ГОСУДАРСТВЕННОЕ УНИТАРНОЕ ПРЕДПРИЯТИЕ
НАУЧНО-ПРОИЗВОДСТВЕННЫЙ ЦЕНТР “ЭЛВИС”

Инструментальное программное обеспечение MF01

Алгоритмичекая модель MF01.

Руководство пользователя

Версия 1.0

9 апреля 2006 г.

Оглавление

Введение.

Основным назначением описанной в данном документе библиотеки классов является построение функционального аналога тракта обработки данных СБИС четырехканального цифрового приемника 1288ХК1Т (MF01). Алгоритмическая модель позволяет посредством компьютерного моделирования оценить реакцию устройства на входное воздействие. Соответствие структур модели и устройства позволяет оценить получаемые параметры без сборки физического устройства. Возможности реализации фильтровых трактов проиллюстрированы набором примеров.

1. Структура модели приемного тракта

Библиотека модели написана на языке C++ и предназначена для моделирования работы построенного устройства и тестирования возможностей реализуемых на базе данного устройства алгоритмов обработки сигналов. Библиотека построена с использованием парадигмы объектно-ориентированного программирования. Для удобства использования библиотека предоставляется пользователю в виде скомпилированной статической библиотеки и набора необходимых для ее включения заголовочных файлов.

Каждый формируемый экземпляр класса MF01_MODEL_ реализует функционально законченный элемент, соответствующий одному каналу приемного тракта MF01 по назначению, выполняемым функциям и управлению, в соответствии с определенными в спецификациях требованиями.

Все используемые в библиотеке классы-модели блоков являются производными от базового класса module_, обуславливающего своей структурой идеологию построения и использования данного программного продукта. Один из вариантов включения модели предполагает использование данного класса в качестве базового для реализуемых пользователем источников и приемников сигнала и использования т.о. единой концепции построения модели на основе конкатенации модулей.

На Рис. 1 в общем виде показана структура модели. Как видно из рисунка, количество сохраняемой в процессе тестирования информации о работе приемного тракта может быть различным и определяется сделанными пользователем настройками (штрих пунктирной линией показаны точки модели доступные для записи временных выборок сигнала).

Возможны два разных метода инициализации модели:

- Первый режим: вызов функции Setup () и передача этой функции в качестве параметра имени файла-источника данных (файлов инициализации) и имени секции в этом файле с параметрами; этот вариант показан на рисунке в виде обобщенного “Блока инициализации” без детализации его связей с другими блоками модели.
- Второй режим: используется прямое обращение в модель для установки параметров через соответствующие функции.

Т.о. первый подход позволяет выполнить всю инициализацию автоматически внутри модели, а второй подход обеспечивает прямое управление параметрами, вынося управление процессом инициализации за пределы модели.

С точки зрения загрузки, обработки данных и получения результата фильтрации возможны два режима использования модели:

- Первый режим: связывание объектов через метод Connect() и автоматическая передача данных по тракту “источник сигнала – модель MF – приемник сигнала”
- Второй режим: архитектура окружения модели полностью предоставлена пользователю; для передачи отсчета в модель используется один из вариантов функции Put(). Отфильтрованные отсчеты с выхода модели накапливаются в выходном FIFO и хранятся там до момента извлечения их пользователем. Для их получения используется один из вариантов функции Get().

Выдача результатов с выхода модели организована на двух альтернативной основе. Если к выходу модели MF подключены блоки-приемники сигнала, то выдача отсчетов производится автоматически. Если таких блоков нет (не обнаружены моделью MF), то отсчеты накапливаются в выходном FIFO.

Методы работы с параметрами инициализации и режимами работы могут комбинироваться в произвольном порядке. Для иллюстрации предоставляемых возможностей далее показаны четыре примера ее использования.

