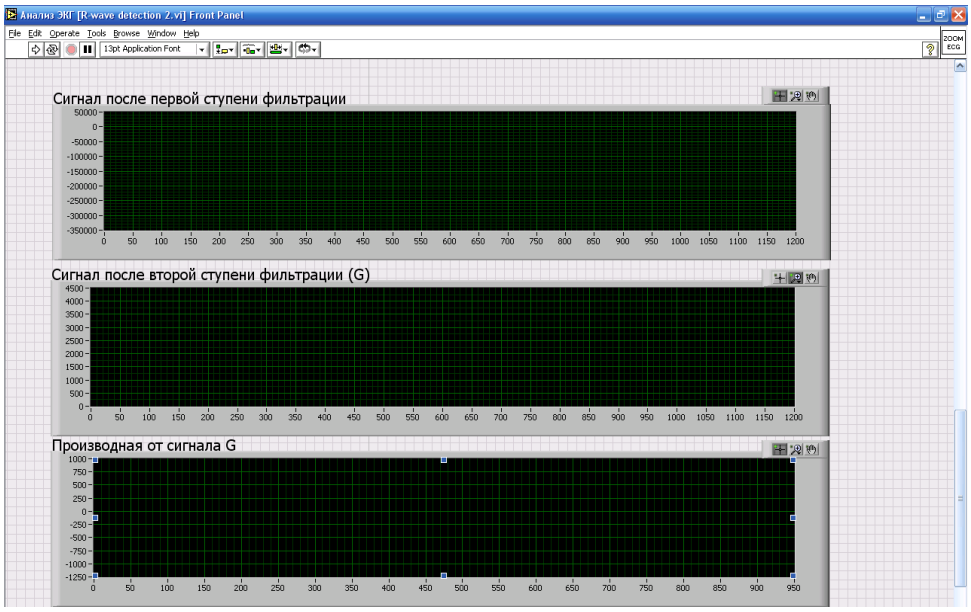


Рис. 3.3 Интерфейс программы ZOOM ECG

Окна для отображения производной в квадрате (P2), интегрального фильтра сигнала P2 и медианного фильтра сигнала P2 изображены на рис. 3.4.

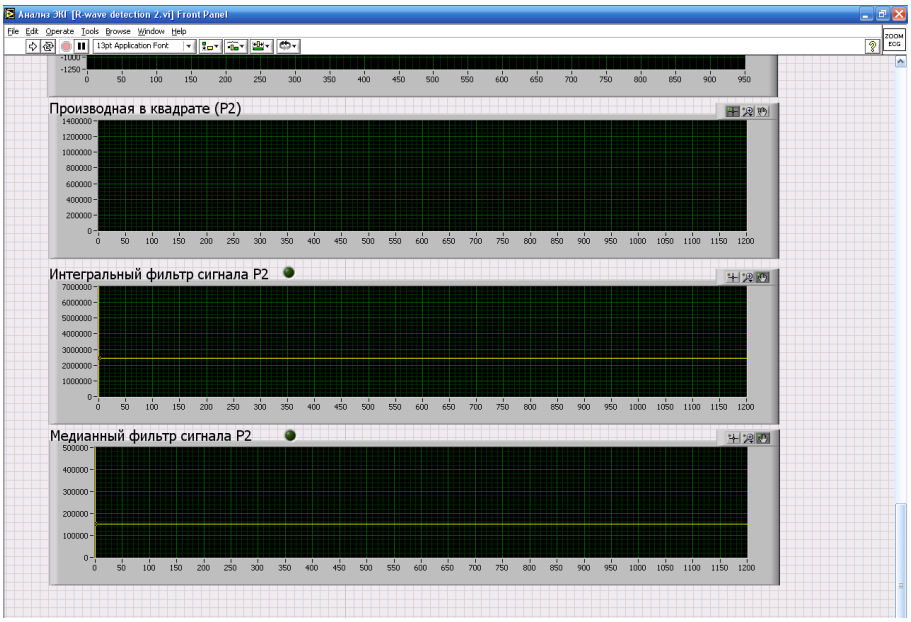


Рис. 3.4 Интерфейс программы ZOOM ECG

Для начала работы необходимо загрузить файл с электрокардиограммой. Процесс загрузки файла продемонстрирован на рис. 3.5.

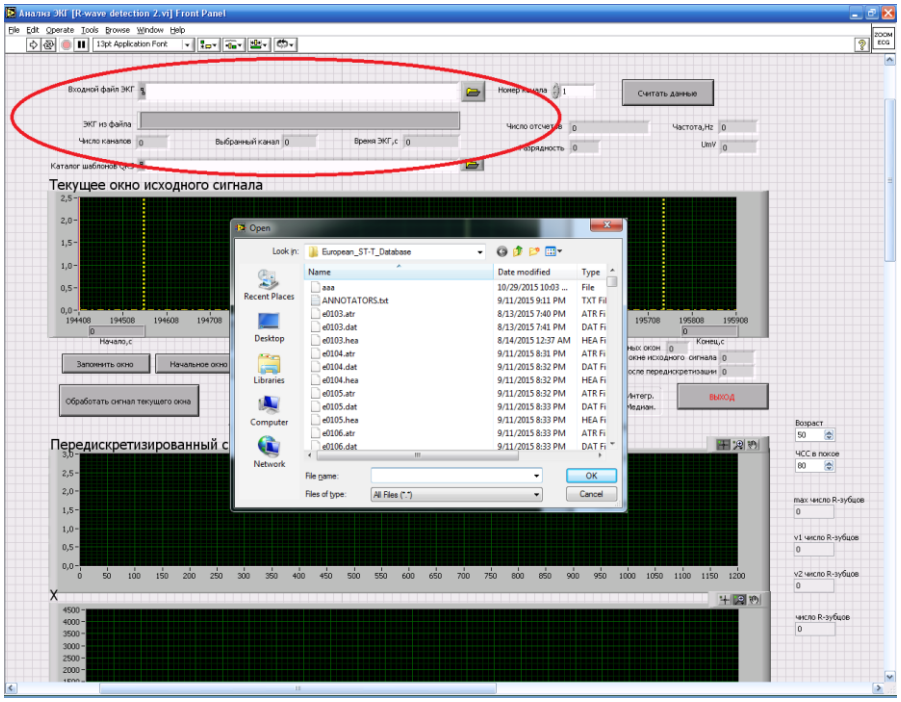


Рис. 3.5 Выбор файла ЭКГ

После загрузки файла ЭКГ в поле для отображения данных, находящемся в центре программы, строится график загруженной ЭКГ (рис. 3.6).

