

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт
Факультет информационных
технологий и электроники

Кафедра
«Информационная безопасность
систем и технологий»

Специальность 10.05.03—

Специализация -

Информационная безопасность
автоматизированных систем
Защищенные автоматизированные
системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРОГРАММНЫЙ МОДУЛЬ ВЫСОКОСКОРОСТНОГО ИНТЕРФЕЙСА
ОБРАБОТКИ ДАННЫХ НА ПЛИС В СОСТАВЕ СКЗИ

Студент

Аса

Асаян Артем Валдимович

Руководитель инженер 2 категории АО «ПНИЭИ»

Миляйкина Ю.Ю.

Нормоконтролёр

Иванов А.П.

Рецензент зам. ген. директора АО «ПНИЭИ»

Сомкин С.А.



Работа допущена к защите (протокол заседания кафедры от 16.06.23 № 10)

Заведующий кафедрой

С.Л. Зефирова

Работа защищена с отметкой

отлично

(протокол заседания ГЭК

от 27.06.23 № 1)

Секретарь ГЭК

А.П. Иванов

А.П. Иванов

Подпись И.Л. Орлова удостоверяю

Ведущий инженер И.Л. Орлова



Обозначения и сокращения

ООД	– окончное оборудование данных
ПЛИС	– программируемая логическая интегральная схема
СКЗИ	– средство криптографической защиты информации
КМ	– канальный модуль
ОЗУ	– оперативное запоминающее устройство
UART	– universal asynchronous receiver-transmitter
CMT	– clock management tile
КМОП	– комплементарная структура металл-оксид-полупроводник
CRC	– cyclic redundancy check

Содержание

Введение	4
1 Обзор существующих реализаций интерфейса UART.....	6
1.1 Общее описание интерфейса UART и языка VHDL	6
1.2 Вариант №1 реализации интерфейса UART	7
1.3 Вариант №2 реализации интерфейса UART	10
1.4 Вариант №3 реализации интерфейса UART	14
2 Общая структура изделия и ПЛИС интерфейса	18
3 Разработка структуры процессов приема и передачи информации	25
3.1 Разработка структуры процесса приема информации.....	25
3.2 Разработка структуры процесса передачи информации	30
4 Внутренний протокол обмена данными	33
5 Разработка интерфейса на языке VHDL	40
6 Проверка работы интерфейса в среде моделирования ISim	45
7 Проверка быстродействия интерфейса	50
8 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие	56
Заключение	61
Список использованных источников	62
Приложение А. Обязательное. Код узла приема данных на языке vhd1	64
Приложение Б. Обязательное. Код узла расчёта контрольной суммы на языке vhd1	73
Приложение В. Обязательное. Код узла генерации частот на языке vhd1	87
Приложение Г. Обязательное. Код узла передачи данных на языке vhd1.....	92
Приложение Д. Обязательное. Код узла переложения бита и генератора частоты 128 МГц на языке vhd1.....	95
Приложение Е. Обязательное. Код узла обработки принятых данных на языке vhd1	101

Введение

Защита критически важной информации в современном мире является одной из наиболее важных задач, особенно в условиях постоянного роста количества возможных атак и возможностей потенциальных нарушителей. Для обеспечения конфиденциальности защищаемой информация в настоящее время используются СКЗИ.

В последние несколько лет СКЗИ получили широкое распространение в различных государственных, муниципальных и коммерческих организациях, учреждениях и структурах, что связано, как с ростом количества угроз безопасности информации, так и с принятием на территории Российской Федерации различных нормативно-правовых актов в области защиты информации, обязывающих подобные организации и учреждения применять СКЗИ для обеспечения безопасности защищаемой информации, такой как государственная тайна или персональные данные граждан. Современные СКЗИ должны обеспечивать обработку огромного количества информации, с как можно более высокой скоростью, а также обладать высокой степенью надёжности. Программная реализация СКЗИ не обеспечивает необходимой скорости преобразования информации, и поэтому широко применяются СКЗИ на основе аппаратной реализации, то есть на ПЛИС.

При проектировании и разработке СКЗИ на базе ПЛИС критически важным является выбор интерфейсов передачи данных, используемых как для обмена информацией внутри самого изделия, так и для взаимодействия с внешним миром. Для таких устройств особенно важны скорость и надёжность интерфейса передачи данных, ведь изменение всего одного бита данных при обмене зашифрованной информацией может привести к потере целого пакета. Также важна эффективность интерфейса с точки зрения использования ресурсов ПЛИС.

От выбора наиболее предпочтительного в поставленных условиях интерфейса передачи данных будет зависеть стабильность, быстродействие и удобство разработки устройства на базе ПЛИС.

Целью данного дипломного проекта является разработка высокоскоростного программного модуля интерфейса обработки данных на программной логической интегральной схеме. Разрабатываемое программный модуль предназначен для использования в составе средства криптографической защиты информации, не составляющей государственную тайну. СКЗИ, в состав которого входит модуль, предназначено для криптографической защиты информации, не составляющей государственную тайну.

В ходе выполнения дипломного проекта было необходимо рассмотреть следующие вопросы:

- обзор существующих реализаций интерфейса UART;
- описание общей структуры изделия и ПЛИС интерфейса;
- разработка процессов приема и передачи информации;
- разработка модуля обработки потока информации на языке VHDL;
- моделирование работы модуля обработки информации;
- проверка работы модуля обработки информации;
- вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие.

1 Обзор существующих реализаций интерфейса UART

1.1 Общее описание интерфейса UART и языка VHDL

Интерфейс UART является одним из самых популярных интерфейсов в электронике и широко применяется в различных областях. Основными преимуществами данного интерфейса являются его простота, надёжность и экономичность [1]. Для реализации UART требуется минимальное количество ресурсов ПЛИС. Также интерфейс является экономичным с точки зрения потребления энергии.

В современном мире существует огромное множество различных реализаций интерфейса UART. Они отличаются различными скоростями передачи данных, форматами и размерами посылки, наличием или отсутствием бита четности, количеством стоп-бит. Некоторые реализации обладают возможностью настройки данных параметров.

В разрабатываемом программном модуле было решено использовать формат посылки UART, представленный на рисунке 1.

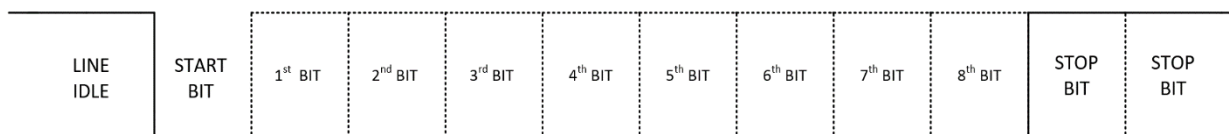


Рисунок 1 – Формат передаваемого кадра в канале связи

Посылка UART, представленная на рисунке 1 состоит из 1 стартового бита, равного логическому нулю, 8 информационных бит, а также 1 или 2 стоп-бит, имеющих значение логической единицы.

Рассматриваемые реализации интерфейса UART написаны на языке VHDL и предназначены для работы на ПЛИС. Язык VHDL (Very high speed integrated circuits Hardware Description Language) – это язык описания аппаратного обеспечения. Он позволяет описывать поведение, т.е. алгоритмы функционирования цифровых систем, а также проводить иерархическое функционально-структурное описание систем, имеет ряд средств для

описания параллельных асинхронных процессов, регулярных структур и в то же время имеет все признаки языка программирования высокого уровня – позволяет создавать свои типы данных, имеет широкий набор арифметических и логических операций [2].

1.2 Вариант №1 реализации интерфейса UART

Данная реализация представляет собой UART-передатчик и UART-приёмник, написанные на языке VHDL для передачи/приёма пакетов размером 8 бит с использованием 1 стоп-бита. За данные параметры отвечают значения databytes = 8 и stopbits = 1.

Передатчик UART представляет собой сдвиговый регистр, в который по стробу записываются данные, и сразу же начинается их сдвиг фронтом частоты бод. Для улучшения синхронизации процесса используется буфер, в который предварительно загружаются данные, которые перегружаются в регистр для сдвига, освобождая буфер для следующей записи.

Далее приведено объявление передатчика на языке vhdl:

```
entity txmit is
  generic
  (
    -- количество бит данных (5,6,7,8)
    databytes: integer := 8;
    -- количество стоп бит (1 или 2)
    stopbits: integer := 1
  );
  port
  (
    -- общий сброс
    rst: in std_logic;
    -- частота бод *16
    clk16x: in std_logic;
    -- загрузка данных в буфер на передачу (falling edge)
```

```

wrn: in std_logic;
-- входные данные
din: in std_logic_vector (databytes - 1 downto 0) ;
-- буфер передатчика пуст
tbre: out std_logic;
-- выход передатчика
txd: out std_logic;
-- выход сигнала rts (для rs485)
rts: out std_logic
);
end txmit;

```

Работа UART приемника начинается по приходу импульса "старт", который формирует разрешение на счетчик, работающий на частоте бод умноженной на 16. Данная частота в 16 раз выше частоты передачи, используемой интерфейсом, она необходима для правильного определения, поступающего из канала бита. Данная схема работает следующим образом:

- ожидается перепад сигнала с «1» до «0», т.е. стартовый бит;
- начинается отсчет 8 тактов частотой бод*16;
- считается, что, когда счетчик достигает значения 7 – это середина стартового бита, т.е. прошла половина времени передачи одного бита;
- счетчик сбрасывается, отсчет начинается заново;
- считается, что момент, когда проходит 16 новых тактов и значение счетчика достигает 15 совпадает с моментом, когда проходит половина интервала передачи первого бита данных;
- полученный в этот момент бит записывается в принимающий регистр;
- данная операция выполняется $N-1+M$, где N – количество бит данных, M – количество стоп-бит [3].

На рисунке 2 представлен принцип работы данной схемы.

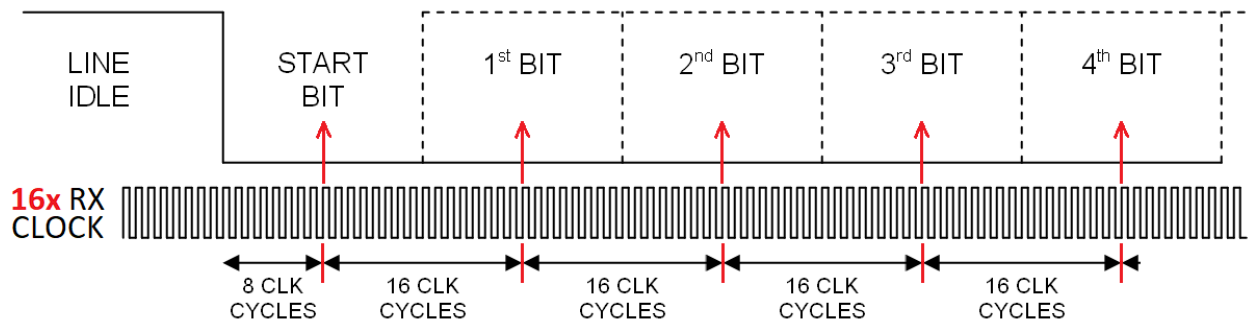


Рисунок 2 – Схема приёма бит

Недостатком данной реализации является то, что в системе должен присутствовать генератор высокой частоты, во много раз превышающий частоту передачи самого интерфейса.

Далее приведено объявление приёмника данной реализации на языке vhdl:

```
entity rcvr is
generic
(
    -- количество бит данных (5,6,7,8)
    --databytes: integer:=8;
    -- количество стоп бит (1 или 2)
    --stopbits: integer:= 1;
    databytes: integer;
    stopbits: integer
);
port
(
    -- общий сброс
    rst: in std_logic;
    -- частота бод*16
    clk16x: in std_logic;
    -- последовательный вход
    rxd: in std_logic;
    -- разрешение чтения данных (1)
```

```

rdn: in std_logic;
-- выходные данные
dout: out std_logic_vector (databytes – 1 downto 0);
-- данные готовы (rising edge)
data_ready: out std_logic
);
End rcvr;

```

Данная реализация подходит для передачи и приёма информации, для которой неправильный приём одного бита данных не несёт серьёзных последствий. Например, видеoinформации или звука. Однако при передаче зашифрованной информации огромное значение имеет надёжность приёмника и передатчика, так как ошибка в приёме всего одного бита зашифрованной информации может привести к потере целого пакета данных. Данная реализация не способна обеспечить достаточную надёжность при приёме бит, так как правильным значением принятого бита считается значение бита в середине времени его передачи. Возможны ситуации, когда время передачи бита, оказывается меньше ожидаемого, в таком случае наличие «коротких» бит в канале связи могут привести к сбою синхронизации и приёму неверной зашифрованной последовательности.

1.3 Вариант №2 реализации интерфейса UART

Данная реализация представляет собой UART-передатчик и UART-приёмник реализованных на языке VHDL для приёма и передачи байт с 2 стоп-битами. Особенностью данной реализации является наличие в UART-приёмнике 4 каналов обработки поступающих бит. Реализация используется в некоторых изделиях АО «ПНИЭИ». Интерфейс предназначен для связи ПЛИС шифратора с интерфейсом Ethernet.

Приёмник в данной реализации ожидает поступления старт-бита. Снятие поступающего бита происходит в 4 канала частотой 72 МГц, сдвинутой по фазе на 0, 90, 180 и 270 градусов.

Схема приема бит показана на рисунке 3.

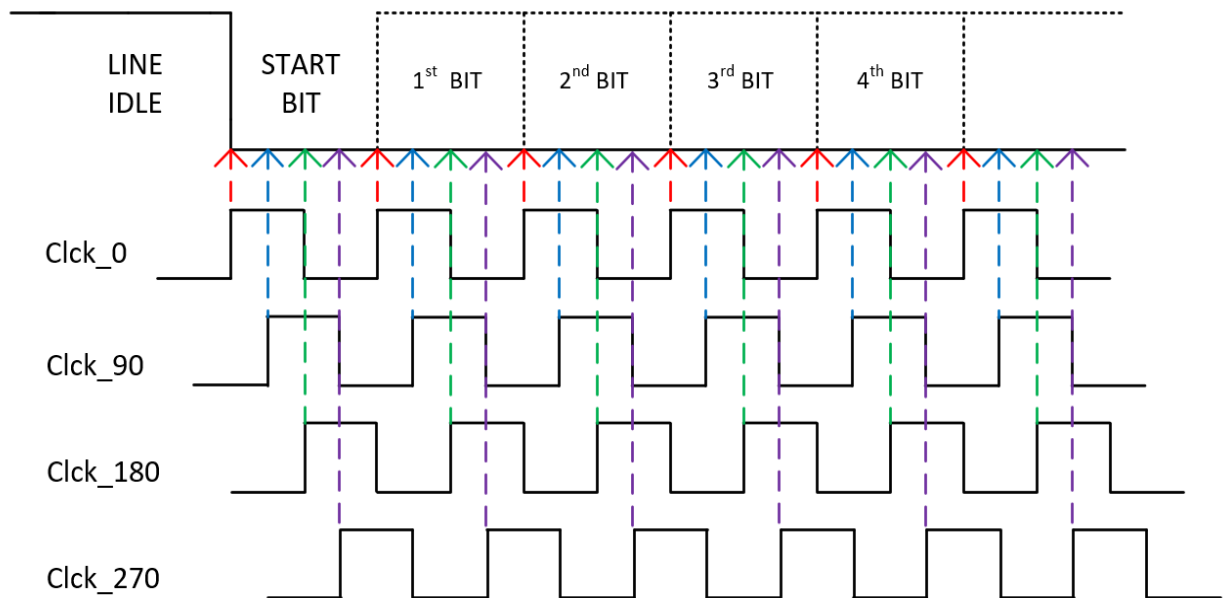


Рисунок 3 – Схема стробирования принимаемых бит

После приёма стартового бита, в каждом канале запускается счетчик, отсчитывающий количество принятых информационных бит. Каждый канал имеет отдельный регистр, в который записываются поступающие информационные биты. Канал, счетчик которого первый досчитывает необходимое количество принятых бит, считается первым принявшим стартовый бит. Информационные биты, которые были записаны в регистр этого канала, являются результатом приёма данных. Все остальные каналы блокируются, данные, записанные в их регистры, не используются.

Код мультиплексора, используемого для выбора канала и регистра:

```

buf_reg_prm<= reg_prm when i_y_prm_byte = '1'
else reg_prm_90 when i_y_prm_byte_90 = '1'
else reg_prm_180 when i_y_prm_byte_180 = '1'
else reg_prm_270;
process (reset, clk_72, i_y_prm_byte_out,buf_reg_prm)
begin
    if reset = '0' then prm_byte_UART<= x"FF"; sbros_blok<= '0';
    elsif clk_72'event and clk_72 = '1' then
        if i_y_prm_byte_out = '1' then
            prm_byte_UART<= buf_reg_prm;

```

```

        sbros_blok<= '1';
    else sbros_blok<= '0'; end if;

    end if;

end process;

i_y_prm_byte_out<= (i_y_prm_byte and blok_0) or
                    (i_y_prm_byte_90 and blok_90) or
                    (i_y_prm_byte_180 and blok_180) or
                    (i_y_prm_byte_270 and blok_270);

i_y_prm_byte_UART <= i_y_prm_byte_out;

```

На рисунке 4 представлена схема мультиплексоров выбора канала.

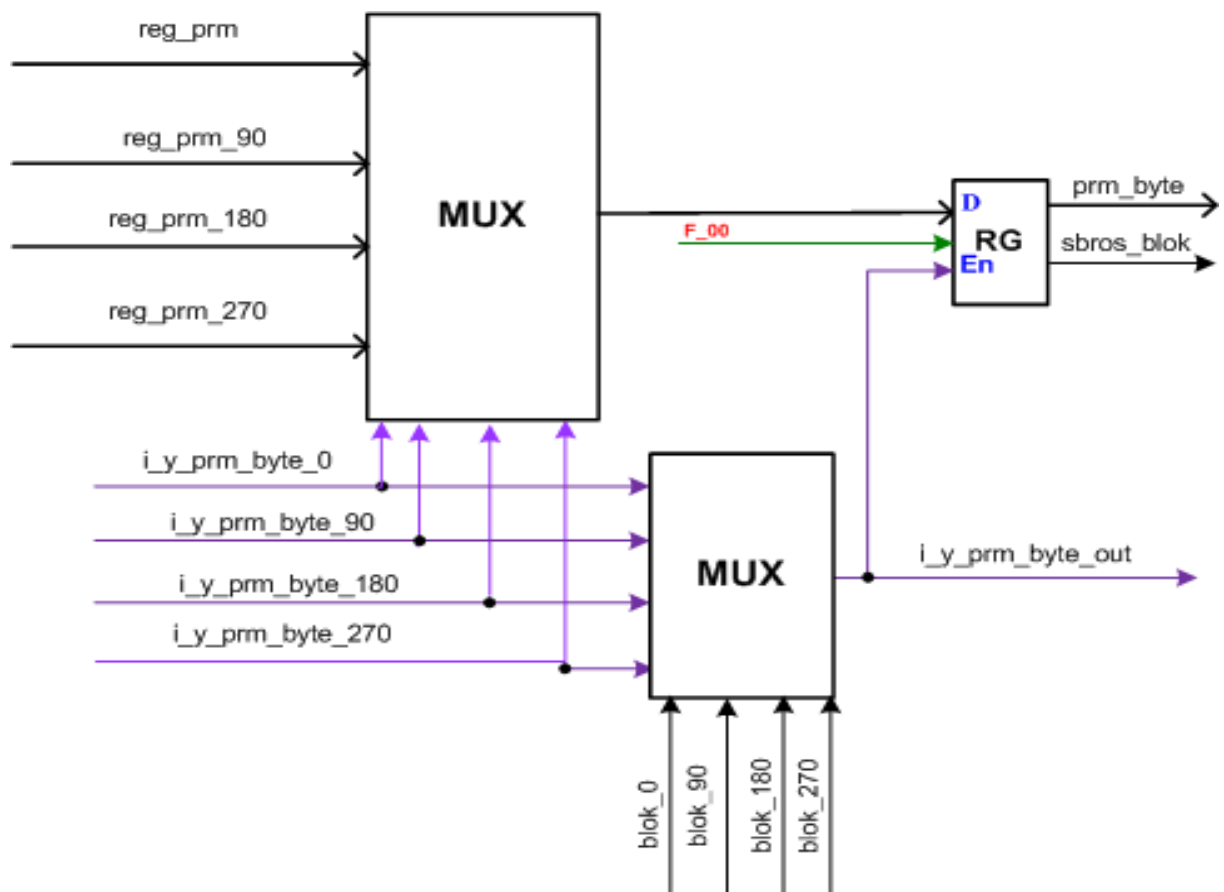


Рисунок 4 – Схема мультиплексоров выбора канала

Сигнал `i_y_prm_byte_0` являются сигналами счетчика, сообщающими о том, что каналом с несдвинутой по фазе частотой было принято 8 информационных бит. Сигналы `i_y_prm_byte_90`, `i_y_prm_byte_180`, `i_y_prm_byte_270` несут ту же информацию, но для каналов с частотой, сдвинутой по фазе на 90, 180 и 270 градусов соответственно.

Сигнал `blok_0`, принимающий значение логической единицы, служит для блокировки всех каналов, кроме канала с исходной частотой. Сигналы `blok_90`, `blok_180`, `blok_270` блокируют все каналы, кроме каналов с частотой, смещенной по фазе на 90, 180, 270 градусов соответственно.

В зависимости от сигналов `i_y_prm_byte` и `blok` выбирается канал, который первым принял 8 бит информации `i_y_prm_byte_out`.

Выходы регистров `reg_prm`, `reg_prm_90`, `reg_prm_180`, `reg_prm_270` передают принятые значения информационных бит из регистров.

В зависимости от значения сигналов `i_y_prm_byte_0`, `i_y_prm_byte_90`, `i_y_prm_byte_180`, `i_y_prm_byte_270`. Так, например, если значение сигнала `i_y_prm_byte_0` равно логической единице, то в буфер принятых данных `buf_reg_prm` записывается значение регистра `reg_prm`.

Из буфера принятых данных информация передается на вход D-триггера, если есть разрешение записи, т.е. сигнал `i_y_prm_byte_out` имеет значение логической единицы. Выходами триггера являются регистр принятых данных `prm_byte` и сигнал сброса блокировок каналов `sbros_block`. Регистр `prm_byte` является выходом узла приёма информации и поступает в узел, в котором происходит дальнейшая обработка принятого байта.

