**Содержание**

Часть 1. Описание устройства и принципов его работы

1.1 Обоснование выбора темы

1.2 Краткое описание и возможные варианты исполнения

1.2.1 Подключение микросхемы ЦАП AD1851

1.2.1.1 Способ 1. Прерывание тактового сигнала

1.2.1.2 Способ 2. Использование регистра сдвига

1.2.1.3 Способ 3. Регистр сдвига для подключения по шине I2S

1.2.2 Подключение микросхемы ЦАП AD1852 (AD1853)

Часть 2. Схемная реализация

2.1 Определение требований к применяемым цифровым микросхемам

2.2 Схемы электрические принципиальные

2.2.1 Схема устройства разделения потока данных для подключения микросхемы ЦАП AD1851

2.2.2 Схема устройства разделения потока данных для монофонического подключения микросхемы ЦАП AD1852 (AD1853)

2.2.3 Общие замечания к схемам

Часть 3. Временные диаграммы

3.1 Временная диаграмма работы устройства разделения данных для микросхемы ЦАП AD1851

3.2 Временная диаграмма работы устройства разделения данных для микросхемы ЦАП AD1852

Часть 4. Реализация схемы в проекте VHDL

Выводы

# Часть 1. Описание устройства и принципов его работы

## 

## 1.1 Обоснование выбора темы

В последнее время техника и технология стремительно развиваются. Эта область также не обходит и электронику. Каждый день выходят новые микросхемы и пополняется элементная база. Однако также в промышленности прослеживается тенденция упрощения и удешевления производства, что приводит к массовому выпуску дешевой, красочной и далеко не самого высокого качества (если не сказать весьма низкого) аппаратуры. Эти тенденции также затрагивают и область аудиотехники.

К сожалению, человеку, желающему качественно слушать любимые записи, сейчас приходится либо смириться с вышесказанным, либо платить достаточно большие суммы за качественную аппаратуру. Есть правда и еще один вариант: эту аппаратуру можно собрать своими руками на современной элементной базе.

Одной из наиболее значимых частей аудио тракта (с учетом распространения цифрового аудио) бесспорно является ЦАП. От качества этого модуля напрямую будет зависеть качество получаемого звукового сигнала. При этом для достижения желаемых результатов часто приходится использовать микросхемы с отличающимися протоколами обмена информацией. Таким образом, встает вопрос о преобразовании цифровых форматов данных.

Наиболее часто встречается ситуация, когда приемник цифрового сигнала способен выдавать данные в формате IIS (наиболее универсальный и полнофункциональный формат), а микросхема ЦАП способна принимать данные в формате Right Justifited. Также следует учесть то, что приемник сигнала выдает данные обоих стерео каналов по одной шине в то время как наиболее качественные микросхемы ЦАП являются монофоническими. Именно вопросу преобразования цифрового потока из формата IIS в Right Justifited с одновременным разделением на каналы и посвящена половина данной курсовой работы. Вторая половина посвящена разделению каналов в случае использования монофонического включения стерео-ЦАП.

## 

## 1.2 Краткое описание и возможные варианты исполнения

В рамках данной курсовой работы предполагается использовать микросхемы ЦАП AD1851 (16bit mono) и AD1852 (24bit 192kHz stereo) производства Analog Devices.

### 

### 1.2.1 Подключение микросхемы ЦАП AD1851

Рассмотрим цифровой интерфейс AD1851:

Схематически внутреннее устройство микросхемы можно представить следующим образом:

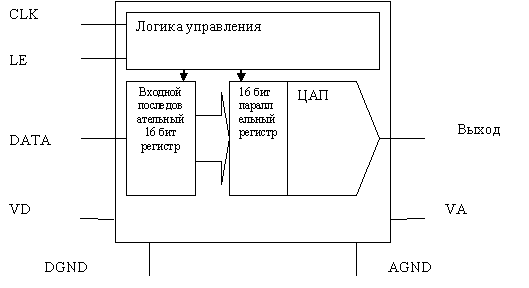


Рис. 1. – Внутренняя структура AD1851

Согласно документации микросхема имеет три цифровых входа: CLK, DATA и LE. Данные поступают во входной последовательный регистр сдвига через вход DATA и тактируются по переднему фронту сигнала CLK. Емкость входного регистра составляет 16 бит. Когда во входной регистр полностью загружены данные, по заднему фронту сигнала LE информация переносится в параллельный 16 битный регистр непосредственно модуля ЦАП, при этом выходная величина (микросхема способна работать как в варианте с токовым выходом, так и в варианте с выходом по напряжению) обновляется и удерживается до следующего перепада сигнала LE от высокого уровня к низкому. Временная диаграмма необходимых входных сигналов представлена на рисунке 2:

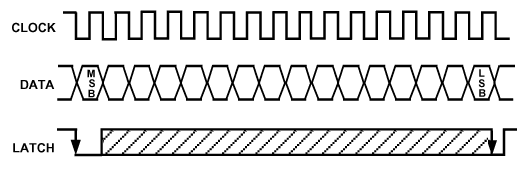


Рис. 2. – Временная диаграмма входных сигналов AD1851

Данный формат сигналов способны выдавать практически все цифровые фильтры. В большинстве случаев они уже имеют входы предустановки под микросхемы ЦАП различной разрядности и отдельные выводы данных (DOR, DOL) для микросхем правого и левого каналов.

Тем не менее, при желании построить ЦАП без передискретизации (достаточно нашумевшее направление в области современного высококачественного аудио) от их применения приходится отказаться. При этом микросхемы ЦАП подключаются через согласующую логику к микросхеме приемника цифровых данных.

Наиболее простым для работы выходным форматом данных цифровых приемников является формат Right Justifited:

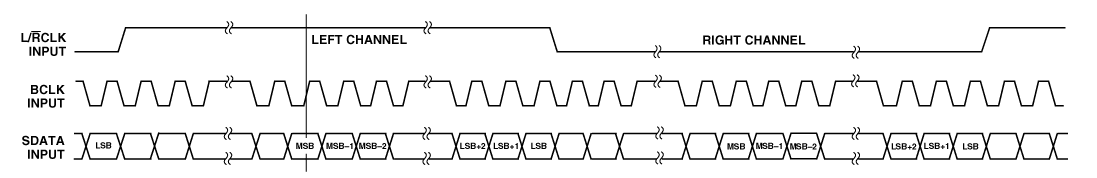


Рис. 3. – формат передачи данных Right Justifited

В этом формате данные левого и правого каналов передаются последовательно по одному проводнику SDATA, данные какого именно канала передаются в данный момент, определяет состояние сигнала LRCLK, тактовые импульсы передаются посредством сигнала SCLK. Первыми передаются данные левого канала (LRCLK=’1’), затем правого (LRCLK=’0’). Данные поступают в формате MSB First, и выровнены относительно перепада сигнала LRCLK по правому (заднему) фронту.

Для правильного преобразования данных необходимо загрузить данные каждого канала в свою микросхему ЦАП и одновременно подать сигнал обновления LE. В противном случае мы получим сдвиг по фазе между аналоговыми сигналами правого и левого каналов, зависящий от частоты сигнала.