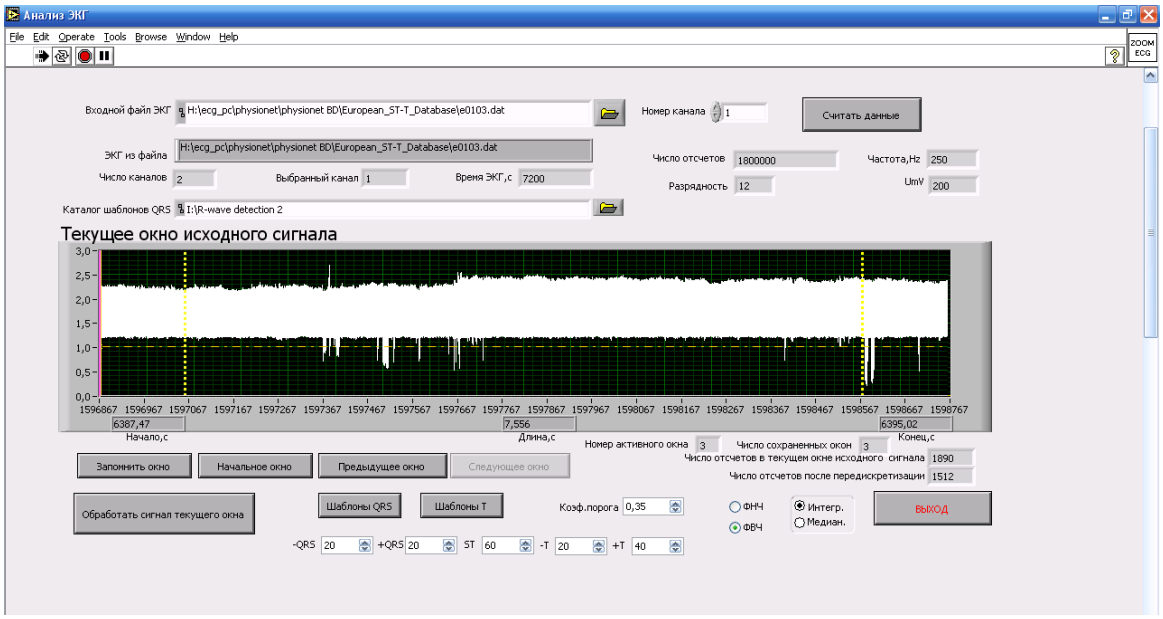


Рис. 3.6 Отображение считанной информации из файла ЭКГ

Затем для подробного просмотра и изучения данных ЭКГ необходимо увеличить некоторую область отображаемого графика. Для этого требуется выделить желтыми рамками необходимый участок ЭКГ и нажать на кнопку «Запомнить окно» (рис. 3.7).

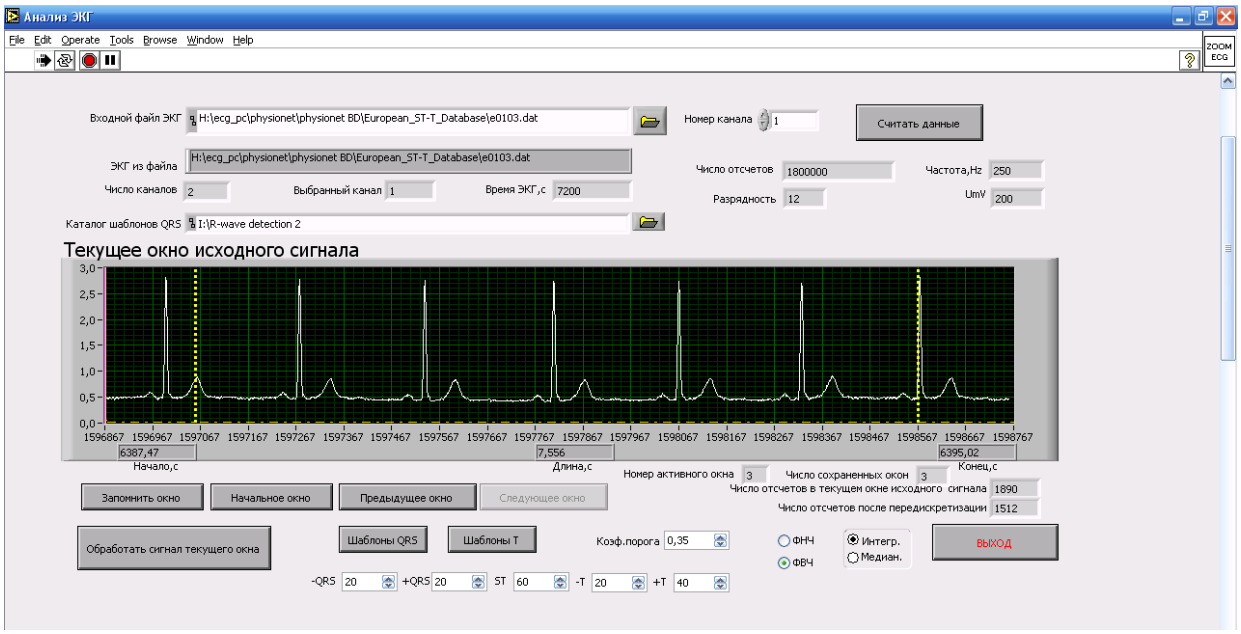


Рис. 3.7 Результат увеличения выбранной области.

После обработки текущего сигнала на экране отображается передискретизированный сигнал текущего окна (рис. 3.8).

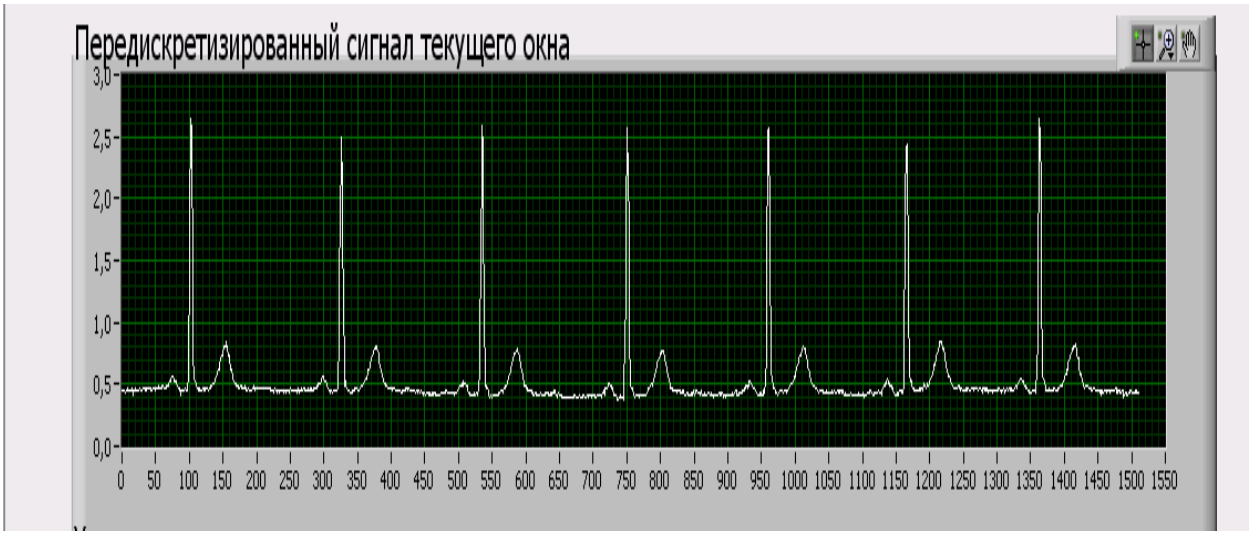


Рис. 3.8 Передискретизированный сигнал текущего окна.

Также помимо передискретизированного сигнала текущего окна отображается и X, что изображено на рис. 3.9.