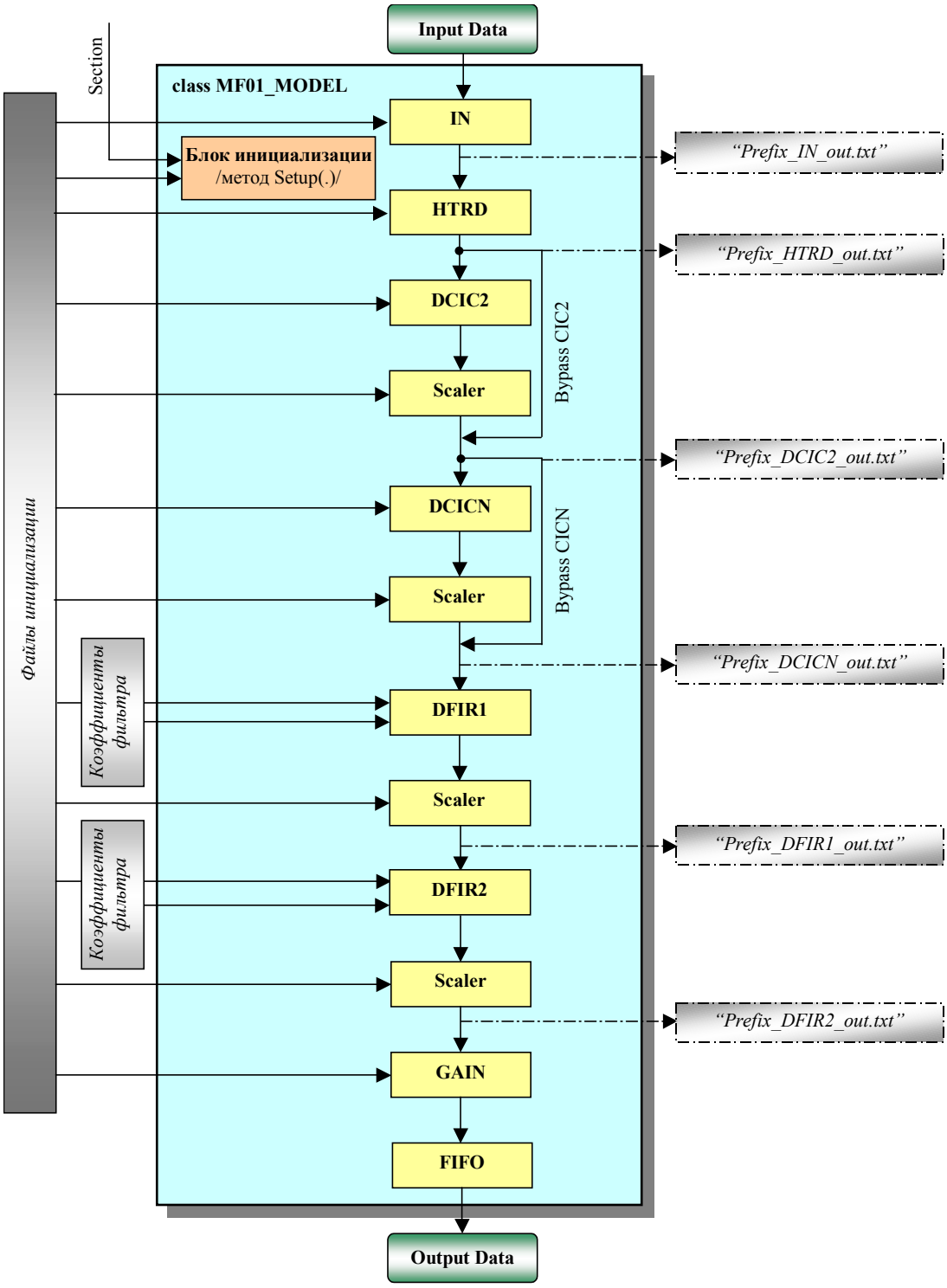

Рис. 1 Блок-схема модели MF.

Далее подробно рассматриваются все доступные пользователю классы и функции данной библиотеки.

1.1. Состав модели

Модель предоставляется в виде готовой к использованию библиотеки и необходимого для ее включения набора заголовочных файлов. В состав модели под Windows входят следующие файлы:

- err.h
- initools.h
- MF_C_Model_defines.h
- mf01_model.h
- mf01_model.lib
- module.h
- packet.h

В Unix версии модели используются следующие файлы:

- err.h
- initools.h
- MF_C_Model_defines.h
- mf01_model.h
- mf01_model.o
- module.h
- packet.h

2. Пакеты данных

При передаче данных между всеми производными от базового класса module_ объектами (блоками устройства) используется указатель на тип данных packet_, который может быть ассоциирован с одной из производных от этого базового класса структур. Основные типы пакетов описаны в “MF_C_Model_defines.h”. Непосредственно для передачи данных в модель и получения их из нее используются соответственно, ***in_sample_*** и ***out_sample_***. Их функции и поля данных следующие:

- ***struct in_sample_***

  public:

  - in_t I, Q; - две компоненты входного комплексного сигнала
  - template<class T>
  - in_sample_(const T i, const T q) – инициализация I и Q значениями i и q соответственно
  - in_sample_() – обнуление I и Q.

