Проектирование модулей приемника и передатчика (RX, TX) (и устройств на их основе) интерфейса UART на языке описания аппаратуры Verilog HDL под микросхему Altera MAX2.

Universal Asynchronous Receiver- Transmitter (UART) – последовательный интерфейс передачи данных, предназначенный для организации связи между цифровыми устройствами. Данные по нему передаются от одного устройства другому по одной физической линии.

Передача данных в UART осуществляется по одному биту в равные промежутки времени (тайм-слоты). Этот временной промежуток определяется скоростью передачи данных UART, эта скорость измеряется в *бодах* (бит в секунду). Для возможных скоростей существует дискретный стандартный ряд.

Таблица 1. Стандартизованные скорости передачи данных по протоколу UART.

300	600	1200	2400	4800	9600	19200	38400	57600	115200	230400	И т.д.

Длительность бита T и скорость передачи данных S связаны выражением: T=1/S.

Помимо информационных бит, в элементарной транзакции существуют обязательным образом стартовый и стоповый биты. Принимающее устройство знает об этом, поэтому из информационного потока оно их вырезает. Зачастую элементарная транзакция передается объемом 1 байт, но встречаются и реализации, где передается 5, 6, 7 или 9 информационных бит. Минимальным пакетом являются обособленные стартовым и стоповым битом информационные биты. Некоторые реализации UART используют несколько стоп- битов для повышения степени синхронизации приемника и передатчика. Для приемника второй и последующие стоп- биты представляются задержкой на линии.

Пассивным состоянием линий является логическая 1. Стартовый бит- логический 0. Приемник ждет перепада из 1 и 0 и отсчитывает от него временной промежуток в половину длительности бита (до середины стартового бита), если в этот момент все еще 0, то запускается процесс приема посылки с заранее оговоренным объемом. Приемник отсчитывает n+1 битовых длительностей, каждый раз фиксируя значение на шине. Первые n значений являются принятыми данными, а последнее значение-проверочное (стоп-бит). Значение стоп- бита всегда равно 1, если приемник детектирует иное, то UART модуль фиксирует ошибку. Биты синхронизации (старт и стоп биты) занимают определенную часть битового потока, то результирующая скорость пропускная способность UART меньше скорости соединения. Например, для 8 битной посылки без бита четности выходит, что используется 10 тайм-слотов, что соответствует 80% полезного сигнала в занятом пространстве битового потока.

При скорости линии в 9600 бод полезная скорость равняется 7860 бод. Временные диаграммы, иллюстрирующие передачу данных по данному протоколу, представлены ниже.

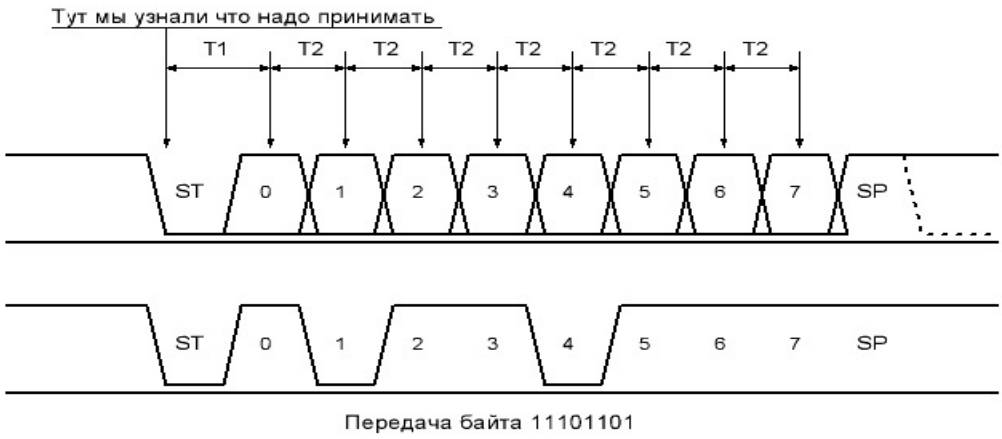

Рис. 1. Пример передачи конкретного байта через одну шину по протоколу UART.

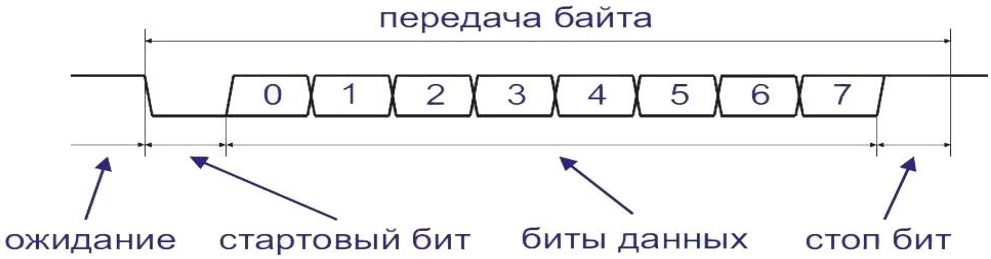

Рис. 2. Пример передачи байта в общем случае с пояснениями через одну шину по протоколу UART.

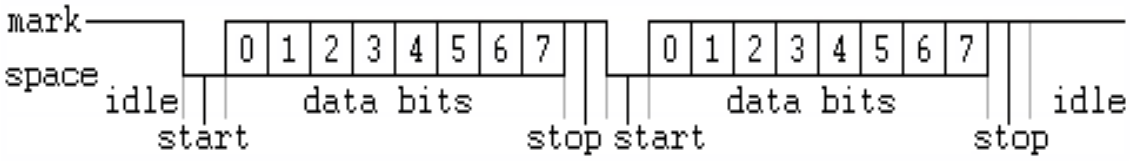

Рис. 3. Передача последовательно двух байт через одну шину по протоколу UART.

