МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2021 г.

Измерение параметров сигнала и воспроизведение видео T2-MI потоков в мультисистемных анализаторах ТВ сигналов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____В. А. Парасич
«___»_____2021 г.

Автор работы,
студент группы КЭ-405
_____Р. А. Матушкин
«___»_____2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____С. В. Сяськов
«___»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2021 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Матушкину Родиону Андреевичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Измерение параметров сигнала и воспроизведение видео T2-MI потоков в мультисистемных анализаторах ТВ сигналов» утверждена приказом по университету от 26 апреля 2021 г. №714-13/12

2. **Срок сдачи студентом законченной работы:** 1 июня 2021 г.

3. **Исходные данные к работе:**

   Обеспечить основной функционал программного обеспечения:

   – воспроизвидение T2-MI потока;

   – получение параметров T2-MI потока.

4. **Перечень подлежащих разработке вопросов:**

   – анализ аналогов, выделение основных проблем;

   – формирование и согласование требований к программному обеспечению;

– проектирование программного обеспечения;

– реализация программного обеспечения;

– тестирование программного обеспечения для подтверждения корректности функционирования реализованного продукта;

– интеграция программного обеспечения в релизную прошивку прибора.

5. **Дата выдачи задания:** 1 февраля 2021 г.

Руководитель работы _____/*В. А. Парасич*/

Студент _____/*Р. А. Матушкин*/

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение	05.02.2021	
Обзор литературы	12.02.2021	
Разработка модели, проектирование	19.03.2021	
Реализация системы	26.03.2021	
Тестирование	20.04.2021	
Оформление	01.05.2021	
Подготовка презентации	25.05.2021	

Руководитель работы _____/*В. А. Парасич*/

Студент _____/*Р. А. Матушкин*/

АННОТАЦИЯ

Р. А. Матушкин. Измерение параметров сигнала и воспроизведение видео T2-MI потоков в мультисистемных анализаторах ТВ сигналов. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 132- с., 29 ил., библиогр. список – 16 наим.

В рамках выпускной квалификационной работы разрабатывается программное обеспечение для измерения параметров сигнала и воспроизведения видео T2-MI потоков. Программное обеспечение должно функционировать на анализаторе ТВ сигналов IT-100.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С развитием технологий на замену аналоговому сигналу пришел цифровой сигнал. Когда мы говорим о сигнале, то обычно подразумеваем электромагнитные колебания, наводящие ЭДС и вызывающие колебания тока в антенне приемника. Получившиеся электромагнитные колебания, или волны, распространяются в пространстве с помощью передаточной антенны. Чтобы эфир не забивался низкочастотными помехами и у разных радиостанций была возможность работать параллельно, не мешая друг другу, колебания, получившиеся от воздействия звука, суммируют, то есть «накладывают» на другие колебания, имеющие постоянную частоту. Последнюю частоту принято называть «несущей», и именно на ее восприятие мы настраиваем свой радиоприемник, чтобы «поймать» аналоговый сигнал радиостанции.

В процессе передачи сигнала от радиостанции к приемнику может произойти всякое. Могут возникнуть сторонние помехи, частота и амплитуда могут измениться. Наконец, и сами передатчик и приемник во время преобразования сигнала вносят некоторую погрешность. Поэтому сигнал, воспроизводимый аналоговым приемником, всегда имеет искажения. Голос может вполне воспроизводиться, несмотря на изменения, но фоном будет шипение или даже какие-то хрипы, вызванные помехами, так же будут видны искажения в изображении. Чем менее уверенным будет прием, тем отчетливее будут эти посторонние шумовые эффекты. Вдобавок эфирный аналоговый сигнал имеет очень слабую степень защиты от постороннего доступа.

Цифровая связь и вещание считаются более защищенными от помех и от внешних воздействий. Все дело в том, что при его использовании аналоговый сигнал на передающей станции зашифровывается в цифровой код. Звуку определенной частоты и громкости присваивается код из радиоимпульсов.

Продолжительность и частота импульсов задана заранее – она одна и у передатчика, и у приемника. Наличие импульса соответствует единице, отсутствие – нулю.

Устройство, преобразующее аналоговый сигнал в цифровой код, называется аналого-цифровым преобразователем (АЦП). А устройство, установленное в приемнике, и преобразующее код в аналоговый сигнал, называется цифро-аналоговым преобразователем (ЦАП).

Во время передачи цифрового сигнала ошибки и искажения практически исключены. Если импульс станет немного сильнее, продолжительнее, или наоборот, то он все равно будет распознан системой как единица. А нуль останется нулем, даже если на его месте возникнет какой-то случайный слабый сигнал. Для АЦП и ЦАП не существует других значений, как 0,2 или 0,9 – только нуль и единица. Благодаря этому помехи на цифровую связь и вещание почти не оказывают влияния.

Более того, цифровой сигнал является и более защищенным от постороннего доступа. Ведь, чтобы ЦАП устройства смог расшифровать его, необходимо, чтобы он знал код расшифровки. АЦП вместе с сигналом может передавать и цифровой адрес устройства, выбранного в качестве приемника. Таким образом, даже если радиосигнал и будет перехвачен, он не сможет быть распознан из-за отсутствия как минимум части кода [1].

Информация, переносимая в цифровых сигналах телевизионного вещания, представляет собой транспортный поток (протокол для передач данных) группы стандартов MPEG-2. Благодаря данному протоколу стало возможно объединение аудио- и видеоданных, их синхронизация, сжатие и многое другое.

Для эфирного вещания потоков были созданы стандарты – DVB (Digital Video Broadcasting), регламентирующие обработку сигнала перед его трансляцией. Этот стандарт разделен на группы, и каждая группа отвечает за

свою сферу применения. Так существуют стандарты DVB-T2 и DVB-T, которые отвечают за эфирные вещания, DVB-S и DVB-S2 – спутниковое вещание и т.д.

Также данные стандарты описывают возможность передачи сразу нескольких транспортных потоков (протоколов передачи) MPEG. Это нужно, например, при необходимости передачи нескольких транспортных потоков MPEG на большие расстояния (что удобно сделать одним сигналом, чтобы диапазон частот, передающих сигналы, был как можно свободнее). Так в стандарте DVB-T2 для этого используется T2-MI поток, инкапсулированный в транспортный поток MPEG, далее такой инкапсулированный протокол будем называть T2-MI over MPEG.

Схема передачи телевизионных данных в России будет выглядеть следующим образом (рисунок 1).

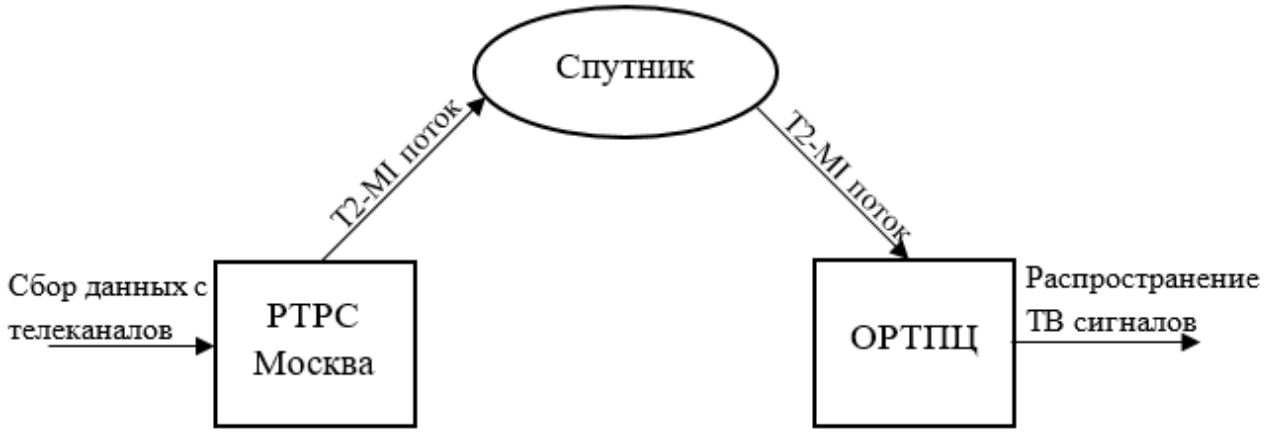

Рисунок 1 - Общая схема передачи телевизионных данных

По рисунку 1 видно, что в Московском филиале РТРС (Российская телевизионная и радиовещательная сеть) данные, полученные от различных телеканалов, преобразуются в сигнал T2-MI over MPEG (на схеме T2-MI поток, о нем поговорим позднее), удобный для передачи на спутник. Затем эти данные получают областные радиотелевизионные передающие центры (ОРТПЦ),

которые в дальнейшем преобразуют их в сигнал другого формата (для его распространения по области), например, в сигнал для наземного транслирования.

Формы, в которых сигнал приходит на РТРС и в которых он транслируется из ОРТПЦ, нам не интересны.

Телевизионные операторы, которым необходимы эти данные со спутника, должны обратиться к своему областному радиотелевизионному передающему центру для их получения. Но не у всех операторов есть возможность получить их от ретрансляционного модуля (который находиться в ОРТПЦ) из-за различных причин, например, из-за сильной территориальной удаленности, и им нужно принимать T2-MI поток напрямую со спутника, откуда вытекает необходимость анализа полученного потока на различные ошибки. Данная ситуация схематично представлена на рисунке 2.

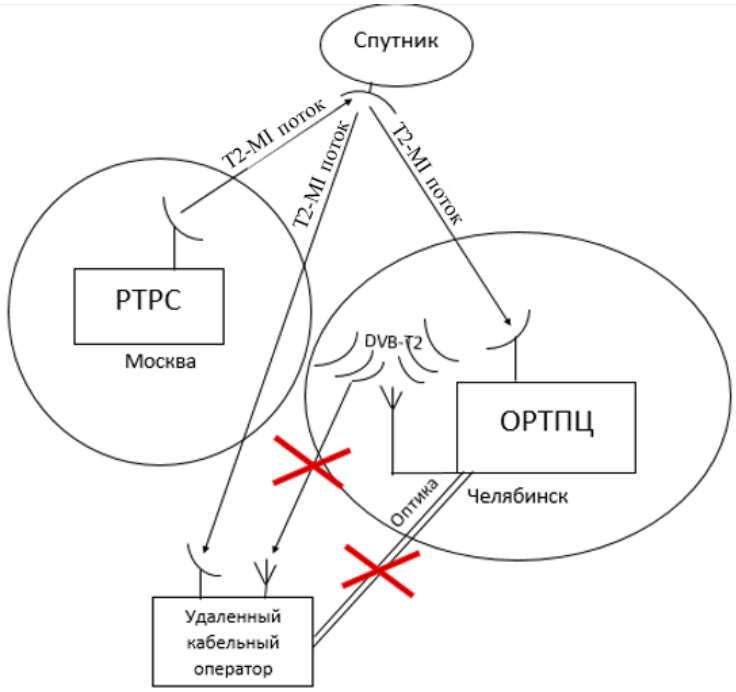

Рисунок 2 - Схема передачи телесигналов в России

Таким образом, с развитием технологий на замену аналоговому сигналу пришло цифровое телевидение. Цифровое телевидение более качественно передает изображение, избавляя его от различных помех. Это стало доступно за

счёт внедрения принципиально новых, цифровых технологий, методов формирования и передачи сигналов ТВ программ, основанных в первую очередь на прогрессе в области цифрового сжатия. Это сжатие основано на стандартах семейства MPEG. Для передачи такого транспортного потока на большие расстояния используют спутник, на него отправляют сигнал формата T2-MI over MPEG (T2-MI поток). Существуют операторы, которым необходимо получать данные (T2-MI поток в оболочке DVB-S2) прямо со спутника. Т.к. они сильно удалены от областного радиотелевизионного передающего центра и не могут получить транспортный поток по оптическому кабелю или принять цифровой сигнал DVB-T2, у этих операторов есть необходимость в анализе полученного ими T2-MI потока.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОБЗОР АНАЛОГОВ

На всех этапах цифрового вещания необходимо следить за состоянием сигнала, для этого используются различные анализаторы.

Анализатор – это измерительное оборудование или прибор, предназначенный для измерения параметров цифровых и аналоговых сигналов.

Рассмотрим некоторые приборы и их возможности по обработке T2-MI потока.

Анализатор АТП-1.

Особенности:

- контроль параметров цифрового транспортного потока, сформированного с кодированием MPEG;

- проверка работоспособности цифровых телевизионных передатчиков стандартов DVB-T/T2;

- анализ транспортных потоков, переносящих T2-MI пакеты;

- анализ T2-MI пакетов;

- интегрируется в систему дистанционного мониторинга сетей цифрового вещания [2].

Изображение прибора АТП-1 можно увидеть на рисунке 3.

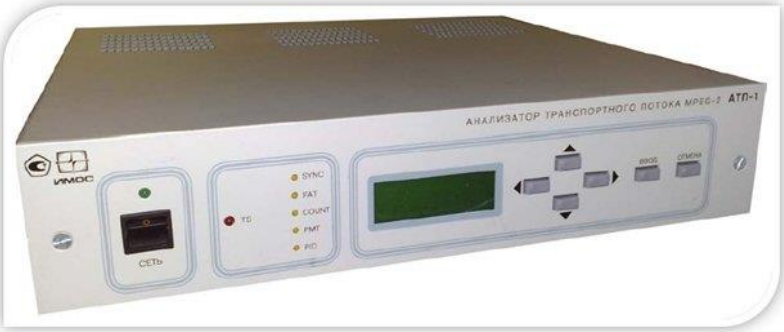

Рисунок 3 - Анализатор АТП-1 [2]

Недостатки:

- стационарность;

- необходимо оборудование для отображения информации полученных прибором;

- отсутствие возможности воспроизведения потоков.

Enensys DiviDual.

Особенности:

- все функции реализованы в одном устройстве: сбор, анализ и воспроизведение сигналов основной полосы частот;

- проверка правильности и анализ DVB T2-MI и MPEG;

- выделение MPEG2-TS с помощью PLP;

- анализ всех параметров T2 (кадр BB, кадр L1 и т. д.);

- компактное устройство с питанием через USB [3].

Изображение прибора Enensys DiviDual можно увидеть на рисунке 4.

Рисунок 4 - Анализатор Enensys DiviDual [3]

Недостатки:

- необходимо оборудование для отображения информации полученных прибором;

- отсутствие возможности воспроизведения потоков.

Ranger neo 4.

Особенности:

- работа с транспортными потоками, которые содержат сигнал T2-MI из различных источников (ASI-TS, IP-TV, файл находящийся в памяти устройства);

- воспроизведение через выход ASI-TS файла, сохраненного в памяти;

- анализ транспортного потока, несущего T2-MI поток;

- анализ и проверка всех полей, определенных в сигнале T2-MI (сигнализация L1, временная метка DVB-T2, каждый из кадров PLP и BB и т. д.).

Изображение прибора Ranger neo 4 можно увидеть на рисунке 5.

Рисунок 5 - Анализатор Ranger neo 4 [4]

Недостаткок – высокая цена.

По существующим на рынке приборам можно сделать вывод, что анализаторы, умеющие работать с T2-MI потоком, требуют дополнительной аппаратуры, как АТП-1 и Evensys Dividual, либо очень дороги, как Ranger Neo 4,

его цена составляет 868 тысяч рублей [4]. Стоит отметить, что двое из них не умеют воспроизводить данные потоки. Поэтому создание своего решения этой проблемы, в котором будут отсутствовать выявленные недостатки, является актуальной задачей.

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

Для решения ранее названной задачи будет использоваться прибор IT-100. Рассмотрим его поподробнее.

IT-100 — это ТВ анализатор, который необходим для измерения сигналов цифрового и аналогового телевидения. С помощью него можно производить настройку кабельных, оптоволоконных или IP сетей приема телевидения и радиовещания.

Он производится в Челябинской области в компании «Планар».

Сейчас рассматриваемый прибор умеет работать со стандартами цифрового телевидения: DVB-C; ITU-T J.83 Annex B, C; DVB-T; DVB-T2; DVB-S; DVB-S2.

Из дополнительных возможностей, связанных с решаемой проблемой, можно выделить следующие:

- анализ MPEG транспортного потока;
- воспроизведение видео из транспортных потоков MPEG [5].

Изображение прибора IT-100 можно увидеть на рисунке 6.

Рисунок 6 - Анализатор IT-100 [5]

В отличие от рассмотренных аналогов, данный прибор, благодаря встроенному видеодекодеру, может воспроизводить видео из транспортных потоков, не требует дополнительного оборудования для своей работы, не дорогой и мобильный. Т.е. в нем отсутствуют все недостатки, выявленные в схожем ему оборудовании.

Но в текущем программном обеспечение IT-100 отсутствуют возможности работы с T2-MI потоком. Добавление функционала для работы с ним будет решением поставленной задачи.

Интерфейс T2-MI всегда инкапсулирован в MPEG поток (т.к. он удобен в передаче), ранее такие инкапсулированные пакеты назывались T2-MI over MPEG.

Рассмотрим структуру транспортного потока MPEG.

Syntax	No. of bits	Mnemonic
transport_packet(){		
sync_byte	**8**	**bslbf**
transport_error_indicator	**1**	**bslbf**
payload_unit_start_indicator	**1**	**bslbf**
transport_priority	**1**	**bslbf**
PID	**13**	**uimsbf**
transport_scrambling_control	**2**	**bslbf**
adaptation_field_control	**2**	**bslbf**
continuity_counter	**4**	**uimsbf**
if(adaptation_field_control == '10' || adaptation_field_control == '11'){		
adaptation_field()		
}		
if(adaptation_field_control == '01' || adaptation_field_control == '11') {		
for (i = 0; i < N; i++){		
data_byte	**8**	**bslbf**
}		
}		
}		

Рисунок 7 – Рекомендации по обработке транспортного потока MPEG [6]

На рисунке 7 видим, что в заголовке пакета находится информация, для корректной его обработки, а после заголовка идут байты данных, в которых могут содержаться данные T2-MI.

Рассмотрим T2-MI поток поподробнее.

Как происходит инкапсуляция можно увидеть на схеме, представленной на рисунке 8.

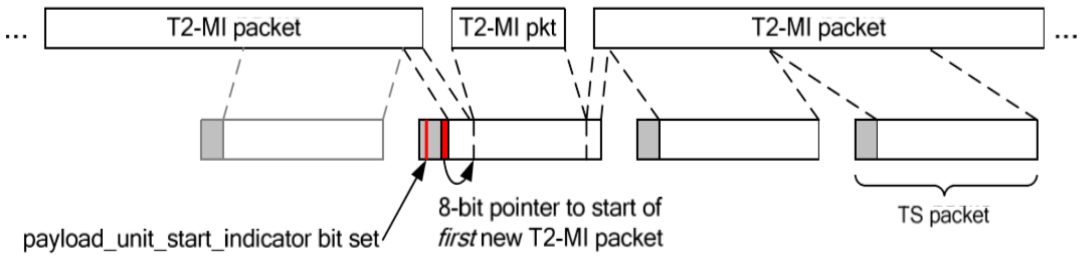

Рисунок 8 - Инкапсуляция T2-MI пакетов в MPEG TS [7, стр. 34]

На рисунке 8 видно, что несколько T2-MI пакетов переносятся друг за другом, при этом они разбиты на несколько частей, хранящихся в пакетах MPEG транспортного потока.

Каждый T2-MI пакет имеет свою структуру – рисунок 9.

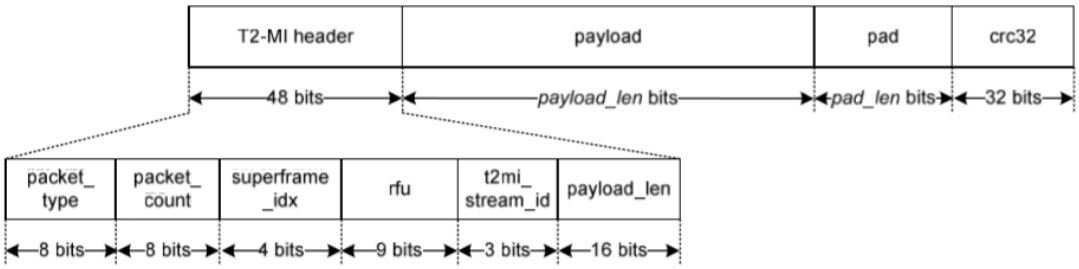

Рисунок 9 - Формат T2-MI пакета [7, стр. 13]

Существует несколько типов пакетов — в одних переносится информация о данных, в других — сами данные, также существуют и другие типы, например, TIMESTAMP (содержит временные метки).

MPEG потоки переносятся в пакетах типа BBF (Baseband Frame). Структура BBF нагрузки – рисунок 10.

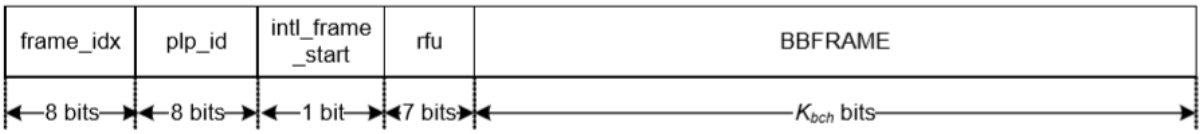

Рисунок 10 - Baseband Frame payload [7, стр. 14]

В T2-MI потоке может быть несколько PLP (Physical Layer Pipe), содержащих свой набор пакетов MPEG со своим списком программ, это как раз-таки необходимо для того, чтобы обеспечить перенос нескольких транспортных потоков MPEG в одном сигнале.

В целом структуру T2-MI можно представить, как “матрёшку”, так как одни потоки находятся внутри других, которые в свою очередь находятся внутри третьих.

Информация о данных, переносящихся в пакетах, находятся в их заголовках, а также в пакетах специального типа. Подробнее о заголовках и пакетах T2-MI можно узнать из стандартов (ETSI TS). На основе этой информации и будет происходить анализ потока, т.к. в заголовках содержится

информация, позволяющая диагностировать какую-либо ошибку и правильно извлекать данные, а в пакетах с информацией находятся различные параметры потока.

Таким образом, необходимо разработать функционал для работы с T2-MI потоком для прибора IT-100.

Главной частью прибора IT-100 является видеодекодер компании Fujitsu - MB86H615, представляющий собой высокоинтегрированный мультистандартный цифровой телевизионный декодер высокой четкости, разработанный для удовлетворения потребностей рынка ТВ-приставок.

В состав декодера входят следующие комплектующие, влияющие на возможность декодирования T2-MI потока:

1) CPU: ARM 1176JZF-S;
2) memory: 16-bit DDR2-800;
3) видео и аудио декодеры транспортного потока MPEG.

Таким образом, видеодекодер умеет работать только с транспортным потоком MPEG. Но он оснащен процессором ARM1176JZF-S, обладающим тактовой частотой 700 МГц. Т.к. даже зная характеристики основных аппаратных составляющих IT-100, тяжело оценить возможность программной обработки T2-MI потока, проведем тесты загруженности CPU при текущей работе.

Для теста нагрузим процессор задачей парсинга (вычленения данных) IP-пакетов, что в целом похоже на задачу парсинга T2-MI пакетов, которую нужно будет реализовать. Будем получать IP-пакеты, содержащие T2-MI поток со скоростью 20 Мбит/с.

В результате были получены следующие сведения по загруженности процессора:

1) поток вычленения информации из IP пакетов – 10,6%;
2) поток получения IP пакетов – 12,03%;

3) бездействие – 45,3%;
4) другие процессы – 31,8%.

Из перечисленного выше видим, что на обработку IP пакетов процессор затрачивает около 12% своей вычислительной мощности, при том, что свободно около 45% этой мощности. Поэтому, даже при воспроизведении потока со скоростью 30 Мбит/с полученным в режиме IP-TV, вычислительной мощности прибора хватит.

Таким образом, декодер умеет работать только с транспортным потоком MPEG, но благодаря своим техническим характеристикам, можно будет анализировать T2-MI поток программно, т.е. за счет вычислительной мощности процессора. Следовательно, решение поставленной проблемы сводится к программированию данного декодера.

Выбор языка программирования:

Для программирования микроконтроллеров можно использовать следующие языки: Assembler, C, MicroPython, MicroBASIC.

Ассемблер – «машинно-ориентированный язык программирования низкого уровня. Представляет собой систему обозначений, используемую для представления в удобно читаемой форме программ, записанных в машинном коде» [8].

Преимущества:

1) позволяет добиться максимального быстродействия;
2) удобен в программировании микроконтроллеров, имеющих ограниченные ресурсы.

Недостатки:

1) громоздкость;
2) трудность восприятия;
3) требуется отличное знание архитектуры и системы команд микроконтроллеров.

C – «компилируемый статически типизированный язык программирования общего назначения» [9].

Преимущества:

1) язык высокого уровня, относительно языка Ассемблер;
2) компиляторы данного языка реализованы почти для всех моделей микроконтроллеров.

Недостатки:

1) слишком низкоуровневый, относительно MicroPython;
2) получаемый программный код конкретной задачи, имеет больший объем, чем код той же задачи, реализованной на Ассемблере.

MicroPython - реализация языка Python, написанная на C и предназначенная для выполнения на микроконтроллерах [10].

Не смотря на возможные положительные качества, главными недостатками MicroPython являются большой объем используемой памяти микроконтроллера и низкое быстродействие кода по сравнению с языками C и Ассемблер. Вследствие этого MicroPython не будет рассматриваться.

MicroBASIC - реализация языка Basic, для программирования микроконтроллеров. Имеет такие же недостатки, как и MicroPython, поэтому он не будет рассматриваться.

Таким образом, будем использовать язык C для разработки программного обеспечения для микроконтроллера. Данный язык более удобен в понимании, чем Ассемблер, программный код, написанный на нем занимает меньше памяти и использует вычислительную мощность процессора более эффективно, чем такой же код на MicroPython. Также язык C имеет множество компиляторов, которые адаптируют программный код под различные микроконтроллеры.

1.3. ВЫВОД

Исследование рынка показало, что все рассмотренные аналоги имеют ряд значительных недостатков. Следовательно, создание нового решения, свободного от выявленных недостатков, является актуальной задачей. Разработка будет представлять собой добавление функционала для работы с T2-MI потоками в прибор IT-100 и будет написано на языке C.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Требования к системе в целом.

1. Наличие возможности работы с потоками из различных источников.
2. Наличие возможности работы с различными типами потоков.
3. Возможность воспроизведения видео из потоков.
4. Возможность получения параметров потока.
5. Наличие возможности выбора PLP для воспроизведения.
6. Наличие возможности выбора программы для воспроизведения.
7. Наличие возможности получения параметров T2-MI.
8. Наличие возможности получения информации о воспроизводимом видео.
9. Наличие возможности получения базовой информации о программах, содержащихся в mpeg потоке.
10. Обеспечение автоматического определения типа воспроизводимого потока.
11. Обеспечение автоматической борьбы с джиттером.
12. Обеспечение отслеживания целостности потока.

Нефункциональные требования к системе в целом.

1. Разработанное ПО должно работать на приборе ИТ-100.
2. Разработанное приложение должно быть написано на языке C.

Подсистемы.

1. Системы источника потока: tsfromfile, tsfrombus, tsfromip.
2. Системы декодирования потока: tsdrcoder, t2midecoder.

3. Системы воспроизведения из определенного источника: fe_player_demod, fe_plaer_file, fe_player_ip.

4. Главная подсистема воспроизведения: fe_plear.

5. Главная система воспроизведения: player.

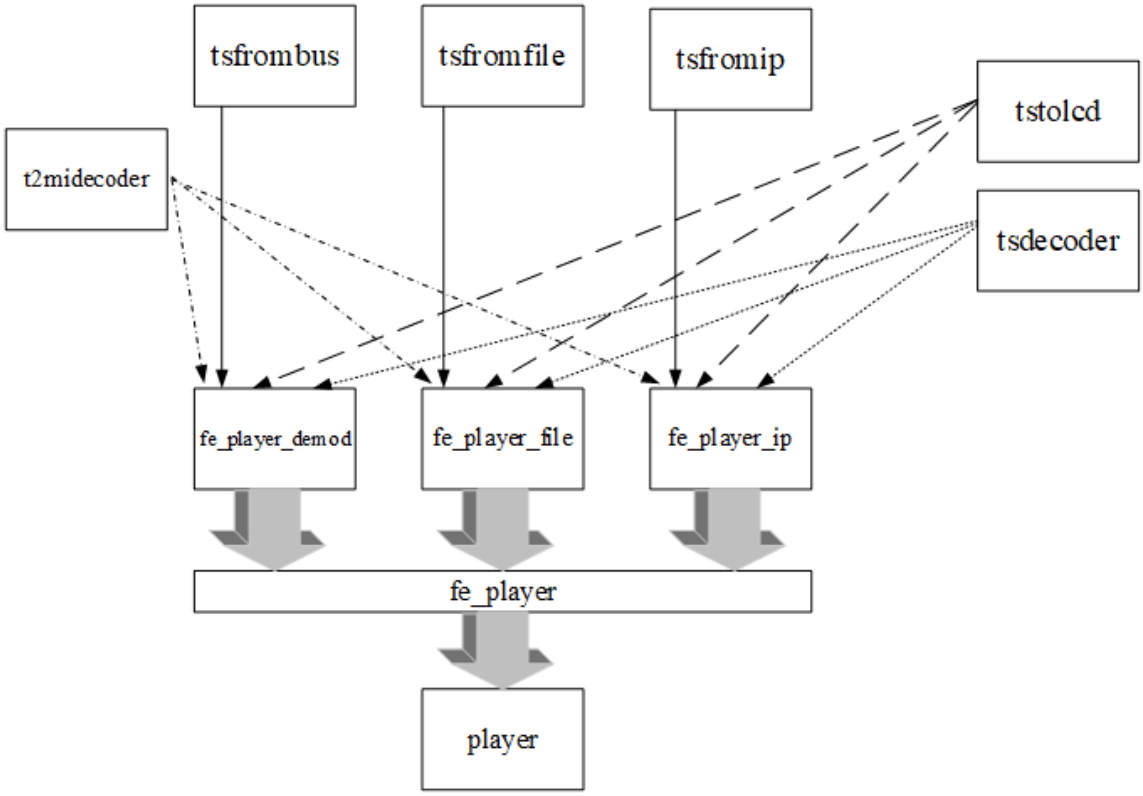

Рисунок 11 – Связи между подсистемами

Функциональные требования к подсистемам.

