Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

На правах рукописи

УДК 004.052.42

ПОТАПОВ ЕВГЕНИЙ ВЛАДИМИРОВИЧ

РАЗРАБОТКА ВЕРИФИКАЦИОННОГО IP БЛОКА ИНТЕРФЕЙСА CSI

Выпускная квалификационная работа бакалавра

Направление подготовки 09.03.01 Информатика и вычислительная техника

|  |
| --- |
| Выпускная квалификационная работа защищена  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г. |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Секретарь ГЭК \_\_\_\_\_\_\_\_\_\_\_\_\_ |

г. Москва

2021

(оборот титульного листа)

Студент-дипломник \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Потапов Е.В. /

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Скитев А.А. /

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Хайлов А.Н. /

Заведующий кафедрой №12 \_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Иванов М.А. /

# АННОТАЦИЯ

Данная работа посвящена разработке верификационного IP для интерфейса CSI. Работа состоит из введения, трех глав, заключения, списка литературы и 1 приложения, изложена на 103 страницах, содержит 42 рисунка, 1 таблицу.

Во введении обосновывается актуальность работы, ставится цель и задачи работы.

В первой главе проводится обзор интерфейса CSI, верификационной методологии UVM, а также окружения VIVO, в составе которого будет использоваться VIP.

Вторая глава посвящена разработке структуры верификационного IP, логики работы его компонентов. Приведены алгоритмы и конечные автоматы для различных составляющих VIP. Также во второй главе установлен план тестирования.

Третья глава описывает процесс реализации разработанного VIP на языке System Verilog, его интеграции в окружение VIVO, а также демонстрирует результаты работы верификационного IP.

В заключении подводится итог проделанной работе.

Ознакомление с работой будет полезно специалистам, планирующим верификацию интерфейса CSI или использующим верификационное окружение VIVO.

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ**

**«МИФИ»**

**ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ**

**Кафедра «Компьютерные системы и технологии»**

Направление 09.03.01 Группа Б17-503

«УТВЕРЖДАЮ»

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / М.А. Иванов /

" 01 " марта 2021 г.

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

**(ДИПЛОМНЫЙ ПРОЕКТ)**

Фамилия, имя, отчество студента: **Потапов Евгений Владимирович**

Тема работы: **Разработка верификационного IP блока интерфейса CSI**

Срок сдачи студентом готовой работы: **1 июня 2021 г.**

Руководитель работы: **Скитев Андрей Андреевич,**

**ст. преподаватель ИИКС НИЯУ МИФИ**

Место выполнения: **НИЯУ МИФИ**

**1. Исходные данные:**

Верификационное IP ядро (VIP) предназначено для выполнения тестирования и отладки высокопроизводительного интерфейса передачи изображений CSI. VIP должен функционировать в соответствии со спецификацией не ниже версии MIPI CSI-2 v1.0.

VIP должен соответствовать следующим требованиям:

1. Быть разработанным по методологии UVM.
2. UVM-драйвер должен формировать транзакции в соответствии со спецификацией протокола.
3. UVM-монитор должен обеспечивать проверку транзакций на соответствие спецификации MIPI CSI-2 v1.0.
4. VIP должен поддерживать передачу данных как минимум по одной и четырем линиям.
5. Минимум VIP должен проверять работу в режиме RAW8

**2. Содержание задания:**

*а) обзорная часть:*

1. Изучить методологию верификации UVM
2. Изучить спецификацию интерфейса CSI.

*б) расчетно-конструкторская, теоретическая, технологическая части:*

1. Разработать структурную схему VIP в соответствии с методологией UVM
2. Разработать алгоритмы работы основных компонент VIP.
3. Определить механизмы конфигурирования VIP.
4. Разработать план проведения тестирования.

*в) экспериментальная часть:*

1. Реализовать компоненты VIP на языке SystemVerilog.
2. Выполнить автономную отладку разработанных компонент.
3. Выполнить комплексную отладку системы.
4. Составить руководство пользователя.

**3. Основная литература:**

1. Universal Verification Methodology (UVM) 1.2 User’s Guide Сайт Accellera URL: www.accellera.org/images/downloads/standards/uvm/uvm\_users\_guide\_1.2.pdf (дата обращения: 01.02.21).
2. Spear, Chris, Tumbush, Greg «SystemVerilog for Verification» - Springer, 2012 – 465c.

**4. Отчетный материал:**

*пояснительная записка;*

*макетно-экспериментальная часть:*

1. Листинги VIP-а на электронном носителе
2. Руководство пользователя

Дата выдачи задания: 15 февраля 2021 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / А.А. Скитев /

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Е.В. Потапов /

# СОДЕРЖАНИЕ

[АННОТАЦИЯ 3](#__RefHeading___Toc17237_135013858)

[СОДЕРЖАНИЕ 6](#__RefHeading___Toc17239_135013858)

[ВВЕДЕНИЕ 8](#__RefHeading___Toc17241_135013858)

[Глава 1 Теоретические основы разработки верификационного IP-блока 9](#__RefHeading___Toc17243_135013858)

[1.1 Обзор интерфейса CSI 9](#__RefHeading___Toc17245_135013858)

[1.1.1 Структура интерфейса CSI 10](#__RefHeading___Toc17247_135013858)

[1.1.2 Особенности работы интерфейса 13](#__RefHeading___Toc17249_135013858)

[1.1.3 Обзор MIPI D-PHY 14](#__RefHeading___Toc17251_135013858)

[1.1.4 Порядок работы интерфейса CSI 17](#__RefHeading___Toc17253_135013858)

[1.2 Обзор методологии UVM 18](#__RefHeading___Toc17255_135013858)

[1.2.1 Назначение и подходы к верификации 18](#__RefHeading___Toc17257_135013858)

[1.2.2 Обзор методологии UVM 20](#__RefHeading___Toc17259_135013858)

[1.3 Обзор используемого окружения 25](#__RefHeading___Toc17261_135013858)

[1.4 Обзор существующих верификационных IP интерфейса CSI 30](#__RefHeading___Toc17263_135013858)

[1.5 Выводы главы 1 31](#__RefHeading___Toc17265_135013858)

[Глава 2 Разработка верификационного IP 33](#__RefHeading___Toc17267_135013858)

[2.1 Определение структуры верификационного IP 33](#__RefHeading___Toc17269_135013858)

[2.1.1 Соединение VIP CSI с компонентами библиотеки «VIVO» 33](#__RefHeading___Toc17271_135013858)

[2.1.2 Причины включения уровня LLP в агент VIVO 34](#__RefHeading___Toc17273_135013858)

[2.1.3 Организация внутренней структуры внешнего агента 37](#__RefHeading___Toc17275_135013858)

[2.2 Разработка алгоритмов работы компонентов VIP 39](#__RefHeading___Toc17277_135013858)

[2.2.1 Разработка алгоритмов работы секвенсоров и драйвера 39](#__RefHeading___Toc17279_135013858)

[2.2.2 Разработка алгоритмов работы компонентов монитора 44](#__RefHeading___Toc17281_135013858)

[2.2.3 Разработка монитора интерфейса D-PHY 49](#__RefHeading___Toc17283_135013858)

[2.3 Разработка механизма конфигурации VIP 50](#__RefHeading___Toc17285_135013858)

[2.4 Составление плана тестирования VIP 53](#__RefHeading___Toc17287_135013858)

[2.5 Выводы главы 2 54](#__RefHeading___Toc17289_135013858)

[Глава 3 Реализация и тестирование 55](#__RefHeading___Toc17291_135013858)

[3.1 Реализация компонентов VIP на языке System Verilog 55](#__RefHeading___Toc17293_135013858)

[3.1.1 Реализация классов транзакций 55](#__RefHeading___Toc17295_135013858)

[3.1.2 Реализация секвенсоров и драйвера 58](#__RefHeading___Toc17297_135013858)

[3.1.3 Реализация компонентов монитора 61](#__RefHeading___Toc17299_135013858)

[3.1.4 Реализация конфигурационных объектов 66](#__RefHeading___Toc17301_135013858)

[3.2 Интеграция разработанного VIP в библиотеку VIVO 67](#__RefHeading___Toc17303_135013858)

[3.3 Тестирование разработанного VIP 70](#__RefHeading___Toc17305_135013858)

[3.3.1 Разработка тестового окружения 70](#__RefHeading___Toc17307_135013858)

[3.3.2 Тестирование работы драйвера с монитором 71](#__RefHeading___Toc17309_135013858)

[3.3.3 Тестирование драйвера с IP CSI 74](#__RefHeading___Toc17311_135013858)

[3.4 Составление руководства пользователя 75](#__RefHeading___Toc17313_135013858)

[3.4 Выводы главы 3 75](#__RefHeading___Toc17315_135013858)

[ЗАКЛЮЧЕНИЕ 76](#__RefHeading___Toc17317_135013858)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 77](#__RefHeading___Toc17319_135013858)

[ПРИЛОЖЕНИЯ 79](#__RefHeading___Toc17321_135013858)

[Приложение А – Руководство пользователя 79](#__RefHeading___Toc17331_135013858)

# ВВЕДЕНИЕ

Цифровые устройства записи, обработки и передачи видео давно стали важной частью многих более сложных устройств. В современном мире камеры различных типов присутствуют практически везде – начиная с потребительских устройств вроде телефонов, планшетов, ПК, заканчивая спутниками и электронными микроскопами.

Чем больше камер находится в устройстве и чем выше запрашиваемая частота кадров для каждой, тем важнее иметь возможность быстро передавать большой объём данных от камер к обрабатывающему устройству. К тому же, зачастую передаваемые данные могут иметь различные форматы.

Большинство популярных интерфейсов имеет серьёзные ограничения как по скорости работы, так и по функциональности и поддерживаемым режимам передачи видео. Необходим универсальный интерфейс, который может использоваться для любых видов видеоданных.

Именно с такой целью в 2005 году был разработан интерфейс MIPI CSI-2, основанный на физическом интерфейсе MIPI D-PHY / C-PHY. Интерфейс поддерживает множество функций, таких как разбиение видеопотока на пакеты, проверка целостности пересылаемых пакетов, передача коротких пакетов произвольного назначения. Протокол работает с множеством различных типов изображений, к тому же, есть возможность задать новые типы данных, если их можно привести к целому количеству байт.

Для верификации устройств, использующих данный интерфейс, требуется верификационный агент, позволяющий отправлять и принимать транзакции различных видов.

# Глава 1 Теоретические основы разработки верификационного IP-блока

Целью работы была разработка блока, поддерживающего основные функции интерфейса CSI. Несмотря на то, что протокол подразумевает использование интерфейса I2C для доступа к регистрам камеры и управления ей, его использование не является необходимым для проверки передачи видео камерой или приёмником. Кроме того, существует множество VIP (разработанных компаниями Cadence, Synopsis и др.) для интерфейса I2C.

# 1.1 Обзор интерфейса CSI

C распространением мобильных телефонов и смартфонов и повышением качества фото и видеокамер возникла потребность в универсальном интерфейсе для соединения камеры и устройства. В 2005 году Mobile Industry Processor Interface (MIPI) Alliance опубликовали спецификацию Camera Serial Interface 2 (CSI-2). Последовательный интерфейс поддерживал высокую скорость передачи данных и обеспечивал низкое потребление энергии, благодаря чему быстро обрёл популярность.

Несмотря на то, что CSI-3, предназначенный для нового физического интерфейса M-PHY, был выпущен в 2012, стандарт CSI-2 продолжает набирать популярность, охватывая новые устройства [1].

В апреле 2017 года была выпущена вторая версия стандарта, поддерживающая новые форматы данных (RAW-16 и RAW-20), а также новые возможности (например, дифференциальная импульсно-кодовая модуляция для уменьшения локальной избыточности). В сентябре 2019 года публикована третья версия, поддерживающая формат RAW-24.

Интерфейс CSI-2, изначально разработанный для камер мобильных телефонов, благодаря своей пропускной способности на данный момент может использоваться практически в любых сферах. Рассмотрим основные особенности протокола.

В данной работе используется первая версия интерфейса, поскольку с ней совместимо тестируемое устройство.

Рассмотрим структуру и основные особенности интерфейса.

# 1.1.1 Структура интерфейса CSI

Важным отличием интерфейса CSI от множества других видео интерфейсов является его разбиение на слои (уровни) [2]. Общая схема взаимодействия между слоями приведена на рисунке 1.1.

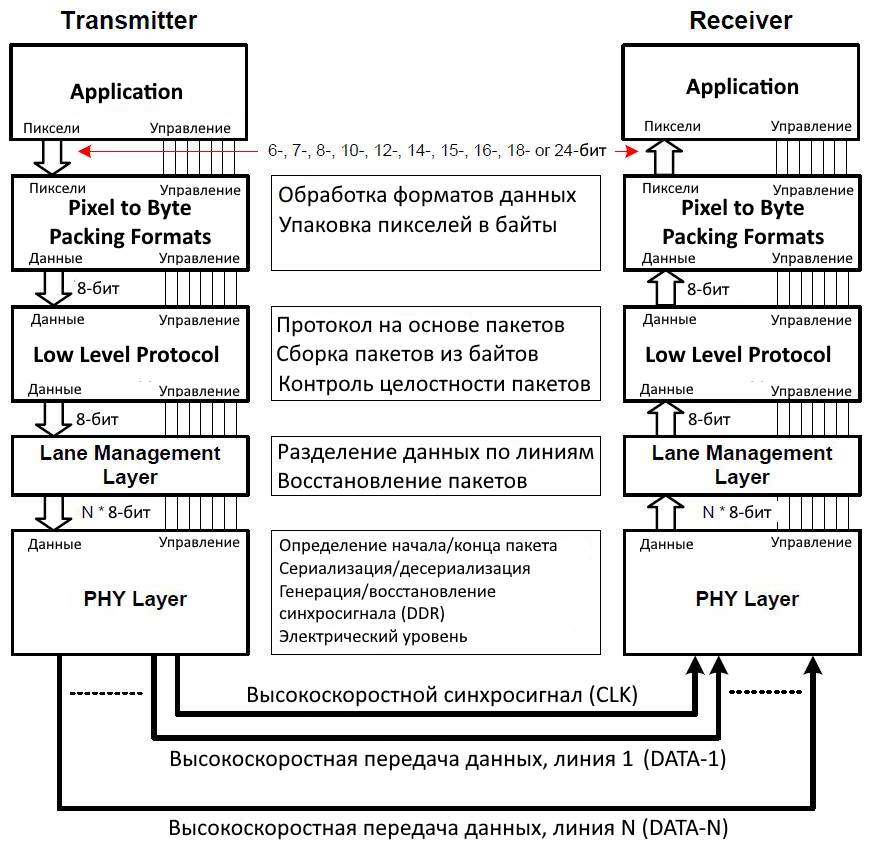


Рисунок 1.1 – Слои интерфейса CSI

Уровень приложения (Application Layer) отвечает за интерпретацию и кодировку данных, получаемых / отправляемых камерой.

Уровень упаковки пикселей в байты (Pixel/Byte Packing/Unpacking Layer), как следует из названия, обеспечивает поддержку протоколом различных форматов данных, таких как RAW, RGB, YCbCr и множество других. Для каждого формата устанавливается свой порядок упаковки. Пример упаковки одного пикселя формата RGB555 в два байта приведён на рисунке 1.2.

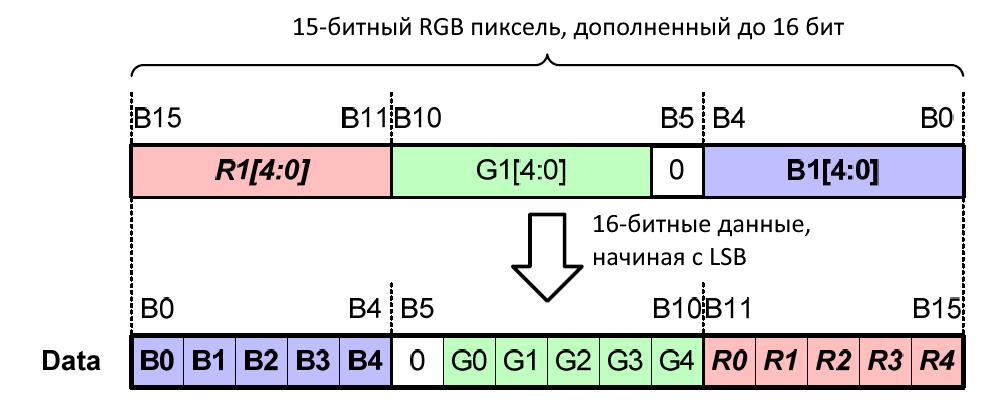


Рисунок 1.2 – Пример упаковки пикселя в слово из двух байтов

Результатом работы данного уровня является поток байтов, готовых к включению в пакет. Протокол поддерживает использование новых типов данных и алгоритмов упаковки, выделяя под них группу номеров.

Протокол нижнего уровня (Low Level Protocol, LLP) управляет сборкой пакетов. Пакеты, разделённые на байты, подаются на следующий уровень начиная с наименее значимых бит.

Уровень управления линиями (Lane Management) обеспечивает разделение данных между несколькими линиями (дифференциальными парами) для ускорения пересылки информации пакета. Пример работы этого уровня для одной и четырёх линий приведён на рисунке 1.3.

Физическим уровнем интерфейса (PHY) является последовательный интерфейс MIPI D-PHY, более подробно рассмотренный ниже. PHY обеспечивает подключение камеры к устройству, управление электрическими уровнями каждой дифференциальной пары.

На физическом уровне также происходит генерация тактового сигнала (PLL), сериализация/десериализация получаемых или отправляемых байтов (в режиме High-Speed). Кроме того, на PHY может возлагаться задача определения начала и конца полученного пакета данных.

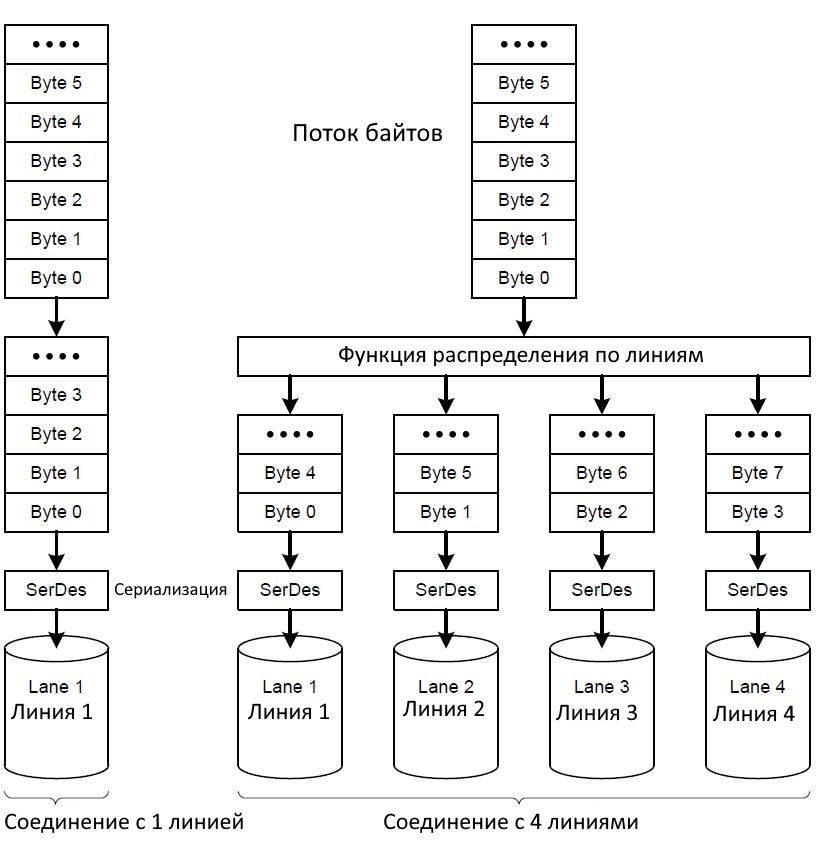


Рисунок 1.3 – Разделение данных по линиям

Пакеты разделяются на 2 типа – короткие и длинные. Короткие включают в себя только заголовок в то время, как длинные пакеты также имеют поле данных и контрольную сумму. Пример длинного пакета приведён на рисунке 1.4.

