DOI: 10.17725/rensit.2023.15.453

Архитектура драйвера контроллера взаимодействия с плоскопанельным дисплеем для ОС Linux

Пугин К.В., Мамросенко К.А., Гиацинтов А.М.
НИИ системных исследований РАН, Центр визуализации и спутниковых информационных технологий, https://niisi.ru/
Москва 117218, Российская Федерация
E-mail: rilian@niisi.ras.ru, mamrosenko_k@niisi.ras.ru, algts@niisi.ras.ru
Поступила 25.05.2023, рецензирована 01.06.2023, принята 08.06.2023, опубликована 06.12.2023.
Представлена действительным членом РАЕН И.Б. Петровым

Аннотация: **Задачей работы является создание архитектуры драйвера контроллера сопряжения с устройством отображения информации. Архитектура обеспечивает реализацию протоколов взаимодействия с плоскопанельным дисплеем в случае, когда контроллер имеет собственные регистры и систему конфигурирования. В отличие от известных решений предлагаемая архитектура позволяет сократить объемы изменения кода реализации в случае модернизации аппаратной части, а также не требует применения автоматической генерации кода драйверов на базе высокоуровневых описаний или разработки специализированных инструментов, таких как предметно-ориентированные языки программирования. Произведен анализ имеющихся в открытых источниках драйверов на базе подсистемы DRM (Digital Rights Management), а также ранее описанных подходов к разработке драйверов DTLC (display transmitter link controller). Также в работе представлена логическая модель компаратора для тестирования генераторов частоты, являющихся неотъемлемой частью всех устройств подсистемы вывода на экран. По данной модели был разработан IP-блок, применявшийся для тестирования драйвера DSI. Апробация результатов была выполнена при разработке драйвера MIPI DSI (Mobile Industry Processor Interface Display Serial Interface) для перспективного контроллера. Проведено тестирование данного драйвера совместно с прототипом устройства и панелью, поддерживающей стандарт MIPI DSI 1.3. Результаты данной работы могут быть использованы при проектировании новых драйверов контроллера сопряжения с устройством отображения информации для Unix-подобных систем как в случае разработки драйвера для готового контроллера, так и при параллельной разработке нового контроллера и драйвера для него.**

Ключевые слова: **драйвер; архитектура; MIPI DSI; встраиваемые системы**

УДК 004.454

Благодарности: Работа выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН "Проведение фундаментальных научных исследований (47 ГП)" по теме № FNEF-2022-0022 "Математическое обеспечение и инструментальные средства для моделирования, проектирования и разработки элементов сложных технических систем, программных комплексов и телекоммуникационных сетей в различных проблемно-ориентированных областях".

Для цитирования: Пугин К.В., Мамросенко К.А., Гиацинтов А.М. Архитектура драйвера контроллера взаимодействия с плоскопанельным дисплеем для ОС Linux. *РЭНСИТ: Радиоэлектроника. Наносистемы. Информационные технологии*, 2023, 15(3):453-462. DOI: 10.17725/rensit.2023.15.453.

Flat Panel Display Controller Driver Architecture for Linux OS

Konstantin V. Pugin, Kirill A. Mamrosenko, Alexander M. Giatsintov
Federal Scientific Center Scientific Research Institute of System Analysis of the RAS, Center of Visualization and Satellite Information Technologies, https://niisi.ru/
Moscow 117218, Russian Federation
E-mail: rilian@niisi.ras.ru, mamrosenko_k@niisi.ras.ru, algts@niisi.ras.ru

Received May 25, 2023, peer-reviewed June 01, 2023, accepted June 08, 2023, published December 06, 2023.

Abstract: **This paper discusses the development of a driver architecture for display transmitter link controller. The architecture ensures the implementation of protocols for interaction with flat panel displays in the case when the controller has its own registers and configuration system. Unlike the known solutions, the proposed architecture makes it possible to reduce the amount of changes in the implementation code in the event of hardware upgrade, and also does not require the use of automatic driver code generation based on high-level descriptions or the development of special tools, such as domain-specific languages. This paper analyses drivers that are based on Direct Rendering Management subsystem and available in open source, as well as previously described approaches to the development of display transmitter link controller drivers. The paper also presents a logical comparator model for testing phase-locked loop devices, which are an integral part of all display output stacks. Based on this model, an IP block was developed, which was used to test the Display Serial Interface driver. Evaluation of the results was carried out in the development of the MIPI Display Serial Interface driver for a promising controller. This driver was tested together with a device prototype and a panel that supports the MIPI Display Serial Interface 1.3 standard. The results provided in this paper can be used both to develop new drivers for existing controllers and new controllers with new drivers.**