Подсистеме player.

1. Управление подсистемой воспроизведения видео.
2. Организация интерфейса пользователя.
3. Размещение видео на экране.

Подсистеме fe_player - предоставление абстрактного API, обеспечивающего возможность управлять воспроизведением видео, независимо

от того, из какого источника это видео поступает. По своей сути API является эквивалентом чисто абстрактного класса в C++.

Подсистема fe_player_demod - получение видео от демодуляторов DVB (эфирное ТВ, кабельное ТВ, спутниковое ТВ).

Подсистема fe_player_file - получение видео из файла.

Подсистема fe_player_ip - получение видео из потока IPTV.

Подсистема tsfrombus.

1. Настройка аппаратуры шины.
2. Получение потока с шины по прерыванию.
3. Передача потока приемнику (tsdecoder/t2midecoder).

Подсистема tsfromfile.

1. Захват MPEG-потока из файла.
2. Определение скорости MPEG-потока в файле.
3. Чтение данных из файла с нужной скоростью.
4. Передача потока приемнику (tsdecoder/t2midecoder).

Подсистема tsfromip.

1. Захват MPEG-потока из IPTV-потока.
2. Открытие IPTV канала.
3. Прием UDP и RTP потока.
4. Извлечение MPEG-потока.
5. Передача потока приемнику (tsdecoder/t2midecoder).

Подсистема tsdecoder.

1. Декодирование MPEG потока.
2. Передача видео и звука на отображение.
3. Определение структуры потока (имя сети, набор программ, параметры программ).
4. Выбор программы для воспроизведения.
5. Передача пакетов с PID видео и звука аппаратным декодерам видео и звука.

Подсистема tst2midecoder.

1. Анализ T2MI потока.
2. Передача вычлененного MPEG потока на tsdecoder.
3. Определение структуры потока (количество PLP, информацию о PLP).
4. Выбор PLP для передачи.
5. Вычленение MPEG пакетов из выбранного PLP.

Подсистема tstolcd.

1. Отображение видео на экране.
2. Настройка размера окна для отображения видео и его позиционирование на экране дисплея.

Данной системой пользуются следующий вид пользователей - специалисты по настройке получения сигнала.

Режимы работы, которые должна выполнять система.

1. Воспроизведение T2-MI потока.

2. Воспроизведение mpeg потока.
3. Получение информации о mpeg потоке.
4. Получение информации о T2-MI потоке.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Объем данных: данные отображаются динамично в зависимости от исследуемого потока и хранятся в памяти прибора до завершения работы с ним. Отдельные подсистемы не нужны.

Временные характеристики: пользователь должен видеть то, какая работа выполняется и понимать, что прибор не завис, время загрузок режимов не должно превышать 5 секунд и во время долгих загрузок необходимо отображать ее процесс.

Входные/выходные данные: входные данные – неизвестный видео поток. Выходные данные – видео, полученное из потока, информация о потоке и подпотоках, если имеются.

Обучение: для системы будет разработана инструкция по работе с ней, с помощью нее пользователь должен понимать, как взаимодействовать с прибором и данной системой для получения нужного ему результата.

Документирование: необходимо будет по ходу написания программного кода, наполнять его описательными комментариями, отражающими краткую информацию об его особенностях.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Ключевым элементом рассматриваемого анализатора является декодер MB86H615. Рассматривать подробно устройство анализатора, а именно подключение выводов всех его составляющих не требуется, т.к. в решаемой задаче должна дополняться уже существующая система, путем добавления в нее нового функционала. Проанализируем текущую систему и спроектируем возможные варианты добавления нового функционала.

Рассмотрим плату it100top с целью определить источники транспортного потока.

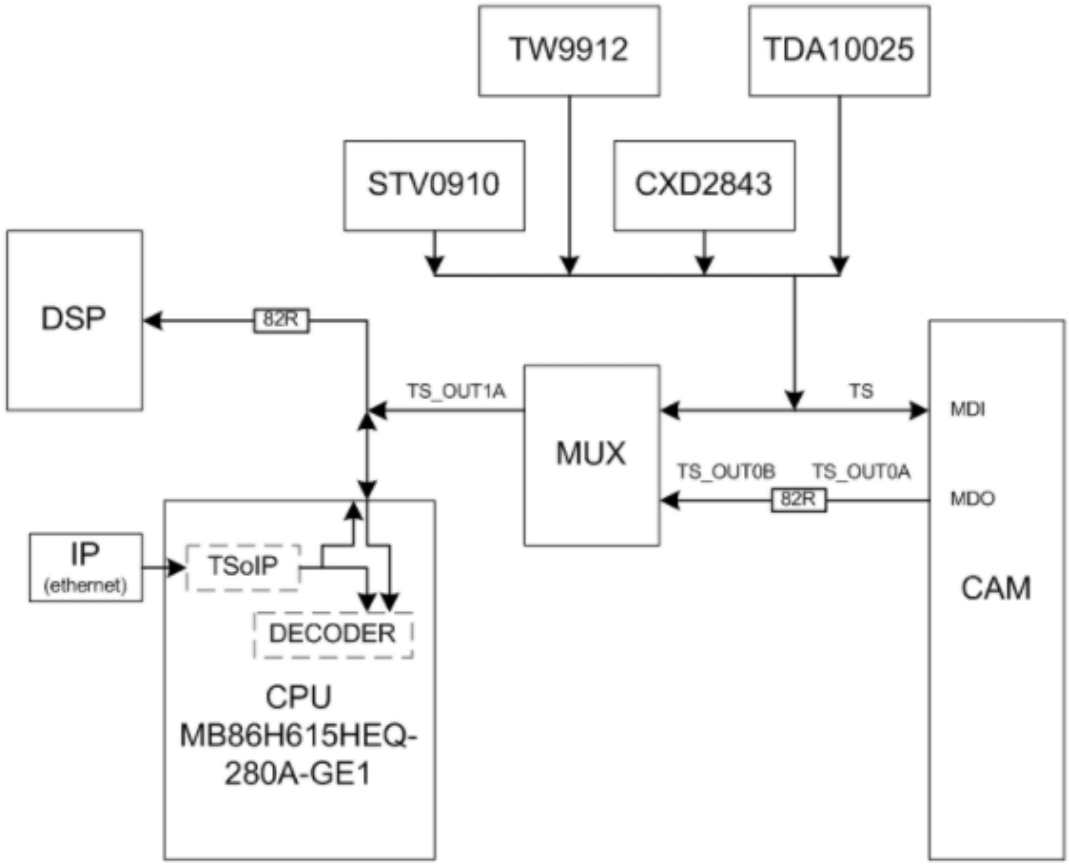

Рисунок 12 – Плата it100top

На рисунке 12 видно, что главный процессор (Fujitsu «MB86H615») подключен к одному из ответвлений шины транспортного потока платы («TS_OUT1A» на рисунке 12). По ней транспортный поток может поступать из нескольких источников:

1) демодулятор «CXD2843»;
2) демодулятор «TDA10025»;
3) демодулятор «STV0910»;
4) декодер «TW9912» (выдает оцифрованный сигнал аналогового ТВ).

Также MB86H615 сам может выступать в роли источника транспортного потока, когда идет прием по IP.

При воспроизведении транспортного потока из файла, данные находящиеся в потоке, подаются на декодер самим CPU (копирование из ПЗУ в буфер декодера).

Таким образом, для работы с потоком из каждого источника существуют такие модули как: tsfromfile (источник – файл), tsfrombus (источник – шина), tsfromip (источник – IP-интерфейс).

Рассмотрим аппаратные элементы декодера (использующиеся в модулях системы воспроизведения видео), через которые проходит поток при его обработке, чтобы найти возможные проблемы в существующей системе.

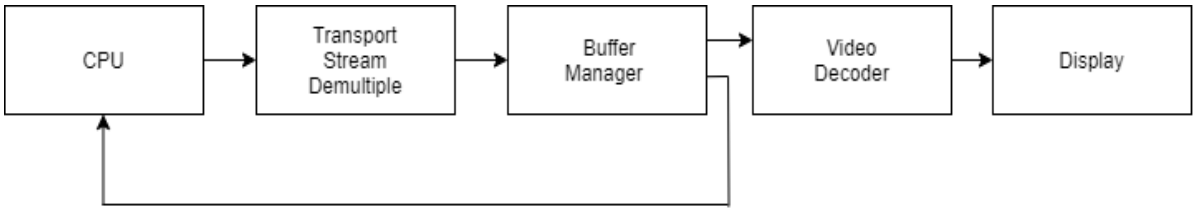

Рисунок 13 – Условная структура элементов, через которые проходит транспортный поток

На рисунке 13 видим, что центральный процессор, исполняя программный код, направляет поток на TSD (Transport Stream Demultiple). Обработанный демультиплексором поток записывается в BM (Buffer Manager). Затем он, при

необходимости направляется на видео декодер, где происходит декодирование его видео составляющей. После чего идет его воспроизведение на дисплее.

Рассмотрим структурные элементы схемы, находящийся на рисунке 12, поподробнее.

Демультиплексор транспортного потока (TSD) может демультиплексировать, скремблировать и дескремблировать транспортные потоки MPEG. Данный функционал выполняется внутри драйверов, предоставляемых компанией Fujitsu, поэтому следить за этим нет необходимости.

Один TSD может поддерживать до 32 каналов PID. Каждый канал может быть сконфигурирован для обработки потока пакетных элементов (PES), либо данных о программе (PSI). Для пакетов PSI TSD может применять различные условия фильтрации, составлять разделы таблицы PSI и проверять CRC раздел, если таковой имеется, за счет этого функционала реализован анализ MPEG-потока.

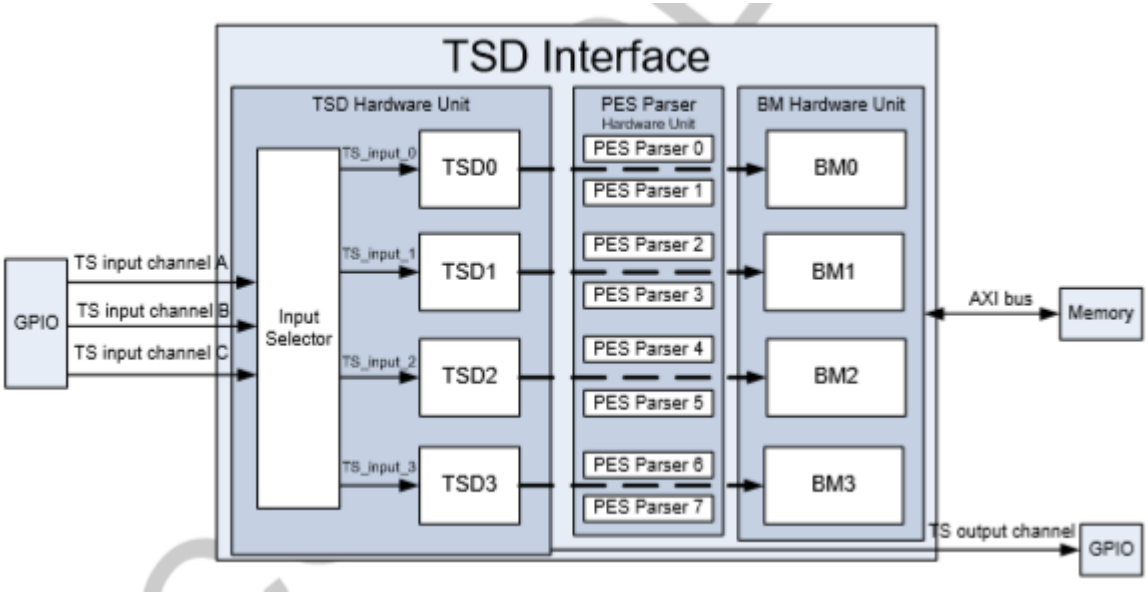

Рисунок 14 - Блок-диаграмма TSD интерфейса [11]

На рисунке 14 видим, что интерфейс TSD содержит четыре TSD (от TSD0 до TSD3), восемь PES парсеров (от PES0 до PES7) и четыре диспетчера буферов (от BM0 до BM3). В текущей системе используется только один TSD, что может вызвать проблемы при воспроизведении потоков, в которых список программ превышает 32 (количество возможных каналов на один TSD). Поэтому нужно будет внести изменения в систему, тем самым позволить ей работать со всеми 4-мя TSD.

Buffer Manager служит промежуточным буфером, между TSD и следующим элементом, в который будет направлен поток. Т.е. из BM, используя API предоставляемое драйвером, будет совершаться считывание данных. Т.к. в текущем драйвере копирование происходит не оптимально, а именно по 4 байта, необходимо его оптимизировать.

Video Decoder преобразует цифровой кодированный видео- и аудиосигнал MPEG в сигнал для воспроизведения на дисплее. Этот элемент управляется скрыто с помощью низкоуровных драйверов Fujitsu, отчего рассматривать его работу не требуется.

Теперь, зная аппаратные ограничения, рассмотрим текущий функционал в модулях получения потока из источника и опишем то, что необходимо добавить для работы с T2-MI потоком.

Модуль tsfromfile.

Функционал:

1) извлечение потока из файла;
2) запись потока в TSD со скоростью, заданной в потоке;
3) измерение скорости транспортного потока MPEG, находящегося в файле.

Стоит отметить, что рассматриваемый модуль записывает поток в TSD0.

Выявленной проблемой является то, что модуль не умеет измерять скорость T2-MI потока, эту возможность необходимо разработать и внедрить. Алгоритм измерения скорости будет описан в следующем пункте.

Необходимо сделать возможность работы с различными TSD.

Модуль tsfrombus.

Функционал:

1) извлечение потока с шины;
2) запись потока в TSD.

Стоит отметить, что рассматриваемый модуль записывает поток в TSD0.

Необходимо сделать возможность работы с различными TSD.

Модуль tsfromip.

Функционал:

1) извлечение потока из программного буфера;
2) запись потока в TSD;

Стоит отметить, что рассматриваемый модуль записывает поток в TSD0.

Необходимо сделать возможность работы с различными TSD.

Добавление возможности работы с различными TSD будет реализовано передачей номера TSD в функцию создания объектов tsfromip.

Рассмотрим взаимодействие модулей при запуске транспортного потока в плеере.

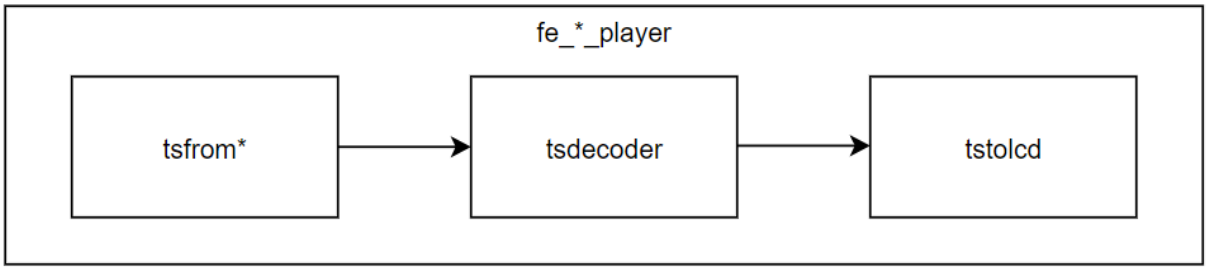

Рисунок 15 – Взаимосвязи модулей режима плеера

На рисунке 15 видим, что модуль fe_player_* (существуют fe_player_demod, fe_player_file, fe_player_ip) управляет подсистемами получения, декодирования и отображения потока. Модуль получения потока из каких-либо источников(tsfrom*) связывается с tsdecoder, который в свою очередь связан с программной единицей отображения (tstolcd). В данной условной схеме взаимосвязи происходит воспроизведение транспортного потока MPEG. Модуль tsdecoder занимается декодированием и воспроизведением потока. Модуль tstolcd отвечает за отображение воспроизводимого потока на экране. Можно сделать вывод, что в схеме нет возможности воспроизвести T2-MI поток.

Для воспроизведения T2-MI потока разработаем новый модуль t2midecoder. Есть несколько возможностей встроить эту программную единицу в текущее программное обеспечение, рассмотрим эти способы.

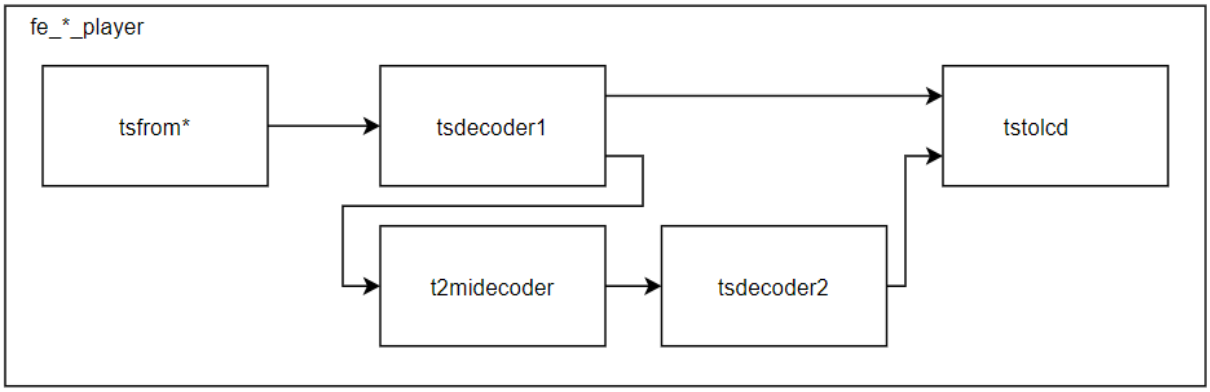

Рисунок 16 – Первый способ взаимосвязи модулей со встроенным t2midecoder

На рисунке 16 видим, что после анализа потока в модуле tsdecoder1, модуль fe_*_player начинает воспроизводить поток, отправляя его в tstolcd (если поток MPEG), либо отправляет поток на декодирование модулю t2midecoder, который деинкапсулирует один транспортный поток MPEG из T2-MI потока. Затем t2midecoder отправляет этот поток на декодирование в отдельный объект tsdecoder2, созданный специально для его обработки. Важным недостатком этого

способа является то, что t2midecoder будет получать поток из аппаратного буфера (ранее ВМ), а не из буфера источника. Из-за чего будет происходить копирование с минимальным КПД, т.к. tsdecoder1 будет передавать 99% потока на t2midecoder (данный процент объясняется тем, сколько T2-MI данных переносится в транспортном потоке MPEG, содержащим T2-MI поток).

Вторым вариантом будет полное закрытие всех объектов fe_player_*, если после анализа было обнаружено, что поток является T2-MI потоком. Затем необходимо будет снова открыть объекты, но добавить объект модуля t2midecoder перед tsdecoder (рисунок 16).

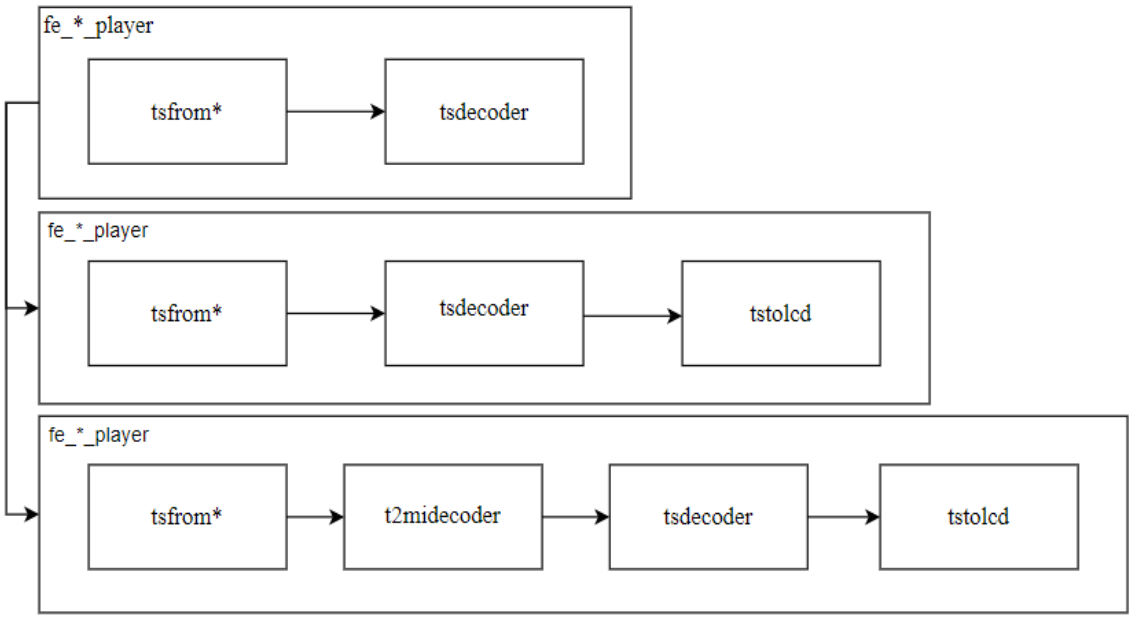

Рисунок 17 – Второй способ взаимосвязи модулей со встроенным t2midecoder

На рисунке 17 видим, что после анализа модулями tsfrom* и tsdecoder, система пересобирается в структуру, находящуюся в середине рисунка (если поток – транспортный поток MPEG), либо в структуру, находящуюся в конце рисунка (если поток – T2-MI поток).

Преимущества первого способа:

1) на первом декодере будет отсекаться часть потока, благодаря чему на t2midecoder будет меньше нагрузки;
2) при необходимости повторного анализа потока, не нужно будет уничтожать все объекты модулей.

Недостатки первого способа:

1) лишнее копирование потока;
2) существование 2-х объектов tsdecoder, их нужно будет согласовывать, т.к. они пользуется одним интерфейсом TSD.

Преимуществом второго способа является отсутствие недостатков первого, но недостатком второго способа является отсутствие преимуществ первого способа. Поэтому реализовывать будем 2-й способ, т.к. лишнее копирование на скорости до 30 Мб/сек сильно нагрузит систему, даже если оно будет происходить на DMA.

Разберем текущий функционал в модуле tsdecoder и опишем то, что необходимо добавить или исправить для работы с T2-MI потоком.

Текущий функционал:

1) декодирование mpeg потока.
2) передача видео и звука на отображение.
3) определение структуры потока (имя сети, набор программ, параметры программ).
4) выбор программы для воспроизведения.
5) передача пакетов с PID видео и звука аппаратным декодерам видео и звука.

Отметим то, что при анализе модуль не запоминает информацию о сервисах, переносящихся в транспортном потоке MPEG, хранящих параметры пакетов с данными. Нужно будет реализовать эту возможность, т.к. именно в пакетах с данными переносится T2-MI поток.

Для декодирования T2-MI потока будет разработан модуль t2midecoder, реализующий следующий функционал:

1) анализ T2-MI потока;
2) передача вычлененного MPEG потока на tsdecoder;
3) определение структуры потока (количество PLP, информацию о PLP);
4) выбор PLP для передачи;
5) вычленение MPEG пакетов из выбранного PLP.

Алгоритм декодирования потока (в результате чего будет вычленяться транспортный поток MPEG и определяться структура потока), будет описан в следующем пункте.

Модулями fe_player_* управляет модуль plrcontext. Он обеспечивает связь между пользователем и плеером, занимается отрисовкой таблиц, отображающих состав программ/PLP, содержащихся в потоке. Также он занимается считыванием информации от пользователя (например, получение выбранного PLP).

Plrcontext разделен на некие состояния, в которых отрисовывается определенная таблица, либо воспроизводится видео. Модуль отвечает за взаимодействие с пользователем (обработка нажатия клавиш и т.п.). Необходимо будет добавить состояние для отрисовки таблицы со списком PLP, в котором будут описаны реакции приложения на нажатые клавиши.

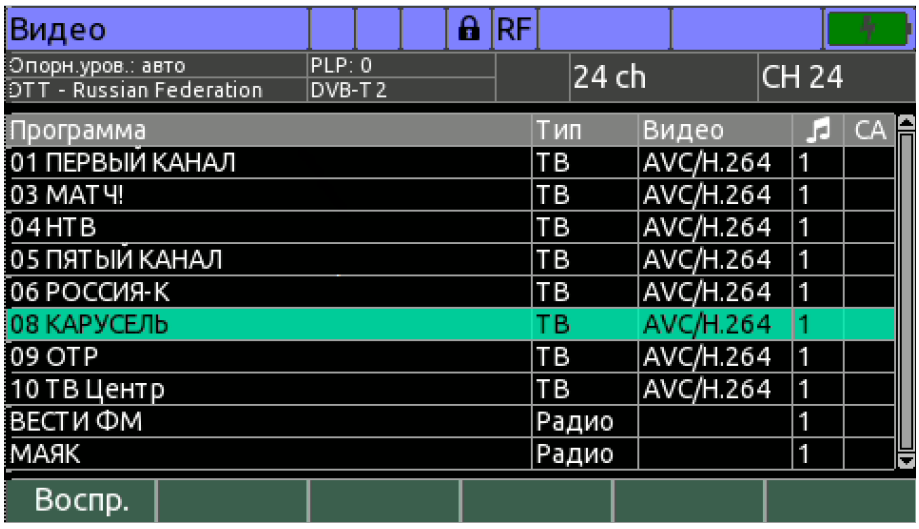

Рисунок 18 – Таблица со списком программ

На рисунке 18 видно, как выглядит таблица со списком программ в текущем программном обеспечении. Спроектируем дизайн таблицы для выбора PLP.

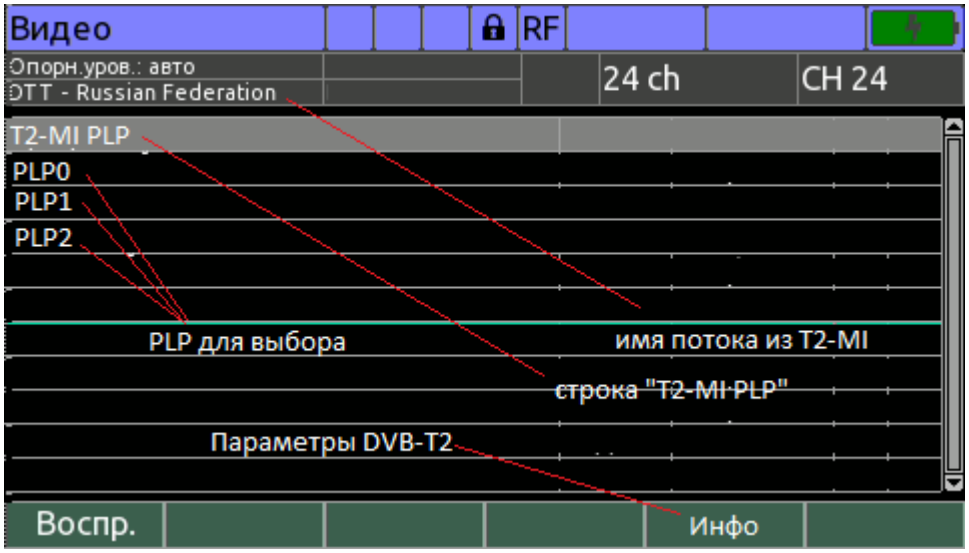

Рисунок 19 – Спроектированная таблица для выбора PLP

На рисунке 19 видим дизайн таблицы, в которой будет происходить выбор PLP. По нажатию кнопки «Инфо» нужно будет отобразить информацию о выбранном PLP.

После выбора PLP нужно будет отобразить его номер, это будет сделано в таблице выбора программы для воспроизведения, как на рисунке 20.

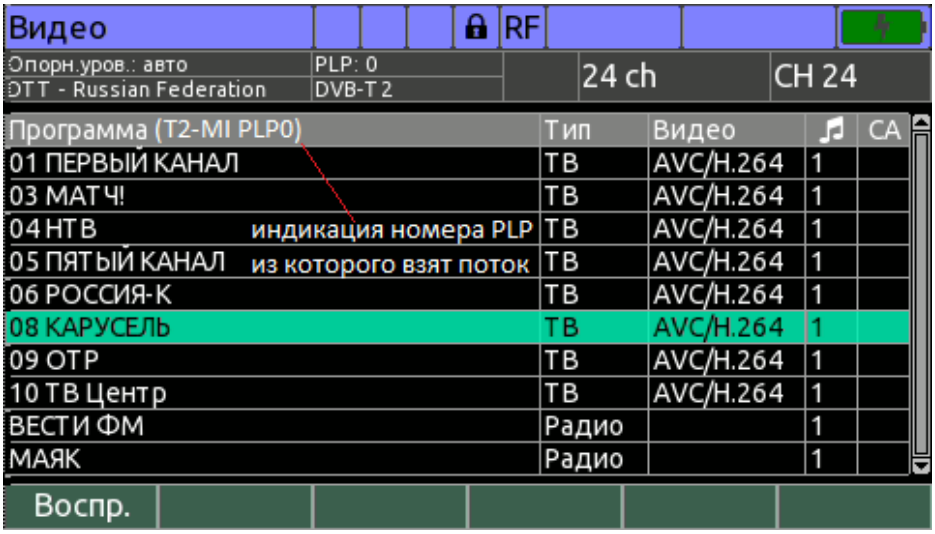

Рисунок 20 - Таблица со списком программ конкретного PLP

3.2. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ

На рисунке 21 представлен обобщённый алгоритм получения T2-MI потока из транспортного потока MPEG.