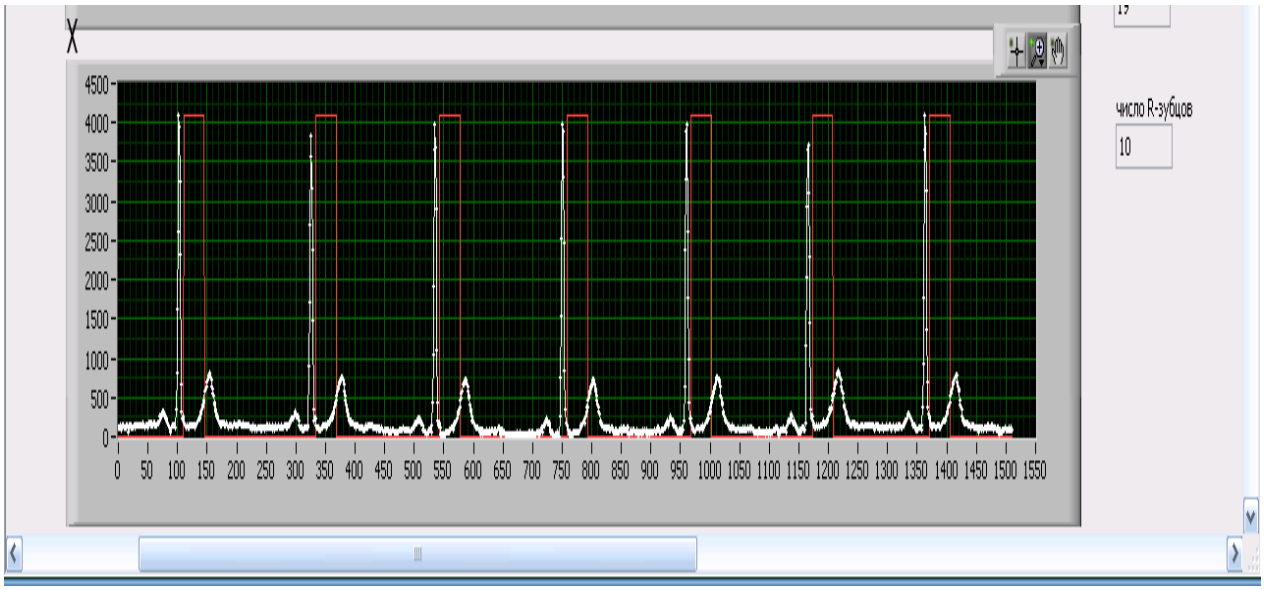


Рис. 3.9 Результат отображения X.

Кроме того в отдельном окне ниже изображаются найденные в выбранной области QRS-комплексы. Можно узнать, сколько полных комплексов QRS было обнаружено.

На рис. 3.10 показаны комплексы QRS.

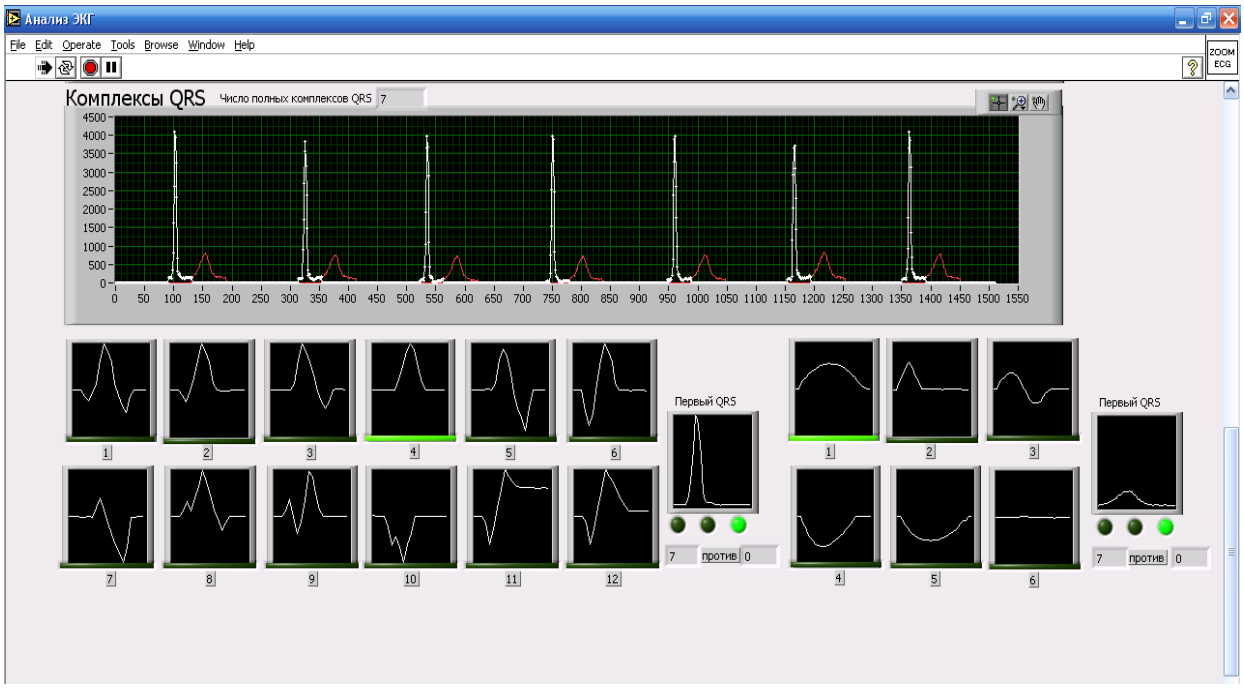


Рис. 3.10 Комплексы QRS.

Помимо всего прочего сбоку можно установить информацию о пациенте, такую как возраст и число сердечных сокращений в покое, а также узнать максимальное число R-зубцов и прочее (см. рис. 3.11).

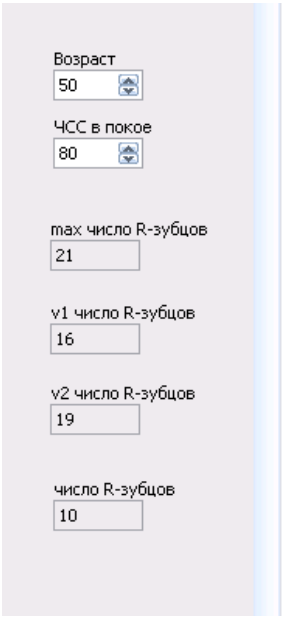


Рис. 3.11 Дополнительная информация.

Далее в специально отведенном окне отображается график сигнала после первой ступени фильтрации, что можно увидеть на рис. 3.12.

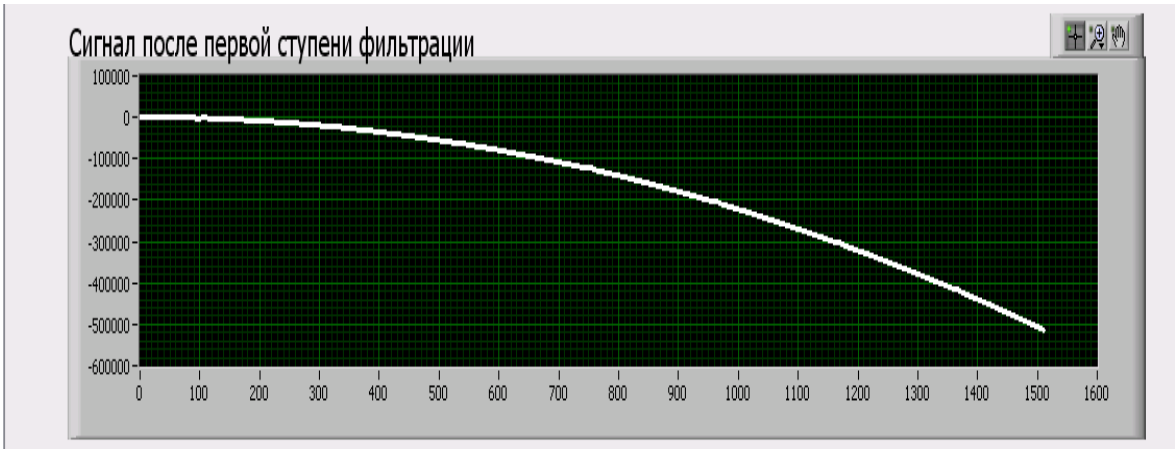


Рис. 3.12 Сигнал после первой ступени фильтрации.

В окне, расположенном чуть ниже, изображается график сигнала после второй ступени фильтрации (G), это показано на рис. 3.13.