- ***struct out_sample_***

  public:

  - out_t I, Q; - две компоненты выходного комплексного сигнала
  - template<class T>;
  - out_sample_(const T i, const T q) – инициализация I и Q значениями i и q соответственно
  - out_sample_() – обнуление I и Q.

3. Базовые классы

Базовыми классами модели в целом являются классы module_ и err_. Класс module_ объявляет основные интерфейсные функции для создания межблочных соединений. При построении модели на основе связывания объектов через метод Connect() рекомендуется так же использовать данный класс в качестве базового для разрабатываемых блоков источника и приемника сигнала. Этот подход проиллюстрирован в примере 3. Ниже описаны основные функции данного базового класса.

3.1. Конструктор

- module_(string name) : public err_(name) – принимает в качестве параметра имя экземпляра класса, которое передает в базовый класс; инициирует начальные значения переменных.

3.2. Функции

public:

- virtual void Connect(module_ &m); – присоединяет к выходу данного модуля модуль &m, если он до этого не был присоединен; контролирует однократность присоединения этого модуля.
- virtual void Unconnect(module_ &m); – отсоединяет присоединенный к выходу данного модуля модуль &m, если он был присоединен; контролирует однократность отсоединения данного модуля.
- virtual void Put(packet_ &p); – вызов функции загружает отсчет внутрь модуля.
- virtual void Clear() – функция очистки модели и сброса сделанных установок;

protected:

- void Out(packet_ &); – выдает отчет на выход экземпляра класса; при этом присоединенные к данному модулю при помощи Connect() модули автоматически получают этот отсчет.

4. class MF01_MODEL_

Интегрирует все классы приемного тракта и реализует экземпляр приемного тракта в целом. Предназначен для общего управления настройкой работы всех узлов MF01, а так же блоков записи сигналов для получения сигналов из промежуточных точек, и объединения всех их в единый тракт.

4.1. Создание и удаление экземпляра класса

Создание и удаление экземпляра класса MF01_MODEL_ выполняется при помощи двух функций, объявленных в файле mf.h

- MF01_model_* Create_MF01_model(string name); - функция вызывает конструктор класса MF01_MODEL_, создавая т.о. экземпляр класса, и возвращает указатель на него. “name” – символьная константа–идентификатор экземпляра класса. Параметр наследуется от базового класса и используется аналогично во всех производных от module_ классах.
- void Create_MF01_model(MF01_model_* t); - удаляет экземпляр класса по предаваемому указателю.

4.2. Функции

Каждая из приведенных ниже функций переопределяет аналогичную функцию базового класса. В режиме комплексного гетеродина (in_type=2) для обработки сигнала используются два канала MF, сигналы которых объединяются после DCICN. Используемый при этом дополнительный канал и его блоки обозначаются через добавление метки “Q” в название блоков: HTRD_Q, DCIC2_Q и DCICN_Q. Каждый из этих каналов программируется отдельно и может иметь отличные от другого канала настройки (теоретически, без учета целесообразности таких настроек). Для установки параметров дополнительного канала используются аналогичные основному каналу функции, так же содержащие в названии метку “Q”, например:

- void set_DCIC2_dr (int dr); - установка коэффициента децимации основного канала
- void set_DCIC2_Q_dr (int dr); - установка коэффициента децимации дополнительного канала

В приведенном ниже описании функций раскрыто назначение функций только основного канала. Полный перечень функций содержится в файле mf01_model.h.

public:

- void Connect(module_ &m); – формирует выход устройства, как выход блока FIFO.
- void Unconnect(module_ &m); – отсоединяет присоединенные к выходу устройства модули.
- void Setup (string file, string section); - загружает параметры модели из секции “section” в файле “file”.

Интерфейсные функции:

- void Put (packet_ &p); – принимает входной сигнал в виде пакета *in_sample_*.
- void Put (int i) – принимает входной сигнал в виде реального целочисленного значения;
- void Put (int i, int q) – принимает входной сигнал в виде двух отсчетов комплексного числа;
- void Put (complex<int> r) – принимает входной отсчет в виде комплексного числа;

- int Get (out_sample_ &s); – извлекает очередной отсчет из FIFO в виде пакета *out_sample_*. При удачном чтении возвращает “1”, в противном случае - “0”
- int Get (complex<int>&s); – извлекает очередной отсчет из FIFO в виде комплексного числа. При удачном чтении возвращает “1”, в противном случае - “0”
- int Get (int& i, int& q); – извлекает очередной отсчет из FIFO в виде двух целочисленных отсчетов. При удачном чтении возвращает “1”, в противном случае - “0”
- int num_available(); – возвращает количество хранимых в FIFO отсчетов.
- unsigned num_in_samples(); - выдает количество поступивших на вход модели отсчетов
- unsigned num_out_samples(); - выдает количество выданных на выход отсчетов.
- void set_in_fmt (int in_fmt); - устанавливает формат поступающих на вход модели данных (см. RX_CFG)
- void set_in_type (int in_type); - устанавливает тип входных данных (см. RX_CFG)
- void set_out_fmt(int out_fmt); - устанавливает формат выходных данных (см. RX_CFG).

