|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ высшего образования |
| **«Национальный исследовательский ядерный университет «МИФИ»** |
| ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ |
| КАФЕДРА «КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ» (№12) |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**на курсовую учебно-исследовательскую работу по дисциплине**

**СХЕМОТЕХНИКА ЦИФРОВЫХ УСТРОЙСТВ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тема: SerDes with align feature** | | | | |
| Студент | Кротов Евгений Геннадьевич | | Группа | С21-501 |
|  | ФИО | |  |  |
| Студент | | Косолобов Дмитрий Сергеевич | | Группа | С21-501 |
|  | | ФИО | |  |  |
|  |  | |  |  |
| Руководитель | | Решетько Валерий Михайлович, старший преподаватель НИЯУ МИФИ | | |
|  | | ФИО, степень, звание, должность | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  | Кротов Е.Г. |
| Студент | подпись |  | Косолобов Д.С |
|  | подпись |  | ФИО |
| Руководитель |  |  | Решетько В.М. |

подпись

|  |  |  |  |
| --- | --- | --- | --- |
| Руководитель |  |  | Сапрыкина В.С. |

подпись

**Москва, 2023**

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc149486289)

[1. Serializer/Deserializer 4](#_Toc149486290)

[1.1. Общая функция 4](#_Toc149486291)

[1.2. Align Feature 5](#_Toc149486292)

[1.3. Язык VHDL 5](#_Toc149486293)

[2. ОПИСАНИЕ РАБОТЫ SERDES БЛОКОВ 7](#_Toc149486294)

[2.1. Условное графическое обозначение 7](#_Toc149486295)

[2.2. Режимы работы блока SerDes 7](#_Toc149486296)

[2.3. Тестирование программы 7](#_Toc149486297)

[2.4. Временные диаграммы 7](#_Toc149486298)

[2.5. Синтез 7](#_Toc149486299)

[ЗАКЛЮЧЕНИЕ 7](#_Toc149486300)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 8](#_Toc149486301)

[Приложение 1 9](#_Toc149486302)

[Код блоков SerDes 9](#_Toc149486303)

[Приложение 2 14](#_Toc149486304)

[Тесты к блоку SerDes 14](#_Toc149486305)

# ВВЕДЕНИЕ

Сериализатор/Десериализатор (SerDes) — пара функциональных блоков, обычно используемых в высокоскоростной связи, для преобразования данных между последовательным и параллельным интерфейсами в обоих направлениях. Термин «SerDes» относится к обобщённым интерфейсам, используемым в различных технологиях и приложениях. Основное использование SerDes — обеспечение высокоскоростной передачи данных по одиночной линии или по дифференциальной паре для уменьшения количества контактов ввода/вывода микросхемы и межсоединений между компонентами.

Цель данной работы — создание SerDes блоков с выравниванием, которые можно протестировать в среде моделирования.

Для достижения поставленной цели в работе решаются следующие задачи:

* изучение теоретического материала по языку VHDL, симулятору ModelSim и САПР Quartus;
* проектирование SerDes блоков с выравниванием;
* написание кода для SerDes блоков с выравниванием;
* тестирование работы SerDes блоков с выравниванием.

# 1. Serializer/Deserializer

## 1.1. Общая функция

Основная структура SerDes разделяется на два функциональных блока: блок параллельно-последовательного преобразователя (PISO, Parallel In Serial Out) и блок последовательно-параллельного преобразователя (SIPO, Serial In Parallel Out), изображенных на Рис.1.1. Есть четыре различных архитектуры SerDes:

— SerDes с параллельным тактированием;

— SerDes с внутренним тактированием;

— 8b/10b SerDes (или более сложные коды);

— SerDes с чередованием битов.

Блок параллельно-последовательного преобразователя PISO обычно имеет параллельный тактовый вход, набор входных линий данных, и вход фиксации данных. Он может использовать внутреннюю или внешнюю фазовую автоподстройку частоты (ФАПЧ) для умножения входящего параллельного тактового сигнала до частоты тактового сигнала последовательного выхода. Простейшая форма PISO имеет только сдвиговый регистр, который принимает параллельные данные с параллельным тактированием, и выдвигает его с более высокой последовательной тактовой частотой. Реализации могут также использовать двойную буферизацию регистра, чтобы избежать метастабильности при передаче данных между частями с разной тактовой частотой.

