4. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

Программное обеспечение позволяет выполнять и визуализировать основные операции, выполняемые циклическими кодами – кодирование, декодирование и коррекция. При этом, программное обеспечение имитирует работу линейных переключательных схем (регистров сдвига), на основе которых зачастую в литературе описывается возможная реализация системы кодирования и декодирования циклического кода [1]. Кроме этого, до широкого развития микропроцессорной техники на основе сдвиговых регистров, осуществлялась аппаратная реализация устройств кодирования и декодирования. Для более наглядного представления работы кодера и декодера генерация очередного такта для переключения регистра сдвига выполняется пользователем, также добавлен счетчик тактов.

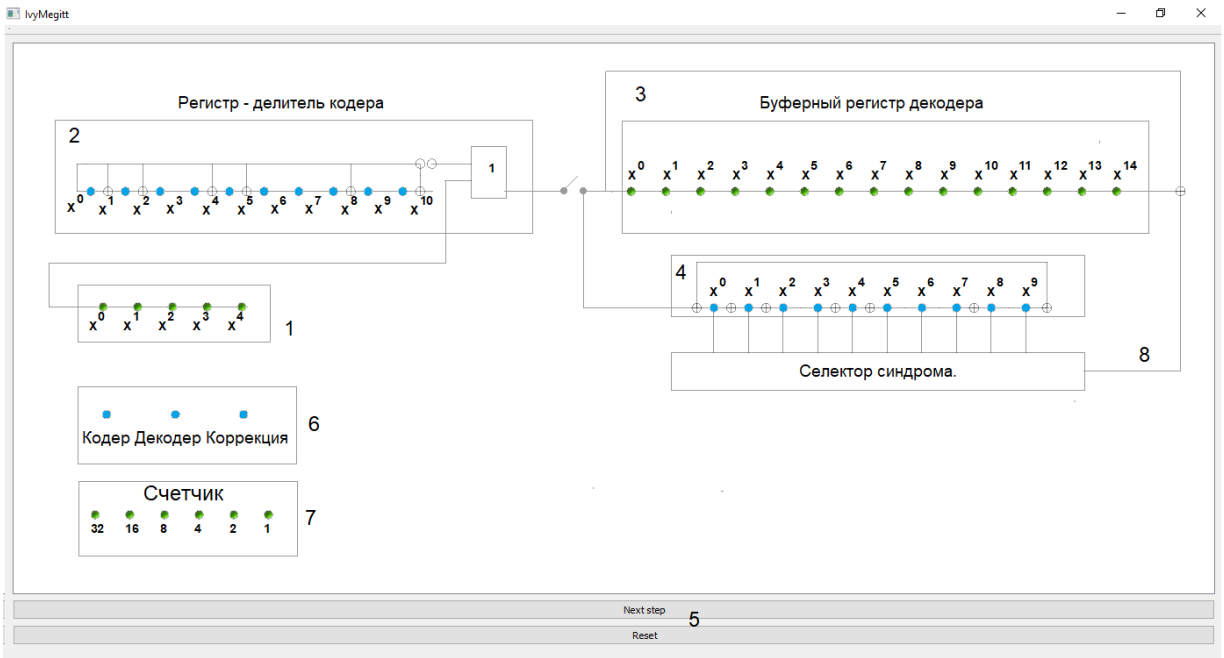

Рис. 4.1. Вид рабочего окна программного обеспечения. 1 регистр для ввода кодируемого сообщения, 2 – регистр-делитель кодера, 3 – буферный кольцевой регистр для хранения закодированного и декодируемого сообщения, 4 – регистр-делитель декодера и селектор синдромов, 5- кнопка генератора очередного такта и сброса, 6 – индикатор текущего режима, 7 – счетчик тактов, 8 сигнал коррекции

Оператор осуществляет ввод кодируемого сообщения при помощи регистра 1, путем нажатия на соответствующий элемент регистра левой клавишей мыши. После того, как сообщение введено, оператору следует нажать на кнопку «Следующий такт», что приведет к запуску кодера, номер такта отображается счетчиком 7. Первые 15 тактов занимает кодирование, состояние регистра делителя кодера отображается блоком 1, первые 5 тактов работы формируется остаток от деления сообщения на порождающий полином, после чего ключ переходит в состояние 2, тем самым отключая обратную связь(цепь деления выключается) и происходит выдача остатка от деления.