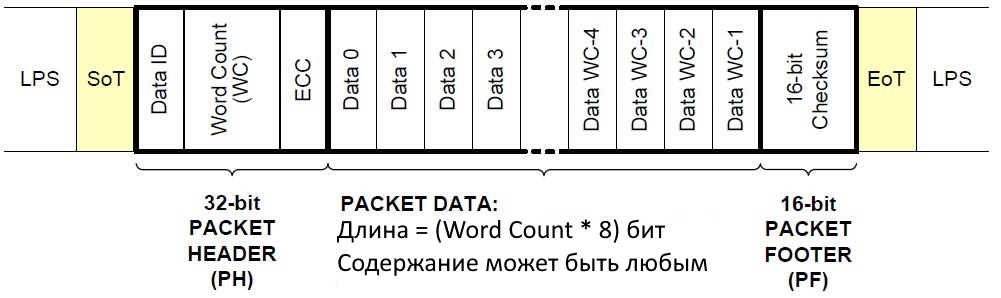


Рисунок 1.4 – Структура пакета данных

В пакет входят следующие поля:

1) Data ID – байт, состоящее из идентификатора виртуального канала (понятие виртуальных каналов объяснено ниже) и кода типа данных. Тип данных позволяет как различать разные форматы информации изображения, так и задавать управляющие короткие пакеты (такие, как пакеты синхронизации, указывающие начало и окончание кадра или строки);

2) Word Count – объём данных в пакете в длинных пакетах, или любая другая информация в коротких пакетах. В частности, в пакетах синхронизации в этом поле передаётся номер кадра или строки;

3) ECC – код коррекции ошибок, позволяющий исправить одну ошибку при передаче заголовка или обнаружить (без исправления) две ошибки;

4) Data – упакованная в байты информация изображения;

5) CRC – 16-битная контрольная сумма, рассчитываемая для данных.

# 1.1.2 Особенности работы интерфейса

Физический интерфейс (PHY) может иметь любое количество линий данных, обычно 1-4. Масштабируемость позволяет в разы увеличить скорость передачи информации за счёт увеличения занимаемого места на кристалле.

Начало отправки пакета на всех линиях происходит синхронно, окончание может быть сдвинуто на один байт в случаях, когда число байт для отправки не кратно количеству линий.

Кроме разделения на линии, в CSI используется разделение на виртуальные каналы (до четырёх), что показано на рисунке 1.5. Внутри каждого виртуального канала должен сохраняться между пакетами формат данных, а также порядок нумерации строк и кадров. Данные из различных виртуальных каналов могут быть перемешаны; номер виртуального канала указывается в каждом отправленном пакете. Таким образом, через один интерфейс могут передаваться данные от разных источников.

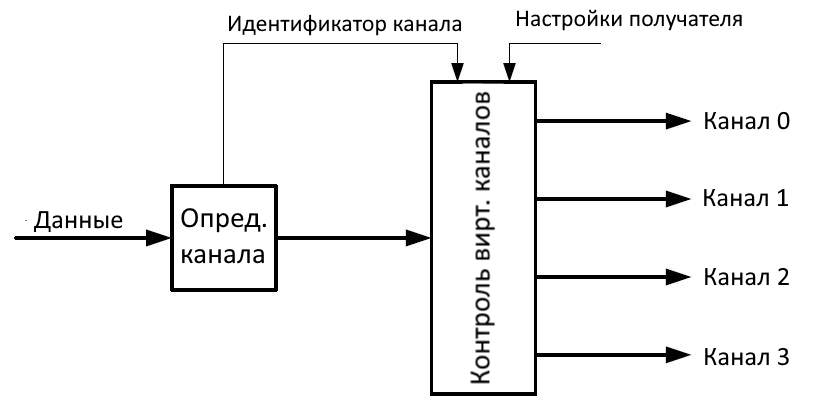


Рисунок 1.5 – Схема работы виртуальных каналов на приёмнике

# 1.1.3 Обзор MIPI D-PHY

Физический интерфейс D-PHY разработан для использования в мобильных устройствах и отличается высокой скоростью передачи данных и низким энергопотреблением в простое [3]. Несмотря на то, что D-PHY является двунаправленным интерфейсом и поддерживает изменение направления передачи данных (Bi-directional turnaround), для CSI используется односторонний режим работы, поскольку видеопоток всегда передаётся от камеры к устройству.

Интерфейс состоит из нескольких дифференциальных пар (линий) – одна передаёт тактовый сигнал (CLK Lane), и от одной до четырёх используются для передачи данных (рис. 1.6).

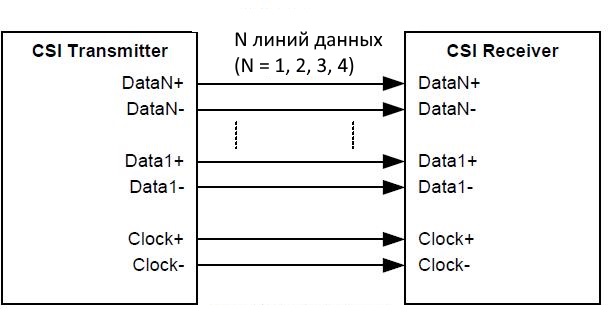


Рисунок 1.6 – Интерфейс D-PHY

Каждая линия находится в одном из трёх режимов, показанных на рис. 1.7. Режим пониженного энергопотребления (Low-Power, LP), обозначенный на рисунке синим, отличается редкими изменениями уровня сигнала (период не менее 50 нс) и используется для управления шиной. Большую часть времени, когда данные для передачи отсутствуют, линии находятся именно в этом состоянии. Всего возможно четыре состояния дифференциальной пары – LP00, LP01, LP10, LP11, обозначающие состояние + и – соответственно.

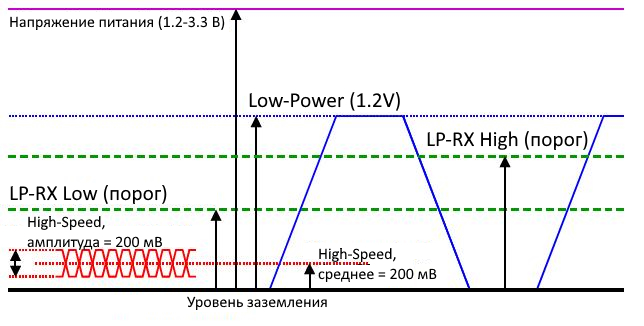


Рисунок 1.7 – Возможные состояния сигнала D-PHY

Режим высокоскоростной передачи данных (High-Speed, HS) позволяет передавать информацию с частотой, ограниченной только физическими параметрами устройства. В данном режиме линия CLK передаёт тактовый сигнал, а линии данных подстраиваются таким образом, чтобы данные были актуальны на каждом фронте синхросигнала (таким образом реализуется удвоенная скорость передачи данных). Линии всегда находятся в противоположных состояниях (HS0 = 01, HS1 = 10 соответственно).

Отдельно можно выделить режим под названием Escape Mode (EM), при котором линия данных (для CLK такого режима нет) всё ещё находится в состоянии Low-Power, но доступны новые функции, такие как:

– Low-Power Data Transmission (LPTD) – передача данных с низкой скоростью без переключения в High-Speed. Не используется в CSI.

– Ultra-Low Power State (ULPS) – режим, при котором линии длительное время находятся в LP00, что позволяет ещё сильнее понизить энергопотребление, когда устройство не используется. Обычно применяется, когда камера переводится в режим сна путём команд через управляющий интерфейс.

Важно отметить, что у CLK линии также есть режим ULPS, переход в который осуществляется более простым образом (без входа в EM).

Поскольку в задачи физического уровня входит работа с асинхронными аналоговыми сигналами, а также генерация PLL, зачастую используются внешние модули D-PHY. Примером может служить IP-модуль, бесплатно распространяемый Lattice Semiconductors. В отличие от множества других, данный IP не задаёт временные параметры работы интерфейса самостоятельно. Сгенерированный модуль позволяет пользователю напрямую управлять режимом работы интерфейса и уровнем сигнала, что расширяет возможности для верификации различных устройств с помощью этого PHY.

# 1.1.4 Порядок работы интерфейса CSI

Общую схему функционирования CSI можно описать следующим образом. Линия CLK либо постоянно находится в состоянии High-Speed (непрерывный тактовый сигнал, от англ. Continuous clock behavior), либо переключается в него перед каждой транзакцией, находясь в режиме Low-Power остальное время (Non-Continuous clock behavior). Пример передачи данных в конфигурации с одной линией приведён на рис. 1.8.

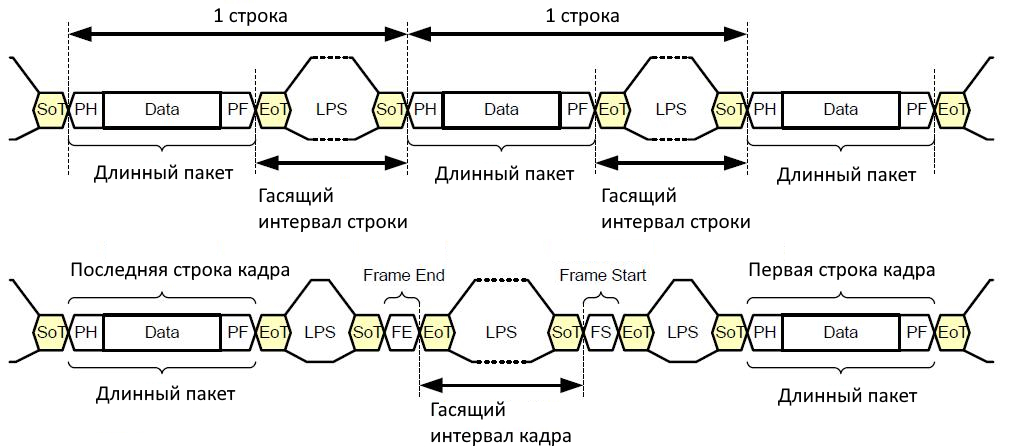


Рисунок 1.8 – Пример работы интерфейса CSI

На рис. 1.8 используются следующие обозначения:

– PH, PF – заголовок и конец пакета,

– Data – поле данных пакета,

– FS, FE – пакеты с информацией о начале и конце кадра,

– SoT (Start of Transaction) – процедура начала транзакции,

– EoT (Start of Transaction) – процедура конца транзакции,

– LPS (Low-Power State) – состояние пониженного энергопотребления.

Линии данных находятся в режиме низкого энергопотребления большую часть времени. При передаче информации изображения все линии одновременно переходят в режим высокоскоростной передачи данных (High-Speed) после процедуры начала транзакции.

После окончания этой процедуры и отправки синхронизирующей последовательности (для настройки приёмника) происходит отправка данных пакета, равномерно распределённого по всем линиям. В одном пакете передаётся либо информация о начале/конце кадра или строки, либо информация одной строки целиком, упакованная определённым образом. Сразу после окончания передачи линии данных переводятся обратно в состояние пониженного энергопотребления (Low-Power) Когда камера длительное время не используется, все линии могут быть переключены в состояние ULPS.

# 1.2 Обзор методологии UVM

# 1.2.1 Назначение и подходы к верификации

Из-за сложности интерфейса CSI, и, в особенности, его физического уровня возникает вопрос о проверке работоспособности (верификации) устройств, включающих в себя этот интерфейс. Обнаружение ошибок в работе устройства после выхода в печать может привести к большим денежным и трудовым затратам. Поэтому необходимо комплексное тестирование устройства, для чего используется верификация.

Верификация [4] – доказательство, что разработанный алгоритм, или, в данном случае, RTL-модель работает, и при этом верно.

Верификацию можно разделить на следующие основные типы:

1) функциональная верификация – проверка соответствия функций, выполняемых устройством, ожидаемым результатам., позволяющая выявить несоответствия работы проектируемого модуля заданной спецификации или алгоритму, своевременно выявить и отладить ошибки;

2) статический анализ кода – проверка исходного кода в соответствии с выбранными критериями на соблюдение правил использования языка;

3) физическая верификация – проверка на соблюдение технологических норм и соответствия физических и логических представлений о работе устройства;

4) прототипирование – проверка работоспособности устройства после его загрузки на ПЛИС.

Для первых трёх видов верификации используется моделирование модулей, написанных на одном из языков описания аппаратуры – VHDL или System Verilog. Последний, вобрав в себя преимущества языков VHDL и Verilog [5], является наиболее популярным и предоставляет программисту такие возможности, как определение собственных типов, разработку сложной иерархии классов (в т. ч. использование наследования) и множество других механизмов [6].

Поскольку инструменты для физической верификации интерфейса CSI существуют [7], далее будем говорить только о функциональной. Самый простой подход к функциональной верификации – направленные тесты. В направленном тесте создаются наборы входных воздействий для проверки всех интересующих состояний модуля и ситуаций системы. Выходные значения сверяют с ожидаемыми. Проблемой этого подхода является плохая масштабируемость [8] – в более сложных устройствах, состоящих из различных модулей, направленный тест не позволяет проверить весь функционал устройства. Для их верификации используется специально подготовленное тестовое окружение (testbench), изображённое на рис. 1.9.

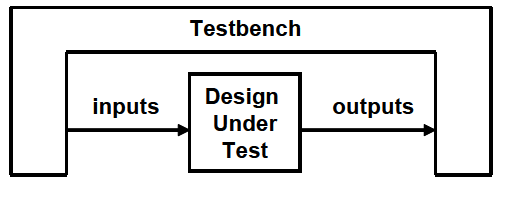


Рисунок 1.9 – Схематичное изображение тестового окружения

Задачей окружения является воздействие на тестируемое устройство (DUT, от англ. Device Under Test) путём эмуляции подключения к другим устройствам (например, отправка транзакций по шине USB, запись в память и др.), и сравнение результатов с ожидаемыми путём доступа к памяти устройства или проверки исходящих сигналов. Используемый в нём код может быть не синтезируемым (таким образом, становится возможным использование классов, массивов, функций и других преимуществ традиционных языков программирования). Кроме того, тестовое окружение должно обеспечивать проверку временных характеристик работы устройства, таких, как частота тактового сигнала, задержки в работе различных интерфейсов и др.

Другой важной целью является увеличение функционального покрытия теста. При верификации сложных схем для полной проверки работы сумматоров и других элементов, имеющих разные задержки в работе в зависимости от входных данных, необходимо покрыть как можно больше возможных значений для исключения всех возможных ошибок. Для этого используется constraint-random тестирование, основанное на том, что случайные входные сигналы ограничиваются с помощью ограничителей (англ. constraints).

# 1.2.2 Обзор методологии UVM

Одна из проблем верификации заключается в том, что для каждого нового устройства необходимо готовить новое окружение, на что уходит большое количество усилий и времени.

Для разработки верификационных IP гораздо удобнее использовать моделирование на уровне транзакций (TLM, Transaction-Level Modelling). Примером транзакции может служить любой объект, несущий какую-либо информацию, которая должна быть передана между устройствами – пакет CSI, пиксель или даже кадр изображения.

Уход от моделирования на уровне сигналов позволяет менее детально прорабатывать взаимодействие между частями тестового окружения, отправляя транзакции целиком.

Наиболее популярной в настоящее время методологией, использующей моделирование на уровне транзакций, является универсальная методология верификации (Universal Verification Methodology, UVM) [9], которая была разработана в 2011 году компаниями Cadence и Mentor Graphics и в 2017 была принята как стандарт IEEE 1800.2-2017.

UVM – это методология функционального верифицирования, стандарт, позволяющий разрабатывать и повторно использовать среды верификации. Аппаратное обеспечение, верифицируемое с помощью UVM, может быть написано на одном из языков описания оборудования – Verilog, SystemVerilog, VHDL или SystemC. UVM ориентирована на работу с симулятором, совместимым, с выбранным языком, но также может быть использована вместе с проверкой на основе утверждений, аппаратным ускорением и эмуляцией.

Согласно методологии UVM, жизненный цикл верификационного окружения состоит из следующих этапов: создание базовых компонентов окружения, разработка процессов генерации тестовых последовательностей, создание тестов для покрытия различных тестовых случаев, поддержка и документация разработанного окружения, при необходимости – перепроектирование окружения для использования с новой версией устройства. Концепция UVM позволяет повторно использовать уже разработанные модули для верификации будущих разработок или легко заменять модули в текущем верификационном окружении. Схема типичной для UVM архитектуры окружения представлена на рисунке 1.10.

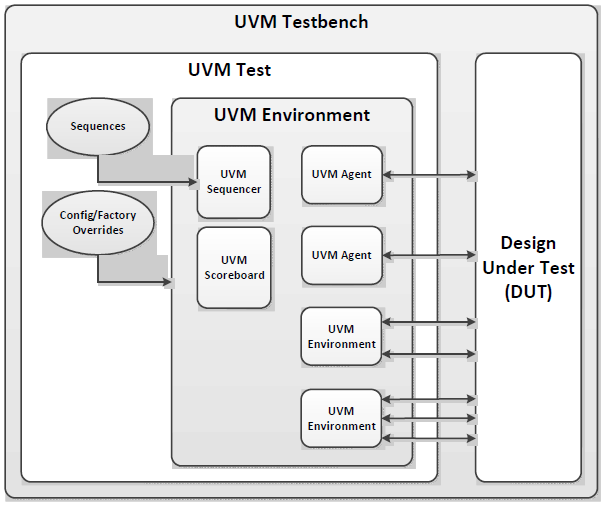


Рисунок 1.10 – Архитектура тестового окружения UVM

В окружении обычно присутствуют следующие элементы [10]:

1) Testbench – верхний модуль, соединяющий тест с проверяемым устройством.

2) DUT (Device Under Test) – тестируемый модуль.

3) Тест (test) – компонент верхнего уровня. Задачей теста является запуск тестовых последовательностей на секвенсорах, настройка элементов тестовой среды через специально установленные методы базы конфигурации. Также возможна подмена экземпляров классов окружения через так называемую «фабрику» UVM при необходимости.

4) Тестовая среда (environment) – компонент, объединяющий верификационные агенты и scoreboard.

5) Класс для сравнения (scoreboard) – компонент, функцией которого является проверка поведения тестируемого устройства путём получения транзакций из разных источников и их сравнения по каким-либо параметрам.

6) Агент (agent) – иерархический компонент, группирующий в себе верификационные компоненты, как правило, относящиеся к определённому интерфейсу DUT. Схема агента приведена на рис. 1.11. В каждом агенте обычно присутствует драйвер и/или монитор, в зависимости от его назначения.



Рисунок 1.11 – Внутреннее устройство агента UVM

7) Драйвер (driver), основываясь на получаемых от секвенсора транзакциях, управляет входными сигналами устройства и задаёт тестовые воздействия, напрямую подключаясь к интерфейсу.

8) Монитор (monitor), получая информацию с выходных сигналов устройства, формирует новые транзакции. Монитор может выполнять некоторую обработку транзакций (например, расчёт контрольной суммы) или делегировать это другим компонентам окружения.

9) Секвенсор располагается внутри агента или среды, в зависимости от его назначения и является базой для запуска тестовых последовательностей. На одном секвенсоре может быть запущена как одна последовательность, так и несколько.

10) Тестовые последовательности (sequence), запускаемые на секвенсорах, генерируют транзакции, определяющие поведение драйверов, агентов, а в отдельных случаях – тестовой среды целиком.

Библиотека UVM представляет собой набор базовых классов, необходимых для создания полноценного верификационного окружения на основе TLM. Диаграмма классов UVM представлена на рис. 1.12.

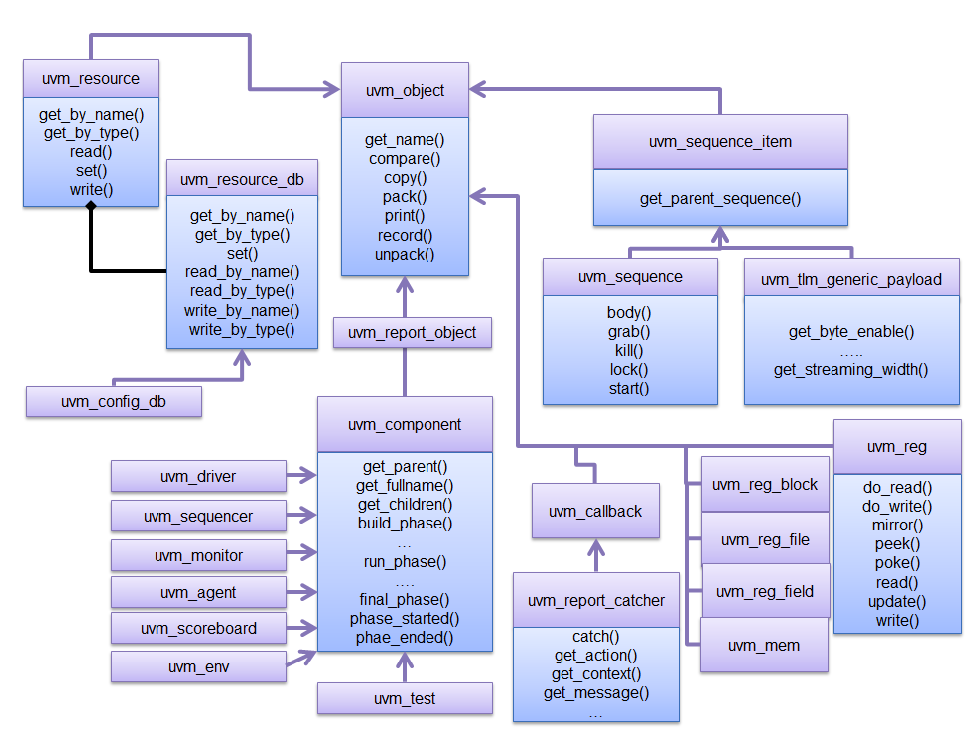


Рисунок 1.12 – Диаграмма классов UVM

В библиотеку входят компоненты (драйвер, монитор и др.), последовательности и транзакции. В каждый базовый класс входят необходимые функции (получение имени, вывод всех полей и т. д.). Существует возможность зарегистрировать отдельные переменные в составе класса как его поля, что позволяет автоматизировать процесс копирования, вывода информации и т. д.