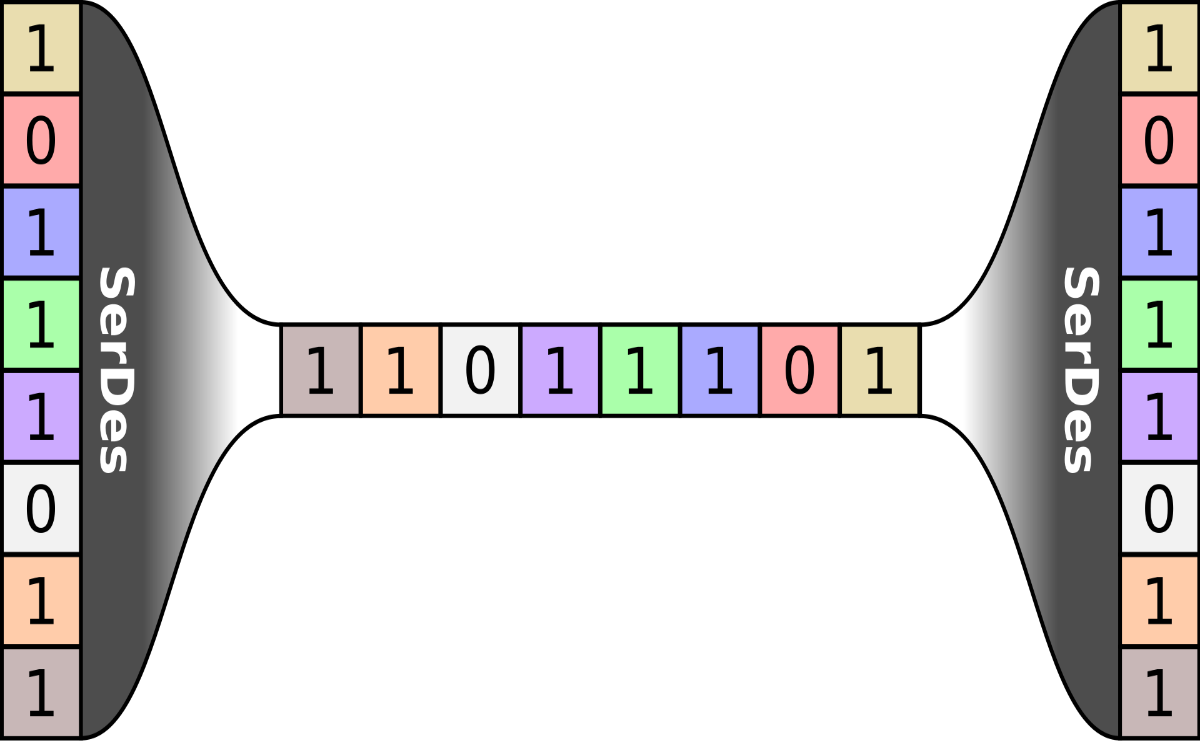


Рис. 1.1 Принцип работы SerDes.

SIPO, блок последовательно-параллельного преобразователя, обычно имеют выход входящей тактовой частоты, набор выходных линий данных и выходные защелки данных. Приёмный тактовый сигнал могут быть восстановлены из данных по методу восстановления тактового сигнала. Однако SerDes, которые не передают тактовые сигналы, используют образцовые такты для синхронизации ФАПЧ с необходимой частой TX, во избежание низкочастотных гармоник, присутствующих в потоке данных. Блок SIPO понижает входящую тактовую частоту для параллельного выхода. Реализации, как правило, имеют два регистра, подключенных как двойной буфер. Один регистр используется для тактирования последовательного потока, а другой используется для хранения данных более медленной, параллельной стороны.

## 1.2. Align Feature

Alignfeature (или Align Feature) — это функция или механизм, который помогает выравнивать и синхронизировать данные, передаваемые через SerDes. Выравнивание данных важно для того, чтобы правильно интерпретировать их на приемной стороне и избежать ошибок при передаче данных на высоких скоростях.

Некоторые типы SerDes включают блок кодирования/декодирования. Целью такого кодирования и декодирования является, как правило, изменение статистических свойств передаваемого сигнала для облегчения восстановления тактового сигнала в приёмнике и обеспечения синхронизации фреймов. Также кодирование может обеспечивать баланс по постоянному току.

## 1.3. Язык VHDL

VHDL – это язык описания аппаратуры (HDL), позволяющий моделировать поведение и структуру цифровых систем на различных уровнях абстракции, начиная с системного уровня и заканчивая уровнем логических вентилей, для целей проектирования, документирования и верификации. С 1987 года VHDL стандартизован Институтом инженеров по электротехнике и электронике (IEEE) в виде стандарта IEEE Std 1076, последней версией которого является IEEE Std 1076-2019. Для моделирования аналоговых и смешанных сигнальных систем был разработан стандартизованный IEEE HDL на основе VHDL под названием VHDL-AMS (официально IEEE 1076.1).

VHDL назван в честь программы Министерства обороны США, в рамках которой он был создан, - Very High Speed Integrated Circuits Program (VHSIC). В начале 1980-х годов в рамках программы VHSIC был проведен поиск нового языка программирования HDL для использования при проектировании интегральных схем, которые предполагалось создать. Результатом этих усилий стал VHDL версии 7.2, выпущенный в 1985 году. В следующем году началась работа по его стандартизации в качестве стандарта IEEE.

# 2. ОПИСАНИЕ РАБОТЫ SERDES БЛОКОВ

## 2.1. Условное графическое обозначение

Входные и выходные порты сериализатора и десериализатора:

Таблица 2.1. — Описание сигналов сериализатора.

| № п.п. | Сигнал | Напр. | Описание |
| --- | --- | --- | --- |
| Входные и выходные порты сериализатора: | | | |
| 1 | *clk\_in\_ser* | in | *Входной тактовый сигнал сериализатора (тип данных: std\_logic).* |
| 2 | *reset\_ser* | in | *Сигнал сброса сериализатора (тип данных: std\_logic).* |
| 3 | *valid\_data:* | in | *Сигнал, указывающий, что передаются валидные данные (тип данных: std\_logic).* |
| 4 | *data\_in\_ser* | in | *Входные данные для сериализации (8-битная последовательность) (тип данных: std\_logic\_vector(7 downto 0)).* |
| 5 | *clk\_in\_ser:* | in | *Входной тактовый сигнал сериализатора (тип данных: std\_logic).* |
| 6 | *clk\_out\_ser* | out | *Выходной тактовый сигнал сериализатора (тип данных: std\_logic).* |
| 7 | *data\_out\_ser* | out | *Выходной бит данных сериализатора (тип данных: std\_logic).* |