Функции установки параметров гетеродина:

- void set_pdith_en (int pdith_en); - включение/выключение дизеринга гетеродина
- void set_NCO_FRQ (frq_t frq); - установка значения частоты
- void set_NCO_PHASE (phase_t ph); - установка начального сдвига фазы
- void set_HTRD (int pdith_en, frq_t nco_frq, phase_t nco_phase); - установка всех параметров гетеродина одновременно

Функции установки параметров DCIC2 (см. RX*_DCIC2):

- void set_DCIC2_mode (int mode); - включение/выключение каскада децимации
- void set_DCIC2_dr (int dr); - установка коэффициента децимации
- void set_DCIC2_scl (int scl); - установка параметров скалера на выходе DCIC2
- void set_DCIC2 (int mode, int dr, int scl); - установка всех параметров DCIC2 одновременно

Функции установки параметров DCICN (см. RX*_DCICN):

- void set_DCICN_mode (int mode); - настройка каскада децимации
- void set_DCICN_dr (int dr); - установка коэффициента децимации
- void set_DCICN_scl_mx (int scl_mx); - переключение мультиплексора на выходе DCICN
- void set_DCICN_scl (int scl); - установка параметров скалера на выходе DCICN
- void set_DCICN (int mode, int dr, int scl_mx, int scl); - установка всех параметров DCICN одновременно

Функции установки параметров DFIR1 (см. RX*_DFIR1_CFG1 и RX*_DFIR1_CFG2):

- void set_DFIR1_sym (int sym); - определяет симметричность или антисимметричность коэффициентов фильтров
- void set_DFIR1_order (int order); - установка порядка фильтра
- void set_DFIR1_dr (int dr); - установка коэффициента децимации
- void set_DFIR1_scl (int scl);); - установка параметров скалера на выходе DFIR1
- void set_DFIR1_dly (int dly); - установка задержки синхронизации

- void load_DFIR1_from_file (string fileCoef); - загрузка коэффициентов фильтров из файла “fileCoef”
- void set_DFIR1_CFG (int sym, int order, int dr, int scl, int dly); - установка всех параметров DFIR1 одновременно
- void set_DFIR1_coeff (int idx, int coeff); - прямое управление загрузкой коэффициентов фильтров; записывает в коэффициент с номером idx значение coeff.

Функции установки параметров DFIR2 (см. RX*_DFIR2_CFG1 и RX*_DFIR2_CFG2):

- void set_DFIR2_sym (int sym); - определяет симметричность или антисимметричность коэффициентов фильтров
- void set_DFIR2_order (int order); - установка порядка фильтра
- void set_DFIR2_dr (int dr); - установка коэффициента децимации
- void set_DFIR2_scl (int scl);); - установка параметров скалера на выходе DFIR2
- void set_DFIR2_dly (int dly); - установка задержки синхронизации
- void load_DFIR2_from_file (string fileCoef); - загрузка коэффициентов фильтров из файла “fileCoef”
- void set_DFIR2_CFG (int sym, int order, int dr, int scl, int dly); - установка всех параметров DFIR2 одновременно
- void set_DFIR2_coeff (int idx, int coeff); - прямое управление загрузкой коэффициентов фильтров; записывает в коэффициент с номером idx значение coeff.

Функции установки параметров FineGain (см. RX*_GAIN_I и RX*_GAIN_Q):

- void set_GAIN_I (int I); - установка реальной части множителя
- void set_GAIN_Q (int Q); - установка мнимой части множителя
- void set_GAIN (int I, int Q); - установка комплексного коэффициента целиком

Функции управления записью данных из промежуточных точек в файл:

- void set_dump_file_prefix(string str); - установка префикса, добавляемого к имени записываемого файла (см. Рис. 1).
- void set_dump(int bitmask, int num); “bitmask” - битовая маска; каждый бит разрешает/запрещает запись данных для одной из точек модели (единица в младшем бите соответствует записи со входа модели); “num” – количество записываемых в файл отсчетов.