Keywords: **driver, architecture, MIPI DSI, embedded systems**

UDC 004.454

Acknowledgments: The work was carried out within the framework of the state task of the Federal State Institution FNTs NIISI RAS "Conducting fundamental scientific research (47 GP)" on topic No. FNEF-2022-0022 "Mathematical support and tools for modeling, designing and developing elements of complex technical systems, software systems and telecommunication networks in various problem-oriented areas".

For citation: Konstantin V. Pugin, Kirill A. Mamrosenko, Alexander M. Giatsintov. Flat Panel Display Controller Driver Architecture for Linux OS. *RENSIT: Radioelectronics. Nanosystems. Information Technologies*, 2023, 15(3):453-462e. DOI: 10.17725/rensit.2023.15.453.

Содержание

1. **Введение (454)**
2. **Анализ предшествующих работ (456)**
3. **Методы исследования (457)**
   3.1. Подходы к разработке драйверов графических контроллеров на примере ОС Linux (457)
   3.2. Использованные методы исследования драйверов DTLC (457)
   3.3. Отличия архитектуры драйверов DTLC для взаимодействия с плоскопанельным дисплеем (458)
4. **Анализ протокола MIPI Display Serial Interface (458)**
   4.1 Особенности работы с дисплеями MIPI DSI (459)
5. **Логическая модель компаратора для тестирования драйверов генераторов частоты (460)**
6. **Заключение (461)**

Литература (461)

1. ВВЕДЕНИЕ

В настоящее время происходит быстрое развитие устройств графического вывода информации для мобильный устройств – многие из новых устройств имеют разрешение до 2560×1600 и динамические частоты обновления, варьирующиеся от 60 до 120 Гц. В связи с этим актуальным становится использование новых стандартов в ходе разработке контроллеров для взаимодействия с дисплеями, в которых имеется возможность передачи видеопотока с высокой скоростью. Для решения данной задачи часто применяются протоколы, требующие создания отдельных устройств – контроллеров сопряжения с устройством отображения информации (display transmitter link controller, далее DTLC), которые дополняют контроллер вывода на экран в части преобразования видеопотока

в требуемую дисплеем форму. Многие протоколы, передающие видеопоток с высокой скоростью (более 1 ГБит/с), требуют динамического взаимодействия обеих сторон и выработки соглашения по допустимым параметрам передачи. Контроллеры для таких протоколов требуют сложного программного управления, отличающегося от устройства к устройству. В протоколах с меньшим объемом передаваемых данных для плоскопанельных дисплеев такого сложного управления и обратной связи зачастую не требуется. Конфигурация плоскопанельного дисплея не изменяется в ходе работы с устройством, а её динамическое изменение не предусматривается. Вслед за [1] DTLC, требующие программного управления с обратной связью, назовем сложными (Complex DTLC). Поскольку число мобильных устройств с дисплеями, требующих новых способов взаимодействия и новых контроллеров, возрастает, то требуются подходы к разработке драйверов для таких (сложных) устройств [2]. Если драйвер DTLC для взаимодействия с плоскопанельными дисплеями разрабатывается одновременно с доработкой устройства, то возникает несколько проблем, часть из которых идентична проблемам, описанным в [1], и часть является уникальной, связанной с сущностью протоколов взаимодействия с плоскопанельными дисплеями:

1. Необходимость подбора панелей и частот под используемую для разработки устройств ПЛИС (программируемая логическая интегральная схема).
2. Возможная несовместимость допустимых частот панелей и контроллеров, для обхода которой требуются дополнительные аппаратные и программные средства.
3. Необходимость тестирования нескольких частот и режимов работы панели при отсутствии возможности замены аппаратных частей устройства.
4. Необходимость обхода интерфейсов ОС, если они не совместимы с тем вариантом протокола, который реализован аппаратно.