Кроме того, библиотека UVM предоставляет классы и инструменты для упрощения создания и настройки окружения. Отдельно стоит выделить *uvm\_config\_db* – конфигурационный класс окружения. Данный класс позволяет менять зарегистрированные поля компонентов, основываясь на их иерархическом имени [11].

Тем не менее, UVM не полностью решает проблему совместимости различных верификационных IP. В случае, если для проверки отдельных частей DUT используются агенты с разной логикой работы, отличающимися методами настройки и несовместимыми форматами транзакций, при сборке окружения большое количество времени уходит на интеграцию VIP от разных производителей в одну тестовую среду.

Одним из методов решения этой проблемы является объединение компонентов, предназначенных для выполнения схожих задач, в отдельную библиотеку, включающую необходимые агенты, сиквенсеры и прочие элементы среды. В таком случае эта библиотека может быть подключена к тестовому окружению более высокого уровня.

# 1.3 Обзор используемого окружения

Разрабатываемый VIP CSI планируется использовать в составе UVM-окружения «VIVO», включающего в себя множество различных инструментов для верификации видео интерфейсов. Архитектура организована таким образом, чтобы была возможность легко создавать необходимое количество интерфейсов и подключенных к ним агентов (VIP).

Схема окружения привязана на рис. 1.13.

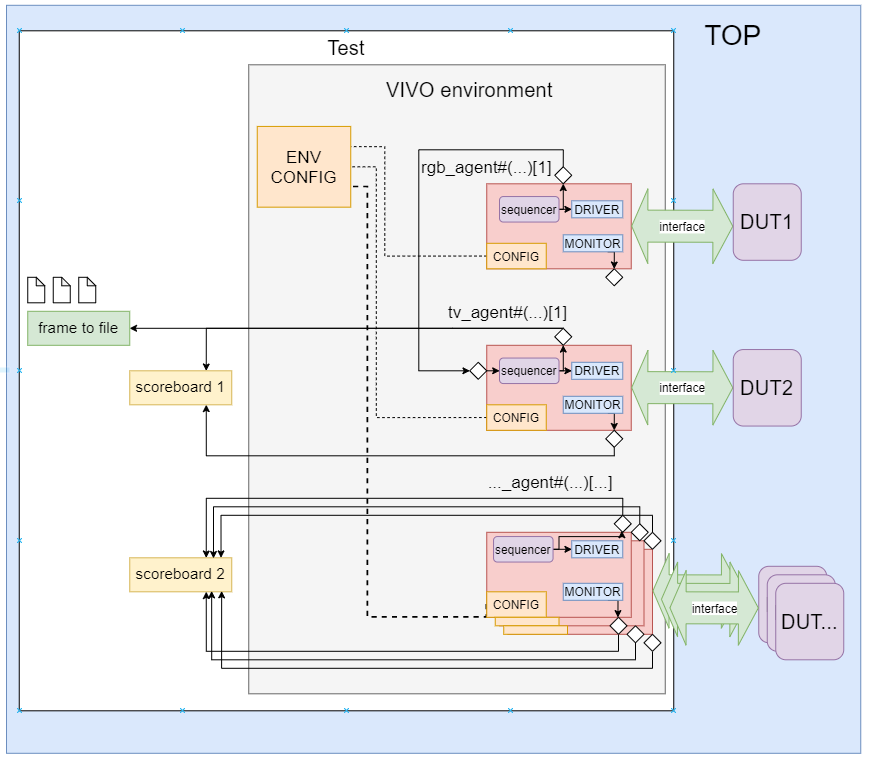


Рисунок 1.13 – Схема окружения VIVO

Основой для всех используемых верификационных IP является базовый агент – генератор изображений *(img\_gen\_agent*). Агент параметризован таким образом, чтобы драйвер и монитор могли быть заменены на любые классы, унаследованные от стандартных драйвера и монитора. Тем не менее, возможно использование любых классов UVM, имеющих аналогичные порты и функции для автоматического подключения к секвенсору или интерфейсу.

Особенностью базового агента является класс конфигурации, который также может быть использован для наследования. Конфигурационный класс содержит всю информацию про отдельный агент, такую, как:

1) Тип агента (определяет используемый VIP);

2) Является ли агент генератором кадров, или получает их с другого агента (используется, когда необходимо подать один кадр на несколько агентов);

3) Активность агента (должен ли агент задавать тестовые воздействия, или же только считывать транзакции с интерфейса);

4) Характеристики видеопотока (горизонтальный и вертикальный размер, а также частота кадров – используется в отдельных VIP);

5) Номер интерфейса для подключения;

6) Любые зарегистрированные в базе UVM целочисленные параметры, характерные для определённого VIP.

В агенте используются два секвенсора – задачей первого является генерация транзакций-кадров, второго – разбиение кадров на транзакции-пиксели для подачи на драйвер. В зависимости от требований к окружению второй секвенсор может не использоваться – в этом случае транзакции-кадры напрямую поступают к драйверу. Схема агента приведена на рисунке 1.14.

Транзакция кадра содержит информацию о номере и размерах кадра, формате цветности, а также массив, содержащий все компоненты изображения, а также флаг последнего кадра. Кроме того, предоставляется функция, позволяющая получить или установить значение заданной компоненты произвольного пикселя кадра. Передача всей информации внутри кадра позволяет при необходимости отключать генерацию пикселей.

Поскольку вся информация содержится в транзакции кадра, транзакция пикселя содержит только указатель на кадр и информацию о своём местоположении. Получение значения компоненты пикселя также ведёт к вызову аналогичной функции кадра.

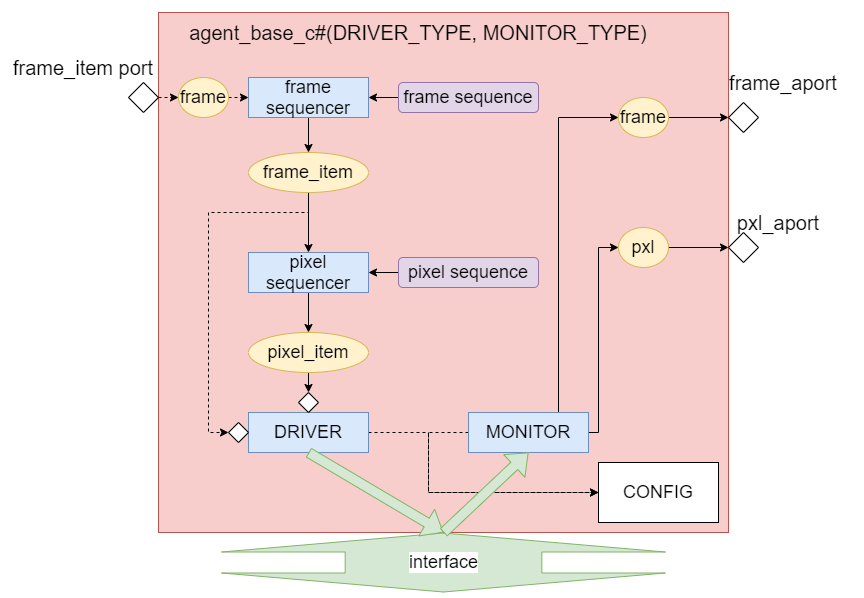


Рисунок 1.14 – Схема параметризованного агента

Для секвенсоров определены две последовательности – одна считывает кадры из файла и преобразует в требуемый формат данных, другая генерирует кадры (и пиксели, при необходимости) случайным образом. Присутствует возможность генерации в соответствии с одним из предопределённых шаблонов.

Все агенты находится внутри среды, где они создаются и настраиваются *(vivo\_env\_base*). Кроме того, среда содержит обобщённый конфигурационный класс, в котором хранятся ссылки на классы конфигурации каждого VIP. Информация о требуемых агентах загружается в начале симуляции обобщённым классом из файла настроек, где параметры каждого должны быть заданы в строго определённом формате.

После создания конфигурационных объектов для агентов всех интерфейсов класс среды использует эту информацию для создания самих VIP.

Пример записи в файле конфигурации:

71 # ***кол-во символов***

1,1 # ***активность и генерация кадров***

1 # ***тип агента***

768,576,25.00 # ***размер и частота кадров***

0,0 # ***номер подключаемого интерфейса и IP***

mode,0 # ***настройки VIP***

interlaced\_input,1

fast\_test,1

Тест соединяет нужные компоненты между собой, а также с классом сравнения (scoreboard), после чего автоматически запускает генераторы кадров и пикселей во всех созданных агентах. Также есть возможность сохранять вывод с мониторов агентов в файлы.

Верхним модулем окружения (testbench) является tb\_top, определяющий все используемые интерфейсы и подключающий к ним драйвера и мониторы верификационных IP в соответствии с их иерархическими именами. Там же могут быть подключены RTL-модули и установлены глобальные параметры окружения, например частоты тактовых сигналов для интерфейсов, момент снятия сброса и т. д.

Важной особенностью используемого окружения является использование макросов для упрощённой генерации кода в классах среды, теста, а также верхнего модуля (топа). Это позволяет упростить изменение вышеперечисленных классов, а также добавление новых, поскольку для любых агентов используются одинаковые операции. С другой стороны, использование макросов накладывает определённые ограничения на используемые верификационные IP.

Например, имя подключаемого пакета (*package*), содержащего файлы VIP, должно совпадать с именем агента в окружении. Кроме того, невозможно использование произвольных имён интерфейсов (поскольку имя содержит номер интерфейса в массиве), а имена драйвера и монитора агентов не должны отличаться между различными VIP.

# 1.4 Обзор существующих верификационных IP интерфейса CSI

На рынке верификационных компонентов уже существуют VIP CSI от производителей Cadence и Synopsys. Оба инструмента поддерживают последние версии спецификации CSI-2, проверки протокола и временных промежутков. В каждом из перечисленных VIP доступны все используемые форматы данных, любое допущенное спецификацией интерфейса количество виртуальных каналов и линий. Поддерживаются все функции, указанные в стандарте CSI-2. Тем не менее, данные верификационные IP имеют ряд недостатков:

Использование версии CSI-2 v3.0, что затрудняет верификацию устройств, выпущенных до 2017 года и не использующих обновлённый стандарт. Устройства, для которых планируется использовать разрабатываемый верификационный IP, используют устаревшие версии протокола;

– Высокая стоимость лицензии на использование;

– Закрытый исходный код, что исключает детальную настройку имён транзакций и компонентов, а также отладочных сообщений в соответствии с требованиями окружения.

Из перечисленных недостатков следует, что интегрирование одного из существующих IP для данного интерфейса приведёт к возникновению множества проблем. Кроме того, использование множества VIP от разных производителей неизбежно приводит к расхождениям в форматах данных и именах, из-за чего приходится реализовывать функции и/или классы, обеспечивающие совместимость окружения с конкретным VIP.

Это делает использование одинаковых механизмов создания и настройки агентов невозможным. Поскольку основной особенностью библиотеки, в составе которой планируется использовать верификационный IP CSI, является универсальность механизма конфигурации, было принято решение о разработке нового VIP.

# 1.5 Выводы главы 1

В данной главе был произведён обзор интерфейса CSI и описаны особенности его функционирования. Были рассмотрены различные уровни работы интерфейса, в том числе – физический уровень D-PHY. Рассмотрен порядок функционирования интерфейса, отправки пакетов и разделения информации по линиям.

Также было рассмотрено понятие верификации и произведена классификация её видов. Поскольку функциональная верификация позволяет обнаружить все алгоритмические ошибки модуля ещё на этапе создания его модели на языке описания аппаратуры, этот вид верификации был рассмотрен более детально. Описаны основные подходы к функциональной верификации и её проблемы. Была подробнее рассмотрена методология UVM, принятая в стандарте IEEE 1800.2-2017. Описано типичное окружение, построенное в соответствии с данной методологией, и подробно рассмотрена его структура. Объяснено назначение компонентов, из которых может быть построено окружение, их основные функции. Приведена диаграмма классов библиотеки UVM для языка SystemVerilog.

Поскольку верификационный IP CSI будет разрабатываться для использования в составе окружения «VIVO», рассмотрена структура данного окружения. Подробно описаны особенности и ограничения различных компонентов, которые будут влиять на разработку VIP. Также рассмотрены существующие VIP для интерфейса CSI и указаны их возможности и недостатки.

Верификационный IP для интерфейса CSI будет разрабатываться на языке SystemVerilog с использованием библиотеки UVM 1.2. В качестве базы и окружения для разработки агента будут использоваться классы библиотеки «VIVO». Для работы с физическим уровнем интерфейса будет использоваться IP D-PHY, разработанный компанией Lattice Semiconductors.

Тестирование VIP будет включать в себя проверку различных возможностей интерфейса, описанных в стандарте, а также подключение к IP CSI. Моделирование будет производиться в программе ModelSim Lattice Edition.

# Глава 2 Разработка верификационного IP

В данной главе рассматривается процесс разработки VIP для интерфейса CSI, описывается структура результирующего компонента, а также алгоритмы работы его составляющих.

# 2.1 Определение структуры верификационного IP

# 2.1.1 Соединение VIP CSI с компонентами библиотеки «VIVO»

Как было показано на рисунке 1, интерфейс CSI разделён на несколько слоёв, работающих независимо друг от друга. Уровнем приложения назовём генерацию и обработку кадров на агенте VIVO. Следующие 2 уровня (Pixel-To-Byte и Low-Level Protocol) могут быть совмещены в один: поскольку существует возможность получить кадр целиком, а не разделять его на пиксели, VIP может извлекать строку целиком и определённым образом, зависящим от используемого формата данных, упаковывать её информацию в поле данных пакета. Это позволяет избавиться от описанной в стандарте CSI-2 упаковки нескольких пикселей в одно n-байтное слово с последующим соединением этих слов в пакет, передающий информацию одной линии. Остальные уровни (Lane Management и PHY) остаются в соответствии со спецификацией. В соответствии с методологией UVM оптимальным решением будет создание внутри верификационного IP нескольких модулей, отвечающих за различные преобразования данных.

Это идёт вразрез с основной идеей целевой библиотеки VIVO, где структура агента предопределена и не меняется, а единственными отличающимися компонентами между разными верификационными IP являются монитор и драйвер (рис. 1.14). Поэтому было принято решение выделить данный VIP в отдельный агент, который будет подключаться в качестве дополнения к базовой среде.

Такой метод имеет следующие преимущества:

– Возможность свободно определять структуру и иерархию компонентов агента CSI.

– Использование отдельного механизма конфигурации.

– Подключение агента к нескольким генераторам изображения, что будет более подробно рассмотрено далее.

Из недостатков можно выделить необходимость отдельно подключать и настраивать новый агент, а не использовать макрос, как это сделано для большинства используемых VIP.

# 2.1.2 Причины включения уровня LLP в агент VIVO

Одной из важных особенностей интерфейса CSI является поддержка виртуальных каналов. С использованием переключения между каналами можно передавать через один физический интерфейс информацию от нескольких источников. Пример одновременной передачи нескольких кадров через различные виртуальные каналы приведён на рисунке 2.1.

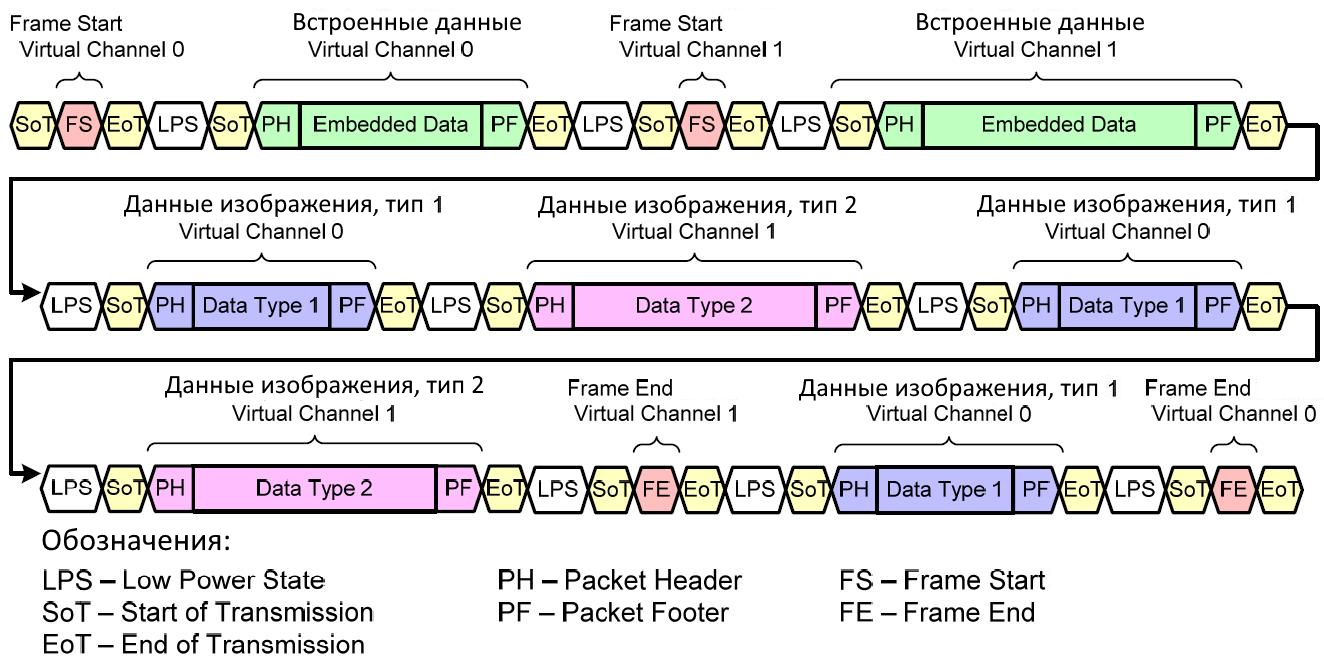


Рисунок 2.1 – Перемешивание виртуальных каналов

Для проверки правильности функционирования данного вида передачи на тестируемом устройстве разрабатываемый VIP должен удовлетворять следующим требованиям:

– Возможность подключения нескольких источников видеоданных (генераторов кадров), работающих в различающихся режимах;

– Наличие переменного количества портов для поступления транзакций;

– Возможность использования нескольких алгоритмов для перемешивания пакетов от разных виртуальных каналов.

Поставленную задачу можно решить путём создания параметризованного секвенсора, имеющего несколько последовательностей. Нужный алгоритм смешивания будет выбираться на уровне теста и реализует требуемый порядок опроса источников и их приоритет. В таком случае реализация уровней Pixel-To-Byte и Low-Level Protocol будет выглядеть как несколько секвенсоров, обрабатывающих транзакции-кадры или транзакции-пиксели.

Приведённая на рисунке 2.2 структура позволяет включить все слои интерфейса в один агент. Тем не менее, предложенная реализация вызовет ряд проблем при интеграции агента в окружение VIVO. Например, наличие нескольких генераторов пакетов внутри одного агента усложняет его конфигурацию. В результате, помимо отдельного формата конфигурации для VIP понадобится также создавать записи для каждого генератора пакетов, входящего в его состав.

Вторая проблема заключается в отсутствии обратной связи с генератором изображений, входящим в библиотеку VIVO. Поскольку первая реализация предполагает подключение к агенту через *analysis\_port*, это приводит к необходимости создания очереди (FIFO) для поступающих кадров, а также к невозможности бесконечной генерации кадров, поскольку результатом будет переполнение данной очереди.

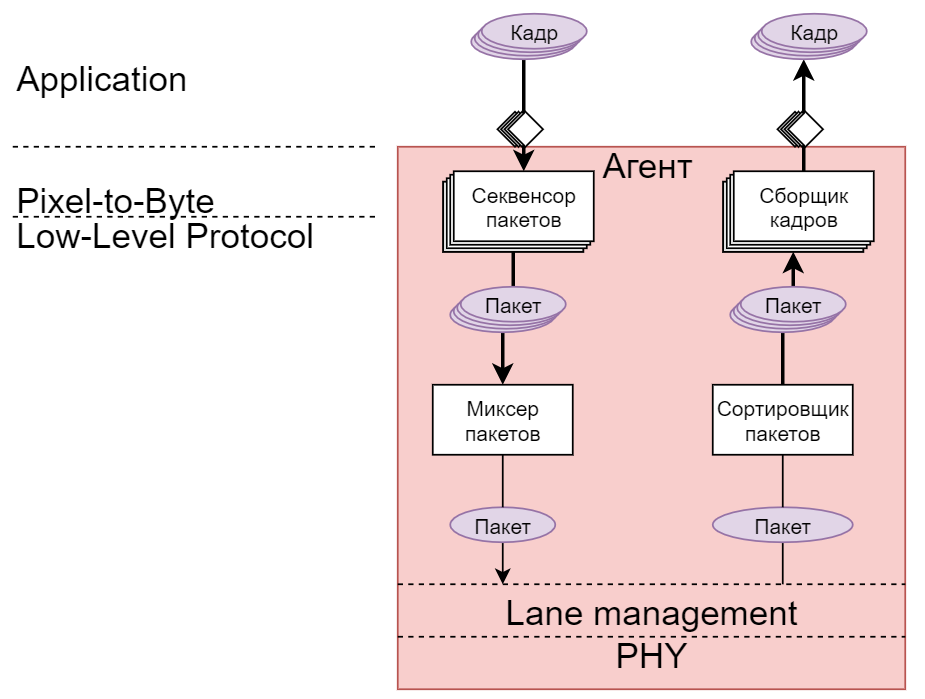


Рисунок 2.2 – Первая реализация перемешивания виртуальных каналов

Решением обеих проблем может стать включение генераторов пакетов в состав агента VIVO. Поскольку на место драйвера может быть подставлен любой класс, имеющий аналогичные порты и методы, каждый секвенсор, генерирующий пакеты на основе полученных кадров, может быть перенесён внутрь агента. В данном агенте нет необходимости в генерации пикселей, поскольку пакеты всегда создаются из строк изображения целиком. Аналогично, монитор агента-приёмника будет заменён на *uvm\_subscriber*, получающий через порт считанные монитором пакеты, принадлежащие к определённому виртуальному каналу, и восстанавливающий из них кадры и пиксели, если это необходимо. Кроме того, часть конфигурации, относящаяся к слоям Pixel-To-Byte и Low-Level-Protocol, которая может различаться для каждого источника изображений, также будет включена в конфигурационные классы агентов-генераторов. Данный метод подключения проиллюстрирован на рисунке 2.3.