Преимуществом данной реализации является то, что для снятия значения бита, используется одна частота, которую не нужно увеличивать в разы, а достаточно сдвинуть по фазе. Так как приём осуществляется одновременно 4 каналами, вероятность ошибки приёма стартового бита и последующих за ним бит информации гораздо меньше, чем в варианте реализации, рассмотренном в предыдущем пункте.

В данной реализации для хранения информации, полученной из каждого канала, используется отдельный регистр. Это усложняет схему, а также является нерациональным использованием ресурсов, так как в приёмнике может быть 4 одинаково заполненных регистра, из которых выбирается только один. Кроме того, прием и передача бит в данной

реализации работают на частоте 72 МГц, что не обеспечивает высокую скорость передачи данных.

1.4 Вариант №3 реализации интерфейса UART

Был рассмотрен вариант реализации харр224 от компании Xilinx [4]. В данной реализации схема использует тактовую частоту равную частоте потока данных. Однако обычно тактовая частота оказывается либо немного быстрее, либо медленнее потока данных. В схеме используется две частоты: тактовая частота (CLK0) и тактовая частота, сдвинутая по фазе на 90 градусов (CLK90). Сдвиг по фазе может быть реализован с помощью функции фиксированного сдвига фазы или с помощью задержки тактового сигнала на 90 градусов.

Входные данные подаются на 4 триггера, два из которых синхронизируются частотой CLK (один по фронту, один по спаду), а два синхронизируются частотой CLK90 (один по фронту, один по спаду). Данная схема показана на рисунке 5.

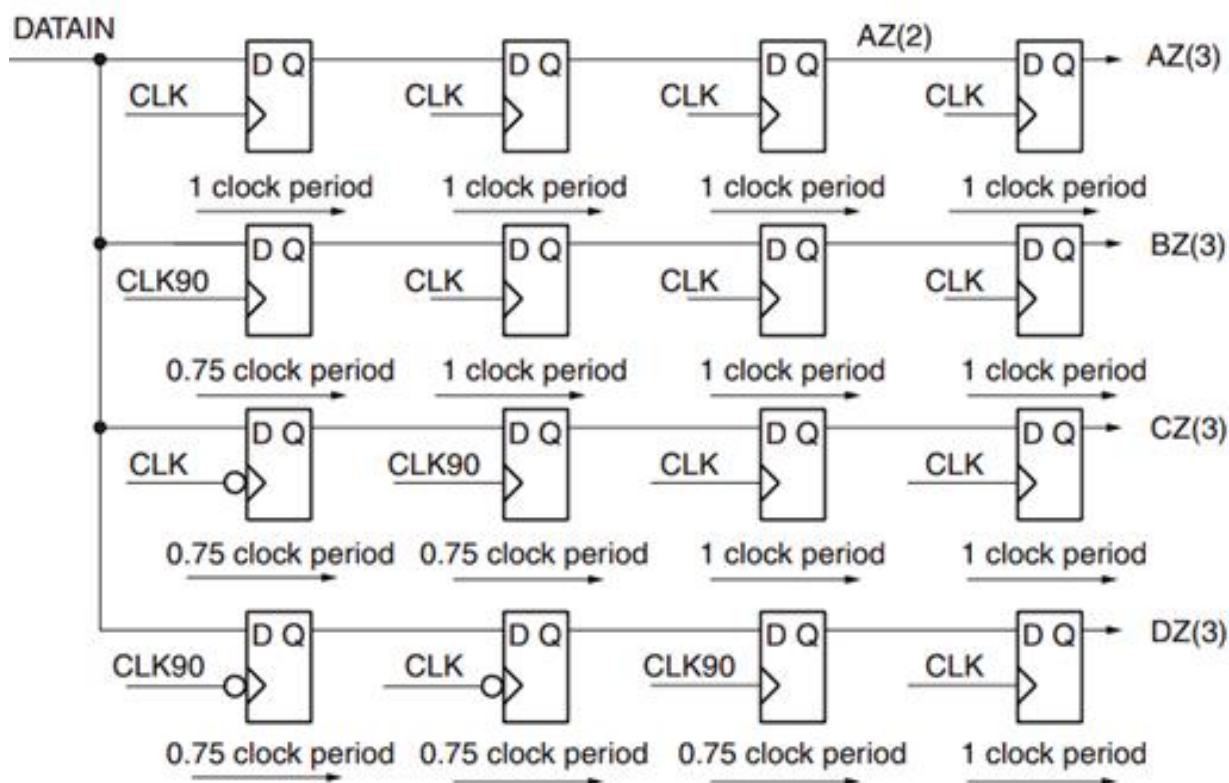


Рисунок 5 –Входной каскад

В данной реализации используется 4 точки выборки разделенных друг от друга на 90 градусов. Поступление данных от входа на триггеры должно быть практически одновременным, задержки должны быть одинаковыми.

Первый триггер синхронизируется на фронте такта, описанного как область времени А. Второй триггер синхронизируется на фронте CLK90 (область времени В), третий триггер синхронизируется на спаде CLK (область времени С), а четвертый синхронизируется на спаде CLK90 (область времени D). Таким образом, если частота тактового сигнала равна, например, 420 МГц, эта логика будет работать на эффективной частоте 1680 МГц.

Полученные точки выборки синхронизируются в несколько этапов для того, чтобы исключить неверные пути логики и устранить проблему метастабильности.

Далее приведен код на языке vhdl, реализующий схему, представленную на рисунке 5:

```
ff_a0: fdc port map (d => data, c => clk, clr => rst, q => az(0));
ff_a1: fdc port map (d => az(0), c => clk, clr => rst, q => az(1));
ff_a2: fdc port map (d => az(1), c => clk, clr => rst, q => az(2));
ff_a3: fdc port map (d => az(2), c => clk, clr => rst, q => az(3));

ff_b0: fdc port map (d => data, c => clk90, clr => rst, q => bz(0));
ff_b1: fdc port map (d => bz(0), c => clk, clr => rst, q => bz(1));
ff_b2: fdc port map (d => bz(1), c => clk, clr => rst, q => bz(2));
ff_b3: fdc port map (d => bz(2), c => clk, clr => rst, q => bz(3));

ff_c0: fdc port map (d => data, c => notclk, clr => rst, q => cz(0));
ff_c1: fdc port map (d => cz(0), c => clk90, clr => rst, q => cz(1));
ff_c2: fdc port map (d => cz(1), c => clk, clr => rst, q => cz(2));
ff_c3: fdc port map (d => cz(2), c => clk, clr => rst, q => cz(3));

ff_d0: fdc port map (d => data, c => notclk90, clr => rst, q => dz(0));
```

ff_d1: fdc port map (d =>dz(0), c =>notclk, clr =>rst, q =>dz(1));

ff_d2: fdc port map (d =>dz(1), c => clk90, clr =>rst, q =>dz(2));

ff_d3: fdc port map (d =>dz(2), c =>clk, clr =>rst, q =>dz(3));

Выбор данных осуществляется с помощью простого мультиплексора, который выбирает биты данных из соответствующей временной области. Для выбора временной области используется схема принятия решений, основанная на распознавании переходов на линии данных. Сигналы AAR до DDP распознают положительные переходы, а сигналы AAN до DDN распознают отрицательные переходы. Условия выбора временной области, следующие:

1. $AAR = BBR = CCP = DDP = 1$, или $AAN = BBN = CCN = DDN = 1$. Область времени А первой обнаружила данные, поэтому выбираются значения битов из области времени С;
2. $AAR = 1$ и $BBR = CCP = DDP = 0$, или $AAN = 1$ и $BBN = CCN = DDN = 0$. Область времени В первой обнаружила данные, поэтому выбираются значения битов из области времени D;
3. $AAR = BBR = 1$ и $CCP = DDP = 0$, или $AAN = BBN = 1$ и $CCN = DDN = 0$. Область времени С первой обнаружила данные, поэтому выбираются значения битов из области времени А;
4. $AAR = BBR = CCP = 1$ и $DDP = 0$, или $AAN = BBN = CCN = 1$ и $DDN = 0$. Область времени D первой обнаружила данные, поэтому выбираются значения битов из области времени В.

По мере того, как тактовый сигнал/данные движутся во времени, мультиплексор может использоваться для выбора данные из одной из четырех доступных временных областей, за исключением двух случаев. Если схема переходит из области А в область D, то передаваемые данные фактически являются недействительными. Если переход происходит из области D в область А, то требуется генерировать два бита данных. Для обозначения недействительности данных или необходимости сгенерировать два бита используется сигнал DV. Если данные недействительны, то он

принимает значения «00», если необходимо сгенерировать два бита, то значение «11».

На рисунке 6 представлена схема мультиплексора для выбора данных.

На представленной схеме AZ(3)-DZ(3) обозначают биты, полученные в каждой временной области, сигналы USEA-USED являются результатом работы схемы принятия решений, основанной на распознавании переходов. Сигнал SDATA(1:0) обозначает выбранные биты, а их действительность указывается сигналом DV(1:0). Первый принятый бит - это SDATA(0). SDATA(1) является вторым принятым битом, но только при условии, что DV имеет значение «11».

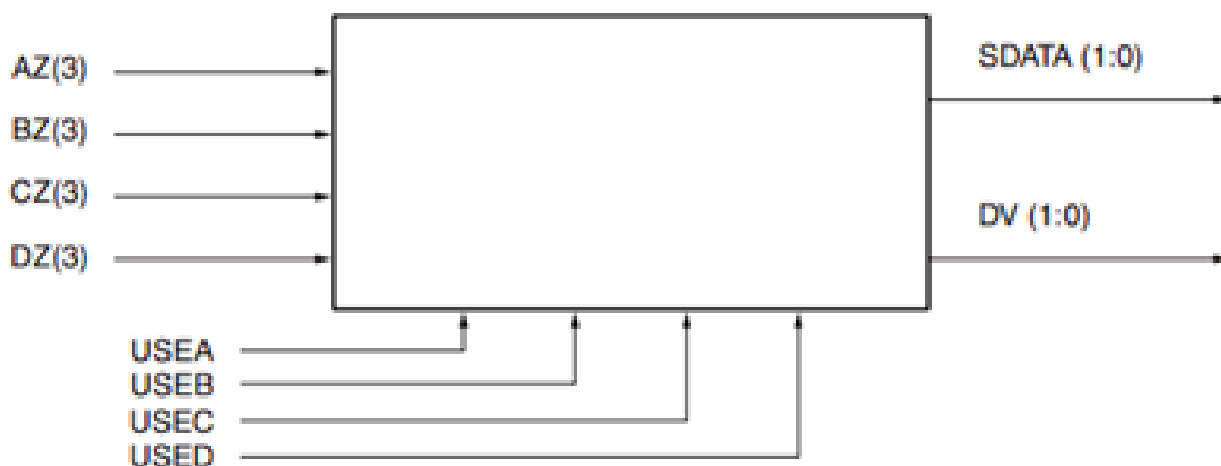


Рисунок 6 – Схема выбора данных

Таким образом, данная реализация является более экономичной, быстрой и простой, чем рассмотренные выше варианты, однако она не лишена недостатков. Недостатком данной реализации является то, что инвертирование исходной частоты и частоты, сдвинутой на 90 градусов добавляет дополнительные задержки, которые могут привести к неправильному считыванию данных, особенно при использовании высоких частот, что является недопустимым при использовании данной схемы в СКЗИ. Также, компания-разработчик Xilinx не рекомендует её для использования в новых проектах.

2 Общая структура изделия и ПЛИС интерфейса

Рассмотренные в предыдущей главе реализации интерфейса UART не подходят для передачи зашифрованной информации, так как имеют существенные недостатки. Было принято решение разработать собственную реализацию интерфейса UART для использования в составе средств криптографического преобразования информации, не составляющей государственную тайну – программный модуль высокоскоростного интерфейса обработки данных. Прототипом для разработки является программный модуль интерфейса обработки данных в изделии криптографической защиты данных и передачи по каналам связи «Аксион (СКЗИ-3)», разработанное в АО «ПНИЭИ». Структура изделия «Аксион (СКЗИ-3)» приведена на рисунке 7.

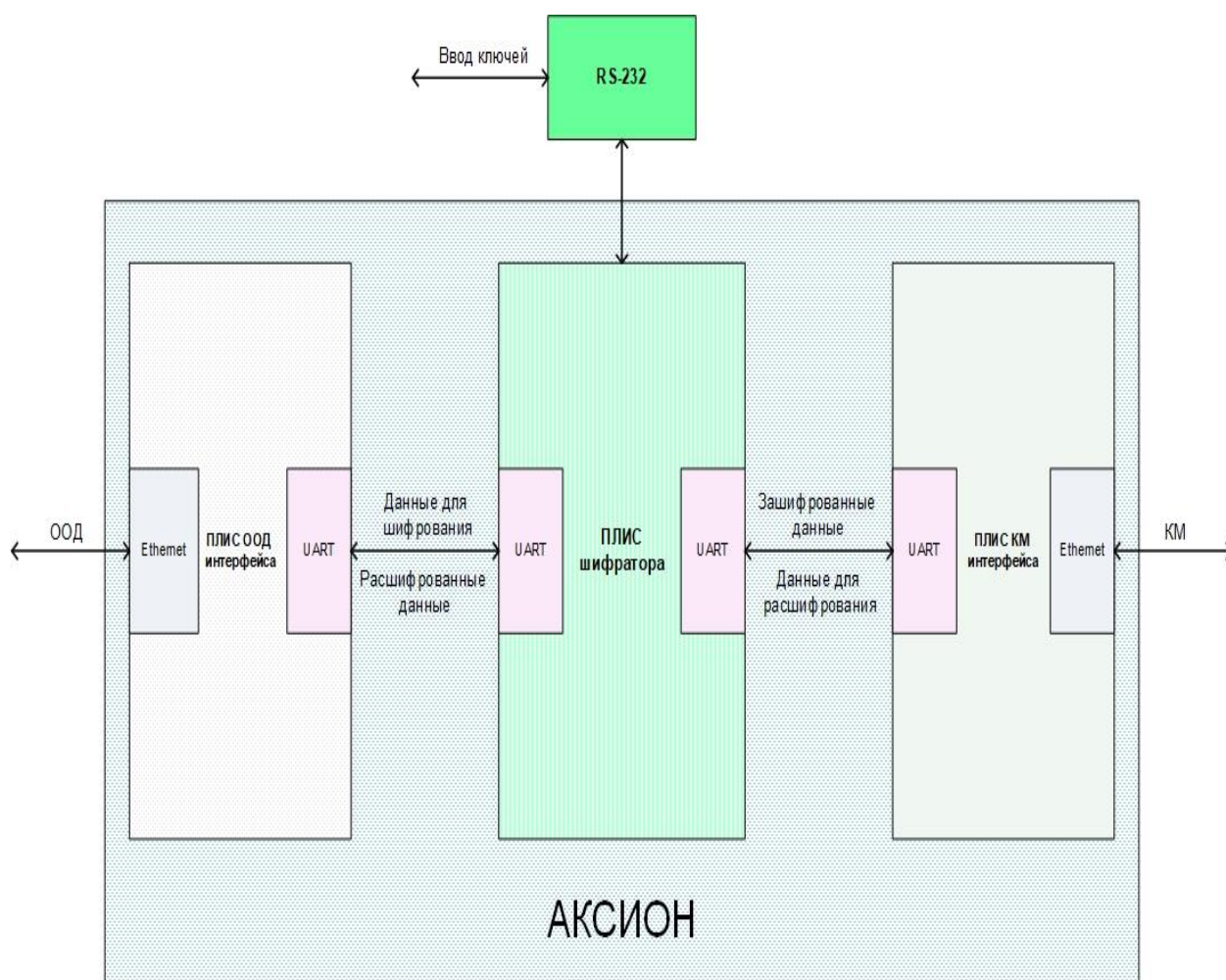


Рисунок 7 – Структура изделия, содержащего модуль обработки информации

Изделие состоит из трех ПЛИС:

- ПЛИС шифратора, выполняющая зашифрование и расшифрование информации;
- ПЛИС интерфейса ООД, передающая информацию из ООД в шифратор и наоборот;
- ПЛИС интерфейса КМ, передающая информацию из КМ в шифратор и наоборот.

Изделие связано с внешним миром по интерфейсу Ethernet, внутренняя связь между ПЛИС осуществляется по интерфейсу UART.

Ethernet — это технология, которая соединяет проводные локальные сети, позволяет устройству взаимодействовать между собой по протоколу, который является общим сетевым языком [5]. Протокол Ethernet имеет множество модификаций: Ethernet, FastEthernet, GigabitEthernet, 10GEthernet. Данные модификации отличаются максимальной скоростью передачи данных. В изделии «Аксион (СКЗИ-3)» используется FastEthernet так как он способен обеспечивать скорость передачи данных 100 Мбит/с.

Данные в сети Ethernet передаются пакетами, называемыми кадрами. Кадр содержит адрес узла, которому он предназначен, адрес отправителя, другие служебные атрибуты, а также сами данные в виде набора битов.

В таблице 1 представлен формат кадра Ethernet.

Таблица 1 – Формат кадра Ethernet.

Содержание поля	Заголовок			Данные	КС
	Адрес получателя	Адрес отправителя	EtherType		
Число байт	6	6	2	46 – 1500	4
	14				

Формат кадра Ethernet включает следующие поля:

- заголовок;
- данные;
- последовательность проверки (контрольная сумма).

Размер поля «Заголовок» составляет 14 байт, оно состоит из:

- MAC – адреса получателя размером 6 байт;
- MAC – адреса отправителя размером 6 байт;
- поля EtherType размером 2 байта.

Поле EtherType указывает на тип протокола, с помощью которого инкапсулированы данные, содержащиеся в полезной нагрузке. В изделии «Аксион (СКЗИ-3)» данным протоколом является протокол IPv4, которому соответствует значение «0800» в шестнадцатеричной системе счисления в поле EtherType.

Поле «Данные» содержит передаваемые данные, в том числе и заголовки протоколов, с помощью которых они были инкапсулированы. Минимальный размер данного поля 46 байт, максимальный размер 1500 байт.

Последовательность проверки представляет собой поле размером 4 байта, содержащее контрольную сумму, вычисленную по алгоритму CRC. Контрольная сумма необходима для контроля целостности переданного кадра.

Скорость физического соединения по стандарту FastEthernet составляет 148 809 кадров в секунду.

В протоколе Ethernet для перед началом каждого кадра передаётся преамбула, представляющая собой последовательность из семи байт, состоящих из чередующихся нулей и единиц. После преамбулы передаётся байт начала кадра, имеющий фиксированное значение «AB» в шестнадцатеричной системе счисления.

Общая скорость работы изделия, измеренная программой iperf составляет 28 Мбит/с.

На рисунке 8 приведена структура ПЛИС интерфейса, в которой будет использоваться разработанный программный модуль высокоскоростного интерфейса обработки данных.

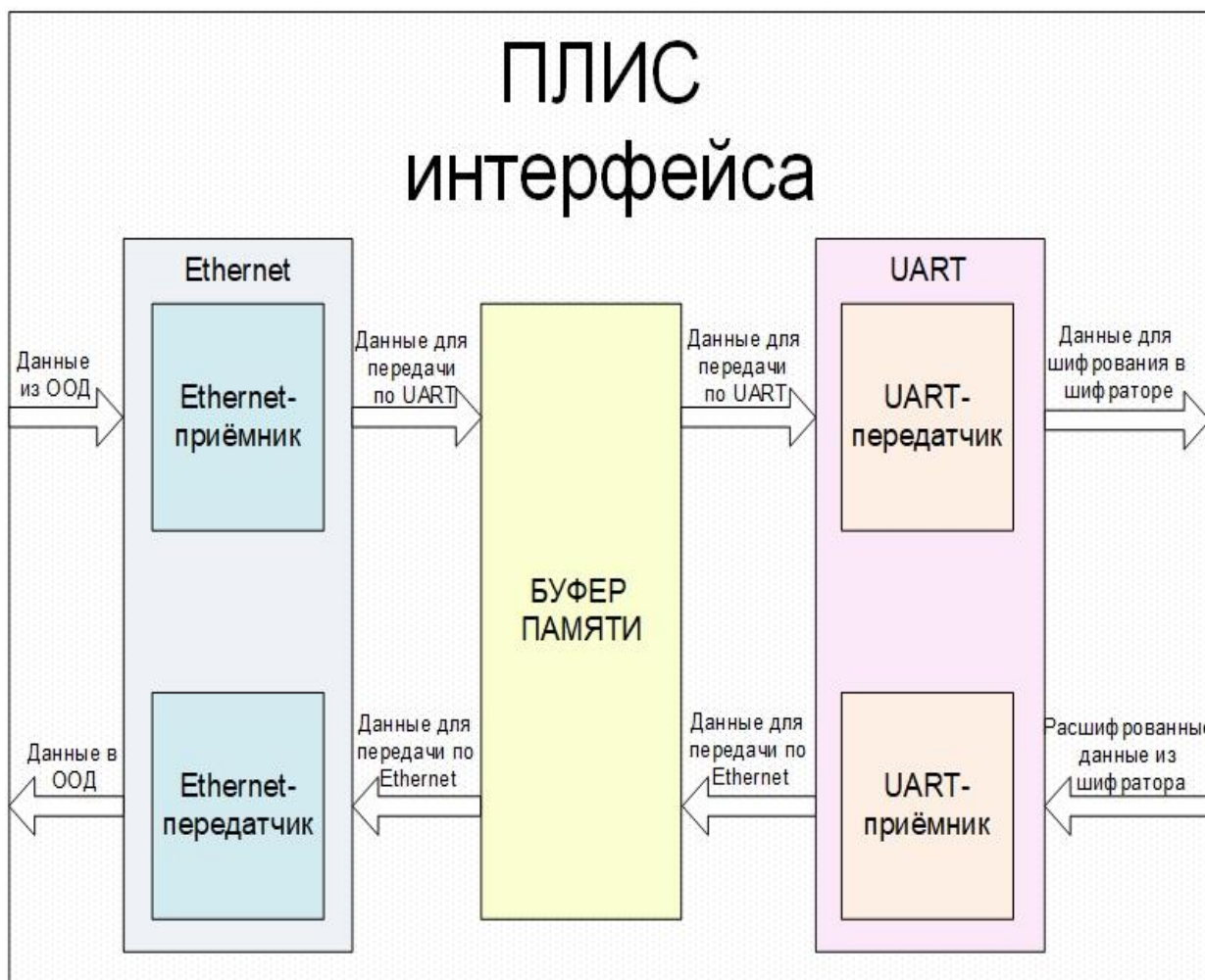


Рисунок 8 – Структура модуля обработки информации

ПЛИС интерфейса состоит из двух блоков – блока Ethernet и блока UART, которые взаимодействуют между собой через буфер памяти. Блоки Ethernet и UART включают приёмник и передатчик.