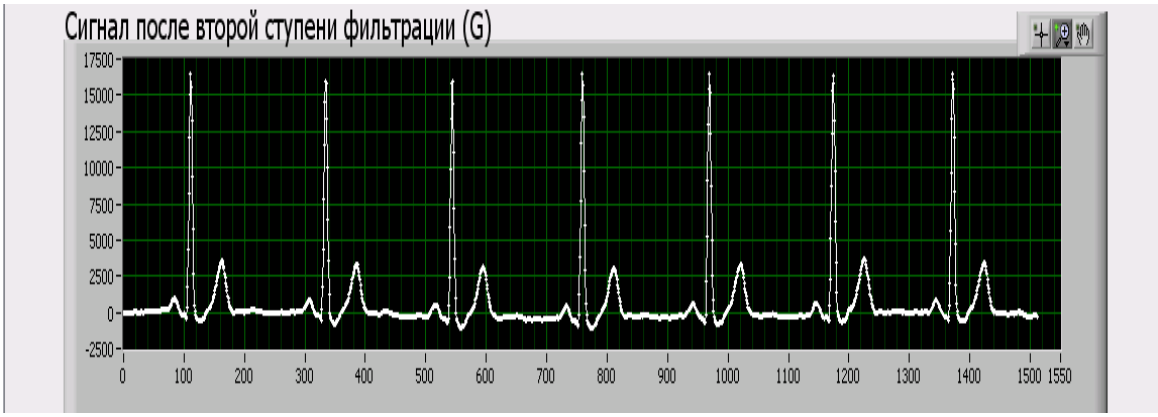


Рис. 3.13 Сигнал после второй ступени фильтрации (G).

Затем можно увидеть график производной от сигнала G (рис. 3.14).

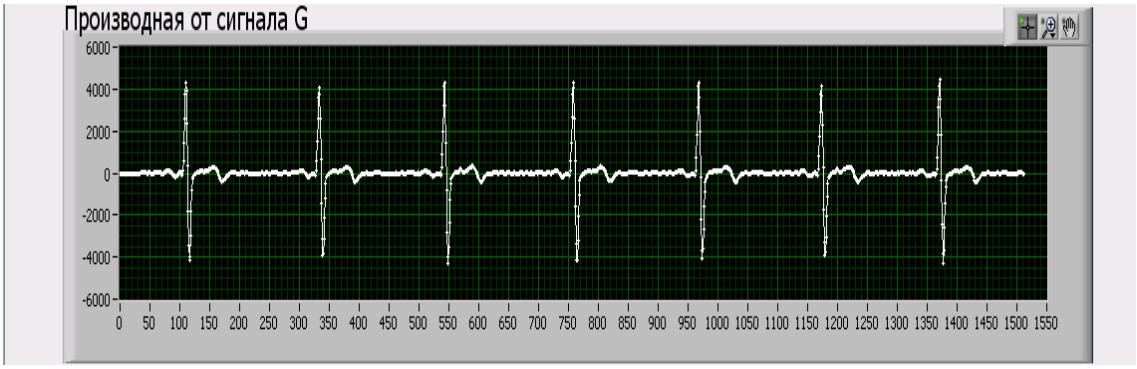


Рис. 3.14 Производная от сигнала G.

Также отображается в отдельном окне и производная в квадрате (P2).

Это показано на рис. 3.15, представленном ниже.

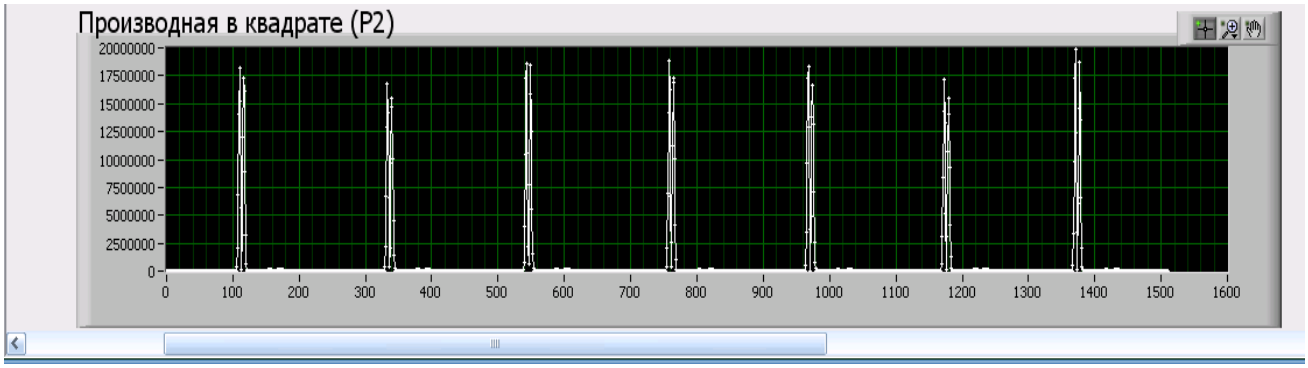


Рис. 3.15 Производная в квадрате (P2).

ЗАКЛЮЧЕНИЕ

Сейчас существует проблема автоматизации работы кардиологов, потому что большое количество программ для работы с кардиологическими данными закрыты или недоступны почти для всех специалистов в этой области медицины. Помимо этого для обработки кардиосигналов с помощью этих программ приходится затрачивать огромное количество не только времени, но и сил. Поэтому необходимость разработки совершенно нового программного обеспечения для работы с электрокардиограммами, которым было бы просто и удобно пользоваться, а также которое было бы общедоступным, является очевидной.

В данной выпускной квалификационной работе была поставлена цель проанализировать существующие алгоритмы обнаружения R-зубца электрокардиосигнала, а также разработать собственный алгоритм, который позволит повысить уровень диагностической информативности и достоверность результатов обследований методом электрокардиографии.

В ходе разработки программы были решены следующие задачи:

1) Проведен аналитический обзор существующих алгоритмов выделения R-зубцов, который включает в себя обзор алгоритмов, основанных на производной, алгоритма Пана и Томпкинса, корреляционного алгоритма и метода, основанного на подсчете числа пересечений нуля;

2) Изучена структура банка данных Physionet и обоснован выбор исходных данных;

3) Обоснован выбор программных и инструментальных средств;

4) Разработана функциональная схема и панель инструментов;

5) Выполнена программная реализация продукта, позволяющего вводить данные ЭКГ из файлов в формате Physionet, просматривать сигнал ЭКГ на графике с возможностью выбора и просмотра отдельных участков

(окон), а также запоминать параметры выбранных окон и передвигаться по ним, находить QRS-комплексы и выделять R-зубцы;

6) Произведено тестирование программного обеспечения, подтвердившее его полную работоспособность.

Таким образом, поставленная цель работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Библиотека аннотации ЭКГ на C++. [Электронный ресурс] URL: https://www.codeproject.com/articles/20995/ecg-annotation-c-library

2. ЭКГ сердца URL: http://www.happydoctor.ru/info/536

3. Электрокардиография URL:https://ru.wikipedia.org/wiki/Электрокардиография

4. LabVIEW URL: https://ru.wikipedia.org/wiki/LabVIEW

5. Physionet.org URL: https://physionet.org

6. Васильков Ю.В., Василькова Н.Н. Компьютерные технологии вычислений в математическом моделировании: учебное пособие. М.: Финансы и статистика, 2004. 256 с.

7. Рослякова А.В., Чупраков П.Г. Сравнительный анализ алгоритмов обнаружения R-зубца электрокардиосигнала. Журнал Вятский медицинский вестник выпуск № 2 / 2012

8. Федосов В.П., Нестеренко А.К. Цифровая обработка сигналов в LabVIEW.

9. Clifford G.D. Signal processing methods for heart rate variability analysis. PhD Thesis. Michaelmas Term. 2002. 244 c.