После 15 тактов работы в регистре 3 будет записано закодированное сообщение, а индикатор 6 переключится в состояние «декодер», после этого становится возможным внести изменение в закодированное сообщение, путем нажатия на соответствующие элементы регистра 3. Последующие 15 тактов работы запустят деление сообщения на порождающий полином, и будет получен синдром ошибки. Следующие 15 тактов работы отвечают за коррекцию – выполняется циклический сдвиг сообщения в регистре 3 и циклический сдвиг синдрома в регистре делителе 4. В случае, если на очередном сдвиге получен селектируемый синдром, то выдается коррекция младшего разряда слова в регистре 3.

Программное обеспечение состоит из трех основных блоков:

1. Алгоритмический блок. Данный блок отвечает за реализацию рассмотренных ранее схем кодирования, декодирования и коррекции ошибки.
2. Блок реализации графического интерфейса. Данный блок содержит группу классов, которые отвечают за формирование изображений ячеек регистров сдвига, обратных связей, визуализацию сумматоров.
3. Главное окно. Данный блок отвечает за формирование графической сцены, размещение управляющих кнопок, а также за размещение элементов блока графического интерфейса на сцене.

На рисунке 4.2 показана диаграмма классов разработанного программного обеспечения

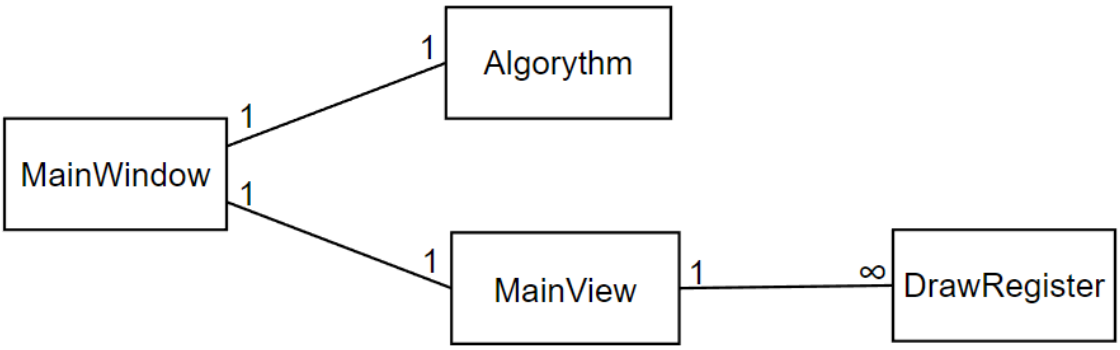

Рис. 4.2. Диаграмма классов

Входной точкой в приложение является класс CMainWindow, он является наследником от QMainWindow, данный класс содержит в качестве приватных полей экземпляр класса CAlgorythm, который реализует работу с циклическим кодом, экземпляр класса CMainView который отвечает за графическую сцену (отношение 1 к 1). Класс CMainView содержит экземпляры базового класса CDrawRegister, которые сужаются до реализации графического представления кодера, декодера, входного регистра, счетчика тактов.

Как было отмечено ранее, программное обеспечение реализовано при помощи фреймворка Qt, в результате чего взаимосвязь между экземплярами классов осуществляется при помощи механизма сигналов и слотов.