Данные из ООД в формате кадра Ethernet попадают в Ethernet-приёмник, обрабатываются и передаются в буфер памяти. Из буфера памяти данные попадают в UART-передатчик, в котором происходит формирование пакета в соответствии с внутренним протоколом и его передача в шифратор для шифрования.

Расшифрованные данные из шифратора попадают в UART-приёмник ПЛИС интерфейса. В приёмнике происходит проверка целостности пакета и формирование квитанции по результатам проверки. Если целостность пакета не была нарушена, данные записываются в буфер памяти, откуда они передаются в Ethernet-передатчик, в котором формируется кадр Ethernet и происходит его последующая передача в ООД.

Буфер памяти организован таким образом, что может содержать в себе до 16 пакетов формата Ethernet, каждый объемом 1,5 Кбайт. Память представляет собой таблицу, состоящую из зон, каждая из которых имеет свой адрес и предназначена для записи одного пакета. Одна зона памяти вмещает в себя 1,5 Кбайта данных, если пакет занимает меньший объём, то оставшаяся память не используется, а следующий пакет записывается в следующую зону с другим адресом. Каждая зона состоит из двухбайтовых строк, в которые записывается информация.

Для интерфейсных модулей используется ПЛИС XC6SLX9 семейства Spartan 6 в корпусе CPG196. Семейство Spartan является массовым FPGA, имеющими довольно низкую стоимость и предназначенные для решения задач, не требующих больших вычислительных мощностей. Кристаллы серии Spartan-6 производятся по КМОП-технологии 45-нм техпроцесса, который обеспечивая оптимальный баланс стоимости, мощности и производительности [6].

Некоторые особенности и свойства ПЛИС семейства Spartan-6 представлены ниже:

- 45-нм техпроцесс, позволяющий снизить затраты и энергопотребление устройства;
- оптимизированный выбор стандартов вводы-вывода информации;
- наличие функции горячего подключения;
- скорости передачи данных до 800 Мбит/с (пиковая пропускная способность 12,8 Гбит/с);
- дополнительная поддержка сдвигового регистра или распределенной оперативной памяти;
- блок управления и синтеза сигналов синхронизации СМТ;
- каждая ПЛИС Spartan-6 имеет от 12 до 268 двухпортовых блочных ОЗУ;

- быстрая блочная оперативная память с блоками по 18 Кб, которые могут быть дополнительно запрограммированы как два независимых блока оперативной памяти по 9 Кб;
- быстрая встроенная обработка благодаря усовершенствованному и недорогому процессору MicroBlaze;
- низкое статическое и динамическое потребление энергии.

Семейство Spartan-6 представляет собой шестое поколение ПЛИС от Xilinx, они были выпущены в 2009 году. С их помощью можно создавать высокопроизводительные и энергоэффективные системы, которые могут быть адаптированы к различным требованиям и задачам. Благодаря своей низкой стоимости и гибкости ПЛИС семейства Spartan-6 являются актуальным решением для изделий массового производства и приложений.

Существуют и более усовершенствованные ПЛИС следующего поколения, однако, такие ПЛИС имеют в своем составе процессорное ядро, что повышает возможность наличия программных «закладок», а также более высокую стоимость.

Основные параметры используемой ПЛИС XC6SLX9 семейства Spartan-6 приведены в таблице 2.

Таблица 2 - Основные параметры ПЛИС

Тип кристалла		XC6SLX9
Число логических ячеек		9152
Конфигурируемые логические блоки (CLBs)	Фрагменты	1430
	Триггеры	11440
	Максимальный объем распределенной ОЗУ (Кб)	90
Фрагменты DSP48A1		16
Блоки ОЗУ	Количество блоков по 18 Кб	32
	Максимальный объем ОЗУ (Кб)	576
Количество блоков управления синхронизацией (CMT)		2

Продолжение таблицы 2

Тип кристалла		XC6SLX9
Блоки контроллера памяти		2
Количество банков ввода/вывода		4
Максимальное число пользовательских выводов		200
Корпус		CPG196
Габариты корпуса	Габариты, мм	8 x 8
	Высота, мм	0,5
Число пользовательских выводов		106

3 Разработка структуры процессов приема и передачи информации

3.1 Разработка структуры процесса приема информации

Для упрощения процесса разработки модуля UART, входящего в состав ПЛИС интерфейса была разработана общая структура процесса приёма потока информации. Структура представлена на рисунке 9.

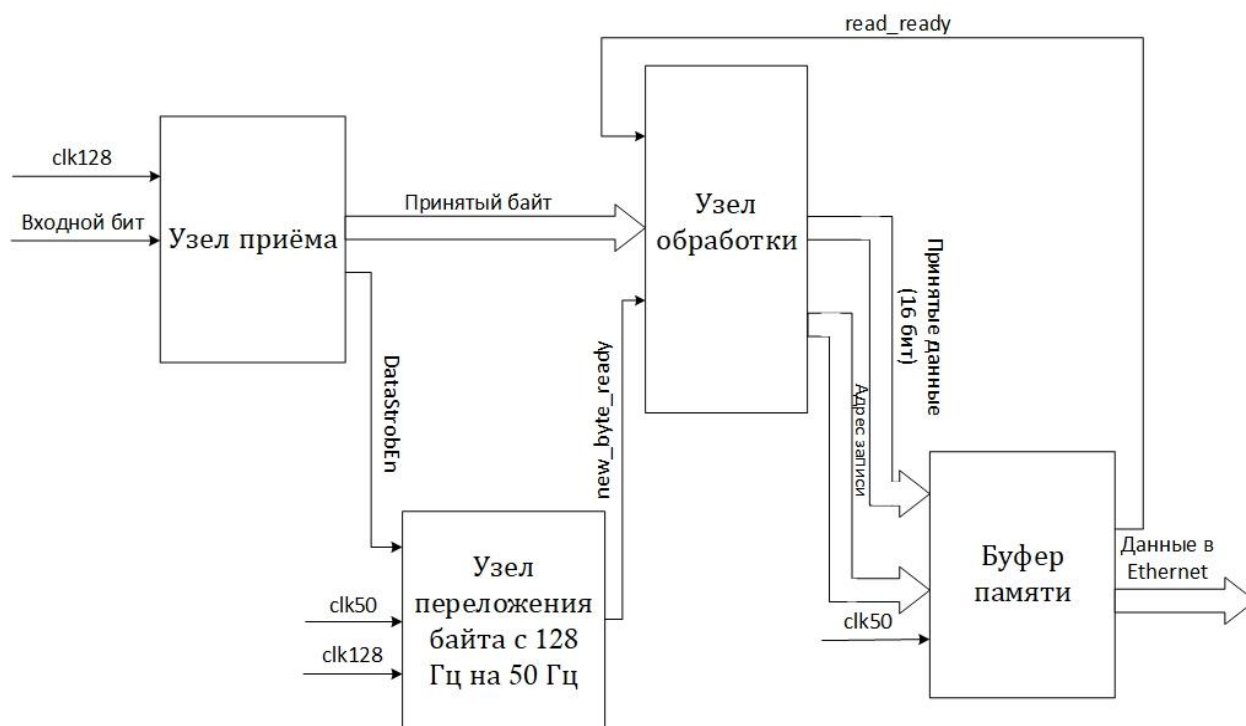


Рисунок 9 – Схема процесса приёма потока информации

Приёмник содержит 4 основных блока:

- буфер памяти;
- узел обработки;
- узел переложения байта с 128 МГц на 50 МГц;
- узел приёма.

Информация из канала по одному биту поступает в узел приёма, работающий на частоте 128 МГц. Когда в узел поступает 8 бит, сигнал DataStrobe передается в узел переложения байта с частоты 128 МГц на частоту 50 МГц, который передает в узел обработки сигнал new_byte_ready, сообщающий о готовности узла приёма к передаче принятого байта.

Принятый узлом приёма байт, переложенный на частоту 50 МГц, передаётся по шине в узел обработки.

В узел обработки информации происходит выделение полезной информации из принятых данных в соответствии с внутренним протоколом, в нём выполняется проверка контрольных сумм CRC принятых пакетов и формирование сигналов, указывающих передатчику на тип квитанции, которую необходимо сформировать по результатам проверки целостности принятого пакета.

Узел обработки передаёт в буфер памяти по шине адрес, на который будет выполняться запись. Буфер памяти передаёт в узел обработки сигнал read_ready, который сообщает о наличии в нём достаточного количества свободной памяти для записи. Узел обработки передаёт в буфер по шине шестнадцатиразрядное двоичное слово.

В буфере памяти принятые байты данных собираются в пакеты и передаются в Ethernet.

Необходимость узла переложения обусловлена тем, что вся схема ПЛИС интерфейса работает на частоте 50 МГц, которая является частотой работы Ethernet. Частота работы 128 МГц характерна только для узла приёмника и узла передатчика, так как скорость их работы должна быть выше скорости работы остальных элементов схемы.

Частота 128 МГц была выбрана для приема, так как именно на такой частоте в виду избыточности передаваемого кадра обеспечивается скорость передачи информации по UART в размере до 100 Мбит, соответствующей скорости работы FastEthernet. Такая скорость работы интерфейса необходима, чтобы пакет принимался и передавался без задержек.

Для узла приёма, который является одним из важнейших элементов схемы приёма, была разработана подробная схема, представленная на рисунке 10.

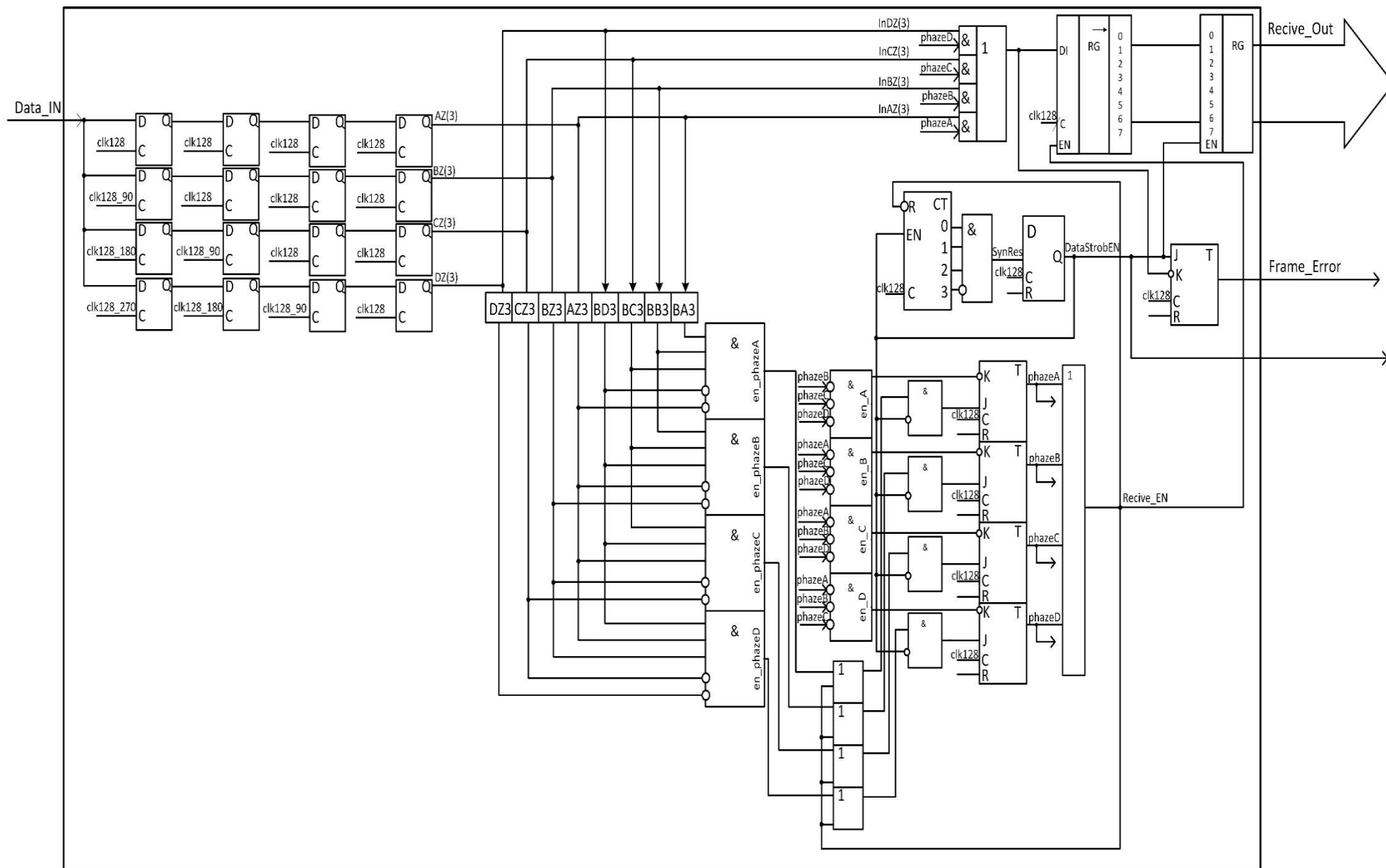


Рисунок 10 – Логическая схема узла приёма

Информация поступает на вход «Data_IN» схемы и передаётся в четыре цепи триггеров практически одновременно. Каждая цепь состоит из четырёх D-триггеров, работающих на тактовой частоте 128 МГц. Первая цепь триггеров осуществляет стробирование информационного сигнала частотой 128МГц, сдвинутой по фазе на 0 градусов. Вторая цепь стробирование информационного сигнала частотой 128 МГц, сдвинутой по фазе на 90 градусов. Третья и четвертая цепи стробируют сигнал частотой 128 МГц, сдвинутой на 180 и 270 градусов соответственно. Так как остальная часть схемы работает на частоте 128 МГц, сдвинутой на 0 градусов, необходимо выровнять периоды информационных сигналов, для чего и используются оставшиеся в цепи триггеры.

Сигналы AZ(3), BZ(3), CZ(3), DZ(3) полученные на выходах цепочки триггеров записываются в старшие четыре разряда восьмибитового буфера. На следующем такте работы схемы, они сдвигаются в младшие разряды буфера, а в старшие разряды записываются новые значения.

Для определения фазы, которая обнаружила стартовый бит, используется схема из конъюнкторов, дизъюнкторов и инверторов. Выбора фазы, по которой будет осуществляться приём информации, основан на поиске перепада сигнала, указывающего на стартовый бит. Если три снятых подряд информационных значения равны логической единице, а следующие два значения равны логическому нулю, то предполагается, что стартовый бит был обнаружен результате стробирования фазой частоты 128 МГц, на которой был обнаружен второй логический ноль. Считается, что информационные биты, принятые на данной частоте, являются правильно принятыми значениями. Значения, записанные в буфере, передаются в четыре конъюнктора. На первый конъюнктор en_phazeA подаётся значения из пяти младших разрядов буфера, при этом четвертое и пятое значения инвертируются. На второй конъюнктор en_phazeB подаются значения из второго, третьего, четвертого, пятого и шестого разрядов буфера, значения из пятого и шестого разрядов инвертированы. На третий конъюнктор en_phazeC

подаются значения из третьего, четвертого, пятого, шестого и седьмого разрядов буфера, значения из шестого и седьмого разрядов инвертированы. На четвёртый конъюнктор en_phazeD подаются значения из четвертого, пятого, шестого, седьмого и восьмого разрядов буфера, значения из седьмого и восьмого разрядов инвертированы.

В конъюнктор en_A с инвертированными входами передаются сигналы phazeB, phazeC, phazeD. В конъюнктор en_B с инвертированными входами передаются сигналы phazeA, phazeC, phazeD. В конъюнктор en_C с инвертированными входами передаются сигналы phazeA, phazeB, phazeD. В конъюнктор D с инвертированными входами передаются сигналы phazeA, phazeB, phazeC. Описание сигналов phazeA, phazeB, phazeC, phazeD представлено ниже.

Сигналы из en_A, en_B, en_C и en_D передаются на инвертированный вход К четырёх соответствующих JK-триггеров. На выходе данных триггеров вырабатываются сигналы phazeA, phazeB, phazeC, phazeD, которые служат для обозначения фазы частоты, по которой выполняется приём информации. Результатом дизъюнкции данных сигналов является сигнал Recive_EN. Сигнал Recive_EN служит для обозначения того, что приём информации осуществляется хотя бы на одной из фаз частоты 128 МГц. На входы J четырёх JK-триггеров подаются значения, которые являются результатами конъюнкций результатов парных дизъюнкций сигнала Recive_EN с сигналами из en_phazeA, en_phazeB, en_phazeC, en_phazeD с сигналом DataStrobEn, который будет описан ниже.

Сигнал DataStrobEn является задержанным на один такт сигналом счетчика, который считает количество принятых бит информации. Сигнал DataStrobEn возникает на восьмом такте счётчика и сигнализирует о том, что байт принят полностью. Сигнал DataStrobEn также подаётся на вход JK-триггера, отвечающего за выработку сигнала Frame_Error, указывающего на то, что при передаче кадра произошла ошибка или была выбрана неправильная фаза при приёме. Сигнал Frame_Error является одним из

выходных сигналов модуля, и принимает значение логической единицы в том случае, если следующий за принятым байтом бит (стоп-бит) не имеет значение логической единицы. Данный сигнал блокирует обработку пакета в дальнейших модулях, так как пакет считается повреждённым.

Сигналы AZ(3), BZ(3), CZ(3), DZ(3) полученные на выходах цепочки триггеров подаются на четыре конъюнктора, на которые также подаются сигналы phazeA, phazeB, phazeC, phazeD соответственно. Результаты парных конъюнкций направляются в дизъюнктор, результат данной дизъюнкции записывается в восьми разрядный регистр, начинающий запись по сигналу Recive_EN. Таким образом, в регистр записываются биты, стробированные фазой частоты 128 МГц, которая была выбрана схемой принятия решений, описанной выше.

Данные в регистр записываются в ходе приёма байта, после того как байт полностью принят, сигнал DataStrobEn разрешает работу второго восьмиразрядного регистра, в который записывается байт из первого регистра. Значения во втором регистре изменяются только после того, как будет принят следующий байт. Данные из второго регистра передаются во внешний модуль для дальнейшей обработки. Сигналы DataStrobEn и Frame_Error также являются выходными сигналами модуля.

3.2 Разработка структуры процесса передачи информации

Для упрощения процесса разработки модуля UART, входящего в состав ПЛИС интерфейса была разработана общая структура процесса передачи потока информации. Структура представлена на рисунке 11.

Передачика содержит 4 основных блока:

- буфер памяти;
- узел обработки;
- узел переложения байта с 50 МГц на 128 МГц;
- узел передачи.

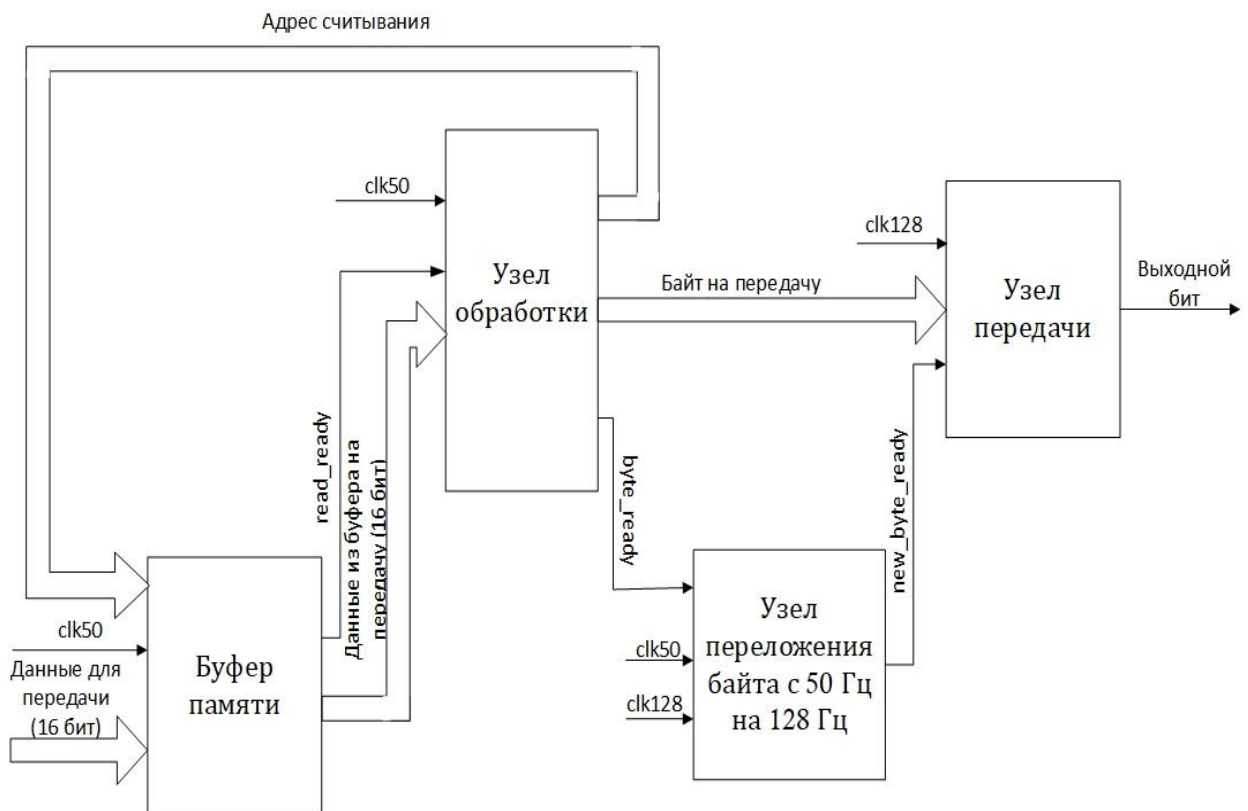


Рисунок 11 – Схема процесса передачи потока информации

Буфер памяти предназначен для хранения пакетов. На вход буфера подаются слова размером два байта с частотой 50 МГц. Буфер, передаёт в узел обработки сигнал `read_ready`, сообщающий о наличии информации для считывания. Узел обработки передаёт в буфер адрес, с которого собирается считать информацию, после чего информация по указанному адресу из буфера передаётся в узел обработки.