Таблица 2.2. — Описание сигналов десериализатора.

| № п.п. | Сигнал | Напр. | Описание |
| --- | --- | --- | --- |
| Входные и выходные порты десериализатора: | | | |
| 1 | *clk\_in\_deser* | in | *Входной тактовый сигнал десериализатора (тип данных: std\_logic).* |
| 2 | *reset\_deser* | in | *Сигнал сброса десериализатора (тип данных: std\_logic).* |
| 3 | *data\_in\_deser* | in | *Входной бит данных для десериализации (тип данных: std\_logic).* |
| 4 | *link\_trained* | out | *Сигнал, указывающий, что выравнивание установлено (тип данных: std\_logic).* |
| 5 | *data\_out\_deser* | out | *Выходные данные десериализатора (8-битная последовательность) (тип данных: std\_logic\_vector(7 downto 0)).* |

Условное графическое обозначение изображено на рисунке 2.1.

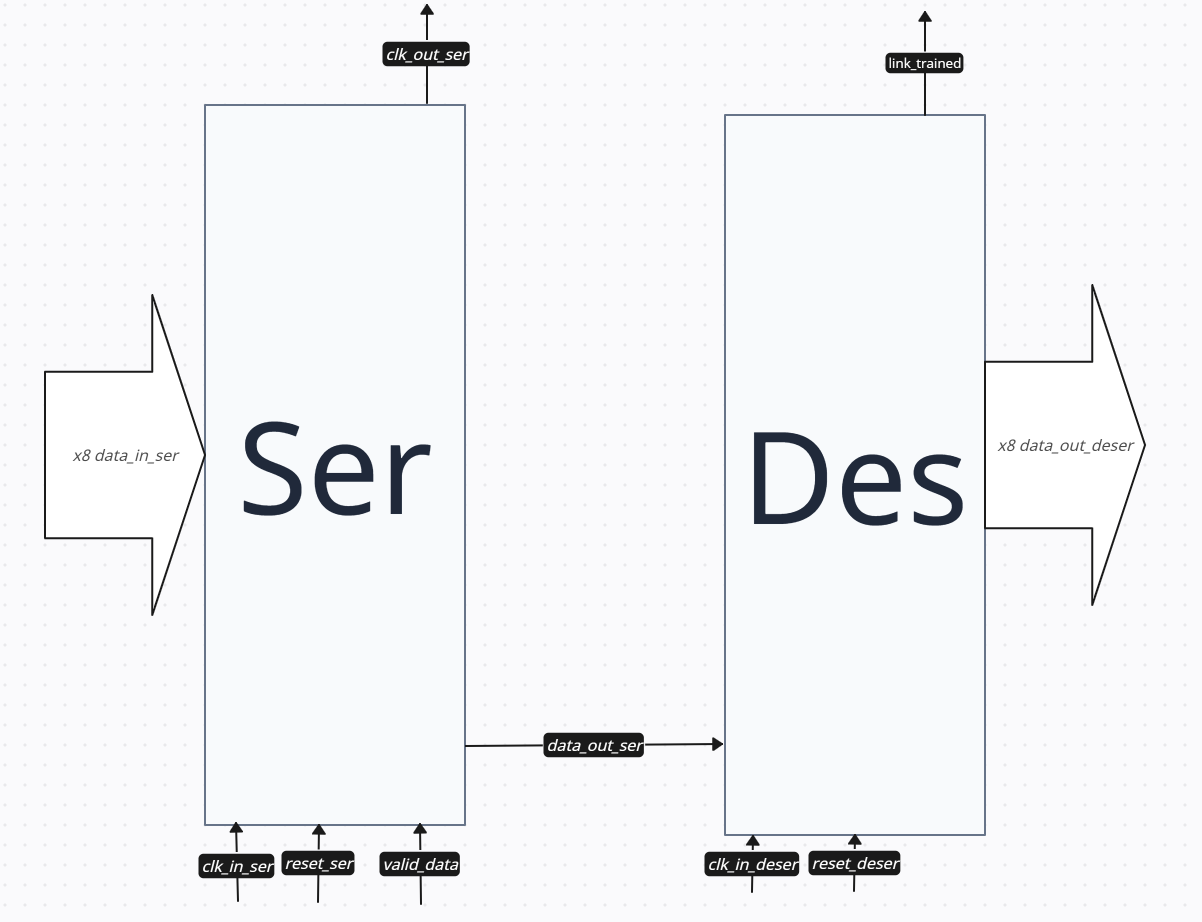


Рис. 2.1. УГО блоков сериализации и десериализации.