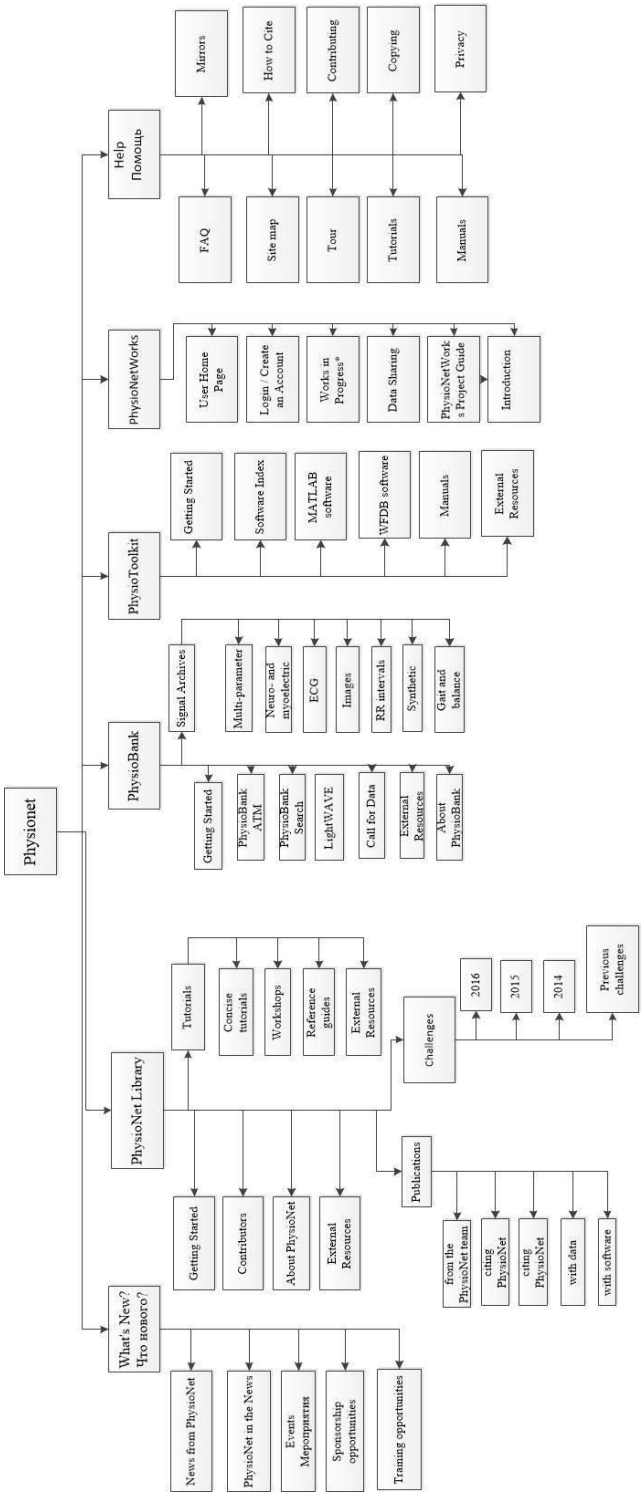
10. Jekova I. Comparison of five algorithms for the detection of ventricular fibrillation from the surface ECG // Physiological Measurement. 2000. Vol. 21. P. 429–439.

11. Зотов Д.Д., "современные методы функциональной диагностики в кардиологии", СПб, 2000.

12. Kohler B.-U., Henning C., Orglmeister R. QRS detection using zero crossing counts // Progress in biomedical research. 2003. Vol. 8(3). P. 138–145.

13. Исаков И. И., Кушаковский М. С., Журавлева Н. Б. Клиническая электрокардиография (нарушения сердечного ритма и проводимости): Руководство для врачей. — Изд. 2-е перераб. и доп. — Л.: Медицина, 1984. — 272 с.