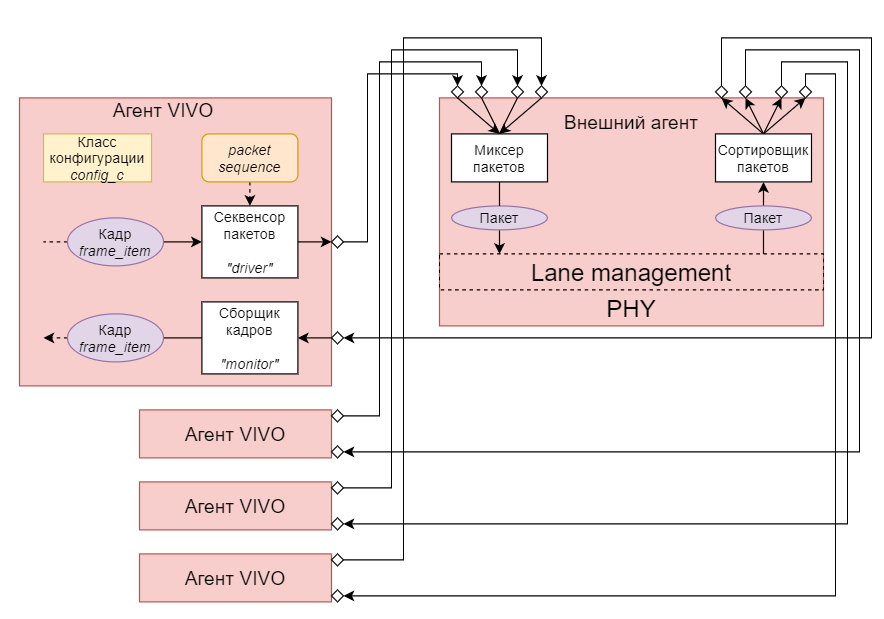


Рисунок 2.3 – Подключение VIP с выделенным слоем LLP

# 2.1.3 Организация внутренней структуры внешнего агента

Поскольку задачу создания пакетов CSI на основе кадров выполняет секвенсор внутри агента VIVO, на вход агент получает несколько потоков пакетов, поступающих по запросу. Структура транзакции-пакета должна соответствовать спецификации CSI-2 и включать в себя все поля, такие как номер виртуального канала, тип пакета, длина, ECC, а также поле данных и CRC для длинных пакетов.

Таким образом, внешний агент должен выполнять следующие задачи:

– Смешивание пакетов, поступающих с нескольких портов, в один поток данных;

– Распределение информации пакетов по нескольким линиям (уровень Lane Management);

– Управление D-PHY: изменение состояния линий, передача данных.

Способ реализации первой задачи был описан в разделе 2.1.2. Выходом секвенсора-миксера является поток перемешанных транзакций-пакетов, по формату не отличающихся от входящих.

Следующий уровень отвечает за укладку информации пакета в единый поток байтов, а также распределение этого потока по заданному количеству линий. Результирующая транзакция будет иметь вид двумерного массива размерностью LxN, где L – количество доступных линий, N – длина транзакции. Уровень PHY реализуется полностью драйвером.

Разработанная структурная схема агента приведена на рисунке 2.4.

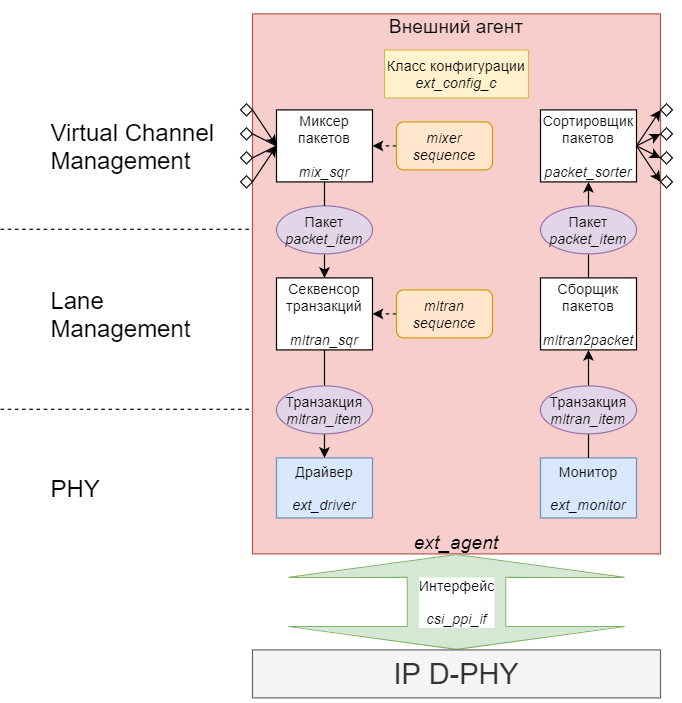


Рисунок 2.4 – Структурная схема внешнего агента CSI

Работа монитора может быть описана аналогичным образом: уровень PHY получает транзакции по интерфейсу, следующий уровень восстанавливает пакеты из потока байтов, после чего пакеты отправляются на нужный порт агента. Монитор в агенте VIVO создаёт транзакции-кадры на основе полученных пакетов.

Во внешний агент также будет входить конфигурационный класс, позволяющий управлять работой монитора и драйвера.

# 2.2 Разработка алгоритмов работы компонентов VIP

В данном разделе описаны алгоритмы работы различных частей верификационного IP CSI, в том числе монитора и драйвера, находящегося внутри агента VIVO

# 2.2.1 Разработка алгоритмов работы секвенсоров и драйвера

Как было показано на рисунке 2.4, перед внешним драйвером находятся ещё 3 ступени, выполняющие различные преобразования над информацией входящих кадров. Рассмотрим их функционирование более подробно.

Первой ступенью является внутренний «драйвер»-секвенсор, входящий в состав агента VIVO. Данный секвенсор загружает кадр, после чего по запросу от внешнего агента отправляет транзакции-пакеты в соответствии с настройками. На данный момент используется следующие типы пакетов:

– Frame Start/End отправляются в начале или конце кадра;

– Line Start/End отправляются на каждой линии (опционально);

– Data – RAW, RGB, YUV.

При включении режима чересстрочной отправки кадров, в соответствии со спецификацией CSI, поля отправляются последовательно как отдельные кадры.

Важно отметить, что нумерация кадров в пакетах Frame Start/End может отличаться от нумерации кадров в приходящих транзакциях: протокол CSI устанавливает, что номер кадра должен либо всегда равняться 0, либо увеличиваться на 1 с каждым кадром, периодически сбрасываясь до 1 (например, 1,2,3,4,1,2…). Алгоритм работы секвенсора пакетов приведён на рисунке 2.5.

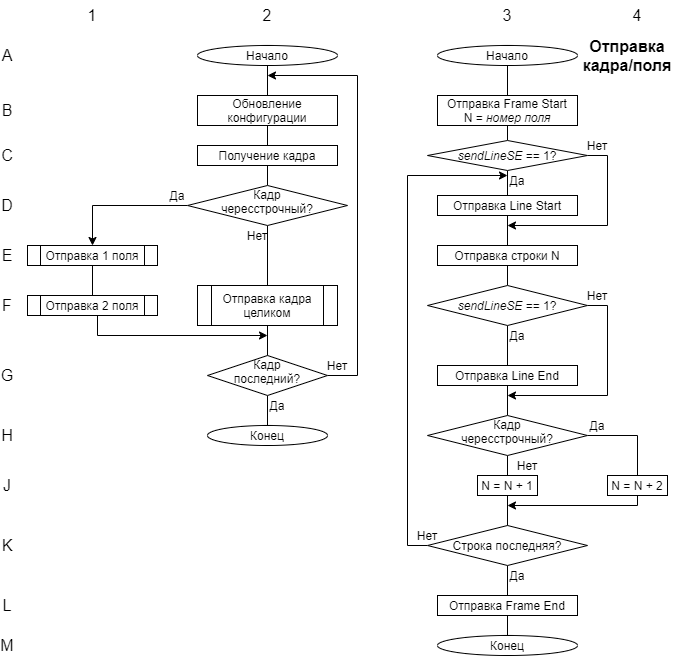


Рисунок 2.5 – Алгоритм работы секвенсора пакетов

Сгенерированные пакеты отправляются на смешивающий секвенсор. На данный момент секвенсор по очереди перебирает все источники, не учитывая размер пакетов, тип данных и т. д. В зависимости от теста, на этом секвенсоре может быть запущена последовательность, удовлетворяющая установленным для окружения требованиям.

Перемешанные пакеты по запросу поступают на секвенсор транзакций. Последовательность, запущенная на этом секвенсоре, превращает данные пакета в набор байтов, распределённых по нескольким линиям. Кроме того, происходит дополнение линий до одинаковой длины инвертированным последним битом. Последний бит данных сохраняется для каждой линии – он будет использоваться внутри внешнего драйвера. Алгоритм работы последовательности, генерирующей транзакции для драйвера, приведён на рисунке 2.6.

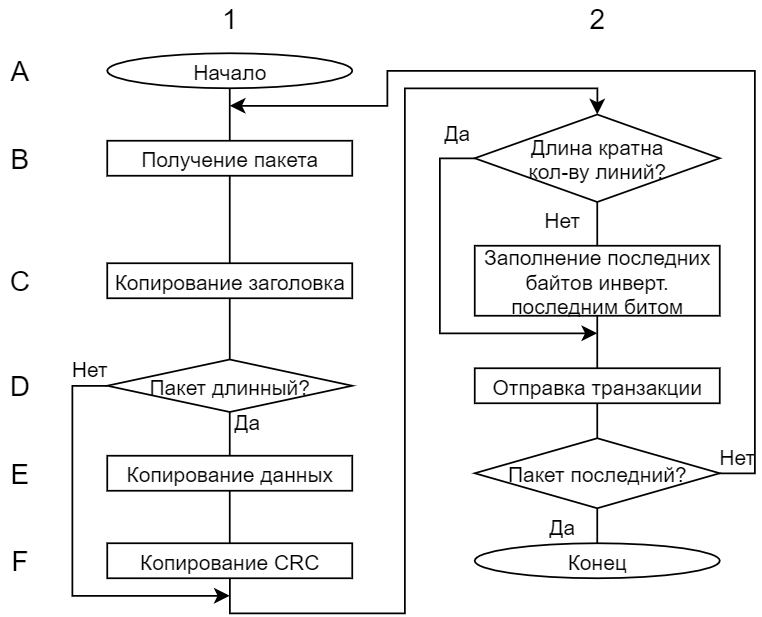


Рисунок 2.6 – Алгоритм генерации транзакций на основе пакетов

Основной задачей драйвера внешнего агента является управление IP Lattice, подключённым к двунаправленному интерфейсу, в соответствии со спецификацией MIPI D-PHY. Поскольку IP-Transmitter реализует только генерацию тактового сигнала (PLL) и управляет интерфейсом на основе управляющих сигналов, у драйвера есть возможность самостоятельно задавать все временные промежутки и контролировать переход интерфейса между различными режимами.

Для реализации всех возможностей CSI-2 v1 драйвер должен иметь возможность выполнять следующие функции:

– Переключение линии CLK между Low-Power и High-Speed;

– Перевод линий DATA в режим High-Speed с выводом синхронизирующей последовательности;

– Вывод данных транзакции по тактовому сигналу IP D-PHY;

– Перевод линий DATA в режим Low-Power с предварительной установкой инвертированного последнего бита (trail) данных для индикации конца транзакции;

– Перевод линий CLK в режим ULPS (Ultra Low-Power State) и обратно;

– Передача команд по линиям DATA в режиме Escape Mode, перевод в режим ULPS и обратно;

Было рассмотрено 2 варианта реализации данного компонента – с использованием конечного автомата, состояния которого определены в спецификации D-PHY, или через функции, реализующие отдельные процедуры целиком. Несмотря на большую гибкость, реализация с помощью автомата имеет ряд недостатков [12]. Поскольку часть возможностей D-PHY (таких, как передача данных в обратном направлении, а также дополнительные функции Escape Mode) не используется в протоколе CSI-2, а переходы между используемыми состояниями зависят только от временных промежутков и ожидания команд от управляющего устройства (что в данном случае не используется, так как драйвер сам запрашивает пакеты у вышестоящих секвенсоров), было принято решение отказаться от использования конечного автомата. Таким образом, поведение драйвера можно описать алгоритмом, изображённым на рисунке 2.7.

На используемом IP D-PHY Transmitter присутствует только один вход, переключающий одновременно все линии между режимами LP/HS. Поэтому нет необходимости разделять управление состоянием линий в отдельные процессы.

Каждая транзакция начинается с перевода линий в режим High-Speed процедурой SoT (Start Of Transaction). После завершения передачи данных на линии выводится инвертированный последний бит (Trail), после чего процедурой EoT (End Of Transaction) линии данных переводятся в Low-Power.

Драйвер поддерживает оба режима работы линии CLK – Continuous (постоянный HS) и Non-Continuous (с переводом в LP между передачей пакетов).

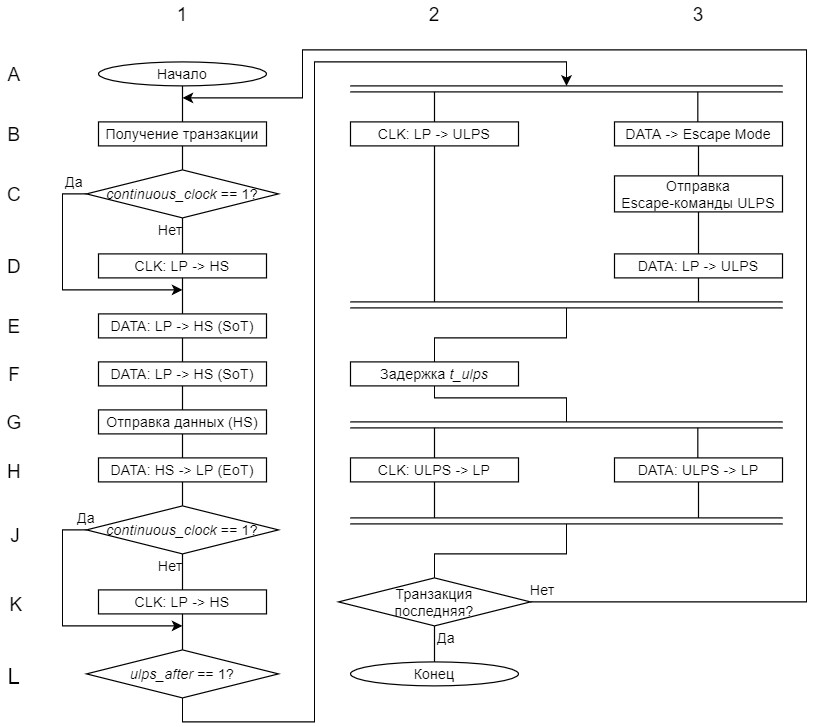


Рисунок 2.7 – Алгоритм работы внешнего драйвера CSI

Для проверки правильности перехода тестируемого устройства в ULPS есть возможность перехода в данное состояние по флагу в транзакции.

# 2.2.2 Разработка алгоритмов работы компонентов монитора

Подобно драйверу, монитор разделяется на несколько компонентов. Первый из них, подобно драйверу, подключён к IP D-PHY Receiver и осуществляет управление его режимами, а также считывает выводимые данные.

Для этого был реализован упрощённый конечный автомат, описанный в спецификации MIPI D-PHY. Это позволяет эмулировать поведение реального приёмника и обнаруживать любые нештатные ситуации.

Состояния, соответствующие неиспользуемым в CSI функциям, были удалены. Кроме того, заменены состояния и переходы, которые невозможно реализовать с используемым IP. Например, состояние RX-HS-SYNC, ожидающее получения синхронизирующей последовательности (00011101), удалено, поскольку последовательность используется для синхронизации IP и не выводится на интерфейс монитора.

Поскольку IP-Receiver также имеет только один вход для переключения режимов, а в режиме Low-Power выводит состояние только одной линии (предполагается, что остальные находятся в таком же), в мониторе разделение процессов по линиям также отсутствует.

При получении любых переходов, не указанных в диаграмме, выводится сообщение об ошибке и монитор ожидает возврата в исходное состояние. Исключениями являются состояния RX\_HS\_Settle (переходы игнорируются для исключения переходных эффектов) и RX\_HS\_Run (где возможен также переход между HS01-HS10).

Диаграммы переходов конечных автоматов приведены на рис. 2.8 (линия CLK) и 2.9 (линии DATA).

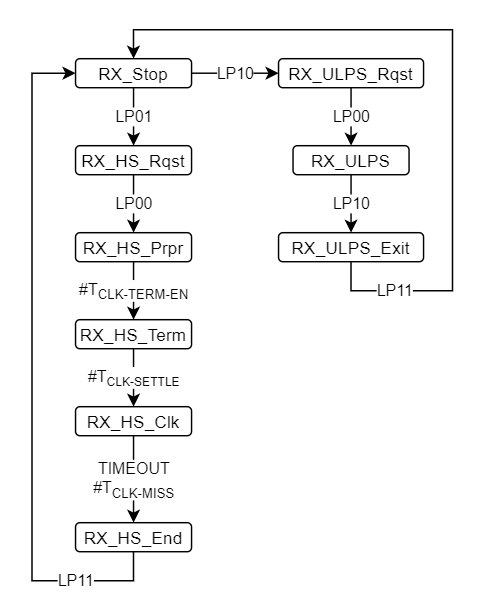


Рисунок 2.8 – Диаграмма переходов автомата CLK

Расшифровка обозначений на рис. 2.8:

– LPxx, HSxx – ожидание перевода линий в определённое состояние;

– #T... – ожидание истечения заданного временного интервала (определён в спецификации D-PHY);

– TIMEOUT – отсутствие переходов на линии CLK в течение заданного времени.

Начальным состоянием для линии CLK является HS-Stop. По переходу линии в LP01 начинается процедура перехода в режим High-Speed, заканчивающаяся состоянием RX\_HS\_Clk. При отсутствии изменений на линии в течение какого-то времени монитор переходит в состояние RX\_HS\_End, где ожидает окончательного перехода в Low-Power. Переход в LP10 из начального состояния является командой на выход в Ultra-Low Power State.

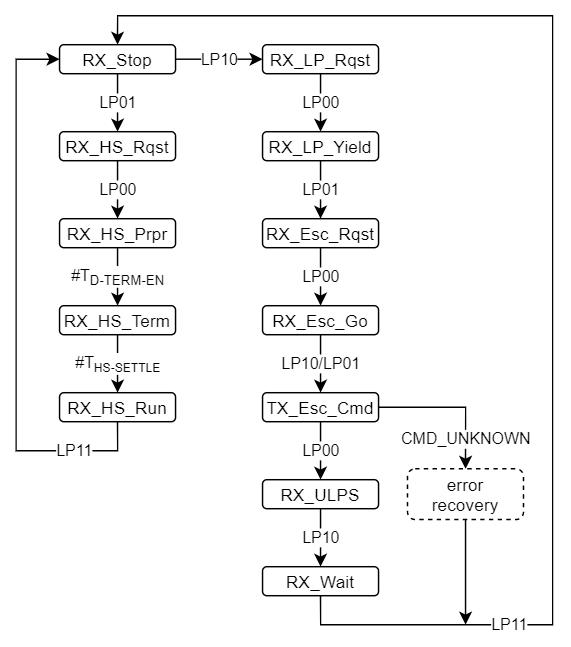


Рисунок 2.9 – Диаграмма переходов автомата DATA

Расшифровка обозначений на рис. 2.9:

– CMD\_ULPS – получение команды на переход в ULPS в режиме Escape Mode;

– CMD\_UNKNOWN – получение неизвестной команды в режиме Escape Mode.