## 2.2. Режимы работы блока SerDes

Режимы работы для сериализатора представлены в таблице 2.3 и таблице 2.4:

Таблица 2.3. Режимы работы блока сериализации данных.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п.п. | Режим работы | Сигнал | Напр. | Описание |
| 1 | Сброс | reset\_ser = '1' | in | В этом режиме сериализатор находится в состоянии сброса, и выравнивание не установлено. |
| 2 | Режим ожидания | valid\_data = '0' | in | В этом режиме сериализатор передает тестовый паттерн. |
| 3 | Режим передачи валидных данных | valid\_data = '1' | in | В этом режиме прекращается передача тестового паттерна и начинается подача данных. Сериализатор принимает данные, которые поступают на вход data\_in\_ser и сериализует их. Входные данные записываются в регистр, и они будут сериализованы в соответствии с состоянием счетчика сериализатора. |

Таблица 2.4. Режимы работы блока десериализации данных.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п.п. | Режим работы | Состояние сигнала | Напр. | Описание |
| 1 | Сброс | reset\_deser = '1' | in | В этом режиме десериализатор находится в состоянии сброса, и выравнивание не установлено. |
| 2 | Режим ожидания (выравнивание) | data\_in\_deser = «1…data…1» | in | В этом десериализатор принимает данные, которые поступают на вход data\_in\_deser и десериализует их. Крайние биты равные «1» сигнализируют о том, что сериализатор находится в состоянии ожидания. Десериализатор получает тестовый паттерн и производит его выравнивание. |
| 3 | Режим десериализации данных | data\_in\_deser = «0…data…0» | in | В этом режиме десериализатор принимает выравненные данные и десериализует их. |

## 2.3. Тестирование программы

В процессе тестирования модуля SerDes были приняты следующие решения:

*Создание тестовых сценариев.*

На основе требований к модулю и его функциональности были разработаны тестовые сценарии, которые определяли порядок проведения тестов, проверяемые параметры и ожидаемые результаты.

*Использование автоматизированных тестов.*

Для ускорения процесса тестирования и повышения его эффективности было принято решение использовать автоматизированные тесты. Они позволяли производить проверку модуля с большой скоростью, а также значительно упрощали процесс сбора и анализа результатов.

*Анализ полученных данных.*

После проведения серии тестов и получения результатов, было принято решение провести статистический анализ собранных данных. Это позволило определить правильность работы модуля.

*Проведение тестирования.*

Для проверки стабильности работы после внесенных изменений было проведено тестирование, которое подтвердило эффективность решений принятых для реализации блоков устройства.

*Валидация результатов.*

По окончании тестирования была проведена валидация полученных результатов и проверка их соответствия требованиям и спецификациям.

Таким образом, благодаря использованию автоматизированных тестов и проведению анализа результатов, удалось обеспечить устойчивую работу модуля SerDes и установить соответствие заявленным параметрам.

## 2.4. Временные диаграммы

На рисунках ниже продемонстрирована работа модуля SerDes.

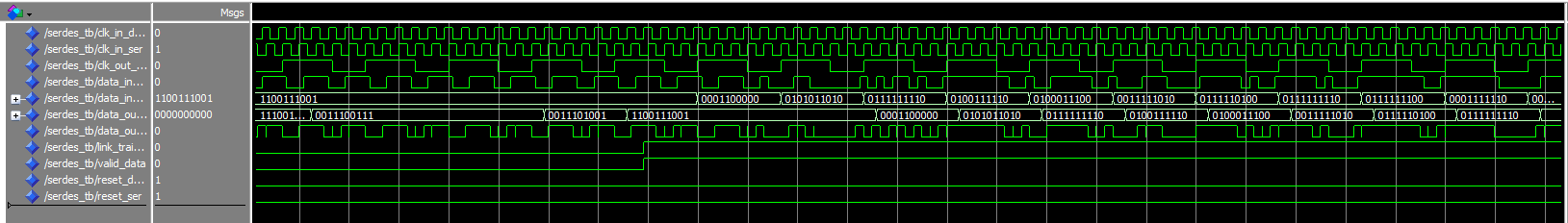


Рис. 2.2. Входные и выходные сигналы блоков.

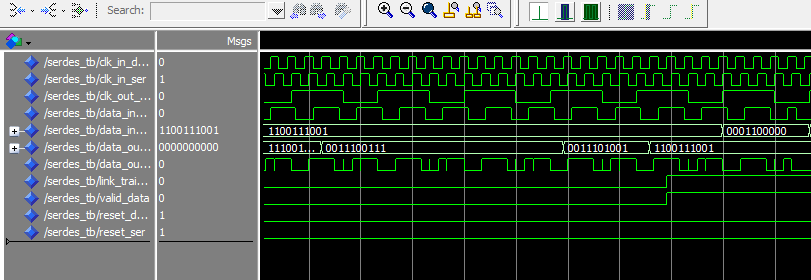


Рис. 2.3. Процесс генерации входных тактовых сигналов для блоков SerDes.