В узле обработки информации производится подготовка пакета к передаче в соответствии с внутренним протоколом обмена. Если полученные из буфера байты являются первыми в пакете, то перед их отправкой передаётся байт-префикс «1В» и байт «23», указывающий на начало пакета. В узле обработки формируется контрольная сумма пакета. После передачи сформированной контрольной суммы отправляются байт-префикс «1В» и байт «24», указывающий на конец пакета. Также в узле обработки происходит формирование квитанций.

Из узла обработки информации байты передаются в узел передачи, который является сдвиговым регистром, работающим на частоте 128 МГц.

На каждом такте в канал передаётся один бит информации. Из узла передачи байты передаются в канал в соответствии с протоколом UART.

Узел переложения байта необходим для переложения информационного сигнала с частоты 50 МГц на частоту 128 МГц.

4 Внутренний протокол обмена данными

Обмен данными интерфейса с шифратором происходит по внутреннему протоколу, который предназначен для обнаружения ошибок и блокирования попадания искаженных пакетов в шифратор. Использование внутреннего протокола позволяет исключить шифрование/расшифрование и передачу пакетов с ошибками, возникшими при передаче информационных пакетов между ПЛИС, в оконечное оборудование данных или в канал связи по интерфейсу Ethernet. Некорректные пакеты отбрасываются и не подлежат какой-либо дальнейшей обработке.

Передаваемые данные между ПЛИС шифратора и ПЛИС интерфейсов должны быть представлены в виде последовательности байтов, каждый байт передаётся в следующем формате:

- 1 бит стартовый;
- 8 бит передаваемых данных;
- 2 бита стоповых.

Стартовый бит является логическим нулем, а стоповый бит – логической единицей. Порядок следования битов при передаче/приеме каждого байта данных – от младшего к старшему.

Передаваемые/принимаемые по интерфейсу данные должны быть представлены в виде пакетов формата, представленного в таблице 3.

Таблица 3 – Формат передаваемого пакета

Содержание поля	Код 1В	УК	Данные	КС	Код 1В	УК
Число байт	1	1	N	2	1	1

В таблице используются следующие обозначения:

- УК – управляющий код;
- КС – контрольная сумма.

Перед управляющим кодом всегда используется байт-префикс «1В». Префикс и управляющий код являются служебными данными.

Управляющий код может принимать следующие значения:

- 23 – начало пакета;
- 24 – конец пакета;
- 53 – пакет успешно принят;
- 4Е – несовпадение контрольной суммы.

Обмен информационными пакетами между ПЛИС шифратора и ПЛИС интерфейсов ведется в дуплексном режиме. ПЛИС шифратора при приеме входящего пакета должно квитировать результат приема. ПЛИС интерфейсов не должна начинать передачу следующего пакета до получения квитанции на предыдущий, переданный в ПЛИС шифратора, или по истечении установленного времени ожидания квитанции.

Квитанция состоит из двух байт:

- 1В – префикс пакета;
- 53 или 4Е – управляющий код.

При приеме пакета данных без ошибок на физическом уровне устройство продолжает обработку пакета на информационно-логическом уровне. При обнаружении ошибки кадрирования или контрольной суммы, обработка принятых данных прекращается, а пакет отбрасывается.

Для избегания ошибочного принятия информационных данных за служебные используется механизм вставки дополнительного байта 1В при передаче пакета. Передающее устройство при обнаружении в поле информационных данных кода 1В добавляет еще один байт 1В. Принимающее устройство при обнаружении в принимаемой информации последовательности байт 1В1В, первый байт 1В из текста исключает, а второй байт 1В считает информационным байтом. При подсчете контрольной суммы дополнительные вставки байт не учитываются.

Контрольная сумма пакета состоит из двух байт, формируется передающим устройством и добавляется в конец поля «данные».

Контрольная сумма является результатом побайтового сложения с циклическим переносом единицы и байтов поля «Данные» по модулю 256.

Принимающее устройство рассчитывает контрольную сумму принимаемого пакета по тому же алгоритму, что и передающее. При совпадении принятой и вновь сформированной контрольных сумм пакет считается принятым правильно и подлежит дальнейшей обработке. При несовпадении контрольных сумм пакет отбрасывается. Далее приведен алгоритм расчета контрольной суммы по алгоритму CRC-16.

Алгоритм CRC решает задачу обнаружения ошибок, возникающих при передаче информации. Процесс использования CRC для обнаружения ошибок может быть описан следующим образом: для k -битной последовательности двоичного кода, которая должна быть передана на передающей стороне, по определенным правилам формируется r -битный контрольный код (код CRC) для проверки целостности передаваемой последовательности. Контрольный код прикрепляется к исходной информации. Размер передаваемого сообщения $k + r$ бит. На принимающей стороне выполняется проверка контрольного кода CRC, чтобы определить, произошла ли ошибка во время передачи [7].

Контрольная сумма типа CRC хорошо подходит для проверки целостности данных при случайных искажениях информации в канале передачи данных, но совершенно не подходит для защиты от намеренного взлома [8].

В качестве полинома-генератора $G(x)$ в алгоритме CRC-16 используется полином $x^{16} + x^{15} + x^2 + 1$. Разрядность контрольной суммы $r=16$.

Существует несколько методов расчета контрольной суммы CRC, в реализуемом программном модуле используется метод с использованием таблиц поиска (lookup table), преимуществом которого является более высокая скорость работы [9]. Для хранения таблицы необходимо выделять память, однако ресурсы ПЛИС, на которой реализуется программный

модуль, позволяют это сделать ради увеличения скорости вычисления контрольной суммы. Суть данного метода заключается в построении таблицы, значения в которой представляют собой заранее рассчитанные контрольные суммы (столбец «Значение по адресу») для каждого байта от 0 до 255 (столбец «Адрес» в таблице). Контрольные суммы рассчитываются с использованием полинома-генератора $G(x)$ и начального значения CRC-16 равного «FFFF» в шестнадцатеричной системе счисления. Расчет значения контрольной суммы для каждого байта происходит по алгоритму, описанному ниже.

Исходный информационный полином (байт) $P(x)$, наибольшая степень которого увеличенная на единицу равна k , необходимо умножить x^r (последовательность двоичного кода сдвинуть влево на r битов), где r - наибольшая степень полинома-генератора $G(x)$. Получившийся результат необходимо разделить полином-генератор $G(x)$, полученный остаток от деления $R(x)$ - полином CRC, значение, записываемое в таблицу. Все операции выполняются в поле $GF(2)$.

Алгоритм вычисления контрольной суммы при помощи таблицы поиска следующий:

- выбирается первый байт массива данных, который рассматривается как адрес в таблице;
- второй байт из массива данных суммируется по модулю 2 со значением, находящемся в таблице по адресу, на который указал предыдущий байт;
- если массив данных не закончился, то полученное значение, рассматривается как адрес в таблице, цикл повторяется.

Алгоритм повторяется до тех пор, пока массив данных не закончится, последнее полученное значение и будет являться контрольной суммой.

Таблица поиска для алгоритма CRC-16 представлена в таблице 4.

Таблица 4 – Справочная таблица CRC

Адрес (0-3F)	Значение	Адрес (40-7F)	Значение	Адрес (80-BF)	Значение	Адрес (C0-FF)	Значение
0x00	0x0000	0x40	0xF001	0x80	0xA001	0xC0	0x5000
0x01	0xC0C1	0x41	0x30C0	0x81	0x60C0	0xC1	0x90C1
0x02	0xC181	0x42	0x3180	0x82	0x6180	0xC2	0x9181
0x03	0x0140	0x43	0xF141	0x83	0xA141	0xC3	0x5140
0x04	0xC301	0x44	0x3300	0x84	0x6300	0xC4	0x9301
0x05	0x03C0	0x45	0xF3C1	0x85	0xA3C1	0xC5	0x53C0
0x06	0x0280	0x46	0xF281	0x86	0xA281	0xC6	0x5280
0x07	0xC241	0x47	0x3240	0x87	0x6240	0xC7	0x9241
0x08	0xC601	0x48	0x3600	0x88	0x6600	0xC8	0x9601
0x09	0x06C0	0x49	0xF6C1	0x89	0xA6C1	0xC9	0x56C0
0x0A	0x0780	0x4A	0xF781	0x8A	0xA781	0xCA	0x5780
0x0B	0xC741	0x4B	0x3740	0x8B	0x6740	0xCB	0x9741
0x0C	0x0500	0x4C	0xF501	0x8C	0xA501	0xCC	0x5500
0x0D	0xC5C1	0x4D	0x35C0	0x8D	0x65C0	0xCD	0x95C1
0x0E	0xC481	0x4E	0x3480	0x8E	0x6480	0xCE	0x9481
0x0F	0x0440	0x4F	0xF441	0x8F	0xA441	0xCF	0x5440
0x10	0xCC01	0x50	0x3C00	0x90	0x6C00	0xD0	0x9C01
0x11	0x0CC0	0x51	0xFCC1	0x91	0xACC1	0xD1	0x5CC0
0x12	0x0D80	0x52	0xFD81	0x92	0xAD81	0xD2	0x5D80
0x13	0xCD41	0x53	0x3D40	0x93	0x6D40	0xD3	0x9D41
0x14	0x0F00	0x54	0xFF01	0x94	0xAF01	0xD4	0x5F00
0x15	0xCFC1	0x55	0x3FC0	0x95	0x6FC0	0xD5	0x9FC1
0x16	0xCE81	0x56	0x3E80	0x96	0x6E80	0xD6	0x9E81
0x17	0x0E40	0x57	0xFE41	0x97	0xAE41	0xD7	0x5E40
0x18	0x0A00	0x58	0xFA01	0x98	0xAA01	0xD8	0x5A00
0x19	0xCAC1	0x59	0x3AC0	0x99	0x6AC0	0xD9	0x9AC1

Продолжение таблицы 4

Адрес (0-3F)	Значение	Адрес (40-7F)	Значение	Адрес (80-BF)	Значение	Адрес (C0-FF)	Значение
0x1A	0xCB81	0x5A	0x3B80	0x9A	0x6B80	0xDA	0x9B81
0x1B	0x0B40	0x5B	0xFB41	0x9B	0xAB41	0xDB	0x5B40
0x1C	0xC901	0x5C	0x3900	0x9C	0x6900	0xDC	0x9901
0x1D	0x09C0	0x5D	0xF9C1	0x9D	0xA9C1	0xDD	0x59C0
0x1E	0x0880	0x5E	0xF881	0x9E	0xA881	0xDE	0x5880
0x1F	0xC841	0x5F	0x3840	0x9F	0x6840	0xDF	0x9841
0x20	0xD801	0x60	0x2800	0xA0	0x7800	0xE0	0x8801
0x21	0x18C0	0x61	0xE8C1	0xA1	0xB8C1	0xE1	0x48C0
0x22	0x1980	0x62	0xE981	0xA2	0xB981	0xE2	0x4980
0x23	0xD941	0x63	0x2940	0xA3	0x7940	0xE3	0x8941
0x24	0x1B00	0x64	0xEB01	0xA4	0xBB01	0xE4	0x4B00
0x25	0xDBC1	0x65	0x2BC0	0xA5	0x7BC0	0xE5	0x8BC1
0x26	0xDA81	0x66	0x2A80	0xA6	0x7A80	0xE6	0x8A81
0x27	0x1A40	0x67	0xEA41	0xA7	0xBA41	0xE7	0x4A40
0x28	0x1E00	0x68	0xEE01	0xA8	0xBE01	0xE8	0x4E00
0x29	0xDEC1	0x69	0x2EC0	0xA9	0x7EC0	0xE9	0x8EC1
0x2A	0xDF81	0x6A	0x2F80	0xAA	0x7F80	0xEA	0x8F81
0x2B	0x1F40	0x6B	0xEF41	0xAB	0xBF41	0xEB	0x4F40
0x2C	0xDD01	0x6C	0x2D00	0xAC	0x7D00	0xEC	0x8D01
0x2D	0x1DC0	0x6D	0xEDC1	0xAD	0xBDC1	0xED	0x4DC0
0x2E	0x1C80	0x6E	0xEC81	0xAE	0xBC81	0xEE	0x4C80
0x2F	0xDC41	0x6F	0x2C40	0xAF	0x7C40	0xEF	0x8C41
0x30	0x1400	0x70	0xE401	0xB0	0xB401	0xF0	0x4400
0x31	0xD4C1	0x71	0x24C0	0xB1	0x74C0	0xF1	0x84C1
0x32	0xD581	0x72	0x2580	0xB2	0x7580	0xF2	0x8581
0x33	0x1540	0x73	0xE541	0xB3	0xB541	0xF3	0x4540
0x34	0xD701	0x74	0x2700	0xB4	0x7700	0xF4	0x8701

Продолжение таблицы 4

Адрес (0-3F)	Значение	Адрес (40-7F)	Значение	Адрес (80-BF)	Значение	Адрес (C0-FF)	Значение
0x35	0x17C0	0x75	0xE7C1	0xB5	0xB7C1	0xF5	0x47C0
0x36	0x1680	0x76	0xE681	0xB6	0xB681	0xF6	0x4680
0x37	0xD641	0x77	0x2640	0xB7	0x7640	0xF7	0x8641
0x38	0xD201	0x78	0x2200	0xB8	0x7200	0xF8	0x8201
0x39	0x12C0	0x79	0xE2C1	0xB9	0xB2C1	0xF9	0x42C0
0x3A	0x1380	0x7A	0xE381	0xBA	0xB381	0xFA	0x4380
0x3B	0xD341	0x7B	0x2340	0xBB	0x7340	0xFB	0x8341
0x3C	0x1100	0x7C	0xE101	0xBC	0xB101	0xFC	0x4100
0x3D	0xD1C1	0x7D	0x21C0	0xBD	0x71C0	0xFD	0x81C1
0x3E	0xD081	0x7E	0x2080	0xBE	0x7080	0xFE	0x8081
0x3F	0x1040	0x7F	0xE041	0xBF	0xB041	0xFF	0x4040

Для проверки целостности информации, последовательность информационных байт вместе с рассчитанными байтами контрольной суммы необходимо запустить в алгоритм ещё раз. Если итоговое значение будет равно 0, значит целостность передаваемой информации не была нарушена, иначе это означает, что во время передачи возникли ошибки.

5 Разработка интерфейса на языке VHDL

На языке VHDL в среде разработки Xilinx ISE DesignSuite 14.7 был разработан программный модуль высокоскоростного интерфейса передачи данных.

Файл `Rec.vhd` содержит в себе код узла приёма информации, отвечающего за стробирование входных информационных бит из UART-приёмника шифратора, а также их передачу в последующие узлы схемы приёмника. Выходами приёмника являются сигналы:

- сигнал `recive_out` является логическим вектором размером один байт, содержит в себе принятый байт данных;
- сигнал `DataStrobEn` является логическим сигналом, сообщает внешним модулям о том, что байт данных принят и готов к последующей обработке;
- сигнал `Frame_Error` является логическим сигналом, который служит для информирования внешних модулей о том, что произошла ошибка при приёме байта, что приводит к блокированию обработки всего пакета.

Алгоритм работы данного узла был описан в разделе 3 данной работы. Код из файла `Rec.vhd` представлен в приложении А.

В файлах `Tabl_CRC_16_mux.vhd` и `blok_CRC_16_mux.vhd` был реализован узел расчёта контрольной суммы CRC-16 методом справочных таблиц. Метод расчёта контрольной суммы был описан в предыдущем разделе данной работы. В файле `Tabl_CRC_16_mux.vhd` содержит справочную таблицу, представленную в предыдущем разделе работы, а файл `blok_CRC_16_mux.vhd` содержит в себе код, отвечающий за расчёт контрольной суммы по таблице. Код узла расчёта контрольной суммы представлен в приложении Б.

С помощью встроенного в среду разработки Xilinx ISE Design Suite 14.7 инструмента Xilinx Core Generator Tool был сформирован код узла генерации частот. Реализовывать данный узел вручную при наличии такого

инструмента было нецелесообразно. В проект был добавлен модуль ClkGen, и с помощью функции Clocking Wizard инструмента Core Generator Tool была задана входная частота узла. Окно ввода представлено на рисунке 12.

Input Clock Information				
Input Clock	Input Freq (MHz)		Input Jitter	Source
	Value	Valid Range		
primary	128.000	5.000 - 500.000	0.010	Single ended clock capable pin

Рисунок 12 – Окно ввода входной частоты генератора.

Далее были заданы необходимые выходные частоты узла генерации, а также их фазовые сдвиги. Поля ввода генерируемых частот представлены на рисунке 13.

Output Clock	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
	Requested	Actual	Requested	Actual	Requested	Actual
CLK_OUT1	128.000	128.000	0.000	0.000	50.000	50.0
☒ CLK_OUT2	128.000	128.000	90.000	90.000	50.000	50.000
☒ CLK_OUT3	128.000	128.000	180.000	180.000	50.000	50.000
☒ CLK_OUT4	128.000	128.000	270.000	270.000	50.000	50.000
☐ CLK_OUT5	100.000	N/A	0.000	N/A	50.000	N/A
☐ CLK_OUT6	100.000	N/A	0.000	N/A	50.000	N/A

Рисунок 13 – Окно ввода генерируемых частот

Окно, демонстрирующее итоговые входные и выходные сигналы узла генерации частот представлено на рисунке 14. На данном рисунке показана входящая в узел частота CLK_IN1 равная 128 МГц, а также частоты CLK_OUT1, CLK_OUT2, CLK_OUT3 и CLK_OUT4, которые являются сдвинутыми по фазе относительно частоты CLK_IN1 на 0, 90, 180 и 270 градусов соответственно.

Код узла генерации частот представлен в приложении В.

Input Clock Summary			
Input Clock	Port Name	Input Freq (MHz)	Input Jitter (UI)
primary	CLK_IN1	128.008	0.010

Output Clock Summary						
Input jitter is not used for calculating Pk-to-Pk Jitter for the DCM. DCM Input jitter is reported in Static Timing within the Clock Uncertainty calculation						
Output Clock	Port Name	Output Freq (MHz)	Phase (degrees)	Duty Cycle (%)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
CLK_OUT1	CLK_OUT1	128.008	0.000	50.0	200.000	150.000
CLK_OUT2	CLK_OUT2	128.008	90.000	50.0	300.000	150.000
CLK_OUT3	CLK_OUT3	128.008	180.000	50.0	300.000	150.000
CLK_OUT4	CLK_OUT4	128.008	270.000	50.0	300.000	150.000

Other Pins	Port Name
RESET	RESET

Рисунок 14 – Входные и выходные сигналы узла

Файл Transiv.vhd содержит в себе код, реализующий узел передачи информации интерфейса UART. Входные сигналы данного узла:

- сигнал `byte_data` является логическим вектором размером один байт, содержит в себе байт данных, который необходимо передать;
- сигнал `byte_ready` является логическим сигналом, сообщает приёмнику о том, что байт данных готов к передаче.

Выходным сигналом узла передачи является сигнал `transiver_out`, который является передаваемым в канал битом данных.

Передатчик, получив сигнал `byte_ready`, формирует вектор размером одиннадцать бит, состоящий из двух стоповых бит в старших разрядах, байта

byte_data и стартового бита в младшем разряде. Вектор по одному биту сдвигается в канал, начиная с младшего разряда. При отсутствии сигнала byte_ready в канал передаётся логическая единица. Код передатчика представлен в приложении Г.

С помощью инструмента Xilinx Core Generator Tool был сформирован код, который генерирует частоту 128 МГц из входящей частоты 50 МГц. Код содержится в файле clk50128. В качестве входной частоты была задана частота 50 МГц, выходными частотами узла были установлены частоты 50 МГц и 128 МГц.

Был написан код на языке vhdl узла переложения бита с частоты 50 МГц на 128 МГц (со 128 МГц на 50 МГц). Входные сигналы данного модуля:

- частота Clk1, с которой осуществляется переложение бита;
- частота Clk2, на которую осуществляется переложение бита;
- сигнал bit_in (сигнал byte_ready от приёмника или DataStrobEn от передатчика).

Появление сигнала bit_in в состоянии логической единицы на входе узла переложения переводит в состояние логической единицы сигнал tmp. При этом сигнал bit_in принимается в узле на той частоте, с которой осуществляется переложение. По такту частоты, на которую осуществляется переложение (Clk2), сигнал tmp1 начинает дублировать сигнал tmp, а сигнал tmp2 начинает дублировать сигнал tmp1. Таким образом, сигнал tmp1 является сигналом tmp, задержанным на время, которое проходит между тактами частот, а сигнал tmp2 является задержанным на один такт частоты Clk2, сигналом tmp1. Сигналы tmp, tmp1 и tmp2 являются внутренними сигналами модуля.

Выходным сигналом узла является сигнал bit_out, являющийся переложённым на новую частоту сигналом bit_in. Длительность сигнала bit_out равна длительности сигнала tmp1 до появления сигнала tmp2, что соответствует периоду частоты Clk2. Код данного узла содержится в файле Perelobj.vhd.

Код из файлов clk50128.vhd и Perelobj.vhd представлен в приложении Д.

Файл Obrabotka.vhd содержит в себе код узла обработки информации, а также является файлом верхнего уровня для модулей:

- Perelobj.vhd;
- Rec.vhd;
- Tabl_CRC_16_mux.vhd;
- blok_CRC_16_mux.vhd.

В данном модуле реализуется фильтрация служебных полей внутреннего протокола обмена информацией, описанного в предыдущем разделе. Модуль способен определить начало и конец пакета, а также сравнить рассчитанную от принятого пакета контрольную сумму с принятой из предыдущего модуля. Также в модуле формируются квитанции, указывающие на успешный приём пакета или ошибку в принятых данных. Код данного модуля представлен в приложении Е.

6 Проверка работы интерфейса в среде моделирования ISim

Для проверки работы разработанного программного модуля были созданы файлы testbench для каждого разработанного узла. Файл testbench позволяет имитировать внешние воздействия, то есть входные сигналы в узлы. Однако, testbench и среда моделирования ISim не учитывают временные задержки, которые могут возникать при реальной работе устройства.

На рисунке 15 показана работоспособность узла выработки сдвинутых частот относительно исходной `clk_in1`. По рисунку видно, что промежуток времени между передними фронтами частот равен 1,953 нс, то есть четверть от периода исходной частоты, что соответствует сдвигу фазы на 90 градусов.