Классический случай описан Паулем Шкритеком в "Справочном руководстве по звуковой схемотехнике", издательство "Мир", глава 17.7.1 "Преобразователи Ц/А в мультиплексном режиме". Цитата: "Если данные на ЦАП двух каналов поступают со сдвигом во времени на dT, то между каналами возникает сдвиг по фазе, зависящей от частоты: dF=360\*f\*dT". И далее: "В случае, когда оба канала обслуживает один ЦАП, работающий с временным уплотнением, временной сдвиг в нормальных условиях составляет dT=T/2=11.3 мксек, что вызывает фазовый сдвиг между сигналами левого и правого канала dF=82 градуса на частоте 20кГц".

Существует несколько способов сделать необходимое соединение.

### 

### 1.2.1.1 Способ 1. Прерывание тактового сигнала

Как уже известно, данные в микросхеме ЦАП сначала попадают во входной последовательный регистр, тактируемые сигналом CLK. В формате Right Justifited данные передаются выровненными по правому фронту сигнала LRCLK, данные левого канала первыми, данные правого канала последующими. Выход микросхем ЦАП необходимо обновить по загрузке данных обоих каналов, т.е. по нарастающему фронту сигнала LRCLK. При отсутствии сигнала CLK данные во входной регистр ЦАП грузиться не будут. При загрузке в регистр свыше 16 бит данных, во входном регистре остаются лишь последние 16 бит информации. Таким образом, необходимо загрузить данные левого канала в одну микросхему ЦАП, затем прервать поступление тактовых импульсов на нее и дождаться загрузки данных правого канала во вторую микросхему. Сигналом обновления при этом будет инвертированный сигнал LRCLK. Он же будет управляющим для прерывателя тактовых импульсов. Входы данных обоих микросхем ЦАП при таком включении параллелятся.

Данную операцию можно произвести с помощью всего одной логической микросхемы:



Рис. 4. – формирование сигналов методом прерывания тактовых импульсов

К сожалению, данный метод не является идеальным, поскольку некоторые микросхемы ЦАП отказываются работать в таком включении. Например, Алекс Петровский на своем сайте писал, что в таком включении отказались работать микросхемы PCM1704 производства Burr-Brown и AD1856 производства Analog Devices.Это происходит в основном из-за требования непрерывности тактового сигнала или из-за повышенной чувствительности логических цепей микросхемы к временным соотношениям между сигналами.

### 

### 1.2.1.2 Способ 2. Использование регистра сдвига

Для использования этого метода также удобно воспользоваться выходным форматом Right Justifited приемника цифровых сигналов. При этом необходимо отметить, что стандартной длиной последовательности данных за один отсчет является 64 бита. При этом на каждый канал приходится по 32 тактовых импульса. Первые 16 импульсов при этом не используются, по оставшимся 16 передаются данные. При этом первый бит данных правого канала передается через 32 тактовых импульса после первого бита данных левого канала. Таким образом, если задержать всю последовательность данных на 32 тактовых импульса, то на выходе схемы задержки будут данные левого канала в тот момент, когда в исходном сигнале будут передаваться данные правого канала. В качестве такой схемы очень удобно использовать 32 разрядный сдвиговый регистр. Обновление выхода микросхем ЦАП происходит по нарастающему фронту сигнала LRCLK. Схематически это можно изобразить следующим образом:



Рис. 5. – использование 32 разрядного сдвигового регистра для разделения цифрового потока

### 

### 1.2.1.3 Способ 3. Регистр сдвига для подключения по шине I2S

В некоторых случаях необходимо принимать данные не в формате Right Justifited, а в формате I2S. Такая необходимость возникает при построении ЦАП не в виде отдельного модуля, а как части готового проигрывателя компакт-дисков или какого-либо другого цифрового аудио устройства. Помимо того данный формат передачи обязательно поддерживается цифровыми приемниками и интерфейсными микросхемами. Таким образом, один раз грамотно спроектированное решение для подключения непосредственно на шину I2S будет не только наиболее правильным, а и наиболее универсальным. Устройство на основе такого решения уже целесообразно выполнить в виде ПЛМ и в дальнейшем использовать как некий "черный ящик" выполняющий все необходимые процедуры.

Рассмотрим формат передачи данных I2S:

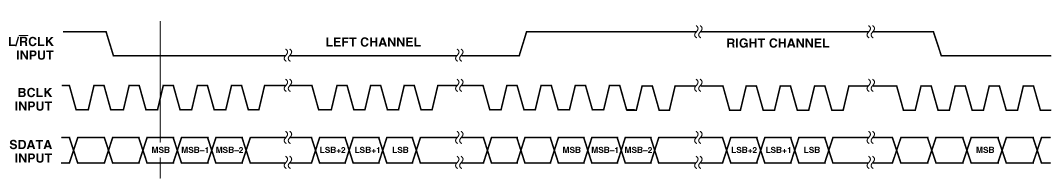


Рис. 6. – формат передачи данных I2S

В этом формате данные левого и правого каналов передаются последовательно по одному проводнику SDATA, данные какого именно канала передаются в данный момент, определяет состояние сигнала LRCLK, тактовые импульсы передаются посредством сигнала SCLK. Первыми передаются данные левого канала (LRCLK=’0’), затем правого (LRCLK=’1’). Данные поступают в формате MSB First, и выровнены относительно перепада сигнала LRCLK по левому (переднему) фронту и задержаны на 1 тактовый импульс.

Легко заметить, что данный формат отличается от Right Justifited лишь выравниванием последовательности данных и инвертированным состоянием сигнала LRCLK. Таким образом, для преобразования формата I2S к формату Right Justifited достаточно задержать всю последовательность данных на (32-[разрядность ЦАП]-1) тактовых импульса и инвертировать сигнал, определяющий данные какого канала передаются в настоящий момент. Далее можно использовать уже имеющийся вариант со сдвиговым регистром.

С учетом того, что минимальная разрядность современных микросхем аудио ЦАП составляет 16 бит, использование 16 битного сдвигового регистра позволяет выбором перемычки осуществлять подключение ЦАП любой разрядности (от 16 до 24 бит) к 16 битному источнику данных без потери качества. Также возможно подключение источника сигнала большей разрядности к микросхеме ЦАП меньшей разрядности с потерей только младших бит данных, что в некоторых случаях также может оказаться удобным, поскольку дает совместимость с новыми форматами.

Структурная схема такого преобразователя форматов данных представлена на следующем рисунке:



Рис. 7. – универсальный сдвиговый регистр

### 

### 1.2.2 Подключение микросхемы ЦАП AD1852 (AD1853)

Микросхема ЦАП AD1852 представляет собой стереофонический дельта-сигма ЦАП с встроенным цифровым фильтром и выходом по напряжению. AD1853 отличается от нее лишь токовым выходом. Данные микросхемы поддерживают все существующие на сегодняшний день наиболее популярные аудио форматы, включая 24bit 192kHz, что делает их очень удобным решением для построения универсального модуля ЦАП пригодного как для высококачественного прослушивания аудио компакт-дисков, так и для просмотра фильмов в формате DVD.