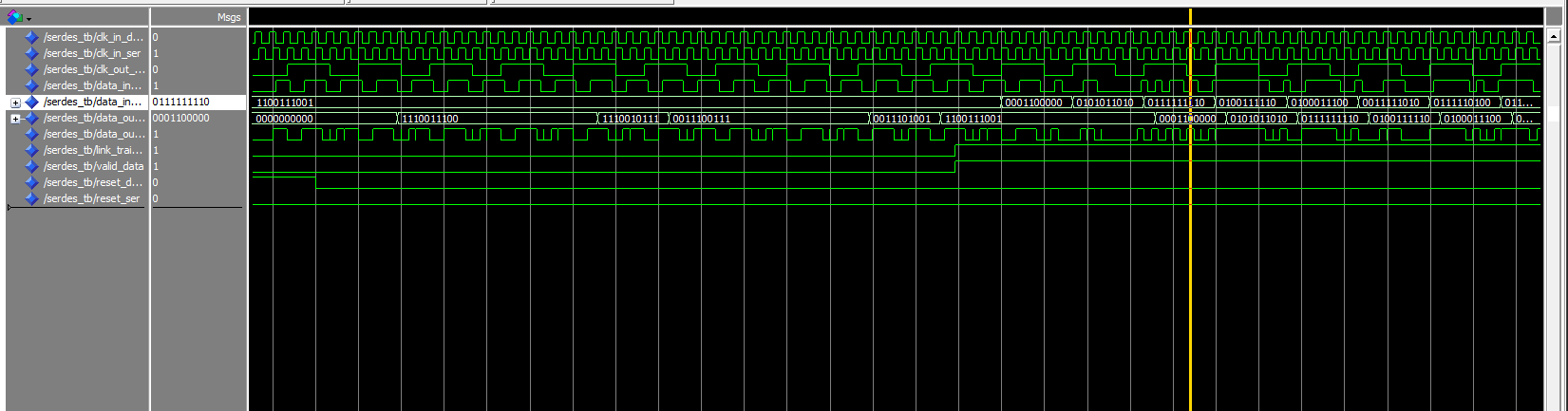


Рис. 2.4. Отправка тестового паттерна и его выравнивание с последующей отправкой валидных данных

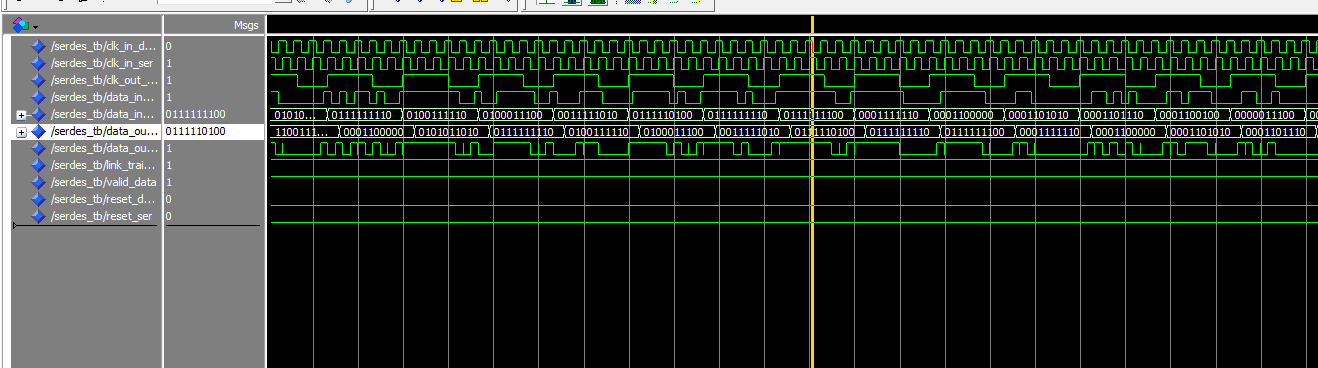


Рис. 2.5. Передача и прием валидных данных.

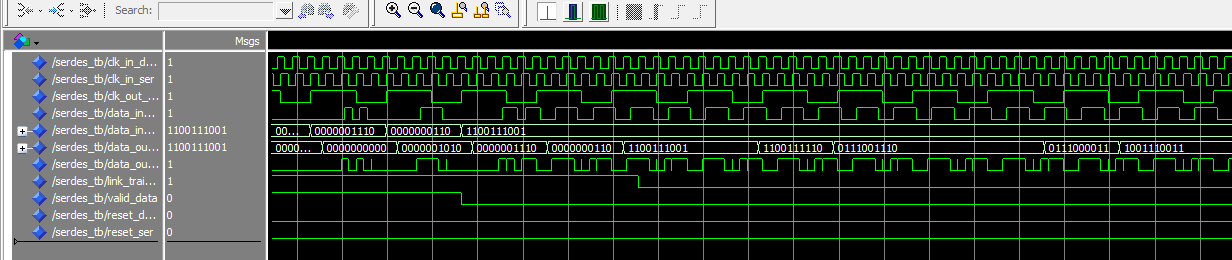


Рис. 2.6. Окончание передачи валидных данных с последующей подачей тестового паттерна.

## 2.5. Синтез

На рис. 2.7. изображена RTL-схема модуля SerDes.