Приложение 1



Приложение 2

```
// ECG_C.cpp: определяет экспортированные функции для приложения
DLL.
//

#include "stdafx.h"
#include "ECG_C.h"

// MY FUNCTION
#include "lib.h"

/*
Функция: ConvNameFile(   const char* name1,
                              wchar_t* name2)
Назначение: Конвертирует имя файла.
Вход:          const char* name1   - исходное имя файла.
Выход:         wchar_t* name2      - выходное имя файла.
*/
void ConvNameFile(const char* name1,
                      wchar_t* name2)
{
    for (int i = 0; i < (int)strlen(name1); i++)
              mbtowc(name2 + i, name1 + i, MB_CUR_MAX);
}


/*
Функция: int ReadSignalSize(const char* nameFileEcg,
                                  int* SignalSize,
                                  int* leads)
Назначение: Определяет размер размер сигнала и число каналов
ЭКГ, записанного в заданном файле ЭКГ.
Вход:          const char* nameFileEcg - имя файла ЭКГ.
Выход:         int* SignalSize               - размер сигнала
(число отсчетов сигнала в кажном канале ЭКГ).
               int* leads                    - число каналов ЭКГ.
Возвращает:     0 - Ошибок нет.
               -1 - Ошибка считывания файла ЭКГ.
Вызываемые функции: ConvNameFile.
Используемые библиотечные элементы: signal.ReadFile,
signal.GetLength, signal.GetLeadsNum.
*/
ECG_C_API
int ReadSignalSize(const char* nameFileEcg,
                       int* SignalSize,
                       int* leads
                       )
{
     wchar_t nameFile[_MAX_PATH] = L"";
```

```
     ConvNameFile(nameFileEcg,nameFile);
     class Signal signal;
    if (signal.ReadFile(nameFile)) {
       *(SignalSize)=signal.GetLength();
        *(leads)=signal.GetLeadsNum();
       return 0;  //Ошибок нет
     } else {
       return -1; //Ошибка считывания файла ЭКГ
     }

}


/*
Функция: ReadSignal(     const char* nameFileEcg,
                              int leadNumber,
                              double* data,
                              int* leads,
                              double* sr,
                              int* bits,
                              int* UmV
                            )
Назначение: Считывает сигнал ЭКГ, по заданному каналу из
заданного файла ЭКГ.
Вход:     const char* nameFileEcg  - имя файла ЭКГ.
          int leadNumber           - номер канала.
Выход:    double* data    - сигнал ЭКГ.
          int* leads           - число каналов ЭКГ.
          double* sr           - частота оцифровки ЭКГ (Гц).
          int* bits       - разрядность АЦП.
          int* UmV        - амплитуда сигнала ЭКГ(mV).
Возвращает:          0 - Ошибок нет.
                    -2 - Ошибка считывания файла ЭКГ.
Вызываемые функции: ConvNameFile.
Используемые библиотечные элементы: signal.ReadFile,
signal.GetLength, signal.GetLeadsNum,
signal.GetSR, signal.GetBits, signal.GetUmV, signal.GetData.
*/
ECG_C_API
int ReadSignal(const char* nameFileEcg,
                  int leadNumber,
                  double* data,
                  int* leads,
                  double* sr,
                  int* bits,
                  int* UmV
                  )
{
     wchar_t nameFile[_MAX_PATH] = L"";
     ConvNameFile(nameFileEcg,nameFile);
//   MessageBox(NULL,nameFile,L"3",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
```

```
     class Signal signal;
    if (signal.ReadFile(nameFile)) {
       int SignalSize=signal.GetLength();
       *(leads)= signal.GetLeadsNum();
      *(sr) = signal.GetSR();
       *(bits)=signal.GetBits();
       *(UmV)=signal.GetUmV();
// ---------------------------обойтись без pData
       double*  pData = signal.GetData(leadNumber);
       for (int i=0; i<SignalSize; i++) data[i]=pData[i];
       return 0;  //Ошибок нет
     } else {
       return -2; //Ошибка считывания файла ЭКГ
     }
}


/*
Функция: int AnnECG(const char* nameDirFilters,
                        double* data,
                        int SignalSize,
                        double sr,
                        int* beats,
                        int* ANN_num,
                        int* ANN_smpl,
                        int* ANN_type
                    )
Назначение: Аннотирование ЭКГ.
Вход:     const char* nameDirFilters    - имя каталога фильтров.
          double* data                  - сигнал ЭКГ.
          int SignalSize                - размер сигнала (число
отсчетов сигнала в кажном канале ЭКГ).
          double sr                     - частота оцифровки ЭКГ
(Гц).
Выход:    int* beats          - Сердечный ритм.
          int* ANN_num   - Число аннотированных точек ЭКГ (размер
массива ANN_smpl и ANN_type).
          int* ANN_smpl  - Номера аннотированных точек ЭКГ в
массиве data (номера аннотированных отсчетов ЭКГ).
          int* ANN_type  - Коды аннотации в аннотированных точках
ЭКГ.
Возвращает:          0 - Ошибок нет.
                    -3 - Ошибка определения QRS-комплекса.
Вызываемые функции: ConvNameFile.
Используемые библиотечные элементы: ann.GetQRS,
ann.GetQrsNumber, ann.GetEctopics, ann.GetPTU,
ann.GetEcgAnnotationSize, ann.GetQrsNumber.
*/
ECG_C_API
int AnnECG(const char* nameDirFilters,
            double* data,
            int SignalSize,
```

```
            double sr,
            int* beats,
            int* ANN_num,
            int* ANN_smpl,
            int* ANN_type
            )
{
     wchar_t nameDirF[_MAX_PATH] = L"";
     ConvNameFile(nameDirFilters,nameDirF);
// MessageBox(NULL,nameDirF,L"3",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
  class EcgAnnotation ann;  //default annotation params
                            //or add your custom ECG params to
annotation class from lib.h
                           // ANNHDR hdr;
                           //  hdr.minbpm = 30;
                            //  etc...
                           // class EcgAnnotation ann( &hdr );
  class Signal signal;

  //  tic();
   int** qrsAnn = ann.GetQRS(data, SignalSize, sr, nameDirF);
//get QRS complexes
    *(beats)=0;
  if (qrsAnn) {
     *(beats)=ann.GetQrsNumber();
      ann.GetEctopics(qrsAnn, ann.GetQrsNumber(), sr);
//label Ectopic beats

     int annNum = 0;
     int** ANN = ann.GetPTU(data, SignalSize, sr, nameDirF,
qrsAnn, ann.GetQrsNumber());     //find P,T waves
     if (ANN) {
        annNum = ann.GetEcgAnnotationSize();
     } else {
        ANN = qrsAnn;
        annNum = 2 * ann.GetQrsNumber();
     }

     //printing out annotation
     for (int i = 0; i < annNum; i++) {
        ANN_smpl[i]=ANN[i][0];
          ANN_type[i]=ANN[i][1];
      }
      *(ANN_num)=annNum;

  } else {
       return -3; //could not get QRS complexes. make sure you
have got \"filters\" directory in the ecg application dir.;
  }
  return 0;
}
```

```
/*
Функция: int RRS_ECG(    double sr,
                              int ANN_num,
                              int* ANN_smpl,
                              int* ANN_type,
                              int* RRS_num,
                              int* RRS_smpl,
                              double* RRS,
                              double* mean_heart_rate
                     )
Назначение: Выделение RRS.
Вход:     double sr      - частота оцифровки ЭКГ (Гц).
          int ANN_num         - Число аннотированных точек ЭКГ
(размер массива ANN_smpl и ANN_type).
          int* ANN_smpl  - Номера аннотированных точек ЭКГ в
массиве data (номера аннотированных отсчетов ЭКГ).
          int* ANN_type  - Коды аннотации в аннотированных точках
ЭКГ.
Выход:    int* RRS_num   - Число точек RRS (размер массива
RSS_smpl и RRS).
          int* RRS_smpl  - Номера точек RRS в массиве data
(номера отсчетов ЭКГ).
          double* RRS         - Значение в точке RRS.
          double* mean_heart_rate
Возвращает:          0 - Ошибок нет,
                    -4 - Ошибка определения RRS.
Используемые библиотечные элементы: ann.GetRRseq, signal.Mean.
*/
ECG_C_API
int RRS_ECG(double sr,
             int ANN_num,
             int* ANN_smpl,
             int* ANN_type,
             int* RRS_num,
             int* RRS_smpl,
           double* RRS,
             double* mean_heart_rate    //
             )
{
  class EcgAnnotation ann;  //default annotation params
                            //or add your custom ECG params to
annotation class from lib.h
                           // ANNHDR hdr;
                           //  hdr.minbpm = 30;
                            //  etc...
                           // class EcgAnnotation ann( &hdr );
  class Signal signal;

   int **ANN;
   ANN = new int* [ANN_num];
```

```
   for (int i = 0; i < ANN_num; i++) {
      ANN[i] = new int[3];
      ANN[i][0]=ANN_smpl[i];
      ANN[i][1]=ANN_type[i];
   }

   //saving RR seq