Микросхемы изначально имеют достаточно высокие качественные параметры и подключаются напрямую к шине I2S. Тем не менее, при желании получения максимального возможно качества, возможно дополнительное улучшение характеристик конечного устройства путем применения монофонического включения данных микросхем. При этом необходимо в каждую микросхему дважды загружать данные соответствующего ей канала, первый раз в исходном виде, второй раз в инвертированном. Так же, как и в случае с подключением AD1851, необходимо следить за одновременностью поступления данных обоих каналов.

Монофоническое подключение микросхемы AD1852 показано на следующем рисунке (взято из файла технической документации от Analog Devices):

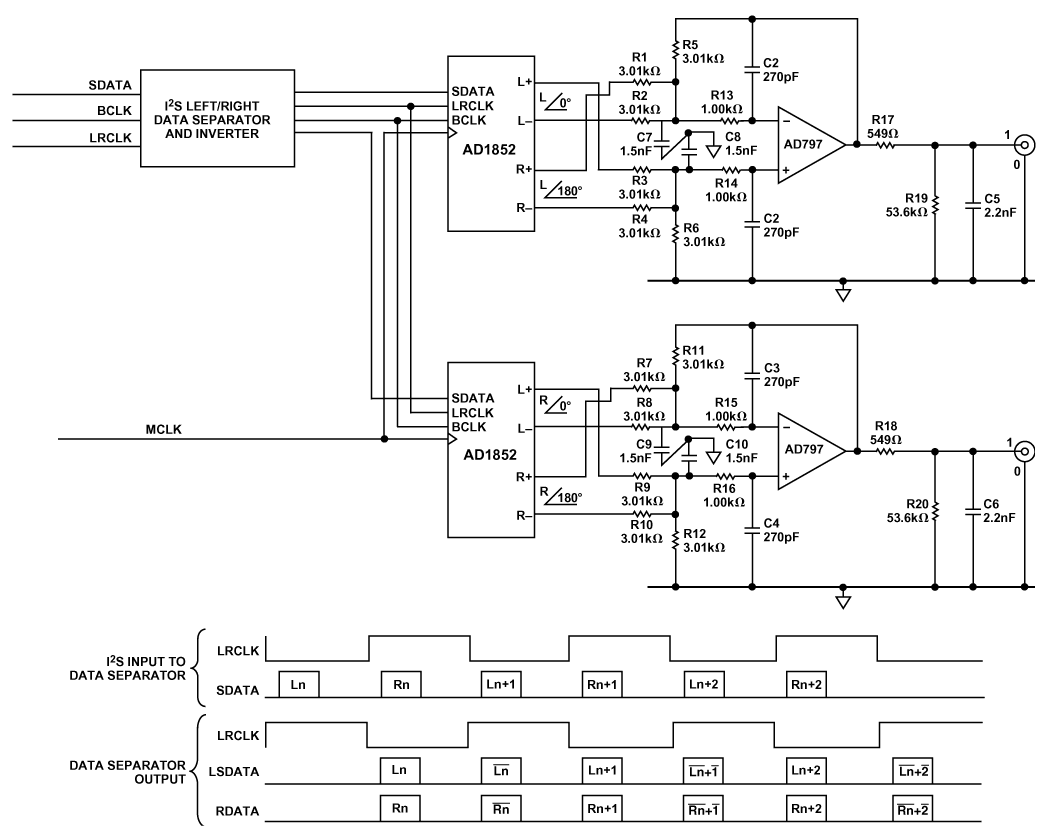


Рис. 8. – монофоническое подключение AD1852

Для осуществления необходимых манипуляций над данными составим устройство на основе регистра сдвига. Его общая структурная схема приведена на рис. 9. Как видно из временной диаграммы работы, максимальный интервал времени, на протяжении которого необходимо сохранять данные, составляет 1 фрейм, или 1 период сигнала LRCLK, или 64 периода сигнала SCLK. При этом инвертированные данные сдвинуты на 32 периода сигнала SCLK относительно данных исходных. Данные разделяются следующим образом:

* Данные снимаются одновременно с несколькими временными задержками, нужная порция данных выбирается схемой выборки



Рис. 9. – структурная схема устройства разделения данных

* В тот момент времени, когда на входной шине передаются данные правого канала, на выходе первого 32 разрядного сдвигового регистра будут находиться данные левого канала
* Во время передачи на входной шине данных левого канала на выходе первого сдвигового регистра будут данные правого канала предыдущего отсчета, а на выходе второго регистра еще только данные левого канала предыдущего отсчета
* Какие именно данные подавать на выходные шины, определяет схема выборки (также по 1й на каждый канал)
* Управление инверторами и схемами выборки осуществляется от сигнала LRCLK
* Сам сигнал LRCLK на выходные шины поступает в инвертированном виде

# Часть 2. Схемная реализация

## 

## 2.1 Определение требований к применяемым цифровым микросхемам

Поскольку обработка данных происходит лишь выбором необходимой последовательности и не предполагает непосредственного изменения битов информации в каждой посылке, данное устройство не вносит искажений в выходной сигнал блока ЦАП прямым образом. Тем не менее, следует учитывать, что музыкальный сигнал определяется не только содержанием пакетов данных, а и временем их прихода. Таким образом, необходимо учитывать частотные свойства применяемых микросхем. Необходимое минимальное быстродействие регистров составляет 64Fs, а лучше иметь запас в 3-4 раза. Для сигналов при просмотре фильмов в формате DVD Fs составляет 192 KHz. Следовательно, регистры должны работать с частотами не менее 13 MHz. С учетом желательности запаса по быстродействию, следует выбирать микросхемы, способные работать на частотах 40 MHz и выше.

Важно помнить, что данные на шине SDATA являются правильными во время нарастающего фронта импульса SCLK. Следовательно, загрузку в регистр необходимо также производить по нарастающему фронту.

Также при проектировании необходимо учитывать, что устройство должно быть легко повторяемым, т.е. желательно отказаться от применения дефицитных компонентов. В данном случае, проблема затрагивает только выбор сдвиговых регистров. Поскольку наиболее легкодоступными являются регистры длиной 8 бит, именно их имеет смысл использовать при построении устройства.

Еще одним фактором, влияющим на выбор микросхем, является то, что практически все современные микросхемы ЦАП и цифровых приемников имеют входные и выходные цепи с КМОП уровнями.

С учетом всего, выше сказанного, останавливаемся на сдвиговом регистре 74HC164. Он представляет собой 8-битный сдвиговый регистр с параллельным выходом, выполненный по КМОП технологии и способный работать на частотах до 50 MHz. Функциональным аналогом данного регистра является микросхема К155ИР8.

Логические микросхемы будем использовать также серии 74HC, что позволит избежать проблем согласования уровней. Быстродействие логических вентилей этой серии позволяет работать на частотах до 100 MHz и выше.

## 

## 2.2 Схемы электрические принципиальные

### 

### 2.2.1 Схема устройства разделения потока данных для подключения микросхемы ЦАП AD1851

Входы и выходы на схеме обозначены так, как они обычно обозначаются у готовых микросхем цифровых фильтров.