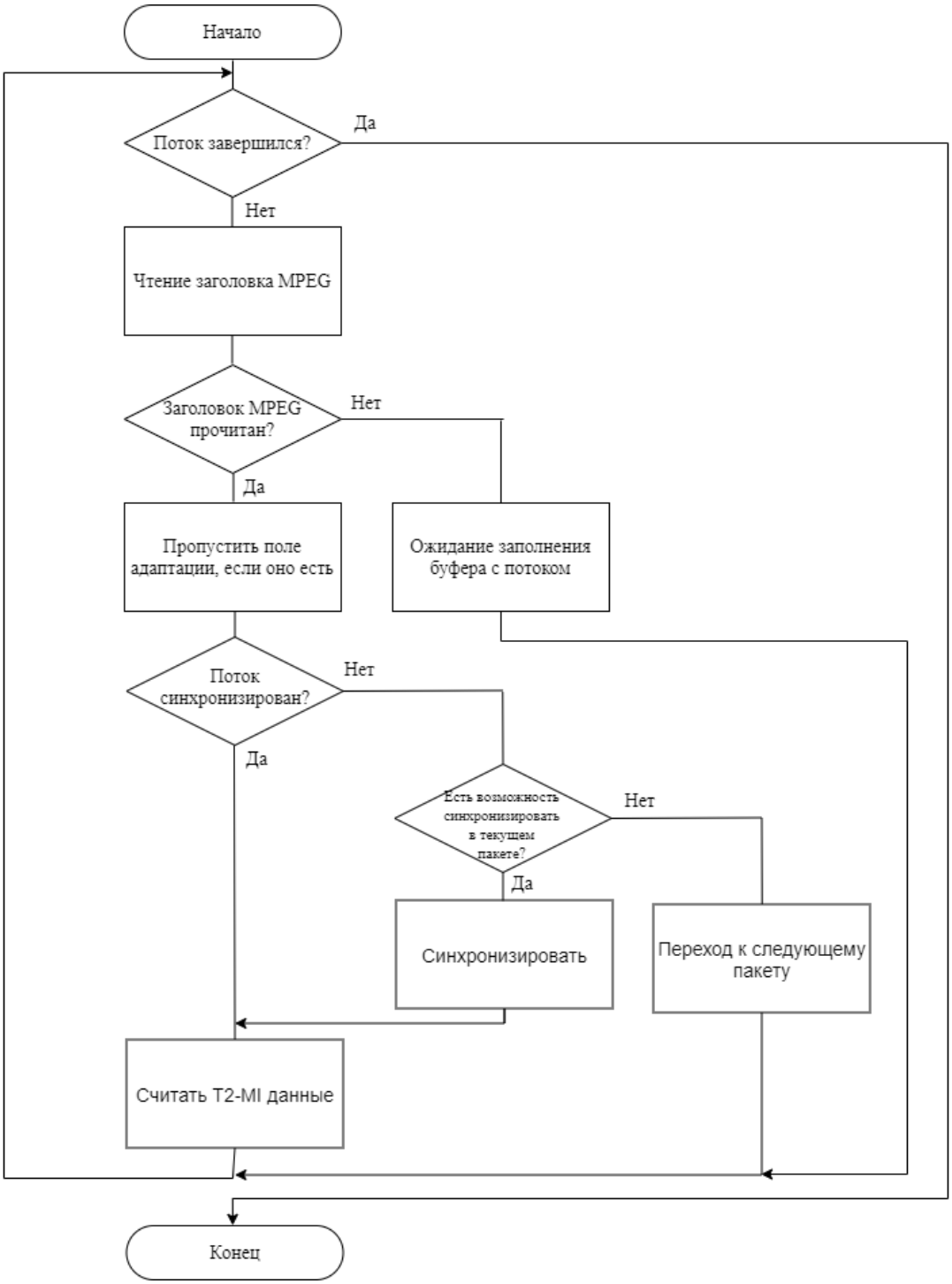

Рисунок 21 – Блок-схема вычленения T2-MI пакета

На рисунке 22 представлен обобщённый алгоритм получения параметров из T2-MI данных полученных после работы алгоритма, представленного на рисунке 21.

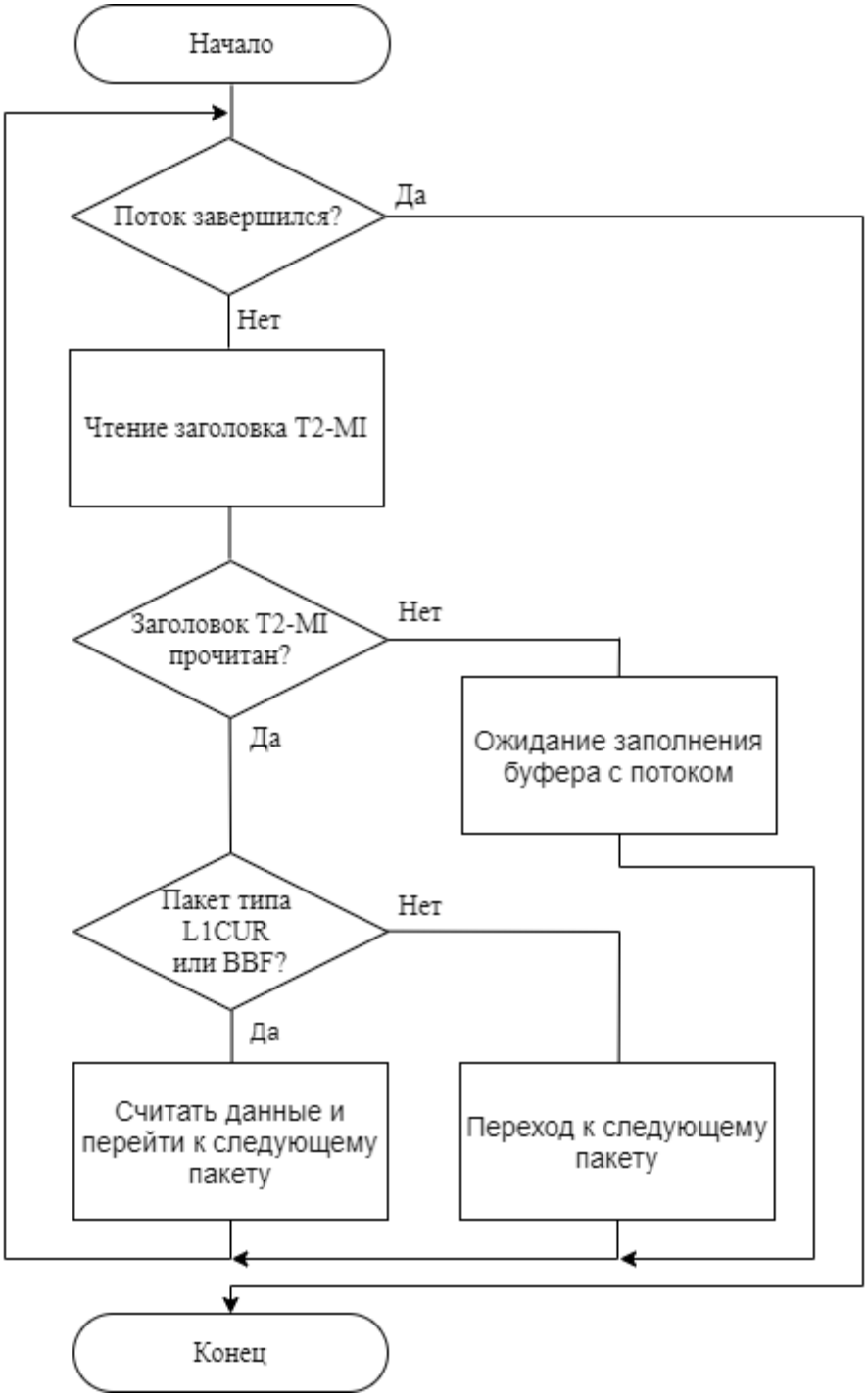

Рисунок 22 – Блок-схема получения пакета с параметрами из T2-MI потока

Алгоритмы, представленные на рисунке 21 и 22 будут использоваться в модуле t2midecoder.

На рисунке 23 представлен обобщённый алгоритм расчета скорости T2-MI потока на основе T2-MI данных, полученных после работы алгоритма, представленного на рисунке 20.

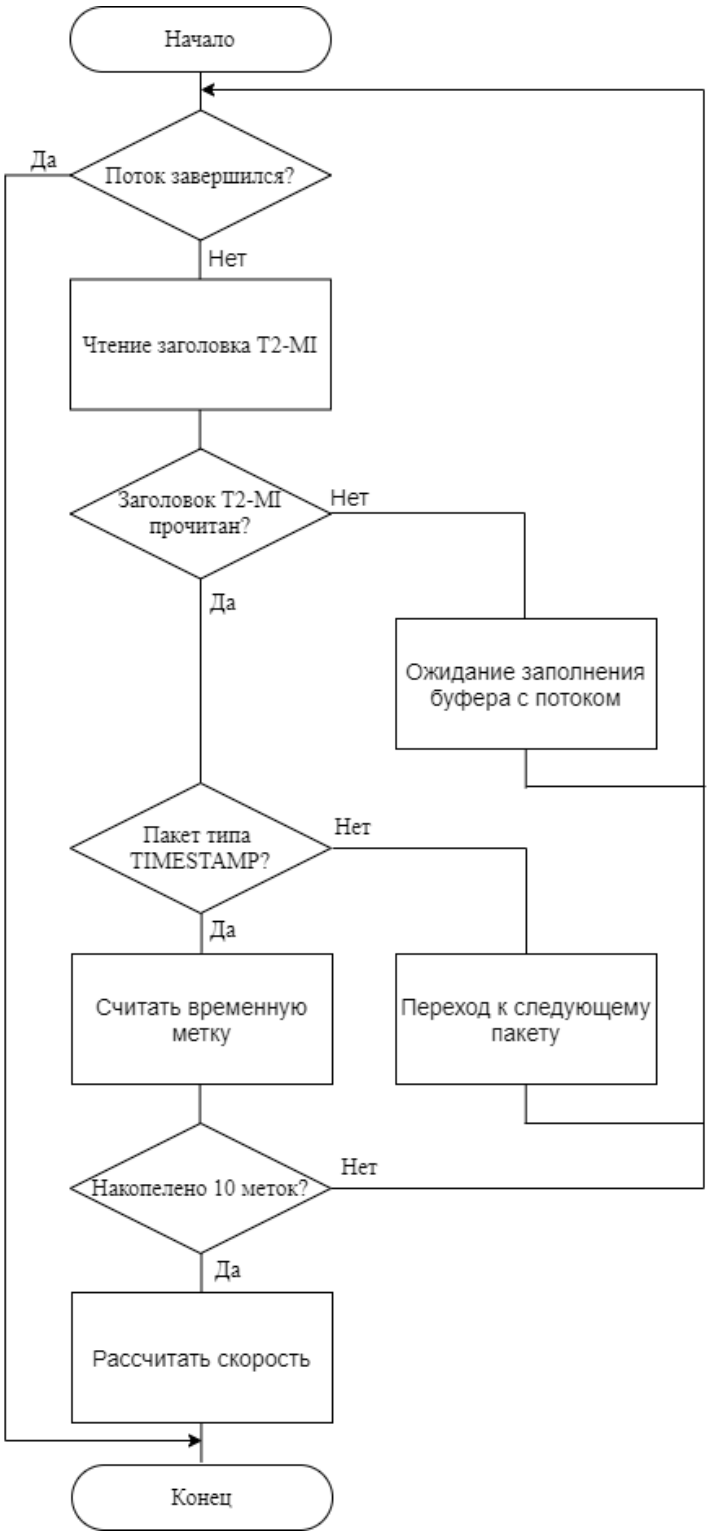

Рисунок 23 – Блок-схема расчета скорости потока T2-MI

Алгоритм рисунка 23 будет находится в отдельном модуле t2mi_measure_bitrate, которым будет пользоваться модуль tsfromfile.

4. РЕАЛИЗАЦИЯ

4.1. РЕАЛИЗАЦИЯ МОДУЛЯ T2MIDECODER

Модуль t2midecoder создан для обработки T2-MI потока и должен выполнять требования, указанные в п.2, для этого были реализованы следующие функции, являющиеся неким API для работы с данной подсистемой.

Таблица 1 – Функции модуля t2midecoder

Названия функций	Описание
t2midecoder_open	Функция открытия подсистемы. Запускает программный поток, в котором происходит обработка T2-MI потока, производит начальную инициализацию подсистемы
t2midecoder_close	Функция закрытия подсистемы. Останавливает программный поток, освобождает память
t2midecoder_play	Функция запуска воспроизведения. Посылает команду программному потоку о том, что необходимо начать передачу данных вычлененных из T2-MI потока на воспроизведение (копирует данные в TSD, управляющимся tsdecoder’ом)

Продолжение таблицы 1

Названия функций	Описание
t2midecoder_pause	Функция остановки воспроизведения. Посылает команду программному потоку о том, что необходимо остановить обработку данных
t2midecoder_get_plp_info	Функция получения информации о PLP. Возвращает указатель по которому хранится информация, полученная из T2-MI потока
t2midecoder_get_l1pre_info	Функция получения информации о L1PRE T2-MI потока. Возвращает указатель, по которому хранится информация, полученная из T2-MI потока
t2midecoder_get_l1post_info	Функция получения информации о L1POST T2-MI потока. Возвращает указатель, по которому хранится информация, полученная из T2-MI потока
t2midecoder_plp_select	Функция выбора PLP для воспроизведения. Посылает команду программному потоку о том, что необходимо передавать информацию указанного PLP

Продолжение таблицы 1

Названия функций	Описание
t2midecoder_plp_deselect	Функция отмены воспроизведения PLP. Посылает команду программному потоку о том, что необходимо остановить передачу информации выбранного PLP

Вся бизнес-логика модуля t2midecoder находится в создаваемом им потоке. Данный поток реализован, как некий цифровой автомат с 4-мя состояниями: сканирование, воспроизведение, пауза, закрытие. Смена состояний происходит по получению определенных команд, посылающихся из функций, описанных в таблице 1.

В состоянии сканирования проистекает накопление пакетов T2-MI, вычленение из них информации о самом потоке и о находящихся в нем PLP.

В состоянии воспроизведения накапливаются не пакеты T2-MI, а пакеты транспортного потока MPEG инкапсулированные в определенный PLP и после накопления определенного количества данных начинается их копирование в TSD с помощью DMA.

В состоянии паузы идет накопление пакетов T2-MI, но они не обрабатываются. Это состояние необходимо для корректной работы всей подсистемы воспроизведения видео.

В состоянии закрытия происходит ожидание того, когда подсистема может быть закрыта. Это состояние необходимо для корректной работы всей подсистемы воспроизведения видео.

Для накопления T2-MI пакетов был написан отдельный модуль t2miparser, т.к. данный функционал необходим не только в этой подсистеме, а еще в подсистеме получения потока из файла для вычисления скорости потока.

T2miparser содержит функции, представленные в таблице 2.

Таблица 2 – Функции модуля t2miaprser

Названия функций	Описание
t2miparser_open	Функция открытия парсера. Производит начальную инициализацию модуля
t2miparser_close	Функция закрытия парсера. Очищает память
t2miparser_get_t2mi_packet	Функция чтения T2-MI пакетов. Функция будет накапливать T2-MI пакет, как только он накопится, функция просигнализирует об этом
t2miparser_skip_to_t2mi_data	Функция пропуска всех данных потока, находящихся до T2-MI данных. Функция будет пропускать все данные до тех пор, пока «не наткнется» на данные T2-MI потока
t2miparser_get_t2mi	Функция получения накопленного пакета T2-MI

Помимо перечисленного в таблице 2, рассматриваемый модуль имеет множество других функций, необходимых для получения конкретных полей из пакета T2-MI. Например: t2miparser_get_length – получение длины пакета, t2miparser_get_t2mi_type – получение типа пакета, t2miparser_get_plp –

получение PLP пакета, get_l1pre_from_packet – получение L1PRE информации из пакета в виде структуры и т.д.

При реализации данной подсистемы было обнаружено, что в драйвере Buffer Manager функция чтения из буфера была плохо оптимизирована и из-за нее, во время воспроизведения видео из IPTV-потока, процессор прибора был сильно нагружен. Это приводило к тому, что прибор работал нестабильно. Было принято решение переписать драйвер.

Неоптимизированность функции заключалось в том, что копирование происходило по 4 либо по 1 байту. В результате оптимизации функция стала копировать 8, 4, 2 либо 1 байт, в зависимости от адреса копируемого байта данных.

Был переписан модуль инициализирующий копирование с помощью DMA. В исходной версии модуля была возможность копировать данные только в TSD0 (TSD под номером 0), в переписанной – номер TSD можно задать при инициализации.

В модуле t2midecoder была реализована борьба с джиттером. Джиттер – задержка между пакетами [12]. Борьба с джиттером, необходима, чтобы поток воспроизводился равномерно, что не всегда возможно, т.к. задержка между пакетами может сильно различаться. Таким образом усреднение задержки между пакетами будет заключается в том, что поток, полученный из источника, отправляется на TSD с задержкой (во время задержки программный поток спит), величина которой зависит от наполненности буфера источника потока (для борьбы с джиттером заполненность должна быть в среднем 50%).

При реализации подсистемы использовался некоторый функционал операционной системы FreeRTOS, а именно семафоры, таймеры, очереди [13]. Использовались библиотека управления кучей (для выделения памяти под

промежуточный буфер (борьба с джиттером) и библиотека для выделения динамической памяти под T2-MI пакеты (аналог vector из C++).

Для отладки модуля в его код были внедрены строки написания логов с 4 режимами:

1) режим, в котором логи не выводятся;

2) режим, в котором пишутся только сообщения об ошибках;

3) режим, в котором, в дополнение к логам из предыдущего режима, пишутся все глобальные события, происходящие в модуле (воспроизведения, остановка и т.д.);

4) режим, в котором, в дополнение к логам из предыдущих режимов, выводится малозначительная информация (получение данных из bbf пакета, обновление потока и т.д.).

Исходный код модуля t2midecoder представлен в листинге 1 приложения А.

Исходный код модуля t2miparser представлен в листинге 2 приложения В.

4.2. РЕАЛИЗАЦИЯ МОДУЛЕЙ ПОЛУЧЕНИЯ ПОТОКА ИЗ ИСТОЧНИКА

Модули tsfromfile, tsfrombus, tsfromip созданы для получения потока из источников, таких как файл, аппаратные демодуляторы (получают поток их эфира, со спутникового сигнала), IP-TV поток. Т.к. T2-MI поток изначально инкапсулирован в транспортный поток MPEG, то большая часть функционала уже присутствует в данных модулях. Поэтому необходимо, как говорилось в п.3, добавить возможность работы с разными TSD. Также для модуля получения потока из файла – реализовать возможность измерения скорости T2-MI потока.

Для возможности работы с разными TSD необходимо во время инициализации передавать номер используемого TSD, затем запоминать его в дескрипторе модуля и в дальнейшем проводить все операции именно с запомненным TSD.

Для измерения скорости T2-MI потока в подсистеме tsfromfile, были написаны 2 дополнительных модуля – mpegPidStatistics, t2miMeasBitrate.

Модуль mpegPidStatistics необходим для определения того, является ли исходный поток T2-MI потоком. Т.к. проводить анализ потока слишком накладно, было принято решение собрать статистику количества различных PID в пакетах транспортного потока MPEG. Если процентное количество одного из PID превышает 97% - можно быть уверенным, что данный поток является T2-MI потоком.

В результате этот модуль имеет одну функцию - mpeg_pid_statistics, в которую передается функтор (выполняющий получение одного пакета mpeg) и количество пакетов в выборке. Возвращает функция массив из структур с двумя полями – сам PID и сколько пакетов с данным PID в выборке. Таким образом по полученной статистике можно будет рассчитать частоту встречи пактов с определенным PID и судить о том является ли поток T2-MI потоком.

Исходный код модуля mpegPidStatistics отображен в листинге 3 приложения С.

Модуль t2miMeasBitrate предназначен для вычисления скорости T2-MI потока, вычисление основывается на существовании в потоке пакетов типа TIMESTAMP, содержащих временные характеристики, выставляемые модулятором. Эти характеристики выставляются с определенным интервалом (например, каждую секунду) и представляют собой временные метки, содержащие время их выставления. Таким образом, разница между двумя

временными метками и количество данных между ними позволяет узнать скорость потока. Алгоритм расчета скорости представлен на рисунке 23.

После реализации t2miMeasBitrate имеет одну функцию - t2mi_meas_bitrate, которая принимает функтор для получения одного пакета MPEG и PID, соответсвующий T2-MI потоку. После своей работы, возвращает скорость потока (бит/сек).

Исходный код модуля t2miMeasBitrate представлен в листинге 4 приложения D.

4.3. РЕАЛИЗАЦИЯ МОДУЛЯ TSDECODER

Tsdecoder создан для декодирования транспортного потока MPEG. Т.к. программное обеспечение уже умеет работать с данным потоком, то большая часть функционала уже присутствует в модуле. Поэтому необходимо, как говорилось в п.3, добавить возможность запоминать информацию о сервисах, переносящих данные, т.к. именно такой тип пакетов переносит T2-MI поток. Из-за того, что управление видео декодером происходит именно в этом модуле, нужно будет обеспечить ему возможность управления Hardware control buffer(HCB) при необходимости.

Существует аппаратное ограничение - воспроизведение потока, может быть только на одном TSD, который сконфигурирован как Hardware control buffer(HCB), т.е. специальный буфер, контролирующийся аппаратно, а не программно. Т.к. во время проектирования был выбран способ связывания модулей, изображенный на рисунке 17, в системе всегда будет использоваться только 1 TSD. Из этого следует, что рассматриваемое аппаратное ограничение не является существенным, но в будущем может понадобиться использовать несколько TSD, вследствие чего необходимо обойти это ограничение. Для обхода

необходимо переписать драйвер Buffer Manager. В изначальной версии драйвера изменять номер BM, который будет являться HCB, можно только перед инициализацией драйвера, за что отвечает функция FAPI_BM_SetHardwareControlledBlock. Добавив в нее проверку возможности смены HCB можно обойти ограничение и при необходимости можно будет в программном коде изменить номер HCB на нужный. Сменить HCB возможно при условии, что BM являющийся HCB, во время данной операции, свободен, т.е. не делает никаких вычислений. Теперь при запуске tsdecoder для воспроизведения, используемый им BM будет становиться HCB.

Во время реализации было принято решение, что для того чтобы все модули работали с одним и тем же TSD, номер TSD будет определяться в модуле tsdecoder, т.к. именно он полноценно работает с данным аппаратным элементом. Для определения с каким TSD работать, в его драйвер была добавлена возможность узнать свободен ли TSD с заданным номером. Этот функционал выполняет функция FAPI_TSD_Is_Used, которая принимает номер TSD и возвращает булевое значение, по которому можно понять используется ли заданный аппаратный элемент. После определения номера TSD, его можно узнать с помощью функции - tsdecoder_get_tsd_index, после чего передать в остальные модули, чтобы они также работали с данным TSD.

В исходном программном обеспечении запоминается информация о сервисах, переносящих видео и аудио. Для того, чтобы запоминать информацию о сервисах, переносящих данные, и не сломать уже существующий функционал, было решено отдельно запоминать сырую информацию о сервисах, т.е. хранить два вида сервисов: сырые сервисы – все сервисы, сервисы – сервисы с видео и аудио. Если оказывается, что количество сырых сервисов равно 1, а количество сервисов аудио и видео равно 0, то можно судить о том, что работа проходит с T2-MI потоком (эта проверка происходит на уровне подсистемы плеера).

Во время тестирования было обнаружено, что при запуске T2-MI потока на воспроизведение, его анализ происходил долго. Это было связано с тем, что в транспортном потоке MPEG существуют некие таблицы с информацией (PAT, PMT, STD и NIT) и в потоке T2-MI отсутствуют таблицы STD. Но судить о том, что поток является T2-MI можно после получения PAT и PMT таблиц. Поэтому был добавлен функционал для передачи информации полученной из PAT и PMT. После чего если определяется, что поток является T2-MI потоком, его анализ завершается.

4.4. РЕАЛИЗАЦИЯ ПОДСИСТЕМЫ ВОСПРОИЗВЕДЕНИЯ

Данная подсистема управляет подсистемами, описанными ранее. Можно сказать, что она предоставляет API для воспроизведения видео и содержит функции, представленные в таблице 3.

Таблица 3 – Функции плеера

Функция	Описание
open	Проведение начальной инициализации, открытие объектов других подсистем
close	Закрытие объектов других подсистем, освобождение памяти
process_report	Получение команд для фронт-енд плеера, например для отображения определенной информации пользователю

Продолжение таблицы 3

Названия функций	Описание
get_content_type	Получение типа контента (аналоговый канал, цифровой канал и т.д.)
is_file_player	Получение информации о том, запущен ли поток из файла
is_cam_available	Получение информации о готовности CAM модуля
select_program	Выбор определенной программы для воспроизведения
deselect_program	Отменить выбор определенной программы для воспроизведения
select_plp	Выбор определенного PLP для воспроизведения
deselect_plp	Отменить выбор определенного PLP для воспроизведения
get_tstolcd_handler	Получить дискриптор модуля tstolcd
get_position	Получить текущую позицию воспроизводимого видео в потоке(при воспроизведении из файла)
set_position	Установить позицию воспроизводимого видео в потоке (при воспроизведении из файла)
pause	Остановить работу плеера
play	Возобновить работу плеера
get_stream_info	Получить информацию о потоке

Продолжение таблицы 3

Названия функций	Описание
is_t2mi	Получить информацию о том, является ли поток T2-MI потоком

В данной подсистеме существуют три вида экземпляров плеера – fe_player_file, fe_player_demod, fe_player_ip. Все они имеют перечисленные в таблице 3 функции, но у них отличается их реализация, т.к. источники потока у них разные.

В процессе реализации были переписаны многие функции, в них были изменены связи между подсистемами, чтобы обеспечить связность, как на рисунке 17.

В функции process_report, были добавлены обработки событий (получаемых из tsdecoder) сигнализирующих о том, что были получены таблицы PAT и PMT, после чего теперь происходят определение типа потока и дальнейшее перестроение подсистемы, для воспроизведения контрактного потока.

Были добавлены обработки событий (получаемых из t2midecoder), индицирующие этапы воспроизведения T2-MI потока, чтобы фронт-енд обработчик знал, какую информацию показывать пользователю.

Были написаны следующие функции: select_plp, deselect_plp, is_t2mi. В функции select_plp совершается вызов функции выбора PLP подсистемы t2midecoder и запускается экземпляр tsdecoder для дальнейшего воспроизведения. Функции deselect_plp содержит обратный процесс функции select_plp. Is_t2mi – говорит о том, какой поток воспроизводиться.

Был реализован функционал, позволяющий хранить информацию о T2-MI потоке, получаемую с помощью одной из функций подсистемы t2midecoder.

Теперь функция get_stream_info возвращает информацию и о T2-MI потоке, если он воспроизведен.

4.4. РЕАЛИЗАЦИЯ ФРОНТ-ЕНД ПОДСИСТЕМЫ ВОСПРОИЗВЕДЕНИЯ

Фронт-енд подсистема воспроизведения представляет собой некий цифровой автомат, который занимается отрисовкой информации для пользователя и получением команд от пользователя.

В цифровом автомате несколько состояний:

1) состояние синхронизации;
2) состояние выбора программы;
3) состояние выбора PLP;
4) состояние воспроизведения;
5) состояние полноэкранного воспроизведения;
6) состояние десинхронизации.

В состоянии синхронизации происходит проверка подключения к источнику потока (например, проверяется, подключен ли спутниковый кабель), а также совершаются синхронизация с потоком и его анализ.

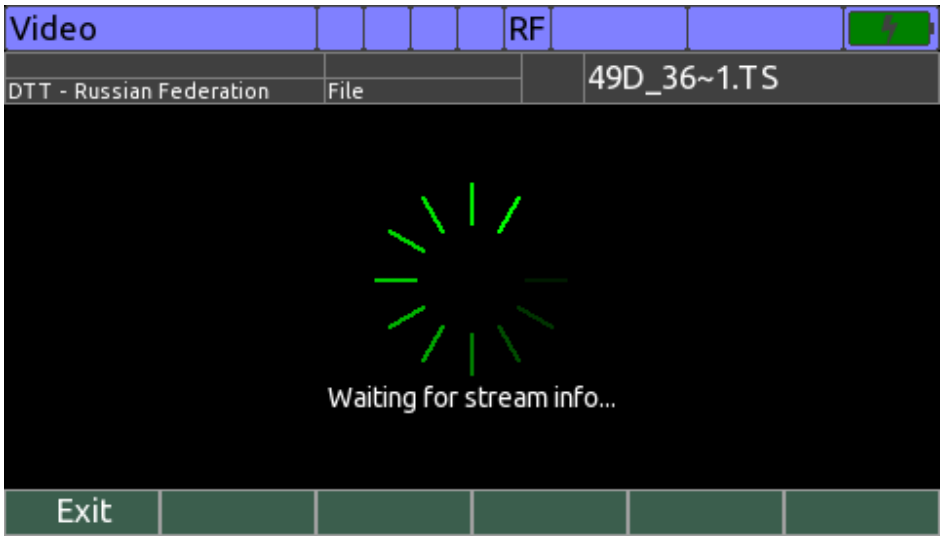

Рисунок 24 – Состояние синхронизации

В состоянии выбора программы пользователю рисуется таблица со списком программ, полученных при анализе транспортного потока MPEG, и краткая информация о самих программах. Пользователь может выбрать необходимую ему программу, после чего выполнится выход из данного состояния.