Для решения данных проблем применяются различные аппаратные и программные подходы: использование верификаторов [3], создание предметно-ориентированных языков (Domain Specific Languages, DSL) [4], автоматическая генерация кода драйверов на основе общего шаблона [5], а также написание драйверов, определяющих неизменные черты серии устройств до конца ее производства, в то время как остальные параметры динамически применяются на основе конфигурационных файлов. В данной статье мы пытаемся ответить на вопрос о том, какие подходы к разработке драйвера DTLC для взаимодействия с плоскопанельными дисплеями необходимо применять, учитывая заданные условия. Вклад исследования состоит в следующем:

- Созданная авторами ранее ([1]) архитектура драйверов DTLC была доработана для случаев, когда DTLC используется для взаимодействия с плоскопанельным дисплеем.
- Для тестирования корректности программирования и корректности интерфейса взаимодействия с драйвером генераторов частоты (далее – ГЧ) на ранних стадиях разработки IP-блока была создана и реализована на ПЛИС логическая модель компаратора, которая применяется совместно с встроенными в них генераторами частоты. Она позволила протестировать корректность программирования генераторов частоты для DTLC в схожих с реальным применением устройства состояниях ПЛИС.

Разработанная архитектура обеспечивает реализацию протоколов взаимодействия с плоскопанельным дисплеем в случае, когда контроллер имеет собственные регистры и систему конфигурирования. В отличие от известных решений предлагаемая архитектура позволяет сократить объемы изменения кода реализации в случае модернизации аппаратной части, а также не требует применения автоматической генерации кода драйверов на базе высокоуровневых описаний или разработки специализированных инструментов, таких как предметно-ориентированные языки программирования. В п. 3.1 описаны подходы разработки драйверов для ОС Linux, на базе которых производилось исследование. В п. 3.3 описана архитектура драйвера DTLC для взаимодействия с плоскопанельными дисплеями, содержащая два слабосвязных компонента – компонент аппаратной привязки и компонент взаимодействия с ОС, которые связаны внутренним API. Компонент аппаратной привязки производит большинство необходимых преобразований приходящих данных в те форматы, которые требуются для программирования конкретного устройства, программирует его регистры и производит обратную связь. Компонент взаимодействия с ОС преобразует запросы, приходящие из других частей системы и передает их при помощи внутреннего API – системы функций и структур, общих для всех модулей аппаратной привязки. В части 4 приведено описание применения данной архитектуры при разработке драйвера для Mobile Industry Processor Interface Display Serial Interface (MIPI DSI) – одного из протоколов DTLC для взаимодействия с плоскопанельными дисплеями. В части 5 описана логическая модель компаратора для тестирования драйвера на ПЛИС в части работы с частотами, которая позволяет тестировать корректность программирования драйверов ГЧ даже при необходимости уменьшения объема проекта для ПЛИС.

2. АНАЛИЗ ПРЕДШЕСТВУЮЩИХ РАБОТ

В некоторых новых работах [6] предлагается создавать системы с несколькими DTLC для взаимодействия с плоскопанельным дисплеем, имеющие общий физический уровень интерфейса (physical interface layer, далее PHY), который может в один момент времени использоваться только одним DTLC. Данное ограничение можно попробовать обойти при помощи архитектуры драйвера DTLC, описанной в работе [7], только разделяемым устройством в данной архитектуре будет являться не синтезатор частоты, а PHY, который будет иметь архитектуру, описанную в работе [6]. Либо возможно сохранить ограничение на возможность одновременной работы, но выделить синтезатор частоты (СЧ) совместно с PHY в отдельный модуль с общей частью драйвера. Поскольку описанный в [6] PHY используется в DTLC для взаимодействия с плоскопанельным дисплеем, то описанная в данной работе архитектура драйвера также может использоваться и для контроллеров с таким PHY (с учетом работы [7]).

В статье [8] описаны подходы к проектированию драйверов DTLC для операционных систем реального времени. Описание подсистемы ОСРВ в данной статье позволяет применять те же модели для разработки драйверов, что и для DRM (Digital Rendering Management) в Linux, с минимальной доработкой в части замены названий компонентов. Это позволяет вести разработку DRM-совместимых моделей для драйверов DTLC для взаимодействия с плоскопанельным дисплеем не только для Linux, но и для других ОС, о чем также сообщается в статье [9]. Эта работа приводит данные, что для операционных систем семейства FreeBSD также применяется DRM

в части написания графических драйверов, поэтому возможно применять разработанные для DRM модели и для написания драйверов DTLC в данных ОС.