Для обеспечения устойчивости данных во время загрузки, сигнал CLK для микросхем ЦАП инвертирован относительно сигнала SCLK, по которому происходит загрузка данных в регистры и сдвиг. Это позволяет загрузить данные в микросхемы ЦАП в тот момент, когда на выходах регистров получены устойчивые состояния данных. Для того чтобы при этом не возникало смещения между сигналом LE и DOR/DOL, сигнал LRCLK тактируется сигналом SCLK. Для этого используется D-триггер 74HC74, функциональным аналогом которого является микросхема К155ТМ2. Также в связи с подачей на микросхемы ЦАП инвертированного сигнала SCLK возникает необходимость обеспечить дополнительное смещение всей последовательности входных данных еще на один клок.



Рис. 10. – схема электрическая принципиальная устройства разделения данных для подключения микросхемы ЦАП AD1851

На входе установлены буферы на микросхеме 74HC04, время распространения сигнала для которой составляет 8nS, что является достаточно малой величиной и не отражается на быстродействии устройства в целом.

Конденсаторы C1-C16 устанавливаются непосредственно возле выводов питания микросхем. C1-C8 – конденсаторы с твердым электролитом SANYO Os-Con SA 10μ x 6.3V, С9-С16 – керамические конденсаторы для поверхностного монтажа типоразмера 1206, емкостью 100n и группой ТКЕ X7R. Такой выбор обусловливает максимальную эффективность подавления помех по питанию при сохранении относительно невысокой стоимости устройства.

### 

### 2.2.2 Схема устройства разделения потока данных для монофонического подключения микросхемы ЦАП AD1852 (AD1853)



Рис. 11. – схема электрическая принципиальная устройства разделения данных для монофонического подключения микросхемы ЦАП AD1852 (AD1853)

Входы и выходы на схеме обозначены подобно обозначениям на схеме из документации на AD1852.

Для обеспечения устойчивости данных во время загрузки, сигнал BCLK\_O для микросхем ЦАП инвертирован относительно сигнала BCLK, по которому происходит загрузка данных в регистры и сдвиг. Это позволяет загрузить данные в микросхемы ЦАП в тот момент, когда на выходах регистров получены устойчивые состояния данных. Для того чтобы при этом не возникало смещения между сигналом LRCLK и SDATA\_L/SDATA\_R, сигнал LRCLK тактируется сигналом BCLK. Мультиплексоры на микросхемах DD10, DD11 управляются уже тактированным сигналом LRCLK. Для этого используется D-триггер 74HC74, функциональным аналогом которого является микросхема К155ТМ2. Поскольку AD1852(AD1853) также использует сигнал MCLK (128Fs/256Fs/384Fs), который связан с сигналом BCLK, сигнал MCLK\_O также инвертирован. Также в связи с подачей на микросхемы ЦАП инвертированного сигнала SCLK возникает необходимость обеспечить дополнительное смещение всей последовательности входных данных еще на один клок, что выполняется D-триггером DD12A.

При необходимости на входе устанавливаются буферные элементы.

Конденсаторы C1-C24 устанавливаются непосредственно возле выводов питания микросхем. C1-C12 – конденсаторы с твердым электролитом SANYO Os-Con SA 10μ x 6.3V, С13-С24 – керамические конденсаторы для поверхностного монтажа типоразмера 1206, емкостью 100n и группой ТКЕ X7R. Такой выбор обусловливает максимальную эффективность подавления помех по питанию при сохранении относительно невысокой стоимости устройства.

### 2.2.3 Общие замечания к схемам

Вход схем RESET подключается к генератору сброса системы. Активным уровнем сброса является уровень логического "0". Данный узел может быть выполнен, к примеру, на микросхеме ADM707.

Несмотря на то, что микросхемы ЦАП часто имеют питание цифровой части 5 V, имеет смысл выделить для них отдельный источник питания, а не использовать один источник для схемы разделения цифрового потока и ЦАП одновременно. Это позволит уменьшить уровень помех, проникающих из цифровой части микросхемы ЦАП в аналоговую часть устройства и улучшит звучание системы.

# Часть 3. Временные диаграммы

### 

### 3.1 Временная диаграмма работы устройства разделения данных для микросхемы ЦАП AD1851

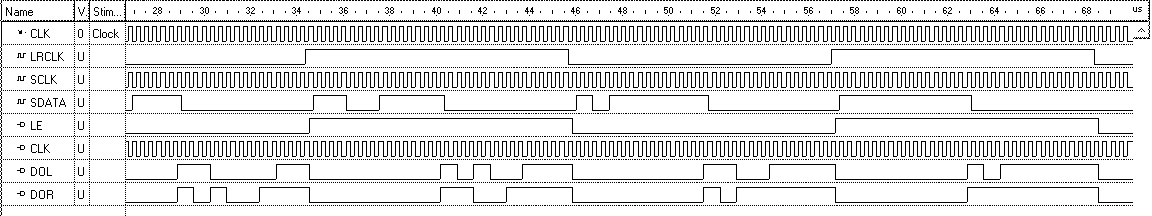
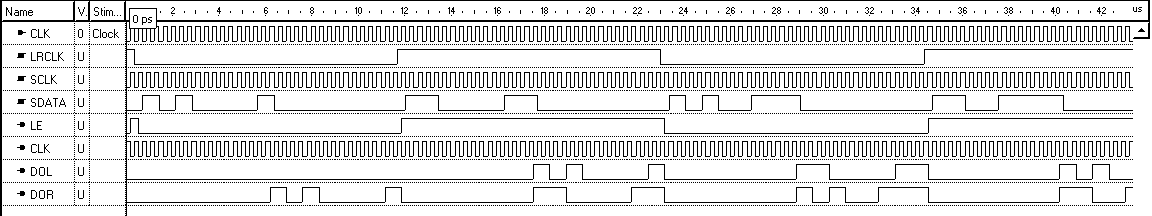


Рис. 12. - Временная диаграмма работы устройства разделения данных для микросхемы ЦАП AD1851