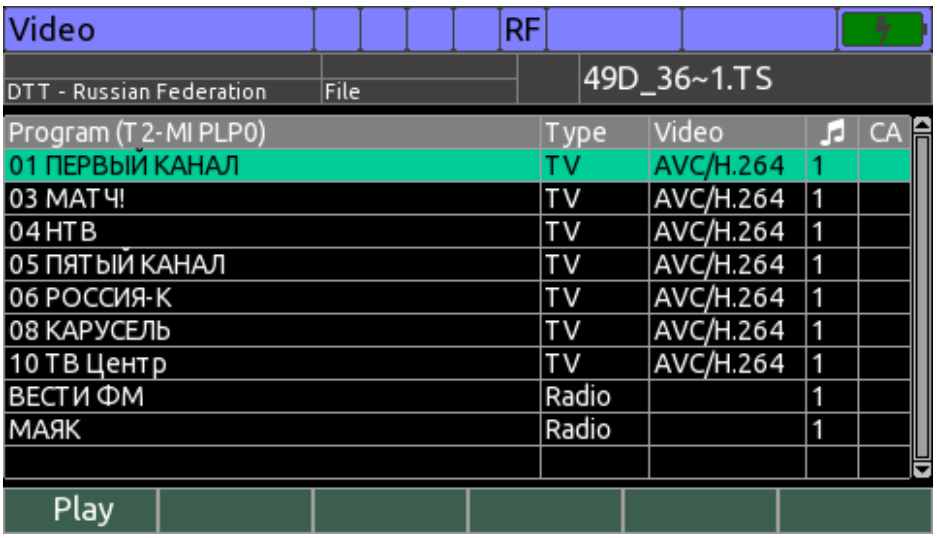

Рисунок 25 – Состояние выбора программ

В состоянии выбора PLP пользователю рисуется таблица со списком PLP, полученным при анализе T2-MI потока. При нажатии на определенную кнопку можно сформировать информацию о потоке, полученную при его анализе. Пользователь может выбрать необходимый ему PLP после чего произойдет выход из данного состояния.

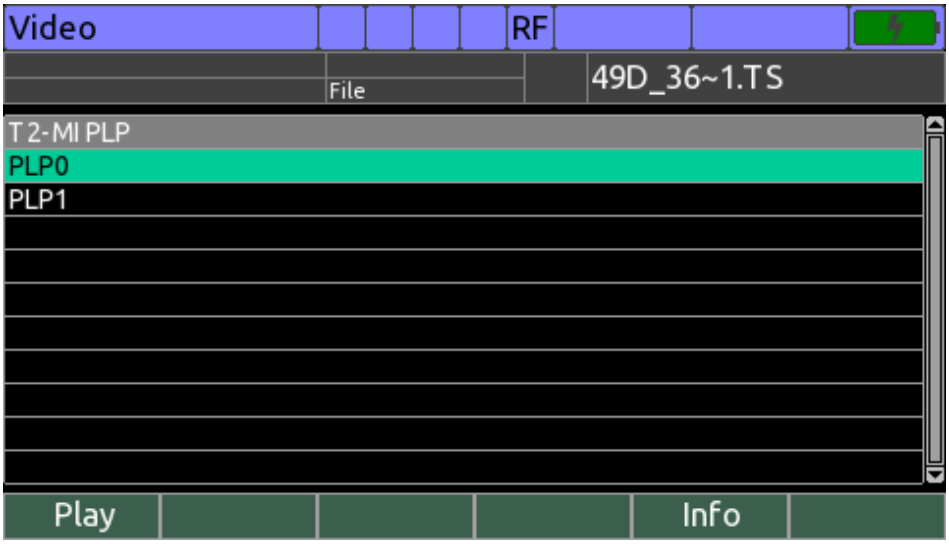

Рисунок 26 – Состояние выбора PLP

В состоянии воспроизведения на экране появляется видео, либо картинка (при воспроизведении аудио), рисуется небольшая таблица с краткой информацией о воспроизводимой программе.

Рисунок 27 – Состояние воспроизведения

В состоянии полноэкранного воспроизведения появляется видео, но по сравнению с состоянием воспроизведения, оно выводится в полноэкранном режиме.

Рисунок 28 – Состояние полноэкранного воспроизведения

В состояние десинхронизации можно попасть при утрате синхронизации с потоком, например, при отключения спутникового кабеля, либо если в потоке много ошибок. В данном состоянии выполняется восстановление синхронизации с потоком или ожидание подключения источника потока. После востановления синхронизации воспроизводятся все команды, полученные от пользователя до десинхронизации, чтобы пользователю не нужно было повторять эти команды.

Состояния переключаются в зависимости от протекаемых в системе воспроизведения процессов и получаемых команд от пользователя.

Состояния реализованы в виде функторов, подменяющих друг друга в самом потоке цифрового автомата. Для внедрения ранее реализованного функционала было необходимо добавить состояние выбора PLP, а также изменить состояние десинхронизации и состояние выбора программы.

Для реализации состояния выбора PLP, был написан отдельный модуль plrinfo_t2mi, реализующий возможности: обработка нажатия клавиш клавиатуры, отрисовка таблиц, вывод дополнительных таблиц с полной информацией о T2-MI потоке.

Для вывода дополнительных таблиц, использовался модуль модальных окон, которому был создан шаблон (модель), состоящий из нескольких таблиц для отображения информации, получаемой из подсистемы t2midecoder.

Для обработки нажатия клавиш и отрисовки таблиц использовались уже написанные библиотеки.

Исходный код модуля plrinfo_t2mi отображен в листинге 5 приложения Е.

5. ТЕСТИРОВАНИЕ

5.1. МЕТОДОЛОГИИ ТЕСТИРОВАНИЯ

Основным методом тестирования был выбран метод use case тестирования.

«Use case — это сценарии, описывающие то как actor (обычно человек, но может быть и другая система) пользуется системой для достижения определенной цели. Варианты использования описываются с точки зрения пользователя, а не системы. Внутренние работы по поддержанию работоспособности системы не являются частью варианта использования» [14].

Этот метод был выбран по причине, что протестировать работу отдельных частей функций является достаточно долгой и трудоемкой задачей. К тому же есть риск испортить прибор, т.к. чтобы отслеживать работу функций и автоматизировать весь процесс тестирования необходимо разобрать прибор и установить дополнительную аппаратуру, позволяющую отслеживать состояние регистров и т.д.

Но сам код, на наличие каких-либо утечек памяти, неопределённого поведения, либо других подобных ошибок, был проверен с помощью статического анализатора, представляющего из себя программное обеспечение, совершающее проверку кода без его компиляции (в явном виде) [15].

Примеры тест-кейсов:

Тест-кейс 0. Выход до получения информации о т2ми потоке.

Шаги:

1) воспроизвести t2-mi поток. (f1 -> video, либо просто enter);

2) во время получения информации о потоке нажать Exit.

Ожидаемый результат - переход в режим «файл менеджер».

Тест-кейс 1. Корректность отображения информации в режиме выбора PLP.

Шаги:

1) воспроизвести t2mi поток. (f1 -> video, либо просто enter);
2) подождать получения информации о потоке.

Ожидаемый результат:

1) список plp, как на рисунке 29;
2) получено имя потока, имя потока может быть не найдено (надпись "nit not found");
3) верно указано имя файла и указано, что поток воспроизводится из файла.

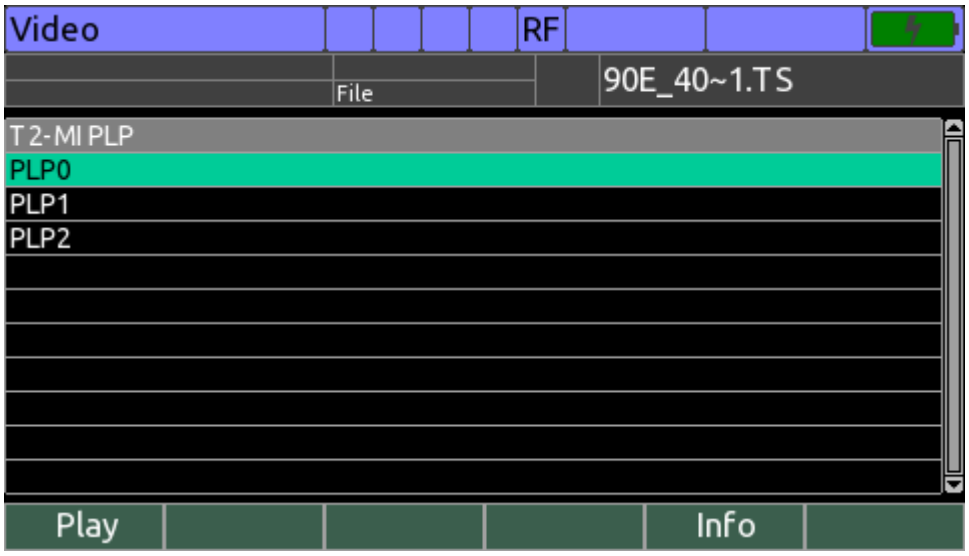

Рисунок 29 – Ожидаемый результат тест-кейса 1

5.2. ТЕСИТРОВАНИЕ РЕЖИМА ВОСПРОИЗВЕДЕНИЯ

Тест-кейсы, описанные в п. 5.1., были переданы двум тестерам. Они должны были заполнить таблицу, состоящую из столбцов: номер тест-кейса, описание, результат (успешно/не успешно), замечание. Каждый тест-кейс было необходимо выполнить 5-10 раз, либо до первого негативного результата.

Далее представлена полученная в результате итогового тестирования таблица (таблица 4).

Таблица 4 – Результаты тестирования воспроизведения из файла

Номер тест кейса	Описание	Результат
0	Выход до получения информации о T2-MI потоке	Успешно
1	Корректность отображения информации в режиме выбора PLP	Успешно
2	Выход при просмотре списка PLP	Успешно
3	Нажатие нефункциональных клавиш в режиме просмотра списка PLP	Успешно
4	Скроллинг списка PLP	Успешно
5	Скроллинг списка PLP с большим шагом	Успешно
6	Смена табличек с информацией о PLP	Успешно
7	Скроллинг таблиц с информацией о PLP	Успешно
8	Нажатие нефункциональных клавиш при просмотре таблиц с информацией о PLP	Успешно

Продолжение таблицы 4

Номер тест кейса	Описание	Результат
9	Смена информации при обращении к различным PLP	Успешно
10	Открытие таблиц с информацией о PLP и их закрытие	Успешно
11	Возвращение к списку PLP во время выхода из поиска информации о mpeg потоке(выбранного PLP)	Успешно
12	Выход из плеера при воспроизведении MPEG потока	Успешно
13	Проверка индикации состояния плеера и отображение поиска имени потока при воспроизведении t2mi	Успешно
14	Проверка индикации состояния плеера и отображение поиска имени потока при воспроизведении mpeg потока	Успешно
15	Корректность отображения информации в режима выбора программы(для конкретного PLP)	Успешно

Продолжение таблицы 4

Номер тест кейса	Описание	Результат
16	Корректность отображения информации в режима выбора программы	Успешно
17	Корректность отображения имени потока при возвращении к списку PLP от списка программ(конкретного PLP)	Успешно
18	Проверка корректности информации для выбранного PLP, при выборе разных PLP без выхода из плеера	Успешно
19	Проверка нажатия нефункциональных клавиш в режиме выбора программы(конкретного PLP)	Успешно
20	Скроллинг программ	Успешно
21	Скроллинг программ(конкретного PLP) с большим шагом	Успешно
22	Воспроизведение видео после выбора программы	Успешно
23	Нажатие нефункциональных кнопок во время воспроизведения видео	Успешно

Продолжение таблицы 4

Номер тест кейса	Описание	Результат
24	Изменение громкости во время воспроизведения видео	Успешно
25	Смена программ во время воспроизведения видео	Успешно
26	Изменение звуковых дорожек	Успешно
27	Воспроизведение видео в полноэкранном режиме	Успешно
28	Нажатие нефункциональных кнопок во время полноэкранного воспроизведения	Успешно
29	Изменение громкости во время полноэкранного режима	Успешно
30	Скроллинг программ в полноэкранном режиме	Успешно
31	Корректность информации после перехода в обычный режим воспроизведения из полноэкранного	Успешно
32	Выключение прибора из всех режимов работы плеера	Успешно
33	Подключение USB кабеля для всех режимов работы плеера	Успешно

Продолжение таблицы 4

Номер тест кейса	Описание	Результат
34	Выбор программ и PLP, при долгом нахождение в режиме выбора программ и PLP	Успешно
35	Подключение USB кабеля после долго нахождения в режимах выбора программ и выбора PLP	Не успешно

Не удовлетворительный результат кейса 35 связан с ошибкой исходного программного обеспечения, т.к. данный кейс происходит достаточно редко, было принято решение отложить исправление выявленной ошибки на будущее.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано программное обеспечение для воспроизведения видео из T2-MI потоков.

Разработанное ПО может работать с T2-MI потоками, полученными из различных источников: IP-TV, спутник, файл.

Во время реализации были отредактированы/созданы около 55 файлов, содержащих программный код.

Процесс анализа потока, несущего 3 PLP со скоростью 30 Мб/сек, занимает около 0,5 секунды.

В режимах воспроизведения со спутника и из файла прибор работает стабильно с потоками, имеющими скорость до 80 Мбит/сек.

При воспроизведении потока IP-TV эффект джиттера не проявляется.

ПО позволяет получить информацию, содержащуюся в T2-MI потоке, а именно L1-PRE, L1-POST и L1-POST для каждого PLP. При этом отдельно выводиться основная информация о потоке.

Есть возможность работы с T2-MI потоком в котором, как отсуствуют PCR метки, так и присутствуют.

Программный код содержит множество комментариев, позволяющих другим программистам, с должным опытом работы в сфере телекоммуникаций, легко понять, что в нем происходит. Также код был множество раз подвергнут рефакторнигу, что повысило его читаемость.

Во время тестирования были проверены множество сценариев взаимодействия пользователя с ПО. И все они, за исключение одного, показали хороший результат.

Весь текст был локализован для отображения, как на русском, так и на английском языке.

Прошивка, с добавленным функционалом, вышла в релиз 11.03.2021 [16].

Были исправлены некоторые недочеты, выявленные в результате анализа обращений пользователей данного прибора.

Перспективы развития системы:

– оптимизация алгоритма измерения скорости T2-MI потока;

– дальнейшая масштабируемость программного обеспечения, в случае необходимости добавления требуемого функционала.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Цифровой и аналоговый сигнал: в чем сходство и различие, достоинства и недостатки? – http://electrik.info/main/school/559-cifrovoy-i-analogovyy-signal-v-chem-shodstvo-i-razlichie-dostoinstva-i-nedostatki.html. Дата обращения: 15.12.2019.
2. Анализатор цифрового телевизионного транспортного потока АТП-1. – http://www.niitv.ru/product-categories/product/20. Дата обращения: 22.12.2019.
3. Enensys DiviDual T2-MI – Проигрыватель/Анализатор/Рекордер потоков T2-MI. – http://www.enensys.ru/enensys-dividual-t2mi-----2i.html Дата обращения: 22.12.2019.
4. RANGER Neo 4. – https://store.telcogroup.ru/katalog/6/promax/ranger-neo-4.html. Дата обращения: 21.12.2019.
5. Анализатор ТВ сигналов мультисистемный IT-100. – http://www.planarchel.ru/Products/Measurement%20instrument/izmeritelnye-pribory-2/it-100/. Дата обращения: 20.12.2019.
6. ISO/IEC 13818-1. Second edition. Information technology — Generic coding of moving pictures and associated audio information: Systems. – ISO/IEC, 2000. – 154 p.
7. ETSI TS 102 773. V1.4.1. Digital Video Broadcasting (DVB); Modulator Interface (T2- MI) for a second generation digital terrestrial television broadcasting system (DVB-T2). – ETSI, 2016. – 57 p.
8. Язык ассемблера. – https://ru.wikipedia.org/wiki/Язык_ассемблера. Дата обращения: 22.04.2021.
9. Си (язык программирования). – https://ru.wikipedia.org/wiki/Си_(язык_программирования). Дата обращения: 22.04.2021.
10. MicroPython. – https://ru.wikipedia.org/wiki/MicroPython. Дата обращения: 22.04.2021.
11. Device Manual. Edition 1.1. Multi Standard Decoder for Set-Top-Boxes - MB86H615. - Fujitsu company, 2012. – 903 p.
12. Качество сетей передачи данных. Программные и аппаратные измерения. – https://habr.com/ru/post/250821. Дата обращения: 22.04.2021.

13. API Reference. – https://www.freertos.org/a00106.html. Дата обращения: 22.04.2021.
14. Тестирование методом черного ящика. – https://habr.com/ru/post/462837/. Дата обращения: 22.04.2021.
15. Статический анализ – от знакомства до интеграции. – https://habr.com/ru/company/pvs-studio/blog/514124/. Дата обращения: 22.04.2021.
16. История изменений ПО ИТ-100. – http://www.planarchel.ru/Products/Measurement%20instrument/izmeritelnye-pribory-2/it-100/changelog. Дата обращения: 22.04.2021.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПОДСИСТЕМЫ T2MIDECODER