Легко видеть, что посылка отправляется младшим битом вперед.

Многие реализации UART имеют возможность контроля целостности принятых данных методом контроля битовой четности. Различают контроль на четность, в котором число единичных бит в посылке-четное число, и контроль на нечетность, при котором сумма единичных бит является нечетным числом.

Таблица 2. Примеры формирования битов четности/нечетности.

Данные	Количество единичных бит	Бит четности	Бит нечетности
00000000	0	0	1
10010100	3	1	0
11110000	4	0	1
11001011	5	1	0

При такой договоренности между приемником и передатчиком появляется автоматическая возможность контроля целостности принятого пакета данных.

Параметры приема-передачи через UART записывают коротком виде так, чтобы эта информация одновременно включала число бит в посылке, наличие и тип бита четности, длительность стоп-бита и скорость передачи UART. Примеры записи проиллюстрированы ниже.

Таблица 3. Примеры короткой записи параметров сообщения по UART.

Короткая запись	Расшифровка			
	Число бит данных в посылке, ед.	Наличие и тип бита четности	Длительность стоп- бита, тайм-слот(ов)	Скорость передачи по UART, бод
9600/8-N-1	8	N (No parity)- без бита четности	1	9600
19200/8-E-1	8	E (Even parity)- бит проверки на четность	1	19200
600/8-O-2	8	O (Odd parity)- бит проверки на нечетность	2	600
2400/5-N-3	5	N	3	2400
19200/10-E-5	10	E	5	19200

Модуль передатчика TX на языке Verilog.

Таблица 4. Описание параметров модуля TX.

Параметр	**Описание параметра**

UART_BAUD_RATE	Физическая скорость передачи по UART.
CLOCK_FREQUENCY	Частота входящего тактового сигнала по проводу IN_CLOCK. Минимальное отношение CLOCK_FREQUENCY/UART_BAUD_RATE равно 2.
PARITY	Параметр бита четности. 0-без бита четности. 1- проверка на четность, 2-проверка на нечетность.
CLKS_PER_BIT_LOG_2	Логарифм отношения CLOCK_FREQUENCY*NUMBER_OF_STOP_BITS/UART_BAUD_RATE, округленный до целого.
NUM_OF_DATA_BITS_IN_PACK	Число информационных бит в пакете данных.
NUM_OF_DATA_BITS_IN_PACK_LOG_2	Логарифм по основанию 2 от числа NUM_OF_DATA_BITS_IN_PACK, округленный до целого.
NUMBER_OF_STOP_BITS	Число стоповых бит в элементарной транзакции.

Таблица 5. Описание входных портов модуля TX.

Входные порты	**Описание**
IN_CLOCK	Входной тактовый сигнал с частотой CLOCK_FREQUENCY.
IN_TX_LAUNCH	Сигнал запуска процесса передачи данных. Если требуется провести один акт передачи, то его длина должна быть больше периода IN_CLOCK, но меньше периода всей посылки.
IN_TX_DATA [NUM_OF_DATA_BITS_IN_PACK-1:0]	Данные, которые будут отправлены после детектирования на шине IN_TX_LAUNCH высокого уровня

	сигнала. Имеет размерность NUM_OF_DATA_BITS_IN_PACK.

Таблица 6. Описание выходных модуля TX.

Выходные порты	**Описание**
OUT_TX_ACTIVE	Тип reg. Данный порт содержит информацию об активности шины .
OUT_TX_SERIAL	Тип reg. Порт TX, который соединяется с RX портом приемного устройства.
OUT_TX_DONE	Тип reg. Порт, на котором кратковременно появляется высокий уровень сигнала после передачи пакета данных (после завершения элементарной транзакции).

Таблица 7. Описание используемых регистров модуля TX.

Используемые регистры	**Размерность регистров**	**Описание регистров**
REG_STATE	[2:0]	Регистр, содержащий информацию о текущем состоянии конечного автомата.
REG_CLOCK_COUNT	[CLKS_PER_BIT_LOG_2:0]	Основной счетчик, который формирует тайм-слоты на основании параметров и приходящих импульсов на входной порт IN_CLOCK.
REG_BIT_INDEX	[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]	Счетчик номера бита во время последовательной передачи данных.
REG_TX_DATA	[NUM_OF_DATA_BITS_IN_PACK-1:0]	Регистр, в котором хранятся данные для последовательной передачи, защелкивающийся при

		детектировании на линии IN_TX_LAUNCH высокого уровня сигнала.
FLAG_DONE_TRANSACTION	1	Регистр, кратковременно хранящий информацию о том, что транзакция окончена.

Таблица 8. Описание состояний конечного автомата модуля TX.

Состояние конечного автомата	**Описание состояния**
STATE_WAIT	Состояние ожидания, в котором передатчик ожидает команды передачи данных. Этому состоянию характерно: удержание на шине TX логической единицы, OUT_TX_ACTIVE равен нулю. Если это первый такт пребывания в состоянии STATE_WAIT после выхода из предыдущего, то: OUT_TX_DONE = 1, OUT_TX_ACTIVE=0, FLAG_DONE TRANSACTION=0. Если это не первый такт, то OUT_TX_DONE сбрасывается в нуль. Таким образом, OUT_TX_DONE после окончания транзакции появляется на один такт IN_CLOCK. При обнаружении на IN_TX_LAUNCH высокого уровня сигнала происходит следующее: 1)OUT_TX_ACTIVE<=1 2)OUT_TX_SERIAL<=0 3)REG_TX_DATA<=IN_TX_DATA 4)REG_STATE<=STATE_TX_START_BIT 1. На сигнальный порт активности шины ставится сигнал 1, что говорит о ее занятости. 2. Шина TX притягивается к земле, что означает начало передачи старт-бита. Преждевременная подтяжка этой шины к нулю оправдана необходимостью уменьшения инерции отклика.