Служебные функции изменения состояния модели:

- void Clear () – функция очистки модели и сброса сделанных установок;
  - Очистка памяти данных и коэффициентов фильтров,
  - Очистка линий задержки в CIC фильтрах
  - Установка счетчиков в исходное состояние
- void Stop () – останов приемного тракта.
  - Очистка памяти данных фильтров,
  - Очистка линий задержки в CIC фильтрах

Информационные функции вероятности перегрузки в каскадах:

- double DCIC2_overflow() – возвращает вероятность перегрузки на выходе DCIC2;
- double DCICN_overflow()– возвращает вероятность перегрузки на выходе DCICN;
- double DFIR1_overflow()– возвращает вероятность перегрузки на выходе DFIR1;
- double DFIR2_overflow()– возвращает вероятность перегрузки на выходе DFIR2.

5. Инициализация параметров. Файлы инициализации.

Параметры блоков в модели устанавливались в соответствии со спецификациями на блоки и параметры MF01. Как уже отмечалось выше, загрузка параметров извне может выполняться двумя способами:

- через файлы инициализации;
- вызовом специальных функций для прямой установки конкретного параметра или группы параметров.

Рассмотрим список устанавливаемых параметров на примере инициализации параметров для стандарта IS95. Ниже показан вариант реализации файлов, использованный в примерах.

5.1. Файл rx_test.ini

Файл содержит настройки для тестирования в заданном пользователем режиме. В примере показан один режим, определяемый настройками, включенными в секцию [Section]. Для загрузки параметров используется следующий вызов функции: Setup (“rx_test.ini”, “Section”). В процессе работы модель открывает файл “rx_test.ini”, находит в нем секцию “Section”, анализирует включенные в эту секцию файлы и устанавливает все необходимые параметры. В случае отсутствия параметра выдается сообщение об ошибке, указывающее какой параметр не инициализирован (не найден).

5.2. Файл IS95_shell.ini

В файле IS95_sell.ini собраны настройки, не относящиеся напрямую к установкам параметров модели по спецификациям. Отдельные параметры могут переопределяться в соответствии с целями тестирования в файле rx_test.ini. Необходимо отметить, что деление настроек по файлам достаточно условное. Как будет показано далее в примерах, принципиальным является название секции и метод ее включения в головной файл.

[WRITE_DATA] – стандартное имя секции, управляющей записью параметров из промежуточных точек модели

write_length = 1024 – количество записываемых точек

IN = 0 – вкл./выкл. записи сигнала со входа устройства (вход гетеродина)

HTRD = 0 – вкл./выкл. записи сигнала с выхода гетеродина

DCIC2 = 0 – вкл./выкл. записи сигнала с выхода CIC2

DCICN = 0 – вкл./выкл. записи сигнала с выхода CICN

DFIR1 = 0 – вкл./выкл. записи сигнала с выхода DFIR1

DFIR2 = 0 – вкл./выкл. записи сигнала с выхода DFIR2

5.3. Файл IS95_rx.ini

В файле IS95_rx.ini собраны типовые настройки приемного тракта, доступные пользователю согласно спецификациям. Ниже приведен примерный вид такого файла, соответствующий условиям работы MF применительно к стандарту IS95.

Как было отмечено выше в режиме комплексного гетеродина используется дополнительный канал, блоки которого обозначаются через добавление метки “Q” в название блоков. Программирование этих блоков производится при помощи секций с аналогичными названиями.

[RX] – стандартные настройки (см. описание RX_CFG)

mode = 0 - режим работы (0..3)

in_type = 0 - тип входных данных (0..7)

in_fmt = 0 - формат входных данных (0..3)

out_fmt = 0 - формат выходных данных (0..3)

[HTRD]

pdith_en=1 – включение и выключение дизеринга (соответственно “1” и “0”)

nco_frq=3492690148 – код генерируемой частоты при заданной частоте дискретизации и выбранной промежуточной частоте

nco_phase=0 – начальный сдвиг фазы

[HTRD_Q] – настройки Q канала

pdith_en=1

nco_frq=3492690148

nco_phase=0

[CIC2]

mode=0 – ключ включения и выключения (соответственно “1” и “0”) каскада децимации на базе фильтра CIC2;

scl=0 – код сигнала, подаваемого на масштабатор на выходе каскада

dr=0 – коэффициент децимации (реальный коэффициент децимации - 1)

[CIC2_Q] – настройки Q канала

mode=0

scl=0

dr=0

[CICN]

mode=3 – ключ включения и выключения (соответственно “1” и “0”) каскада децимации на базе фильтра CICN и задания режима: принимает 4-ре значения:

- "0" – фильтр выключен;
- "1" – режим CIC4;
- "2" – режим CIC5;
- "3" – режим CIC6;

dr=11 – коэффициент децимации (реальный коэффициент децимации - 1)

scl=3 – код управления масштабатором на выходе каскада (Scaler).

scl_mx=1 – является переключателем мультиплексора на выходе каскада: старшие/младшие разряды (1-MSB 0-LSB).