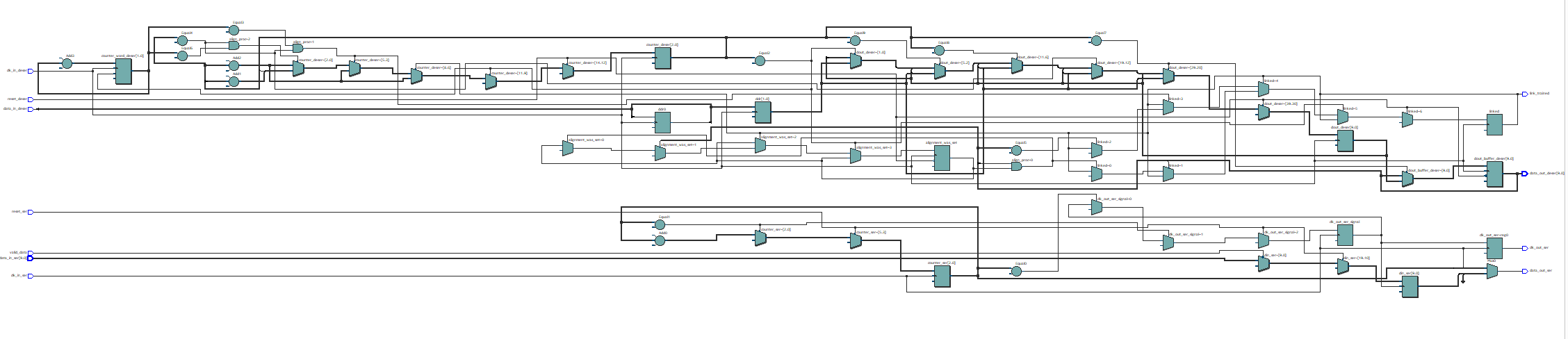


Рис.2.7 RTL-схема блока SerDes.

# ЗАКЛЮЧЕНИЕ

В ходе курсовой работы был разработаны модули сериализации и десериализации SerDes с функцией выравнивания на языке VHDL. Работа модулей соответствует заявленной спецификации.

В ходе работы были получены навыки работы с языком описания оборудования VHDL, приобретен опыт работы с программным обеспечением Quartus Prime Standart, ModelSim и опыт работы в команде.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. SerDes [Электронный ресурс]. — Режим доступа: Интернет: https://ru.wikipedia.org/wiki/SerDes (дата обращения: 01.10.2023)
2. ModelSim - Altera Edition [Электронный ресурс]. — Режим доступа: Интернет: http://altera.ru/soft\_modelsim-ae.html (дата обращения: 29.09.2023)
3. FPGA Design Software - Intel® Quartus® Prime [Электронный ресурс]. — Режим доступа: Интернет: https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html. (дата обращения: 29.09.2023)
4. VHDL [Электронный ресурс]. — Режим доступа: Интернет: https://ru.wikipedia.org/wiki/VHDL (дата обращения: 29.09.2023)

# Приложение 1

## Код блоков SerDes

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

entity SerDes is

Port (

clk\_in\_ser : in std\_logic := '0';

reset\_ser : in std\_logic := '0';

valid\_data : in std\_logic := '0'; -- '1'-передаются валидные данные, '0'-тестовый паттерн

data\_in\_ser : in std\_logic\_vector(9 downto 0) := (others => '0');

data\_out\_ser : out std\_logic := '0';

clk\_out\_ser : out std\_logic := '0';

clk\_in\_deser : in std\_logic := '0';

reset\_deser : in std\_logic := '0';

link\_trained : out std\_logic := '0'; -- '1' когда выравнивание установлено

data\_in\_deser : in std\_logic := '0';

data\_out\_deser : out std\_logic\_vector(9 downto 0) := (others => '0')

);

end SerDes;

architecture Behavioral of SerDes is

signal data\_selection\_ser : std\_logic\_vector(3 downto 0);

signal counter\_ser : std\_logic\_vector(2 downto 0) := (others => '0');

signal din\_ser : std\_logic\_vector(9 downto 0) := (others => '0');

signal clk\_out\_ser\_signal : std\_logic := '0';

signal dout\_deser : std\_logic\_vector(9 downto 0) := (others => '0');

signal dout\_buffer\_deser : std\_logic\_vector(9 downto 0) := (others => '0');

signal counter\_deser : std\_logic\_vector(2 downto 0) := (others => '0');

signal data\_selection\_deser : std\_logic\_vector(2 downto 0) := (others => '0');

signal counter\_word\_deser : std\_logic\_vector(1 downto 0) := (others => '0');

signal linked : std\_logic := '0';

signal alignment\_was\_set : std\_logic := '0';

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

signal ddr0 : std\_logic;

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

signal test\_pattern : std\_logic\_vector(9 downto 0) := "1100111001";

begin

----------------------------Сериализатор------------------------------------

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

-- Генерация выходного тактового сигнала сериализатора

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

clk\_out\_deser\_proc : process (clk\_in\_ser)

begin

if rising\_edge(clk\_in\_ser) then

if reset\_ser = '1' then

clk\_out\_ser\_signal <= '0';

else

if counter\_ser = "100" then

clk\_out\_ser\_signal <= '1';

elsif counter\_ser = "010" then

clk\_out\_ser\_signal <= '0';

end if;

end if;

clk\_out\_ser <= clk\_out\_ser\_signal;

end if;

end process;

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

-- Счетчик сериализатора

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

counter\_ser\_proc : process (clk\_in\_ser)

begin

if rising\_edge(clk\_in\_ser) then

if reset\_ser = '1' then

counter\_ser <= (others => '0');

else

if counter\_ser = "100" then

counter\_ser <= (others => '0');

else

counter\_ser <= counter\_ser + 1;

end if;

end if;

end if;

end process;

data\_selection\_ser <= counter\_ser & (not clk\_in\_ser);