	3. Происходит защелкивание данных с входной шины IN_TX_DATA во внутренний регистр. 4. Переход конечного автомата в следующее состояние.
STATE_TX_START_BIT	В этом состоянии модуль притягивает шину TX к нулю на время длительности одного тайм-слота. Таким образом происходит передача старт-бита. В это состояние автомат входит после детектирования высокого уровня сигнала на IN_LAUNCH еще в состоянии STATE_WAIT. После отсчета необходимого числа тактов основного генератора происходит переход в состояние STATE_TX_DATA_BITS
STATE_TX_DATA_BITS	В этом состоянии происходит передача информационных бит в порядке “младшим битом вперед”, а также происходит итерация двух счетчиков: основного счетчика тайм-слота и счетчика битов. Внешним счетчиком является счетчик битов, внутренним — счетчик тайм-слота. На каждый сброс внутреннего счетчика увеличивается на единицу внешний. Параллельно с изменением состояний счетчиков на линии TX удерживается текущий бит передаваемых данных. Когда счетчик битов досчитывает до конечного элемента в пакете данных, он сбрасывается и изменяет состояние автомата в зависимости от параметра PARITY. Если транзакция не предусматривает наличие бита четности, то происходит переход в состояние STATE_TX_STOP_BIT, если параметры задают бит четности, то автомат переходит в STATE_PARITY_BIT.
STATE_PARITY_BIT	В данном состоянии на шине TX удерживается бит четности/нечетности на основании анализа числа единичных битов в посылке через специальную функцию в течении одного тайм-слота. После этого

	происходит переход в STATE_TX_STOP_BIT.
STATE_TX_STOP_BIT	В этом состоянии на шине TX удерживается высокий уровень сигнала. Время удержания равно NUMBER_STOP_BITS * (время одного тайм-слота). По истечении времени удержания происходит следующее: 1)FLAG_DONE_TRANSACTION<=1 2)REG_CLOCK_COUNT<=0 3)REG_STATE<=STATE_WAIT 1. В индикатор окончания процесса транзакции записывается логическая единица. 2. Сбрасывается счетчик тайм-слота. 3. Переход автомата в состояние STATE_WAIT.

Программный код модуля TX

```
module UART_FPGA_TX
 #(
        parameter       UART_BAUD_RATE                  =9600;
        parameter       CLOCK_FREQUENCY                 =19200;
        parameter       PARITY                          =2;
        parameter       CLKS_PER_BIT_LOG_2              =5;
        parameter       NUM_OF_DATA_BITS_IN_PACK        =8;
        parameter       NUM_OF_DATA_BITS_IN_PACK_LOG_2  =3;
        parameter       NUMBER_STOP_BITS                =4
 )
 (
  input     IN_CLOCK,
  input     IN_TX_LAUNCH,
  input [NUM_OF_DATA_BITS_IN_PACK-1:0] IN_TX_DATA,
  output reg  OUT_TX_ACTIVE,
  output reg  OUT_TX_SERIAL,
  output reg  OUT_TX_DONE
  );

  parameter CLKS_PER_BIT            = CLOCK_FREQUENCY/UART_BAUD_RATE ;
  parameter STATE_WAIT              = 3'b000;
  parameter STATE_TX_START_BIT      = 3'b001;
  parameter STATE_TX_DATA_BITS      = 3'b010;
  parameter STATE_PARITY_BIT        = 3'b101;
  parameter STATE_TX_STOP_BIT       = 3'b011;
  reg [2:0]                                     REG_STATE;
  reg [CLKS_PER_BIT_LOG_2:0]                    REG_CLOCK_COUNT;
  reg [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]        REG_BIT_INDEX;
  reg [NUM_OF_DATA_BITS_IN_PACK-1:0]            REG_TX_DATA;
  reg                                           FLAG_DONE_TRANSACTION;
```

```
initial begin
                REG_STATE                                       = STATE_WAIT;
                REG_CLOCK_COUNT                                 = 0;
                REG_BIT_INDEX                                   = 0;
                REG_TX_DATA                                     = 0;
                FLAG_DONE_TRANSACTION                           = 0;
                OUT_TX_ACTIVE                                   = 0;
                OUT_TX_SERIAL                                   = 1;
                OUT_TX_DONE                                     = 0;
end
always @(posedge IN_CLOCK)
begin
   case (REG_STATE)
   STATE_WAIT :
   begin
        if(FLAG_DONE_TRANSACTION==1)
        begin
                OUT_TX_DONE                             <= 1'b1;
                OUT_TX_ACTIVE                           <= 1'b0;
                FLAG_DONE_TRANSACTION                   <=0;
        end
        else
                OUT_TX_DONE                             <= 1'b0;

                OUT_TX_SERIAL                           <= 1'b1;
      if (IN_TX_LAUNCH == 1'b1)
      begin
                OUT_TX_ACTIVE                           <= 1'b1;
                REG_TX_DATA                             <= IN_TX_DATA;
                REG_STATE                               <= STATE_TX_START_BIT;
                OUT_TX_SERIAL                           <= 1'b0;
                OUT_TX_DONE                             <= 1'b0;
      end
   end
   STATE_TX_START_BIT :
   begin
        if (REG_CLOCK_COUNT < CLKS_PER_BIT-2)
                REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
        else
        begin
                REG_CLOCK_COUNT   <= 0;
                REG_STATE               <= STATE_TX_DATA_BITS;
        end
    end
    STATE_TX_DATA_BITS :
    begin
      OUT_TX_SERIAL <= REG_TX_DATA[REG_BIT_INDEX];
      if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)
                REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
      else
      begin
          REG_CLOCK_COUNT <= 0;
          if (REG_BIT_INDEX < NUM_OF_DATA_BITS_IN_PACK-1)
                REG_BIT_INDEX <= REG_BIT_INDEX + 1;
          else
          begin
                REG_BIT_INDEX <= 0;
                if(PARITY==0)
                        REG_STATE   <= STATE_TX_STOP_BIT;
                else
                        REG_STATE   <= STATE_PARITY_BIT;
          end
      end
```

```
        end
        STATE_PARITY_BIT:
        begin
            case(PARITY)
                    1: OUT_TX_SERIAL <= (sum_of_bits(REG_TX_DATA)%2==0)? 0:1;
                    2: OUT_TX_SERIAL <= (sum_of_bits(REG_TX_DATA)%2==0)? 1:0;
            endcase
            if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)
                    REG_CLOCK_COUNT    <= REG_CLOCK_COUNT + 1;
            else
            begin
                    REG_CLOCK_COUNT                 <= 0;
                    REG_STATE                       <= STATE_TX_STOP_BIT;
            end
        end
        STATE_TX_STOP_BIT:
        begin
            OUT_TX_SERIAL <= 1'b1;
            if (REG_CLOCK_COUNT < CLKS_PER_BIT*NUMBER_STOP_BITS-1)
                    REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
            else
            begin
                    FLAG_DONE_TRANSACTION           <=1;
                    REG_CLOCK_COUNT                 <= 0;
                    REG_STATE                       <= STATE_WAIT;
            end
        end
        default :
            REG_STATE <= STATE_WAIT;
      endcase
    end
    function sum_of_bits;
                    input [NUM_OF_DATA_BITS_IN_PACK-1:0] value;
                    reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] sum=0;
                    reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] i;
                    for (i=0;i<=NUM_OF_DATA_BITS_IN_PACK-1;i=i+1)
                            sum=sum+value[i];
                    sum_of_bits=sum;
    endfunction
endmodule
```