[CICN] – настройки Q канала

mode=3

dr=11

scl=3

scl_mx=1

[DFIR1]

file_coef=dfir_cf.dat – файл с коэффициентами КИХ фильтра-дециматора

oder=15 – порядок фильтра (количество коэффициентов - 1)

sym=0 – антисимметричный или симметричный (соотв. “1” и “0”)

dr=1 – коэффициент децимации (реальный коэффициент децимации - 1)

scl=5 – код управления масштабатором на выходе каскада (коэффициент усиления)

[DFIR2]

file_coef= ch_is95r.dat – файл с коэффициентами канального фильтра

sym=0

dr=0 – децимация выключена (реальный коэффициент децимации - 1)

oder=52

scl=8

[GAIN] – имя секции, содержащей коэффициенты для точной амплитудной и фазовой регулировки выходного сигнала тракта

I=16384 – реальная часть коэффициента

Q=0 – мнимая часть коэффициента

6. Примеры использования

6.1. Пример 1

В данном примере показан один из наиболее простых вариантов включения модели. Инициализация модели выполняется функцией Setup(). Входное воздействие представляет собой выборку шумового сигнала с нормальным распределением, записанную в файле .../distr/example/data/noise.dat. Фильтруемые данные читаются из файла и предаются в модель посредством вызова функции Put(int). Отфильтрованные данные сохраняются в FIFO. По окончании чтения файла, содержащего входное воздействие, данные из FIFO считываются и записываются в выходной файл. Здесь необходимо подчеркнуть, что данное FIFO ни в коей мере не являются моделью FIFO на выходе реального устройства. Оно организовано как бесконечная очередь в которой накапливаются и хранятся отсчеты до тех пор, пока они не будут извлечены пользователем.

Необходимые для включения первого примера использования файлы собраны в папке .../distr/example/Example1. Для установки параметров экземпляра класса используется файл rx_tests.ini, который, в свою очередь, включает файлы инициализации, расположенные в папке …/distr/include/. В каждом из каталогов примеров есть небольшая программа на Matlab, позволяющая посмотреть спектр входного сигнала и полученного в результате работы модели отфильтрованного сигнала. При построении спектра отфильтрованного сигнала часть начальных отсчетов полученного вектора отбрасываются для исключения из рассмотрения переходного процесса.

Ниже приведен листинг файла rx_main.cpp с необходимыми комментариями.

```
#include "../../lib/mf01_model.h"
#include <fstream>

void main(){

  //создаем экземпляр класса и инициализируем его параметры
  MF01_model_ &Rx = *Create_MF01_model("RX");
  Rx.Clear();
  Rx.Setup("rx_tests.ini","Section");

  int i, q, sample=0;
  ifstream infile;
  ofstream outfile;

  //открываем файл с входным воздействием и читаем, пока не будет
  //достигнут конец файла
  infile.open("../data/noise.dat");
  do{
    infile>>sample;
    Rx.Put(sample);
  }while (!infile.eof());
  infile.close();

  //открываем файл и записываем выходной сигнал, до тех пор,
  //пока не будут прочитаны все сохраненные в FIFO отсчеты
  outfile.open("output.dat");
  do{
    Rx.Get(i, q);
    outfile<<i<<" "<<q<<endl;
  }while (Rx.num_available()>0);
  outfile.close();
```

```
  //удаление экземпляра класса
  Delete_MF01_model(&Rx);
};
```

6.2. Пример 2

Как отмечалось раньше, все управление работой модели может быть выполнено пользователем в удобной для него форме. Этот пример показывает как загружать отдельные параметры или группы параметров в модель. Для инициализации модели могут быть использованы любые из доступных функций в произвольном порядке. В качестве иллюстрации этого коэффициенты DFIR1 загружены прямым заданием коэффициентов в модели, а коэффициенты DFIR2 загружены из файла через вызов соответствующей функции.

Для формирования входного воздействия использован генератор случайных чисел. Выходной сигнал также сохраняется в виде файла. Если в первом примере запись в файл осуществлялась после приема всего входного вектора сигнала, то в данном примере отсчеты сохраняются по мере их появления на выходе устройства.