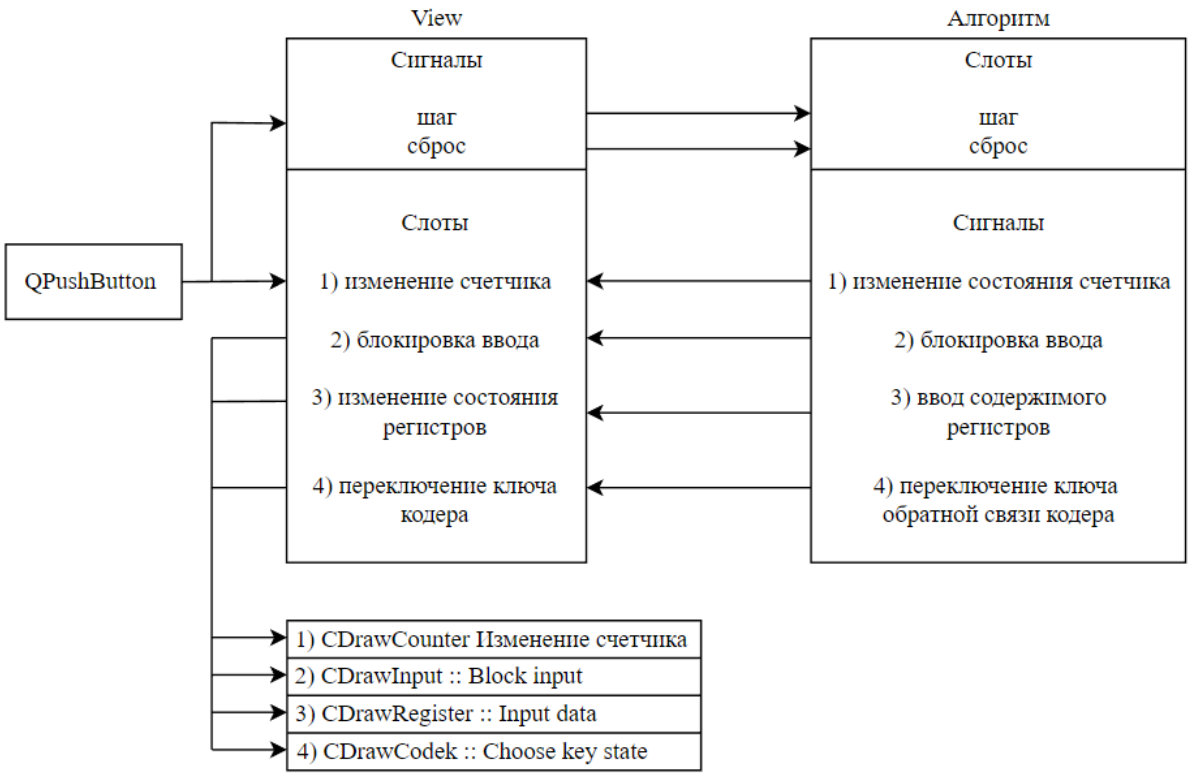

Рис. 4.3. Механизм сигналов и слотов

Два класса приложения используют передачу данных при помощи механизма сигналов и слотов – это класс CMainWindow и CAlgoryhm. Привязка сигналов к соответствующим слотам происходит в конструкторе класса CMainWindow. Класс CMainWindow содержит в себе поля-экземпляры класса CAlgorythm и CMainView, последний из которых содержит в себе указатели на экземпляры класса CDrawRegister, которые отвечают за визуализацию регистров сдвига, в результате чего требуется привязка CAlgorythm к CMainView. Класс CMainView содержит два сигнала – перехода на следующий шаг и сброса всех регистров в начальное состояние. Данные сигналы напрямую связаны с сигналами класса QPushButton, который обеспечивает создания элемента графического интерфейса известный, как «кнопка». Эти сигналы привязаны к соответствующим слотам класса CAlgorythm. По нажатию кнопки «следующий шаг» соответствует переходу к следующему такту работы кодера или декодера, в зависимости от текущего состояния.

Класс CAlgorythm имеет следующие слоты. Слоты 1-2 (см. рисунок 4.3) отвечают за блокирование возможности ввода информации пользователем в определенный регистр, изменения состояния счетчика тактов, а также для переключения индикаторов состояния. Блокировка необходима, например, после того, как запущен процесс кодирования, в этом случае происходит блокировка вводного регистра, а также буферного регистра. Также этот слот осуществляет разблокирование указанного регистра. Например, выполняется разблокировка буферного регистра после кодирования для обеспечения возможности ввода пользователем ошибки в сообщение. Слоты группы 3 отвечают за передачу содержимого регистров для вывода. Система визуализации регистров реализована таким образом, что в класс-регистр передается только информация о выводимой информации, формирование состояния регистра выполняется классом CAlgorythm. Слот 4 необходим для переключения ключа обратной связи кодера.

Класс CMainView, как было отмечено выше, содержит указатели на объекты классов группы CDrawRegister. Класс CDrawRegister имеет виртуальный метод запроса и установки данных, вызов которого происходит в соответствующем слоте класса CMainView.

4.1. Реализация циклического кода

Устройство систематического кодирования и декодер Мегитта реализованы в виде класса CAlgorythm, который является наследником от QObject, наследование необходимо для обеспечения работы механизма сигнал-слот. Регистры сдвига реализованы при помощи std::bitset. Такая коллекция позволяет осуществить операцию доступа по индексу к каждому разряду, а также выполнять логические операции и операции сдвига. Аналогичными возможностями обладает коллекция QBitArray фреймворка Qt, но при этом такая коллекция не имеет встроенной возможности логического сдвига на заданное число разрядов. Также в виде коллекции bitset хранится

маска, предназначенная для введения обратной связи в регистрах-делителях. Маска хранит в себе двоичное представление порождающего полинома без старшей степени.