Полученные результаты при моделировании TX модуля при разных наборах параметров

Набор параметров (1):

parameter PARITY	=0
parameter NUM_OF_DATA_BITS_IN_PACK	=8
parameter NUMBER_STOP_BITS	=1

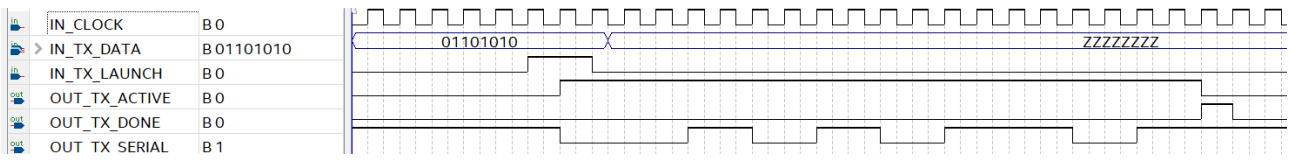

Рис. 4. Отправка байта 01101010 модулем TX при наборе параметров (1).

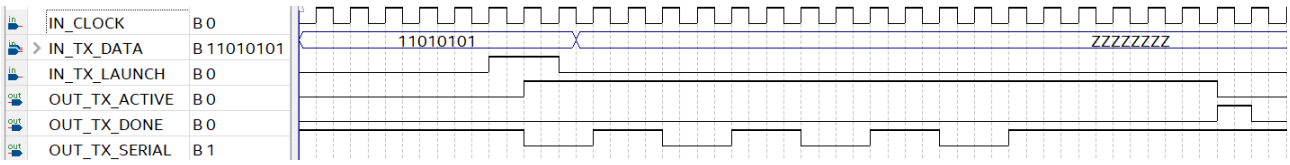

Рис. 5. Отправка байта 11010101 модулем TX при наборе параметров (1).

Набор параметров (2):

parameter PARITY =0
parameter NUM_OF_DATA_BITS_IN_PACK =8
parameter NUMBER_STOP_BITS =2

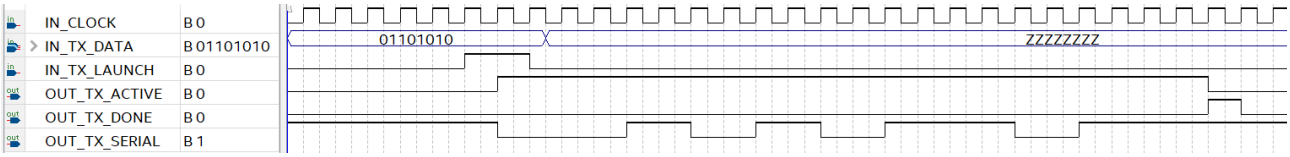

Рис. 6. Отправка байта 01101010 модулем TX при наборе параметров (2).