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

-- Прием данных сериализатором/отправка тестового паттерна

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

process (clk\_out\_ser\_signal)

begin

if rising\_edge(clk\_out\_ser\_signal) then

if reset\_ser = '1' then

din\_ser <= (others => '0');

else

if valid\_data = '0' then

din\_ser <= test\_pattern;

else

din\_ser <= data\_in\_ser;

end if;

end if;

end if;

end process;

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

-- Сериализация данных

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

serializing\_proc : process (data\_selection\_ser, din\_ser)

begin

case data\_selection\_ser is

when"0000" => data\_out\_ser <= din\_ser(9);

when"0001" => data\_out\_ser <= din\_ser(8);

when"0010" => data\_out\_ser <= din\_ser(7);

when"0011" => data\_out\_ser <= din\_ser(6);

when"0100" => data\_out\_ser <= din\_ser(5);

when"0101" => data\_out\_ser <= din\_ser(4);

when"0110" => data\_out\_ser <= din\_ser(3);

when"0111" => data\_out\_ser <= din\_ser(2);

when"1000" => data\_out\_ser <= din\_ser(1);

when"1001" => data\_out\_ser <= din\_ser(0);

when others => data\_out\_ser <= '0';

end case;

end process;

---------------------------Десериализатор-----------------------------------

link\_trained <= linked;

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

-- Прием данных десериализатором

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

ddr0 <= data\_in\_deser when falling\_edge(clk\_in\_deser);

ddr(1) <= ddr0 when rising\_edge(clk\_in\_deser);

ddr(0) <= data\_in\_deser when rising\_edge(clk\_in\_deser);

din\_deser <= ddr;

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

-- Выравнивание данных

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

align\_proc : process (clk\_in\_deser)

begin

if rising\_edge(clk\_in\_deser) then

if reset\_deser = '1' then

counter\_deser <= "000";

linked <= '0';

else

if counter\_deser = "100" then

counter\_deser <= "000";

else

if linked = '1' then

counter\_deser <= counter\_deser + 1;

if dout\_buffer\_deser(9) = '0' then

alignment\_was\_set <= '1';

elsif dout\_buffer\_deser(9) = '1' and alignment\_was\_set = '1' then

linked <= '0';

alignment\_was\_set <= '0';

end if;

else

if counter\_word\_deser = "01" and counter\_deser = "001" then

if dout\_buffer\_deser = test\_pattern then

linked <= '1';

else

linked <= linked;

end if;

counter\_deser <= counter\_deser + 1;

elsif counter\_word\_deser = "11" and counter\_deser = "001" then

counter\_deser <= counter\_deser + 2;

else

counter\_deser <= counter\_deser + 1;

end if;

end if;

end if;

end if;

end if;

end process;

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

-- Счетчик записанных комбинаций

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

word\_counter\_proc : process (clk\_in\_deser)

begin

if rising\_edge(clk\_in\_deser) then

if counter\_deser = "100" then

counter\_word\_deser <= counter\_word\_deser + 1;

end if;

end if;

end process;

data\_selection\_deser <= counter\_deser;

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

-- Десериализация

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

deserializing\_proc : process (clk\_in\_deser)

begin

if rising\_edge(clk\_in\_deser) then

if reset\_deser = '1' then

dout\_deser <= (others => '0');

else

if data\_selection\_deser = "000" then

dout\_deser(9 downto 8) <= din\_deser;

dout\_buffer\_deser <= dout\_deser;

elsif data\_selection\_deser = "001" then

dout\_deser(7 downto 6) <= din\_deser;

elsif data\_selection\_deser = "010" then

dout\_deser(5 downto 4) <= din\_deser;

elsif data\_selection\_deser = "011" then

dout\_deser(3 downto 2) <= din\_deser;

elsif data\_selection\_deser = "100" then

dout\_deser(1 downto 0) <= din\_deser;

end if;

end if;

end if;

end process;

data\_out\_deser <= dout\_buffer\_deser;

end Behavioral;

# Приложение 2

## Тестер к блоку SerDes

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.numeric\_std.all;

use work.SerDes;

use work.SerDes\_tester;

entity SerDes\_tb is

end SerDes\_tb;

architecture struct of SerDes\_tb is

-- signals

SIGNAL clk\_in\_deser : STD\_LOGIC := '0';

SIGNAL clk\_in\_ser : STD\_LOGIC := '0';

SIGNAL clk\_out\_ser : std\_logic := '0';

SIGNAL data\_in\_deser : STD\_LOGIC := '0';

SIGNAL data\_in\_ser : STD\_LOGIC\_VECTOR(9 DOWNTO 0) := "1100111001";

SIGNAL data\_out\_deser : STD\_LOGIC\_VECTOR(9 DOWNTO 0) := (others => '0');

SIGNAL data\_out\_ser : STD\_LOGIC := '0';

SIGNAL link\_trained : STD\_LOGIC := '0';

SIGNAL valid\_data : STD\_LOGIC := '0';

SIGNAL reset\_deser : STD\_LOGIC := '0';

SIGNAL reset\_ser : STD\_LOGIC := '0';