Ниже приведен листинг файла rx_main.cpp с необходимыми комментариями. Необходимые файлы собраны в папке .../distr/example/Example2.

```
#include "../../lib/mf01_model.h"
#include <fstream>

//используемый в модели экземпляр класса
MF01_model_ &Rx = *Create_MF01_model("RX");

void Setup (){ //функция инициализации параметров
  Rx.set_dump_file_prefix("Test_");//устанавливаем префикс имени
                        //записываемых из промежут. точек файлов
  Rx.set_dump(1, 1024); //управление записью данных из промежуточных
                        //точек модели
                        //будет записано 1024 отсчетов с входа модели
                        //в файл "Test_RX_PATH_IN_out.txt"

  Rx.set_in_fmt(0);

  //Установка параметров гетеродина
  Rx.set_HTRD(1, 3492690148, 0);
  Rx.set_in_type(0);

  // Установка параметров CIC2
  Rx.set_DCIC2(0, 0, 0);

  // Установка параметров CICN
  Rx.set_DCICN(3, 11, 1, 3);

  // Установка параметров DFIR1
  int dfir_cf[]={194, 913, 1167, -1609, -5981, -2558, 14493, 32767};
  int i;

  Rx.set_DFIR1_CFG(0, 15, 1, 5, 0);
  for (i=0; i<8; i++) Rx.set_DFIR1_coeff(i, dfir_cf[i]);

  // Установка параметров DFIR2
  Rx.set_DFIR2_CFG(0, 52, 0, 6, 0);
  Rx.load_DFIR2_from_file("../../include/ch_is95r.dat");

  Rx.set_GAIN (16384, 0);
```

```
  Rx.set_out_fmt(0);
};

void main(){

  int i, q, count=0, sample=0;

  Rx.Clear();
  Setup();

  ofstream outfile;
  outfile.open("output.dat");

  do{
    do{//генерируем отсчеты и проверяем, что они лежат в интервале
       //допустимых входных значений
      sample=32767*(1-2.0*rand()/RAND_MAX);
    } while(sample>32767||sample<-32767);

    Rx.Put(sample);//загрузка отсчета в модель

    if(Rx.num_available()>0){//если есть отфильтрованный отсчет, то
                             //записываем его в файл
      if(Rx.Get(i, q)==1)   outfile<<i<<" "<<q<<endl;
      count++;
    }
  }while(count<1250);//проверяем количество принятых отсчетов

  outfile.close();
};
```

6.3. Пример 3

В данном примере показан вариант построения модели с использованием классов, производных от базового класса module_. Оболочка среды моделирования строится с использованием классов Source и Check, раскрытых ниже. На приведенном ниже Рис. 2 в общем виде показана блок схема соединения классов для запуска тестов в этом режиме.

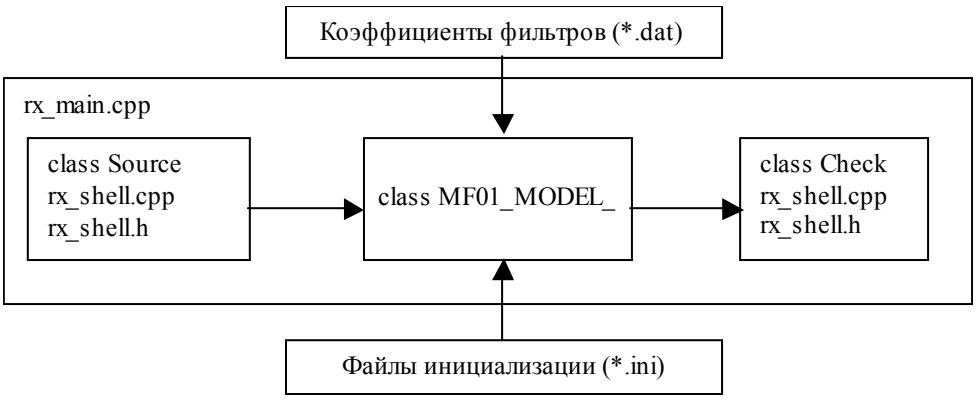

Рис. 2 Блок-схема модели

Как видно из рисунка, файл rx_main.cpp объединяет модель тестирования в единое целое. Кроме того, он подключает файлы инициализации, определяет в каком порядке и в каком объеме будет выполняться тестирование, а также непосредственно инициирует выполнение тестов.

Необходимые для включения третьего примера использования файлы собраны в папке .../distr/example/Example 3.

Инициализации модели, так же как и в первом примере, выполняется путем вызова функции Setup (string filename, string section). Входное воздействие создается источником сигнала – экземпляром класса Source, который считывает сигнал из файла и подает его на вход MF (использован тот же файл с входным воздействием, что и в случае первого примера).