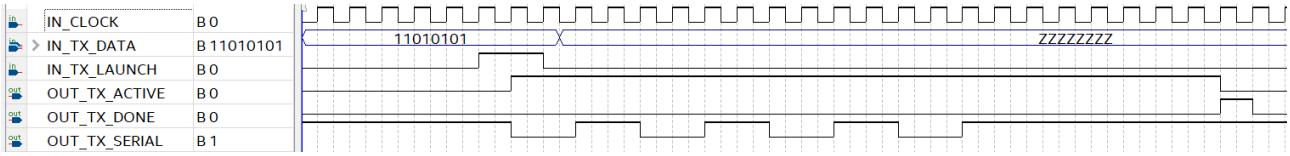

Рис. 7. Отправка байта 11010101 модулем TX при наборе параметров (2).

Набор параметров (3):

parameter PARITY =0
parameter NUM_OF_DATA_BITS_IN_PACK =8
parameter NUMBER_STOP_BITS =4

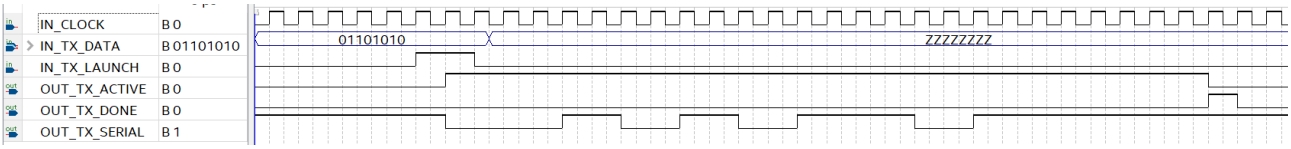

Рис. 8. Отправка байта 01101010 модулем TX при наборе параметров (3).

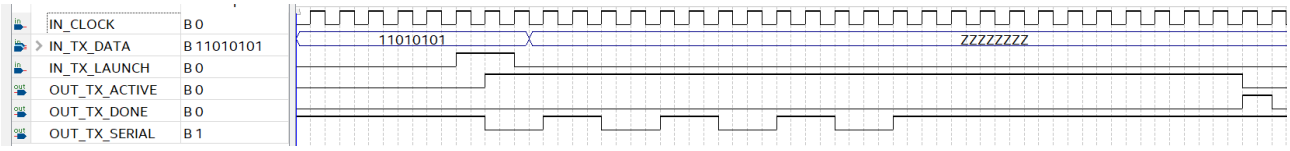

Рис. 9. Отправка байта 11010101 модулем TX при наборе параметров (3).

Набор параметров (4):

parameter PARITY =1
parameter NUM_OF_DATA_BITS_IN_PACK =8
parameter NUMBER_STOP_BITS =1

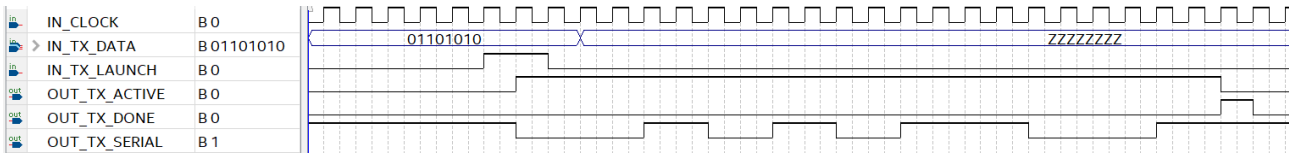

Рис. 10. Отправка байта 01101010 модулем TX при наборе параметров (4).

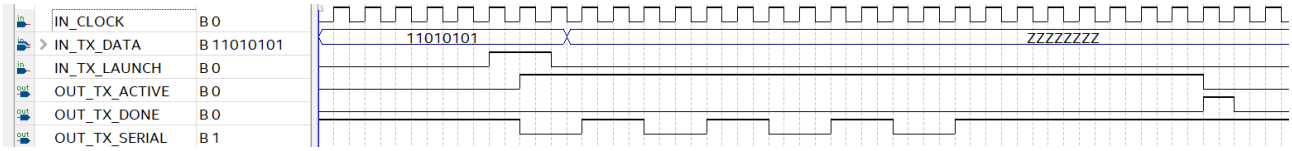

Рис. 11. Отправка байта 11010101 модулем TX при наборе параметров (4).

Набор параметров (5):

parameter PARITY	=1
parameter NUM_OF_DATA_BITS_IN_PACK	=8
parameter NUMBER_STOP_BITS	=2

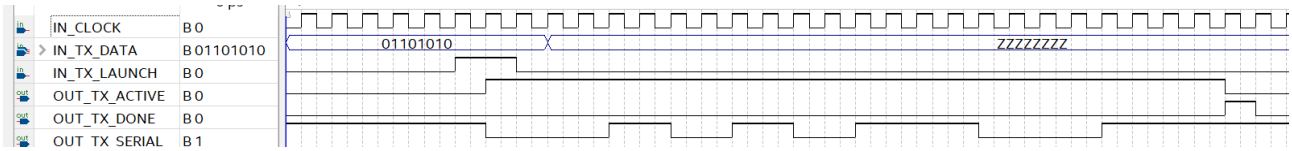

Рис. 12. Отправка байта 01101010 модулем TX при наборе параметров (5).

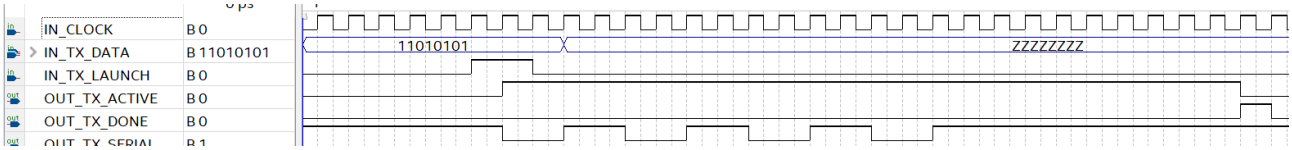

Рис. 13. Отправка байта 11010101 модулем TX при наборе параметров (5).