   vector<double> rrs;
   vector<int> rrsPos;
   if (ann.GetRRseq(ANN, ANN_num, sr, &rrs, &rrsPos)) {
     for (int i = 0; i < (int)rrs.size(); i++) {
       RRS_smpl[i]=rrsPos[i];
       RRS[i]=rrs[i];
     }
     *(RRS_num)=(int)rrs.size();
     *(mean_heart_rate)=signal.Mean(&rrs[0], (int)rrs.size());
  } else {
       return -4; //
  }
  return 0;
}

/*
Функция: int Border(     int num,
                              int* smpl,
                              int start,
                              int end,
                              int *b1,
                              int *b2
                            )
Назначение: Выделение участка аннотации или RRS, лежащего в
заданном интервале (окне) ЭКГ.
Вход:     int num         - Число аннотированных точек (размер
массива ANN_smpl и ANN_type)
                              или число точек RRS (размер массива
RSS_smpl и RRS).
          int* smpl - Номера аннотированных точек ЭКГ в массиве
data (номера аннотированных отсчетов ЭКГ)
                              или номера точек RRS в массиве data
(номера отсчетов ЭКГ).
          int start - Начальная точка (начальный отсчет)
заданного интервала (окна) ЭКГ.
          int end         - Конечная точка (конечный отсчет)
заданного интервала (окна) ЭКГ.
Выход:    int* b1         - Номер первой аннотированной точки
(индекс в массивах ANN_smpl и ANN_type)
                              или номер первой точки RRS (индекс
в массивах RRS_smpl и RRS),
                              принадлежащей заданному интервалу
ЭКГ.
          int* b2         - Номер последней аннотированной точки
(индекс в массивах ANN_smpl и ANN_type)
```

```
                                или номер последней точки RRS
(индекс в массивах RRS_smpl и RRS),
                                принадлежащей заданному интервалу
ЭКГ.
Возвращает:          0
*/
ECG_C_API
int Border(int num,
             int* smpl,
             int start,
             int end,
             int *b1,
             int *b2
             )
{
     int i1=0,i2;
     if (start>end) return -1;
     while (!((i1>=num)||(smpl[i1]>=start))) i1++;
     i2=i1;
     while (!((i2>=num)||(end<=smpl[i2]))) i2++;
     *(b1)=i1;
     *(b2)=i2;
     return 0;
}



/*
Функция: int ContourECG( double* data,
                                    int ANN_num,
                                    int* ANN_smpl,
                                    int* ANN_type,
                                    int start,
                                    int end,
                                    double* contour
                         )
Назначение: Расчет изолинии.
Вход:     double* data   - сигнал ЭКГ.
          int ANN_num         - Число аннотированных точек ЭКГ
(размер массива ANN_smpl и ANN_type).
          int* ANN_smpl  - Номера аннотированных точек ЭКГ в
массиве data (номера аннотированных отсчетов ЭКГ).
          int* ANN_type  - Коды аннотации в аннотированных точках
ЭКГ.
          int start      - Начальная точка (начальный отсчет)
заданного интервала (окна) ЭКГ.
          int end             - Конечная точка (конечный отсчет)
заданного интервала (окна) ЭКГ.
Выход:    double* contour - Значение изолинии (y=contour - прямая
изолинии).
Возвращает:          0 - Ошибок нет,
```

```
                    -2 - Ошибка расчета изолинии (число
анализируемых точек =0, попытка деления на 0).
*/
ECG_C_API
int ContourECG(double* data,
                    int ANN_num,
                    int* ANN_smpl,
                    int* ANN_type,
                    int start,
                    int end,
                    double* contour
                       )
{
     int b1,b2,i,id,k=0;
     double s, sk=0;
     *(contour)=0;
     if (Border(ANN_num,ANN_smpl,start,end,&b1,&b2)!=0) return -
1;
     i=b1;
     while (i<=b2){
          while (!((i>b2)||(ANN_type[i]==43))) i++;
          if (i<=b2){
               id=ANN_smpl[i]; i++;
               while
(!((i>b2)||(ANN_type[i]==43)||(ANN_type[i]==1))) i++;
               if  ((i<=b2) && (ANN_type[i]==1)) {
                    s=0.0;
                    for (int j=id ; j<=ANN_smpl[i]; j++)
     s=s+data[j];
                    s=s/(ANN_smpl[i]-id+1);
                    sk=sk+s; k++;
               }
               i++;
          }
     }
     if (k==0) return -2;
     *(contour)=sk/k;
     return 0;
}


/*
Функция: int ContourECG_a(    double* data,
                                   int ANN_num,
                                   int* ANN_smpl,
                                   int* ANN_type,
                                   int start,
                                   int end,
                                   double* contour
                              )
Назначение: Расчет изолинии.
```

```
              * Функция ContourECG_a, в отличие от функции
ContourECG, реализована в автоматном стиле.
Вход:     double* data   - сигнал ЭКГ.
          int ANN_num         - Число аннотированных точек ЭКГ
(размер массива ANN_smpl и ANN_type).
          int* ANN_smpl  - Номера аннотированных точек ЭКГ в
массиве data (номера аннотированных отсчетов ЭКГ).
          int* ANN_type  - Коды аннотации в аннотированных точках
ЭКГ.
          int start      - Начальная точка (начальный отсчет)
заданного интервала (окна) ЭКГ.
          int end             - Конечная точка (конечный отсчет)
заданного интервала (окна) ЭКГ.
Выход:    double* contour - Значение изолинии (y=contour - прямая
изолинии).
Возвращает:          0 - Ошибок нет,
                    -2 - Ошибка расчета изолинии (число
анализируемых точек =0, попытка деления на 0).
*/

ECG_C_API
int ContourECG_a(double* data,
                    int ANN_num,
                    int* ANN_smpl,
                    int* ANN_type,
                    int start,
                    int end,
                    double* contour
                       )
{
     int b1,b2,i,st=0,e=0,k=0;
     double s, sk=0;
     *(contour)=0;
     if (Border(ANN_num,ANN_smpl,start,end,&b1,&b2)!=0) return -
1;

     i=b1;

_m1:if (i>b2)            { goto _m5;}
     if (ANN_type[i]==45){ st=ANN_smpl[i]; i++; goto _m2; }// t)
     else                { i++; goto _m1; }

_m2:if (i>b2)            { goto _m5;}
     if (ANN_type[i]==43){ st=ANN_smpl[i]; i++; goto _m3; }// p)
     if (ANN_type[i]==1) { e=ANN_smpl[i];  i++; goto _m4; } // N
    if (ANN_type[i]==45){ st=0;    e=0; goto _m1; }
     // t)
     else                { i++;  goto _m2;}

_m3:if (i>b2)            { goto _m5;}
     if (ANN_type[i]==1) { e=ANN_smpl[i];  i++; goto _m4; } // N
```

```
    if (ANN_type[i]==45){ st=0;    e=0; goto _m1; }
     // t)
     else                { i++;  goto _m3;}

_m4:s=0; for (int j=st; j<=e; j++) s=s+data[j];
     s=s/(e-st+1);  sk=sk+s; k++;
     st=0;     end=0;
     goto _m1;

_m5:if (k==0) return -2;
    else *(contour)=sk/k;

    return 0;
}

/*
Функция: int SeachL(int LT[],
                        int uLT[],
                        int *numL,
                        int L[],
                        int lenL,
                        int m_uLT,
                        int m_LT)
Назначение: Определяет уникальность заданной линии аннотации ЭКГ
и проверяет возможность
               добавления уникальной линии аннотации в структуру
гнездового типа.
Вход:     int* LT        - Уникальные линии аннотации
(последовательность кодов аннотации) хранящиеся в гнездовой
структуре.
          int* uLT  - Указатели (курсоры) на начало гнезд в
массиве LT.
          int* numL - Число уникальных линий аннотации равно
(размер массива uLT равен numL+1).
          int* L         - Текущая линия аннотации.
          int lenL  - Длина текущей линии аннотации.
          int m_uLT - Максимальный размер массива uLT.
          int m_LT  - Максимальный размер массива LT.
Возвращает:     0 - линия аннотации не уникальна.
                1 - линия аннотации уникальна и может быть
добавлена функцией AddL в гнездовую структуру (массивы LT, uLT).
               -1 - линия аннотации уникальна, но для ее
добавления в гнездовую структуру нет места в массиве LT.
               -2 - линия аннотации уникальна, но для ее
добавления в гнездовую структуру нет места в массиве uLT.
*/
int SeachL(int LT[],
             int uLT[],
             int *numL,
             int L[],
             int lenL,
             int m_uLT,
```

```
              int m_LT)
{
     int f=0, i=0, j=0;
     while ((i<*numL)&&(f==0))
          if (uLT[i+1]-uLT[i]==lenL) {
               j=0;
               while (LT[uLT[i]+j]==L[j]) j++;
               if (j<lenL) i++; else f=1;
          }
          else i++;
     if (f==1) return 0;
     if (*numL+2>m_uLT) return -2;
     if (uLT[*numL]+lenL>m_LT) return -1;
     return 1;