Конструктор данного класса осуществляет инициализацию масок обратных связей регистров делителей кодера и декодера, а также запускает формирование селектора синдромов. При этом содержимое селектора заполняется в конструкторе класса, путем чтения соответствующего файла. Файл, содержащий векторы ошибок и их синдромы был сформирован однократно на этапе разработки программного обеспечения. По этой причине формирование таблицы было реализовано при помощи «тривиального» алгоритма, в основе которого лежит перебор 16 разрядных чисел без знака. В процессе перебора происходит отбор подходящих чисел по двум критериям – наличие старшей единицы, а также необходимого веса. Данные числа далее используются как основа для вектора ошибки, каждое число переводится в bitset, после чего запускается работа штатного регистра-делителя декодера для получения синдрома. Наличие старшей единицы – особенность работы декодера Мегитта, а необходимый вес – это кратность ошибки.

Программное обеспечение использует БЧХ-полином $x^{10}+x^8+x^5+x^4+x^2+x+1$, который порождает код (15,5) и обеспечивает кодовое расстояние равное 7, что позволяет корректировать любые случайные ошибки кратностью до трех [3].

Система кодирования, декодирования и коррекции выполнена в виде конечного автомата, текущее состояние сохраняется при помощи переменной-перечисления. Помимо текущего состояния необходимо хранить информацию о текущем такте – это необходимо для изменения состояния ключа в кодере, отвечающего за выключение обратной связи, переход к следующему режиму (состоянию), а также останов процедуры коррекции.

Класс CAlgorythm имеет слот, привязанный к кнопке «Следующий шаг», после того как слот вызван, осуществляется проверка, в каком состоянии сейчас находится алгоритм. В случае, если алгоритм находится в режиме кодирования, то производится проверка на нулевой такт – исходное состояние алгоритма. В этом случае происходит сброс коллекций bitset, при помощи которых реализована работа регистров сдвига. Кроме этого, данный слот принимает два указателя типа void (такой тип обусловлен тем, что не требуется регистрация типа для обеспечения работы механизма сигнал-слот). Первый указатель передает содержимое вводного регистра, а второй содержимое буферного регистра, который хранит закодированное сообщение. В случае нулевого такта происходит копирование содержимого вводного регистра в соответствующую переменную класса CAlgorythm. Далее осуществляется сдвиг буферного регистра на один разряд, это необходимо для подготовки младшего разряда, в который будет записан старший разряд кодируемого слова. После чего, в зависимости от номера такта (фактически состояния ключа в обратной связи кодера), осуществляется либо такт деления, при котором в регистре делителе формируется фрагмент остатка от деления и осуществляется запись соответствующего разряда кодируемого сообщения в буферный регистр, либо такт передачи остатка от деления в буферный регистр.

При выполнении такта деления (ключ закрыт, обратная связь включена), в систему визуализации передается сигнал о изменении состояния входного регистра (на каждом очередном такте происходит его сдвиг на один разряд). В случае, если ключ обратной связи разомкнут, то старший разряд регистра – делителя записывается в младший разряд буферного регистра, после чего осуществляется сдвиг регистра-делителя на один разряд. Также осуществляется передача сигнала в систему визуализации о состоянии ключа и буферного регистра. После того, как выполнены первые k тактов, после которых в регистре делителе содержится остаток от деления, а в буферном регистре в младшем разряде содержится старший разряд кодируемого

сообщения, осуществляется передача сигнала в систему визуализации об «отключении» вводного регистра и переводе его в неактивное состояние. Далее осуществляется передача номера такта для обновления значения счетчика, при этом номер такта преобразуется в bitset.

После того, как выполнены операции, соответствующие такту работы кодера, осуществляется проверка номера такта для определения необходимости перехода в следующее состояние. В случае, если номер такта равен 15 (это означает, что процесс кодирования завершен), то осуществляется передача сигнала в систему визуализации для изменения состояния на «декодирование», кроме этого, происходит инициализация значения отдельного счетчика тактов декодера значением 15.

Декодирование также занимает 15 тактов, после чего регистр-делитель содержит синдром ошибки. При вызове слота «следующий шаг» в режиме декодирования, при 15 такте осуществляется сброс регистра-делителя, и осуществляется запись буферного регистра в буферный регистр декодера. Это необходимо на случай, если пользователь после осуществления кодирования ввел ошибку в закодированное сообщение, для этого, как было отмечено выше, используется указатель, передаваемый в слот. Далее осуществляется очередной такт работы регистра-делителя декодера. Алгоритм выполнения очередного такта работы регистра-делителя зависит от счетчика тактов декодера, это обусловлено тем, что первые 15 тактов осуществляется деление и формирование синдрома, а вторые 15 тактов производится коррекция. При коррекции необходимо осуществлять сдвиг синдрома по модулю порождающего полинома, что осуществляется благодаря работе цепи обратной связи и нулевыми входными данными. При аппаратной реализации декодера Мегитта это достигается тем, что буферный регистр приемника обнуляется после первых n тактов работы. При программной реализации в данном случае, используется единый буферный регистр для хранения закодированного и декодируемого сообщения, в результате чего используется

условный оператор для определения необходимости ввода нулевого значения в регистр-делитель.