Набор параметров (6):

parameter PARITY	=1
parameter NUM_OF_DATA_BITS_IN_PACK	=8
parameter NUMBER_STOP_BITS	=4

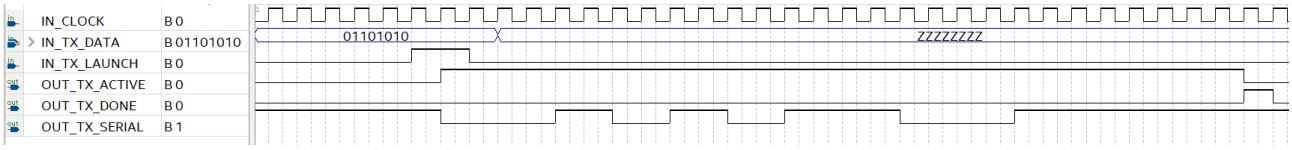

Рис. 14. Отправка байта 01101010 модулем TX при наборе параметров (6).

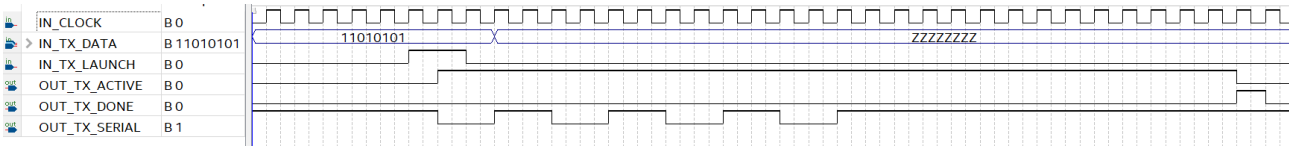

Рис. 15. Отправка байта 11010101 модулем TX при наборе параметров (6).

Набор параметров (7):

parameter PARITY	=2
parameter NUM_OF_DATA_BITS_IN_PACK	=8
parameter NUMBER_STOP_BITS	=1

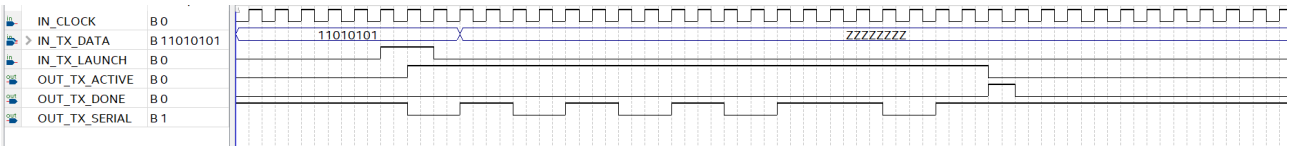

Рис. 16. Отправка байта 11010101 модулем TX при наборе параметров (7).

Набор параметров (8):

parameter PARITY	=2
parameter NUM_OF_DATA_BITS_IN_PACK	=8
parameter NUMBER_STOP_BITS	=1

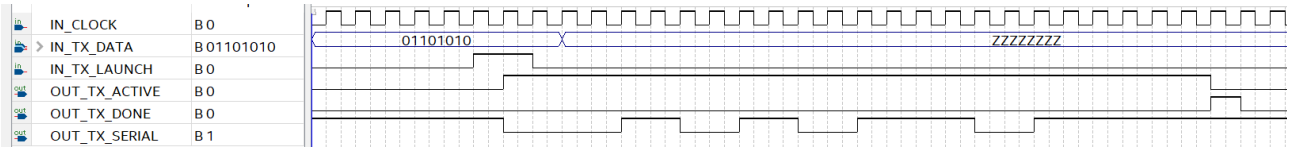

Рис. 17. Отправка байта 01101010 модулем TX при наборе параметров (8).

Набор параметров (9):

parameter PARITY	=2
parameter NUM_OF_DATA_BITS_IN_PACK	=8
parameter NUMBER_STOP_BITS	=4

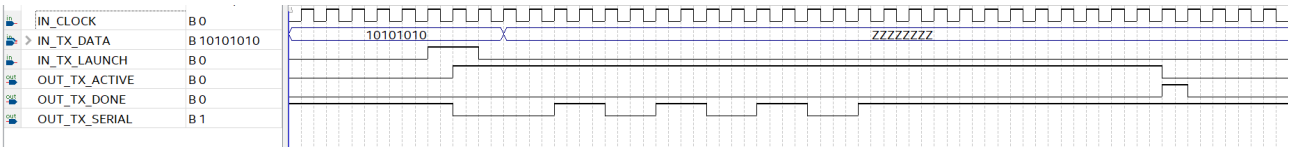

Рис. 18. Отправка байта 10101010 модулем TX при наборе параметров (9).

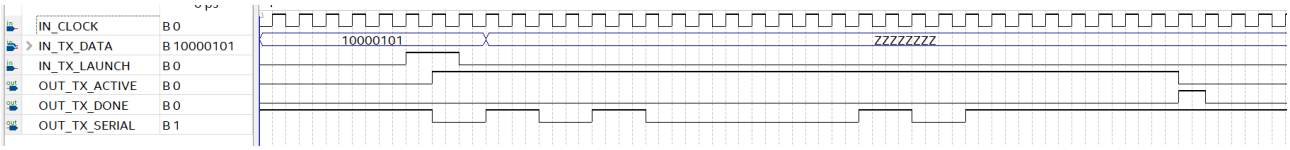

Рис. 19. Отправка байта 10000101 модулем TX при наборе параметров (9).