3. МЕТОДЫ ИССЛЕДОВАНИЯ

3.1. Подходы к разработке драйверов графических контроллеров на примере ОС Linux

Известно несколько подходов, связанных с использованием тех или иных известных инструментов графической подсистемы, применяемых при разработке драйверов DTLC под Linux. Рассмотрим наиболее популярные из них:

1. DRM. Как было сказано в [10], интерфейс взаимодействия с DTLC является частью инфраструктуры Direct Rendering Management (DRM). Если рассматривать подсистему DRM как программную модель реального устройства вывода, то все плоскопанельные дисплеи и интерфейсы их взаимодействия с DTLC соотносятся с типом panel, т.к. определяют способ работы графического вывода с фиксированными частотами. Однако, реализация типа panel требует, чтобы для каждого типа панели существовал свой драйвер даже при совпадении контроллеров, поэтому чаще всего драйвера DTLC для фиксированных панелей реализуют в виде связки bridge + panel, либо, что гораздо реже, encoder + panel. Второй вариант, в отличие от варианта с использованием bridge, не соотносится с внешней моделью, хоть и возможен практически [11].
2. User Mode Setting (UMS), или реализация выставления режимов и взаимодействия с hardware внутри X – сервера. Проблемы в реализации и использовании UMS были выявлены в 2006 году [12], после чего разработка драйверов для DTLC данным способом фактически не производилась.
3. FrameBuffer device (fbdev). Данный интерфейс ядра предшествовал KMS (Key Management Service) модулю DRM и был наиболее распространен до 2008-2009 года, когда была принята в ядро первая версия KMS. Интерфейс fbdev не предусматривал реализацию настройки режимов в ядре, и потому требовал реализации UMS, что приводило к указанным выше проблемам. Разработка драйверов именно для устройств DTLC после появления KMS данным способом не производилась.
4. Реализации драйверов под Android Display Framework (ADF) существуют, но обладают следующими недостатками: стандартные, независимые от аппаратной реализации, функции протокола необходимо разрабатывать с нуля в каждом драйвере, что повышает вероятность появления ошибок и увеличивает трудоемкость задачи. Также вследствие архитектуры ADF (она является развитием fbdev) возникает необходимость совмещения драйверов DTLC и контроллера вывода изображения на экран. Реализации драйверов контроллеров в ряде других Unix-совместимых систем (к примеру, FreeBSD) также происходят с использованием DRM ([9]), как наиболее развитой открытой системы для реализации сложных инструментов вывода на экран.

3.2. Использованные методы исследования драйверов DTLC

В ходе работы был проведен анализ существующих открытых драйверов DTLC для ОС Linux, в ходе которого особый акцент уделялся драйверам для мобильных платформ. На базе данного анализа была выработана собственная архитектура драйвера Display Serial Interface (DSI), которая позволяет быстро создать драйвер для прототипа устройства и после выпуска

аппаратной части перенести данный драйвер на серийное устройство. В ходе предшествующего анализа и тематического исследования, а также с использованием материалов из [1], архитектура была расширена на все драйверы Complex DTLC для взаимодействия с плоскопанельным дисплеем.

3.3. Отличия архитектуры драйверов DTLC для взаимодействия с плоскопанельным дисплеем

Для DTLC с динамической панелью наиболее применимой на сегодняшний момент предлагается считать архитектуру из [1], представленную на **Рис. 1**.

Для адаптации данной архитектуры для применения в системах с плоскопанельным дисплеем необходимо использовать структуру drm_panel, которая будет работать только с интерфейсами модели DRM, и драйвер панели должен быть полностью совместим с моделью DRM. Доработанная архитектура представлена на **Рис. 2**.