Аналогично, начальным состоянием для линии DATA является LP11, или RX-Stop. На момент получения LP01 линия CLK должна быть в состоянии RS\_HS\_Clk (в противном случае выводится сообщение об ошибке). Состояние RX\_HS\_Run включает в себя всё взаимодействие в режиме High-Speed. В отличие от CLK, на линиях DATA сигналом для выхода из режима High-Speed является переход в состояние LP11.

По последовательности LP11-10-00-01-00 монитор переходит в режим Escape Mode. Единственной поддерживаемой командой является переход в ULPS; в противном случае выводится сообщение об ошибке.

Важно отметить, что за данные, полученные за определённое время до перехода в RX\_Stop из режима High-Speed, игнорируются, во избежание переходных эффектов. Вся информация собирается в объект-транзакцию и отправляется далее.

Следующим шагом является восстановление пакетов из транзакций. Алгоритм сборки одного пакета показан на рис. 2.10. Последний бит транзакции отбрасывается. Перед созданием пакета выполняется ряд проверок, позволяющих определить проблемы с транзакцией (например, сильное различие длин «полезной» информации на разных линиях, окончание полезных данных не на границе байт и т. д.).

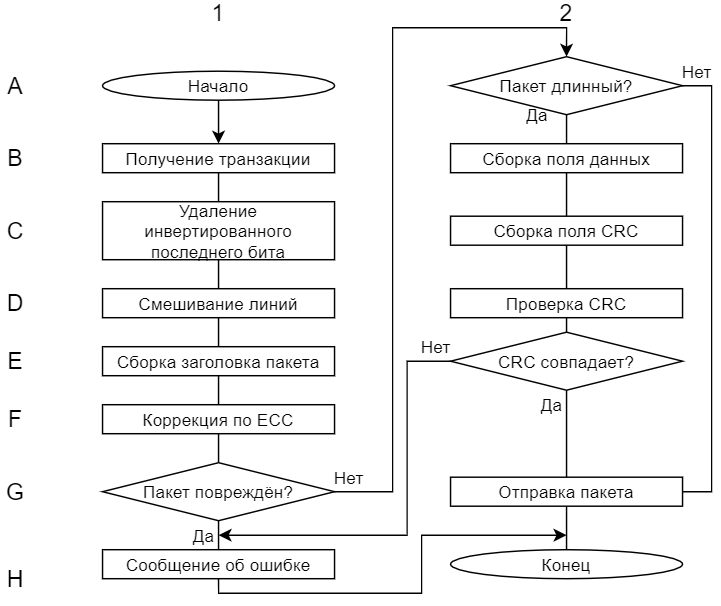


Рисунок 2.10 – Алгоритм работы монитора-сборщика пакетов

После заполнения полей пакета выполняется проверка его целостности. Первым шагом является коррекция заголовка с помощью кода ECC. В случае, если пакет длинный, проверяется контрольная сумма поля данных (CRC). Пакеты с неисправимыми заголовками или несовпадающей контрольной суммой отбрасываются.

После сборки все пакеты направляются на монитор-сортировщик. Сортировщик распределяет пакеты в зависимости от номера виртуального канала и отправляет их на подключённые агенты VIVO.

Монитор, находящийся внутри агента VIVO, обрабатывает полученные пакеты. На данном уровне проверяется порядок пакетов (например, недопустимо получение пакета с данными до пакета Frame Start) и проверка последовательности и содержания коротких пакетов (начало/конец кадров и строк). Алгоритм работы внутреннего монитора приведён на рис. 2.11.

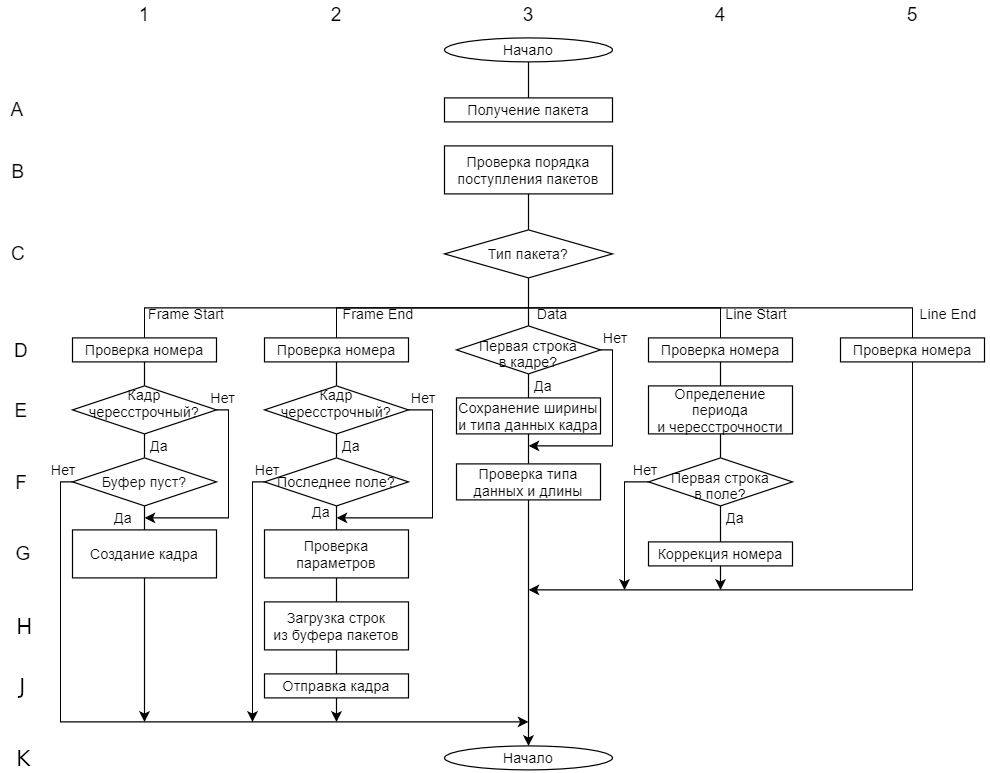


Рисунок 2.11 – Алгоритм работы внутреннего монитора

Пакеты данных сохраняются в буфере и отправляются при получении пакета Frame End. Исключением являются кадры, в которых номера строк идут с определённым периодом (>1). Такое поведение означает, что используется чересстрочная развёртка. В данном случае несколько кадров, переданных через интерфейс, собираются в целый кадр перед отправкой.

# 2.2.3 Разработка монитора интерфейса D-PHY

Поскольку монитор работает с интерфейсом посредством взаимодействия с IP, отсутствует возможность проверки временных интервалов переключения сигнала. Поэтому было принято решение разработать отдельный компонент для контроля за последовательностью и временными промежутками переключений состояния линий интерфейса, так называемый «*protocol checker*».

Был рассмотрен вариант реализации с помощью подмножества языка SVA (System Verilog Assertions). Но, поскольку в переключениях интерфейса используется логика конечного автомата, реализация проверки данным методом подмножестве не только усложнила бы отладку компонента, но и уменьшила бы гибкость его функционирования. Например, многие временные параметры должны быть масштабируемыми в зависимости от заданной полосы пропускания интерфейса. Кроме того, реализация в виде компонента позволяет встроить в него отдельный класс конфигурации, позволяющий включать и отключать некоторые более строгие проверки или менять количество используемых линий.

Для того, чтобы правильно отслеживать поведение драйвера сигнала, в чекере используется конечный автомат, соответствующий D-PHY передатчика. Но, в отличие от автомата, приведённого в спецификации CSI, где большинство переходов происходят по команде от вышестоящего устройства или по истечению заданного временного интервала, в данном случае для определения переходов используется изменение состояния линии.

Например, монитор D-PHY переходит в состояние TX-Stop (LP11) сразу после того, как передатчик также совершил переход в это состояние и переход отобразился на выходном сигнале. Таким образом можно как отслеживать невозможные переходы, так и проверять временные промежутки между ними. Диаграммы, отображающие направления и условия переходов конечного автомата монитора D-PHY для линий CLK и DATA приведён на рисунке 2.12. Обозначения аналогичны обозначениям автомата монитора, SYNC\_SEQUENCE – успешное получение синхронизирующей последовательности.

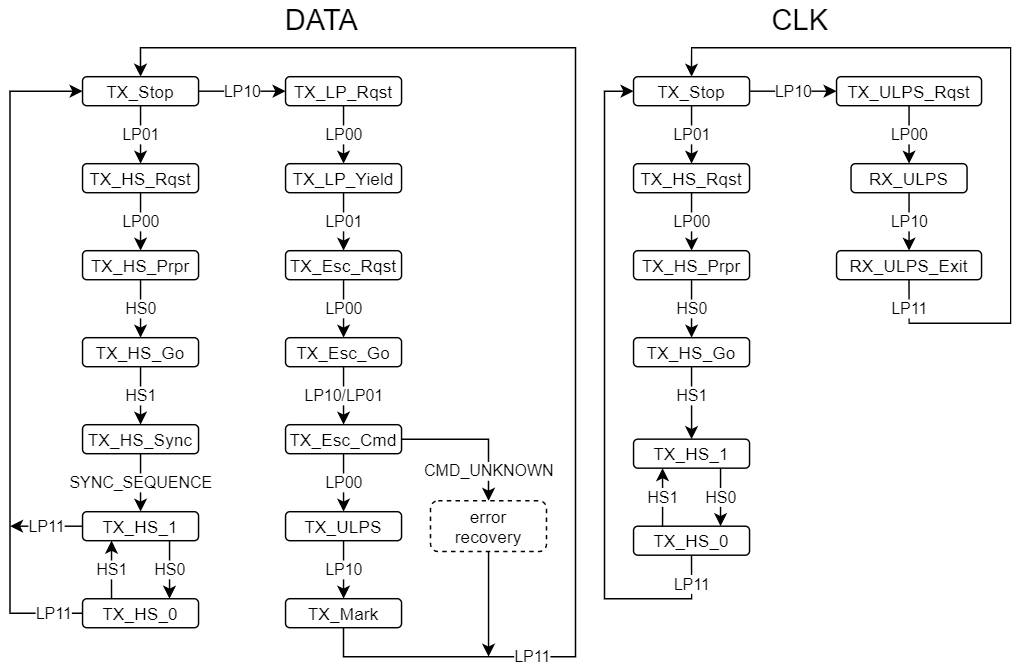


Рисунок 2.12 – Диаграммы переходов автоматов монитора D-PHY

# 2.3 Разработка механизма конфигурации VIP

Для более гибкого управления как параметрами выходного сигнала драйвера и настройки монитора необходима возможность изменения отдельных параметров и флагов агента под конкретный тест.

Наиболее простым методом является добавление конфигурационного объекта. Это позволяет вынести все изменяемые параметры в отдельный класс, таким образом отделив их от переменных, используемых компонентами VIP, а также даёт возможность задать конфигурацию агента путём передачи объекта целиком (вместо установки отдельных полей в config\_db).

Поскольку сам агент разделён на 2 части (монитор и драйвер в составе агента VIVO + внешний агент для взаимодействия с интерфейсом), конфигурация тоже разделяется между конфиг. объектом агента VIVO и агентом интерфейса.

Конфигурационный объект агента VIVO управляет генерацией пакетов на основе полученных кадров, а также параметрами проверки пакетов, получаемых монитором. Также из этого объекта секвенсор кадров, находящийся внутри агента, получает информацию о размере, чересстрочности и формате данных каждого кадра. Таким образом, в конфигурационный объект агента VIVO должны входить следующие поля:

Для драйвера:

– Размер кадра;

– Формат данных в соответствии со спецификацией CSI (численное значение поля равно полю *Data Type* пакета). На основе формата при получении кадра и при генерации пакетов может быть получена информация о запрашиваемой цветовой схеме и размерности полей кадра;

– Период нумерации кадров для драйвера (кадры нумеруются, начиная от 1 до n, после чего следующий кадр идёт снова с номером 1);

– Включение / отключение отправки пакетов Line Start / Line End;

– Включение чересстрочной отправки кадров.

Для монитора:

– Включение строгой проверки наличия пакетов Line Start / Line End (при отключённой опции наличие пакетов в начале и конце каждой строки не проверяется, поскольку их отправка является опциональной);

– Включение строгой проверки порядка поступления полей (1,2,3,4). Включение данной опции позволяет более точно проверять соответствие номеров первых и последних строк каждого поля, но запрещает использование обратного порядка полей (используется, например, в NTSC).

Другой конфигурационный объект, размещённый внутри внешнего агента, влияет на более низкие уровни (Lane Management и PHY) и управляет разбиением пакетов на транзакции по линиям (и сборку пакетов из транзакций), отправкой и получением транзакций по интерфейсу D-PHY. Его атрибуты:

– Временные параметры функционирования драйвера и монитора, в т. ч. полоса пропускания интерфейса.

Для драйвера:

– Количество используемых виртуальных каналов. Несмотря на то, что максимальное кол-во каналов (портов для подключения) задаётся параметром при создании агента, данная опция позволяет опрашивать только часть портов при получении данных секвенсором-миксером;

– Количество используемых линий интерфейса. Аналогично – может быть меньше максимального значения;

– Переключение между постоянным и прерывистым поведением линии CLK (переводится ли тактовый сигнал в режим Low-Power между отправкой пакетов);

– Включение перевода линий данных в ULPS в конце каждого кадра.

– Включение перевода линии CLK в ULPS одновременно с линиями данных.

Для монитора:

– Включение вывода ошибок при получении неизвестной команды в Escape Mode.

Аналогичный объект используется в мониторе интерфейса D-PHY. Его конфигурация позволяет менять количество проверяемых линий данных, а также включать строгую проверку периода тактового сигнала в режиме High-Speed (при включении период сигнала может отличаться не более чем на 15%).

# 2.4 Составление плана тестирования VIP

Для проверки различных функций верификационного IP составлен план проведения тестирования. Данный план приведён в таблице 2.1.

Таблица 2.1. План проведения тестирования VIP

|  |  |  |
| --- | --- | --- |
| **Проверяемая функция** | **Тест** | **Ожидаемый результат** |
| Взаимодействие драйвера с IP D-PHY | Вывод кадра через драйвер (RAW8, 1 линия) | Появление данных на временной диаграмме |
| Взаимодействие монитора с IP D-PHY | Подключение монитора к драйверу через IP TX-RX | Получение кадров, успешное сравнение |
| Работа с несколькими линиями | Работа агентов с IP, использующим 4 линии | Получение кадров, успешное сравнение |
| Работа CLK в разных режимах | Переключение CLK в режим non-continuous | Получение кадров, успешное сравнение |
| Функционирование Escape Mode и ULPS | Перевод линий CLK и DATA в ULPS | Продолжение работы после выхода из ULPS |
| Работа с несколькими виртуальными каналами | Подключение двух источников/приёмников данных разного типа | Получение кадров (на каждом приёмнике), успешные сравнения |
| Отправка и обработка чересстрочных кадров | Отправка чересстрочных кадров | Получение полей, успешное сравнение |
| Совместимость сигнала драйвера с IP | Подключение к IP CSI с парсером пакетов | Получение кадров, успешное сравнение |

# 2.5 Выводы главы 2

В данной главе была описана разработка верификационного IP CSI.

Была разработана внутренняя архитектура агента. VIP разделён на 2 части - драйвер и монитор в составе агента библиотеки VIVO, а также внешний агент, взаимодействующий с интерфейсом CSI.

Внутренние драйвер и монитор обеспечивают получение пакетов CSI из кадров изображения и обратное преобразование пакетов в кадры. Внешний агент состоит из 6 компонентов, разделённых на 3 уровня. Каждый из уровней реализует какое-либо преобразование данных - смешивание потоков виртуальных каналов в общий поток (распределение выходного потока по виртуальным каналам), создание транзакций на основе пакетов (восстановление пакетов из транзакций) и, наконец, вывод транзакций на интерфейс через IP PHY в соответствии со спецификацией D-PHY (получение транзакций с интерфейса).

Кроме того, был разработан монитор («*protocol checker*»), обеспечивающий проверку временных параметров интерфейса D-PHY.

Для компонентов VIP были разработаны механизмы конфигурации. Все настройки хранятся в конфигурационных объектах, находящихся внутри агентов или монитора интерфейса D-PHY.

Для комплексной проверки правильности функционирования VIP был разработан план проведения тестирования.

# Глава 3 Реализация и тестирование

В данной главе описывается процесс реализации компонентов VIP на языке SystemVerilog с использованием библиотеки UVM, процесс интеграции VIP в библиотеку VIVO, а также результаты тестирования разработанного агента.

# 3.1 Реализация компонентов VIP на языке System Verilog

В данном разделе рассматривается реализация различных частей VIP на языке SystemVerilog. Приведены определения классов транзакций, последовательностей и компонентов. Для сокращения места, занимаемого рисунками, ключевые слова “endfunction” / “endtask” удалены из определений классов.

# 3.1.1 Реализация классов транзакций

В разработанном VIP используется 2 типа TLM-транзакций – пакеты CSI и транзакции, распределённые по нескольким линиям данных.

Транзакция для интерфейса (обозначим её как mltran – “multi-lane transaction”) приведена на рисунке 3.1. Помимо поля данных и его длины транзакция содержит флаги *is\_last* и *ulps\_after* (когда этот флаг установлен, после передачи транзакции драйвер переводит линию в состояние ULPS). Поскольку в последние байты некоторых линий могут быть заполнены инвертированным последним битом данных (в случае, если размер пакета не кратен количеству линий), последние биты также хранятся в виде отдельного поля.

Также в классе транзакции определена функция установки длины, автоматически обновляющая длину поля данных, и функция печати, позволяющая UVM выводить всё содержание транзакции при помощи функции *.sprint()*.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.1 – Определение класса mltran

Рассмотрим класс пакета более подробно (рис. 3.2). В этот класс включены все поля, входящие в типичный пакет (поля *channelID, dataType, wordCount, ecc, data, crc*). Флаг *is\_long* упрощает определение длины пакета и обновляется при каждом изменении его типа; *is\_last* выставляется для последнего сгенерированного пакета и является для следующего секвенсора / драйвера сигналом к завершению работы.

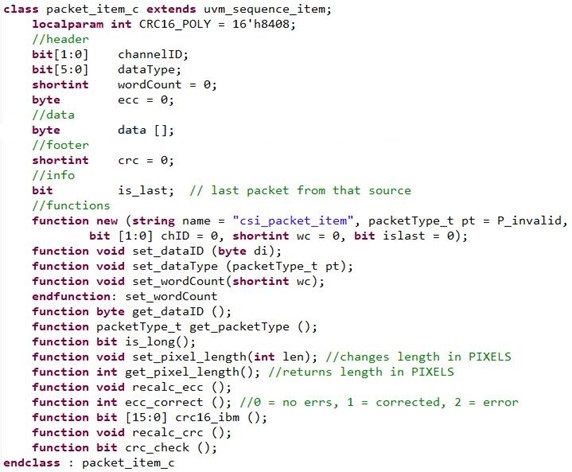


Рисунок 3.2 – Определение класса пакета

Кроме полей, для данного класса определён ряд функций, позволяющих упростить и автоматизировать работу с полями и данными пакета:

*– get\_dataID()* позволяет получить первый байт пакета (Data ID) целиком. Используется при упаковке пакета в транзакции;

*– get/set\_dataType()* позволяют задать/получить тип пакета в специальном типе *enum packetType\_t;*

*– set\_<имя\_поля>()* устанавливают какое-либо поле кадра и пересчитывают код ECC. Функция *set\_wordCount()* также меняет длину поля данных для длинных пакетов;

*– set/get\_pixel\_length()* – позволяет получить или установить длину поля данных в пикселях. Кол-во пикселей пересчитывается в количество байт и обратно в зависимости от установленного формата данных пакета;

*– recalc\_ecc()* пересчитывает поле ECC на основе заголовка пакета.

Код ECC рассчитывается на основе матрицы, приведённой в спецификации CSI. Матрица задана константой в том же файле, что и класс пакета;

*– correct\_ecc()* дополнительно сравнивает рассчитанное значение ECC с текущим содержанием поля ECC и по возможности корректирует содержание пакета. Возвращаемое функцией значение указывает на наличие ошибок и успех их исправления;

*– crc16\_ibm()*, *recalc\_crc()*, *crc\_check()* работают с полем контрольной суммы поля данных пакета. Первая функция возвращает рассчитанное значение, вторая подставляет его в соответствующее поле пакета, третья сравнивает и возвращает “1” в случае обнаружения ошибок.