Набор параметров (10):

parameter PARITY	=0
parameter NUM_OF_DATA_BITS_IN_PACK	=5
parameter NUMBER_STOP_BITS	=1

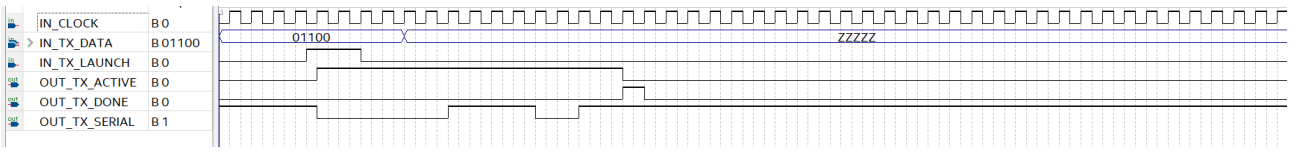

Рис. 20. Отправка пакета 01100 модулем TX при наборе параметров (10).

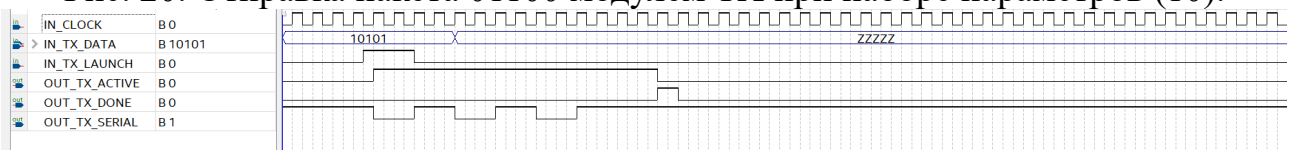

Рис. 21. Отправка пакета 10101 модулем TX при наборе параметров (10).

Набор параметров (11):

parameter PARITY =0
parameter NUM_OF_DATA_BITS_IN_PACK =5
parameter NUMBER_STOP_BITS =2

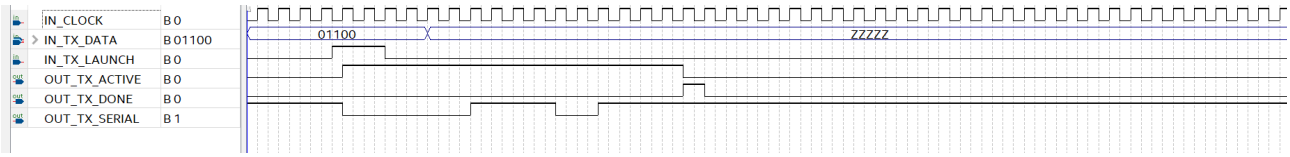

Рис. 22. Отправка пакета 01100 модулем TX при наборе параметров (11).

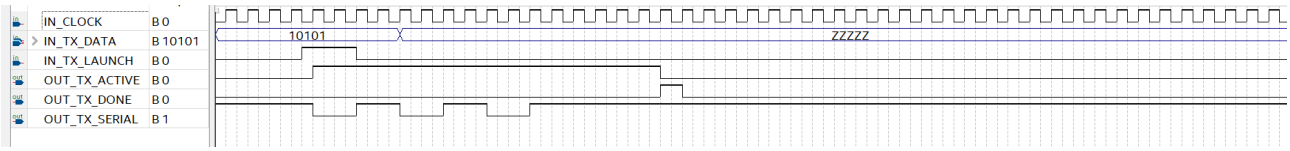

Рис. 23. Отправка пакета 10101 модулем TX при наборе параметров (11).

Набор параметров (12):

parameter PARITY =1
parameter NUM_OF_DATA_BITS_IN_PACK =5
parameter NUMBER_STOP_BITS =1

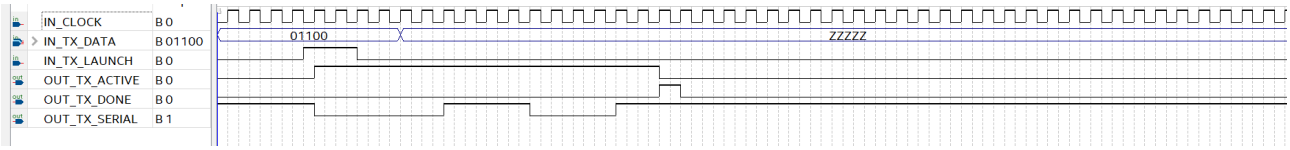

Рис. 24. Отправка пакета 01100 модулем TX при наборе параметров (12).

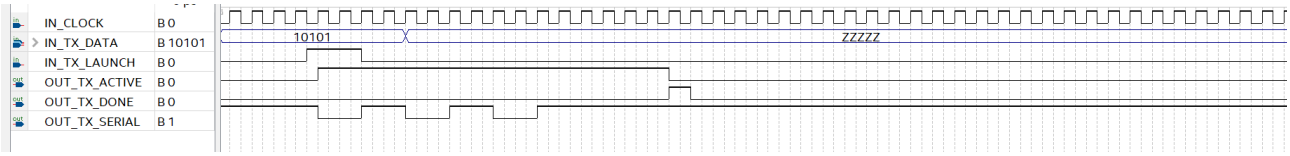

Рис. 25. Отправка пакета 10101 модулем TX при наборе параметров (12).