Листинг 1 – Исходный код t2midecoder

```
/* debug levels:
0 - no messages out
1 - only error messages out
2 - all the events messages out
3 - full trace*/
#define DEBUG                          3

#include "t2midecoder.h"
#include "../../../lib/utils/macroCheckers.h"
#include <fapi/sys_services.h>
#include <fapi/drv_tsd.h>
#include <syscrash.h>
#include "..\mod\public\src\itpub\lib\array\array.h"
#include <fapi\reg_tsd.h>
#include "..\..\lib\parser\t2mi\t2miparser.h"
#include "fapi/drv_timer.h"
#include <fapi/drv_dma.h>
#include <fapi/drv_mmu.h>

/* T2MIDECODER thread name */
#define THREAD_NAME                   "t2midec"
/* T2MIDECODER thread stack size in bytes */
#define THREAD_STACK_SIZE_B           ( 8 * 1024 )
/* T2MIDECODER thread priority */
#define THREAD_PRIORITY               RTOS_PRIORITY_HIGH
// lenght of thread queue
#define THREAD_QUEUE_LEN              1
// value of the packet_type field corresponding to the type of the L1CURRENT packet
#define PACKET_TYPE_OF_L1CURRENT      16
// value of the packet_type field corresponding to the type of the Baseband Frame packet
#define PACKET_TYPE_OF_BBF            0
// special value which used when plp not selected
#define PLP_NOT_SELECTED              0xFFFF
// special value for scan_state used when l1current info found
#define L1CUR_FOUND                   1
/* Invalid value for TSD block index */
#define TSD_INVALID                   0xFFFFFFFF
// fill buffer desirable level
#define DESIRABLE_FILL_LEVEL_PCT 50
// first bitrate approximation
#define FIRST_APPROXIMATION_BR 30000000
// MPEG2 TS packet size in bytes
#define TSPKT_SIZE_B                      188
// stream burst size in bytes (must be multiple TSPKT_SIZE_B and 1024)
#define STREAM_BURST_SIZE_B               ( TSPKT_SIZE_B * 512 )
// number of bytes (first 3 byte in Baseband Frame payload and first 10 bytes in bbframe)
#define BYTES_BBF_HEADERS 13
//size of buffer from which dma will copy
#define SIZE_OF_DMA_BUF (256 * TSPKT_SIZE_B)
// number of bytes t2mi header
#define BYTES_T2MI_HEADER 6
// number of bytes (6 bytes t2mi header, first 3 byte in Baseband Frame payload and first 10 bytes in bbframe)
```

```
#define BYTES_T2MI_HEADERS (BYTES_T2MI_HEADER + BYTES_BBF_HEADERS)
// special value determins that tsfifo buffer is terminated
#define TS_FIFO_TERMINATE 0x7FFFFFFF
/* get current time stamp in us */
#define GET_TIMESTAMP_US( )             \
    (uint32_t)FAPI_TIMER_GetTimeStamp( FAPI_TIMER_RESOLUTION_1_USEC )

/* debug output */
#define DEBUGID                         "[t2midec]: "
#define DEBUGOUT( level, ... )          \
    FAPI_SYS_PRINT_DEBUG( level, DEBUGID __VA_ARGS__ )

static FAPI_TSD_TsInputSettingsT ts_input_cfg =
{
    .version                        = FAPI_TSD_VERSION,
    .tsSelect                       = FAPI_TS_B,
    .automaticSync                  = 0,
    .packetStartPolarity            = 0,
    .enablePolarity                 = 0,
    .clockInvert                    = 0,
    .serialParallelMode             = 0x3,
    .packetBufferOverflow           = 0,
    .writeEndian                    = 0x00,
    .sbz                            = 0,
    .syncByteDistance               = 0,
    .tsErrorSettings.version        = FAPI_TSD_VERSION,
    .tsErrorSettings.removeTsPacket  = 0,
    .tsErrorSettings.irqMode        = 0x2,
    .tsErrorSettings.countStartValue = 0,
    .tsErrorSettings.sbz            = 0,
};

// TS-FIFO consumer configuration
static const X_TSFIFO_CONSUMER_CFG g_stream_cfg =
{
    .burst_size_b = STREAM_BURST_SIZE_B,
    .f_use_receiver_context = false,
    .f_kill_jitter = true,
    .cb =
    {
        .post_push = NULL,
        .pre_shutdown = NULL,
    },
    .aux = NULL
};

typedef enum E_PLAYING_STATE {

    GOING_TO_T2MI = 0,
    READING_T2MI_HEAD,
    READING_BBF_HEAD,
    T2MI_SYNC,
    SENDING,
    SKIPPING,

} E_PLAYING_STATE;

typedef struct X_T2MI_SEND_CONTEXT {

    uint8_t mpeg_bytes_to_go;
```

```
    int prev_counter_t2mi;

    uint16_t t2mi_bytes_done;
    uint16_t t2mi_data_length;
    uint16_t t2mi_length;
    uint8_t t2mi_headers[BYTES_T2MI_HEADERS];

    bool mpeg_send_sync;
    int32_t remaning_bytes_mpeg_packet_to_send;
    uint8_t* mpeg_packets_buf;
    uint32_t buf_cur_ptr;

    E_PLAYING_STATE playing_state;
    int32_t err;

} X_T2MI_SEND_CONTEXT;

typedef struct T2MI_SCAN_CONTEXT {
    uint8_t scan_state;
    uint32_t bbf_scan_state;
    uint8_t bbf_info_counter;
} X_T2MI_SCAN_CONTEXT;

typedef enum E_THREAD_CMD {

    START_SCAN,
    PLAY,
    CLOSE,
    PAUSE,
    SELECT_PLP,
    DESELECT_PLP,

} E_THREAD_CMD;

typedef struct X_THREAD_MSG
{
    E_THREAD_CMD cmd;
    union
    {
        uint8_t plp;
    }data;

} X_THREAD_MSG;

typedef enum E_THREAD_STATE {

    STOPPED,
    PLAYING,
    SCANNING,
    WAITING_CLOSURE,

} E_THREAD_STATE;

typedef struct X_CONS_CONTEXT {
    int32_t              slice_bytes_to_go;
    uint8_t*             data;
    uint32_t             cur_ptr;
    uint32_t             receive_timestamp;
    uint32_t             send_timestamp;
    //time during which you need to process the slice of mpeg stream
```

```
    uint32_t             total_slice_time;
    unsigned int         stream_rate_bps;
    uint32_t             prev_population_pct;
    uint32_t             tsd_index;
    RTOS_SemaphoreT      dma_sem;
} X_CONS_CONTEXT;

typedef struct X_T2MIDECODER_CONTEXT {

    RTOS_SemaphoreT      cmd_sem;
    RTOS_MailqueueT      thread_queue;
    RTOS_ThreadT         hthread;
    X_T2MIDECODER_CFG    cfg;
    E_THREAD_STATE       state;
    int32_t              thread_err;
    int                  is_appeared_msg;
    H_T2MIPARSER         h_t2miparser;

    tTV_DVBT2_L1PRE      info_L1PRE;
    tTV_DVBT2_PLP*       info_plp;
    tTV_DVBT2_L1POST     info_L1POST;

    bool                 scan_completed;
    X_T2MI_SCAN_CONTEXT  scan_context;

    uint16_t             selected_plp;
    X_T2MI_SEND_CONTEXT  send_context;

    X_CONS_CONTEXT       cons_context;
} X_T2MIDECODER_CONTEXT;

typedef struct X_DMA_START_CFG
{
    /* src @in: pointer to data source */
    const void *src;
    /* data_len @in: data lenght */
    uint32_t data_len;
    /* wait_cycles @in: waitcycles between two write accesses */
    uint32_t wait_cycles;
    /* finishCallback @in: DMA finish callback */
    FAPI_DMA_CallbackT finishCallback;
    /* aux @in: user defined parameter */
    void* aux;

} X_DMA_START_CFG;

static void thread( void* opt );
static bool read_tsfifo_cb( uint8_t* read_buf, uint8_t bytes, void* aux );
static void notify( H_T2MIDECODER handler, E_T2MIDECODER_EVENT_ID id,
                              const X_T2MIDECODER_EVENT_DATA* data );
static void notify_brief( H_T2MIDECODER handler, E_T2MIDECODER_EVENT_ID id );
static void notify_err( H_T2MIDECODER hdl );
static void stopped  ( H_T2MIDECODER handler );
static void playing  ( H_T2MIDECODER handler );
static void scanning ( H_T2MIDECODER handler );
static uint32_t get_min(uint32_t a, uint32_t b);
static void get_info_from_packet_bbf( H_T2MIDECODER handler, const uint8_t* t2mi );
static void free_hdl(H_T2MIDECODER hdl);
static void send_msg_to_thread( H_T2MIDECODER handler, X_THREAD_MSG* cmd );
static int32_t refresh_thread_context(H_T2MIDECODER handler);
```

```
static bool is_plp_present( H_T2MIDECODER hdl, uint8_t plp );
static uint32_t get_tsd_packet_buffer( uint32_t tsd_ind );

static void go_to_t2mi_data(H_T2MIDECODER handler);
static void reading_t2mi_head(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons);
static void reading_bbf_head(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons, uint8_t selected_plp);
static void t2mi_sync(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons);
static void sending(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons);
static void skipping(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons);
static void react_to_thread_error( H_T2MIDECODER hdl );

static bool check_sync_t2mi( X_T2MI_SEND_CONTEXT* cnt, const uint8_t* t2mi );
static int32_t reset_sending( X_CONS_CONTEXT* cnt_cons, X_T2MI_SEND_CONTEXT* cnt );
static int32_t send_to_tsd( X_T2MI_SEND_CONTEXT* send_cnt, uint32_t bytes, X_CONS_CONTEXT* cons_cnt );
static int32_t read_t2mi( X_CONS_CONTEXT* cnt_cons, X_T2MI_SEND_CONTEXT* cnt_send, unsigned int read_lvl, bool
skip );
static int32_t read_tsfifo( X_CONS_CONTEXT* cnt_cons, uint8_t* read_buf, uint8_t bytes );
static int32_t skip_tsfifo( X_CONS_CONTEXT* cnt_cons, uint8_t bytes );
static int32_t fetch_data( H_TSFIFO h_tsfifo, X_CONS_CONTEXT* cons_cnt );
static void wait_cmd( H_T2MIDECODER hdl );
static uint32_t calculate_total_time( X_TSFIFO_STATE* stream_state, X_CONS_CONTEXT* cons_cnt );
static void wait_slice_time( X_CONS_CONTEXT* cons_cnt );
static void dma_cb ( FAPI_DMA_RequestT* dma_req, void* opt );
static int32_t dma_start_ts_packet_copy( const X_DMA_START_CFG *cfg );
static int32_t get_info_from_packet_l1cur( H_T2MIDECODER handler, const uint8_t* t2mi_packet );
static int32_t get_info_from_t2mi_packet( H_T2MIDECODER handler , const uint8_t* t2mi);
static void prepare_to_close( H_T2MIDECODER hdl );

/* open T2MIDECODER entity
cfg @in: decoder configuration
err @out: TSRECV_ERR_OK - success, <0 - failed (error code provided)
@return: T2MIDECODER entity handler (NULL - failed to open)
note: after opening decoder SCANNING starts automatically
 */
H_T2MIDECODER t2midecoder_open( const X_T2MIDECODER_CFG *cfg,
                                    int32_t *err )
{
    X_T2MIDECODER_CONTEXT *hdl = NULL;

    hdl = FAPI_SYS_MALLOC( sizeof( *hdl ) );
    IF_NULL_SETSTATUS_GOTO(hdl, *err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);
    memset( hdl, 0, sizeof( *hdl ) );

    IF_NULL_SETSTATUS_GOTO(cfg, *err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

    hdl->cfg = *cfg;
    hdl->scan_completed = false;
    hdl->selected_plp = PLP_NOT_SELECTED;

    hdl->send_context.prev_counter_t2mi = -1;
    hdl->send_context.remaning_bytes_mpeg_packet_to_send = -1;

    hdl->send_context.mpeg_packets_buf = FAPI_MMU_Malloc( FAPI_MMU_HeapHandleSys1,
                                      SIZE_OF_DMA_BUF );
    IF_NULL_SETSTATUS_GOTO( hdl->send_context.mpeg_packets_buf,
                            *err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR,
                            FUNCEND );

    hdl->cons_context.tsd_index = TSD_INVALID;
    hdl->cons_context.stream_rate_bps = FIRST_APPROXIMATION_BR;
```

```
    const X_T2MIPARSER_CFG cfg_parse = {
        .aux = &hdl->cons_context,
        .read = read_tsfifo_cb,
        .t2mi_pid = cfg->pid,
    };

    hdl->h_t2miparser = t2miparser_open( &cfg_parse, err );
    IF_FAILED_GOTO( (*err) == T2MIPARSER_ERR_OK, FUNCEND );

    hdl->cmd_sem = RTOS_CreateSemaphore( 0 );
    IF_NULL_SETSTATUS_GOTO( hdl->cmd_sem, *err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND );

    hdl->cons_context.dma_sem = RTOS_CreateSemaphore( 0 );
    IF_NULL_SETSTATUS_GOTO( hdl->cons_context.dma_sem , *err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND );

    hdl->thread_queue = RTOS_CreateMailqueue( THREAD_QUEUE_LEN, sizeof( X_THREAD_MSG ) );
    IF_NULL_SETSTATUS_GOTO(hdl->thread_queue, *err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);

    hdl->hthread = RTOS_CreateThread( NULL,
                                      THREAD_STACK_SIZE_B,
                                      THREAD_PRIORITY,
                                      thread,
                                      hdl,
                                      NULL,
                                      THREAD_NAME );
    IF_NULL_SETSTATUS_GOTO(hdl->hthread, *err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);

    *err = FAPI_OK;

FUNCEND:
    if( *err != FAPI_OK ) {
        if(hdl != NULL) {

            if(hdl->hthread != NULL) {
                send_msg_to_thread(hdl, &(X_THREAD_MSG){ .cmd = CLOSE } );
            }

            free_hdl( hdl );
            hdl = NULL;
        }

        DEBUGOUT( 1, "failed to open (%s)\n", syserr_getmsg( *err ) );
    }
    else {
        DEBUGOUT( 2, "opened\n" );
    }

    return hdl;
}

/* close t2mi decoder
handler @in: t2mi decoder handler
@return: TSRECV_ERR_OK - success, <0 - failed (error code provided):
         TSRECV_ERR_BAD_INPUT_PARAMETERS - bad input parameters */
int32_t t2midecoder_close( H_T2MIDECODER hdl )
{
    int32_t err = 0;

    IF_NULL_SETSTATUS_GOTO(hdl, err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND);
```

```
    send_msg_to_thread( hdl, &(X_THREAD_MSG){ .cmd = CLOSE } );

    free_hdl( hdl );
    hdl = NULL;

FUNCEND:
    if( err != FAPI_OK ) {
        DEBUGOUT( 1, "failed to close (%s)\n", syserr_getmsg( err ) );
    }
    else {
        DEBUGOUT( 2, "closed\n" );
    }
    return err;
}

/* t2mi decoder play
handler @in: t2mi decoder handler
@return: TSRECV_ERR_OK - success, <0 - failed (error code provided)
*/
int32_t t2midecoder_play( H_T2MIDECODER handler, uint32_t tsd_index )
{
    int32_t err = TSRECV_ERR_OK;
    IF_NULL_SETSTATUS_GOTO(handler, err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO( handler->state != WAITING_CLOSURE, err, TSRECV_ERR_UNEXPECTED_CALL, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO(handler->selected_plp != PLP_NOT_SELECTED, err, TSRECV_ERR_UNEXPECTED_CALL,
FUNCEND);

    IF_FAILED_SETSTATUS_GOTO( ( tsd_index == FAPI_TSD0 ) || ( tsd_index == FAPI_TSD1 ) ||
                              ( tsd_index == FAPI_TSD2 ) || ( tsd_index == FAPI_TSD3 ),
                              err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

    if( handler->cons_context.tsd_index != tsd_index ) {

        handler->cons_context.tsd_index = tsd_index;

        err = FAPI_TSD_Configure( tsd_index, &ts_input_cfg, NULL );
        IF_FAILED_GOTO( err == 0, FUNCEND );
    }

    send_msg_to_thread( handler, &(X_THREAD_MSG){ .cmd = PLAY } );

FUNCEND:

    if( err != FAPI_OK ) {
        DEBUGOUT( 1, "failed to play plp (%s)\n", syserr_getmsg( err ) );
    }
    else {
        DEBUGOUT( 2, "plp is playing\n" );
    }

    return err;
}

/* select plp for decoding
handler @in: t2mi decoder handler
program_number @in: plp id
@return: TSRECV_ERR_OK - success, <0 - failed (error code provided):
```

```
         TSRECV_ERR_BAD_INPUT_PARAMETERS - bad input parameters
         TSRECV_ERR_UNEXPECTED_CALL - unexpexted call
note: plp can be selected only after t2mi scanning is done */
int32_t t2midecoder_plp_select( H_T2MIDECODER handler,
                                uint8_t plp )
{
    int32_t err = 0;

    IF_NULL_SETSTATUS_GOTO( handler, err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND );

    IF_FAILED_SETSTATUS_GOTO( handler->state != WAITING_CLOSURE, err, TSRECV_ERR_UNEXPECTED_CALL, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO(handler->scan_completed, err, TSRECV_ERR_UNEXPECTED_CALL, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO( is_plp_present(handler, plp), err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND );

    send_msg_to_thread( handler, &(X_THREAD_MSG){ .cmd = SELECT_PLP,
                                                  .data.plp = plp } );
FUNCEND:
    if( err != FAPI_OK ) {
        DEBUGOUT( 1, "failed to select plp%u - (%s)\n", syserr_getmsg( err ) );
    }
    else {
        DEBUGOUT( 2, "plp selected\n" );
    }

    return err;
}

/* deselect plp for decoding
handler @in: t2mi decoder handler
@return: TSRECV_ERR_OK - success, <0 - failed (error code provided):
         TSRECV_ERR_BAD_INPUT_PARAMETERS - bad input parameters
         TSRECV_ERR_UNEXPECTED_CALL - unexpexted call */
int32_t t2midecoder_plp_deselect( H_T2MIDECODER handler)
{
    int32_t err = 0;

    IF_NULL_SETSTATUS_GOTO(handler, err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO( handler->state != WAITING_CLOSURE, err, TSRECV_ERR_UNEXPECTED_CALL, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO(handler->selected_plp != PLP_NOT_SELECTED, err, TSRECV_ERR_UNEXPECTED_CALL,
FUNCEND);

    send_msg_to_thread (handler, &(X_THREAD_MSG){ .cmd = DESELECT_PLP } );

FUNCEND:
    if( err != FAPI_OK ) {
        DEBUGOUT( 1, "failed to deselect plp%u - (%s)\n", syserr_getmsg( err ) );
    }
    else {
        DEBUGOUT( 2, "plp deselected\n" );
    }

    return err;
}

/* t2mi decoder pause
handler @in: t2mi decoder handler
```

```
@return: TSRECV_ERR_OK - success, <0 - failed (error code provided) */
int32_t t2midecoder_pause( H_T2MIDECODER handler )
{
    int32_t err = TSRECV_ERR_OK;

    IF_NULL_SETSTATUS_GOTO(handler, err, TSRECV_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

    IF_FAILED_SETSTATUS_GOTO( handler->state != WAITING_CLOSURE, err, TSRECV_ERR_UNEXPECTED_CALL, FUNCEND);

    send_msg_to_thread( handler, &(X_THREAD_MSG){ .cmd = PAUSE } );

FUNCEND:
    if( err != FAPI_OK ) {
        DEBUGOUT( 1, "failed to pause(%s)\n", syserr_getmsg( err ) );
    }
    else {
        DEBUGOUT( 2, "plp paused\n" );
    }
    return err;
}

/* get information about t2mi PLPs
handler @in: t2midecoder handelr
@return: information (NULL - no information available)
note: data and pointer itself are valid only after t2mi scanning is done
 */
const tTV_DVBT2_PLP* t2midecoder_get_plp_info( H_T2MIDECODER handler )
{
    return ( handler != NULL && handler->scan_completed)? handler->info_plp : NULL;
}

/* get l1pre information
handler @in: t2midecoder handler
@return: information (NULL - no information available)
note: data and pointer itself are valid only after t2mi scanning is done
 */
const tTV_DVBT2_L1PRE* t2midecoder_get_l1pre_info( H_T2MIDECODER handler )
{
    return ( handler != NULL && handler->scan_completed)? &handler->info_L1PRE : NULL;
}

/* get l1post information
handler @in: t2midecoder handler
@return: information (NULL - no information available)
note: data and pointer itself are valid only after t2mi scanning is done
 */
const tTV_DVBT2_L1POST* t2midecoder_get_l1post_info( H_T2MIDECODER handler )
{
    return ( handler != NULL && handler->scan_completed)? &handler->info_L1POST : NULL;
}

static void send_msg_to_thread( H_T2MIDECODER handler, X_THREAD_MSG* msg )
{
    handler->is_appeared_msg++;

    RTOS_SetMailqueue(handler->thread_queue, msg,
                      0, RTOS_NO_SUSPEND );
    RTOS_GetSemaphore( handler->cmd_sem, RTOS_SUSPEND );
}
```

```
static void thread( void* opt )
{
    X_T2MIDECODER_CONTEXT* hdl = (X_T2MIDECODER_CONTEXT*)opt;
    X_CONS_CONTEXT* cons_cnt = &hdl->cons_context;
    hdl->state = SCANNING;

    IF_FAILED_GOTO( tsfifo_set_consumer_cfg( hdl->cfg.hstream_in, &g_stream_cfg ) == TSFIFO_ERR_OK, FUNCEND );

    do {
        X_THREAD_MSG msg;
        uint32_t is_cmd = RTOS_GetMailqueue(hdl->thread_queue, &msg, RTOS_NO_SUSPEND);

        if( is_cmd ) {

            switch( msg.cmd )
            {
                case START_SCAN:

                    hdl->state = SCANNING;
                    break;

                case PLAY:

                    hdl->state = PLAYING;
                    break;

                case CLOSE:

                    goto FUNCEND;

                case PAUSE:

                    hdl->state = STOPPED;
                    break;

                case SELECT_PLP:

                    hdl->selected_plp = msg.data.plp;
                    notify_brief( hdl, T2MIDECODER_EVENT_PLP_SELECTED );
                    break;

                case DESELECT_PLP:

                    hdl->selected_plp = PLP_NOT_SELECTED;
                    hdl->state = STOPPED;
                    break;

                default:

                    syscrash( "unknown command in t2midecoder" );
                    break;
            }

            hdl->is_appeared_msg--;

            RTOS_SetSemaphore( hdl->cmd_sem, RTOS_NO_SUSPEND );

        }

        if( hdl->state != WAITING_CLOSURE && !cons_cnt->data ) {
```

```
    int32_t fetch_state = fetch_data( hdl->cfg.hstream_in, cons_cnt );
    if( fetch_state < 0 )
    {
        DEBUGOUT(1, "error fetch data from tsfifo -> %d\n", fetch_state);
        prepare_to_close( hdl );
        notify_err( hdl );
    }
    if( fetch_state == TS_FIFO_TERMINATE )
    {
        DEBUGOUT(3, "tsfifo terminated\n");
        if( cons_cnt->data ) {
            int32_t err = tsfifo_pop( hdl->cfg.hstream_in, cons_cnt->data );
            cons_cnt->data = NULL;
            if( err < 0 ) {
                DEBUGOUT(1, "error pop data tsfifo -> %d\n", err);
                notify_err( hdl );
            }
        }
        prepare_to_close( hdl );
    }
}

switch( hdl->state ) {
    case SCANNING:

        scanning( hdl );
        break;

    case STOPPED:

        stopped( hdl );
        break;

    case PLAYING:

        playing( hdl );
        break;

    case WAITING_CLOSURE:

        wait_cmd( hdl );
        break;

    default:

        syscrash( "unknown state in t2midecoder" );
        break;
}

if( hdl->thread_err < 0 ) {
    react_to_thread_error( hdl );
}

if( cons_cnt->data &&
    (!cons_cnt->slice_bytes_to_go || hdl->state == WAITING_CLOSURE) )
{

    int32_t err = tsfifo_pop( hdl->cfg.hstream_in, cons_cnt->data );
    cons_cnt->data = NULL;
```

```
            if( err < 0 ) {
                DEBUGOUT(1, "error pop data tsfifo -> %d\n", err);
                prepare_to_close( hdl );
                notify_err( hdl );
                continue;
            }

            wait_slice_time( cons_cnt );
        }


    } while( true );

FUNCEND:
    RTOS_SetSemaphore( hdl->cmd_sem, RTOS_NO_SUSPEND );

    RTOS_DestroyThread( RTOS_GetThread( ) );
}

// notify user about event
// handler @in: TS receiver handler
// event @in: event to notify user with
static void notify( H_T2MIDECODER handler,
                    E_T2MIDECODER_EVENT_ID id,
                    const X_T2MIDECODER_EVENT_DATA* data )
{
    if( handler->cfg.cb != NULL )
    {
        const X_T2MIDECODER_EVENT event =
        {
            .id = id,
            .data = *data
        };

        handler->cfg.cb( handler, &event, handler->cfg.opaq );
    }
}


// notify user about event (without data)
// handler @in: t2midecoder handler
// id @in: event ID to notify with
static void notify_brief( H_T2MIDECODER handler, E_T2MIDECODER_EVENT_ID id )
{
    notify( handler, id, &(const X_T2MIDECODER_EVENT_DATA){ 0 } );
}

//realization of the scan state of the t2mi stream
static void scanning( H_T2MIDECODER handler )
{
    X_CONS_CONTEXT* cons_cnt = &handler->cons_context;

    while( cons_cnt->slice_bytes_to_go )
    {
        if( handler->is_appeared_msg )
            break;

        int32_t t2mi_ready = t2miparser_get_t2mi_packet( handler->h_t2miparser );

        if( t2mi_ready < 0 ) {
            handler->thread_err = t2mi_ready;
```

```
            DEBUGOUT(3, "scanning error -> %d\n", handler->thread_err);
            break;
        }

        if( t2mi_ready && !handler->scan_completed ) {
            const uint8_t* t2mi = t2miparser_get_t2mi( handler->h_t2miparser );

            handler->thread_err = get_info_from_t2mi_packet( handler , t2mi );

            if( handler->thread_err < 0 ) {
                DEBUGOUT(3, "get info from packet error -> %d\n", handler->thread_err);
                break;
            }

            if( handler->scan_completed ) {
                notify_brief( handler, T2MIDECODER_EVENT_SCAN_FINISHED );
            }
        }

    }
}

//realization of the play state of the t2mi stream
static void playing( H_T2MIDECODER handler )
{
    X_T2MI_SEND_CONTEXT* send_cnt = &handler->send_context;
    X_CONS_CONTEXT* cons_cnt = &handler->cons_context;

    while( cons_cnt->slice_bytes_to_go )
    {

        switch( send_cnt->playing_state ) {
            case GOING_TO_T2MI:

                go_to_t2mi_data( handler );

                break;
            case READING_T2MI_HEAD:

                reading_t2mi_head(send_cnt, cons_cnt);

                break;
            case READING_BBF_HEAD:

                reading_bbf_head(send_cnt, cons_cnt, (uint8_t)handler->selected_plp);

                break;
            case T2MI_SYNC:

                t2mi_sync(send_cnt, cons_cnt);

                break;
            case SENDING:

                sending(send_cnt, cons_cnt);

                break;
            case SKIPPING:

                skipping(send_cnt, cons_cnt);
```

```
                break;
        }

        if( send_cnt->err < 0 ) {
            handler->thread_err = send_cnt->err;
            send_cnt->err = 0;
            DEBUGOUT(3, "error playing -> %d\n", handler->thread_err);
            break;
        }
    }

}

//realization of the stop state t2midecoder's
static void stopped( H_T2MIDECODER handler )
{
    X_CONS_CONTEXT* cons_cnt = &handler->cons_context;

    while( cons_cnt->slice_bytes_to_go )
    {
        if(handler->is_appeared_msg)
            break;

        int32_t t2mi_ready = t2miparser_get_t2mi_packet( handler->h_t2miparser );

        if( t2mi_ready < 0 ) {
            handler->thread_err = t2mi_ready;
            DEBUGOUT(3, "error stopped -> %d\n", handler->thread_err);
            break;
        }
    }
}

static uint32_t get_min(uint32_t a, uint32_t b)
{
    return ( a < b )? a : b;
}

//get information about cur PLP from bbf packages
static void get_info_from_packet_bbf( H_T2MIDECODER handler, const uint8_t* t2mi )
{
    int plp_index;
    X_T2MI_SCAN_CONTEXT* scan_cnt = &handler->scan_context;

    for( plp_index = 0; plp_index < handler->info_L1POST.NumPLPs; plp_index++ )
    {
        if(handler->info_plp[plp_index].Id == t2miparser_get_plp( t2mi ) )
        {
            break;
        }
    }

    if( !( scan_cnt->bbf_scan_state & (1 << plp_index) ) ) {
        handler->info_plp[plp_index].Npd = t2miparser_get_npd( t2mi );

        handler->info_plp[plp_index].Issy = t2miparser_get_issyi( t2mi );

        scan_cnt->bbf_scan_state |= (1 << plp_index);
        scan_cnt->bbf_info_counter++;
```

```
         DEBUGOUT(3, "plp%d info gotten\n", plp_index);
    }
}

static void free_hdl(H_T2MIDECODER hdl)
{
   if( hdl != NULL ) {

        if(hdl->cmd_sem != NULL) {
            RTOS_DestroySemaphore(hdl->cmd_sem);
        }

        if(hdl->cons_context.dma_sem != NULL) {
            RTOS_DestroyMailqueue(hdl->thread_queue);
        }

        if(hdl->h_t2miparser != NULL) {
            t2miparser_close( hdl->h_t2miparser );
        }

        if( hdl->info_plp != NULL ) {
            FAPI_SYS_FREE( hdl->info_plp );
        }

        if( hdl->send_context.mpeg_packets_buf != NULL ) {
            FAPI_MMU_Free( FAPI_MMU_HeapHandleSys1, hdl->send_context.mpeg_packets_buf );
        }

        FAPI_SYS_FREE(hdl);
    }
}

//refresh sending, t2miparser
static int32_t refresh_thread_context(H_T2MIDECODER handler)
{
    int32_t err = t2miparser_reset( handler->h_t2miparser );
    IF_FAILED_GOTO(err == 0, FUNCEND);

    err = reset_sending( &handler->cons_context, &handler->send_context );
    IF_FAILED_GOTO(err == 0, FUNCEND);

FUNCEND:
    if( err != 0 ) {
        DEBUGOUT(3, "error refresh thread -> %d\n", err );
    }
    return err;
}

/* is plp present in info
hdl @in: t2midecoder handler
plp @in: plp
@return: true - present
         false - not present */
static bool is_plp_present( H_T2MIDECODER hdl, uint8_t plp )
{
    for(int i = 0; i < hdl->info_L1POST.NumPLPs; i++ ) {
        if( plp == hdl->info_plp[i].Id ) {

            return true;
```

```
        }
    }

    return false;
}


static bool read_tsfifo_cb( uint8_t* read_buf, uint8_t bytes, void* aux )
{
    int32_t err = FAPI_OK;
    (void)aux;

    if( read_buf ) {
        err = read_tsfifo( aux, read_buf, bytes);
    }
    else {
        err = skip_tsfifo( aux, bytes );
    }

    return !(err < 0);
}

/* get reference to tsd buf with linear auto mode
tsd_ind @in: number of tsd(FAPI_TSD0, FAPI_TSD1, FAPI_TSD2, FAPI_TSD3)
@return: reference to tsd buf */
static uint32_t get_tsd_packet_buffer( uint32_t tsd_ind ) {

    static const uint32_t lookup[] =
    {
        [FAPI_TSD0] = FREG_TSD0_LINEAR_PACKET_AUTO0,
        [FAPI_TSD1] = FREG_TSD1_LINEAR_PACKET_AUTO0,
        [FAPI_TSD2] = FREG_TSD2_LINEAR_PACKET_AUTO0,
        [FAPI_TSD3] = FREG_TSD3_LINEAR_PACKET_AUTO0,
    };

    return lookup[tsd_ind];
}

// read from tsfifo in special buffer in send context
// bytes_should_be @in: indicate amount of bytes which should be in buffer
// @return: -1 - if failed
//          another - done
static int32_t read_t2mi( X_CONS_CONTEXT* cnt_cons, X_T2MI_SEND_CONTEXT* cnt_send, unsigned int
bytes_should_be, bool skip )
{
    int32_t err = FAPI_OK;

    uint8_t bytes = (uint8_t)get_min( cnt_send->mpeg_bytes_to_go,
                                      bytes_should_be - cnt_send->t2mi_bytes_done );
    if( bytes ){
        if( !skip ) {
            err = read_tsfifo( cnt_cons, (uint8_t*)cnt_send->t2mi_headers + cnt_send->t2mi_bytes_done, bytes );
        }
        else {
            err = skip_tsfifo( cnt_cons, bytes );
        }
        IF_FAILED_GOTO( err >= 0, FUNCEND);

        cnt_send->mpeg_bytes_to_go -= bytes;
        cnt_send->t2mi_bytes_done += bytes;
```

```
    }

FUNCEND:
    return err;
}

//reset sending context
static int32_t reset_sending( X_CONS_CONTEXT* cnt_cons, X_T2MI_SEND_CONTEXT* cnt )
{
    int32_t err = 0;
    skip_tsfifo( cnt_cons, cnt->mpeg_bytes_to_go );
    cnt_cons->slice_bytes_to_go = 0;
    cnt_cons->cur_ptr = 0;
    cnt->mpeg_bytes_to_go = 0;
    cnt->t2mi_bytes_done = 0;
    cnt->t2mi_data_length = 0;
    cnt->t2mi_length = 0;
    cnt->remaning_bytes_mpeg_packet_to_send = -1;
    cnt->prev_counter_t2mi = -1;
    cnt->buf_cur_ptr = 0;
    cnt->mpeg_send_sync = false;
    cnt->playing_state = GOING_TO_T2MI;

    return err;
}


static bool check_sync_t2mi( X_T2MI_SEND_CONTEXT* cnt, const uint8_t* t2mi )
{
    IF_FAILED_RETURN_EXPR( cnt->prev_counter_t2mi >= 0, true );

    IF_FAILED_RETURN_EXPR( ( t2mi[1] == (cnt->prev_counter_t2mi + 1) ||
                             ( t2mi[1] == 0 && cnt->prev_counter_t2mi == 255 ) ),
                           false );

    return true;
}

//send mpeg packets to dma
// note: firstly func accumulate dma buffer(size --> SIZE_OF_DMA_BUF) and then dma send is initialized
static int32_t send_to_tsd( X_T2MI_SEND_CONTEXT* send_cnt, uint32_t bytes, X_CONS_CONTEXT* cons_cnt) {
    int32_t err = 0;

    while( bytes ) {

        if( send_cnt->remaning_bytes_mpeg_packet_to_send == 187 ) {
           send_cnt->buf_cur_ptr++;
        }

        uint32_t bytes_by_loop = get_min( bytes, (uint32_t)send_cnt->remaning_bytes_mpeg_packet_to_send );

        err = read_tsfifo( cons_cnt,
                           (uint8_t*)(send_cnt->mpeg_packets_buf + send_cnt->buf_cur_ptr),
                           (uint8_t)bytes_by_loop );
        IF_FAILED_GOTO(err == 0, FUNCEND);

        send_cnt->buf_cur_ptr += bytes_by_loop;
        bytes -= bytes_by_loop;
        send_cnt->remaning_bytes_mpeg_packet_to_send -= bytes_by_loop;
```

```
        if( !(send_cnt->remaning_bytes_mpeg_packet_to_send) ) {

            send_cnt->remaning_bytes_mpeg_packet_to_send = 187;
        }

        if( send_cnt->buf_cur_ptr == SIZE_OF_DMA_BUF ) {
            X_DMA_START_CFG dma_cfg =
            {
                .aux = cons_cnt,
                .finishCallback = dma_cb,
                .data_len = SIZE_OF_DMA_BUF,
                .src = send_cnt->mpeg_packets_buf,
                .wait_cycles = 0,
            };

            err = dma_start_ts_packet_copy( &dma_cfg );
            if( err != TSGRAB_ERR_OK )
            {
                DEBUGOUT(1, "error dma\n");
            }

            (void)RTOS_GetSemaphore( cons_cnt->dma_sem, RTOS_SUSPEND );
            send_cnt->buf_cur_ptr = 0;
        }

    }

FUNCEND:
    return err;
}

static void go_to_t2mi_data( H_T2MIDECODER hdl )
{
    X_T2MI_SEND_CONTEXT* cnt_send = &hdl->send_context;
    X_CONS_CONTEXT* cnt_cons = &hdl->cons_context;

    cnt_send->err = 0;

    if( !cnt_send->mpeg_bytes_to_go ) {
        int32_t t2mi_bytes_recived = 0;

        while ( t2mi_bytes_recived == 0 && cnt_cons->slice_bytes_to_go ) {
            t2mi_bytes_recived = t2miparser_skip_to_t2mi_data( hdl->h_t2miparser );
        }

        if( t2mi_bytes_recived < 0 ) {
            cnt_send->err = t2mi_bytes_recived;
            DEBUGOUT(3, "error going to t2mi data -> %d\n", t2mi_bytes_recived);
            return;
        }
        if(t2mi_bytes_recived > 0) {
            cnt_send->mpeg_bytes_to_go = (uint8_t)t2mi_bytes_recived;
        }
    }

    if( cnt_send->t2mi_bytes_done < BYTES_T2MI_HEADER ) {
        cnt_send->playing_state = READING_T2MI_HEAD;
        return;
    }
```

```
    if( t2miparser_get_t2mi_type(cnt_send->t2mi_headers) == PACKET_TYPE_OF_BBF ) {
        if( cnt_send->t2mi_bytes_done < BYTES_T2MI_HEADERS ) {
            cnt_send->playing_state = READING_BBF_HEAD;
            return;
        }
        if( t2miparser_get_plp( cnt_send->t2mi_headers ) == hdl->selected_plp ) {
            if( !cnt_send->mpeg_send_sync ) {
                cnt_send->playing_state = T2MI_SYNC;
                return;
            }
            if( cnt_send->t2mi_data_length ) {
                cnt_send->playing_state = SENDING;
                return;
            }
        }
    }
    cnt_send->playing_state = SKIPPING;
}

static void reading_t2mi_head(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons)
{
    cnt->err = read_t2mi( cnt_cons, cnt, BYTES_T2MI_HEADER, false );
    IF_FAILED_GOTO( cnt->err >= 0, FUNCEND );

    if( cnt->t2mi_bytes_done == BYTES_T2MI_HEADER ) {

        if( !check_sync_t2mi(cnt, cnt->t2mi_headers) ) {

            cnt->err = T2MIPARSER_ERR_DESYNCTHRONIZATION;
            DEBUGOUT( 3, "t2mi stream desyncthronization, prev counter t2mi - %d, cur counter t2mi - %d\n",
                        cnt->prev_counter_t2mi,
                        cnt->t2mi_headers[1] );
            goto FUNCEND;
        }
        cnt->prev_counter_t2mi = cnt->t2mi_headers[1];

        cnt->t2mi_length = (uint16_t)t2miparser_get_length( cnt->t2mi_headers ) - BYTES_T2MI_HEADER;

        if( t2miparser_get_t2mi_type( cnt->t2mi_headers ) != PACKET_TYPE_OF_BBF ) {
            cnt->playing_state = SKIPPING;
            return;
        }

        cnt->playing_state = READING_BBF_HEAD;
    }


FUNCEND:
    if( cnt->err < 0 ) {
        DEBUGOUT(3, "error reading t2mi head -> %d\n", cnt->err);
        return;
    }

    if( !cnt->mpeg_bytes_to_go ) {
        cnt->playing_state = GOING_TO_T2MI;
        return;
    }

}
```

```
static void reading_bbf_head(X_T2MI_SEND_CONTEXT* cnt, X_CONS_CONTEXT* cnt_cons, uint8_t selected_plp)
{
    cnt->err = read_t2mi( cnt_cons, cnt, BYTES_T2MI_HEADERS, false );
    IF_FAILED_GOTO( cnt->err >= 0, FUNCEND );

    if( cnt->t2mi_bytes_done == BYTES_T2MI_HEADERS )
    {
        cnt->t2mi_length -= BYTES_BBF_HEADERS;

        if( t2miparser_get_plp( cnt->t2mi_headers ) == selected_plp ) {
            cnt->t2mi_data_length = t2miparser_get_dfl( cnt->t2mi_headers ) / 8;
        }
        else {
            cnt->playing_state = SKIPPING;
            return;
        }

        if( !cnt->mpeg_send_sync ) {
            cnt->playing_state = T2MI_SYNC;
            return;
        }

        cnt->playing_state = SENDING;
    }

FUNCEND:
    if( cnt->err < 0 ){
        DEBUGOUT(3, "error reading t2mi bbf head -> %d\n", cnt->err);
        return;
    }

    if( !cnt->mpeg_bytes_to_go ) {
        cnt->playing_state = GOING_TO_T2MI;
        return;
    }

}

static void t2mi_sync(X_T2MI_SEND_CONTEXT* cnt_send, X_CONS_CONTEXT* cnt_cons)
{
    uint32_t syncd = t2miparser_get_syncd( cnt_send->t2mi_headers );

    cnt_send->err = read_t2mi( cnt_cons, cnt_send,
                (syncd / 8) + BYTES_T2MI_HEADERS, true);
    IF_FAILED_GOTO( cnt_send->err >= 0, FUNCEND );

    if( cnt_send->t2mi_bytes_done == BYTES_T2MI_HEADERS + (syncd / 8) ) {
        cnt_send->remaning_bytes_mpeg_packet_to_send = 187;
        cnt_send->t2mi_data_length -= (syncd / 8);
        cnt_send->t2mi_length -= (syncd / 8);
        cnt_send->mpeg_send_sync = true;
        cnt_send->playing_state = SENDING;
    }

FUNCEND:
    if( cnt_send->err < 0 ) {
        DEBUGOUT(3, "error sync t2mi -> %d\n", cnt_send->err);
        return;
    }
```

```
    if( !cnt_send->mpeg_bytes_to_go ) {
        cnt_send->playing_state = GOING_TO_T2MI;
        return;
    }
}

static void sending(X_T2MI_SEND_CONTEXT* cnt_send, X_CONS_CONTEXT* cnt_cons)
{
    if( cnt_send->t2mi_data_length ) {
        uint32_t bytes = get_min( cnt_send->t2mi_data_length, cnt_send->mpeg_bytes_to_go );

        cnt_send->t2mi_data_length -= bytes;
        cnt_send->mpeg_bytes_to_go -= bytes;
        cnt_send->t2mi_length -= bytes;

        cnt_send->err = send_to_tsd(cnt_send, bytes, cnt_cons);
    }

    if( cnt_send->err < 0 ){
        DEBUGOUT(3, "error sending t2mi -> %d\n", cnt_send->err);
        return;
    }

    if( cnt_send->t2mi_data_length == 0 ) {
        cnt_send->playing_state = SKIPPING;
        return;
    }

    if( !cnt_send->mpeg_bytes_to_go ) {
        cnt_send->playing_state = GOING_TO_T2MI;
        return;
    }
}

static void skipping(X_T2MI_SEND_CONTEXT* cnt_send, X_CONS_CONTEXT* cnt_cons)
{
    if( cnt_send->t2mi_length && cnt_send->mpeg_bytes_to_go ) {
        uint32_t bytes = get_min( cnt_send->t2mi_length, cnt_send->mpeg_bytes_to_go );

        cnt_send->mpeg_bytes_to_go -= bytes;
        cnt_send->t2mi_length -= bytes;
        cnt_send->err = skip_tsfifo( cnt_cons, (uint8_t)bytes );

    }
    else if( cnt_send->mpeg_bytes_to_go ) {
        uint32_t bytes = cnt_send->mpeg_bytes_to_go;

        cnt_send->err = skip_tsfifo( cnt_cons, (uint8_t)bytes );
        cnt_send->mpeg_bytes_to_go -= bytes;
    }

    if( cnt_send->err < 0 ) {
        DEBUGOUT(3, "error skipping t2mi -> %d\n", cnt_send->err);
        return;
    }

    if( !cnt_send->mpeg_bytes_to_go ) {

        cnt_send->playing_state = GOING_TO_T2MI;
```

```
        if( cnt_send->t2mi_length == 0  ) {
            cnt_send->t2mi_bytes_done = 0;
        }

        return;
    }

    if( cnt_send->t2mi_length == 0  ) {

        cnt_send->playing_state = READING_T2MI_HEAD;
        cnt_send->t2mi_bytes_done = 0;
        return;
    }

}

//reacting on error
//note: this func need to fix desyncthronization error
static void react_to_thread_error( H_T2MIDECODER hdl)
{
    if( hdl->thread_err == T2MIPARSER_ERR_DESYNCTHRONIZATION )
    {
        hdl->thread_err = refresh_thread_context( hdl );
        DEBUGOUT(3, "stream was refreshed with err code - %d\n", hdl->thread_err);
    }
    if( hdl->thread_err < 0 && hdl->state != WAITING_CLOSURE ) {
        DEBUGOUT(1, "fix thread error failed -> %d\n", hdl->thread_err);
        prepare_to_close( hdl );
        notify_err( hdl );
    }
}

static int32_t read_tsfifo( X_CONS_CONTEXT* cnt_cons, uint8_t* read_buf, uint8_t bytes )
{
    if( cnt_cons->slice_bytes_to_go >= bytes ) {

        memcpy( read_buf, &cnt_cons->data[cnt_cons->cur_ptr], bytes);

        cnt_cons->cur_ptr += bytes;
        cnt_cons->slice_bytes_to_go -= bytes;

        return 0;
    }
    else {
        return -1;
    }
}
static int32_t skip_tsfifo( X_CONS_CONTEXT* cnt_cons, uint8_t bytes )
{
    if( cnt_cons->slice_bytes_to_go >= bytes ) {

        cnt_cons->slice_bytes_to_go -= bytes;
        cnt_cons->cur_ptr += bytes;
        return 0;
    }
    else {
        return -1;
    }
}
```

```
static int32_t fetch_data( H_TSFIFO h_tsfifo, X_CONS_CONTEXT* cons_cnt ) {
    X_TSFIFO_STATE state;

    if( !cons_cnt->slice_bytes_to_go ) {

        cons_cnt->slice_bytes_to_go = tsfifo_fetch_ex( h_tsfifo, (const void**)&cons_cnt->data, 100, &state );
        cons_cnt->cur_ptr = 0;

        if( cons_cnt->slice_bytes_to_go < 0 ) {
            return cons_cnt->slice_bytes_to_go;
        }
        else if( state.f_terminate ) {
            cons_cnt->total_slice_time = 0;

            return TS_FIFO_TERMINATE;
        }
        else if ( cons_cnt->slice_bytes_to_go > 0 ) {
            cons_cnt->receive_timestamp = GET_TIMESTAMP_US();

            cons_cnt->total_slice_time = calculate_total_time( &state, cons_cnt );
        }

    }
    return 0;
}

//waiting for command in the thread
static void wait_cmd( H_T2MIDECODER hdl )
{
    X_THREAD_MSG msg;

    RTOS_GetMailqueue(hdl->thread_queue, &msg,
                      RTOS_SUSPEND);
    RTOS_SetMailqueue(hdl->thread_queue, &msg,
                      0, RTOS_NO_SUSPEND );
}


static uint32_t calculate_total_time( X_TSFIFO_STATE* stream_state, X_CONS_CONTEXT* cons_cnt )
{
    float corrKfc = 1 + ( ( (int32_t) stream_state->fifo_population_pct -
                            (int32_t) DESIRABLE_FILL_LEVEL_PCT ) / 100.0f ) * 0.2f +
                      ( ( (int32_t)stream_state->fifo_population_pct - (int32_t)cons_cnt->prev_population_pct
) / 100.0f ) * 0.8f;

    // защита от резкого падения скорости из-за которой буфер может переполниться
    if( corrKfc < 0.8f ) {
        corrKfc = 0.8f;
    }

    if( stream_state->fifo_population_pct > 60 || stream_state->fifo_population_pct < 40 ) {
        DEBUGOUT(3, "unstable buffer fullness -> %d\n", stream_state->fifo_population_pct);
    }

    cons_cnt->stream_rate_bps = (uint32_t)(cons_cnt->stream_rate_bps * corrKfc);

    cons_cnt->prev_population_pct = stream_state->fifo_population_pct;

    return (uint32_t)( ( (uint64_t)cons_cnt->slice_bytes_to_go * 8 * 1000000) /
                       ( cons_cnt->stream_rate_bps ) );
```

```
}

//wait for the time during which the slice of mpeg stream should be processed
static void wait_slice_time( X_CONS_CONTEXT* cons_cnt )
{
    cons_cnt->send_timestamp = GET_TIMESTAMP_US();

    uint32_t time_to_fetch = cons_cnt->total_slice_time - (cons_cnt->send_timestamp - cons_cnt-
>receive_timestamp);
    if( (int32_t)time_to_fetch > 1000 ) {
        RTOS_Sleep( time_to_fetch / 1000 );
    }

}

static void dma_cb ( FAPI_DMA_RequestT* dma_req, void* opt )
{
    (void)dma_req;
    X_CONS_CONTEXT* cnt = (X_CONS_CONTEXT*)opt;
    (void)RTOS_SetSemaphore( cnt->dma_sem, RTOS_NO_SUSPEND );
}

static int32_t dma_start_ts_packet_copy( const X_DMA_START_CFG *cfg )
{
    int32_t err = FAPI_OK;
    X_CONS_CONTEXT* cnt = (X_CONS_CONTEXT*)cfg->aux;
    FAPI_DMA_RequestT* dma_req = NULL;

    if( cfg == NULL )
    {
        err = FAPI_ERR_BAD_PARAMETER;
        goto L_FUNCEND;
    }

    if( err != FAPI_OK )
    {
        goto L_FUNCEND;
    }

    dma_req = FAPI_DMA_AllocateRequest( FAPI_DMA_FEATURE_SDRAM_COPY, 1 );

    if( !dma_req )
    {
        err = FAPI_ERR_OUT_OF_MEMORY;
        goto L_FUNCEND;
    }

    dma_req->preEnableCallback                      = 0;
    dma_req->preEnableOptDataPtr                    = 0;
    dma_req->postEnableCallback                     = 0;
    dma_req->finishCallback                         = cfg->finishCallback;
    dma_req->finishOptDataPtr                       = cfg->aux;
    dma_req->changeCallback                         = 0;
    dma_req->optData                                = 0;
    dma_req->autoRestart                            = 0;
    dma_req->splitCount                             = 0;
    dma_req->finishCallbackExecInIsr                = 1;
    dma_req->channelConfig.chConfig_Enable          = 1;
    dma_req->channelConfig.chConfig_LinkedListEnable = 0;
    dma_req->channelConfig.chConfig_TsPacketControl  = ( 0x1 << ( cnt->tsd_index & 0xF ) );
```

```
    dma_req->channelConfig.chLength                 = cfg->data_len;
    dma_req->channelConfig.chLladdr                 = 0;
    dma_req->channelConfig.chRdaddr_OffsetAddress   = (uint32_t)cfg->src;
    dma_req->channelConfig.chRdline_Lines           = 0x3FF;
    dma_req->channelConfig.chRdline_LoopIncrement   = 0;
    dma_req->channelConfig.chRdinc_LineLength       = 0xFFF;
    dma_req->channelConfig.chRdinc_LineIncrement    = 0;
    dma_req->channelConfig.chRdlpaddr_LoopAddress   = 0;
    dma_req->channelConfig.chWraddr_OffsetAddress   = get_tsd_packet_buffer( cnt->tsd_index );
    dma_req->channelConfig.chWrline_Lines           = 2;
    dma_req->channelConfig.chWrline_LoopIncrement   = 0;
    dma_req->channelConfig.chWrinc_LineLength       = TSPKT_SIZE_B;
    dma_req->channelConfig.chWrinc_LineIncrement    = 0;
    dma_req->channelConfig.chWrlpaddr_LoopAddress   = 0;
    dma_req->channelConfig.chConfig_RdEndianSwap    = 0;
    dma_req->channelConfig.chConfig_WrPeripheralAddress =
                                    FAPI_DMA_FEATURE_INDEX_REGISTER_COPY;
    dma_req->channelConfig.chWait_EnableOnRead  = 0;
    dma_req->channelConfig.chWait_EnableOnWrite = 1;
    dma_req->channelConfig.chConfig_WrEndianSwap = 1;
    dma_req->channelConfig.chWait_Cycles = cfg->wait_cycles;

    if( FAPI_DMA_SendRequest( dma_req ) != FAPI_OK )
    {
        FAPI_DMA_ReleaseRequest( dma_req );
    }

L_FUNCEND:

    return err;
}

//get information about t2mi from l1cur package
static int32_t get_info_from_packet_l1cur( H_T2MIDECODER handler, const uint8_t* t2mi_packet )
{
    int32_t err = TSRECV_ERR_OK;

    handler->info_L1PRE = get_l1pre_from_packet( t2mi_packet );

    handler->info_L1POST = get_l1post_from_packet( t2mi_packet );

    handler->info_plp = FAPI_SYS_MALLOC( handler->info_L1POST.NumPLPs * sizeof(tTV_DVBT2_PLP) );
    IF_NULL_SETSTATUS_GOTO(handler->info_plp, err, TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);
    memset( handler->info_plp, 0, handler->info_L1POST.NumPLPs * sizeof(tTV_DVBT2_PLP));

    for( uint8_t i = 0; i < handler->info_L1POST.NumPLPs; i++ )
    {
        err = get_plp_info( t2mi_packet, i, &handler->info_plp[i]);
        IF_FAILED_GOTO( err == 0, FUNCEND);
    }

FUNCEND:
    if( err != TSRECV_ERR_OK) {
        if( handler->info_plp != NULL ) {
            FAPI_SYS_FREE( handler->info_plp );
            handler->info_plp = NULL;
        }
    }

    return err;
```

```
}

//get information about t2mi from random packeges
static int32_t get_info_from_t2mi_packet( H_T2MIDECODER handler , const uint8_t* t2mi )
{
    int32_t err = 0;
    X_T2MI_SCAN_CONTEXT* scan_cnt = &handler->scan_context;
    if( !(scan_cnt->scan_state & L1CUR_FOUND) ) {
        if( t2miparser_get_t2mi_type( t2mi ) ==
            PACKET_TYPE_OF_L1CURRENT ) {

            err = get_info_from_packet_l1cur( handler, t2mi );
            IF_FAILED_RETURN_EXPR( err >= 0, err );

            IF_FAILED_RETURN_EXPR( handler->info_L1POST.NumPLPs, -1 );

            scan_cnt->scan_state |= L1CUR_FOUND;
            DEBUGOUT(3, "l1cur info gotten\n")
        }
    }
    else if( t2miparser_get_t2mi_type( t2mi ) ==
             PACKET_TYPE_OF_BBF )
    {
        get_info_from_packet_bbf( handler, t2mi );

        if( scan_cnt->bbf_info_counter == ( handler->info_L1POST.NumPLPs ) ) {
            handler->scan_completed = true;
        }
    }

    return 0;
}

static void prepare_to_close( H_T2MIDECODER hdl )
{
    (void)tsfifo_set_consumer_shutdown_ready( hdl->cfg.hstream_in );
    hdl->state = WAITING_CLOSURE;
}

static void notify_err( H_T2MIDECODER hdl )
{
    notify_brief( hdl, T2MIDECODER_EVENT_ERR );
    hdl->thread_err = 0;
}
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД МОДУЛЯ T2MIPARSER

Листинг 2 – Исходный код t2miparser