В данном варианте архитектура драйвера также сохраняет все преимущества, описанные в [1], при этом приобретая возможность работать с плоскопанельным дисплеем в тех случаях, когда реализация обратной связи в панели соответствует соответствующим пунктам протокола. В случаях, когда для

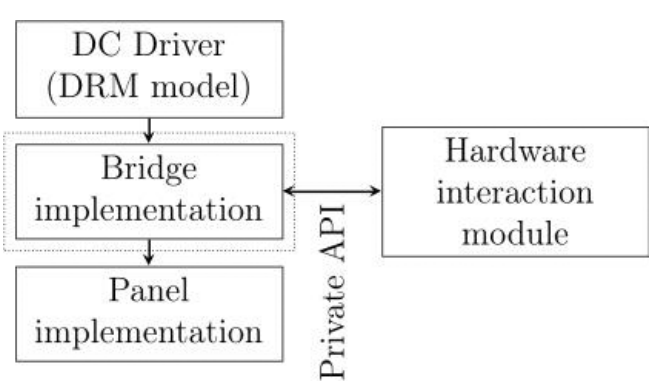

Рис. 1. *Архитектура драйвера DTLC для реализации в рамках подсистемы DRM.*

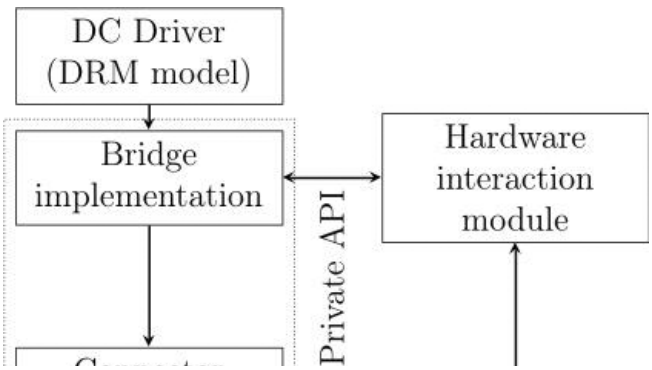

Рис. 2. *Архитектура драйвера DTLC для взаимодействия с плоскопанельными дисплеями для реализации в рамках подсистемы DRM.*

взаимодействия с интерфейсом панели требуются нестандартные расширения, данная архитектура требует доработок.

4. АНАЛИЗ ПРОТОКОЛА MIPI DISPLAY SERIAL INTERFACE

Одним из самых используемых протоколов для передачи данных мониторам является, наряду с eDP (embedded DisplayPort – встраиваемым вариантом DisplayPort, разработка драйверов которого была описана в [1]) MIPI DSI, применяемый ведущими производителя мобильных устройств на Android. Данный протокол является протоколом с обратной связью, которая осуществляется посредством Display Command Set (DCS) – части протокола MIPI DSI, определяющего формат команд и ответов участников взаимодействия, а также перечень стандартных команд, которые должны поддерживаться всеми совместимыми устройствами. Для реализации протокола MIPI DSI в DRM применяется тип drm_encoder (для реализации драйвера контроллера), drm_panel (для реализации драйвера панели) и drm_connector либо drm_bridge (для реализации взаимодействия панели с контроллером) [11]. В ядре Linux для драйверов MIPI DSI на базе DRM существует программная модель, которая дополняет общую модель

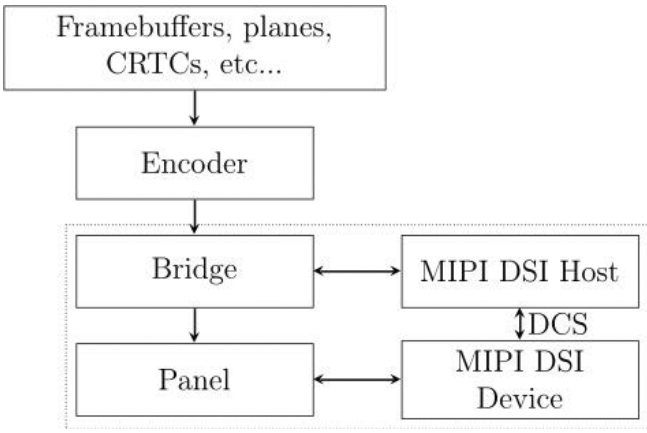

Рис. 3. *Программная модель MIPI DSI в рамках общей модели DRM.*

DRM. Дополненная модель представлена на **Рис. 3**.

MIPI DSI, как и схожий с ним стандарт eDP, реализует в себе не только базовый вывод изображения, но и дополнительные функции (к примеру, передачу различных данных по протоколу DCS). Для реализации данных функций со стороны контроллера MIPI DSI в DRM существует объект MIPI DSI Host (mipi_dsi_host), а для реализации со стороны панели – объект MIPI DSI Device (mipi_dsi_device), а также несколько вспомогательных функций только для работы с контроллерами и панелями MIPI DSI.