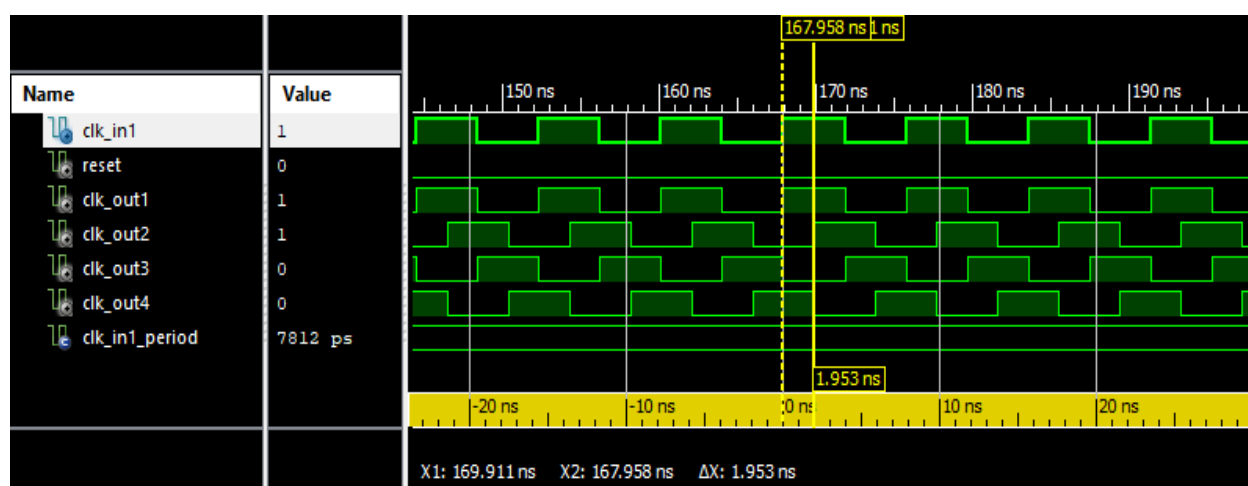


Рисунок 15 – Выработка частот

На рисунке 16 показана работоспособность узла переложения бита с частоты 128 МГц на частоту 50 МГц.

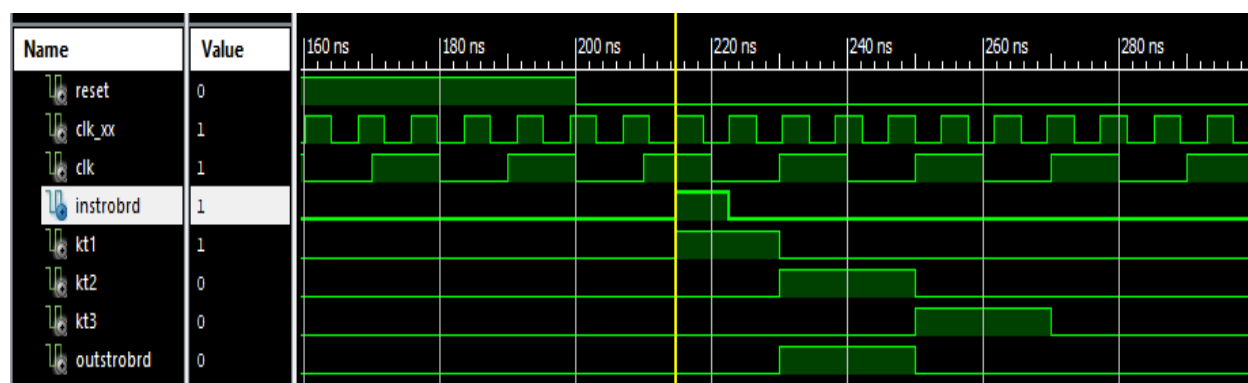


Рисунок 16 – Переложение бита с частоты 128 МГц на частоту 50 МГц

На рисунке 17 показана работоспособность узла переложения бита с частоты 50 МГц на частоту 128 МГц.

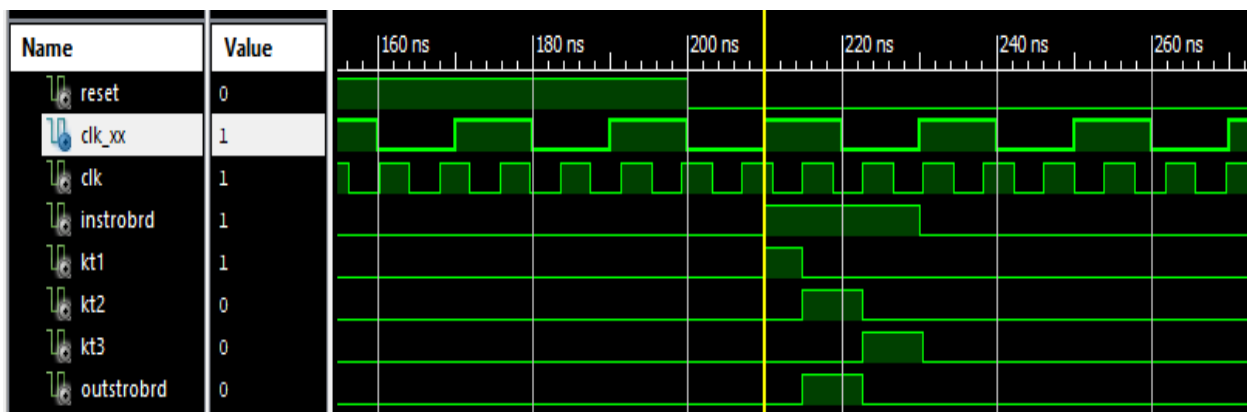


Рисунок 17 – Переложение бита с частоты 50 МГц на частоту 128 МГц

На рисунках 18 и 19 показана проверка работоспособности узла приёма, код которого содержится в файле Rec.vhd. Сигнал data имитирует входную информационную последовательность бит «0001101100100011000000000001000100100010001100110100010001010101011001100111011110001000100110010001101100100100», данная битовая последовательность в шестнадцатеричной системе счисления имеет вид «1b23001122334455667788991b24».

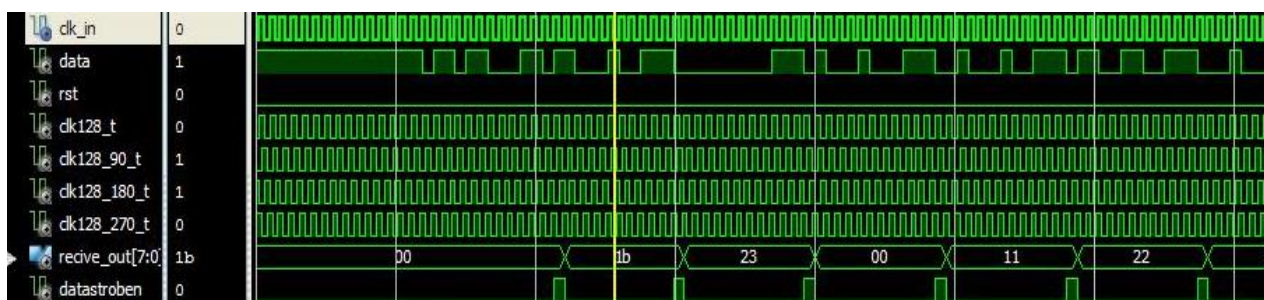


Рисунок 18 – Проверка работоспособности узла приёма в начале пакета

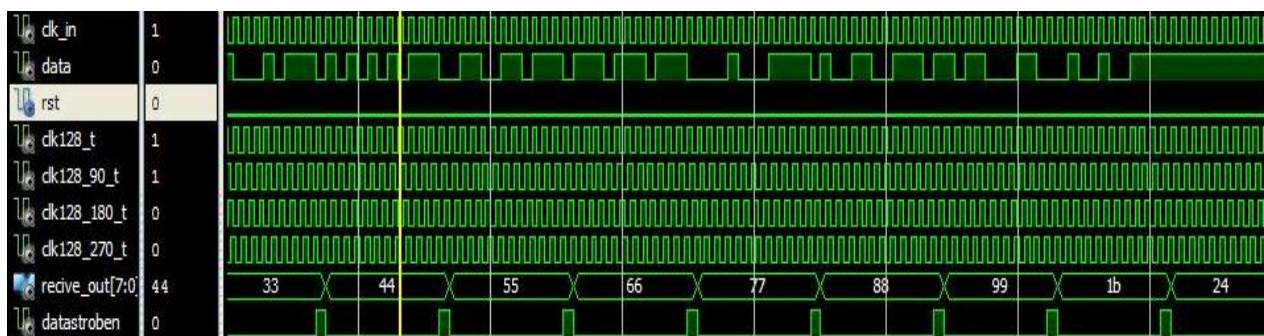


Рисунок 19 – Проверка работоспособности узла приёма в конце пакета

Как видно на рисунках, выходная последовательность байт (сигнал `recv_out`) соответствует входящей последовательности, что говорит о корректности работы модуля.

Для тестирования передатчика к нему на вход был подключен выход `recv_out` узла приёмника. На рисунках 20 и 21 представлены результаты тестов узла передачи.

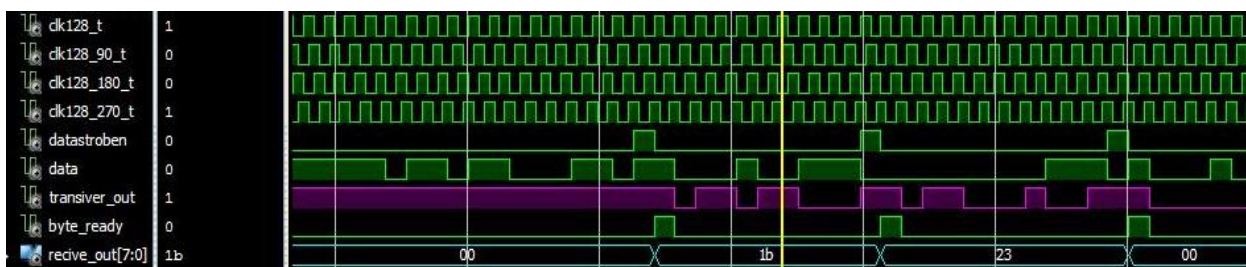


Рисунок 20 – Проверка работоспособности узла передачи крупным планом



Рисунок 21 – Проверка работоспособности узла передачи

Как видно по рисункам входящая последовательность `data` и выходная последовательность приёмника `transiver_out` совпадают. На рисунке показано, что передача начинается после появления сигнала `byte_ready`, который появляется в момент, когда на вход поступает байт для передачи.

Была проверена работоспособность узла обработки информации, тестирование модуля выполнялось совместно с модулями приёма, передачи, расчета контрольной суммы и переложения бита результат представлен на рисунках 22 и 23. На данных рисунках представлена обработка пакета данных, переданного в модуль обработки из модуля приёма, определение начала и конца пакета, а также расчет и сравнение контрольной суммы.



Рисунок 22 – Проверка работоспособности узла обработки в начале пакета

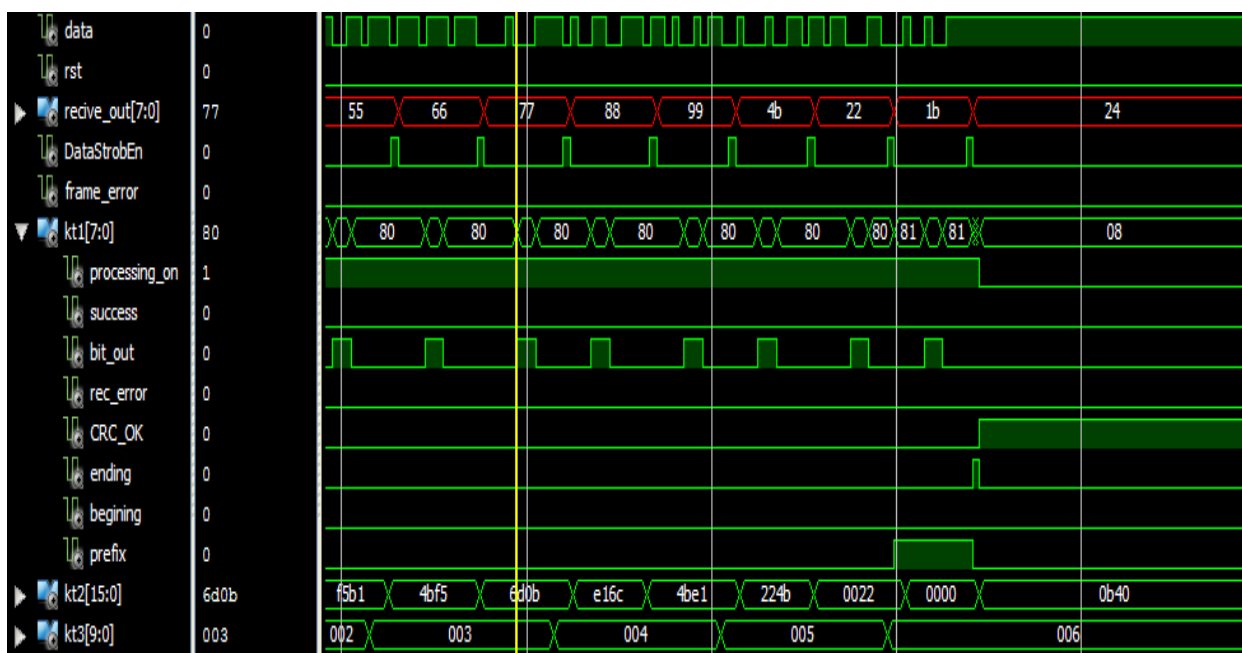


Рисунок 23 – Проверка работоспособности узла обработки в конце пакета

На вход приёмника data была передана последовательность бит «00011011001000110000000000010001001000100011001101000100010101010110011001110111100010001001100100011011010010110010001000100100». На выходе приёмника recive_out была получена последовательность байт, которая имеет вид «1b23001122334455667788994b221b24». Байт 4b22 соответствует коду контрольной суммы CRC-16, рассчитанному для последовательности байт «00112233445566778899». Сигнал recive_out был направлен на вход узла обработки.

Как видно по рисунку 22, сигнал `prefix`, отвечающий за обнаружение байта-префикса «1b» срабатывает в момент поступления этого байта в модуль. Импульсный сигнал `beginning` появляется после сигнала префикса, в момент приёма байта «23», обозначающего начало пакета. Как можно видеть на рисунке 22, на следующий такт после появления этого сигнала появляется сигнал `processing_on`, обозначающий начало обработки принятого сообщения.

На рисунке 22 также можно увидеть сигнал `bit_out`, возникающий только после приёма начала пакета, отвечающий за стробирование информационных байт. На рисунке 22 видно, что сигнал `bit_out` является переложенным на частоту 50 МГц сигналом `DataStrobEn`, поступающим из приёмника.

Сигнал `kt3` представляет собой счётчик, который считает размер информационного пакета в формате двухбайтовых слов. Как можно видеть по рисункам 22 и 23, значение сигнала меняется каждые два принятых информационных байта. На рисунке 23 можно увидеть, что итоговый размер принятого пакета равен шести.

На рисунке 23 можно увидеть сигнал `prefix`, отвечающий за обнаружение байта-префикса, а также импульсный сигнал `ending`, реагирующего на байт «24», обозначающий конец пакета.

Сигнал `kt2` представляет собой вектор контрольных сумм, рассчитанных от значений обрабатываемых информационных байт. Как можно видеть на рисунке 23, значение контрольной суммы после обработки всех информационных байт и принятых байт контрольной суммы равно 0, что свидетельствует о том, что целостность не была нарушена.

7 Проверка быстродействия интерфейса

Быстродействие изделия было проверено с помощью программы для измерения скорости `iperf`. Кроссплатформенная программа со свободно распространяемым исходным кодом `iperf` это генератор сетевого трафика, предназначенный для проверки скорости и пропускной способности сети, и состоящая из клиентской и серверной части.

На компьютере 1 запущена программа-клиент, генерирующая поток информации со скоростью 80 Мбит/с, на компьютере 2 запущена программа-сервер, рассчитывающая фактическую скорость прохождения пакетом пути, показанного на рисунке 24. Размер пакета составляет 1420 байт, то есть максимальный размер IP-пакета.

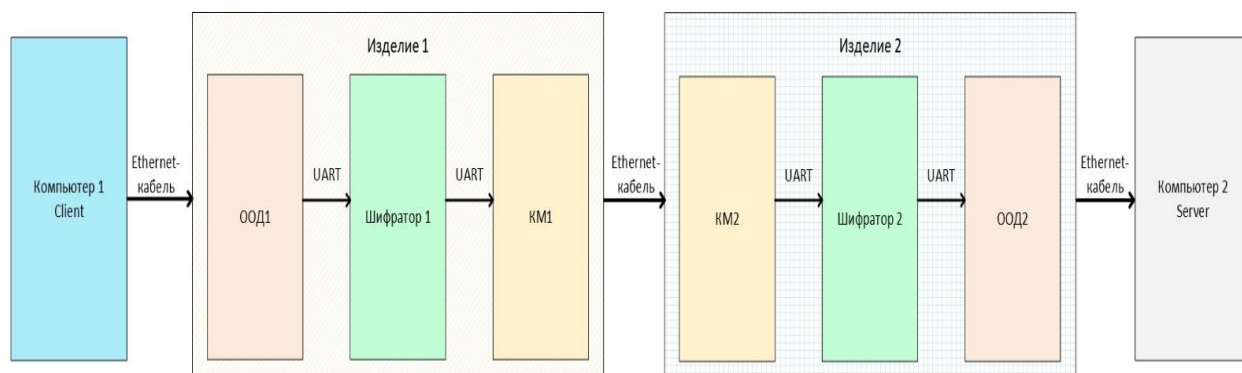


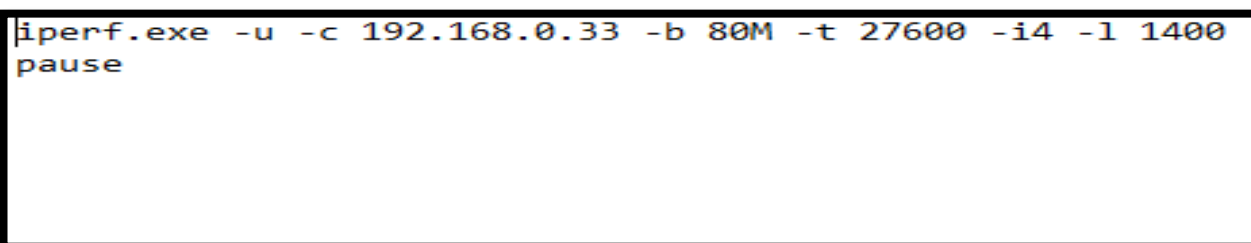
Рисунок 24 – Схема стенда для проверки быстродействия

На схеме показан следующий путь пакета информации:

- передача пакета с компьютера 1 в изделие 1 по интерфейсу Ethernet;
- прием пакета по интерфейсу Ethernet в ПЛИС интерфейса ООД изделия 1 от компьютера 1;
- передача пакета из ПЛИС интерфейса ООД изделия 1 в ПЛИС шифратора изделия 1 по UART;
- зашифрование пакета в ПЛИС шифратора изделия 1;
- передача зашифрованного пакета из ПЛИС шифратора изделия 1 в ПЛИС интерфейса КМ изделия 1 по интерфейсу UART;

- передача зашифрованного пакета из ПЛИС интерфейса КМ изделия 1 в ПЛИС интерфейса КМ изделия 2 по интерфейсу Ethernet;
- прием зашифрованного пакета из ПЛИС интерфейса КМ изделия 1 в ПЛИС интерфейса КМ изделия 2 по интерфейсу Ethernet;
- передача зашифрованного пакета в ПЛИС шифратора изделия 2 от ПЛИС интерфейса КМ изделия 2 по интерфейсу UART;
- расшифрование пакета в ПЛИС шифратора изделия 2;
- передача расшифрованного пакета из ПЛИС шифратора изделия 2 в ПЛИС интерфейса КМ изделия 2 по интерфейсу UART;
- передача зашифрованного пакета из ПЛИС интерфейса КМ изделия 2 в компьютер 2 по интерфейсу Ethernet.

На первом компьютере была запущена клиентская часть `iperf` с параметрами, представленными на рисунке 25.



```
iperf.exe -u -c 192.168.0.33 -b 80M -t 27600 -i4 -l 1400
pause
```

Рисунок 25 – Параметры запуска `iperf`

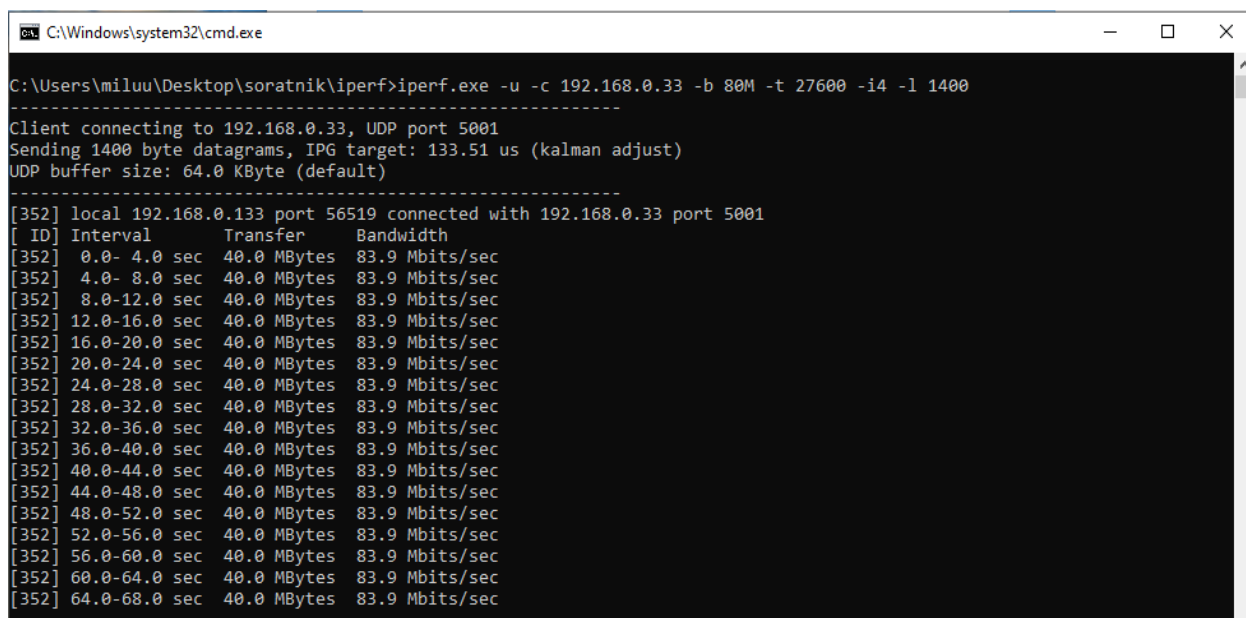
Параметры запуска утилиты `iperf`:

- ключ «- u» обозначает, что передача генерируемого сетевого трафика будет осуществляться с использованием протокола транспортного уровня UDP;
- ключ с параметром «- c 192.168.0.33» обозначает, что на данном устройстве программа запущена в режиме клиента и подключена к серверу с `ip`-адресом 192.168.0.33;
- ключ с параметром «- b 80M» обозначает скорость, с которой необходимо осуществлять передачу генерируемого трафика в Мбит/с;
- ключ с параметром «- t 27600» обозначает время, в течении которого будет осуществляться передача генерируемого трафика;

– ключ «-i4» задаёт интервал, с которым будут выводиться сообщения;

– ключ с параметром «-l 1400» задаёт размер информационной части генерируемых пакетов равный 1400 байтам.