# 3.1.2 Реализация секвенсоров и драйвера

Для совместимости с агентом VIVO первая ступень драйвера (секвенсор, генерирующий пакеты на основе кадров) назван “*driver\_c*”. Ссылка на конфигурационный объект агента (*config\_c*) позволяет получать настройки от конфигурационного объекта среды. Кроме того, в секвенсоре присутствует ссылка на родительский агент (*p\_agent*). Определения последовательностей показаны на рис. 3.3. Запускаемая на первом секвенсоре последовательность (*packet\_base\_seq\_c*) запрашивает кадры с генератора кадров до получения транзакции-кадра с тегом “*is\_last*”. По запросу от внешнего агента создаётся очередной пакет посредством функции *sendPacket\_short* (все поля пакета передаются функции в качестве атрибутов) или *sendPacket\_line* (формат данных берётся из конфигурационного объекта; поле данных заполняется на основе информации строки кадра).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.3 – Определения последовательностей

Для каждого формата данных определён свой порядок заполнения пакета данными строки; поддерживаются все определённые в спецификации форматы.

Вторая последовательность запускается на секвенсоре-миксере. Максимальное количество виртуальных каналов передаётся в виде параметра. При необходимости последовательность может быть заменена на более сложный алгоритм выборки пакетов.

Третья последовательность (*mltran\_base\_seq\_c*) разбивает информацию пакетов по нескольким линиям. Функция *fill\_mltran()* помещает переданный байт на определённое место в транзакции и увеличивает счётчики (*mlt\_lane*, *mlt\_pos*).

Драйвер содержит в себе все функции, необходимые для управления интерфейсом через IP D-PHY и отправки полученных транзакций. Определение класса драйвера приведено на рисунке 3.4.

Поскольку сигналы, управляющих выводом в состоянии High-Speed, влияют на выходной сигнал IP D-PHY с задержкой 20нс, вывод остальных сигналов также задерживается драйвером (для сохранения временных интервалов, заданных в конфигурации). Следующие функции непосредственно управляют входными сигналами IP D-PHY:

*– init\_if(), stop\_if()* включают и отключают IP;

*– clk\_lp()*, *data\_lp()* переводят линии CLK и DATA соответственно в состояние LPxy и устанавливают заданные значения сигналов;

*– clk\_hs()* переключает линию CLK в состояние HS. Бит *enable* позволяет приостановить передачу тактового сигнала;

*– data\_hs()* переключает линии DATA в состояние HS и выводит на набор байт по следующему фронту синхросигнала.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.4 – Определение класса драйвера

Важно отметить, что неправильное использование этих функций может привести к некорректной реакции приёмника.

Момент вызова и текущее состояние при вызове функций никак не проверяются. Поэтому необходимы функции, реализующие целиком процедуры переключения линий в соответствии с протоколом D-PHY (например, Start / End of Transaction).

В классе драйвера определены следующие операции:

*– dphy\_clk\_switch\_lp/hs()* реализуют процедуру переключения линии CLK между состояниями Low-Power и High-Speed;

*– dphy\_data\_sot()* (процедура начала транзакции) переводит линии данных в режим HS с отправкой синхронизирующей последовательности;

*– dphy\_data\_eot()* – процедура конца транзакции – вывод инвертированного последнего бита данных в течении времени THS-TRAIL с последующим переводом линий в режим LP.

Следующие функции используются для перехода в Escape Mode и ULPS (Ultra-Low Power State) и обратно:

*– dphy\_clk\_ulps\_enter/exit()* – перевод линий CLK из состояния LP в ULPS (Ultra-Low Power State) и обратно;

*– dphy\_lp\_soh()* – отправка одного бита в кодировке Spaced One-Hot (используется в режиме Escape Mode);

*– dphy\_escape\_cmd()* – отправка 8-битной команды в Escape Mode;

*– dphy\_data\_ulps\_enter()* – отправка команды для перехода ULPS (задана константой *esc\_cmd\_ULPS*) и перевод шины в это состояние;

*– dphy\_data\_ulps\_exit()* – выход из состояния ULPS.

Для передачи каждой транзакции последовательно вызываются функции начала транзакции, передачи данных и конца транзакции. В случае прерывистого поведения тактового сигнала перед транзакцией также происходит перевод линии CLK в режим High-Speed.

# 3.1.3 Реализация компонентов монитора

Как было сказано в главе 2, монитор разделён на 4 части. Первой является внешний монитор, непосредственно работающий с интерфейсом – *ext\_monitor\_c*, определение которого приведено на рис. 3.5.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 3.5 – Определение класса внешнего монитора

Аналогично драйверу, следующие функции переводят IP в заданное состояние:

*– clk\_lp, clk\_hs()* – переключение режима линии CLK;

*– data\_lp\_switch(), data\_hs\_switch()* – переключение линий DATA;

*– data\_hs\_get() –* получение следующего байта режиме High-Speed.

Важно отметить, что переключение режима IP не влияет на интерфейс, и наоборот – переключение режима функционирования интерфейса IP-передатчиком не влияет на состояние IP-приёмника.

Причина заключается в том, что для «отображения» текущего состояния при симуляции IP-передатчик использует изменение “силы” сигналов (англ. *strength*) языка System Verilog. Разница между уровнями силы заметна на временных диаграммах, но никак не может быть проверена приёмником – подобно тому, как без использования АЦП невозможно определить режим работы аналогового интерфейса D-PHY. Поэтому в соответствии с протоколом переключение режима приёмника происходит через время TD-TERM-EN после получения запроса на перевод шины в режим High-Speed (LP11–LP01–LP00).

В отличие от драйвера, все процедуры протокола (например, начало или конец транзакции) реализуются конечным автоматом. Состояние автомата для разных линий хранятся в переменных *data\_state* и *clk\_state*. Сам автомат реализуется внутри функций *dphy\_clkSM()* и *dphy\_dataSM()*. После перехода в режим High-Speed запускается функция *getTran()*, собирающая данные до момента выхода из этого состояния и отправляющая собранную транзакцию через порт. В режиме Escape Mode запускается функция *getEscapeCmd()*, получающая команду в кодировке Spaced One-Hot. В зависимости от содержания команды выполняется переход в ULPS или вывод сообщения об ошибке, но для конкретных IP могут быть добавлены дополнительные опции.

Остальные компоненты монитора реализованы в виде наследников класса *uvm\_subscriber*, поскольку данный класс предоставляет встроенный *analysis\_port* для приёма транзакций и базовые функции для корректной его работы.

Компонент *mltran2packet\_с* собирает пакет из полученных транзакций. Определение класса данного компонента приведено на рисунке 3.6.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.6 – Определение класса сборщика пакетов

При получении каждой транзакции через порт запускается функция *processTran()*. Функция *removeTrail()* определяет фактическую длину пакета, отбрасывая инвертированный последний бит, находящийся в конце транзакции. В случае, если длина для различных линий различается больше чем на 1 байт, или длина в битах не кратна 8, выводится сообщение об ошибке. После определения длин «полезные» данные смешиваются в одну очередь и функцией *mlt2pkt* создаёт новый пакет. Здесь же происходит проверка целостности заголовка (коррекция по ECC) и поля данных (CRC).

Сгенерированные и распределённые по каналам пакеты поступают на внутренний монитор агента VIVO. Определение монитора приведено на рисунке 3.7. При получении пакета запускается функция *processPacket()*.

Перед обработкой содержимого пакета проверяется его тип функцией *checkPacket\_FLSE()* (Frame/Line Start/End). Данная функция, используя механизм конечного автомата из 3х состояний (состояние хранится в переменной *cFLSE\_state*), позволяет выявить ошибки порядка поступления пакетов. Примером такой ошибки может служить повторное поступление пакетов Frame Start или пакет с данными вне кадра / строки.

Далее в зависимости от типа пакета производится изменение переменных состояния. Так, при получении пакета Frame Start номер полученного кадра сравнивается с предыдущим. Порядок поступления номеров позволяет обнаружить ошибки (например, подача кадров в неправильном порядке). При наличии пакетов Line Start сделать вывод о чересстрочности кадра. Пакеты с данными проверяются на соответствие типа и длины пакета параметрам кадра, заданным в конфигурации, после чего сохраняются. После получения пакета Frame End (в случае, если кадр поступает в чересстрочном формате, требуется получение нескольких таких пакетов) информация сохранённых пакетов записывается в транзакцию-кадр и отправляется. Аналогично генератору пакетов, для каждого формата данных существует свой алгоритм заполнения строки данными пакета.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.7 – Определение класса внутреннего монитора

Монитор для интерфейса D-PHY реализуется подобно монитору CSI, с использованием конечного автомата.

# 3.1.4 Реализация конфигурационных объектов

Все конфигурационные объекты реализуются подобным образом. На рисунке 3.8 в качестве примера приведено определение конфигурационного объекта внешнего агента. Часть временных параметров удалена.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.8 – Определение класса конфигурационного объекта

Все перечисленные атрибуты объекта являются настройками агента, перечисленными в 2.3. Функция *cfg\_update()* обновляет временные параметры, зависящие от пропускной способности интерфейса, а также запускает функцию *cfg\_check()*.

Функция *cfg\_check* является механизмом самопроверки для конфигурационного объекта и выводит сообщения об ошибке, когда какие-либо настройки (например, количество линий, превышающее максимальное) делают корректную работу агента невозможной или приводят к системным ошибкам. Кроме того, происходит проверка временных параметров (для драйвера). В случае, если параметры не соответствуют требованиям, заданным в спецификации интерфейса CSI, выводится предупреждающее сообщение.

# 3.2 Интеграция разработанного VIP в библиотеку VIVO

Для новых блоков в окружение VIVO введены 2 типа интерфейса. Первый, *csi\_ppi\_if*, используется для соединения внешних агентов (монитора или драйвера) с IP D-PHY. Для сохранения возможности автоматического подключения с помощью макросов было решено объединить интерфейсы, совместимые с IP-передатчиком и IP-приёмником в один интерфейс, создав отдельные порты для монитора и драйвера. Тем не менее, поскольку некоторые сигналы используются обоими портами и каждый порт предназначен для взаимодействия с отдельным IP, одновременное включение внутри внешнего агента монитора и драйвера невозможно. По этой причине при инициализации агента (в фазе *build\_phase*) в зависимости от установленного поведения создаются либо компоненты монитора, либо компоненты драйвера.

Второй интерфейс, *csi\_bidir\_if*, подключается вручную в top-модуле окружения и соединяет используемые IP между собой и/или с тестируемым устройством (DUT).

Поскольку основным принципом библиотеки VIVO является автоматическая генерация агентов на основе базового агента-генератора изображений, встроить внешний агент в ту же верификационную среду без серьёзного изменения функций генерации не представляется возможным.

Вместо этого создан дочерний класс среды и конфигурационный класс, хранящий ссылки на конфигурационные объекты новых агентов и монитора D-PHY. Общая структура нового окружения приведена на рисунке 3.9.

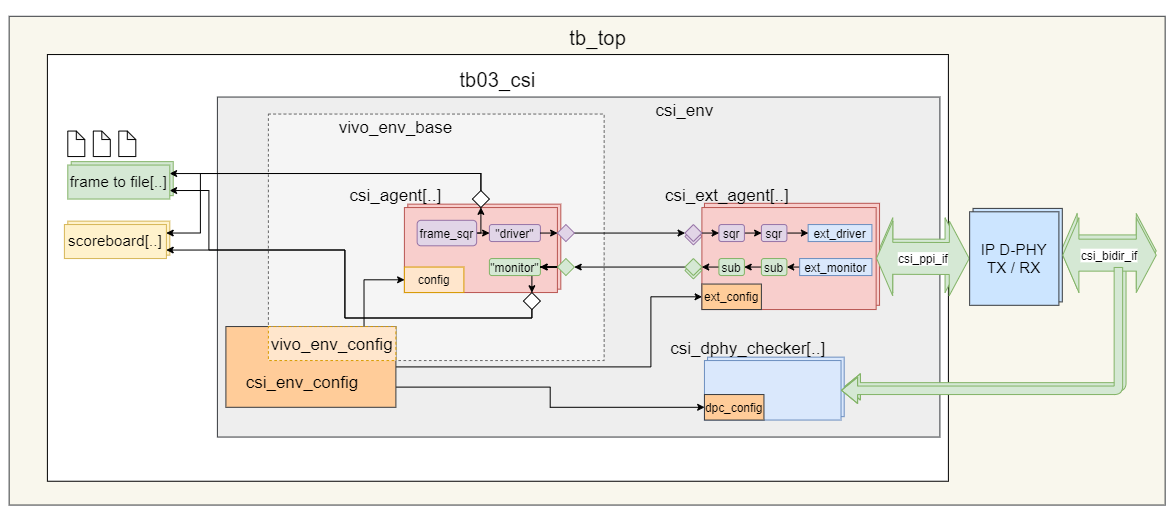


Рисунок 3.9 – Верификационное окружение с внешними агентами CSI

В оригинальном окружении VIVO конфигурационный объект среды содержал функцию *read\_from\_file()*, обеспечивающую создание конфигов по записям в файле конфигурации. В данный класс были добавлены конфигурационные объекты для внутренних агентов CSI, задачей которых является генерация пакетов из кадров и наоборот. Поскольку интерфейсы для агентов, подключённых к разным IP TX и IP RX, имеют отличающиеся номера, анализ поля *if\_id* у созданных конфигурационных объектов CSI позволяет определить количество и типы требуемых внешних агентов. Таким образом, количество внутренних агентов, связанных с одним внешним, будет равно количеству виртуальных каналов внешнего агента. Такое подключение схематично изображено на рисунке 3.10.

Полученные значения сохраняются в конфиг. объекты *csi\_ext\_cfg[]*. Помимо этого, создаются объекты для мониторов всех интерфейсов *csi\_bidir\_if*. Количество таких интерфейсов определяется внешним макросом.

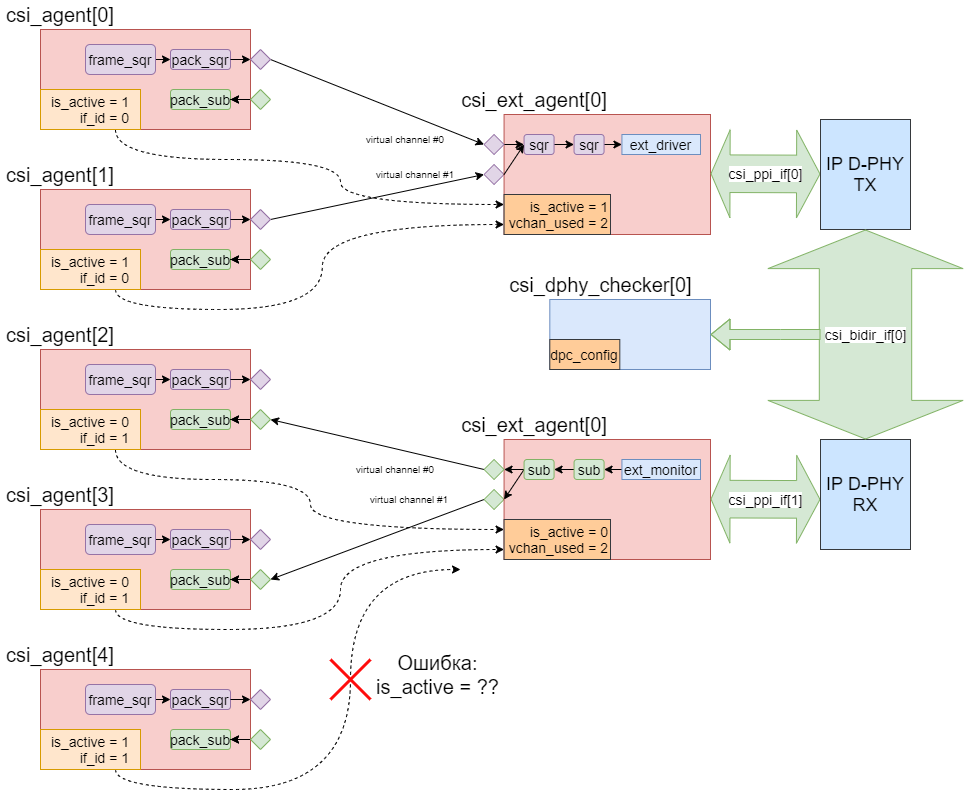


Рисунок 3.10 – Методика подключения внешних агентов

На основе созданных конфигурационных объектов среда *csi\_env* создаёт необходимое количество агентов каждого типа. В UVM-фазе *connect\_phase* используемые порты внешних агентов автоматически подключаются к внутренним. Важно отметить, что номера виртуальных каналов выбираются по порядку объявления. Например, если внутренние агенты с номерами 2, 3, 5 будут подключаться к одному интерфейсу, то внутри этого интерфейса им будут присвоены виртуальные каналы 0, 1, 2.

# 3.3 Тестирование разработанного VIP

Для проверки соответствия разработанного верификационного IP заданным требованиям было проведено его тестирование в составе обновлённого окружения.

# 3.3.1 Разработка тестового окружения

Для работы с новым окружением также был разработан новый класс теста и top-модуль. Класс теста реализует создание всех компонентов окружения (тестовая среда, её конфигурационный объект, драйверы тактовых сигналов, а также классы для сравнения кадров (компараторы – *frame\_cmp\_c*)) в фазе UVM *build\_phase*. В то же время происходит чтение файла конфигурации и создание конфиг. объектов для различных интерфейсов, в т. ч. CSI.

В фазе *connect\_phase* агенты выборочно подключаются к компараторам кадров. В базовой версии теста на компаратор подаются кадры с порта драйвера (сразу после их генерации) и кадры, полученные монитором (восстановленные из пакетов, переданных по интерфейсу CSI).

В фазе *run\_phase* агенты CSI и мониторы D-PHY включаются с определённой задержкой, после чего на секвенсорах активных агентов происходит запуск последовательностей.

Модуль *tb\_top* отвечает за подключение теста к IP. При помощи макросов происходит создание интерфейсов и их привязка к драйверам и мониторам по определённому шаблону. После генерации интерфейсов к ним подключаются используемые IP. Кроме того, устанавливаются параметры тактовых сигналов. При необходимости в этом модуле могут устанавливаться и другие параметры внешних агентов CSI.

# 3.3.2 Тестирование работы драйвера с монитором

Первым этапом тестирования стала проверка работоспособности драйвера, подключённого напрямую к монитору такого же агента. Установлены временные характеристики, соответствующие спецификации CSI, и включены все проверки на мониторе. Временные диаграммы передачи пакета (рис. 3.11) соответствуют процедурам, описанным в спецификации CSI (рис. 3.12).

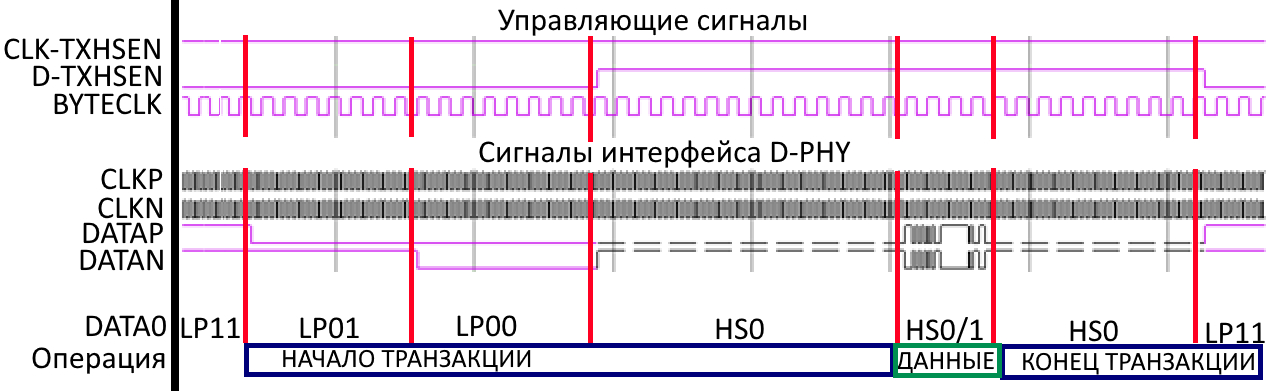


Рисунок 3.11 – Временная диаграмма отправки длинного пакета

На диаграмме используются следующие обозначения сигналов:

*–*TXHSEN – сигналы, управляющие переключением передатчика между режимами LP (0) / HS (1) для линий CLK и DATA (D);

*–*BYTECLK – внутренний тактовый сигнал передатчика, за период которого отправляется 1 байт данных;

*–*CLKP/N – линия тактового сигнала;

*–*DATAP/N – линии данных.

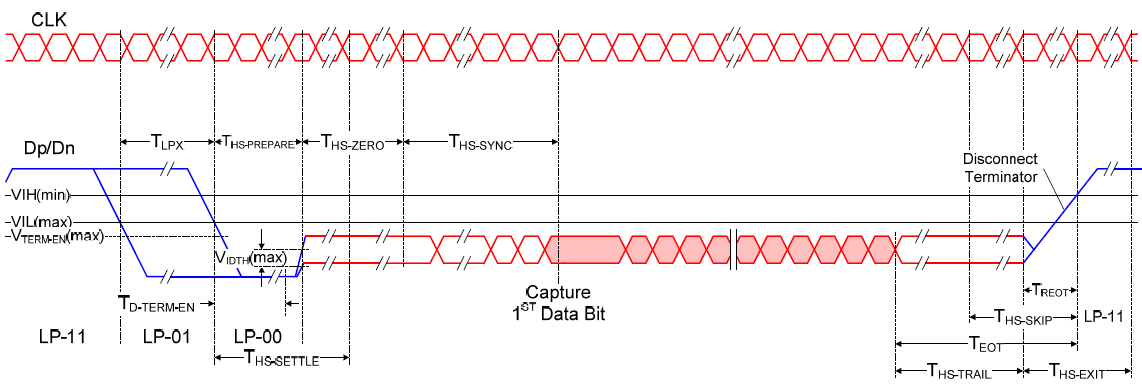


Рисунок 3.12 – Диаграмма отправки пакета, приведённая в спецификации

На приведённых диаграммах для всех линий чёрным пунктиром отмечено состояние High-Speed, фиолетовой линией – состояние Low-Power.