4.1. Особенности работы с дисплеями MIPI DSI

Подавляющее большинство PHY для MIPI DSI (далее D-PHY) [13] либо используют пакетный режим, либо требуют учета определенных констант для протокола MIPI DSI. Это приводит к тому, что, в отличие от других DTLC, в MIPI DSI для всех режимов необходимо пересчитывать временные и частотные характеристики. Также, в отличие от пакетного протокола DisplayPort, частота передачи данных в DSI не является постоянной, а выводится из пиксельной частоты по формуле

$$clk_{hs} = clk_{pix} \cdot BPP \cdot lanes,$$

где *lanes* – число активных линий передачи (от 1 до 4), BPP – число битов в пикселе, а clk_{pix} – пиксельная частота. В ГЧ контроллера DSI необходимо передавать именно битовую частоту [14]. В протоколе DSI используется определение разрешения экрана по стандарту Display Monitor Timings (DMT, см. [15]), но, в отличие от остальных протоколов, для DSI необходимо пересчитывать все характеристики горизонтальных линий для того, чтобы невидимая часть экрана могла вместить заголовки пакетов. Обозначим горизонтальные характеристики как: *HBP* – Front Porch, *HFP* – Back Porch, *HSA* – Hsync Active, *HACT* – H Active, *PULSE_CLK* – значение синхронизации импульса, *BPP* – число битов в пикселе. Для каждой характеристики (обозначим как X) примем X_{DPI} – входящая характеристика X, X_{DSI} – пересчитанная характеристика X. Также для пересчета важны константы протокола, такие как $HDR = 6$ (заголовок пакета, по одному на каждый тайминг, должен быть в невидимой области), $HSS = 4$ (заголовок начала горизонтальной синхронизации), $HSE = 4$ (заголовок конца горизонтальной синхронизации). Для пересчета таймингов применяются следующие формулы:

$$HBP_{DSI} = (HBP_{DPI} \cdot BPP/8) - HDR_{HBP}$$

$$HFP_{DSI} = (HFP_{DPI} \cdot BPP/8) - HDR_{HACT} - HDR_{HFP}$$

$$HSA_{DSI} = (HSA_{DPI} \cdot BPP/8) - HSS - HDR_{HSA} - HSE$$

$$HACT_{DSI} = HACT_{DPI} \cdot BPP/8$$

$$PULSE\_CLK = ((HACT_{DPI} + HSA_{DPI} + HBP_{DPI} + HFP_{DSI}) \cdot BPP/8) - HSA_{DSI} - 20.$$

Результатами данных расчетов всегда должны являться целые числа. Если число получается нецелым, то подобрать альтернативу – задача драйвера. Поскольку в общем стеке DRM средства для этого отсутствуют, то эта задача переадресуется драйверу контроллера. Разные модели контроллеров могут иметь различные дополнительные ограничения, такие как невозможность выбрать некоторые битовые частоты или необходимость выбора специфических вертикальных таймингов.

Для реализации возможности определения дополнительных ограничений значений контроллера используется модуль для аппаратного взаимодействия, изначально предложенный в [6], но с доработкой архитектуры для фиксированной панели (3). Также возможно уменьшение Vertical Front Porch (*VFP*, см. [15]) для уменьшения потребления энергии при работе в высокоскоростном режиме, которое производится в зависимости от модели контроллера и панели, поэтому работа с уменьшением *VFP* производится в модуле аппаратного взаимодействия. Для тестирования корректности программирования генератора частоты передачи использовался компаратор тестирования (5), который позволил избежать ошибок некорректности программирования ГЧ при тестировании драйвера DTLC для контроллера MIPI DSI.