```
#define DEBUG                          3
//if need to check counter in mpeg stream
#define CHECK_SYNC_MPEG
#include "t2miparser.h"

#include <fapi/sys_services.h>
#include "../../../lib/utils/macroCheckers.h"
#include "../../../lib/mpeglib/include/TS_types.h"
#include "..\mod\public\src\itpub\lib\array\array.h"
#include "fapi/drv_timer.h"

// длина в байтах payload mpeg пакета
#define LENGHT_PAYLOAD_MPEG          184
// длина в байтах header t2mi пакета
#define LENGHT_T2MI_HEAD             6
// длина в байтах crc32 t2mi пакета
#define LENGHT_CRC_32                4
// спец. значение, для ообозначения того что пакет накоплен
#define ACCUMULATED_PACKET           1
// индекс, uint8_t массива с т2ми пакетом, с которого начинается l1post
#define L1POST_START_INDEX (8 + 21 + 2)
// длинав в битах полей информации о конкретном PLP хранимой в Configurable L1-post
#define BITS_OF_PLP_INFO (8+3+5+1+3+8+8+3+3+1+2+10+8+8+1+1+1+11+2+1+1)
bool is_c_counter;
#pragma pack(push, 1)
//mpeg header without adaptation field
typedef struct {
    uint8         sync_byte;

    uint16        PID:13;
    uint16        transport_priority:1;
    uint16        payload_unit_start_indicator:1;
    uint16        transport_error_indicator:1;

    uint8         continuity_counter:4;
    uint8         adaptation_field_control:2;
    uint8         transport_scrambling_control:2;

} Mpeg_Header;
#pragma pack(pop)

typedef struct T2MIPacket {
    ARRAY_T*             Array;
    uint8*               ptr;
} X_T2MIPacket;

typedef struct X_T2MIPARSER_CONTEXT {
      X_T2MIPARSER_CFG cfg;

      uint8_t bytes_to_go;
    uint8_t discontinuity_indicator;
```

```
#ifdef CHECK_SYNC_MPEG
    bool prev_pkt_dublicate;
      int prev_counter_mpeg;
#endif

      int prev_counter_t2mi;
      bool T2MISync;
      bool is_accumulated_packet;
      X_T2MIPacket t2mi_packet;
} X_T2MIPARSER_CONTEXT;

static void free_hdl( H_T2MIPARSER hdl );
static Mpeg_Header get_mpeg_head( H_T2MIPARSER hdl, int32_t* err );
static uint8_t handle_adaptation_field_if_present( H_T2MIPARSER hdl, const Mpeg_Header* mpeg_head, int32_t* err
);
static int32_t sync_up( H_T2MIPARSER hdl );
static int32_t accumulate_t2mi( H_T2MIPARSER hdl );
static int32_t read_t2mi( H_T2MIPARSER hdl, unsigned int read_lvl);
static uint32_t get_min(uint32_t a, uint32_t b);
static int32_t drop_packet( H_T2MIPARSER hdl );
static int32_t get_lenght_payload_and_pad_t2mi( const uint8_t* t2mi );
static bool is_config_valid( const X_T2MIPARSER_CFG* cfg );

#ifdef CHECK_SYNC_MPEG
static bool check_sync_mpeg( H_T2MIPARSER hdl, const Mpeg_Header* MpegHead );
#endif

static bool check_sync_t2mi( H_T2MIPARSER hdl );
static tTV_DVBT2_MODE get_fft_mode( uint8_t S2 );
static uint8_t generate_mask(uint32_t num_bits, uint32_t byte_position );
static uint32_t read_bits(uint32_t num_bits, const uint8_t* buf, uint32_t* cur_bit_pos);
static int32_t t2miparser_parse( H_T2MIPARSER hdl, bool f_accumulate_packet );

H_T2MIPARSER t2miparser_open( const X_T2MIPARSER_CFG* cfg, int32_t* err )
{
      X_T2MIPARSER_CONTEXT *hdl = NULL;
      *err = T2MIPARSER_ERR_OK;

    IF_FAILED_SETSTATUS_GOTO( is_config_valid( cfg ),
                              *err, T2MIPARSER_ERR_BAD_INPUT_PARAMETERS,
                              FUNCEND );

      hdl = FAPI_SYS_MALLOC( sizeof(X_T2MIPARSER_CONTEXT) );
      IF_NULL_SETSTATUS_GOTO( hdl, *err,
                             T2MIPARSER_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);
      memset( hdl, 0, sizeof(X_T2MIPARSER_CONTEXT) );

      hdl->t2mi_packet.Array = array_new( ARRAY_FLAG_CLEAR, sizeof(uint8_t) );
    IF_NULL_SETSTATUS_GOTO( hdl->t2mi_packet.Array, *err,
                             TSRECV_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);

    hdl->t2mi_packet.ptr = array_get_item_ref(hdl->t2mi_packet.Array, uint8, 0);
      hdl->cfg = *cfg;
      hdl->prev_counter_mpeg = -1;
      hdl->prev_counter_t2mi = -1;

FUNCEND:
      if( *err != T2MIPARSER_ERR_OK ) {

             free_hdl( hdl );
```

```
            hdl = NULL;
      }

      return hdl;
}

int32_t t2miparser_close( H_T2MIPARSER hdl )
{
      int32_t err = T2MIPARSER_ERR_OK;

      IF_NULL_SETSTATUS_GOTO( hdl, err,
                              T2MIPARSER_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

    if ( hdl->bytes_to_go ) {

        IF_FAILED_RETURN_EXPR( hdl->cfg.read( NULL, (hdl->bytes_to_go),
                               hdl->cfg.aux ),
                               T2MIPARSER_ERR_STREAM_ERROR );
    }

      free_hdl( hdl );

FUNCEND:
      return err;
}

// функция чтения t2mi пакетов
// hdl @in: дескриптор т2мипарсера
// f_accumulate_packet @in: флаг, уставливается, когда необходимо накалпивать пакеты t2mi
// @return: 1)если f_accumulate_packet == true: 1 - пакет накоплен, 0 - пакет не накоплен, <0 - ошибка
//          2)если f_accumulate_packet == false: >0 - количество t2mi данных в байтах, которые необходимо
обработать
//                                     ,0 - синхронизация с потоком не завершена, <0 - ошибка
// note:
//      - если, после того как накопился пакет(функция вернула 1, когда f_accumulate_packet == true), снова
вызвать t2miparser_parse(...),
//        накопленный пакет будет очищен
//      - когда f_accumulate_packet == false, функция будет синхронизировать мпег поток и пропуск данные потока
до t2mi данных
//        (т.е. пропускать заголовок, и поле адаптации mpeg потока), когда синхронизация завершится будет
возвращено оставшиеся в mpeg пакете
//        количество байт, которые и являются t2mi потоком, и перед следующим вызовом функции пользователь
должен считать из потока это количество байт.
//      - когда f_accumulate_packet == true, функция будет накапливать t2mi пакет, как только он накопится -
вернет 1, после чего можно будеть получить
//        указатель на накопленный пакет - t2miparser_get_t2mi(...)
//      - Перед сменой режима работы нужно вызвать t2miparser_reset(...).
//
static int32_t t2miparser_parse( H_T2MIPARSER hdl, bool f_accumulate_packet )
{
    int32_t err = T2MIPARSER_ERR_OK;

    IF_NULL_RETURN_EXPR( hdl, T2MIPARSER_ERR_BAD_INPUT_PARAMETERS);

    if( hdl->is_accumulated_packet ) {

        err = drop_packet( hdl );
        IF_FAILED_RETURN_EXPR(err == 0, err);
    }
```

```
     if( !(hdl->bytes_to_go) ) {

        Mpeg_Header MpegHead = get_mpeg_head( hdl, &err );
        IF_FAILED_RETURN_EXPR(err == 0, err );

        hdl->bytes_to_go = LENGHT_PAYLOAD_MPEG;

        if( MpegHead.PID == hdl->cfg.t2mi_pid ) {

            hdl->bytes_to_go = handle_adaptation_field_if_present( hdl, &MpegHead,
                                                                   &err );
            IF_FAILED_RETURN_EXPR(err == 0, err );

#ifdef CHECK_SYNC_MPEG

            if( !hdl->discontinuity_indicator ) {
                if( !check_sync_mpeg(hdl, &MpegHead) ) {

                    IF_FAILED_RETURN_EXPR( hdl->cfg.read( NULL, hdl->bytes_to_go, hdl->cfg.aux ),
                                           T2MIPARSER_ERR_STREAM_ERROR );

                    hdl->bytes_to_go = 0;

                    return T2MIPARSER_ERR_DESYNCTHRONIZATION ;
                }
            }

            hdl->prev_counter_mpeg = MpegHead.continuity_counter;

#endif

            if( MpegHead.payload_unit_start_indicator && !hdl->T2MISync ) {
                err = sync_up(hdl);
                IF_FAILED_RETURN_EXPR(err == 0, err);
            }

            else if( MpegHead.payload_unit_start_indicator ) {
                IF_FAILED_RETURN_EXPR( hdl->cfg.read( NULL, 1, hdl->cfg.aux ),
                                       T2MIPARSER_ERR_STREAM_ERROR );

                hdl->bytes_to_go -= 1;
            }
        }
        else {
            IF_FAILED_RETURN_EXPR( hdl->cfg.read( NULL, (hdl->bytes_to_go),
                                                  hdl->cfg.aux ),
                                   T2MIPARSER_ERR_STREAM_ERROR );

            hdl->bytes_to_go = 0;
        }
    }
    if ( hdl->T2MISync ) {

        if( !f_accumulate_packet ) {
            uint8_t t2mi_data_bytes = hdl->bytes_to_go;
            hdl->bytes_to_go = 0;

            return t2mi_data_bytes;
        }
```

```
        err = accumulate_t2mi( hdl );

        IF_FAILED_RETURN_EXPR(err == 0, err);

        if( hdl->is_accumulated_packet == true ) {
            return ACCUMULATED_PACKET;
        }

    }

    else if ( hdl->bytes_to_go != 0 ) {

        IF_FAILED_RETURN_EXPR( hdl->cfg.read( NULL, hdl->bytes_to_go,
                                              hdl->cfg.aux ),
                               T2MIPARSER_ERR_STREAM_ERROR );

        hdl->bytes_to_go = 0;
    }

    return err;
}

int32_t t2miparser_get_t2mi_packet( H_T2MIPARSER hdl )
{
    return t2miparser_parse( hdl, true );
}

int32_t t2miparser_skip_to_t2mi_data( H_T2MIPARSER hdl )
{
    return t2miparser_parse( hdl, false );
}

int32_t t2miparser_reset( H_T2MIPARSER hdl )
{
    int32_t err = T2MIPARSER_ERR_OK;

    IF_NULL_SETSTATUS_GOTO( hdl, err, T2MIPARSER_ERR_BAD_INPUT_PARAMETERS, FUNCEND );

      hdl->T2MISync = false;
      hdl->prev_counter_t2mi = -1;
    hdl->discontinuity_indicator = 0;

#ifdef CHECK_SYNC_MPEG
    hdl->prev_counter_mpeg = -1;
    hdl->prev_pkt_dublicate = false;
#endif

    if( hdl->bytes_to_go ) {
        IF_FAILED_SETSTATUS_GOTO( hdl->cfg.read( NULL, (hdl->bytes_to_go),
                                                 hdl->cfg.aux ),
                                  err, T2MIPARSER_ERR_STREAM_ERROR, FUNCEND);
        hdl->bytes_to_go = 0;
    }

    err = drop_packet( hdl );

FUNCEND:

    return err;
}
```

```
unsigned int t2miparser_get_t2mi_type( const uint8_t* t2mi)
{
    return (unsigned int)t2mi[0];
}

const uint8_t* t2miparser_get_t2mi( H_T2MIPARSER hdl )
{
    IF_NULL_RETURN_EXPR( hdl, NULL );

    IF_FAILED_RETURN_EXPR( hdl->is_accumulated_packet, NULL );

      return hdl->t2mi_packet.ptr;
}

uint16_t t2miparser_get_length_payload( const uint8_t* t2mi ) {
    return t2mi[4] * 256 + t2mi[5];
}

unsigned int t2miparser_get_length( const uint8_t* t2mi) {
    return (unsigned int)get_lenght_payload_and_pad_t2mi( t2mi ) +
           LENGHT_T2MI_HEAD + LENGHT_CRC_32;
}

uint8_t t2miparser_get_subseconds_unit( const uint8_t* t2mi )
{
    static uint8_t lookup[] =
    {
        [0] = 131,
        [1] = 40,
        [2] = 48,
        [3] = 56,
        [4] = 64,
        [5] = 80,
    };

    return lookup[(t2mi[6] & 0x0F)];
}

uint64_t t2miparser_get_seconds_since_2000( const uint8_t* t2mi )
{
    return ( (uint64_t)t2mi[7] << 32) + ( (uint64_t)t2mi[8] << 24 ) +
             ( (uint64_t)t2mi[9] << 16 ) + ( (uint64_t)t2mi[10]  << 8 ) + (uint64_t)t2mi[11];
}

uint32_t t2miparser_get_subseconds( const uint8_t* t2mi )
{
    return ( (uint32_t)t2mi[12] << 19 ) + ( (uint32_t)t2mi[13] << 11 ) +
             ( (uint32_t)t2mi[14] << 3 ) + ( (t2mi[15]&0xE0) >> 5 );
}

uint8_t t2miparser_get_super_frame_index( const uint8_t* t2mi )
{
    return ( (uint8_t)t2mi[2] >> 4 );
}

uint8_t t2miparser_get_plp( const uint8_t* t2mi )
{
    return t2mi[7];
}
```

```
//ISSYI (1 bit) from MATYPE-1 in bbframe header
uint8_t t2miparser_get_issyi( const uint8_t* t2mi )
{
    return ( t2mi[9] & 0x08 ) >> 3;
}