### 3.2 Временная диаграмма работы устройства разделения данных для микросхемы ЦАП AD1852

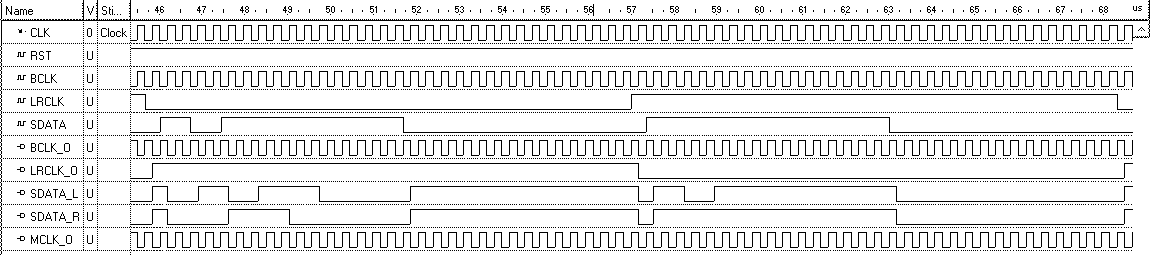
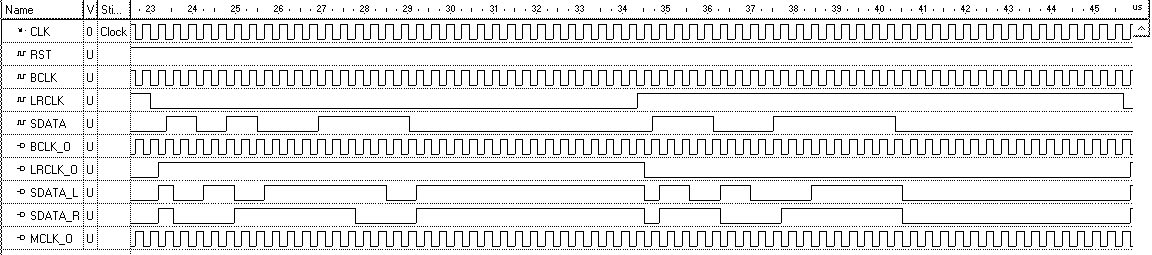
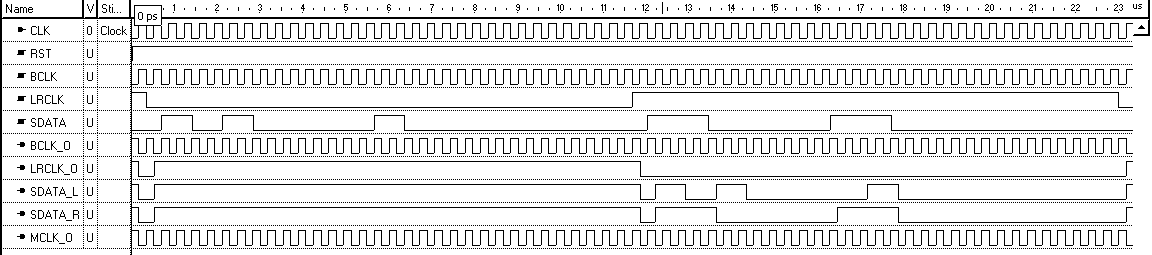


Рис. 13. - Временная диаграмма работы устройства разделения данных для микросхемы ЦАП AD1852

# Часть 4. Реализация схемы в проекте VHDL

Поскольку полученные схемы не всегда удобно размещать в конечном устройстве ввиду их громоздкости, имеет смысл создать проект на языке VHDL и составить код для программирования кристалла ПЛМ. Помимо значительного сокращения занимаемой на плате площади это позволит также поднять общее быстродействие модуля разделения потоков данных, снизить наводимые цифровые помехи, промоделировать устройство без пайки.

Коды на языке VHDL представлены ниже с краткими пояснениями, где это необходимо.

Для начала выберем необходимые компоненты и составим их модели:

--Components.vhd

--16-bit Serial to Parallel Shift Register with asynchronous reset

library ieee ;

use ieee.std\_logic\_1164.all ;

entity SPREG16R is

port ( RST : in std\_logic ;

CLK : in std\_logic ;

SI : in std\_logic ;

Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8 : out std\_logic ;

Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 : out std\_logic

) ;

end SPREG16R ;

architecture v1 of SPREG16R is

signal sreg16 : std\_logic\_vector (15 downto 0) ;

begin

process (RST,CLK)

begin

if RST = '1' then

sreg16 <= (others => '0') ;

elsif CLK = '1' and CLK'event then

sreg16 <= sreg16(14 downto 0) & SI ;

end if ;

end process ;

Q15 <= sreg16(15) after 1 ns ;

Q14 <= sreg16(14) after 1 ns ;

Q13 <= sreg16(13) after 1 ns ;

Q12 <= sreg16(12) after 1 ns ;

Q11 <= sreg16(11) after 1 ns ;

Q10 <= sreg16(10) after 1 ns ;

Q9 <= sreg16(9) after 1 ns ;

Q8 <= sreg16(8) after 1 ns ;

Q7 <= sreg16(7) after 1 ns ;

Q6 <= sreg16(6) after 1 ns ;

Q5 <= sreg16(5) after 1 ns ;

Q4 <= sreg16(4) after 1 ns ;

Q3 <= sreg16(3) after 1 ns ;

Q2 <= sreg16(2) after 1 ns ;

Q1 <= sreg16(1) after 1 ns ;

Q0 <= sreg16(0) after 1 ns ;

end v1 ;

--32-bit Serial to Parallel Shift Register with asynchronous reset

library ieee ;

use ieee.std\_logic\_1164.all ;

entity SPREG32R is

port ( RST : in std\_logic ;

CLK : in std\_logic ;

SI : in std\_logic ;

Q31,Q30,Q29,Q28,Q27,Q26,Q25,Q24 : out std\_logic;

Q23,Q22,Q21,Q20,Q19,Q18,Q17,Q16 : out std\_logic;

Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8 : out std\_logic ;

Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 : out std\_logic

) ;

end SPREG32R ;

architecture v1 of SPREG32R is

signal sreg32 : std\_logic\_vector (31 downto 0) ;

begin

process (RST,CLK)

begin

if RST = '1' then

sreg32 <= (others => '0') ;

elsif CLK = '1' and CLK'event then

sreg32 <= sreg32(30 downto 0) & SI ;

end if ;

end process ;

Q31 <= sreg32(31) after 1 ns ;

Q30 <= sreg32(30) after 1 ns ;

Q29 <= sreg32(29) after 1 ns ;

Q28 <= sreg32(28) after 1 ns ;

Q27 <= sreg32(27) after 1 ns ;

Q26 <= sreg32(26) after 1 ns ;

Q25 <= sreg32(25) after 1 ns ;

Q24 <= sreg32(24) after 1 ns ;

Q23 <= sreg32(23) after 1 ns ;

Q22 <= sreg32(22) after 1 ns ;

Q21 <= sreg32(21) after 1 ns ;

Q20 <= sreg32(20) after 1 ns ;

Q19 <= sreg32(19) after 1 ns ;

Q18 <= sreg32(18) after 1 ns ;

Q17 <= sreg32(17) after 1 ns ;

Q16 <= sreg32(16) after 1 ns ;

Q15 <= sreg32(15) after 1 ns ;

Q14 <= sreg32(14) after 1 ns ;

Q13 <= sreg32(13) after 1 ns ;

Q12 <= sreg32(12) after 1 ns ;

Q11 <= sreg32(11) after 1 ns ;

Q10 <= sreg32(10) after 1 ns ;

Q9 <= sreg32(9) after 1 ns ;

Q8 <= sreg32(8) after 1 ns ;

Q7 <= sreg32(7) after 1 ns ;

Q6 <= sreg32(6) after 1 ns ;

Q5 <= sreg32(5) after 1 ns ;

Q4 <= sreg32(4) after 1 ns ;

Q3 <= sreg32(3) after 1 ns ;

Q2 <= sreg32(2) after 1 ns ;

Q1 <= sreg32(1) after 1 ns ;

Q0 <= sreg32(0) after 1 ns ;

end v1 ;

--64-bit Serial to Parallel Shift Register with asynchronous reset

library ieee ;

use ieee.std\_logic\_1164.all ;

entity SPREG64R is

port ( RST : in std\_logic ;

CLK : in std\_logic ;

SI : in std\_logic ;

Q63,Q62,Q61,Q60,Q59,Q58,Q57,Q56 : out std\_logic;

Q55,Q54,Q53,Q52,Q51,Q50,Q49,Q48 : out std\_logic;

Q47,Q46,Q45,Q44,Q43,Q42,Q41,Q40 : out std\_logic;