После проверки драйвера в конфигурации с одной линией их количество было увеличено до 4. Кроме того, тактовый сигнал был переведён в прерывистый (non-continuous) режим, в результате чего линия CLK переводилась в режим Low-Power после отправки пакета. Передача пакетов с суммарной длиной, не кратной количеству линий, также завершилась успешно. На рис. 3.13 приведена диаграмма моделирования с выделенными этапами передачи данных и переходов линий между состояниями. Для сокращения места длинные участки без изменений сжаты на диаграмме.

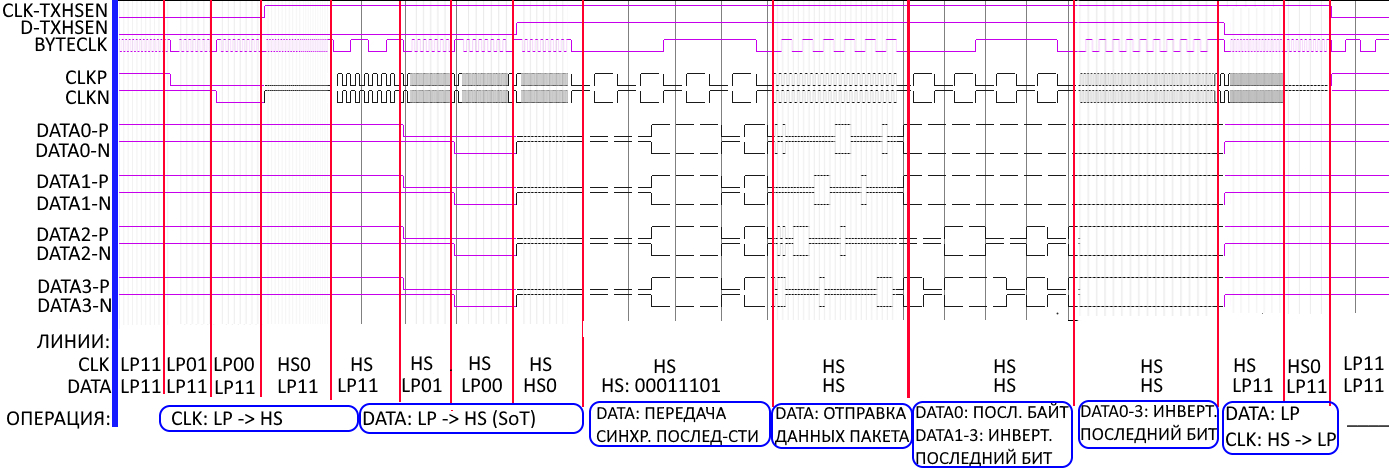


Рисунок 3.13 – Работа драйвера в конфигурации с 4 линиями  
и прерывистым режимом работы линии CLK

Также была проведена проверка перехода в режим Escape Mode и Ultra-Low Power State. Диаграмма перевода линий CLK и DATA в данный режим приведена на рисунке 3.14. Длинные промежутки также сжаты.

Поскольку особенности последующих тестов не различимы на временной диаграмме, в качестве рисунков будет приводится результат сравнения отправленных и полученных кадров. В журнале симуляции *cmp\_count* – количество принятых кадров, *err\_count* – количество не совпавших кадров.

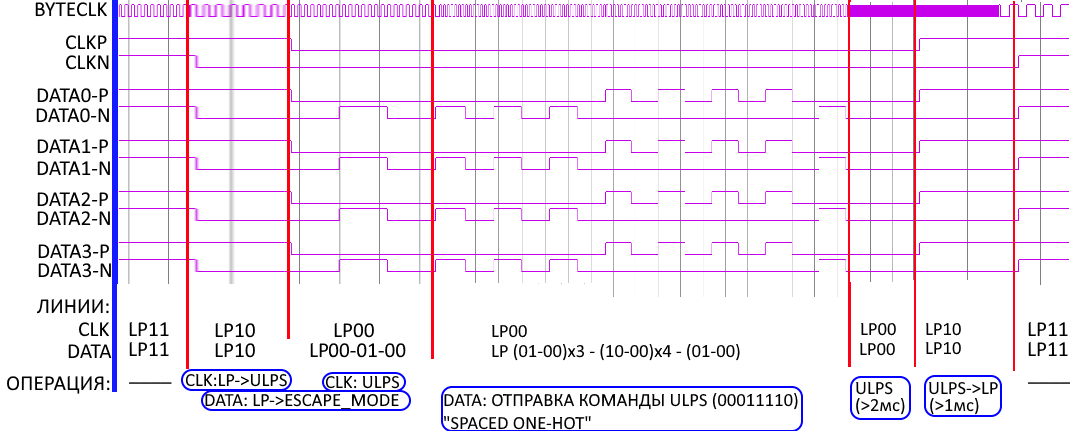


Рисунок 3.14 – Перевод линий CLK и DATA в состояние ULPS

После проверки различных режимов работы драйвера и монитора с установкой разных флагов было увеличено количество виртуальных каналов. В тесте, результаты которого приведены на рисунке 3.15, использовалось 2 генератора (и, соответственно, 2 приёмника) изображений с различающимися размерами и форматом данных кадра. Кроме того, через один из каналов кадры подавались чересстрочно, с передачей пакетов начала и конца строки. Тест прошёл успешно, все кадры были получены без ошибок.

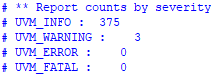
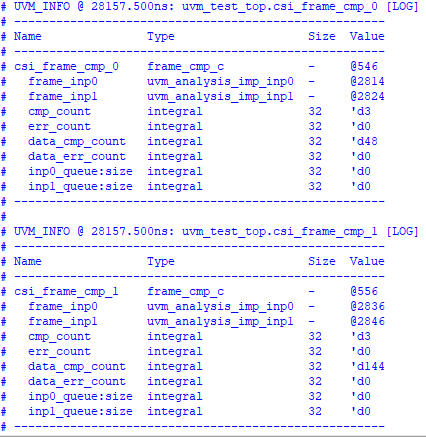


Рисунок 3.15 – Проверка работы нескольких виртуальных каналов

# 3.3.3 Тестирование драйвера с IP CSI

Для дальнейшего использования драйвера необходимо удостовериться, что выводимые им данные и их временные характеристики, будут совместимы с используемыми IP для интерфейса CSI. В качестве примера был взят бесплатно распространяемый IP-приёмник, разработанный Lattice Semiconductors. Данный модуль реализует функции всех слоёв протокола и подаёт на выход все поля полученного пакета. Для сбора данных с данного IP был разработан тестовый монитор. Восстановленный на мониторе пакет без дополнительной обработки отправляется на монитор выделенного внутреннего агента CSI. Интерфейсу для тестового внешнего агента присвоен номер 255 (максимальный из возможных). После добавления нового монитора в тестовую среду и отладки его работы вывод данного монитора был сравнён с генерируемыми кадрами. Результат тестирования приведён на рисунке 3.16.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.16 – Результат тестирования с IP CSI

Поскольку отправленные кадры совпали как с принятыми через монитор VIP, так и через IP CSI, можно сделать вывод, что VIP соответствует требованиям спецификации CSI.

# 3.4 Составление руководства пользователя

Для удобства использования разработанного драйвера в составе библиотеки VIVO было составлено руководство пользователя.

В руководстве подробно описаны различные возможности настройки агента, приведены предельные значения временных параметров и расшифрованы возможные ошибки. Также описан процесс подключения верификационного IP к тестовому окружению. Руководство пользователя приведено в приложении А.

# 3.4 Выводы главы 3

В данной главе был описан процесс реализации VIP на языке SystemVerilog в соответствии с алгоритмами и графами автоматов, описанных в главе 2.

Было произведено интегрирование разработанного агента интерфейса CSI в окружение VIVO, разработка среды, позволяющей автоматически подключать, внешний агент к базовому окружению.

Было произведено комплексное тестирование различных функций драйвера и монитора. Кроме того, проведена проверка соответствия драйвера требованиям спецификации путём подключения агента к IP интерфейса CSI.

Также было составлено руководство пользователя для разработанного VIP.

# ЗАКЛЮЧЕНИЕ

В данной работе был проведён обзор интерфейса CSI и его физического уровня (MIPI01 D-PHY). Рассмотрены различные методики верификации, структура и иерархия классов методологии UVM (Universal Verification Methodology), определённой в стандарте IEEE 1800.2–2017. Также была описана структура верификационного окружения VIVO, в составе которого планируется использование верификационного IP.

Была разработана структура верификационного агента для интерфейса CSI в соответствии с методологией UVM. Подробно описаны алгоритмы работы различных его компонентов, составлены графы переходов для конечных автоматов мониторов интерфейса CSI и его физического уровня D-PHY. Разработан механизм конфигурации различных компонентов VIP и составлен план проведения тестирования.

Все разработанные компоненты были реализованы на языке описания оборудования System Verilog в соответствии с методологией UVM. Для интеграции полученного агента в окружение VIVO тестовая среда окружения была модифицирована, что позволило автоматически создавать внешние агенты для интерфейса CSI без внесения серьёзных изменений в механизм конфигурации среды. Общий объём написанного кода – более 5000 строк.

Было проведено комплексное тестирование разработанного VIP в составе окружения VIVO. Произведена отладка всех функций и вариантов конфигурации агента. Проверка совместимости выводимого драйвером сигнала со спецификацией CSI реализована путём его подключения к IP CSI с парсером пакетов, разработанным Lattice Semiconductors.

Составлено руководство пользователя Реализованный верификационный IP CSI готов к использованию в составе библиотеки VIVO. Таким образом, поставленная задача выполнена в полном объёме.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. New Version of Most Widely Used Camera and Imaging Interface—MIPI CSI—Designed to Build Capabilities for Greater Machine Awareness  
   URL: <https://www.mipi.org/node/1800> (дата обращения: 06.04.2021).
2. MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2) – MIPI, 2009. URL: <http://caxapa.ru/thumbs/799244/MIPI_Alliance_Specification_for_Camera_S.pdf> (дата обращения: 06.04.2021)
3. MIPI Alliance Specification for D-PHY – MIPI, 2009. URL: <http://www.jmrcubed.com/vr/ref_tech/mipi_d_phy_specification_v01-00-00.pdf> (дата обращения: 06.04.2021)
4. Сhris S. SystemVerilog for Verification // Springer Science+Business Media – 2008 – 400 с. URL: <https://www.springer.com/gp/book/9781441945617> (дата обращения: 06.04.2021)
5. Rob Dekker. What’s the Difference Between VHDL, Verilog, and SystemVerilog? // What’s Difference Between – Electronics Design  
   URL: [https://www.electronicdesign.com/resources/whats-the-difference-between/article/21800239](https://www.electronicdesign.com/resources/whats-the-difference-between/article/21800239/whats-the-difference-between-vhdl-verilog-and-systemverilog) (дата обращения: 22.03.2021)
6. Stephen Bailey. Comparison of VHDL, Verilog and SystemVerilog – Model Technology. URL: <https://wikis.ece.iastate.edu/cpre584/images/8/8a/01-LanguageWhitePaper.pdf> (дата обращения: 06.04.2021)
7. Joonas Puskala. High-Speed Camera Serial Interface Verigication – Tampere University of Technology, 2014. URL: <https://core.ac.uk/download/pdf/250163148.pdf> (дата обращения: 06.04.2021)
8. Tomas L. Anderson Verifying SoCs from the Inside Out // ChipDesign,  
   URL: <http://chipdesignmag.com/display.php?articleId=5153> (дата архивации страницы: 09.04.2014)
9. IEEE 1800.2-2017 Universal Verification Methodology – IEEE, 2017. URL: <https://www.francisz.cn/download/IEEE_Standard_UVM_1800.2-2017.pdf> (дата обращения: 06.04.2021)
10. Universal Verification Methodology (UVM) 1.2 User’s Guide – Accellera, 2015. URL: <https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf> (дата обращения: 06.04.2021)
11. Hierarchal Testbench Configuration Using uvm\_config\_db – Synopsys, 2014. URL: <https://www.synopsys.com/content/dam/synopsys/services/whitepapers/hierarchical-testbench-configuration-using-uvm.pdf> (дата обращения: 06.04.2021)
12. Finite State Machine: Mealy State Machine and Moore State Machine // URL: <https://www.elprocus.com/finite-state-machine-mealy-state-machine-and-moore-state-machine/> (дата обращения: 30.03.2021)
13. UVM 1.2 Class Reference. URL: <https://verificationacademy.com/verification-methodology-reference/uvm/docs_1.2/html/index.html> (дата обращения: 06.04.2021)
14. CrossLink High-Speed I/O MIPI D-PHY and DDR Interfaces Technical Note. URL: <http://www.latticesemi.com/-/media/LatticeSemi/Documents/ApplicationNotes/AD/CrossLinkHighSpeedIOMIPIDPHYandDDRInterfaces.ashx?document_id=51653> (дата обращения: 06.04.2021)
15. CSI-2/DSI D-PHY Receiver Submodule IP User Guide. URL: <http://www.latticesemi.com/-/media/LatticeSemi/Documents/UserManuals/E---I-Folder-3/FPGA-IPUG-02025-1-4-CSI2-DSI-DPHY-Receiver-Submodule-IP.ashx?document_id=52136#:~:text=The%20Lattice%20Semiconductor%20CSI%2D2,equipment%2C%20and%20many%20other%20applications> (дата обращения: 06.04.2021)

# ПРИЛОЖЕНИЯ

# Приложение А – Руководство пользователя

# ВЕРИФИКАЦИОННЫЙ IP CSI-2 v1.0 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

# Введение

В данном руководстве описываются принципы использования верификационного IP для интерфейса CSI-2, выпущенного Mobile Industry Processor Interface (MIPI) Alliance в 2005 году. Данный VIP был разработан в соответствии с методологией Universal Verification Methodology (IEEE 1800.2-2017) с применением библиотеки классов UVM.

Данный VIP разработан для использования в составе библиотеки верификационных компонентов для видео-интерфейсов «VIVO». Библиотека «VIVO» создана для верификации устройств, использующих множество различных интерфейсов для ввода и/или вывода информации изображений и видео. Конфигурационный механизм окружения позволяет легко настраивать набор верификационных IP под конкретное устройство, управлять генерацией и направлением потока видеоданных. Предоставляются возможности для гибкой конфигурации используемых VIP. Настройки каждого из агентов могут быть изменены в том числе и во время симуляции с помощью интерфейса транзакций.

Использование верификационного IP вне указанной библиотеки невозможно, поскольку компоненты «VIVO» использовались в качестве базы для разработки агента.

Используемый интерфейс совместим с IP-блоками D-PHY RX/TX, разработанными Lattice Semiconductors [15].

# 1 Обзор возможностей верификационного IP

Данный VIP предоставляет множество инструментов для верификации устройств, использующих интерфейс CSI-2 v1.0. В состав верификационного IP входят:

– Драйвер, способный получать транзакции-кадры с нескольких источников и отправлять их через интерфейс CSI посредством взаимодействия с IP D-PHY TX;

– Монитор, принимающий транзакции с IP D-PHY TX и выдающий полученные кадры, разделённые по номеру виртуального канала;

– Чекер, подключаемый напрямую к интерфейсу D-PHY и обеспечивающий проверку корректности переходов между состояниями, а также временных характеристик работы передатчика.

Для наиболее подробной проверки функционирования любого передатчика монитор и чекер должны использоваться совместно для обнаружения ошибок на всех уровнях функционирования передатчика CSI.

# 2 Подключение VIP к верификационному окружению

Поскольку не все компоненты данного VIP используют механизм автоматической генерации окружения, используемый в библиотеке «VIVO», использование базового окружения *vivo\_env*, входящего в состав библиотеки, требует ручного подключения компонентов к тестовой среде и не рекомендуется.

Вместе с файлами верификационного IP предоставляется модифицированная тестовая среда *csi\_env*, в которую уже включены агенты для интерфейса CSI, и дополненный для использования с этой средой класс конфигурации *csi\_env\_cfg*. Указанные классы полностью включают в себя базовый функционал окружения «VIVO» и не требуют серьёзного изменения окружения при замене базовых компонентов на изменённые. Подключение тестовой среды «VIVO» описано в руководстве пользователя для данной библиотеки.

Типичное подключение VIP интерфейса CSI приведено на рисунке 1.

Агенты *csi\_env.csi\_agent[..]* создают на основе полученных кадров пакеты, соответствующие формату спецификации CSI. Полученные пакеты отправляются на *csi\_env.csi\_ext\_agent[..]*. Номер агента соответствует номеру интерфейса, выбранному при настройке агентов-генераторов пакетов. К одному верификационному агенту *csi\_env.csi\_ext\_agent[..]* может быть подключено несколько генераторов пакетов.

Отдельным компонентом является чекер (*csi\_env.csi\_dphy\_checker[..]*) интерфейса D-PHY. Такие чекеры создаются автоматически для каждого интерфейса *csi\_bidir\_if[..]*, в зависимости от макроса *OHT\_VIVO\_CSI\_BIDIR\_IF\_N*.

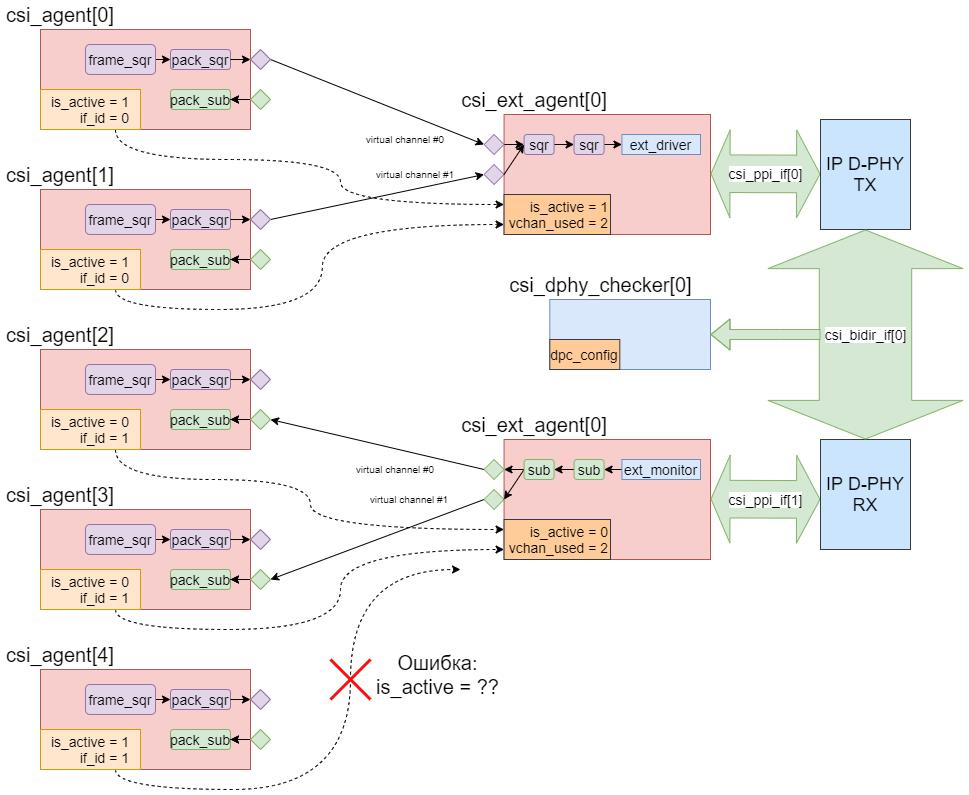


Рисунок 1 – Пример подключения агентов CSI

Для успешного соединения VIP с интерфейсом необходимо:

1) Установить макрос *OHT\_VIVO\_CSI\_PPI\_IF\_N* таким образом, чтобы номера используемых агентами CSI интерфейсов не превышали значение макроса;

2) В верхнем модуле окружения подключить к интерфейсу *csi\_ppi\_if[..]* с соответствующим номером IP, обеспечивающий взаимодействие с интерфейсом D-PHY в направлении, совпадающем с настройками агентов, подключённых к данному интерфейсу. RTL-код для данных блоков свободно распространяется и может быть сгенерирован в бесплатном программном обеспечении Lattice Diamond;

3) Установить макрос *OHT\_VIVO\_CSI\_BIDIR\_IF\_N* – количество требуемых интерфейсов D-PHY;

4) Соединить используемый IP с требуемым интерфейсом *csi\_bidir\_if[..]* (D-PHY).