//NPD (1 bit) from MATYPE-1 in bbframe header
uint8_t t2miparser_get_npd( const uint8_t* t2mi )
{
    return ( t2mi[9] & 0x04 ) >> 2;
}

tTV_DVBT2_L1PRE get_l1pre_from_packet( const uint8_t* t2mi) {
    const uint8_t* begin_l1pre = t2mi + 8;
    tTV_DVBT2_L1PRE l1pre;

    l1pre.Type = begin_l1pre[0];

    l1pre.BwExt = (begin_l1pre[1] & 0x80) >> 7;

    l1pre.S1 = (begin_l1pre[1] & 0x70) >> 4;

    l1pre.S2 = begin_l1pre[1] & 0x0F;

    l1pre.Mixed = l1pre.S2 & 0x01;

    if( !((uint8_t)l1pre.S1 & 0x06) ) {
        l1pre.FftMode = get_fft_mode(l1pre.S2);
    }

    l1pre.L1Rep = (begin_l1pre[2] & 0x80) >> 7;

    l1pre.Gi = (begin_l1pre[2] & 0x70) >> 4;

    l1pre.Papr = begin_l1pre[2] & 0x0F;

    l1pre.Mod = (begin_l1pre[3] & 0xF0) >> 4;

    l1pre.Cr = (begin_l1pre[3] & 0x0C) >> 2;

    l1pre.Fec = begin_l1pre[3] & 0x03;

    l1pre.L1PostSize = begin_l1pre[4] * 1024 + begin_l1pre[5] * 4 + ((begin_l1pre[6] & 0xC0) >> 6);

    l1pre.L1PostInfoSize = (begin_l1pre[6] & 0x3F) * 4096 + begin_l1pre[7] * 16 + ((begin_l1pre[8] & 0xF0) >>
4);

    l1pre.Pp = begin_l1pre[8] & 0x0F;

    l1pre.TxIdAvailability = begin_l1pre[9];

    l1pre.CellId = begin_l1pre[10] * 256 + begin_l1pre[11];

    l1pre.NetworkId = begin_l1pre[12] * 256 + begin_l1pre[13];

    l1pre.SystemId = begin_l1pre[14] * 256 + begin_l1pre[15];

    l1pre.NumFrames = begin_l1pre[16];

    l1pre.NumSymbols = begin_l1pre[17] * 16 + ((begin_l1pre[18] & 0xF0) >> 4);
```

```
    l1pre.Regen = (begin_l1pre[18] & 0x0E) >> 1;

    l1pre.PostExt = begin_l1pre[18] & 0x01;

    l1pre.NumRfFreqs = (begin_l1pre[19] & 0xE0) >> 5;

    l1pre.RfIdx = (begin_l1pre[19] & 0x1C) >> 2;

    l1pre.T2Version = (begin_l1pre[19] & 0x03) * 4 + ((begin_l1pre[20] & 0xC0) >> 6);

    if( l1pre.T2Version == TV_DVBT2_V131 ) {
        l1pre.L1PostScrambled = (begin_l1pre[20] & 0x20) >> 5;

        l1pre.T2BaseLite = (begin_l1pre[20] & 0x10) >> 4;
    }

    l1pre.Crc32 = (uint32_t)(begin_l1pre[21] << 24) + (uint32_t)(begin_l1pre[22] << 16) +
                  (uint32_t)(begin_l1pre[23] << 8) + begin_l1pre[24];
    return l1pre;
}

tTV_DVBT2_L1POST get_l1post_from_packet( const uint8_t* t2mi) {
    tTV_DVBT2_L1POST l1post;
    uint32_t cur_bit = 0;
    uint8_t* l1post_start = (uint8_t*)&t2mi[L1POST_START_INDEX];
    tTV_DVBT2_L1PRE l1pre = get_l1pre_from_packet( t2mi );

    l1post.SubSlicesPerFrame = (uint16_t)read_bits(15, l1post_start, &cur_bit);
    l1post.NumPLPs = (uint8_t)read_bits(8, l1post_start, &cur_bit);
    l1post.NumAux = (uint8_t)read_bits(4, l1post_start, &cur_bit);
    l1post.AuxConfigRFU =  (uint8_t)read_bits(8, l1post_start, &cur_bit);

    for( int32_t i = 0; i < l1pre.NumRfFreqs; i++ ) {
        l1post.RfIdx = (uint8_t)read_bits(3, l1post_start, &cur_bit);
        l1post.Freq = read_bits(32, l1post_start, &cur_bit);
    }

    if( (l1pre.S2 & 0x01) == 1 ) {
        l1post.FefType = (uint8_t)read_bits(4, l1post_start, &cur_bit);
        l1post.FefLength = read_bits(22, l1post_start, &cur_bit);
        l1post.FefInterval = (uint8_t)read_bits(8, l1post_start, &cur_bit);
    }

    return l1post;
}

int32_t get_plp_info( const uint8_t* t2mi, uint8_t plp_ind, tTV_DVBT2_PLP* l1plp )
{
    int32_t err = T2MIPARSER_ERR_OK;

    uint32_t cur_bit = L1POST_START_INDEX * 8;
    tTV_DVBT2_L1PRE l1pre = get_l1pre_from_packet( t2mi );

    //add bits of SubSlicesPerFrame
    cur_bit += 15;

    uint8_t num_plp = (uint8_t)read_bits(8, t2mi, &cur_bit);
    IF_FAILED_SETSTATUS_GOTO( plp_ind < num_plp, err, T2MIPARSER_ERR_BAD_INPUT_PARAMETERS, FUNCEND );
    //add bits of NumAux, AuxConfigRFU
```

```
cur_bit = cur_bit + (4 + 8);

for( int32_t i = 0; i < l1pre.NumRfFreqs; i++ ) {
    //add bits of RfIdx, Freq
    cur_bit = cur_bit + (3 + 32);
}

if( (l1pre.S2 & 0x01) == 1 ) {
    //add bits of FefType, FefLength, FefInterval
    cur_bit = cur_bit + (4 + 22 + 8);
}

uint8_t cur_plp_ind = 0;
while( cur_plp_ind != plp_ind)
{
    cur_bit = cur_bit + BITS_OF_PLP_INFO;
    cur_plp_ind++;
}

l1plp->Id   = (uint8_t)read_bits(8, t2mi, &cur_bit);

l1plp->Type = read_bits(3, t2mi, &cur_bit);

l1plp->Payload = read_bits(5, t2mi, &cur_bit);

l1plp->Ff = (uint8_t)read_bits(1, t2mi, &cur_bit);

l1plp->FirstRfIdx = (uint8_t)read_bits(3, t2mi, &cur_bit);

l1plp->FirstFrmIdx = (uint8_t)read_bits(8, t2mi, &cur_bit);

l1plp->GroupId = (uint8_t)read_bits(8, t2mi, &cur_bit);

l1plp->PlpCr = read_bits(3, t2mi, &cur_bit);

l1plp->Constell = read_bits(3, t2mi, &cur_bit);

l1plp->Rot = (uint8_t)read_bits(1, t2mi, &cur_bit);

l1plp->Fec = read_bits(2, t2mi, &cur_bit);

l1plp->NumBlocksMax = (uint16_t)read_bits(10 , t2mi, &cur_bit);

l1plp->FrmInt = (uint8_t)read_bits(8, t2mi, &cur_bit);

l1plp->TilLen = (uint8_t)read_bits(8, t2mi, &cur_bit);

l1plp->TilType = (uint8_t)read_bits(1, t2mi, &cur_bit);

l1plp->InBandFlag = (uint8_t)read_bits(1, t2mi, &cur_bit);

l1plp->InBandBFlag = (uint8_t)read_bits(1, t2mi, &cur_bit);

l1plp->Rsvd = (uint16_t)read_bits(11, t2mi, &cur_bit);

l1plp->PlpMode = (uint8_t)read_bits(2, t2mi, &cur_bit) >> 1;

read_bits(1, t2mi, &cur_bit);

read_bits(1, t2mi, &cur_bit);
```

```
    l1plp->InBandBTSRate = 0;

FUNCEND:
    return err;
}

uint16_t t2miparser_get_syncd( const uint8_t* t2mi ) {
    return t2mi[16] * 256 + t2mi[17];
}

uint16_t t2miparser_get_dfl( const uint8_t* t2mi ) {
    return t2mi[13] * 256 + t2mi[14];
}

static int32_t get_lenght_payload_and_pad_t2mi( const uint8_t* t2mi ) {
    uint16_t LenghtPayloadT2MI = t2miparser_get_length_payload( t2mi );

    LenghtPayloadT2MI = ( LenghtPayloadT2MI + 7 ) / 8;
    return LenghtPayloadT2MI;
}

static void free_hdl( H_T2MIPARSER hdl )
{
    if( hdl != NULL ) {

        if( hdl->t2mi_packet.Array != NULL ) {
            array_free( hdl->t2mi_packet.Array, true );
        }

        FAPI_SYS_FREE(hdl);
    }
}

static int32_t sync_up( H_T2MIPARSER hdl )
{
    uint8 HeaderOffset;
    IF_FAILED_RETURN_EXPR( hdl->cfg.read( &HeaderOffset, sizeof( HeaderOffset ), hdl->cfg.aux ),
                           T2MIPARSER_ERR_STREAM_ERROR );

    hdl->bytes_to_go -= sizeof( HeaderOffset );

    IF_FAILED_RETURN_EXPR( hdl->cfg.read( NULL, HeaderOffset, hdl->cfg.aux ),
                           T2MIPARSER_ERR_STREAM_ERROR );

    hdl->bytes_to_go -= HeaderOffset;
    hdl->T2MISync = true;

    return T2MIPARSER_ERR_OK;
}

static int32_t accumulate_t2mi( H_T2MIPARSER hdl )
{
    int32_t err = T2MIPARSER_ERR_OK;

    while( hdl->bytes_to_go ) {

        if( hdl->t2mi_packet.Array->count < LENGHT_T2MI_HEAD ) {

           err = read_t2mi( hdl, (unsigned int)LENGHT_T2MI_HEAD );
```

```
        IF_FAILED_RETURN_EXPR(err == 0, err);
        }

        else {

            err = read_t2mi( hdl, (unsigned int)t2miparser_get_length( hdl->t2mi_packet.ptr ) );
            IF_FAILED_RETURN_EXPR(err == 0, err);

            if( t2miparser_get_length( hdl->t2mi_packet.ptr ) == hdl->t2mi_packet.Array->count ) {

                IF_FAILED_RETURN_EXPR( check_sync_t2mi( hdl ),
                                       T2MIPARSER_ERR_DESYNCTHRONIZATION);

                hdl->prev_counter_t2mi = (int)hdl->t2mi_packet.ptr[1];

                hdl->is_accumulated_packet = true;
                break;
            }

        }
    }
    return err;
}

static Mpeg_Header get_mpeg_head( H_T2MIPARSER hdl, int32_t* err )
{
    *err = T2MIPARSER_ERR_OK;

    Mpeg_Header head;
    const int32 mask[2] = {1, -1};

    IF_FAILED_SETSTATUS_GOTO( hdl->cfg.read( (uint8_t*)&head,
                                             sizeof(head), hdl->cfg.aux ),
                              *err, T2MIPARSER_ERR_STREAM_ERROR, FUNCEND );

    TS_Interleaving16( (uint8_t*)&head, mask );

FUNCEND:
    return head;
}

//read discontinuity_indicator and skip other data from adaptation field if it present
static uint8_t handle_adaptation_field_if_present( H_T2MIPARSER hdl, const Mpeg_Header* mpeg_head, int32_t*
err)
{
    *err = T2MIPARSER_ERR_OK;

    if( mpeg_head->adaptation_field_control == 2 ||
        mpeg_head->adaptation_field_control == 3 )
    {
        uint8_t lengthAD;
        IF_FAILED_SETSTATUS_GOTO( hdl->cfg.read( &lengthAD, sizeof(lengthAD), hdl->cfg.aux),
                                  *err, T2MIPARSER_ERR_STREAM_ERROR, FUNCEND );

        if( lengthAD > 0 ) {
            IF_FAILED_SETSTATUS_GOTO( hdl->cfg.read( &hdl->discontinuity_indicator,
                                                     sizeof( hdl->discontinuity_indicator ), hdl->cfg.aux),
                                      *err, T2MIPARSER_ERR_STREAM_ERROR, FUNCEND );
            hdl->discontinuity_indicator >>= 7;
```

```
            IF_FAILED_SETSTATUS_GOTO( hdl->cfg.read( NULL, lengthAD - sizeof( hdl->discontinuity_indicator ),
hdl->cfg.aux ),
                                      *err, T2MIPARSER_ERR_STREAM_ERROR, FUNCEND );
        }

        return ( LENGHT_PAYLOAD_MPEG - sizeof( lengthAD ) - lengthAD );//(183-(uint32_t)lengthAD);
    }

FUNCEND:
    return LENGHT_PAYLOAD_MPEG;
}

// read_lvl сколько байт т2ми пакета считать
static int32_t read_t2mi( H_T2MIPARSER hdl, unsigned int read_lvl)
{
    int32_t err = FAPI_OK;
    X_T2MIPacket* T2MIpacket = &hdl->t2mi_packet;

    uint8_t bytes = (uint8_t)get_min( hdl->bytes_to_go,
                                      read_lvl - T2MIpacket->Array->count );
    uint32_t index = T2MIpacket->Array->count;

    if( T2MIpacket->Array->capacity < read_lvl ) {
        err = array_set_capacity( T2MIpacket->Array,
                                  T2MIpacket->Array->count + read_lvl + 2048);
        IF_FAILED_RETURN_EXPR(err == 0, err);

        hdl->t2mi_packet.ptr = array_get_item_ref(T2MIpacket->Array, uint8, 0);
    }

    IF_FAILED_RETURN_EXPR( hdl->cfg.read( (T2MIpacket->ptr + index), bytes, hdl->cfg.aux),
                           T2MIPARSER_ERR_STREAM_ERROR);

    hdl->bytes_to_go -= bytes;

    err = array_set_count( T2MIpacket->Array,
                           T2MIpacket->Array->count + bytes );
    IF_FAILED_RETURN_EXPR( err == 0, err );


    return err;
}

static uint32_t get_min( uint32_t a, uint32_t b )
{
    return ( a < b )? a : b;
}

static int32_t drop_packet( H_T2MIPARSER hdl ) {
    int32_t err = T2MIPARSER_ERR_OK;
    X_T2MIPacket* T2MIpacket = &(hdl->t2mi_packet);

    if( T2MIpacket->Array->count ) {
        err = array_set_count( T2MIpacket->Array, 0);
        IF_FAILED_RETURN_EXPR(err >= 0, err);

        T2MIpacket->ptr = array_get_item_ref(T2MIpacket->Array, uint8, 0);
    }

    hdl->is_accumulated_packet = false;
```

```
    return err;
}

static bool is_config_valid( const X_T2MIPARSER_CFG* cfg )
{
    IF_NULL_RETURN_EXPR( cfg, false );
    IF_NULL_RETURN_EXPR( cfg->read, false );

    return true;
}

#ifdef CHECK_SYNC_MPEG
static bool check_sync_mpeg( H_T2MIPARSER hdl, const Mpeg_Header* MpegHead )
{
    IF_FAILED_RETURN_EXPR( hdl->prev_counter_mpeg >= 0, true );

    if( MpegHead->adaptation_field_control == 0 || MpegHead->adaptation_field_control == 2 ) {
        if(MpegHead->continuity_counter != hdl->prev_counter_mpeg) {
            return false;
        }
        return true;
    }
    else {

        if( MpegHead->continuity_counter == (hdl->prev_counter_mpeg + 1) ||
            (MpegHead->continuity_counter == 0 && hdl->prev_counter_mpeg == 15) )
        {
            hdl->prev_pkt_dublicate = false;
            return true;
        }
        else if( MpegHead->continuity_counter == hdl->prev_counter_mpeg &&
                 !hdl->prev_pkt_dublicate )
        {
            hdl->prev_pkt_dublicate = true;
            return true;
        }
        else {
            return false;
            hdl->prev_pkt_dublicate = false;
        }
    }
}
#endif

static bool check_sync_t2mi( H_T2MIPARSER hdl )
{
    IF_FAILED_RETURN_EXPR( hdl->prev_counter_t2mi >= 0, true );

    IF_FAILED_RETURN_EXPR( ( hdl->t2mi_packet.ptr[1] == (hdl->prev_counter_t2mi + 1) ||
                             ( hdl->t2mi_packet.ptr[1] == 0 && hdl->prev_counter_t2mi == 255 ) ),
                           false );

    return true;
}

static tTV_DVBT2_MODE get_fft_mode( uint8_t S2 ) {

    static uint8_t lookup[] =
    {
        [0] = TV_DVBT2_M2K,
```

```
        [1] = TV_DVBT2_M8K,
        [2] = TV_DVBT2_M4K,
        [3] = TV_DVBT2_M1K,
        [4] = TV_DVBT2_M16K,
        [5] = TV_DVBT2_M32K,
        [6] = TV_DVBT2_M8K,
    };

    return lookup[(S2 & 0x0E) >> 1];
}

/* составление маски для байта
num_bits @in: количество 1-х битов в маске
byte_position @in: позиция начала установки 1-х битов (0-1-2-3-4-5-6-7)
@return: маска для 1 байта */
static uint8_t generate_mask(uint32_t num_bits, uint32_t byte_position )
{

    uint8_t mask = (uint8_t) (( 1 << num_bits ) - 1);

    return (uint8_t)( mask << (8 - byte_position - num_bits) );

}

/* чтение указанного количества битов из произвольного места буфера
num_bits @in: кол-во битов, которые нужно считать
buf @in: буфер из которого будет происходить считывание
cur_bit_pos @in: номер бита с которого начинать чтение,
note:
    -после чтения к значению *cur_bit_pos будет добовлено количество считанных битов.
@return: прочитанные биты */
static uint32_t read_bits(uint32_t num_bits, const uint8_t* buf, uint32_t* cur_bit_pos)
{
    uint32_t write_buf = 0;

    while (num_bits > 0) {

        uint32_t cur_byte = *cur_bit_pos / 8;
        uint32_t remaning_bits_in_byte = 8 - ((*cur_bit_pos % 8) );

        if (num_bits < remaning_bits_in_byte) {

            uint8_t mask = generate_mask( num_bits, (*cur_bit_pos % 8) );
            write_buf += ( buf[cur_byte] & mask ) >>
                         ( 8 - ( *cur_bit_pos + num_bits ) % 8 );
            *cur_bit_pos += num_bits;
            num_bits -= num_bits;
        }
        else {

            uint8_t mask = generate_mask( remaning_bits_in_byte, (*cur_bit_pos % 8) );
            write_buf += (uint32_t)( buf[cur_byte] & mask ) <<
                           ( num_bits - remaning_bits_in_byte );
            *cur_bit_pos += remaning_bits_in_byte;
            num_bits -= remaning_bits_in_byte;
        }

    }
    return write_buf;
}
```

ПРИЛОЖЕНИЕ С

ИСХОДНЫЙ КОД МОДУЛЯ MPEGPIDSTATISTICS

Листинг 3 – Исходный код mpegPidStatistics

```
#include "mpegPidStatistics.h"

#include "lib/utils/macroCheckers.h"
//TODO вынести array на другой уровень
#include "..\mod\public\src\itpub\lib\array\array.h"

// спец значение, обозначающее ситуакию, когда указаный PID не был найден в текущей статистике
#define PID_NOT_FOUND -1

static uint16_t get_pid( const uint8_t* const mpeg_head );
static int32_t find_pid( uint16_t pid, const PID_STATISTICS* const pid_list, uint32_t count_list );
static int32_t make_statistics( unsigned int count_packets, F_MPEG_PS_GET_NEXT_PACKET next,
                                void* aux, ARRAY_T* pid_statistics );

ARRAY_T* mpeg_pid_statistics( unsigned int packets_count, F_MPEG_PS_GET_NEXT_PACKET next, void* aux,
                                    int32_t* err )
{
      *err = MPEG_PS_ERR_OK;
      ARRAY_T* pid_statistics = NULL;
      IF_NULL_SETSTATUS_GOTO( next, *err, MPEG_PS_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

      pid_statistics = array_new( ARRAY_FLAG_CLEAR, sizeof(PID_STATISTICS) );
      IF_NULL_SETSTATUS_GOTO( pid_statistics, *err, MPEG_PS_ERR_MEMORY_ALLOCATION_ERROR, FUNCEND);

      *err = array_set_capacity( pid_statistics, 10 );
      IF_FAILED_GOTO( *err >= 0, FUNCEND);

      *err = make_statistics( packets_count, next, aux, pid_statistics);
      IF_FAILED_GOTO( *err == MPEG_PS_ERR_OK, FUNCEND);

FUNCEND:
      if(*err != MPEG_PS_ERR_OK ){
            if( pid_statistics != NULL) {
                  array_free( pid_statistics, true );
                  pid_statistics = NULL;
```

```
		}
	}

	return pid_statistics;
}

// получение значения PID из пакета
static uint16_t get_pid( const uint8_t* const mpeg_head )
{
	return (uint16_t)( (mpeg_head[1] & 0x1F) << 8) + mpeg_head[2];
}



//  поиск указанного pid в текущей статистике
//    pid @in: pid, который будет искаться в статистике
//    pid_list @in: текущая статистика
//  count_list @in: количество записей в текущей статистике
//    @return: индекс записи в которой содержится информация о выбранном pid
//
static int32_t find_pid( uint16_t pid, const PID_STATISTICS* const pid_list, uint32_t count_list )
{
	int32_t idx = PID_NOT_FOUND;

	for( uint32_t i = 0; i < count_list ; i++ )
	{
		if( pid_list[i].pid == pid )
		{
			idx = (int32_t) i;
			break;
		}
	}

	return idx;
}

//    составление статистики PID из mpeg пакетов
//    count_packets @in: количество пакетов, на основе который будет составлена статистика
//    next @in: функтор получения следующего пакета
//  aux @in: пользовательские данные передаваемые в next
//  pid_statistics @out: составленная статистика
```

```
//    @return: ошибка, если != MPEG_PS_ERR_OK, см. коды ошибок
//
static int32_t make_statistics( unsigned int count_packets, F_MPEG_PS_GET_NEXT_PACKET next,
                                 void* aux, ARRAY_T* pid_statistics ) {
      int32_t err = MPEG_PS_ERR_OK;
      uint8_t mpeg_packet[188];
      PID_STATISTICS* pid_list = array_get_item_ref( pid_statistics, PID_STATISTICS, 0);

      for( unsigned int i = 0;  ( i < count_packets || count_packets == 0 ); i++) {

            IF_FAILED_SETSTATUS_GOTO( next( mpeg_packet, aux ), err, MPEG_PS_ERR_GET_PACKET, FUNCEND);

            uint16_t pid = get_pid( mpeg_packet );
            int32_t idx = find_pid( pid, pid_list, pid_statistics->count );

            if( idx != PID_NOT_FOUND ) {
                  pid_list[idx].count++;
            }
            else {
                  if( pid_statistics->count == pid_statistics->capacity ) {
                        err = array_set_capacity( pid_statistics, pid_statistics->capacity + 1 );
                        IF_FAILED_GOTO( err >= 0, FUNCEND);

                        pid_list = array_get_item_ref( pid_statistics, PID_STATISTICS, 0);
                  }

                  pid_list[ pid_statistics->count ].pid = pid;
                  pid_list[ pid_statistics->count ].count = 1;

                  array_set_count( pid_statistics, pid_statistics->count + 1 );
                  IF_FAILED_GOTO( err >= 0, FUNCEND);
            }
      }

FUNCEND:

      return err;
}
```

ПРИЛОЖЕНИЕ D

ИСХОДНЫЙ КОД МОДУЛЯ T2MIMEASBITRATE

Листинг 4 – Исходный код t2miMeasBitrate

```
#include "t2miMeasBitrate.h"

#include "../../../lib/utils/macroCheckers.h"
#include "../../../lib/parser/t2mi/t2miparser.h"

// значение поля packet_type соответсвующее типу пакета DVB-T2 timestamp
#define PACKET_TYPE_OF_TIMESTAMP 32

// размер выборки битрейта, на основе которой будет считаться средний битрейт
#define BITRATE_SAMPLING_SIZE 10

// спец значение обознчающее отсутствие значения
#define NOT_EXIST 255

/* интервал пакетов, если в течение этого интервала не встретится TIMESTAMP пакет, то в потоке их нет */
#define TIMESTAMP_INTERVAL_MPEG_PACKETS_MAX 25000

// контекст чтения данных из потока
typedef struct X_READ_CONTEXT {
	// количество прочитанных байт
	uint64_t bytes_done;
	// параметры потока
	T2MI_MB_STREAM_CFG* stream_cfg;
} X_READ_CONTEXT;

// информация которую необходимо хранить для расчета битрейта
typedef struct X_MEAS_BR_CONTEXT {
	const uint8_t* t2mi;
	//предыдущее значения поля seconds_since_2000
	uint64_t prev_seconds_since_2000;
	//предыдущее значения поля subseconds
	uint32_t prev_subseconds;
	//предыдущее значения количества считанных байт
	uint64_t prev_bytes_done;
	//предыдущее значения поля super_frame_idx
	uint8_t prev_super_frame_idx;
	// флаг синхронизации super frame. Устанавливается, когда находится начало super frame
	bool super_frame_sync;
	// контекст чтения данных из потока
	X_READ_CONTEXT* reading;
} X_MEAS_BR_CONTEXT;

static bool is_cfg_valid( const T2MI_MB_STREAM_CFG* const streamCfg );
static bool read_stream( uint8_t* read_buf, uint8_t bytes, void* aux );
static uint32_t meas_bitrate( X_MEAS_BR_CONTEXT* cont );
static uint32_t find_avg(uint32_t* array, int size);

//TODO* Null timestamp?
// измерение скорости потока (с которой встречаются пакеты timestamp)
// streamCfg @in: параметры потока
// err @out: ошибка, если != T2MI_MB_ERR_OK, см. коды ошибок
//  @return: скорость потока(бит/сек), если 0 - см. коды ошибок
//
uint32_t t2mi_meas_bitrate( const T2MI_MB_STREAM_CFG* const streamCfg,
                            int32_t* err )
{
	*err = T2MI_MB_ERR_OK;
	uint32_t bitrate = 0;
	unsigned int bitrates_counter = 0;
	uint32_t bitrates[BITRATE_SAMPLING_SIZE];
	H_T2MIPARSER h_parser = NULL;

	IF_FAILED_SETSTATUS_GOTO( is_cfg_valid( streamCfg ),
	                          *err, T2MI_MB_ERR_BAD_INPUT_PARAMETERS, FUNCEND);

	X_READ_CONTEXT reading = { 0, (T2MI_MB_STREAM_CFG*)streamCfg };
```

```
	const X_T2MIPARSER_CFG parser_cfg = { &reading, read_stream, streamCfg->t2mi_pid };
	h_parser = t2miparser_open( &parser_cfg, err );
	IF_NULL_GOTO(h_parser, FUNCEND);

	X_MEAS_BR_CONTEXT meas_cont = {
		.super_frame_sync = false,
		.reading = &reading,
		.prev_super_frame_idx = NOT_EXIST,
	};

	uint32_t mpeg_packets_counter = 0;
	while( bitrates_counter < BITRATE_SAMPLING_SIZE ) {
		int32_t t2mi_ready = t2miparser_get_t2mi_packet( h_parser );

		if( t2mi_ready > 0 ) {

			meas_cont.t2mi = t2miparser_get_t2mi( h_parser );
			if( t2miparser_get_t2mi_type( meas_cont.t2mi ) == PACKET_TYPE_OF_TIMESTAMP ) {
				uint32_t local_bitrate = meas_bitrate( &meas_cont );

				if( local_bitrate ) {
					bitrates[bitrates_counter] = local_bitrate;

					bitrates_counter++;
				}
			}
		}

		else if( t2mi_ready == T2MIPARSER_ERR_DESYNCTHRONIZATION ) {
			*err = t2miparser_reset( h_parser );
			IF_FAILED_GOTO(*err == T2MIPARSER_ERR_OK, FUNCEND);

			meas_cont.super_frame_sync = false;
			meas_cont.prev_super_frame_idx = NOT_EXIST;
		}

		else if( t2mi_ready < 0 ) {
			*err = t2mi_ready;
			break;
		}
		if( mpeg_packets_counter ==  TIMESTAMP_INTERVAL_MPEG_PACKETS_MAX ) {
			break;
		}
		mpeg_packets_counter++;
	}

	if( *err >= 0 && bitrates_counter ) {
		bitrate = find_avg( bitrates, (int)bitrates_counter );
		*err = T2MI_MB_ERR_OK;
	}

FUNCEND:
	if(h_parser != NULL) {
		t2miparser_close( h_parser );
	}

	return bitrate;
}

// проверка корректности настроечных параметров
// streamCfg @in: параметры потока
// @return: true - параметры корректны
//          false - параметры некорректны
//
static bool is_cfg_valid( const T2MI_MB_STREAM_CFG* const streamCfg )
{
	IF_NULL_RETURN_EXPR( streamCfg, false );
	IF_NULL_RETURN_EXPR( streamCfg->read, false );

	return true;
}

// функция чтения данных из потока, передается в t2miparser
// подробнее: F_T2MIPARSER_READ_NEXT
// note: в функции идет подсчет количества запрашиваемых из потока байт, это необходимо для расчета битрейта
static bool read_stream( uint8_t* read_buf, uint8_t bytes, void* aux )
{
```

```
    X_READ_CONTEXT* cont = (X_READ_CONTEXT*)aux;

    cont->bytes_done += bytes;

    return cont->stream_cfg->read( read_buf, bytes, cont->stream_cfg->aux );
}

// расчет битрейта
// cont @in: данные необходимые для расчета
// @return: битрейт(бит/сек), 0 - при невозможности расчитать битрейт
//
static uint32_t meas_bitrate( X_MEAS_BR_CONTEXT* cont )
{
        uint32_t bitrate = 0;
        uint64_t delta_second_since_2000 = 0;
        uint32_t delta_subseconds = 0;
        uint64_t delta_bytes_done = 0;

        if( cont->super_frame_sync &&
                 cont->prev_super_frame_idx != t2miparser_get_super_frame_index( cont->t2mi )) {

                 uint64_t seconds_since_2000 = t2miparser_get_seconds_since_2000( cont->t2mi );
                 uint32_t subseconds = t2miparser_get_subseconds( cont->t2mi );
                 uint8_t subseconds_unit = t2miparser_get_subseconds_unit( cont->t2mi );

                 if( subseconds >= cont->prev_subseconds) {
                          delta_subseconds = subseconds - cont->prev_subseconds;
                          delta_second_since_2000 = (seconds_since_2000 - cont->prev_seconds_since_2000) &
                                                    0x000000FFFFFFFFFF;