5. ЛОГИЧЕСКАЯ МОДЕЛЬ КОМПАРАТОРА ДЛЯ ТЕСТИРОВАНИЯ ДРАЙВЕРОВ ГЕНЕРАТОРОВ ЧАСТОТЫ

Для большинства DTLC драйверу необходимо корректно программировать частоты генераторов частоты, непосредственно связанных с выводом изображения. В некоторых контроллерах интерфейс программирования данных генераторов может быть встроен непосредственно в интерфейс программирования контроллера и выполняться теми же способами, что и все остальные операции настройки. Несмотря на разработку уменьшенных генераторов частоты [16], на ПЛИС и эмуляторах зачастую отсутствует возможность динамической настройки генераторов частоты, что сильно затрудняет тестирование контроллеров DTLC с возможностью смены подключенного монитора без отключения контроллера, например, DisplayPort или HDMI. Для каждого значения частоты требуется переразводка проекта для ПЛИС. Для тех контроллеров DTLC, где панель фиксирована, такая проблема присутствует в гораздо меньшей степени, поскольку панели имеют малый диапазон частот, на которых возможен вывод изображения. На таких системах зачастую проявляется другая проблема драйверов на ПЛИС - программируемое значение драйверов ГЧ не учитывается в работе ПЛИС, что позволяет контроллерам на ПЛИС при использовании драйверов с некорректным программированием ГЧ показывать изображение. При этом некорректное программирование проявляется только на готовых устройствах. Для решения данной проблемы нами был предложен блок тестирования для реализации на ПЛИС, логическая модель которого состоит из следующих элементов (**Рис. 4**):

- Вход, на который поступает частота, заданная при разводке ПЛИС.
- Блок расчетов, который преобразует входные данные в значение частоты (округленное до целого числа). Точное устройство блока расчетов определяется интерфейсом программирования ГЧ, либо той частью интерфейса программирования контроллера, который программирует ГЧ.
- Компаратор, который сравнивает вывод блока расчетов с частотой, заданной при разводке ПЛИС. При положительном результате сравнения компаратор посылает сигнал включения на переключатель пропуска частоты. Допустимое отклонение расчетной

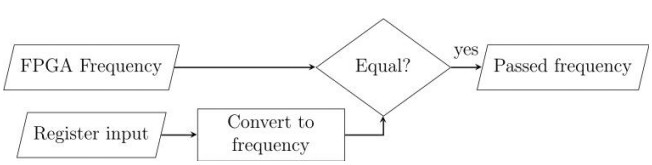

Рис. 4 . *Модель компаратора.*

частоты от входящей зависит от разрабатываемого устройства (для DTLC оно зачастую не превышает 1 КГц).

- Переключатель пропуска частоты, который включается по сигналу от компаратора.

Данная модель была реализована в виде IP-блока на ПЛИС. Применение данной модели для тестирования драйверов DTLC для взаимодействия с плоскопанельным дисплеем позволило добиться корректного программирования ГЧ для контроллера DTLC еще до изготовления первого прототипа. Корректность программирования была подтверждена уже на этапе использования инструментов тестирования, таких как эмуляторы или ПЛИС.

6. ЗАКЛЮЧЕНИЕ

Описанные в статье подходы позволяют сократить объемы изменения кода реализации в случае модернизации аппаратной части. Также данные подходы позволяют снизить число модификаций драйвера для поддержки семейств устройств (в том числе и для одного устройства с различиями в разных блоках, например, разных ГЧ). Модель генератора частоты, описанная в статье, при реализации позволит уже на ранних этапах разработки генераторов частоты проводить тестирование их драйверов, а также драйверов устройств, использующих генераторы. Дальнейшими направлениями являются: определение применимости полученной архитектуры к разработке драйверов DTLC для встраиваемых систем с внешней шиной (например, использование DTLC на PCI Express), исследования по конкретным протоколам DTLC (по HDMI), а также определение возможностей применения компаратора для тестирования драйверов генераторов частоты, используемых в других частях систем на кристалле.