COMPONENT SerDes

PORT (

clk\_in\_deser : IN STD\_LOGIC := '0';

clk\_in\_ser : IN STD\_LOGIC := '0';

clk\_out\_ser : OUT STD\_LOGIC := '0';

data\_in\_deser : IN STD\_LOGIC := '0';

data\_in\_ser : IN STD\_LOGIC\_VECTOR(9 DOWNTO 0) := (others => '0');

data\_out\_deser : OUT STD\_LOGIC\_VECTOR(9 DOWNTO 0) := (others => '0');

data\_out\_ser : OUT STD\_LOGIC := '0';

link\_trained : OUT STD\_LOGIC := '0';

valid\_data : IN STD\_LOGIC := '0';

reset\_deser : IN STD\_LOGIC := '0';

reset\_ser : IN STD\_LOGIC := '0'

);

END COMPONENT;

COMPONENT SerDes\_tester

PORT (

clk\_in\_deser : OUT STD\_LOGIC := '0';

clk\_in\_ser : OUT STD\_LOGIC := '0';

clk\_out\_ser : IN STD\_LOGIC := '0';

data\_in\_deser : OUT STD\_LOGIC := '0';

data\_in\_ser : OUT STD\_LOGIC\_VECTOR(9 DOWNTO 0) := (others => '0');

data\_out\_deser : IN STD\_LOGIC\_VECTOR(9 DOWNTO 0) := (others => '0');

data\_out\_ser : IN STD\_LOGIC := '0';

link\_trained : IN STD\_LOGIC := '0';

valid\_data : OUT STD\_LOGIC := '0';

reset\_deser : OUT STD\_LOGIC := '0';

reset\_ser : OUT STD\_LOGIC := '0'

);

END COMPONENT;

--FOR ALL : SerDes USE ENTITY my\_project1.SerDes;

--FOR ALL : SerDes\_tester USE ENTITY my\_project1.SerDes\_tester;

BEGIN

U\_0 : entity work.SerDes

PORT MAP (

clk\_in\_deser => clk\_in\_deser,

clk\_in\_ser => clk\_in\_ser,

clk\_out\_ser => clk\_out\_ser,

data\_in\_deser => data\_in\_deser,

data\_in\_ser => data\_in\_ser,

data\_out\_deser => data\_out\_deser,

data\_out\_ser => data\_out\_ser,

link\_trained => link\_trained,

valid\_data => valid\_data,

reset\_deser => reset\_deser,

reset\_ser => reset\_ser

);

U\_1 : entity work.SerDes\_tester

PORT MAP (

clk\_in\_deser => clk\_in\_deser,

clk\_in\_ser => clk\_in\_ser,

clk\_out\_ser => clk\_out\_ser,

data\_in\_deser => data\_in\_deser,

data\_in\_ser => data\_in\_ser,

data\_out\_deser => data\_out\_deser,

data\_out\_ser => data\_out\_ser,

link\_trained => link\_trained,

valid\_data => valid\_data,

reset\_deser => reset\_deser,

reset\_ser => reset\_ser

);

END struct;

# Приложение 3

## Тестовые данные

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

entity SerDes\_tester is

Port (

clk\_in\_ser : out std\_logic := '0';

reset\_ser : out std\_logic := '0';

valid\_data : out std\_logic := '0'; -- '1'-передаются валидные данные, '0'-тестовый паттерн

data\_in\_ser : out std\_logic\_vector(9 downto 0) := "1100111001";

data\_out\_ser : in std\_logic := '0';

clk\_out\_ser : in std\_logic := '0';

clk\_in\_deser : out std\_logic := '0';

reset\_deser : out std\_logic := '0';

link\_trained : in std\_logic := '0'; -- '1' когда выравнивание установлено

data\_in\_deser : out std\_logic := '0';

data\_out\_deser : in std\_logic\_vector(9 downto 0) := (others => '0')

);

end SerDes\_tester;

ARCHITECTURE flow OF SerDes\_tester IS

-- constants

constant clk\_in\_ser\_period : time := 3.3333 ns;

constant test\_pattern : std\_logic\_vector(9 downto 0) := "1100111001";

BEGIN

clk\_gen\_proc : process

begin

clk\_in\_ser <= '1';

data\_in\_deser <= data\_out\_ser;

wait for clk\_in\_ser\_period\*2/8;

clk\_in\_deser <= '0';

data\_in\_deser <= data\_out\_ser;

wait for clk\_in\_ser\_period\*2/8;

clk\_in\_ser <= '0';

data\_in\_deser <= data\_out\_ser;

wait for clk\_in\_ser\_period\*2/8;

clk\_in\_deser <= '1';

data\_in\_deser <= data\_out\_ser;

wait for clk\_in\_ser\_period\*2/8;

end process;

data\_gen\_proc : process

begin

reset\_deser <= '1';

reset\_ser <= '1';

wait for 35 ns;

reset\_ser <= '0';

wait for 25 ns;

reset\_deser <= '0';

wait until link\_trained = '1';

--wait for 100 ns;

valid\_data <= '1';

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0001100000";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0101011010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0111111110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0100111110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0100011100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0011111010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0111110100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0111111110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0111111100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0001111110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0001100000";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0001101010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0001101110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0001100100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000001010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000010100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000000000";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000001010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000001110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000000110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000001010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000010100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011100";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000011110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000000000";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000001010";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000001110";

wait until rising\_edge(clk\_out\_ser);

data\_in\_ser <= "0000000110";

wait until rising\_edge(clk\_out\_ser);

valid\_data <= '0';

data\_in\_ser <= test\_pattern;

wait;

end process;

END flow;