                 }
                 else {
                          delta_subseconds = subseconds - cont->prev_subseconds + subseconds_unit * 1000000;
                          delta_second_since_2000 = (seconds_since_2000 - cont->prev_seconds_since_2000 - 1) &
                                                    0x000000FFFFFFFFFF;
                 }

                 delta_bytes_done = cont->reading->bytes_done - cont->prev_bytes_done;

                 if( !seconds_since_2000 && !cont->prev_seconds_since_2000 ) {
                          bitrate = (uint32_t)(( delta_bytes_done * 8 * subseconds_unit * 1000000 ) /
                                    delta_subseconds);
                 }
                 else {
                          bitrate = (uint32_t)( ( delta_bytes_done * 8 * subseconds_unit * 1000000 ) /
                                               (delta_second_since_2000 * subseconds_unit + delta_subseconds) );
                          //TODO не было тестов расчета битрейта, когда в t2mi потоке используется поле second_since_2000
                 }

                 cont->prev_seconds_since_2000 = seconds_since_2000;
                 cont->prev_subseconds = subseconds;
                 cont->prev_bytes_done = cont->reading->bytes_done;

        }
        else if( cont->prev_super_frame_idx != NOT_EXIST &&
                      cont->prev_super_frame_idx != t2miparser_get_super_frame_index( cont->t2mi ) ) {

                 cont->super_frame_sync = true;
                 cont->prev_seconds_since_2000 = t2miparser_get_seconds_since_2000( cont->t2mi );
                 cont->prev_subseconds = t2miparser_get_subseconds( cont->t2mi );
                 cont->prev_bytes_done = cont->reading->bytes_done;
        }

        cont->prev_super_frame_idx = t2miparser_get_super_frame_index( cont->t2mi );

        return bitrate;
}

// вычисление среднего значения элементов массива
// array @in: массив значений
// size @in: размер массива
// @return: среднее значение
//
static uint32_t find_avg(uint32_t* array, int size)
{
        uint32_t avg = 0;
        if( size ) {
```

```
                for( int i = 0; i < size; i++ ) {
                        avg += array[i];
                }

                avg /= (uint32_t)size;
        }

        return avg;
}
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД СОСТОЯНИЯ ПОДСИСТЕМЫ ФРОНТ-ЕНД ПЛЕЕРА - PLRINFO_T2MI

Листинг 5 – Исходный код plrinfo_t2mi

```
#define DEBUG_LEVEL 0

//=============================================================================================
//ПОДКЛЮЧdАЕМЫЕ ФАЙЛЫ ========================================================================
//
#include <itapi.h>
#include <itpub\macroCheckers.h>
#include <itpub\lib\chinfopanel\chinfopanel.h>
#include "..\..\..\msg\msg.h"
#include "plrcontext.h"
#include "plrstate.h"
#include "plrinfo.h"
#include "plrinfo_t2mi.h"
#include "plrplay.h"
#include "plrsync.h"
#include "plrsyncrecover.h"
#include "..\..\..\lib\modals\modals.h"
#include "..\..\..\widgets\modinfo\modinfo.h"
//=============================================================================================
//МАКРООПРЕДЕЛЕНИЯ ============================================================================
//
/*btnbar layout*/
#define BTNBAR_BTNS_CNT           6

#define BTNBAR_PICTURE_KEYIND     0
#define BTNBAR_NOTUSED1_KEYIND    1
#define BTNBAR_NOTUSED2_KEYIND    2
#define BTNBAR_NOTUSED3_KEYIND    3
#define BTNBAR_INFO_KEY_KEYIND    4
#define BTNBAR_NOTUSED4_KEYIND    5

#define BTNBAR_PICTURE_KEY        KEY_F1
#define BTNBAR_NOTUSED1_KEY       KEY_F2
#define BTNBAR_NOTUSED2_KEY       KEY_F3
#define BTNBAR_NOTUSED3_KEY       KEY_F4
#define BTNBAR_INFO_KEY           KEY_F5
```

```
#define BTNBAR_NOTUSED4_KEY       KEY_F6

/*workspace*/
#define WORKSPACE_WIDTH     GET_LCD_WORK_WIDTH()
#define SCREEN_HEIGHT       GET_LCD_WORK_HEIGHT()
#define WORKSPACE_HEIGHT    ( GET_LCD_WORK_HEIGHT() - GET_BTNBAR_HEIGHT() )
#define INFOFLD_H           CHINFOPANEL_HEIGHT_PX

#define TBL_VERT_OFFSET           5
#define TXTOUT_FONT_REG_14P       TXTOUT_SYNT_FONT( 14, FONT_OFFSET_GENERIC )

/*serv table*/
#define TBL_ITEM_IND_TO_ROW_NUM( _ind )      ( (uint16_t)( (_ind) + 1 ) )
#define TBL_ROW_NUM_TO_ITEM_IND( _row )      ( (_row) - 1 )
#define ITEMS_CNT_TO_TBL_ROW_CNT( _count )   ( (uint16_t)( (_count) + 1 ) )

#define TBL_ROWS_CNT               11
#define TBL_CLMNS_CNT              1

#define TBL_ROW_HH                 16

#define TBL_XX                     ( 0 )
#define TBL_YY                     ( INFOFLD_H + TBL_VERT_OFFSET )
#define TBL_WW                     ( WORKSPACE_WIDTH )
#define TBL_HH                     ( TABLE_GET_HEIGHT_VS_ROWNUM( TBL_ROWS_CNT, \
                                                                 TBL_ROW_HH,   \
                                                                 1, 0 ) )

#define TBL_CLMN_PLP_ID     0

#define TBL_CLMN_PLP_ID_WW      TABLE_GET_FREEWIDTH( TBL_WW,               \
                                                     TBL_CLMNS_CNT,        \
                                                     STD_MENU_CLMN_SPACE,  \
                                                     1, 0 )

//=============================================================================================
//ТИПЫ ========================================================================================
//
typedef enum E_SERVSTAT
{
       SERVSTAT_SERVICES_ABSENT,
       SERVSTAT_SERVICES_PRESENT,
       SERVSTAT_COUNT
} E_SERVSTAT;
```

```
typedef struct FKEYBAR_KEY_T
{
	E_FKEYBAR_KEYTYPE type;
	const CHAR *const *const name;
} FKEYBAR_KEY_T;

typedef struct LOCALDATA_T
{
	H_MODINFO h_modinfo;
	H_MODALS h_modals;
	X_TABLE* h_tbl;
} LOCALDATA_T;

//=============================================================================================
//ЛОКАЛЬНЫЙ ИНТЕРФЕЙС ==========================================================================
//
/*state functors*/
static int32_t init( H_PLRSTATE hstate, void* aux );
static int32_t final( H_PLRSTATE hstate, void* aux );
static int32_t on_feplayer_event( H_PLRSTATE hstate, void* hcontext );

/*key handlers*/
static void on_key_play( kbReactHndlT hkbreact, const kbReactParT *par, void* aux );
static void on_key_info( kbReactHndlT hkbreact, const kbReactParT *par, void* aux );
static void on_key_exit( kbReactHndlT hndl, const kbReactParT *par, void* aux );
static void on_key_tbl_nav_up( kbReactHndlT hkbreact, const kbReactParT *par, void* aux );
static void on_key_tbl_nav_down( kbReactHndlT hkbreact, const kbReactParT *par, void* aux );
static void on_key_tbl_nav_up_fast( kbReactHndlT hkbreact, const kbReactParT *par, void* aux );
static void on_key_tbl_nav_down_fast( kbReactHndlT hkbreact, const kbReactParT *par, void* aux );

/*locals*/
static LOCALDATA_T* localdata_init( void );
static void localdata_final( LOCALDATA_T* h_localdata );
static void tbl_callback( void *pObjDescr, const X_QUERY *pxQ, X_REPLY *pxR, void* aux );
static WORD get_table_marker_step( X_TABLE* htbl, bool up );

static LONG mode_modinfo_entry( H_MODALS hmanager, DWORD entity, void* opaq );
//=============================================================================================
//ПЕРЕМЕННЫЕ ===================================================================================
//

//=============================================================================================
//КОНСТАНТЫ ====================================================================================
//
/*player state: info*/
```

```
const PLRSTATE_T plrinfo_t2mi_state = { .init = init, .on_feplayer_event = on_feplayer_event, .final = final };

/*key reactor: services absent*/
static const kbReactRegT kbreact_evarr_serv_absent[] =
{
	{ KEY_EXIT | KB_ST_KEY_PRESSED,		on_key_exit },
};

/*key reactor: services present*/
static const kbReactRegT kbreact_evarr_serv_present[] =
{
	{ BTNBAR_PICTURE_KEY | KB_ST_KEY_PRESSED,   on_key_play },
	{ BTNBAR_INFO_KEY | KB_ST_KEY_PRESSED, on_key_info },
	{ KEY_ENTER | KB_ST_KEY_PRESSED,   on_key_play },

	{ KEY_EXIT | KB_ST_KEY_PRESSED,		on_key_exit },

	{ KEY_UP | KB_ST_KEY_PRESSED,	on_key_tbl_nav_up },
	{ KEY_UP | KB_ST_STICK_MODE,	on_key_tbl_nav_up },
	{ KEY_DOWN | KB_ST_KEY_PRESSED,	on_key_tbl_nav_down },
	{ KEY_DOWN | KB_ST_STICK_MODE,	on_key_tbl_nav_down },
	{ KEY_UP | KB_AUX_KEY_HOLDED | KB_ST_KEY_PRESSED,   on_key_tbl_nav_up_fast },
	{ KEY_UP | KB_AUX_KEY_HOLDED | KB_ST_STICK_MODE,    on_key_tbl_nav_up_fast },
	{ KEY_DOWN | KB_AUX_KEY_HOLDED | KB_ST_KEY_PRESSED, on_key_tbl_nav_down_fast },
	{ KEY_DOWN | KB_AUX_KEY_HOLDED | KB_ST_STICK_MODE,  on_key_tbl_nav_down_fast },
};

//key reactor: array of pointers to array
static const kbReactRegT (*kbreact_arrays[SERVSTAT_COUNT])[] = {
	[SERVSTAT_SERVICES_ABSENT] = &kbreact_evarr_serv_absent,
	[SERVSTAT_SERVICES_PRESENT] = &kbreact_evarr_serv_present,
};

//key reactor: array of sizes of array
static const uint16_t kbreact_arrays_sizes[SERVSTAT_COUNT] = {
	[SERVSTAT_SERVICES_ABSENT] = cellsof( kbreact_evarr_serv_absent ),
	[SERVSTAT_SERVICES_PRESENT] =  cellsof( kbreact_evarr_serv_present ),
};

/*fkeybar: btns types*/
static const FKEYBAR_KEY_T fkeybar_keys[SERVSTAT_COUNT][FKEYBAR_KEYS_COUNT] = {
	[SERVSTAT_SERVICES_ABSENT] = {
		[BTNBAR_PICTURE_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
		[BTNBAR_NOTUSED1_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
		[BTNBAR_NOTUSED2_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
```

```
                [BTNBAR_NOTUSED3_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
                [BTNBAR_INFO_KEY_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
                [BTNBAR_NOTUSED4_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED,  ML(empty) }
        },
        [SERVSTAT_SERVICES_PRESENT] = {
                [BTNBAR_PICTURE_KEYIND] = { FKEYBAR_KEYTYPE_CLICK, ML(play_a)},
                [BTNBAR_NOTUSED1_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
                [BTNBAR_NOTUSED2_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
                [BTNBAR_NOTUSED3_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED, ML(empty) },
                [BTNBAR_INFO_KEY_KEYIND] = { FKEYBAR_KEYTYPE_CLICK, ML(info_a) },
                [BTNBAR_NOTUSED4_KEYIND] = { FKEYBAR_KEYTYPE_NOT_USED,  ML(empty) }
        },
};

/* modalinfo entity description */
static const X_MODAL_ENTITY mod_info_entity = {
        .stimulus = NULL,
        .entry = (mode_modinfo_entry),
    .priority = IT08_NOMINAL_PRIORITY,
    .stack_4b = 4 * 1024 / 4,
    .name = "modinfo",
};

//=============================================================================================
//ИНТЕРФЕЙСНЫЕ ФУНКЦИИ ========================================================================
//

//=============================================================================================
//ЛОКАЛЬНЫЕ ФУНКЦИИ ===========================================================================
//



// Инициализация сотояниіа ---------------------------------------------------------------
static int32_t init( H_PLRSTATE hstate, void* aux )
{
        (void)hstate;
        H_PLRCONTEXT hcontext = aux;
        H_FKEYBAR hfkeybar = hcontext->hfkeybar;
        kbReactHndlT hkbreact;
        const X_FE_PLAYER_STREAM_INFO* hinfo = NULL;
        int32_t status = PLRINFO_T2MI_OK;
        const X_FE_PLAYER_DSCR* feplayerdescr = hcontext->feplayerdescr;
        E_SERVSTAT servstat;
        uint16_t tbl_mrow;
```

```
hinfo = feplayerdescr->get_stream_info( hcontext->hfeplayer );
IF_NULL_SETSTATUS_GOTO( hinfo, status, PLRINFO_T2MI_ERR_INVALID_INFO, L_FUNC_END );

servstat = ( hinfo->digital.l1post_info->NumPLPs == 0 ) ? SERVSTAT_SERVICES_ABSENT :
                                                          SERVSTAT_SERVICES_PRESENT;

hkbreact = kbreact_open( NULL, hcontext );
IF_NULL_SETSTATUS_GOTO( hkbreact, status, PLRINFO_T2MI_ERR_KBREACT_OPEN_FAILED, L_FUNC_END );
hcontext->hkbreact = hkbreact;
for( uint32_t i = 0; i < kbreact_arrays_sizes[servstat]; ++i )
{
        status = kbreact_reg( hcontext->hkbreact, &(*kbreact_arrays[servstat])[i] );
        IF_FAILED_GOTO( status == 0, L_FUNC_END );
}

hcontext->localdata = localdata_init();
LOCALDATA_T* h_localdata = hcontext->localdata;

h_localdata->h_modals = modals_open( &( const X_MODALS_CFG ){.mode = MODALS_DISPATCH_FROM_APPL},
                                     NULL );
IF_NULL_SETSTATUS_GOTO( h_localdata->h_modals, status,
                            PLRINFO_T2MI_ERR_MODALS_OPEN_FAILED, L_FUNC_END );

h_localdata->h_modinfo = modinfo_create();
IF_NULL_SETSTATUS_GOTO( h_localdata->h_modinfo, status,
                            PLRINFO_T2MI_ERR_NO_MEMORY, L_FUNC_END );

for( size_t keyid = FKEYBAR_KEY_F1; keyid <= FKEYBAR_KEY_LAST; ++keyid )
{
        fkeybar_set_type( hfkeybar, keyid, FKEYBAR_LAYER_BASIC,
                          fkeybar_keys[servstat][keyid].type );
        fkeybar_set_name( hfkeybar, keyid, FKEYBAR_LAYER_BASIC,
                          fkeybar_keys[servstat][keyid].name[gui_get_lng()] );
}

FILL_SCREEN( C_BCKG_MAIN );
chinfopanel_show( hcontext->hpanel );
fkeybar_show( hfkeybar );
TYP_TABLE_CREATE( h_localdata->h_tbl,// ID
        TBL_XX,// X
        TBL_YY,// Y
        TBL_WW,// Ширина
        TBL_HH,// Высота
        TBL_CLMNS_CNT,// Количество столбцов
```

```
                ITEMS_CNT_TO_TBL_ROW_CNT( hinfo->digital.l1post_info->NumPLPs ),// Количество строк
                ( ( uint16_t [TBL_CLMNS_CNT] ) {
                        [TBL_CLMN_PLP_ID] = TBL_CLMN_PLP_ID_WW
                        } ),// Массив ширин столбцов
                TBL_ROW_HH,// Высота строк
                tbl_callback,// Функтор обратного вызова
                hcontext,// Параметры обратного вызова
                0,// Маска статичных столбцов
                0,// Использовать гор. прокрутку
                1,// Использовать вер. прокрутку
                1,// Использовать заголовок
                1 );// Использовать курсор

        tbl_mrow = TBL_ITEM_IND_TO_ROW_NUM( hcontext->plp_ind );
        if( tbl_mrow >= ITEMS_CNT_TO_TBL_ROW_CNT( hinfo->digital.l1post_info->NumPLPs ) )
        {
                tbl_mrow = TBL_ITEM_IND_TO_ROW_NUM( 0 );
                hcontext->plp_ind = 0;
        }
        TABLE_JUMPTO( h_localdata->h_tbl, TBL_CLMN_PLP_ID, tbl_mrow );

L_FUNC_END:
        if( status != 0 )
        {
                FRW_DEBUG_PRINT( 1, "[fe\\player\\info.c]: 'init' err %d\n", status );
        }

        return status;
}

// Финализация состояния ---------------------------------------------------------------------------
static int32_t final( H_PLRSTATE hstate, void* aux )
{
        (void)hstate;
        H_PLRCONTEXT hcontext = aux;


        kbreact_close( hcontext->hkbreact );
        fkeybar_hide( hcontext->hfkeybar );
        chinfopanel_hide( hcontext->hpanel );
        localdata_final( hcontext->localdata );


        return PLRINFO_T2MI_OK;
}
```

```
// Событие от фронт-энда плеера ------------------------------------------------------------------
static int32_t on_feplayer_event( H_PLRSTATE hstate, void* aux )
{
	(void)hstate;
	H_PLRCONTEXT hcontext = aux;
	int32_t status = PLRINFO_T2MI_OK;


	switch( hcontext->feplayer_event )
	{
	case FE_PLAYER_EVENT_STREAM_UNLOCKED:
		plrcontext_switch_state_to( hcontext, &plrsyncrecover_state );
		break;

	case FE_PLAYER_EVENT_FAILED:
		status = PLRINFO_T2MI_ERR_FE_PLAYER_FAILED;
		break;

	case FE_PLAYER_EVENT_CAM_READY:
		hcontext->iscamready = true;
		break;

	case FE_PLAYER_EVENT_CAM_NOT_READY:
		hcontext->iscamready = false;
		break;

	//@todo stream end handler
	case FE_PLAYER_EVENT_STREAM_END:
		break;

	case FE_PLAYER_EVENT_STREAM_LOCKED:
	case FE_PLAYER_EVENT_STREAM_INFO_UPDATED:
	case FE_PLAYER_EVENT_PROGRAM_SELECTED:
	case FE_PLAYER_EVENT_SOUND_AVAILABLE:
	case FE_PLAYER_EVENT_SOUND_NOT_AVAILABLE:
		break;

	case FE_PLAYER_EVENT_VOID:
		break;
	}

	return status;
}

// Services table callback -----------------------------------------------------------------------
```

```
static void tbl_callback( void *pObjDescr, const X_QUERY *pxQ, X_REPLY *pxR, void* aux )
{
	static const CHAR* const* str_clmns_names[] =
	{
		[TBL_CLMN_PLP_ID] = ML( name_plp_table_hdr )
	};

	(void) pObjDescr;
	int32_t status = PLRINFO_T2MI_OK;


	pxR->xData.pvFont   = TXTOUT_FONT_REG_14P;
	pxR->fDataType      = DT_TEXT;

	if( pxQ->wRow == 0 )
	{
		pxR->wTextC = C_TXT_MAIN;
		pxR->wCellC = C_BCKG_INACTIVE;
		pxR->fAlign = pxQ->bClmn < 3? ALIGN_LEFT : ALIGN_CENTER;
		pxR->xData.strText = str_clmns_names[pxQ->bClmn][gui_get_lng()];
		pxR->fDeleteStrBuf  = 0;
	}
	else
	{
		char* str = NULL;
		H_PLRCONTEXT hcontext = aux;
		int32_t itemind = TBL_ROW_NUM_TO_ITEM_IND( pxQ->wRow );
		const X_FE_PLAYER_STREAM_INFO* hinfo = NULL;
		const X_FE_PLAYER_DSCR* feplayerdescr = hcontext->feplayerdescr;
		const tTV_DVBT2_PLP* hplpinfo;

		hinfo = feplayerdescr->get_stream_info( hcontext->hfeplayer );
		IF_NULL_SETSTATUS_GOTO( hinfo, status, PLRINFO_T2MI_ERR_INVALID_INFO, L_FUNC_END );

		hplpinfo = &hinfo->digital.plp_info[itemind];

		pxR->wTextC = ( pxQ->wRow == pxQ->wMarkedRow ) ? C_TXT_SELECTED : C_TXT_MAIN;
		pxR->wCellC = ( pxQ->wRow == pxQ->wMarkedRow ) ? C_BCKG_SELECTED : C_BCKG_MAIN;
		pxR->fAlign = ALIGN_LEFT;

		switch( pxQ->bClmn )
		{
			case TBL_CLMN_PLP_ID:
			{
				str = pvPortMalloc( 7 );
```

```
                        sprintf( str, "PLP%d", hplpinfo->Id );
                }
                break;
        }

        pxR->xData.strText = ( str != NULL ) ? str : pxR->xData.strText;
        pxR->fDeleteStrBuf = ( str != NULL ) ? 1 : 0;
    }

L_FUNC_END:
    if( status != PLRINFO_T2MI_OK )
    {
        FRW_DEBUG_PRINT( 1, "[fe\\player\\info.c]: 'tbl_callback' err %d\n", status );
    }
}

// Выход из автомата ------------------------------------------------------------------------------
static void on_key_exit( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
    (void)hkbreact;
    (void)par;
    H_PLRCONTEXT hcontext = aux;


    hcontext->working = false;
}

// Просмотр изображения ---------------------------------------------------------------------------
static void on_key_play( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
    (void)hkbreact;
    (void)par;
    H_PLRCONTEXT hcontext = aux;
    const X_FE_PLAYER_STREAM_INFO* hinfo = NULL;
    const X_FE_PLAYER_DSCR* feplayerdescr = hcontext->feplayerdescr;
    const tTV_DVBT2_PLP* hplpinfo;
    int32_t status = PLRINFO_T2MI_OK;

    hinfo = feplayerdescr->get_stream_info( hcontext->hfeplayer );
    IF_NULL_SETSTATUS_GOTO( hinfo, status, PLRINFO_T2MI_ERR_INVALID_INFO, L_FUNC_END );

    hplpinfo = &hinfo->digital.plp_info[hcontext->plp_ind];
    hcontext->plp_id = hplpinfo->Id;
```

```
	if( feplayerdescr->is_file_player( hcontext->hfeplayer ) )
	{
		status = feplayerdescr->set_position( hcontext->hfeplayer, 0 );
		IF_FAILED_SETSTATUS_GOTO(status == 0, status, PLRINFO_T2MI_ERR_SET_POSITION_FAILED, L_FUNC_END);
	}

	feplayerdescr->select_plp( hcontext->hfeplayer, hcontext->plp_id );
	hcontext->plp_selected = true;
	plrcontext_switch_state_to( hcontext, &plrsync_state );

L_FUNC_END:
	if( status != PLRINFO_T2MI_OK )
	{
		FRW_DEBUG_PRINT( 1, "[fe\\player\\info.c]: 'on_key_play' err %d\n", status );
	}
}

// Просмотр информации о потоке ------------------------------------------------------------------
static void on_key_info( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
	(void)hkbreact;
	(void)par;
	H_PLRCONTEXT hcontext = (H_PLRCONTEXT)aux;
    LOCALDATA_T* h_localdata = hcontext->localdata;

    const X_FE_PLAYER_DSCR* feplayerdescr = hcontext->feplayerdescr;
    const X_FE_PLAYER_STREAM_INFO* raw_info =
                    feplayerdescr->get_stream_info( hcontext->hfeplayer );

	tTV_EVDATA_LOCK info;
	memset( &info, 0, sizeof(info)) ;

	info.System = TV_SYSTEM_DVBT2;
	//TODO* common plp?
	memset( &info.SystemParam.DVBT2_Param.PLPInfoCommon, 0,
		sizeof(info.SystemParam.DVBT2_Param.PLPInfoCommon) );

	for(int i = 0; i < raw_info->digital.l1post_info->NumPLPs; i++ ) {
		if( raw_info->digital.plp_info[i].Type == TV_DVBT2_PLP_TYPE_COMMON ) {
			info.SystemParam.DVBT2_Param.PLPInfoCommon =
                                    raw_info->digital.plp_info[i];
	      break;
		}
    }
```

```
	info.SystemParam.DVBT2_Param.PLPInfoData =
				raw_info->digital.plp_info[hcontext->plp_ind];
	info.SystemParam.DVBT2_Param.L1Pre = *raw_info->digital.l1pre_info;
	info.SystemParam.DVBT2_Param.L1Post = *raw_info->digital.l1post_info;

	modinfo_set_locked_t2mi( h_localdata->h_modinfo, (const tTV_EVDATA_LOCK*)&info );

	modals_entity_launch_offlist( h_localdata->h_modals,
				&mod_info_entity,
				h_localdata,
				true );

}

// вход в виджет modinfo, см. modinfo_entry() в modinfo.h-----------------------------------------
static LONG mode_modinfo_entry( H_MODALS hmanager, DWORD entity, void* opaq )
{
  (void)hmanager;
  (void)entity;

  LOCALDATA_T* h_localdata = (LOCALDATA_T*)opaq;

  return modinfo_entry(h_localdata->h_modinfo);
}

//-----------------------------------------------------------------------------------------------
static void on_key_tbl_nav_up( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
	(void)hkbreact;
	(void)par;
	H_PLRCONTEXT hcontext = aux;
	LOCALDATA_T* h_localdata = hcontext->localdata;
	X_TABLE* htable = (X_TABLE*)h_localdata->h_tbl;
	uint16_t tbl_mrow;


	TABLE_SCROLL( htable, TABLE_CUR_UP, 1 );
	TABLE_GET_PAR( htable, TABLE_REQ_MARKED_ROW, &tbl_mrow );
	hcontext->plp_ind = TBL_ROW_NUM_TO_ITEM_IND( tbl_mrow );
}

//-----------------------------------------------------------------------------------------------
static void on_key_tbl_nav_down( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
	(void)hkbreact;
```

```
	(void)par;
	H_PLRCONTEXT hcontext = aux;
	LOCALDATA_T* h_localdata = hcontext->localdata;
	X_TABLE* htable = (X_TABLE*)h_localdata->h_tbl;
	uint16_t tbl_mrow;


	TABLE_SCROLL( htable, TABLE_CUR_DOWN, 1 );
	TABLE_GET_PAR( htable, TABLE_REQ_MARKED_ROW, &tbl_mrow );
	hcontext->plp_ind = TBL_ROW_NUM_TO_ITEM_IND( tbl_mrow );
}

//-----------------------------------------------------------------------------------------------
static void on_key_tbl_nav_up_fast( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
	(void)hkbreact;
	(void)par;
	H_PLRCONTEXT hcontext = aux;
	LOCALDATA_T* h_localdata = hcontext->localdata;
	X_TABLE* htable = (X_TABLE*)h_localdata->h_tbl;
	uint16_t tbl_mrow;


	TABLE_SCROLL( htable, TABLE_CUR_UP, get_table_marker_step( htable, true ) );
	TABLE_GET_PAR( htable, TABLE_REQ_MARKED_ROW, &tbl_mrow );
	hcontext->plp_ind = TBL_ROW_NUM_TO_ITEM_IND( tbl_mrow );
}

//-----------------------------------------------------------------------------------------------
static void on_key_tbl_nav_down_fast( kbReactHndlT hkbreact, const kbReactParT *par, void* aux )
{
	(void)hkbreact;
	(void)par;
	H_PLRCONTEXT hcontext = aux;
	LOCALDATA_T* h_localdata = hcontext->localdata;
	X_TABLE* htable = (X_TABLE*)h_localdata->h_tbl;
	uint16_t tbl_mrow;


	TABLE_SCROLL( htable, TABLE_CUR_DOWN, get_table_marker_step( htable, false ) );
	TABLE_GET_PAR( htable, TABLE_REQ_MARKED_ROW, &tbl_mrow );
	hcontext->plp_ind = TBL_ROW_NUM_TO_ITEM_IND( tbl_mrow );
}

// Get table marker step ------------------------------------------------------------------------
```

```
//
static WORD get_table_marker_step( X_TABLE* htbl, bool up )
{
	WORD step;
	WORD rows;
	WORD sel;


	TABLE_GET_PAR( htbl, TABLE_REQ_VISBL_ROW_EXCHEAD, &step );
	step /= 2;
	TABLE_GET_PAR( htbl, TABLE_REQ_MARKED_ROW_EXCHEAD, &sel );

	if( up )
	{
		step = sel > step? step : sel;
	}
	else
	{
		TABLE_GET_PAR( htbl, TABLE_REQ_ROW_NUM_EXCHEAD, &rows );
		step = rows - 1 - sel > step? step : rows - 1 - sel;
	}

	return step;
}

static LOCALDATA_T* localdata_init( void )
{
	LOCALDATA_T* h_localdata = NULL;

	h_localdata = pvPortMalloc( sizeof(*h_localdata) );
	IF_NULL_GOTO( h_localdata, L_FUNC_END );

	memset( h_localdata, 0, sizeof(*h_localdata) );

L_FUNC_END:
	return h_localdata;
}

static void localdata_final( LOCALDATA_T* h_localdata )
{
	if( h_localdata != NULL )
	{
		if( h_localdata->h_modals != NULL )
		{
			modals_close( h_localdata->h_modals );
```

```
            }

            if( h_localdata->h_modinfo != NULL )
            {
                    modinfo_destroy( h_localdata->h_modinfo );
            }

            if( h_localdata->h_tbl != NULL )
            {
                    TABLE_DESTROY( h_localdata->h_tbl )
            }

            vPortFree( h_localdata );
      }
}
```