Содержимое регистра-делителя зависит от состояния обратной связи, в случае если старший разряд регистра-делителя ненулевой, то в соответствии с конфигурацией обратной связи, которая задается порождающим полиномом, осуществляется добавление единицы к соответствующим ячейкам регистра-делителя. При программной реализации используется условный оператор, сравнивающий старший разряд регистра с нулем, в случае если значение ненулевое, то происходит поразрядное суммирование bitset, реализующего регистр-делитель и маски, задающей порождающий полином, как показано на рисунке 4.4.

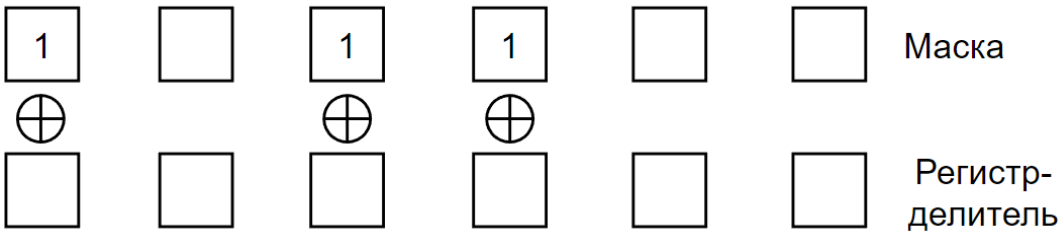

Рис. 4.4. Введение обратной связи в регистре делителе

Также следует отметить, что в процессе работы декодера, при формировании синдрома, регистр-делитель может в себе содержать один из селектируемых синдромов, что может привести к ошибочной коррекции, в результате чего введен защитный механизм, запрещающий работу селектора на тактах декодирования.

После выполнения 15 тактов работы декодера формируется остаток от деления – синдром, а буферный регистр содержит сообщение, готовое к коррекции. Алгоритм переходит в режим коррекции. В этом случае происходит очередной такт работы регистра-делителя, но в этом случае уже происходит сдвиг синдрома по модулю порождающего полинома.

Селектор синдромов реализован в виде коллекции QMap [4], которая имеет возможность оперативного поиска по ключу. В данном случае синдром является ключом, факт наличия ключа определяется при помощи метода коллекции contains. Селектор хранит синдромы в виде 16 разрядных чисел без знака по причине того, что коллекция bitset не имеет определенных операторов сравнения. В случае, если синдром в селекторе найден, то возможно внесение коррекции в младший разряд буферного регистра, устанавливается соответствующий флаг, и осуществляется передача сигнала в систему визуализации для оповещения пользователя о вносимой на данном такте коррекции. Далее осуществляется циклический сдвиг буферного регистра, при котором в соответствии с флагом коррекции изменяется старший разряд сообщения.

4.2. Визуализация и ввод данных

Визуализация регистров сдвига, при помощи которых реализованы алгоритмы кодирования и декодирования, осуществляется при помощи круговых индикаторов, которые могут иметь четыре цвета – красный, зеленый, синий и серый. Зеленый и синий цвета используется для индикации исходного сообщения. Зеленый цвет соответствует нулю двоичного кода, а синий единице. Красный цвет предназначен для отображения искаженного разряда, а серый цвет обозначает неактивное состояние данного индикатора. Неактивное состояние предназначено для запрета ввода данных в соответствующий регистр. Такая необходимость возникает, например, в процессе кодирования, на случай если между тактами работы кодера не были внесены изменения в кодируемое сообщение, что приведет к неверной интерпретации получаемого результата. Аналогичным образом, при помощи круговых индикаторов осуществляется визуализация текущего состояния – кодирование, декодирование или коррекция. Кроме этого, круговыми индикаторами в двоичном коде отображается текущий номер такта. Ввод

данных в определенную ячейку регистра сдвига осуществляется путем нажатия мышью на соответствующий индикатор. Нажатие мыши изменяет состояние соответствующей ячейки.