Q39,Q38,Q37,Q36,Q35,Q34,Q33,Q32 : out std\_logic;

Q31,Q30,Q29,Q28,Q27,Q26,Q25,Q24 : out std\_logic;

Q23,Q22,Q21,Q20,Q19,Q18,Q17,Q16 : out std\_logic;

Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8 : out std\_logic;

Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 : out std\_logic

) ;

end SPREG64R ;

architecture v1 of SPREG64R is

signal sreg64 : std\_logic\_vector (63 downto 0) ;

begin

process (RST,CLK)

begin

if RST = '1' then

sreg64 <= (others => '0') ;

elsif CLK = '1' and CLK'event then

sreg64 <= sreg64(62 downto 0) & SI ;

end if ;

end process ;

Q63 <= sreg64(63) after 1 ns ;

Q62 <= sreg64(62) after 1 ns ;

Q61 <= sreg64(61) after 1 ns ;

Q60 <= sreg64(60) after 1 ns ;

Q59 <= sreg64(59) after 1 ns ;

Q58 <= sreg64(58) after 1 ns ;

Q57 <= sreg64(57) after 1 ns ;

Q56 <= sreg64(56) after 1 ns ;

Q55 <= sreg64(55) after 1 ns ;

Q54 <= sreg64(54) after 1 ns ;

Q53 <= sreg64(53) after 1 ns ;

Q52 <= sreg64(52) after 1 ns ;

Q51 <= sreg64(51) after 1 ns ;

Q50 <= sreg64(50) after 1 ns ;

Q49 <= sreg64(49) after 1 ns ;

Q48 <= sreg64(48) after 1 ns ;

Q47 <= sreg64(47) after 1 ns ;

Q46 <= sreg64(46) after 1 ns ;

Q45 <= sreg64(45) after 1 ns ;

Q44 <= sreg64(44) after 1 ns ;

Q43 <= sreg64(43) after 1 ns ;

Q42 <= sreg64(42) after 1 ns ;

Q41 <= sreg64(41) after 1 ns ;

Q40 <= sreg64(40) after 1 ns ;

Q39 <= sreg64(39) after 1 ns ;

Q38 <= sreg64(38) after 1 ns ;

Q37 <= sreg64(37) after 1 ns ;

Q36 <= sreg64(36) after 1 ns ;

Q35 <= sreg64(35) after 1 ns ;

Q34 <= sreg64(34) after 1 ns ;

Q33 <= sreg64(33) after 1 ns ;

Q32 <= sreg64(32) after 1 ns ;

Q31 <= sreg64(31) after 1 ns ;

Q30 <= sreg64(30) after 1 ns ;

Q29 <= sreg64(29) after 1 ns ;

Q28 <= sreg64(28) after 1 ns ;

Q27 <= sreg64(27) after 1 ns ;

Q26 <= sreg64(26) after 1 ns ;

Q25 <= sreg64(25) after 1 ns ;

Q24 <= sreg64(24) after 1 ns ;

Q23 <= sreg64(23) after 1 ns ;

Q22 <= sreg64(22) after 1 ns ;

Q21 <= sreg64(21) after 1 ns ;

Q20 <= sreg64(20) after 1 ns ;

Q19 <= sreg64(19) after 1 ns ;

Q18 <= sreg64(18) after 1 ns ;

Q17 <= sreg64(17) after 1 ns ;

Q16 <= sreg64(16) after 1 ns ;

Q15 <= sreg64(15) after 1 ns ;

Q14 <= sreg64(14) after 1 ns ;

Q13 <= sreg64(13) after 1 ns ;

Q12 <= sreg64(12) after 1 ns ;

Q11 <= sreg64(11) after 1 ns ;

Q10 <= sreg64(10) after 1 ns ;

Q9 <= sreg64(9) after 1 ns ;

Q8 <= sreg64(8) after 1 ns ;

Q7 <= sreg64(7) after 1 ns ;

Q6 <= sreg64(6) after 1 ns ;

Q5 <= sreg64(5) after 1 ns ;

Q4 <= sreg64(4) after 1 ns ;

Q3 <= sreg64(3) after 1 ns ;

Q2 <= sreg64(2) after 1 ns ;

Q1 <= sreg64(1) after 1 ns ;

Q0 <= sreg64(0) after 1 ns ;

end v1 ;

-- D Flip Flop w asynchronous reset

library ieee ;

use ieee.std\_logic\_1164.all ;

entity DFFR is

port (RST : in std\_logic ;

CLK : in std\_logic ;

D : in std\_logic ;

Q : out std\_logic ;

QN : out std\_logic

) ;

end DFFR ;

architecture v1 of DFFR is

signal n1 : std\_logic ;

begin

process (RST,CLK)

begin

if RST = '1' then

n1 <= '0' after 1 ns ;

elsif (CLK'event and CLK='1') then

n1 <= D after 1 ns ;

end if ;

end process ;

Q <= n1 after 1 ns ;

QN <= not n1 after 1 ns ;

end v1 ;

-- 2-input NAND

library ieee ;

use ieee.std\_logic\_1164.all ;

entity NAND2 is

port (

IN1,IN0 : in std\_logic ;

Z : out std\_logic

) ;

end NAND2 ;

architecture v1 of NAND2 is

begin

Z <= not (IN1 and IN0) after 1 ns ;

end v1 ;

-- Inverter

library ieee ;

use ieee.std\_logic\_1164.all ;

entity INV is

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end INV ;

architecture v1 of INV is

begin

Z <= not IN0 after 1 ns ;

end v1 ;

-- Buffer

library ieee ;

use ieee.std\_logic\_1164.all ;

entity BUF is

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end BUF ;

architecture v1 of BUF is

begin

Z <= IN0 after 1 ns ;

end v1 ;

-- 2-1 Multiplexer

library ieee ;

use ieee.std\_logic\_1164.all ;

entity MUX21 is

port (IN0,IN1,SEL : in std\_logic ;

Z : out std\_logic

) ;

end MUX21 ;

architecture v1 of MUX21 is

begin

Z <= IN0 after 1 ns when SEL = '0' else IN1 ;

end ;

-- 4-1 Multiplexer

library ieee ;

use ieee.std\_logic\_1164.all ;

entity MUX41 is

port (IN0,IN1,IN2,IN3,SEL0,SEL1 : in std\_logic ;

Z : out std\_logic

) ;

end MUX41 ;

architecture v1 of MUX41 is

signal SEL : std\_logic\_vector(1 downto 0) ;

begin

SEL <= SEL1 & SEL0 ;

Z <= IN0 when SEL = "00" else

IN1 when SEL = "01" else

IN2 when SEL = "10" else

IN3 ;

end ;

Далее составим модели для каждого устройства разделения данных, при этом дополнительно оптимизируем набор элементов и соединений для получения наибольшего возможного соответствия между задержками распространения выходных сигналов.