**Примечания**

– Поскольку используемые IP TX и IP RX имеют общие сигналы, к одному агенту может быть подключён только один IP. Отсюда у одного агента всегда функционирует только одна составляющая – либо монитор, либо драйвер.

– Номера интерфейсов *csi\_ppi\_if[..] (1)* никаким образом не связаны с номерами интерфейсов *csi\_bidir\_if[..] (2)*.

– Даже в случае, если два интерфейса 1 подключены к одному интерфейсу 2, они должны иметь различные номера.

– При использовании новой тестовой среды агенты *csi\_env.csi\_ext\_agent[..]* подключаются к созданным агентам *csi\_env.csi\_agent[..]* автоматически. Дополнительных действий от пользователя не требуется.

# 3 Конфигурация верификационного IP

# 3.1 Настройки генерации пакетов

Генератор пакетов настраивается подобно другим агентам библиотеки «VIVO». Из файла конфигурации, определённого макросом *OHT\_VIVO\_CFG\_FILE\_PATH*, считываются конфигурационные записи следующего формата:

91 длина записи в символах

1,1 *is\_active*, *generator*

6#csi тип интерфейса (6 = csi)

4,4,60.00 ширина, высота кадра, [не используется]

0,0 номер интерфейса, [не используется]

frame\_n\_period,4 другие параметры (имя,значение)

intFormat,42

sendLineSE,0

interlaced,0

В первой строке записи указывается её размер. Размер записи без дополнительных параметров – 11 байт, каждой строки, добавляющей дополнительные параметры – 20 байт. Первая строка файла должна содержать сумму размеров всех записей.

Параметр “is\_active” управляет режимом работы агента (1 = драйвер, 0 = монитор). В случае, если агент работает в режиме драйвера, параметр “generator” позволяет управлять генерацией кадров (1 = кадры генерируются внутри агента, 0 = агент получает кадры через порт, например, от другого агента).

Основные параметры описаны более подробно в руководстве на библиотеку «VIVO».

После основных параметров следуют уникальные для данного VIP параметры в виде пары «*имя, значение*», где *значение* – целое число. Список всех настроек для монитора и драйвера приведён в таблице 1.

Таблица 1. Параметры конфигурации генератора/приёмника пакетов

|  |  |  |
| --- | --- | --- |
| Имя параметра | Описание | Значение по умолч. |
| **Настройки драйвера** | | |
| *intFormat* | Установка используемого формата данных. В случае, если агент являтся генератором кадров, также влияет на параметры кадров. Номера для различных форматов указаны в спецификации CSI и указываются в десятичном виде. Примеры:  42 – RAW8 (0x2A);  36 – RGB888 (0x24). | 42 |
| *sendLineSE* | Включение отправки пакетов начала и конца строк (Line Start / End):  0 – пакеты LS/LE не отправляются;  1 – пакеты LS/LE отправляются. | 0 |
| *interlaced* | Включение чересстрочной отправки кадров:  0 – прогрессивная развёртка;  1 – чересстрочная развёртка (первое поле содержит нечётные линии);  2 – чересстрочная развёртка (первое поле содержит чётные линии). | 0 |
| *frame\_n\_period* | Период номеров отправляемых кадров.  Пример: для периода 3, кадры будут отправляться со следующими номерами: 1, 2, 3, 1, 2, 3, … | 4 |
| **Настройки монитора** | | |
| *checkLineSE* | Включение проверки наличия пакетов LS/LE для каждой строки:  0 – проверка отключена;  1 – проверка включена, при отсутствии пакетов будет выводиться ошибка. | 0 |
| *checkFieldOrder* | Включение проверки порядка полей для чересстрочной развёртки:  0 – проверка отключена;  1 – проверка включена, при обратном порядке полей будет выводиться ошибка. | 0 |

# 3.2 Настройки взаимодействия с интерфейсом

Агенты имеют 2 параметра: макс. количество доступных линий (*LANES\_MAX)* и макс. количество виртуальных каналов *(VCHAN\_MAX*). Первый параметр должен быть установлен исходя из количества линий используемых IP, второй – в соответствии с максимально возможным количеством агентов-генераторов пакетов, подключённых к одному интерфейсу. В среде *csi\_env* данные параметры задаются макросами.

Следующие настройки не используют механизм конфигурации «VIVO» и должны быть установлены с использованием uvm\_config\_db. Пример установки одного из параметров для агента №1 интерфейса 0:

*uvm\_config\_db#(bit)::set(root, "\*.csi\_ext\_agent\_1\_if0.cfg", "en\_clk\_ulps", 1);*

Установка параметров при помощи config\_db должна быть выполнена до начала фазы UVM *build\_phase*, поскольку в данной фазе происходит создание агентов с установленной конфигурацией. Список всех настроек для внешнего агента приведён в таблице 2.

Существует возможность установки параметров напрямую (*csi\_env.csi\_ext\_agent[..].cfg.<имя\_параметра>*). Такой метод рекомендуется использовать для задержки включения (установки параметра *enabled*), а также для изменения временных параметров во время работы.

Таблица 2. Параметры конфигурации внешнего агента интерфейса CSI

|  |  |  |
| --- | --- | --- |
| **Имя параметра** | **Описание** | **Значение по умолч.** |
| *enabled* | Включение агента.  Для драйвера момент включения означает, что можно начинать взаимодействие с интерфейсом. Монитор должен быть включён после снятия сброса с DUT и инициализации интерфейса (на интерфейсе D-PHY не должно быть X) | 0 |

Продолжение таблицы 2.

| **Имя параметра** | **Описание** | **Значение по умолч.** |
| --- | --- | --- |
| *lanes\_used* | Количество используемых линий. Влияет на использование линий присоединённого интерфейса. Не должно превышать *LANES\_MAX.* | 4 |
| *vchan\_used* | Количество используемых виртуальных каналов. Влияет на количество используемых портов (агентов-генераторов/приёмников пакетов) при выводе пакетов CSI. Не должно превышать значение параметра LANES\_MAX. | 4 |
| *data\_rate* | Пропускная способность используемого IP. Используется для расчёта полупериода тактового сигнала (линии CLK) в режиме High-Speed:  t\_ui= 1/data\_rate | 1000 |
| **Настройки драйвера** | | |
| *continuous\_clock* | Переключение режима линии CLK:  0 – прерывистый (non-continuous);  1 – постоянный (continuous).  Подробности описаны в спецификации CSI. | 1 |
| *ulps\_after\_frame* | Включение режима ULPS (Ultra-Low Power State) на время t\_ulps после отправки каждого кадра. Может использоваться для проверки возможности перехода приёмника в этот режим. | 0 |
| *en\_clk\_ulps* | Включение перевода линии CLK в режим ULPS вместе с линиями DATA:  0 – линия CLK не переводится в ULPS;  1 – линия CLK переводится в ULPS. | 1 |
| **Настройки монитора** | | |
| *strictEscapeCheck* | Включение жёсткой проверки команд в режиме Escape Mode:  0 – при получении неизвестной команды выводится предупреждение;  1 – при получении неизвестной команды выдаётся ошибка. | 1 |

Продолжение таблицы 2.

| **Имя параметра** | **Описание** | **Значение по умолч.** |
| --- | --- | --- |
| **Временные параметры (драйвер) – измеряются в наносекундах (нс)** | | |
| *t\_lpx* | Длительность состояния в режиме LP. | 50 |
| *t\_hs\_prepare* | Время подачи LP-00 перед переходом в HS-0 при переключении линии DATA в режим High-Speed. | 65 |
| *t\_hs\_zero* | Время подачи HS-0 перед синхронизирующей последовательностью при переключении линии DATA в режим HS. | 100 |
| *t\_hs\_trail* | Время подачи инвертированного последнего бита после окончания HS-транзакции по линии DATA. | 80 |
| *t\_hs\_exit* | Время подачи LP-11 после выхода из режима High-Speed. | 120 |
| *t\_wakeup* | Время подачи состояния Mark-1 при выходе из режима ULPS. | 1,100,000 |
| *t\_clk\_prepare* | Время подачи LP-00 перед переходом в HS-0 при переключении линии CLK в режим High-Speed. | 70 |
| *t\_clk\_zero* | Время подачи HS-0 перед началом тактового сигнала при переключении линии CLK в режим HS. | 260 |
| *t\_clk\_pre* | Время, на протяжении которого тактовый сигнал (CLK) в режиме High-Speed подаётся перед началом перехода любой линии данных в HS. | 16 |
| *t\_clk\_post* | Время, на протяжении которого тактовый сигнал (CLK) в режиме High-Speed подаётся после перехода последней линии данных в HS. | 140 |
| *t\_clk\_trail* | Время подачи HS-0 на линию CLK после окончания тактового сигнала. | 80 |
| *t\_ulps* | Длительность режима ULPS. | 2,000,000 |
| **Временные параметры (монитор) — измеряются в наносекундах (нс)** | | |
| *t\_d\_term\_en* | Время переключения линии DATA в режим HS при запуске High-Speed тактового сигнала (отсчитывается с момента перехода в LP-00 при перекл. в HS). | 20 |
| *t\_hs\_settle* | Время, в течение которого игнорируются любые переключения линии DATA (отсчитывается с момента перехода в LP-00 при перекл. в HS). | 110 |
| *t\_hs\_skip* | Время **перед** переключением в LP-11 (при переключении в Low-Power), на протяжении которого игнорируются любые переключения линии DATA. | 45 |
| *t\_clk\_settle* | Время, в течение которого игнорируются любые переключения линии CLK (отсчитывается с момента перехода в LP-00 при перекл. в HS) | 200 |
| *t\_clk\_term\_en* | Время переключения линии CLK в режим HS при начале High-Speed транзакции (отсчитывается с момента перехода в LP-00 при перекл. в HS) | 30 |
| *t\_clk\_miss* | Время отсутствия тактового сигнала, после которого монитор переключает линию CLK в режим Low-Power | 40 |

***Примечания***

– Подробнее назначения различных временных интервалов разобраны в спецификации CSI-2. Для визуального представления большинство временных отрезков отображены на рисунках 2 и 3.

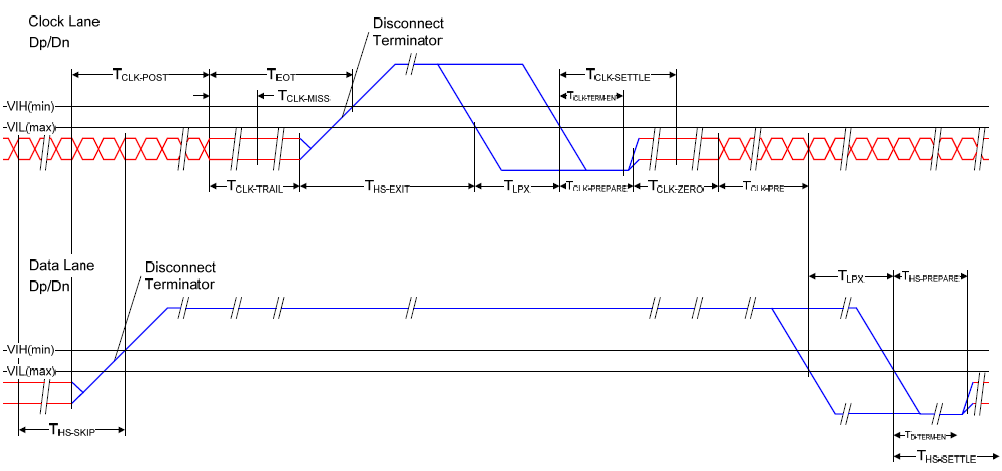


Рисунок 2 – Временные параметры, связанные с линией CLK

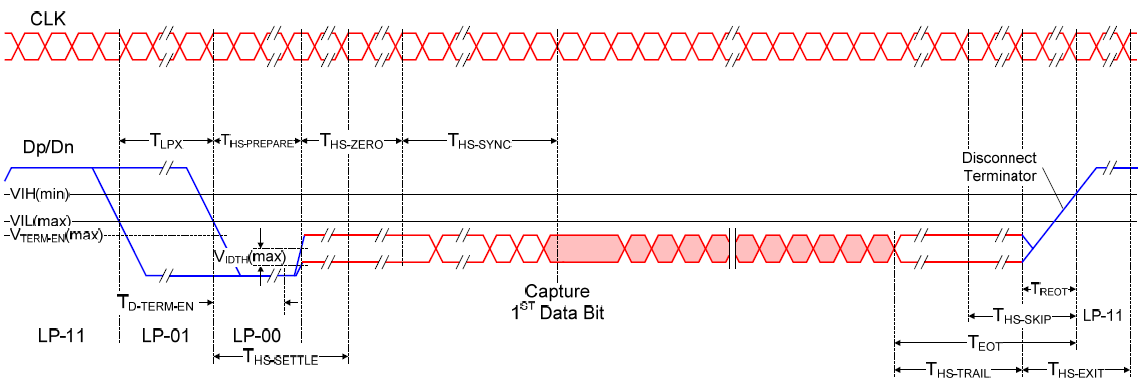


Рисунок 3 – Временные параметры, связанные с линией DATA

– Вызов функции *csi\_env.csi\_ext\_agent[..]*.*cfg\_update()* обновляет временные ограничения в соответствии со спецификацией, основываясь на новом значении *data\_rate*. В случае, если какой-либо из параметров выходит за границы определённых для него интервалов, выводится предупреждение.

– Предупреждения могут быть проигнорированы, но выводимый драйвером сигнал может быть не распознан на приёмнике (аналогично, монитор может неправильно воспринять входящий сигнал).

# 3.3 Настройки чекера интерфейса D-PHY

Чекер имеет только один жёсткий параметр: максимальное количество доступных линий (*CSI\_LANES\_MAX)*.

Настройки чекера также должны быть установлены с использованием uvm\_config\_db. Список всех настроек чекера приведён в таблице 3. Пример установки одного из параметров для чекера интерфейса 0:

*uvm\_config\_db#(int)::set(root, "\*.csi\_dphy\_checker\_1\_if1.cfg", "lanes", 4);*

Существует возможность установки параметров напрямую (*csi\_env.csi\_dphy\_checker[..].cfg.<имя\_параметра>*). Такой метод рекомендуется использовать для задержки включения (установки параметра *enabled*), а также для изменения временных параметров во время работы.

Таблица 3. Параметры конфигурации чекера D-PHY.

|  |  |  |
| --- | --- | --- |
| **Имя параметра** | **Описание** | **Значение по умолч.** |
| *enabled* | Включение чекера.  Чекер должен быть включён после снятия сброса с DUT и инициализации интерфейса (на интерфейсе D-PHY не должно быть X). | 0 |
| *lanes* | Количество проверяемых линий. Поведение других линий будет игнорироваться. Не должно превышать значение параметра *CSI\_LANES\_MAX.* | 4 |
| *data\_rate* | Пропускная способность используемого IP. Используется для расчёта полупериода тактового сигнала (линии CLK):  *t\_ui = 1/data\_rate* | 1000 |
| *strictEscapeCheck* | Включение жёсткой проверки команд в режиме Escape Mode:  0 – при обнаружении неизвестной команды выводится предупреждение;  1 – при получении неизвестной команды выдаётся ошибка. | 1 |
| *strictTuiCheck* | Включение жёсткой проверки отклонения фронта тактового сигнала:  0 – длительность тактового сигнала проверяется только на превышение максимального значения;  1 – отклонение любого фронта не может превышать 15% от длительности сигнала (на основе полученных ранее синхронизирующих импульсов). | 1 |

***Примечания***

– Временные ограничения, проверяемые чекером, рассчитываются на основе *data\_rate*. Формулы и конкретные значения ограничений приведены в спецификации CSI-2 [Ы].

# 4 Описание выводимых сообщений

Во время работы VIP могут выводиться различные виды сообщений – ошибки, предупреждения, а также информация журнала. В таблице 3 описаны основные типы сообщений.

Таблица 3. Основные типы сообщений, генерируемых VIP CSI

|  |  |
| --- | --- |
| **Тип сообщения** | **Описание** |
| **Ошибки** | |
| CSI\_UNKNOWN | Получение неподдерживаемого / несуществующего формата данных.  Методы исправления: проверить используемые форматы данных в настройках или связанных модулях. |
| CSI\_SPL | Ошибка при установке длины пакета в пикселях. Возможные причины:  – установка пиксельной длины короткого пакета;  – превышение максимальной длины (0хFFFF байт).  Методы исправления: проверить размер кадра. |
| CSI\_CLKSM\_ERR | Ошибка конечного автомата линии CLK монитора.  Причина – некорректное поведение передатчика D-PHY. |
| CSI\_DATASM\_ERR | Ошибка конечного автомата линии DATA монитора.  Причина – некорректное поведение передатчика D-PHY. |
| CSI\_EXT\_UNKCMD | Получение неизвестной команды Escape Mode. |
| CSI\_ECC\_ERR | Обнаружение неисправимой ошибки ECC пакета.  Причина – нарушение целостности полученного пакета. |
| CSI\_QSIZE | Ошибка в восстановлении длины пакета.  Причина – несовпадение фактической длины пакета с указанной. |

Продолжение таблицы 3.

| **Тип сообщения** | **Описание** |
| --- | --- |
| CSI\_FLIPPEDBIT | Ошибка в определении последнего бита.  Причина – размер пакета не кратен 8 битам, или после отправки пакета на шину не подаётся инвертированный последний бит данных. |
| CSI\_CRC | Ошибка при проверке CRC.  Причина – нарушение целостности поля данных пакета. |
| CSI\_BIGVCID | Номер виртуального канала полученного пакета превышает количество используемых каналов.  Методы исправления: увеличить количество каналов |
| CSI\_FRAME\_NUM\_ORDER | Неправильный порядок номеров полученных кадров |
| CSI\_FE\_NUM\_MISMATCH | Несовпадение номеров кадра в пакетах Frame Start и Frame End |
| CSI\_LS\_NUM\_MISMATCH | Несовпадение номера строки с ожидаемым.  Возможная причина: различная нумерация строк в кадрах одного потока |
| CSI\_LE\_NUM\_MISMATCH | Несовпадение номеров строки в пакетах Line Start и Line End |
| CSI\_LINE\_NUM\_ORDER | Неправильный порядок номеров полученных строк.  Возможные причины:  – обратный порядок номеров  – различная нумерация строк в кадрах одного потока  – разный размер полей одного кадра  – отсутствие фрагмента кадра |
| CSI\_LINEMM\_FORM | Несовпадение форматов строк внутри одного кадра |
| CSI\_LINEMM\_LEN | Несовпадение длин строк внутри одного кадра |
| CSI\_FRAME\_SIZE | Несовпадение размера кадра с настройками монитора |
| CSI\_FRAME\_FORMAT | Несовпадение формата данных кадра с настройками монитора |

Продолжение таблицы 3.

| **Тип сообщения** | **Описание** |
| --- | --- |
| CSI\_CFSE | Ошибки порядка полученных пакетов.  Возможные причины:  – данные вне пакетов Frame Start / Frame End  – данные вне пакетов Line Start / Line End  – пакеты Line Start / End вне пакетов Frame Start / End  – пакет Frame End до получения пакета Line End  – повторное получение пакетов Line / Frame Start без получения пакетов Line / Frame End |
| NOVIF | Отсутствие виртуального интерфейса.  Методы исправления: проверить подключение виртуальных интерфейсов. |
| CFGERR | Отсутствие указателя на конфигурационный класс агента.  Методы исправления: проверить наличие конфигурации. |
| NOCFG | Отсутствие конфигурационного класса агента.  Методы исправления: проверить наличие конфигурации. |
| **Предупреждения** | |
| CSI\_CALCLEN | Не удалось преобразовать число байт в пакете в целое число пикселей.  Возможные проблемы: неправильное декодирование полученной строки (в частности, последних пикселей).  Рекомендации: проверить соответствие размера кадра требованиям для выбранного формата данных. |
| CSI\_EXT\_UNKCMD | Получение неизвестной команды Escape Mode. |
| CSI\_ECC\_WRN | Обнаружение исправимой ошибки ECC пакета. |
| **Другие сообщения:** | |
| CSI\_PACK | Журнал генератора пакетов. |
| CSI\_MIX | Журнал миксера пакетов. |

Продолжение таблицы 3.

| **Тип сообщения** | **Описание** |
| --- | --- |
| CSI\_MLTRAN | Журнал генератора транзакций D-PHY из пакетов. |
| CSI\_EXT\_DRV | Журнал драйвера интерфейса CSI. |
| CSI\_CLKSM\_LOG | Журнал автомата линии CLK монитора. |
| CSI\_DATASM\_LOG | Журнал автомата линии DATA монитора. |
| CSI\_EXT\_MON | Журнал монитора интерфейса CSI. |
| CSI\_MLT2PKT | Журнал сборщика пакетов монитора. |
| CSI\_ECC | Успешная проверка ECC пакета. |
| CSI\_CRC | Успешная проверка CRC пакета. |
| CSI\_SORTER | Журнал сортировщика пакетов монитора. |
| CSI\_MON | Журнал монитора пакетов. |

Каждое сообщение содержит сведения, необходимые для исправления возникшей ошибки, иерархическое имя вызвавшего ошибку компонента, а также время возникновения ошибки.