Система визуализации реализуется при помощи двух классов. Первый CIndicatorNode является наследником от QGraphicsPixmapItem, и предназначен для реализации индикаторов. Базовый класс QGraphicsPixmapItem является классом, предоставляющим растровое изображение. Класс QGraphicsPixmapItem унаследован от класса QGraphicsItem. [5]

Наследование от QGraphicsPixmapItem позволяет задать вид индикатора в виде внешнего изображения. Конструктор данного класса имеет параметр, при помощи которого определяется местоположение индикатора на сцене, точка задается в виде объекта типа QPointF. Также данный класс имеет метод, возвращающий описывающий прямоугольник и текущий размер индикатора. Описывающий прямоугольник задается исходя из размеров загружаемого изображения и масштабного коэффициента. Регистр в общем случае может содержать произвольное количество элементов, что приводит к произвольному количеству экземпляров класса-наследника от QGraphicsPixmapItem. В результате чего, каждый регистр представлен в виде экземпляра класса CDrawRegister, который является наследником от коллекции QList и наследника от QGraphicsItem. Наследование от коллекции-списка, предназначено для упрощения хранения индикаторов, а наследование от QGraphicsItem предназначено для возможности формирования графического представления обратных связей, сумматоров и ключа. Класс CDrawRegister является основой для визуализации регистров и счетчика и является базовым. Базовый класс хранит информацию об описывающем прямоугольнике, а также входных и выходных точках регистров, которые хранятся в координатах сцены. Это необходимо для визуализации связей между регистрами. Базовый класс также хранит информацию о типе регистра,

а также флаг блокировки возможности ввода пользователем данных в регистр. Также класс имеет метод, позволяющий снять или установить блокировку для всех его элементов. Конструктор базового класса обеспечивает создание нужного количества экземпляров класса CIndicatorNode, а также устанавливает индикаторы на сцену при помощи метода moveBy, реализованного в базовом классе QGraphicsPixmapItem. При этом координата положения каждого индикатора рассчитывается исходя из точки положения первого индикатора и шага между ними, данные параметры передаются в конструктор класса CDrawRegister. После того, как индикатор создан, определено его положение, указатель на вновь созданный индикатор записывается в список (класс CDrawRegister за счет наследования является списком). Также в конструкторе фиксируется тип регистра.

Кроме этого, базовый класс имеет виртуальные методы, предназначенные для установки и чтения данных из регистров. Необходимость формирования виртуальных методов обусловлена применением bitset для реализации регистров. При этом bitset является шаблонным классом, параметров которого задается количество разрядов в bitset. В результате чего возникает необходимость передачи данных с различным количеством разрядов. Виртуальные методы в качестве параметров имеют указатели типа void*, а приведение к нужному типу обеспечивается переопределенными методами в наследниках.

На основе базового класса CDrawRegister формируются классы, предназначенные для визуализации регистров. Класс CDrawCoder предназначен для визуализации кодера, его конструктор принимает следующие параметры: количество ячеек регистра делителя (фактически количество индикаторов CIndicatorNode), шаг между индикаторами и начальная точка первого индикатора, при вызове конструктора эти параметры передаются в конструктор базового класса, также в конструктор базового класса передается константа, означающая тип регистра. Кодер содержит в себе ключ управления обратной связью, в результате чего, в класс добавлен флаг

состояния ключа, изначально ключ установлен в положение, соответствующее замкнутой обратной связи. Также данный класс имеет метод изменения состояния ключа. В конструкторе класса осуществляется определение местоположения сумматоров, положения линии обратной связи, а также границ, в пределах которых осуществляется визуализация кодера. Положение сумматоров определяется исходя из расстояния между индикаторами, величина которого передается в конструктор класса CDrawRegister. Рассмотрим схематичное представление определения местоположения сумматора при помощи рисунка 4.5.