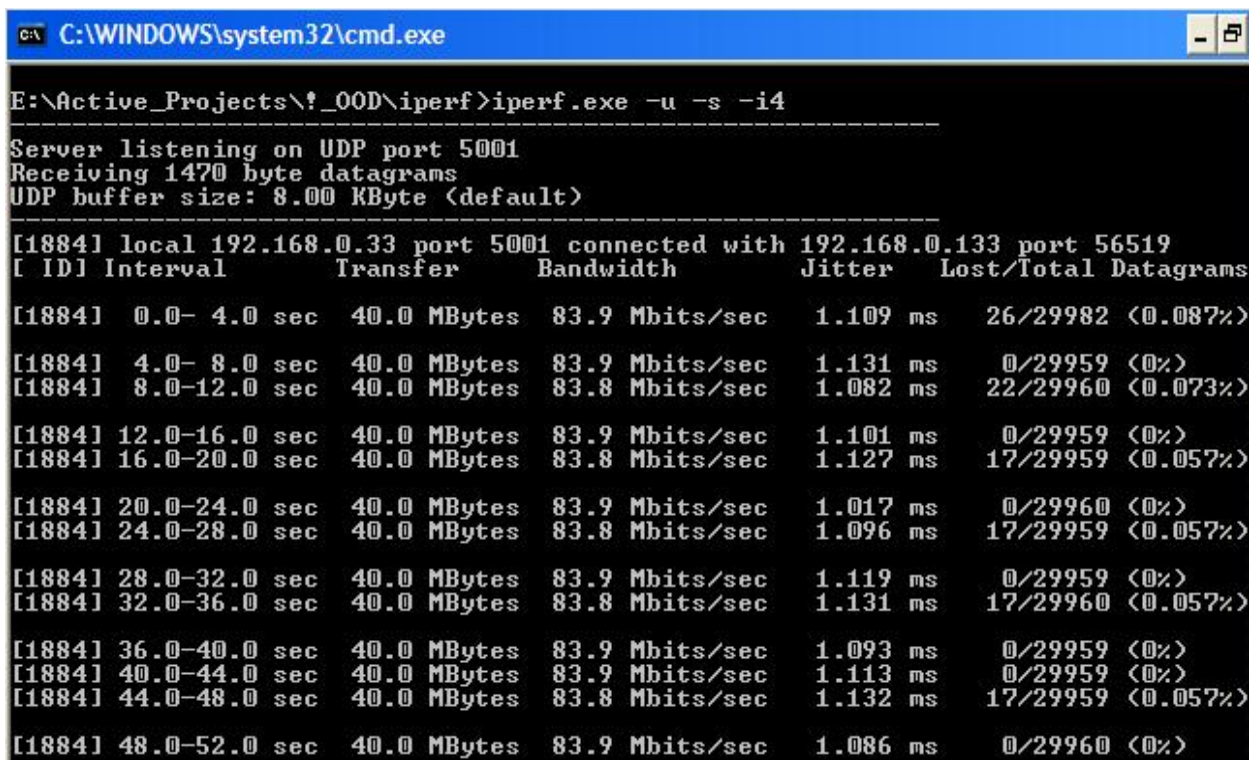
Окно клиентской части программы iperf представлено на рисунке 26.



```
C:\Windows\system32\cmd.exe
C:\Users\miluu\Desktop\soratnik\iperf>iperf.exe -u -c 192.168.0.33 -b 80M -t 27600 -i4 -l 1400
-----
Client connecting to 192.168.0.33, UDP port 5001
Sending 1400 byte datagrams, IPG target: 133.51 us (kalman adjust)
UDP buffer size: 64.0 KByte (default)
-----
[352] local 192.168.0.133 port 56519 connected with 192.168.0.33 port 5001
[ ID] Interval      Transfer      Bandwidth
[352] 0.0- 4.0 sec  40.0 MBytes  83.9 Mbits/sec
[352] 4.0- 8.0 sec  40.0 MBytes  83.9 Mbits/sec
[352] 8.0-12.0 sec  40.0 MBytes  83.9 Mbits/sec
[352] 12.0-16.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 16.0-20.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 20.0-24.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 24.0-28.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 28.0-32.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 32.0-36.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 36.0-40.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 40.0-44.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 44.0-48.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 48.0-52.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 52.0-56.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 56.0-60.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 60.0-64.0 sec 40.0 MBytes  83.9 Mbits/sec
[352] 64.0-68.0 sec 40.0 MBytes  83.9 Mbits/sec
```

Рисунок 26 – Клиентская сторона iperf

Окно серверной части программы iperf представлено на рисунке 27.



```
C:\WINDOWS\system32\cmd.exe
E:\Active_Projects\!\_OOD\iperf>iperf.exe -u -s -i4
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 8.00 KByte (default)
-----
[1884] local 192.168.0.33 port 5001 connected with 192.168.0.133 port 56519
[ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
[1884] 0.0- 4.0 sec  40.0 MBytes  83.9 Mbits/sec  1.109 ms    26/29982 <0.087%>
[1884] 4.0- 8.0 sec  40.0 MBytes  83.9 Mbits/sec  1.131 ms     0/29959 <0%>
[1884] 8.0-12.0 sec  40.0 MBytes  83.8 Mbits/sec  1.082 ms    22/29960 <0.073%>
[1884] 12.0-16.0 sec 40.0 MBytes  83.9 Mbits/sec  1.101 ms     0/29959 <0%>
[1884] 16.0-20.0 sec 40.0 MBytes  83.8 Mbits/sec  1.127 ms    17/29959 <0.057%>
[1884] 20.0-24.0 sec 40.0 MBytes  83.9 Mbits/sec  1.017 ms     0/29960 <0%>
[1884] 24.0-28.0 sec 40.0 MBytes  83.8 Mbits/sec  1.096 ms    17/29959 <0.057%>
[1884] 28.0-32.0 sec 40.0 MBytes  83.9 Mbits/sec  1.119 ms     0/29959 <0%>
[1884] 32.0-36.0 sec 40.0 MBytes  83.8 Mbits/sec  1.131 ms    17/29960 <0.057%>
[1884] 36.0-40.0 sec 40.0 MBytes  83.9 Mbits/sec  1.093 ms     0/29959 <0%>
[1884] 40.0-44.0 sec 40.0 MBytes  83.9 Mbits/sec  1.113 ms     0/29959 <0%>
[1884] 44.0-48.0 sec 40.0 MBytes  83.8 Mbits/sec  1.132 ms    17/29959 <0.057%>
[1884] 48.0-52.0 sec 40.0 MBytes  83.9 Mbits/sec  1.086 ms     0/29960 <0%>
```

Рисунок 27 – Серверная сторона iperf

Как видно по рисункам 26 и 26 фактическая скорость передачи информации, рассчитанная на сервере, совпадает заданной на клиенте и равна 83,9 Мбит/с. Также в окне программы показано отношение потерянных за интервал в 4 секунды пакетов к общему количеству отправленных за этот интервал времени пакетов, а также их процентное соотношение. Процент потерянных пакетов не превышает 0,1 процента. Низкий процент потерянных пакетов связан, в том числе с тем, что в разработанном модуле используется буфер, способный хранить в себе до 16 пакетов, что уменьшает вероятность отбрасывания пакетов в случае возможных задержек, которые могут возникнуть при их обработке.

На сервере была запущена программа анализатор трафика Wireshark для компьютерных сетей Ethernet. Программа позволяет в режиме реального времени анализировать входящий и исходящий с устройства трафик, просматривать содержимое пакетов, сохранять результаты анализа, а также выполнять фильтрация анализируемой информации. С помощью Wireshark был просмотрен один из IP-пакетов, переданных от клиента.

На рисунке 28 представлено содержание пакета, на нём можно увидеть, что размер принятого пакета составляет 1442 байт, размер информационной части составляет 1400 байт, оставшиеся 42 байта являются заголовком кадра. Для передачи информации на канальном уровне использовался протокол Ethernet, на сетевом уровне использовался протокол IPv4, а на транспортном использовался протокол UDP. Указанные в протоколе ip адреса источника (192.168.0.133) и адресата (192.168.0.33) сообщения совпадают с адресами клиента и сервера соответственно. В поле Data отображается переданная внутри сгенерированного пакета информация. Данные о пакете, полученные при помощи программы Wireshark подтверждают, что сгенерированный трафик соответствует параметрам, заданным программе iperf при её запуске.

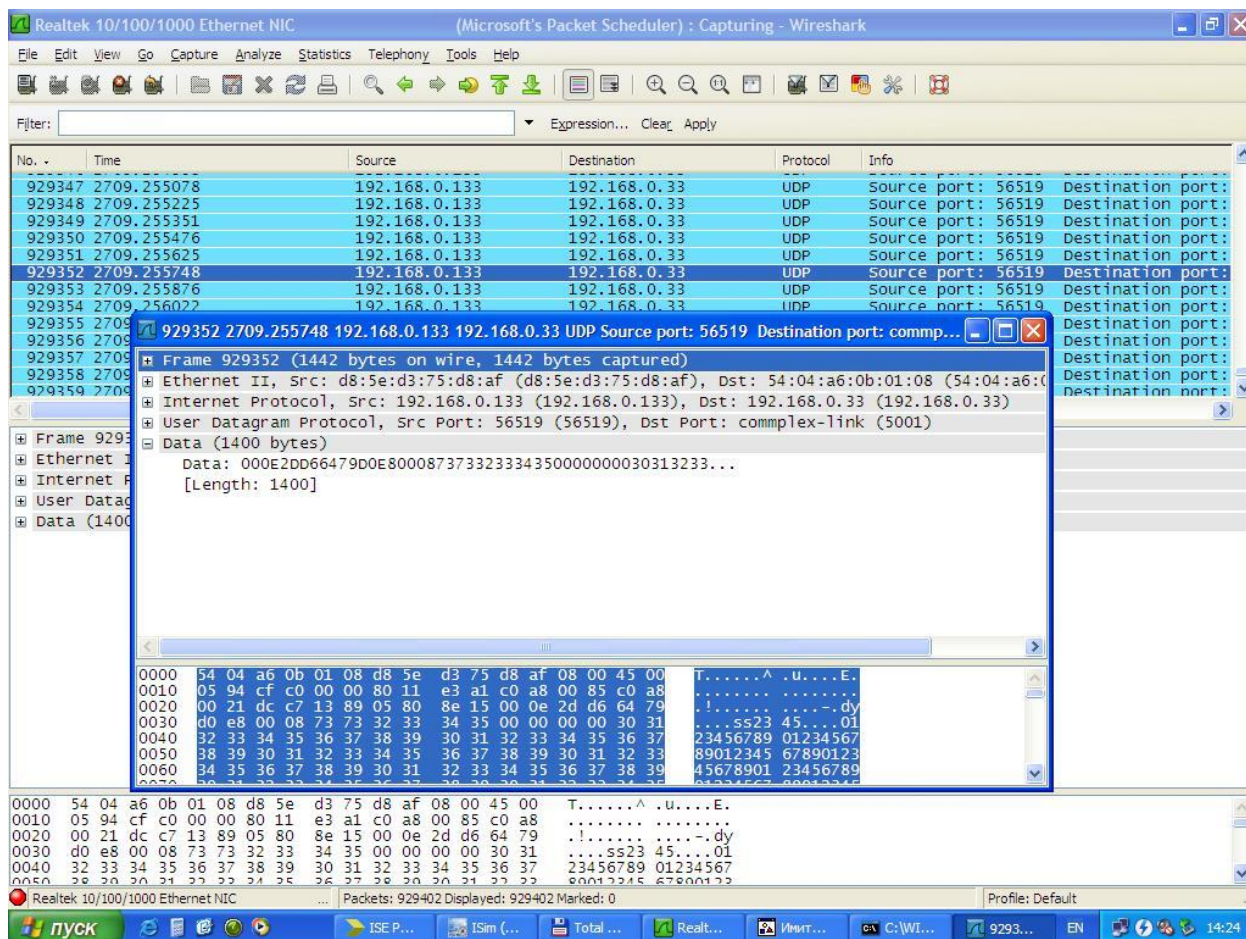


Рисунок 28 – Содержание принятого пакета

Проверка быстродействия была выполнена для различных размеров информационных пакетов. В таблице 5 представлены размеры генерируемых программой `iperf` пакетов и соответствующие им максимальные и минимальные скорости передачи и приёма пакета. Как видно по результатам, представленным в таблице, скорость передачи и приёма растёт с увеличением размера пакета до тех пор, пока не достигает заданной. Это связано с тем, что на малых размерах пакета возникает большая избыточность в передаваемом сообщении, за счёт добавления служебных байт, необходимых для работы по внутреннему протоколу, из-за чего пропускная способность уменьшается. При передаче пакетов большого размера добавляемые байты практически не вносят избыточности и мало влияют на пропускную способность канала.

Таблица 5 – Проверка быстродействия на пакетах разного размер

Размер пакета, байт	Скорость передачи, Мбит/с		Скорость приёма, Мбит/с	
	min	max	min	max
50	40,8	42,6	7,16	9,52
100	59,2	60,3	23,5	25,3
150	69,4	69,5	53,1	55
200	75,1	75,3	61,2	62
210	76	76,2	61,8	63,5
220	76,9	77	62,6	65,1
230	77,6	77,8	64,7	66
240	78,4	78,5	66,2	67,2
250	79	79,2	66,2	67
270	80,4	80,5	66,8	69,2
280	80,9	81,1	69	69,4
300	81,9	82	69,5	70,5
400	83,9	83,9	74,3	74,6
600	83,9	83,9	78,7	78,7
700	83,9	83,9	79,7	79,8
800	83,9	83,9	80,9	81
1000	83,9	83,9	82,3	82,6
1200	83,9	83,9	83,3	83,5
1400	83,9	83,9	83,8	84

8 Вредные психофизиологические факторы, влияющие на органы зрения специалиста по защите информации. Методы и средства физической культуры, снижающие их воздействие

Специалисты по информационной безопасности в ходе выполнения своих профессиональных и должностных обязанностей подвергаются различным воздействиям, которые негативно влияют на их здоровье.

Работа специалиста по защите информации связана с длительным пребыванием за компьютером, чтением множества документов и сканированием информации на экране монитора. Все это может привести к различным заболеваниям органов зрения.

Одним из главных факторов, влияющих на зрение, является сильный свет, который идет от монитора. При этом, если монитор находится в темном помещении, то глаза вынуждены адаптироваться к яркости монитора, что также негативно сказывается на зрении. Кроме того, мониторы имеют низкую частоту обновления изображения, что может вызывать усталость глаз и головную боль.

Еще одним фактором, влияющим на зрение, является утомление глаз при длительном чтении мелкого шрифта и сканировании большого количества информации. При этом глаза вынуждены работать на пределе возможностей, что также может привести к различным заболеваниям.

Компьютерный зрительный синдром – это симптомокомплекс, объединяющий признаки астенопии и синдрома «сухого глаза», возникающий в результате продолжительной работы с компьютером или другим электронным устройством. Данный синдром может привести к зрительному утомлению, развитию привычного напряжения аккомодации, спазма аккомодации [10]. В дальнейшем значительно снижается работоспособность человека, появляются головные боли, раздражительность, снижение остроты зрения.

Астенопия (усталость глаз) - является разновидностью компьютерного синдрома. Резь и боль в глазах, слезотечение, покраснение век и глазных яблок, боль в области лба, быстрая утомляемость - все это явные признаки так называемой астенопии, нередко возникающей у пользователей компьютеров. Быстрое утомление глаз наступает во время зрительной работы, особенно при малом расстоянии от глаза до объекта.

«Поражающими» факторами, вызывающими чрезмерное зрительное утомление при компьютерном зрительном синдроме, служат:

- неправильная организация рабочего места;
- проблемы качества изображения на мониторе компьютера (или другого устройства);
- сложности зрительного восприятия изображения проекционного типа;
- влияние на организм пользователя (оператора) электромагнитного излучения компьютера.

Наиболее значимая причина астенопии – неправильная организация рабочего места пользователя компьютера и нарушение эргономики зрительного труда. Исследования показывают, что оптимальным для работы за компьютером является расположение центра монитора на расстоянии от 10 до 25 см ниже горизонтальной линии взора при рабочей дистанции до монитора от 50 до 70 см. При этом угол взора, по разным данным, должен быть не менее 15 или 20 градусов.

Синдром сухого глаза - состояние недостаточной увлажненности поверхности роговицы и конъюнктивы вследствие нарушения качества и количества слезной жидкости и нестабильности слезной пленки. При работе за монитором человек предельно сконцентрирован и, соответственно, интервалы моргания увеличиваются, что приводит к высыханию поверхности глаза. Дополнительными факторами, провоцирующим появления синдрома, помимо длительного и непрерывного пребывания за

экраном монитор, являются запыленность помещений и работа кондиционеров.

Симптомы синдрома сухого глаза:

- раздражение (покраснение) глаз;
- ощущение инородного тела в глазах;
- чувство жжения;
- слизистое отделение в виде нитей;
- периодическое «затуманивание»;
- постепенное снижение остроты зрения;
- светобоязнь.

При регулярной и длительной работе с персональным компьютером необходимо пользоваться рядом правил для уменьшения компьютерного зрительного синдрома.

Должно быть организовано комфортное рабочее место. Оно должно быть достаточно освещено, световое поле равномерно распределено по всей площади рабочего пространства, лучи света не должны попадать прямо в глаза.

Необходимо использовать компьютер с хорошим монитором и правильными настройками. Размеры монитора по диагонали лучше выбирать от 50 см и выше. На мониторе (не жидкокристаллический дисплей) необходимо установить защиту от мелькания, выбрав соответствующие параметры. В частности, необходимо выбрать максимальную частоту обновления экрана (не менее 85 Гц). Для жидкокристаллических экранов достаточно 60 Гц – их система сама поддерживает оптимальную частоту.

Оптимальным цветовым сочетанием для чтения считается традиционное – черный текст на белом фоне. Размер шрифта стоит устанавливать в интервале от 12 до 14. Необходимо поддерживать чистоту экрана от пыли и налета, так как они ухудшают четкость изображения.

Экран монитора стоит устанавливать таким образом, чтобы его центр находился на расстоянии от 10 до 20 см ниже глаз пользователя.

Расстояние от пользователя до экрана – от 60 до 70 см, при этом клавиатура, наоборот, должна располагаться в непосредственной близости в 30 - 40 см.

Экран монитора должен соответствовать следующим требованиям:

- яркость экрана должна быть не менее 100 кд/м²;
- освещенность поверхности экрана не должна быть более 300 лк;
- яркость бликов на экране — не более 40 кд/м²;
- размер светящейся точки — не более 0,56 (для цветного дисплея);
- контраст изображения знака — не менее 0,8.

Для более комфортного восприятия из настроек рабочего стола лучше исключить сине-фиолетовые цвета (они являются наиболее утомительными для глаз), а предпочтение отдать зеленовато-коричневой гамме.

Время работы за компьютером, если возможно, не более 4 часов в день.

Необходимо делать обязательные паузы во время работы на близком расстоянии от монитора через каждые 20 - 30 минут.

Полезно устраивать перерывы на две-три минуты при работе на компьютере с применением комплекса упражнений для глаз один раз в час:

- каждые 2 часа рекомендуется делать упражнения на расслабление мышц глаз, шеи и спины. Покрутить головой, подвигать плечами, несколько раз наклониться. Это позволяет сбросить накопившееся напряжение;
- рекомендуется пару минут расслабленно посмотреть за окном вдаль;
- с закрытыми глазами выполнять вращения по кругу глазными яблоками по часовой и против часовой стрелки. Четыре раза в одном направлении и четыре в другом. Веки должны быть расслабленными;
- с закрытыми глазами смотреть вверх, затем вниз. Повторить восемь раз;
- взгляд прямо перед собой, после 30 секунд быстрого моргания.

Повторить три раза.

Можно использовать специальные капли для глаз, гели комфортности, замещающие слезу для защиты роговицы от пересыхания и уменьшения дискомфорта при длительной работе за компьютером.

В случаях постоянного нахождения в помещениях с работающими кондиционерами или вентиляторами, необходимо направлять воздушную струю в сторону, для того чтобы не увеличивать испарение влаги со слизистой глаза.

Необходимо регулярно посещать офтальмолога, не реже чем раз в год. Регулярная проверка зрения специалистом поможет выявить возможные проблемы на ранних стадиях и принять меры профилактики [11].

Заключение

В процессе выполнения дипломного проекта был произведена разработка программного модуля высокоскоростного интерфейса обработки данных на ПЛИС в составе СКЗИ.

В процессе выполнения работы был произведен обзор трёх различных вариантов реализации интерфейса UART. В результате обзора и сравнения реализаций было принято решение о необходимости разработки программного модуля высокоскоростного интерфейса обработки информации. Разработана общая структура ПЛИС интерфейса, а также структуры процессов приёма и передачи информации.

Кроме того, был описан внутренний протокол обмена данными ПЛИС интерфейса с шифратором, в том числе и алгоритм расчета контрольной суммы CRC.

На заключительном этапе было разработан программный модуль высокоскоростного интерфейса обработки информации для СКЗИ на ПЛИС на языке VHDL, реализующий обмен информацией между ООД и шифратором и КМ и шифратором, проверена его работоспособность с помощью среды моделирования ISim. Была выполнена проверка скорости работы интерфейса с помощью программного обеспечения ipref, предварительно была разработана схема тестового стенда.

В результате выполнения выпускной квалификационной работы был разработан программный модуль высокоскоростного интерфейса обработки данных готовый к использованию в составе изделия «АКСИОН (СКЗИ-3)».

Таким образом, техническое задание на дипломное проектирование выполнено в полном объеме.