--DS1851.vhd

--Data Separator for AD1851/AD1861/AD1862/AD1865 parallel DAC

library ieee ;

use ieee.std\_logic\_1164.all ;

entity DS1851 is

port (

RST : in std\_logic;

SCLK : in std\_logic;

SDATA : in std\_logic;

LRCLK : in std\_logic;

RL1 : in std\_logic;

RL0 : in std\_logic;

LE : out std\_logic;

CLK : out std\_logic;

DOL : out std\_logic;

DOR : out std\_logic

);

end DS1851;

architecture v1 of DS1851 is

component SPREG16R

port ( RST : in std\_logic ;

CLK : in std\_logic ;

SI : in std\_logic ;

Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8 : out std\_logic ;

Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 : out std\_logic

) ;

end component ;

component SPREG32R

port ( RST : in std\_logic ;

CLK : in std\_logic ;

SI : in std\_logic ;

Q31,Q30,Q29,Q28,Q27,Q26,Q25,Q24 : out std\_logic;

Q23,Q22,Q21,Q20,Q19,Q18,Q17,Q16 : out std\_logic;

Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8 : out std\_logic ;

Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 : out std\_logic

) ;

end component ;

component DFFR

port (RST : in std\_logic ;

CLK : in std\_logic ;

D : in std\_logic ;

Q : out std\_logic ;

QN : out std\_logic

) ;

end component ;

component INV

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end component;

component BUF

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end component ;

component MUX41

port (IN0,IN1,IN2,IN3,SEL0,SEL1 : in std\_logic ;

Z : out std\_logic

) ;

end component ;

signal nLRCLK, pLRCLK, nSCLK, pSCLK, nSDATA, pSDATA : std\_logic;

signal i24b, i20b, i18b, i16b : std\_logic;

signal RESET, iDOL, iDOR, iCLK, iLE : std\_logic;

begin

RSTI: INV port map (IN0 => RST, Z => RESET);

DD1A: INV port map (IN0 => SCLK, Z => nSCLK);

DD1D: INV port map (IN0=> nSCLK, Z => pSCLK);

DD1B: INV port map (IN0 => SDATA, Z => nSDATA);

DD1E: INV port map (IN0 => nSDATA, Z => pSDATA);

DD1C: INV port map (IN0 => LRCLK, Z => nLRCLK);

DD1F: INV port map (IN0 => nLRCLK, Z => pLRCLK);

CLKB: BUF port map (IN0 => nSCLK, Z => iCLK);

DD2: SPREG16R port map (RST => RESET, CLK => pSCLK, SI => pSDATA,

Q7 => i24b, Q11 => i20b, Q13 => i18b, Q15 => i16b);

DDMX: MUX41 port map (IN0 => i24b, IN1 => i20b, IN2 => i18b, IN3 => i16b,

SEL0 => RL0, SEL1 => RL1, Z => iDOR);

DD4: DFFR port map (RST => RESET, D => pLRCLK, CLK => pSCLK, Q => iLE);

DD5: SPREG32R port map (RST => RESET, CLK => pSCLK, SI => iDOR,

Q31 => iDOL);

BUFE: BUF port map (IN0 => iLE, Z => LE);

BUFC: BUF port map (IN0 => iCLK, Z => CLK);

BUFL: BUF port map (IN0 => iDOL, Z => DOL);

BUFR: BUF port map (IN0 => iDOR, Z => DOR);

end v1 ;

--DS1853.vhd

--Data Separator for AD1852/AD1853 delta-sigma DAC

library ieee ;

use ieee.std\_logic\_1164.all ;

entity DS1853 is

port (

RST : in std\_logic;

BCLK : in std\_logic;

SDATA : in std\_logic;

LRCLK : in std\_logic;

MCLK : in std\_logic;

BCLK\_O : out std\_logic;

LRCLK\_O : out std\_logic;

SDATA\_L : out std\_logic;

SDATA\_R : out std\_logic;

MCLK\_O : out std\_logic

);

end DS1853;

architecture v1 of DS1853 is

component SPREG64R

port ( RST : in std\_logic ;

CLK : in std\_logic ;

SI : in std\_logic ;

Q63,Q62,Q61,Q60,Q59,Q58,Q57,Q56 : out std\_logic;

Q55,Q54,Q53,Q52,Q51,Q50,Q49,Q48 : out std\_logic;

Q47,Q46,Q45,Q44,Q43,Q42,Q41,Q40 : out std\_logic;

Q39,Q38,Q37,Q36,Q35,Q34,Q33,Q32 : out std\_logic;

Q31,Q30,Q29,Q28,Q27,Q26,Q25,Q24 : out std\_logic;

Q23,Q22,Q21,Q20,Q19,Q18,Q17,Q16 : out std\_logic;

Q15,Q14,Q13,Q12,Q11,Q10,Q9,Q8 : out std\_logic;

Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0 : out std\_logic

) ;

end component ;

component DFFR

port (RST : in std\_logic ;

CLK : in std\_logic ;

D : in std\_logic ;

Q : out std\_logic ;

QN : out std\_logic

) ;

end component ;

component INV

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end component;

component BUF

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end component ;

component MUX21

port (IN0,IN1,SEL : in std\_logic ;

Z : out std\_logic

) ;

end component ;

signal iBCLK, iSDATA, iLRCLK, iMCLK : std\_logic;

signal RESET, nLRCLK, pLRCLK : std\_logic;

signal o31, o63, o31n, o31p, o63n, bSDATA: std\_logic;

begin

INBB: BUF port map (IN0 => BCLK, Z => iBCLK);

INBS: BUF port map (IN0 => SDATA, Z => bSDATA);

INBL: BUF port map (IN0 => LRCLK, Z => iLRCLK);

INBM: BUF port map (IN0 => MCLK, Z => iMCLK);

RSTI: INV port map (IN0 => RST, Z => RESET);

DD12A: DFFR port map (RST => RESET, D => iLRCLK,

CLK => iBCLK, Q => pLRCLK, QN => nLRCLK);

DD12B: DFFR port map (RST => RESET, D => bSDATA,

CLK => iBCLK, Q => iSDATA);

DD1: SPREG64R port map (RST => RESET, SI => iSDATA,

CLK => iBCLK, Q31 => o31, Q63 => o63);

O31B: BUF port map (IN0 => o31, Z => o31p);

O31I: INV port map (IN0 => o31, Z => o31n);

O63I: INV port map (IN0 => o63, Z => o63n);

MUXL: MUX21 port map (IN0 => o63n, IN1 => o31p,

SEL => pLRCLK, Z => SDATA\_L);

MUXR: MUX21 port map (IN0 => o31n, IN1 => iSDATA,

SEL => pLRCLK, Z => SDATA\_R);

INVL: INV port map (IN0 => pLRCLK, Z => LRCLK\_O);

INVB: INV port map (IN0 => iBCLK, Z => BCLK\_O);

INVM: INV port map (IN0 => iMCLK, Z => MCLK\_O);

end v1 ;

Далее необходимо составить код для генератора отладочной последовательности.