Приемник сигнала, – экземпляр класса Check, - принимает выходную последовательность и записывает результат обработки в файл.

В отличии от примера 1 здесь был использован другой подход к написанию файла инициализации. В примере 1 для добавления параметров в секцию [Section] был использована директива ***#source*** в следующем виде:

#source "../../include/IS95_shell.ini"

В данном примере для установки тех же параметров добавлена новая секция:

[Section.WRITE_DATA]

Эти два подхода абсолютно тождественны по своему действию, поскольку директива ***#source*** как раз указывает модели что включаемая из файла секция [WRITE_DATA] должна рассматриваться как упомянутая секция [Section.WRITE_DATA].

Ниже приведены листинги файлов rx_main.cpp, rx_shell.cpp и rx_shell.h с необходимыми комментариями.

```
#include "../../lib/mf01_model.h"
#include "rx_shell.h"
#include <fstream>

void main(){

  //создаем экземпляры классов
  MF01_model_ &Rx = *Create_MF01_model("RX");
  Rx.Clear();
  Rx.Setup("rx_tests.ini","Section");
  Source &Sr =  *new Source("Sr");
  Check &Chk = *new Check("Chk");

  int i, q, sample=0;
  //устанавливаем соединения между объектами
  Sr.Connect(Rx);
  Rx.Connect(Chk);

  do{
    Sr.Put(sample); //вызов функции приводит к передаче одного
                    //отсчета на вход MF
  }while (Chk.count<1250);//подсчет количества полученных
                          //на выходе отсчетов

  Delete_MF01_model(&Rx);
  delete &Chk;
  delete &Sr;
};

-------------------------------------------------------------------------

#ifndef __RX_SHELL_H__
#define __RX_SHELL_H__
```

```
#include "../../lib/MF_C_Model_defines.h"
#include "../../lib/module.h"
#include <fstream>

class Source : public module_
{
public:
  Source(string moduleName);
  ~Source();
  void Put(int input=0);
  ifstream file;
};

class Check : public module_
{
public:
  Check(string moduleName);
  ~Check();
  ofstream file;
  void Put(packet_ &pack);//переопределенная функция базового класса
  int count;
};

#endif

--------------------------------------------------------------------------
#include "rx_shell.h"

Source::Source(string name) : module_(name){
  file.open("../data/noise.dat");
}

Source::~Source(){
  if(file.is_open()) file.close();
}

void Source::Put(int y){
  int in;
  if(!file.eof())
    file>>in;

  in_sample_ out_p; //тип передаваемого в модель пакета in_sample_
  out_p.I=in;
  out_p.Q=0;
  Out(out_p);//вызов функции Out() приводит к автоматической передаче
             //пакета всем присоединенным к источнику сигнала модулям
}

//////////////////////////////////////////////////////////////////////////

Check::Check(string name) : module_(name){
  file.open("output.dat");
  count=0;
}

Check::~Check(){
  if(file.is_open()) file.close();
}

void Check::Put(packet_ &pack){//функция Put() автоматически
                               //принимает из модели пакеты
  out_sample_ &in_p=(out_sample_ &) pack; //тип получаемого из модели
```

```
                                                    //пакета out_sample_
    file<<in_p.I<<" "<<in_p.Q<<endl;
    count++;
  }
```

6.4. Пример 4

В данном примере показан запуск тестирования с командной строки. Формат команды для вызова модели выглядит следующим образом:

Example4.exe input_file ini_file output_file,

где “input_file” – файл с входным воздействием,

“ini_file” – файл с параметрами инициализации,

“output_file” – имя файла в который следует записать выходной сигнал.

Запуск программы осуществляется посредством вызова файла run.bat, расположенного в каталоге примера 4. В остальном данный пример подобен примеру 1 и не требует подробных комментариев.

Ниже приведен листинг файла rx_main.cpp с.

```
#include "../../lib/mf01_model.h"
#include <fstream>

void main(int argc, char **argv){

  MF01_model_ &Rx = *Create_MF01_model("RX");
  Rx.Clear();

  Rx.Setup(argv[2],"Section");

  int i, q, sample=0;
  ifstream infile;
  ofstream outfile;

  infile.open(argv[1]);
  outfile.open(argv[3]);

  for(;;){
    if (infile.eof())break;
    infile>>sample;
    Rx.Put(sample);
    if(Rx.num_available()>0){
      Rx.Get(i, q);
      outfile<<i<<" "<<q<<endl;
    }
  }

  infile.close();
  outfile.close();
  Delete_MF01_model(&Rx);
};
```