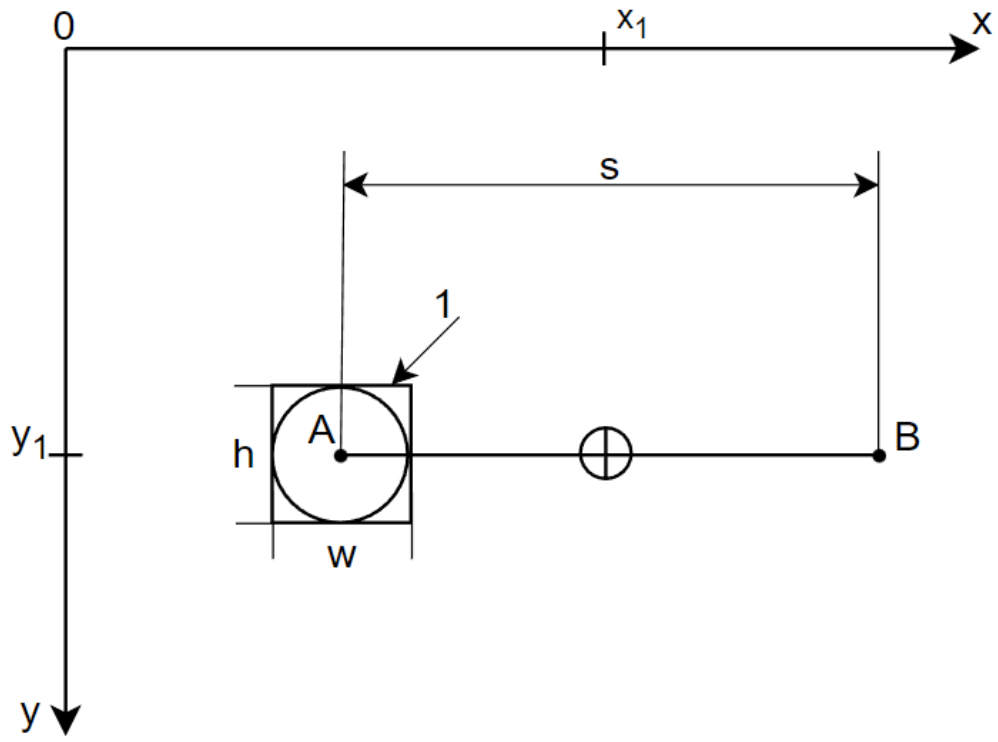

Рис. 4.5. Определение местоположения сумматора. 1 – индикатор для которого определяется положение сумматора, A – центр индикатора 1, B – точка расположения следующего индикатора, w,h – ширина и высота описывающего прямоугольника, x1, y1 – координаты сумматора, s – шаг между индикаторами

Класс CDrawRegister является наследником от класса коллекции QList и хранит указатели на объекты класса CIndicatorNode, которые в свою очередь предназначены для индикации состояния ячеек регистра сдвига. Положение сумматоров определяется при помощи цикла, который осуществляет обход

списка. При этом, сумматоры между ячейками регистра-делителя располагаются исходя из структуры порождающего полинома кода. В результате чего, в процессе обхода определяется необходимость наличия сумматора «справа» от индикатора. Если соответствующая ненулевая степень порождающего полинома присутствует, то осуществляется расчет положения сумматора. Пусть индикатор 1 показанный на рисунке 4.5 требует формирования сумматора. Точка A определяется при помощи метода pos() класса CIndicatorNode, а положение точки B определяется исходя из шага s между индикаторами. Горизонтальная координата x1 сумматора определяется исходя из горизонтальной координаты точки A и расстояния между точками A и B по горизонтальной координате. Вертикальная координата y1 определяется исходя из координаты точки A. Размер сумматора задается константным. Полученные координаты сумматоров сохраняются в списке. Положение линий обратной связи определяется исходя из размера и положения первого индикатора регистра сдвига, как показано на рисунке 4.6

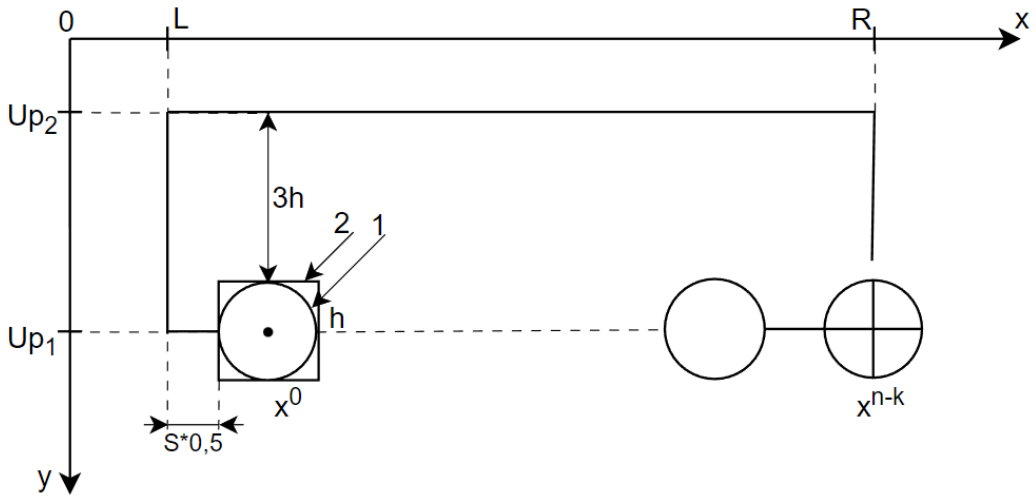

Рис. 4.6. Определение положения линии обратной связи регистра-делителя кодера. x^0 – индикатор младшего разряда, x^(n-k) - старшего, 1 – индикатор, 2 – описывающий прямоугольник объекта-индикатора, h – высота описывающего прямоугольника, Left, Right – границы проведения линии обратной связи по горизонтальной координате, Up1,2 – границы по вертикальной координате

Вертикальная граница (Up2) определяется исходя из центральной точки индикатора степени x^0, которая задается на этапе создания индикатора и высоты описывающего индикатор прямоугольника, а координата Up1 определяется исходя из вертикальной координаты центральной точки индикатора. Граница слева (Left) по горизонтальной координате вычисляется исходя из координат точки центра младшего индикатора, а граница (Right) равна сумматору при старшей степени порождающего полинома и определена ранее.