--Tester1851.vhd

--Test signal generator for DS1851

library ieee ;

use ieee.std\_logic\_1164.all ;

entity Tester1851 is

port (

CLK : in std\_logic

);

end Tester1851;

architecture v1 of Tester1851 is

component TS1851

port (

RST : out std\_logic;

SCLK : in std\_logic;

SDATA : out std\_logic;

LRCLK : out std\_logic

);

end component ;

component DS1851

port (

RST : in std\_logic;

SCLK : in std\_logic;

SDATA : in std\_logic;

LRCLK : in std\_logic;

RL1 : in std\_logic;

RL0 : in std\_logic;

LE : out std\_logic;

CLK : out std\_logic;

DOL : out std\_logic;

DOR : out std\_logic

);

end component ;

component BUF

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end component ;

signal SCLK, SDATA, LRCLK, RST : std\_logic;

signal LD1, LD2, LD3, RD1, RD2, RD3, DSReg : std\_logic\_vector (31 downto 0);

begin

D0: BUF port map (IN0 => CLK, Z => SCLK);

D2: DS1851 port map (RST => RST, SCLK => SCLK,

SDATA => SDATA, LRCLK => LRCLK,

RL1 => '1', RL0 => '1');

Test: process

begin

LD1 <= "01100110000000011000000000000000";

RD1 <= "01111000000001111000000000000000";

LD2 <= "01100110000111111000000000000000";

RD2 <= "01111000011111111000000000000000";

LD3 <= "01100111111111111000000000000000";

RD3 <= "01111111111111111000000000000000";

SDATA <= '0';

LRCLK <= '1';

RST <= '0';

wait for 50ns;

RST <= '1';

wait on SCLK until SCLK='0';

DSReg <= LD1;

wait for 10ns;

LRCLK <= '0';

SDATA <= DSReg(31);

LD1L: for i in 30 downto 0 loop

wait on SCLK until SCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop LD1L;

wait on SCLK until SCLK='0';

DSReg <= RD1;

wait for 10ns;

LRCLK <= '1';

SDATA <= DSReg(31);

RD1L: for i in 30 downto 0 loop

wait on SCLK until SCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop RD1L;

wait on SCLK until SCLK='0';

DSReg <= LD2;

wait for 10ns;

LRCLK <= '0';

SDATA <= DSReg(31);

LD2L: for i in 30 downto 0 loop

wait on SCLK until SCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop LD2L;

wait on SCLK until SCLK='0';

DSReg <= RD2;

wait for 10ns;

LRCLK <= '1';

SDATA <= DSReg(31);

RD2L: for i in 30 downto 0 loop

wait on SCLK until SCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop RD2L;

wait on SCLK until SCLK='0';

DSReg <= LD3;

wait for 10ns;

LRCLK <= '0';

SDATA <= DSReg(31);

LD3L: for i in 30 downto 0 loop

wait on SCLK until SCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop LD3L;

wait on SCLK until SCLK='0';

DSReg <= RD3;

wait for 10ns;

LRCLK <= '1';

SDATA <= DSReg(31);

RD3L: for i in 30 downto 0 loop

wait on SCLK until SCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop RD3L;

wait on SCLK until SCLK='0';

DSReg <= LD1;

wait for 10ns;

LRCLK <= '0';

wait;

end process;

end v1;

--Tester1853.vhd

--Test signal generator for DS1853

library ieee ;

use ieee.std\_logic\_1164.all ;

entity Tester1853 is

port (

CLK : in std\_logic

);

end Tester1853;

architecture v1 of Tester1853 is

component TS1853

port (

RST : out std\_logic;

SCLK : in std\_logic;

SDATA : out std\_logic;

LRCLK : out std\_logic

);

end component ;

component DS1853

port (

RST : in std\_logic;

BCLK : in std\_logic;

SDATA : in std\_logic;

LRCLK : in std\_logic;

MCLK : in std\_logic;

BCLK\_O : out std\_logic;

LRCLK\_O : out std\_logic;

SDATA\_L : out std\_logic;

SDATA\_R : out std\_logic;

MCLK\_O : out std\_logic

);

end component ;

component BUF

port (

IN0 : in std\_logic ;

Z : out std\_logic

) ;

end component ;

signal BCLK, SDATA, LRCLK, RST : std\_logic;

signal LD1, LD2, LD3, RD1, RD2, RD3, DSReg : std\_logic\_vector (31 downto 0);

begin

D0: BUF port map (IN0 => CLK, Z => BCLK);

D2: DS1853 port map (RST => RST, BCLK => BCLK,

SDATA => SDATA, LRCLK => LRCLK,

MCLK => BCLK);

Test: process

begin

LD1 <= "01100110000000011000000000000000";

RD1 <= "01111000000001111000000000000000";

LD2 <= "01100110000111111000000000000000";

RD2 <= "01111000011111111000000000000000";

LD3 <= "01100111111111111000000000000000";

RD3 <= "01111111111111111000000000000000";

SDATA <= '0';

LRCLK <= '1';

RST <= '0';

wait for 50ns;

RST <= '1';

wait on BCLK until BCLK='0';

DSReg <= LD1;

wait for 10ns;

LRCLK <= '0';

SDATA <= DSReg(31);

LD1L: for i in 30 downto 0 loop

wait on BCLK until BCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop LD1L;

wait on BCLK until BCLK='0';

DSReg <= RD1;

wait for 10ns;

LRCLK <= '1';

SDATA <= DSReg(31);

RD1L: for i in 30 downto 0 loop

wait on BCLK until BCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop RD1L;

wait on BCLK until BCLK='0';

DSReg <= LD2;

wait for 10ns;

LRCLK <= '0';

SDATA <= DSReg(31);

LD2L: for i in 30 downto 0 loop

wait on BCLK until BCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop LD2L;

wait on BCLK until BCLK='0';

DSReg <= RD2;

wait for 10ns;

LRCLK <= '1';

SDATA <= DSReg(31);

RD2L: for i in 30 downto 0 loop

wait on BCLK until BCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop RD2L;

wait on BCLK until BCLK='0';

DSReg <= LD3;

wait for 10ns;

LRCLK <= '0';

SDATA <= DSReg(31);

LD3L: for i in 30 downto 0 loop

wait on BCLK until BCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop LD3L;

wait on BCLK until BCLK='0';

DSReg <= RD3;

wait for 10ns;

LRCLK <= '1';

SDATA <= DSReg(31);

RD3L: for i in 30 downto 0 loop

wait on BCLK until BCLK='0';

DSReg <= DSReg(30 downto 0) & '0';

wait for 10ns;

SDATA <= DSReg(31);

end loop RD3L;

wait on BCLK until BCLK='0';

DSReg <= LD1;

wait for 10ns;

LRCLK <= '0';

wait;

end process;

end v1;

## Выводы

При выполнении работы были рассмотрены вопросы реализации разделения цифрового потока от приемника сигнала SPDIF (и/или AES/EBU) для подачи его непосредственно на микросхемы ЦАП. Из набора наиболее часто используемых способов реализации были выбраны наиболее универсальные и лучшим образом отвечающие требованиям стандарта.

При схемной реализации была выполнена оптимизация схемы для получения наилучших условий прохождения сигнала. Также была затронута проблема обеспечения качественной фильтрации питания в аппаратуре с большим числом цифровых микросхем.

В результате выполнения работы были получены работающие схемы, пригодные к воплощению в готовом устройстве. Работа схем была промоделирована на языке VHDL. Помимо VHDL моделей был получен готовый код, пригодный для прошивки кристалла ПЛМ.

Тем не менее, основным результатом работы следует считать закрепление знаний по дисциплинам "Прикладная теория цифровых автоматов", "Цифровая схемотехника" и "Моделирование на языке VHDL", а также повышение опыта работы с соответствующими средствами и способами разработки.