ЛИТЕРАТУРА

1. Pugin KV, Mamrosenko KA, Reshetnikov VN. Display Transmitter Link Controller Design Technology for Linux OS. *Software Journal:Theory and Applications*, 2019, 4:10-17, doi: 10.15827/2311-6749.33.406.
2. Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media, Inc., 2005. ISBN: 0-596-00590-3.
3. Dileep KP, Raghavendra A, Suman M, Devesh G, Srikanth SV. Rules Based Automatic Linux Device Driver Verifier and Code Assistance. *Proc. IEEE International Conference on Recent Advances and Innovations in Engineering*. Jaipur, India: IEEE, 2014. ISBN: 978-1-4799-4040-0.
4. Lisboa EB, Silva L, Lima T, Chaves I, Barros E. An Approach to Concurrent Development of Device Drivers and Device Controller. *Proc. 11th International Conference on Advanced Communication Technology, pp.* 571-575. Phoenix Park: IEEE, 2009. ISBN: 978-89-5519-139-4.
5. Jung Choon Park, Yong Hoon Choi, Tae Ho Kim. Domain Specific Code Generation For Linux Device Driver. *Proc. 10th International Conference Advanced Communication Technology, pp.* 101-104. Gangwon-Do, Korea (South): IEEE, Feb, 2008. ISBN: 978-89-5519-136-3. DOI: 10.1109/ICACT.2008.4493721.12.
6. Sunil Kumar CR, Aruna Kumar, Sanjib Basu. Novel Circuit Architecture for Configurable eDP and MIPI DPHY IO. *Proc. 2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems* (*VLSID*), pp. 98-101. Bangalore, India: IEEE, 2022. DOI: 10.1109/VLSID2022.2022.00030.
7. Konstantin V. Pugin, Kirill A. Mamrosenko, Alexander M. Giatsintov. Software Architecture for Display Controller and

Operating System Interaction. *RENSIT: Radioelektronika. Nanosistemy. Informacionnye Tehnologii*, 2021, 13(1):87-94. DOI: 10.17725/rensit.2021.13.087.

8. Bazhenov PS, Giatsintov AM, Mamrosenko KA. Approaches to providing data visualization on devices using modern real time operating systems. *Software & Systems,* 2021, 3:433-439. DOI: 10.15827/0236-235X.135.433-439.
9. Emmanuel Vadot. Adventure in DRMland Or How to Write a FreeBSD ARM64 DRM Driver. *Proceedings AsiaBSDCon*, 2019, pp. 9-13, (AsiaBSDCon. Tokyo, Japan: BSD Research, 2019). DOI: 10.25263/asiabsdcon2019/p01a.
10. Konstantin V. Pugin, Kirill A. Mamrosenko, Alexander M. Giatsintov. Visualization of Graphic Information in General-Purpose Operating Systems. *RENSIT: Radioelektronika. Nanosistemy. Informacionnye Tehnologii*, 2019, 11(2):217-224e. DOI: 10.17725/rensit.2019.11.217.
11. Linux GPU Driver Developer's Guide. 2019. URL: https://dri.freedesktop.org/docs/drm/gpu/index.html (Access mode: 06.03.2019).
12. Verhaegen Luc. X and Modesetting: Atrophy Illustrated. 2006. URL: https://people.freedesktop.org/~libv/X_and_Modesetting_-_Atrophy_illustrated_(paper).pdf.
13. Kiyong Kwon, Dongwon Kang, Geon-Woo Ko, Seok-Young Kim, Seon-WookKim. Low-Cost Unified Pixel Converter from the MIPI DSI Packets intoArbitrary Pixel Sizes. *Electronics,* 2022, 11(8):1221. DOI:10.3390/electronics11081221.
14. Yeming Liu, Chengyue He. A Design of MIPI DSI Interface for LCD Display Driver. *Journal of Physics: Conference Series,* 2022, 2221(1):012015. DOI: 10.1088/1742-6596/2221/1/012015.
15. VESA and Industry Standards and Guidelines for Computer Display Monitor Timing (DMT), Version 1.0, Rev. 13. 39899 *Video Electronics Standards Association,* 2013, 105 p.
16. Hye-Hyun Lee, Yeon-Seob Song, Kang-Yoon Lee. Modeling of Nano-Scale PLL Using Verilog HDL. *Proc. 13th International Conference on Information and Communication Technology Convergence* (ICTC). IEEE, 2022, 2101-2104. DOI: 10.1109/ICTC55196.2022.9952654.

Пугин Константин Витальевич
программист
НИИ Системных исследований РАН
36/1, Нахимовский просп., Москва 117218, Россия
E-mail: rilian@niisi.ras.ru

Мамросенко Кирилл Анатольевич
к.т.н., руководитель Центра
НИИ Системных исследований РАН
36/1, Нахимовский просп., Москва 117218, Россия
E-mail: mamrosenko_k@niisi.ras.ru

Гиацинтов Александр Михайлович
к.т.н., старший научный сотрудник
НИИ Системных исследований РАН
36/1, Нахимовский просп., Москва 117218, Россия
E-mail: algts@niisi.ras.ru.