}

/*
Функция: void AddL( int LT[],
                         int uLT[],
                         int *numL,
                         int L[],
                         int lenL
            )
Назначение: Добавляет линию аннотации ЭКГ в структуру гнездового
типа.
Вход:     int* L         - Текущая линия аннотации.
          int lenL  - Длина текущей линии аннотации.
Выход:    int* LT        - Уникальные линии аннотации
(последовательность кодов аннотации) хранящиеся в гнездовой
структуре.
          int* uLT  - Указатели (курсоры) на начало гнезд в
массиве LT.
          int* numL - Число уникальных линий аннотации равно
(размер массива uLT равен numL+1).
*/

void AddL(int LT[],
            int uLT[],
            int *numL,
            int L[],
            int lenL)
{
     for (int i=0; i<lenL; i++)
          LT[uLT[*numL]+i]=L[i];
     uLT[*numL+1]=uLT[*numL]+lenL;
     *numL=*numL+1;
}
/*
Функция: int ALine( int* AT,
                         int lenA,
                         int cl,
                         int* LT,
```

```
                        int* uLT,
                        int* numL,
                        int m_LT,
                        int m_uLT,
                        int* L,
                        int m_L
                )
Назначение: Выделение линий аннотации ЭКГ с заданным начальным
кодом аннотации.
                * Функция ALine реализована в автоматном стиле.
Вход:     int* AT       - Коды аннотации в аннотированных точках
окна ЭКГ.
          int lenA  - Число аннотированных точек окна ЭКГ (размер
массива AT).
          int cl        - Начальный код линии аннотации (если
cl==-1, то по умолчанию cl=44 - код "(t" ).
          int m_LT  - Максимальный размер массива LT.
          int m_uLT - Максимальный размер массива uLT.
          int* L        - Текущая линия аннотации.
          int m_L       - Максимальный размер массива L.
Выход:    int* LT       - Уникальные линии аннотации
(последовательность кодов аннотации) хранящиеся в гнездовой
структуре.
          int* uLT  - Указатели (курсоры) на начало гнезд в
массиве LT.
          int* numL - Число уникальных линий аннотации равно
(размер массива uLT равен numL+1).
Возвращает:     0 - Не найден начальный код линии аннотации.
                1 - Ошибок нет, линии аннотации выделены.
               -1 - Для массива LT требуется размер больше, чем
m_LT.
               -2 - Для массива uLT требуется размер больше, чем
m_uLT.
               -3 - Для массива L требуется размер больше, чем
m_L.
Вызываемые функции: SeachL, AddL.
*/
ECG_C_API
int ALine(          int* AT,
                    int lenA,
                    int cl,
                    int* LT,
                    int* uLT,
                    int* numL,
                    int m_LT,
                    int m_uLT,
                    int* L,
                    int m_L
                )
{
int i,c,r=0,lenL;
if (cl==-1) cl=44; //по умолчанию код "(t"
```

```
//wchar_t SW[20];
//MessageBox(NULL,_itow(lenA, SW, 10),L"1",MB_OK |MB_DEFBUTTON1
| MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);

_m1:
//MessageBox(NULL,_itow(1, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          i=0; uLT[0]=0; *numL=0;  goto _m2;

_m2:
//MessageBox(NULL,_itow(2, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          if (i>=lenA)   { r=0; goto _m5;}
          if (AT[i]!=cl) { i++; goto _m2; }
          if (AT[i]==cl) { L[0]=cl; lenL=1; i++; goto _m3; }

_m3:
//MessageBox(NULL,_itow(3, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          if (i>=lenA)   { r=1; goto _m5;}
          if (lenL>=m_L) { r=-3; goto _m5;}
          if (AT[i]!=cl) {
//MessageBox(NULL,_itow(AT[i], SW, 10),L"2",MB_OK |MB_DEFBUTTON1
| MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);

               L[lenL]=AT[i]; lenL++; i++; goto _m3; }
          if (AT[i]==cl) {
//MessageBox(NULL,_itow(AT[i], SW, 10),L"2",MB_OK |MB_DEFBUTTON1
| MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);

               c=SeachL(LT,uLT,numL,L,lenL,m_uLT,m_LT); goto _m4;
}

_m4:
//MessageBox(NULL,_itow(4, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          if (c<0)       { r=c; goto _m5;}
          if (c==0)      { goto _m2;  }
          if (c==1)      { AddL(LT,uLT,numL,L,lenL); goto _m2; }

_m5:
//MessageBox(NULL,_itow(5, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          return r;

}

/*
Функция: int SLine( int* AT,
                         int* AS,
                         int lenA,
```

```
                        int* LT,
                        int* uLT,
                        int numL,
                        int  nL,
                        int* L,
                        int* LS,
                        int* lenLS,
                        int m_LS
                    )
Назначение: Нахождение номеров первых аннотированных точек линии
с заданным номером.
              * Функция SLine реализована в автоматном стиле.
Вход:     int* AT        - Коды аннотации в аннотированных точках
окна ЭКГ.
          int* AS        - Номера аннотированных точках окна ЭКГ.
          int lenA  - Число аннотированных точек окна ЭКГ (размер
массива AT и AS).
          int* LT        - Уникальные линии аннотации
(последовательность кодов аннотации) хранящиеся в гнездовой
структуре.
          int* uLT  - Указатели (курсоры) на начало гнезд в
массиве LT.
          int numL  - Число уникальных линий аннотации равно
(размер массива uLT равен numL+1).
          int  nL        - Номер линии аннотации (0<=nl<numL
numL+1 - размер массива uLT).
          int* L         - Текущая линия аннотации.
          int m_LS  - Максимальный размер массива LS.
Выход:    int* LS        - Номера первых аннотированных точек
линии с номером nL.
          int* lenLS     - Размер массива LS.
Возвращает:     0 - Линия аннотации с номером nL не существует
(массив LS - пустой).
               1 - Ошибок нет, номера первых точек линии
аннотации с номером nL выделены.
              -4 - Для массива LS требуется размер больше, чем
m_LS.
*/
ECG_C_API
int SLine(          int* AT,
                    int* AS,
                    int lenA,
                    int* LT,
                    int* uLT,
                    int numL,
                    int  nL,
                    int* L,
                    int* LS,
                    int* lenLS,
                    int m_LS
                )
{
```

```
int i,j,k=0,cl,r=0,lenL;

//wchar_t SW[20];
//MessageBox(NULL,_itow(nL, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);

_m1: if (nL>=numL) { r=0; goto _m7;}
//MessageBox(NULL,_itow(1, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          lenL=uLT[nL+1]-uLT[nL];  i=uLT[nL]; cl=LT[i]; j=0;
k=0;      goto _m2;

_m2:
//MessageBox(NULL,_itow(2, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          if (j>=lenL)   { i=0; goto _m3;}
          L[j]=LT[i]; i++; j++; goto _m2;
_m3:
//MessageBox(NULL,_itow(3, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          if (i>=lenA)   { r=1; goto _m7;}
          if (AT[i]!=cl) { i++; goto _m3;}
          if (AT[i]==cl) { j=0; goto _m4;}

_m4:
          if ((i>=lenA)) { r=1; goto _m7;}
          if (j>=lenL)   goto _m5;
          if (AT[i]==L[j]){ i++; j++; goto _m4; }
          if (AT[i]!=L[j]){ goto _m3; }
_m5:
          if (AT[i]==cl) {
//MessageBox(NULL,_itow(j, SW, 10),L"j",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
//MessageBox(NULL,_itow(i, SW, 10),L"i",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
//MessageBox(NULL,_itow(AT[i], SW, 10),L"AT[i]",MB_OK
|MB_DEFBUTTON1 | MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);

          LS[k]=AS[i-lenL]; k++; goto _m6;
          }
          goto _m3;
_m6:
//MessageBox(NULL,_itow(5, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          if (k>=m_LS)   { r=-4; goto _m7;}
          goto _m3;

_m7:
//MessageBox(NULL,_itow(6, SW, 10),L"1",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);
          *lenLS=k;
          return r;
```

```
}

//   wchar_t SW[20];
//   MessageBox(NULL,_itow(d, SW, 10),L"2",MB_OK |MB_DEFBUTTON1 |
MB_ICONEXCLAMATION | MB_DEFAULT_DESKTOP_ONLY);



/*

// Пример экспортированной переменной
ECG_C_API int nECG_C=0;

// Пример экспортированной функции.
ECG_C_API int fnECG_C(void)
{
     return 42;
}


// Конструктор для экспортированного класса.
// см. определение класса в ECG_C.h
CECG_C::CECG_C()
{
     return;
}

*/
```

Выпускная квалификационная работа написана мною совершенно самостоятельно. Все использованные в работе материалы концепции из опубликованной научной литературы и источников имеют ссылки на них.

«__»·_____·_____

_____ _____

(*подпись)* *(Ф. И. О.)*