Список использованных источников

- 1 Долин, С. Великий и могучий UART /С. Долин// Журнал «Хакер». – 2009. – №125. – С. 114-118.
- 2 Бибило П.Н., Основы языка VHDL /Бибило П.Н. – М.: СОЛОН-Р, 2002 – 224 с.: ил.
- 3 Sardi Irfansyah, DESIGN AND IMPLEMENTATION OF UART WITH FIFO BUFFER USING VHDL ON FPGA, Department of Electrical Engineering, Gunadarma University – 2019 URL: https://ictactjournals.in/paper/IJME_Vol_5_Iss_1_Paper_4_724_730.pdf (дата обращения к ресурсу: 25.02.2023)
- 4 Nick Sawyer, XAPP224 (v2.5) Data Recovery // AMD Xilinx Inc. URL: <http://www.xilinx.com/> (дата обращения к ресурсу: 28.03.2023)
- 5 Технология Ethernet. Обзор, описание, формат кадра // MicroTechnics URL: <https://microtechnics.ru/tehnologiya-ethernet-obzor-opisanie-format-kadra/> (дата обращения к ресурсу: 20.04.2023)
- 6 Зотов В., Особенности архитектуры нового поколения ПЛИС FPGA фирмы Xilinx серии Spartan 6 // Компоненты и технологии. URL: <https://kit-e.ru/fpga/osobennosti-arhitektury-novogo-pokoleniya-plis-fpga-firmy-xilinx-serii-spartan-6>. №9 2009 г. (дата обращения к ресурсу: 23.03.2023)
- 7 Как посчитать контрольную сумму CRC32, CRC16, CRC8? // Soltau URL: <https://soltau.ru/index.php/themes/dev/item/461-kak-> (дата обращения к ресурсу: 05.05.2023)
- 8 Быстрое вычисление контрольной суммы CRC: таблица против матрицы / А.Н. Мальчуков, А. Н. Осокин // Журнал «Прикладная информатика». - 2010. - №2. - С. 58-63.
- 9 Ross N. Williams, A painless guide to CRC error detection algorithms, Rocksoft Pty Ltd, Australia – 1993 URL: <https://docs.yandex.ru/docs/view?tm=1684879853&tld=ru&lang=ru&name=crcguide.pdf> (дата обращения к ресурсу: 05.05.2023)

10 Бржеский, В.В. Компьютерный зрительный синдром: четверть века противоречий / В.В. Бржеский, Т.Н. Воронцова, С.Ю. Голубев // Офтальмологический портал URL: <https://organum-visus.ru/> (дата обращения: 21.05.2023).

11 Исакова, Е.В. Работа с компьютером и компьютерный зрительный синдром / Е.В. Исакова// Вятский медицинский вестник. - 2011. - №3-4. - С. 32-35.

Приложение А

(обязательное)

Код узла приема данных на языке vhd1

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library unisim;
use unisim.vcomponents.all;

entity reciver_node is
port
(
    Clk128: in std_logic;
    Clk128_90: in std_logic;
    Clk128_180: in std_logic;
    Clk128_270: in std_logic;
    data: in std_logic;
    rst: in std_logic;
    recive_out: out std_logic_vector (7 downto 0);
    DataStrobEn: out std_logic;
    Frame_Error: out std_logic
);
end reciver_node;

```

architecture behavioral of reciver_node is

```

signal az: std_logic_vector (3 downto 0);
signal bz: std_logic_vector (3 downto 0);
signal cz: std_logic_vector (3 downto 0);
signal dz: std_logic_vector (3 downto 0);
signal buf_in: std_logic_vector (7 downto 0);
signal buf_in_1: std_logic_vector (3 downto 0);
signal buf_in_2: std_logic_vector (3 downto 0);

```

```

signal reg_first: std_logic_vector (7 downto 0);
signal reg_second: std_logic_vector (7 downto 0) ;
signal en_phazeA: std_logic;
signal en_phazeB: std_logic;
signal en_phazeC: std_logic;
signal en_phazeD: std_logic;
signal en_A: std_logic;
signal en_B: std_logic;
signal en_C: std_logic;
signal en_D: std_logic;
signal phazeA: std_logic;
signal phazeB: std_logic;
signal phazeC: std_logic;
signal phazeD: std_logic;
signal recive_en: std_logic;
signal byte_ready: std_logic;
signal data_in_regf: std_logic;
signal synres: std_logic;
signal count: std_logic_vector(3 downto 0);

```

```

attribute RLOC: string;
attribute IOB: string;
attribute RLOC of ff_a0: label is "X0Y0";
attribute RLOC of ff_b0: label is "X1Y0";
attribute RLOC of ff_c0: label is "X2Y0";
attribute RLOC of ff_d0: label is "X3Y0";
attribute IOB of ff_a0: label is "FALSE";
attribute IOB of ff_b0: label is "FALSE";
attribute IOB of ff_c0: label is "FALSE";
attribute IOB of ff_d0: label is "FALSE";

```

begin

buf_in_1 <= dz(3) & cz(3) & bz(3) & az(3);

en_phazeA<=(buf_in(0) and buf_in(1) and buf_in(2) and not buf_in(3) and not buf_in(4)) ;

en_phazeB<=(buf_in(1) and buf_in(2) and buf_in(3) and not buf_in(4) and not buf_in(5)) ;

en_phazeC<=(buf_in(2) and buf_in(3) and buf_in(4) and not buf_in(5) and not buf_in(6)) ;

en_phazeD<=(buf_in(3) and buf_in(4) and buf_in(5) and not buf_in(6) and not buf_in(7)) ;

en_A <= (not phazeB and not phazeC and not phazeD);

en_B <= (not phazeA and not phazeC and not phazeD);

en_C <= (not phazeA and not phazeB and not phazeD ;

en_D <= (not phazeA and not phazeB and not phazeC);

Rew_buf: process (Clk128, rst)

begin

if rst = '1' then

buf_in_2 <= (others=>'0');

elsif Clk128'event and Clk128 = '1' then

buf_in_2 <= buf_in_1;

end if;

end process Rew_buf;

buf_in <= buf_in_1 & buf_in_2;

```

Aphaze: process (Clk128, rst)
begin
    if rst = '1' then
        phazeA <= '0';
    elsif Clk128'event and Clk128 = '1' then
        if en_A = '1' then
            phazeA<= ((recive_en or en_phazeA) and not
byte_ready);
        end if;
    end if;
end process Aphaze;

```

```

Bphaze: process (Clk128, rst)
begin
    if rst = '1' then
        phazeB <= '0';
    elsif Clk128'event and Clk128 = '1' then
        if en_B = '1' then
            phazeB<= ((recive_en or en_phazeB) and not
byte_ready);
        end if;
    end if;
end process Bphaze;

```

```

Cphaze: process (Clk128, rst)
begin
    if rst = '1' then
        phazeC <= '0';
    elsif Clk128'event and Clk128 = '1' then
        if en_C = '1' then

```

```

                                phazeC<= ((recive_en or en_phazeC) and not
byte_ready);

                                end if;

                                end if;

                                end process Cphaze;
Dphaze: process (Clk128, rst)
begin
    if rst = '1' then
        phazeD <= '0';
    elsif Clk128'event and Clk128 = '1' then
        if en_D = '1' then
            phazeD<= ((recive_en or en_phazeD) and not
byte_ready);

                                end if;

                                end if;

                                end process Dphaze;

recive_en<=(phazeA or phazeB or phazeC or phazeD) ;

data_in_regf<=(buf_in(4) and phazeA) or (buf_in(5) and phazeB) or
(buf_in(6) and phazeC) or (buf_in(7) and phazeD) ;

freg: process (Clk128, rst)
begin
    if recive_en = '0' then
        reg_first <= (others => '0');
    elsif Clk128'event and Clk128 = '1' then
        if recive_en = '1' then
            reg_first <=data_in_regf & reg_first(7 downto 1);
        end if;
    end if;
end process freg;

```

```

        end if;
    end process freg;

sreg: process (Clk128, rst)
begin
    if rst = '1' then
        reg_second<= (others => '0');
    elsif Clk128'event and Clk128 = '1' then
        if byte_ready = '1' then
            reg_second<=reg_first;
        end if;
    end if;
end process sreg;

recv_out<=reg_second;

counter: process (Clk128, rst)
begin
    if recv_en = '0' or byte_ready = '1' then
        count <= (others => '0');
    elsif Clk128'event and Clk128 = '1' then
        if recv_en = '1' then
            count<=count + 1;
        end if;
    end if;
end process counter;

synres <= count(0) and count(1) and count(2) and not count(3);
br_delay: process (Clk128, rst)
begin
    if rst = '1' then

```

```

        byte_ready <= '0';
    elsif Clk128'event and Clk128 = '1' then
        byte_ready<=synres;
    end if;
end process br_delay;

error: process (Clk128, rst)
begin
    if rst = '1' then
        Frame_Error<= '0';
    elsif Clk128'event and Clk128 = '1' then
        if (byte_ready and not data_in_regf) = '1' then
            Frame_Error<= '1';
        end if;
    end if;
end process error;

```

```

DataStrobEn<=byte_ready;

```

```

ff_a0: fdc port map (d => data, c => Clk128, clr => rst, q => az(0));
ff_a1: fdc port map (d => az(0), c => Clk128, clr => rst, q => az(1));
ff_a2: fdc port map (d => az(1), c => Clk128, clr => rst, q => az(2));
ff_a3: fdc port map (d => az(2), c => Clk128, clr => rst, q => az(3));

```

```

ff_b0: fdc port map (d => data, c => Clk128_90, clr => rst, q => bz(0));
ff_b1: fdc port map (d => bz(0), c => Clk128, clr => rst, q => bz(1));
ff_b2: fdc port map (d => bz(1), c => Clk128, clr => rst, q => bz(2));
ff_b3: fdc port map (d => bz(2), c => Clk128, clr => rst, q => bz(3));

```

```

ff_c0: fdc port map (d => data, c => Clk128_180, clr => rst, q => cz(0));

```

```

ff_c1: fdc port map (d => cz(0), c => Clk128_90, clr => rst, q => cz(1));
ff_c2: fdc port map (d => cz(1), c => Clk128, clr => rst, q => cz(2));
ff_c3: fdc port map (d => cz(2), c => Clk128, clr => rst, q => cz(3));

ff_d0: fdc port map (d => data, c => Clk128_270, clr => rst, q => dz(0));
ff_d1: fdc port map (d => dz(0), c => Clk128_180, clr => rst, q => dz(1));
ff_d2: fdc port map (d => dz(1), c => Clk128_90, clr => rst, q => dz(2));
ff_d3: fdc port map (d => dz(2), c => Clk128, clr => rst, q => dz(3));

end behavioral;

```

Приложение Б

(обязательное)

Код узла расчёта контрольной суммы на языке vhd1

Содержание файла Tabl_CRC_16_mux.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Tabl_CRC_16_mux is
  Port
  (
    adr_tabl: in std_logic_vector (7 downto 0);
    Tabl_CRC_16_out: out std_logic_vector (15 downto 0)
  );
end Tabl_CRC_16_mux;

architecture Behavioral of Tabl_CRC_16_mux is

begin

  Tabl_CRC_16_out <=
    x"0000"    when adr_tabl = X"00"  else
    x"C0C1"    when adr_tabl = x"01"  else
    x"C181"    when adr_tabl = x"02"  else
    x"0140"    when adr_tabl = x"03"  else
    x"C301"    when adr_tabl = x"04"  else
    x"03C0"    when adr_tabl = x"05"  else
    x"0280"    when adr_tabl = x"06"  else
    x"C241"    when adr_tabl = x"07"  else
    x"C601"    when adr_tabl = x"08"  else
    x"06C0"    when adr_tabl = x"09"  else
    x"0780"    when adr_tabl = x"0A"  else
```

x"C741"	when adr_tabl = x"0B"	else
x"0500"	when adr_tabl = x"0C"	else
x"C5C1"	when adr_tabl = x"0D"	else
x"C481"	when adr_tabl = x"0E"	else
x"0440"	when adr_tabl = x"0F"	else
x"CC01"	when adr_tabl = x"10"	else
x"0CC0"	when adr_tabl = x"11"	else
x"0D80"	when adr_tabl = x"12"	else
x"CD41"	when adr_tabl = x"13"	else
x"0F00"	when adr_tabl = x"14"	else
x"CFC1"	when adr_tabl = x"15"	else
x"CE81"	when adr_tabl = x"16"	else
x"0E40"	when adr_tabl = x"17"	else
x"0A00"	when adr_tabl = x"18"	else
x"CAC1"	when adr_tabl = x"19"	else
x"CB81"	when adr_tabl = x"1A"	else
x"0B40"	when adr_tabl = x"1B"	else
x"C901"	when adr_tabl = x"1C"	else
x"09C0"	when adr_tabl = x"1D"	else
x"0880"	when adr_tabl = x"1E"	else
x"C841"	when adr_tabl = x"1F"	else
x"D801"	when adr_tabl = x"20"	else
x"18C0"	when adr_tabl = x"21"	else
x"1980"	when adr_tabl = x"22"	else
x"D941"	when adr_tabl = x"23"	else
x"1B00"	when adr_tabl = x"24"	else
x"DBC1"	when adr_tabl = x"25"	else
x"DA81"	when adr_tabl = x"26"	else
x"1A40"	when adr_tabl = x"27"	else
x"1E00"	when adr_tabl = x"28"	else

x"DEC1"	when adr_tabl = x"29"	else
x"DF81"	when adr_tabl = x"2A"	else
x"1F40"	when adr_tabl = x"2B"	else
x"DD01"	when adr_tabl = x"2C"	else
x"1DC0"	when adr_tabl = x"2D"	else
x"1C80"	when adr_tabl = x"2E"	else
x"DC41"	when adr_tabl = x"2F"	else
x"1400"	when adr_tabl = x"30"	else
x"D4C1"	when adr_tabl = x"31"	else
x"D581"	when adr_tabl = x"32"	else
x"1540"	when adr_tabl = x"33"	else
x"D701"	when adr_tabl = x"34"	else
x"17C0"	when adr_tabl = x"35"	else
x"1680"	when adr_tabl = x"36"	else
x"D641"	when adr_tabl = x"37"	else
x"D201"	when adr_tabl = x"38"	else
x"12C0"	when adr_tabl = x"39"	else
x"1380"	when adr_tabl = x"3A"	else
x"D341"	when adr_tabl = x"3B"	else
x"1100"	when adr_tabl = x"3C"	else
x"D1C1"	when adr_tabl = x"3D"	else
x"D081"	when adr_tabl = x"3E"	else
x"1040"	when adr_tabl = x"3F"	else
x"F001"	when adr_tabl = x"40"	else
x"30C0"	when adr_tabl = x"41"	else
x"3180"	when adr_tabl = x"42"	else
x"F141"	when adr_tabl = x"43"	else
x"3300"	when adr_tabl = x"44"	else
x"F3C1"	when adr_tabl = x"45"	else
x"F281"	when adr_tabl = x"46"	else

x"3240"	when adr_tabl = x"47"	else
x"3600"	when adr_tabl = x"48"	else
x"F6C1"	when adr_tabl = x"49"	else
x"F781"	when adr_tabl = x"4A"	else
x"3740"	when adr_tabl = x"4B"	else
x"F501"	when adr_tabl = x"4C"	else
x"35C0"	when adr_tabl = x"4D"	else
x"3480"	when adr_tabl = x"4E"	else
x"F441"	when adr_tabl = x"4F"	else
x"3C00"	when adr_tabl = x"50"	else
x"FCC1"	when adr_tabl = x"51"	else
x"FD81"	when adr_tabl = x"52"	else
x"3D40"	when adr_tabl = x"53"	else
x"FF01"	when adr_tabl = x"54"	else
x"3FC0"	when adr_tabl = x"55"	else
x"3E80"	when adr_tabl = x"56"	else
x"FE41"	when adr_tabl = x"57"	else
x"FA01"	when adr_tabl = x"58"	else
x"3AC0"	when adr_tabl = x"59"	else
x"3B80"	when adr_tabl = x"5A"	else
x"FB41"	when adr_tabl = x"5B"	else
x"3900"	when adr_tabl = x"5C"	else
x"F9C1"	when adr_tabl = x"5D"	else
x"F881"	when adr_tabl = x"5E"	else
x"3840"	when adr_tabl = x"5F"	else
x"2800"	when adr_tabl = x"60"	else
x"E8C1"	when adr_tabl = x"61"	else
x"E981"	when adr_tabl = x"62"	else
x"2940"	when adr_tabl = x"63"	else
x"EB01"	when adr_tabl = x"64"	else

x"2BC0"	when adr_tabl = x"65"	else
x"2A80"	when adr_tabl = x"66"	else
x"EA41"	when adr_tabl = x"67"	else
x"EE01"	when adr_tabl = x"68"	else
x"2EC0"	when adr_tabl = x"69"	else
x"2F80"	when adr_tabl = x"6A"	else
x"EF41"	when adr_tabl = x"6B"	else
x"2D00"	when adr_tabl = x"6C"	else
x"EDC1"	when adr_tabl = x"6D"	else
x"EC81"	when adr_tabl = x"6E"	else
x"2C40"	when adr_tabl = x"6F"	else
x"E401"	when adr_tabl = x"70"	else
x"24C0"	when adr_tabl = x"71"	else
x"2580"	when adr_tabl = x"72"	else
x"E541"	when adr_tabl = x"73"	else
x"2700"	when adr_tabl = x"74"	else
x"E7C1"	when adr_tabl = x"75"	else
x"E681"	when adr_tabl = x"76"	else
x"2640"	when adr_tabl = x"77"	else
x"2200"	when adr_tabl = x"78"	else
x"E2C1"	when adr_tabl = x"79"	else
x"E381"	when adr_tabl = x"7A"	else
x"2340"	when adr_tabl = x"7B"	else
x"E101"	when adr_tabl = x"7C"	else
x"21C0"	when adr_tabl = x"7D"	else
x"2080"	when adr_tabl = x"7E"	else
x"E041"	when adr_tabl = x"7F"	else
x"A001"	when adr_tabl = x"80"	else
x"60C0"	when adr_tabl = x"81"	else
x"6180"	when adr_tabl = x"82"	else

x"A141"	when adr_tabl = x"83"	else
x"6300"	when adr_tabl = x"84"	else
x"A3C1"	when adr_tabl = x"85"	else
x"A281"	when adr_tabl = x"86"	else
x"6240"	when adr_tabl = x"87"	else
x"6600"	when adr_tabl = x"88"	else
x"A6C1"	when adr_tabl = x"89"	else
x"A781"	when adr_tabl = x"8A"	else
x"6740"	when adr_tabl = x"8B"	else
x"A501"	when adr_tabl = x"8C"	else
x"65C0"	when adr_tabl = x"8D"	else
x"6480"	when adr_tabl = x"8E"	else
x"A441"	when adr_tabl = x"8F"	else
x"6C00"	when adr_tabl = x"90"	else
x"ACC1"	when adr_tabl = x"91"	else
x"AD81"	when adr_tabl = x"92"	else
x"6D40"	when adr_tabl = x"93"	else
x"AF01"	when adr_tabl = x"94"	else
x"6FC0"	when adr_tabl = x"95"	else
x"6E80"	when adr_tabl = x"96"	else
x"AE41"	when adr_tabl = x"97"	else
x"AA01"	when adr_tabl = x"98"	else
x"6AC0"	when adr_tabl = x"99"	else
x"6B80"	when adr_tabl = x"9A"	else
x"AB41"	when adr_tabl = x"9B"	else
x"6900"	when adr_tabl = x"9C"	else
x"A9C1"	when adr_tabl = x"9D"	else
x"A881"	when adr_tabl = x"9E"	else
x"6840"	when adr_tabl = x"9F"	else
x"7800"	when adr_tabl = x"A0"	else

x"B8C1"	when adr_tabl = x"A1" else
x"B981"	when adr_tabl = x"A2" else
x"7940"	when adr_tabl = x"A3" else
x"BB01"	when adr_tabl = x"A4" else
x"7BC0"	when adr_tabl = x"A5" else
x"7A80"	when adr_tabl = x"A6" else
x"BA41"	when adr_tabl = x"A7" else
x"BE01"	when adr_tabl = x"A8" else
x"7EC0"	when adr_tabl = x"A9" else
x"7F80"	when adr_tabl = x"AA" else
x"BF41"	when adr_tabl = x"AB" else
x"7D00"	when adr_tabl = x"AC" else
x"BDC1"	when adr_tabl = x"AD" else
x"BC81"	when adr_tabl = x"AE" else
x"7C40"	when adr_tabl = x"AF" else
x"B401"	when adr_tabl = x"B0" else
x"74C0"	when adr_tabl = x"B1" else
x"7580"	when adr_tabl = x"B2" else
x"B541"	when adr_tabl = x"B3" else
x"7700"	when adr_tabl = x"B4" else
x"B7C1"	when adr_tabl = x"B5" else
x"B681"	when adr_tabl = x"B6" else
x"7640"	when adr_tabl = x"B7" else
x"7200"	when adr_tabl = x"B8" else
x"B2C1"	when adr_tabl = x"B9" else
x"B381"	when adr_tabl = x"BA" else
x"7340"	when adr_tabl = x"BB" else
x"B101"	when adr_tabl = x"BC" else
x"71C0"	when adr_tabl = x"BD" else
x"7080"	when adr_tabl = x"BE" else

x"B041"	when adr_tabl = x"BF" else
x"5000"	when adr_tabl = x"C0" else
x"90C1"	when adr_tabl = x"C1" else
x"9181"	when adr_tabl = x"C2" else
x"5140"	when adr_tabl = x"C3" else
x"9301"	when adr_tabl = x"C4" else
x"53C0"	when adr_tabl = x"C5" else
x"5280"	when adr_tabl = x"C6" else
x"9241"	when adr_tabl = x"C7" else
x"9601"	when adr_tabl = x"C8" else
x"56C0"	when adr_tabl = x"C9" else
x"5780"	when adr_tabl = x"CA" else
x"9741"	when adr_tabl = x"CB" else
x"5500"	when adr_tabl = x"CC" else
x"95C1"	when adr_tabl = x"CD" else
x"9481"	when adr_tabl = x"CE" else
x"5440"	when adr_tabl = x"CF" else
x"9C01"	when adr_tabl = x"D0" else
x"5CC0"	when adr_tabl = x"D1" else
x"5D80"	when adr_tabl = x"D2" else
x"9D41"	when adr_tabl = x"D3" else
x"5F00"	when adr_tabl = x"D4" else
x"9FC1"	when adr_tabl = x"D5" else
x"9E81"	when adr_tabl = x"D6" else
x"5E40"	when adr_tabl = x"D7" else
x"5A00"	when adr_tabl = x"D8" else
x"9AC1"	when adr_tabl = x"D9" else
x"9B81"	when adr_tabl = x"DA" else
x"5B40"	when adr_tabl = x"DB" else
x"9901"	when adr_tabl = x"DC" else

x"59C0"	when adr_tabl = x"DD" else
x"5880"	when adr_tabl = x"DE" else
x"9841"	when adr_tabl = x"DF" else
x"8801"	when adr_tabl = x"E0" else
x"48C0"	when adr_tabl = x"E1" else
x"4980"	when adr_tabl = x"E2" else
x"8941"	when adr_tabl = x"E3" else
x"4B00"	when adr_tabl = x"E4" else
x"8BC1"	when adr_tabl = x"E5" else
x"8A81"	when adr_tabl = x"E6" else
x"4A40"	when adr_tabl = x"E7" else
x"4E00"	when adr_tabl = x"E8" else
x"8EC1"	when adr_tabl = x"E9" else
x"8F81"	when adr_tabl = x"EA" else
x"4F40"	when adr_tabl = x"EB" else
x"8D01"	when adr_tabl = x"EC" else
x"4DC0"	when adr_tabl = x"ED" else
x"4C80"	when adr_tabl = x"EE" else
x"8C41"	when adr_tabl = x"EF" else
x"4400"	when adr_tabl = x"F0" else
x"84C1"	when adr_tabl = x"F1" else
x"8581"	when adr_tabl = x"F2" else
x"4540"	when adr_tabl = x"F3" else
x"8701"	when adr_tabl = x"F4" else
x"47C0"	when adr_tabl = x"F5" else
x"4680"	when adr_tabl = x"F6" else
x"8641"	when adr_tabl = x"F7" else
x"8201"	when adr_tabl = x"F8" else
x"42C0"	when adr_tabl = x"F9" else
x"4380"	when adr_tabl = x"FA" else

```

        x"8341"    when adr_tabl = x"FB"  else
        x"4100"    when adr_tabl = x"FC"  else
        x"81C1"    when adr_tabl = x"FD"  else
        x"8081"    when adr_tabl = x"FE"  else
        x"4040";
end Behavioral;