Циклические коды принято описывать в полиномиальном виде, при этом каждый разряд регистров сдвига строго соответствует элементу полинома, в регистрах делителях это остаток от деления, а в буферном регистре сообщение. В результате этого требуется вывод информации о соответствии степени полинома и разряда регистра сдвига. Для этого используется информация о точках положения сумматоров, а также информация о промежуточных точках между ячейками регистров, где суммирование не производится. Размер шрифта определяется исходя из задаваемого масштабного коэффициента.

Положение ключа определяется исходя из середины правой вертикальной линии обратной связи. Положение элемента логического «или» определяется исходя из координат линии обратной связи, а его размер определяется исходя из шага между сумматорами (для упрощения). Как было отмечено ранее, каждый объект CDrawRegister имеет метод, возвращающий координаты «точки входа» и «точки выхода», это необходимо для визуального представления взаимосвязей между элементами. Формирование линий соединения вводного регистра, кодера, буферного регистра и декодера осуществляется в конструкторе класса CMainView, непосредственным вызовом метода вывода линий графической сцены. Такими точками для кодера является точка ввода и вывода данных на логическом «или».

4.3. Блок главного окна

Главное окно приложения реализовано в виде класса CMainView наследника от класса QGraphicsView. Класс QGraphicsView предоставляет виджет для отображения содержимого QGraphicsScene. QGraphicsView отображает содержимое QGraphicsScene в прокручиваемой области. [5] Экземпляр данного класса расположен в классе CMainWindow, который в свою очередь является наследником от QMainWindow, данный класс содержит в себе центральный виджет, на котором располагаются все виджеты, из которых состоит графический интерфейс приложения. Таких виджетов в общем виде три – кнопки QPushButton сброса и следующего шага, а также сцены «обернутой» в QGraphicsView, который представлен экземпляром класса CMainView. При этом используется менеджер-компоновщик QVBoxLayout, который располагает виджеты по вертикали, как показано на рисунке 4.7.

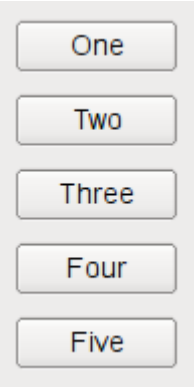

Рис. 4.7. Пример работы менеджера компоновки

В первую очередь в компоновщик добавляется «сцена», после чего происходит добавление кнопок, после чего создается промежуточный виджет, для которого будет назначен данный компоновщик, а далее промежуточный виджет будет установлен как «центральный» для главного окна приложения. Данные операции выполняются в конструкторе класса главного окна, также, в данном конструкторе, как было отмечено ранее происходит привязка сигналов и слотов.

Рассмотрим алгоритм размещения элементов на сцене. Графические элементы создаются в конструкторе класса CMainView, осуществляется передача таких параметров, как количество ячеек в регистрах сдвига, начальное положение регистров. Начальное положение и шаг рассчитываются исходя из размера окна, что позволяет корректно отображать окно приложения после запуска вне зависимости от установленного разрешения экрана. После того, как был осуществлен вызов конструкторов всех графических элементов, осуществляется привязка элементов к сцене. Сцена является приватным компонентом класса CMainView. На сцене необходимо разместить ячейки-индикаторы, а также линии обратной связи, сумматоры и другие элементы. Индикаторы и «линии» принадлежат двум отдельным классам. При этом индикаторы хранятся в классе-регистре (CDrawRegister), который в свою очередь является наследником от списка, в результате чего возможно осуществление обхода этого списка непосредственно после его создания. Таким образом, осуществляется обход, в ходе которого происходит добавление индикаторов на сцену (их положение было определено в конструкторе класса CDrawRegister), после чего происходит добавление экземпляра CDrawRegister на сцену. Как было отмечено ранее, каждый класс-регистр обладает возможностью запроса данных о точках на сцене, которые соответствуют точкам входа и выхода схемы. На основе этих данных происходит добавление взаимосвязей между элементами, путем обращения непосредственно к графической сцене – осуществляется вызов метода добавления линии на сцену.

После того, как все графические элементы добавлены на сцену, происходит «регистрация» сцены на виджете CMainView.