```

Содержание файла block_CRC_16_mux.vhd:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity blok_CRC_16_mux is
Port
(
    reset: in std_logic;
    clk_in: in std_logic;
    i_nach_yst: in std_logic;
    prm_byte: in std_logic_vector(7 downto 0);
    enable_CS: in std_logic;
    CRC_16: out std_logic_vector(15 downto 0)
);
end blok_CRC_16_mux;

```

architecture Behavioral of blok_CRC_16_mux is

```

component Tabl_CRC_16_mux
Port
(

```

```

        adr_tabl: in std_logic_vector(7 downto 0);
        Tabl_CRC_16_out: out std_logic_vector(15 downto 0)
    );
end component;

signal Tabl_CRC_16_out: std_logic_vector (15 downto 0);
signal reg_CS: std_logic_vector (15 downto 0);
signal adr_tabl,buf_prm_byte: std_logic_vector (7 downto 0);
signal i_sdwig_reg_CRC,i_rd_tabl_CRC: std_logic;

begin
pr_000: process (reset,clk_in,i_nach_yst, prm_byte, enable_CS)
begin
if reset = '1' then
    buf_prm_byte <= x"00";
elsif clk_in'event and clk_in = '1'    then
    if i_nach_yst = '1' then
        buf_prm_byte <= (others =>'0');
    elsif enable_CS = '1'    then buf_prm_byte <= prm_byte;
    end if;
end if;
end process pr_000;

pr_002: process (reset, clk_in, enable_CS)
begin
if reset ='1' then
    i_sdwig_reg_CRC <= '0';
    i_rd_tabl_CRC <= '0';
elsif clk_in'event and clk_in = '1'    then
    i_sdwig_reg_CRC <= enable_CS;

```

```

        i_rd_tabl_CRC <= i_sdwig_reg_CRC;
    end if;
end process pr_002;

pr_003: process (reset,clk_in,i_sdwig_reg_CRC,reg_CS(7 downto 0),
buf_prm_byte)
begin
    if reset ='1' then
        adr_tabl <= x"00";
    elsif clk_in'event and clk_in = '1' then
        if i_sdwig_reg_CRC= '1' then
            adr_tabl <= reg_CS(7 downto 0) xor buf_prm_byte;
        end if;
    end if;
end process pr_003;

Tabl_CRC: Tabl_CRC_16_mux
Port map
(
    adr_tabl => adr_tabl,
    Tabl_CRC_16_out => Tabl_CRC_16_out
);
pr_001: process (reset,clk_in,i_nach_yst,i_rd_tabl_CRC,Tabl_CRC_16_out)
begin
    if reset ='1' then
        reg_CS <= x"0000";
    elsif clk_in'event and clk_in = '1' then
        if i_nach_yst = '1' then
            reg_CS <= (others =>'0');
        elsif i_rd_tabl_CRC = '1' then

```

```

reg_CS <= Tabl_CRC_16_out xor (x"00" & reg_CS (15
downto 8));

end if;

end if;

end process pr_001;

CRC_16 <= reg_CS;

end Behavioral;

```

Приложение В

(обязательное)

Код узла генерации частот на языке vhd1

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

```

```

library unisim;
use unisim.vcomponents.all;

```

```

entity ClkGen is
port
(
    CLK_IN1: in std_logic;
    CLK_OUT1: out std_logic;
    CLK_OUT2: out std_logic;
    CLK_OUT3: out std_logic;
    CLK_OUT4: out std_logic;
    RESET: in std_logic
);
end ClkGen;

```

```

architecture xilinx of ClkGen is
    attribute CORE_GENERATION_INFO: string;
    attribute CORE_GENERATION_INFO of xilinx: architecture is
"ClkGen,clk_wiz_v3_6,{component_name=ClkGen,use_phase_alignment=true,use_min_o_jitter=false,use_max_i_jitter=false,use_dyn_phase_shift=false,use_inclk_switchover=false,use_dyn_reconfig=false,feedback_source=FDBK_AUTO,primit_type_sel=PLL_BASE,num_out_clk=4,clk_in1_period=7.777,clk_in2_period=7.777,use_power_down=false,use_reset=true,use_locked=false,use_inclk_stopped=false,u

```

```
se_status=false,use_freeze=false,use_clk_valid=false,feedback_type=SINGLE,clock_mgr_type=AUTO>manual_override=false}";
```

```
signal clkkin1: std_logic;  
signal clkfbout: std_logic;  
signal clkfbout_buf: std_logic;  
signal clkout0: std_logic;  
signal clkout1: std_logic;  
signal clkout2: std_logic;  
signal clkout3: std_logic;  
signal clkout4_unused: std_logic;  
signal clkout5_unused: std_logic;  
signal locked_unused: std_logic;
```

```
begin
```

```
pll_base_inst: PLL_BASE
```

```
generic map
```

```
(  
    BANDWIDTH => "OPTIMIZED",  
    CLK_FEEDBACK => "CLKFBOUT",  
    COMPENSATION=> "SYSTEM_SYNCHRONOUS",  
    DIVCLK_DIVIDE=> 1,  
    CLKFBOUT_MULT=> 4,  
    CLKFBOUT_PHASE=> 0.000,  
    CLKOUT0_DIVIDE=> 4,  
    CLKOUT0_PHASE=> 0.000,  
    CLKOUT0_DUTY_CYCLE=> 0.500,  
    CLKOUT1_DIVIDE=> 4,  
    CLKOUT1_PHASE=> 90.000,  
    CLKOUT1_DUTY_CYCLE=> 0.500,
```

```

        CLKOUT2_DIVIDE    => 4,
        CLKOUT2_PHASE     => 180.000,
        CLKOUT2_DUTY_CYCLE => 0.500,
        CLKOUT3_DIVIDE    => 4,
        CLKOUT3_PHASE     => 270.000,
        CLKOUT3_DUTY_CYCLE => 0.500,
        CLKIN_PERIOD      => 7.777,
        REF_JITTER        => 0.010
    )
    port map
    (
        CLKFBOUT=> clkfbout,
        CLKOUT0=> clkout0,
        CLKOUT1=> clkout1,
        CLKOUT2=> clkout2,
        CLKOUT3=> clkout3,
        CLKOUT4=> clkout4_unused,
        CLKOUT5=> clkout5_unused,
        LOCKED=> locked_unused,
        RST=> RESET,
        CLKFBIN=> clkfbout_buf,
        CLKIN=> clk_in1
    );
    clkf_buf: BUFG
    port map
    (
        O => clkfbout_buf,
        I => clkfbout
    );
    clkout1_buf: BUFG

```

```

port map
(
    O=> CLK_OUT1,
    I=> clkout0
);
clkout2_buf: BUFG
port map
(
    O=> CLK_OUT2,
    I=> clkout1
);
clkout3_buf: BUFG
port map
(
    O=> CLK_OUT3,
    I => clkout2
);
clkout4_buf: BUFG
port map
(
    O=> CLK_OUT4,
    I=> clkout3
);
end xilinx;

```

Приложение Г
(обязательное)

Код узла передачи данных на языке vhd1

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library unisim;
use unisim.vcomponents.all;

entity Transiv is
port
(
    Clk128: in std_logic;
    byte_data: in std_logic_vector(7 downto 0);
    rst: in std_logic;
    transiver_out: out std_logic;
    byte_ready: in std_logic
);
end Transiv;

architecture Behavioral of Transiv is
    signal DataStrobEN: std_logic;
    signal msg: std_logic_vector(10 downto 0);
    signal tmp: std_logic;
begin

    tmp<= msg(0);

    trans: process (Clk128, rst)
    begin
        if rst ='1' then
            msg <=(others=>'1');
        elsif Clk128'event and Clk128 ='1' then

```

```
        if byte_ready ='1' then
            msg <="11" & byte_data & '0';
        else
            msg<='1' & msg (10 downto 1);
        end if;
    end if;
end process trans;

transiver_out <= tmp;

end Behavioral;
```

Приложение Д
(обязательное)

Код узла переложения бита и генератора частоты 128 МГц на языке vhd1

Содержимое файла Pereloy.vhd:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Pereloy is
port
(
 Clk: in std_logic;
 Clk2: in std_logic;
 rst: in std_logic;
 bit_in: in std_logic;
 bit_out: out std_logic
);
end Pereloy;

architecture Behavioral of Pereloy is
 signal tmp: std_logic;
 signal tmp1: std_logic;
 signal tmp2: std_logic;
 signal res: std_logic;
begin
 process (Clk1, rst)
 begin
 if rst = '1' then
 tmp<='0';
 elsif Clk1'event and Clk1 = '1' then
 if bit_in = '1' then
 tmp<= '1';
 elsif res ='1' then
 tmp<='0';
 end if;
 end if;
 end process;
end Behavioral;

```

        end if;
    end if;
end process;
process (Clk2, rst)
begin
    if rst = '1' then
        tmp1<='0';
    elsif Clk2'event and Clk2 = '1' then
        tmp1<= tmp;
        tmp2<=tmp1;
    end if;
end process;

res<=tmp1 and not tmp2;
bit_out<=res;
end Behavioral;

```

Содержимое файла Clk50128.vhd:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
library unisim;
use unisim.vcomponents.all;

entity clk50128 is
port
(
    CLK_IN1: in std_logic;

```

```

        CLK_OUT1: out std_logic;
        CLK_OUT2: out std_logic;
        RESET: in std_logic

```

```

    );

```

```

end clk50128;

```

architecture xilinx of clk50128 is

```

    attribute CORE_GENERATION_INFO: string;

```

```

    attribute CORE_GENERATION_INFO of xilinx : architecture is

```

```

"clk50128,clk_wiz_v3_6,{component_name=clk50128,use_phase_alignment=true,
use_min_o_jitter=false,use_max_i_jitter=false,use_dyn_phase_shift=false,use_incl
k_switchover=false,use_dyn_reconfig=false,feedback_source=FDBK_AUTO,prim
type_sel=PLL_BASE,num_out_clk=2,clk1n1_period=20.000,clk1n2_period=20.00
0,use_power_down=false,use_reset=true,use_locked=false,use_inclk_stopped=fals
e,use_status=false,use_freeze=false,use_clk_valid=false,feedback_type=SINGLE,c
lock_mgr_type=AUTO>manual_override=false}";

```

```

    signal clk1n1: std_logic;

```

```

    signal clkfbout: std_logic;

```

```

    signal clkfbout_buf: std_logic;

```

```

    signal clkout0: std_logic;

```

```

    signal clkout1: std_logic;

```

```

    signal clkout2_unused: std_logic;

```

```

    signal clkout3_unused: std_logic;

```

```

    signal clkout4_unused: std_logic;

```

```

    signal clkout5_unused: std_logic;

```

```

    signal locked_unused: std_logic;

```

```

begin

```

```

    clk1n1_buf: IBUFG

```

```

    port map

```

```

(
    O => clk_in1,
    I => CLK_IN1
);

pll_base_inst: PLL_BASE
generic map
(
    BANDWIDTH=> "OPTIMIZED",
    CLK_FEEDBACK=> "CLKFBOUT",
    COMPENSATION=> "SYSTEM_SYNCHRONOUS",
    DIVCLK_DIVIDE=> 1,
    CLKFBOUT_MULT=> 18,
    CLKFBOUT_PHASE=> 0.000,
    CLKOUT0_DIVIDE=> 18,
    CLKOUT0_PHASE=> 0.000,
    CLKOUT0_DUTY_CYCLE=> 0.500,
    CLKOUT1_DIVIDE=> 7,
    CLKOUT1_PHASE=> 0.000,
    CLKOUT1_DUTY_CYCLE=> 0.500,
    CLKIN_PERIOD=> 20.000,
    REF_JITTER=> 0.010
)
port map
(
    CLKFBOUT=> clkfbout,
    CLKOUT0=> clkout0,
    CLKOUT1=> clkout1,
    CLKOUT2=> clkout2_unused,
    CLKOUT3=> clkout3_unused,
    CLKOUT4=> clkout4_unused,

```

```

        CLKOUT5=> clkout5_unused,
        LOCKED=> locked_unused,
        RST=> RESET,
        CLKFBIN=> clkfbout_buf,
        CLKIN=> clkkin1
    );
    clkf_buf: BUFG
    port map
    (
        O => clkfbout_buf,
        I => clkfbout
    );
    clkout1_buf: BUFG
    port map
    (
        O=> CLK_OUT1,
        I=> clkout0
    );
    clkout2_buf: BUFG
    port map
    (
        O=> CLK_OUT2,
        I=> clkout1
    );
end xilinx;

```

Приложение Е

(обязательное)

Код узла обработки принятых данных на языке vhd1

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
library unisim;
use unisim.vcomponents.all;

entity Obrabotka is
port
(
    Clk50: in std_logic;
    Clk128: in std_logic;
    Clk128_90: in std_logic;
    Clk128_180: in std_logic;
    Clk128_270: in std_logic;
    data: in std_logic;
    recive_out: out std_logic_vector (7 downto 0);
    DataStrobEn: out std_logic;
    Frame_Error: out std_logic;
    rst: in std_logic;
);
end Obrabotka;

```

architecture Behavioral of Obrabotka is

component blok_CRC_16_mux is

```

Port
(
    reset: in std_logic;
    clk_in: in std_logic;
    i_nach_yst: in std_logic;

```

```

        prm_byte: in std_logic_vector(7 downto 0);
        enable_CS: in std_logic;
        CRC_16: out std_logic_vector (15 downto 0
    );
end component;

component reciver_node is
port
(
    Clk128: in std_logic;
    Clk128_90: in std_logic;
    Clk128_180: in std_logic;
    Clk128_270: in std_logic;
    data: in std_logic;
    rst: in std_logic;
    recive_out: out std_logic_vector (7 downto 0);
    DataStrobEn: out std_logic;
    Frame_Error: out std_logic
);
end component;

component Perelej is
port
(
    Clk1: in std_logic;
    Clk2: in std_logic;
    rst: in std_logic;
    bit_in: in std_logic;
    bit_out: out std_logic
);

```

end component;

```
signal DataStrobEN_t, bit_out: std_logic;
signal temp_flag: std_logic;
signal temp_flag1: std_logic;
signal res_flag: std_logic;
signal recive_out_t: std_logic_vector (7 downto 0);
signal Res_vect: std_logic_vector (15 downto 0);
constant recive_pref: std_logic_vector (7 downto 0):= x"1b";
constant recive_beg: std_logic_vector (7 downto 0):= x"23";
constant recive_end: std_logic_vector (7 downto 0):= x"24";
constant recive_suc: std_logic_vector (7 downto 0):= x"53";
signal prefix: std_logic;
signal prefix_t: std_logic;
signal begining: std_logic;
signal begining_t: std_logic;
signal ending: std_logic;
signal ending_t: std_logic;
signal success: std_logic;
signal processing_on: std_logic;
signal flag1: std_logic;
signal flag2: std_logic;
signal CRC_16: std_logic_vector (15 downto 0);
signal temp_proc: std_logic;
signal temp_proc1: std_logic;
signal res_proc: std_logic;
signal bit_out_1: std_logic;
signal CRC_OK: std_logic;
signal CRC_OK_t: std_logic;
signal rec_error: std_logic;
```

```

signal rec_error_t: std_logic;
signal rec_error_t1: std_logic;
signal count: std_logic_vector (9 downto 0);
signal kvit: std_logic_vector(15 downto 0);
begin

prefix <= '1' when recive_out_t = recive_pref else '0';

process (Clk50, rst)
begin
    if rst = '1' then
        prefix_t <='0';
    elsif Clk50'event and Clk50 ='1' then
        if prefix = '1' then
            prefix_t <= '1';
        else
            prefix_t <= '0';
        end if;
    end if;
end process;

process (Clk50, rst)
begin
    if rst = '1' then
        begining_t <='0';
    elsif Clk50'event and Clk50 ='1' then
        begining_t <= begining;
    end if;
end process;

```

```

process (Clk50, rst)
begin
    if rst = '1' then
        processing_on <='0';
    elsif Clk50'event and Clk50 = '1' then
        if prefix_t = '1' and begining = '1' then
            processing_on<='1';
        elsif ending = '1' then
            processing_on<='0';
        end if;
    end if;
end process;

```

```

process (Clk50, rst)
begin
    if rst = '1' then
        temp_proc <='0';
    elsif Clk50'event and Clk50 = '1' then
        temp_proc<=processing_on;
        temp_proc1<=temp_proc;
    end if;
end process;

```

```

res_proc<=temp_proc and not temp_proc1;

```

```

process (Clk50, rst)
begin
    if rst = '1' then
        temp_flag <='0';
    elsif Clk50'event and Clk50 = '1' then

```

```

        temp_flag<=flag2;
        temp_flag1<=temp_flag;
    end if;
end process;

res_flag<=temp_flag and not temp_flag1;

process (Clk50, rst)
begin
    if rst = '1' then
        Res_vect <= (others => '0');
        flag1 <= '0';
        flag2 <= '0';
    elsif Clk50'event and Clk50 = '1' then
        if processing_on = '1' and bit_out = '1' and begining_t='0' and
flag1 = '0' then

            Res_vect (15 downto 8) <=recive_out_t;
            flag1 <= '1';
            flag2 <= '0';
        elsif flag1 = '1' and bit_out = '1' then
            Res_vect (7 downto 0) <=recive_out_t;
            flag2 <= '1';
            flag1<='0';
        elsif ending = '1' then
            flag1 <='0';
            flag2 <='0';
        end if;
    end if;
end process;

```

```

process(Clk50,rst)
begin
    if rst = '1' then
        bit_out_1<='0';
    elsif Clk50'event and Clk50 = '1' then
        if processing_on = '1' and bit_out = '1' and begining_t='0' then
            bit_out_1<=bit_out;
        else
            bit_out_1<='0';
        end if;
    end if;
end process;

```

```

begining <= '1' when recive_out_t = recive_beg and prefix_t = '1' else '0';
ending <= '1' when recive_out_t = recive_end and prefix_t = '1' else '0' ;
success <= '1' when recive_out_t = recive_suc and prefix_t = '1' else '0';
CRC_OK <='1' when CRC_16 = x"0000" else '0';

```

```

Process (Clk50, rst)
begin
    if rst='1' or begining = '1' then
        kvit<=(others=>'0');
    elsif Clk50'event and Clk50='1' then
        if rec_error_t1 = '0' and ending ='1'then
            kvit<=x"1b53";
        elsif rec_error_t1 = '1' then
            kvit<=x"1b4E";
        end if;
    end if;
end process;

```

```

process (Clk50, rst)
begin
    if rst='1' then
        CRC_OK_t <= '0';
    elsif Clk50'event and Clk50='1' then
        if ending= '1' then
            CRC_OK_t<=CRC_OK;
        elsif begining ='1' then
            CRC_OK_t<='0';
        end if;
    end if;
end process;

```

```

counter: process (Clk50, rst)
begin
    if begining = '1' then
        count <= (others => '0');
    elsif Clk50'event and Clk50 = '1' then
        if res_flag = '1' then
            count<=count + 1;
        end if;
    end if;
end process counter;

```

```

reciver_node_I: reciver_node
PORT MAP
(
    Clk128=> Clk128,
    Clk128_90=> Clk128_90,

```

```

        Clk128_180=> Clk128_180,
        Clk128_270=> Clk128_270,
        data=> data,
        rst=> rst,
        recive_out=> recive_out_t,
        DataStrobEn=> DataStrobEn_t,
        Frame_Error=> rec_error
    );

    recive_out <= recive_out_t;
    DataStrobEN<=DataStrobEn_t;

    process (Clk50, rst)
    begin
        if rst ='1' then
            rec_error_t<='0';
        elsif Clk50'event and Clk50= '1' then
            if rec_error ='1' then
                rec_error_t<='1';
            elsif begining = '1' then
                rec_error_t <='0';
            end if;
        end if;
    end process;

    rec_error_t1<=rec_error_t and not CRC_OK_t;

    Frame_Error<=rec_error_t1;

    Pereloy_I: Pereloy

```

```

port map
(
    Clk1 =>Clk128,
    Clk2 =>Clk50,
    rst => rst,
    bit_in=>DataStrobEn_t,
    bit_out=>bit_out
);

blok_CRC_16_mux_I: blok_CRC_16_mux
Port map
(
    reset =>    rst,
    clk_in=>Clk50,
    i_nach_yst=>res_proc,
    prm_byte=>recive_out_t,
    enable_CS=>    bit_out_1,
    CRC_16 =>CRC_16
);

end Behavioral;

```