Набор параметров (13):

parameter PARITY =1
parameter NUM_OF_DATA_BITS_IN_PACK =5
parameter NUMBER_STOP_BITS =2

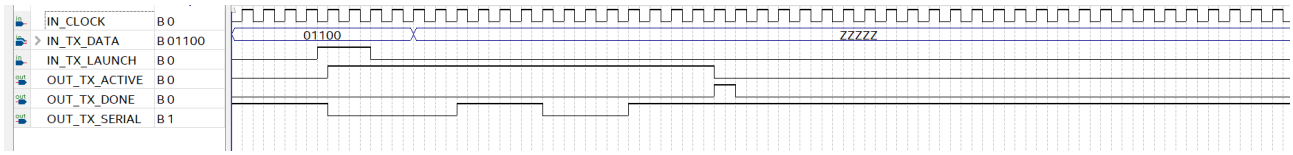

Рис. 26. Отправка пакета 01100 модулем TX при наборе параметров (13).

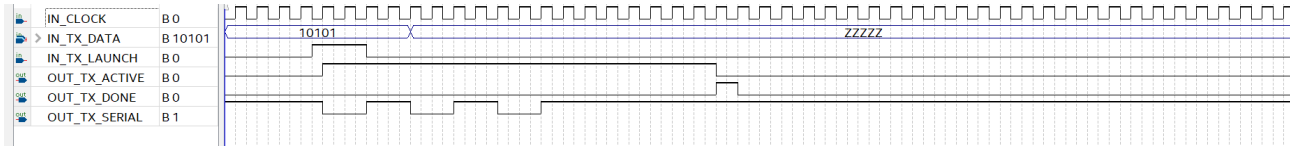

Рис. 27. Отправка пакета 10101 модулем TX при наборе параметров (13).

Набор параметров (14):

parameter PARITY =2
parameter NUM_OF_DATA_BITS_IN_PACK =5
parameter NUMBER_STOP_BITS =1

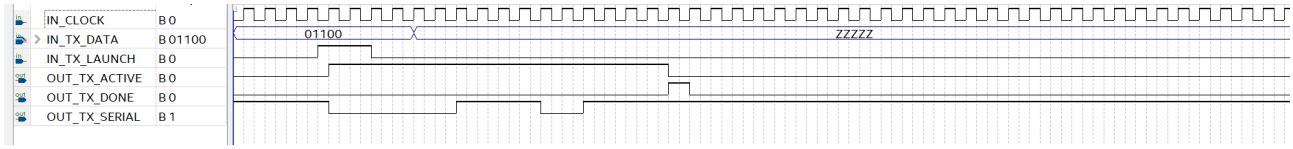

Рис. 28. Отправка пакета 01100 модулем TX при наборе параметров (14).

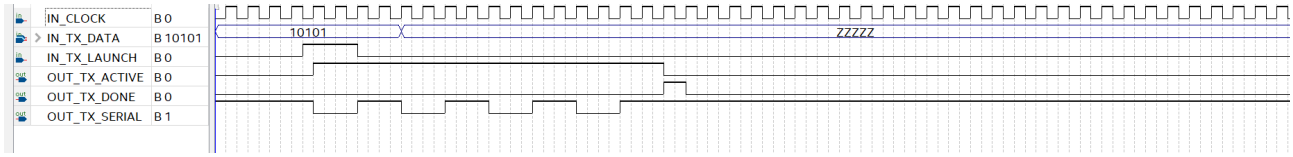

Рис. 29. Отправка пакета 10101 модулем TX при наборе параметров (14).

Набор параметров (15):

parameter PARITY =2
parameter NUM_OF_DATA_BITS_IN_PACK =5
parameter NUMBER_STOP_BITS =4

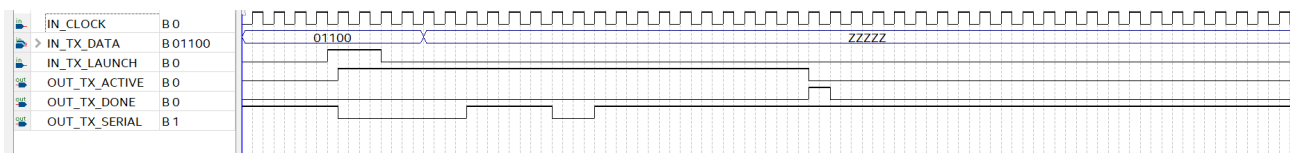

Рис. 30. Отправка пакета 01100 модулем TX при наборе параметров (15).

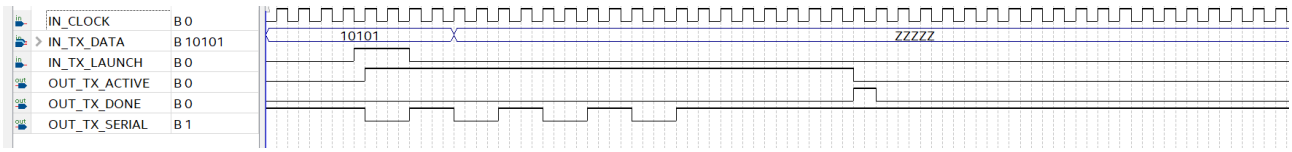

Рис. 31. Отправка пакета 10101 модулем TX при наборе параметров (15).

Набор параметров (16):

parameter PARITY	=0
parameter NUM_OF_DATA_BITS_IN_PACK	=10
parameter NUMBER_STOP_BITS	=1

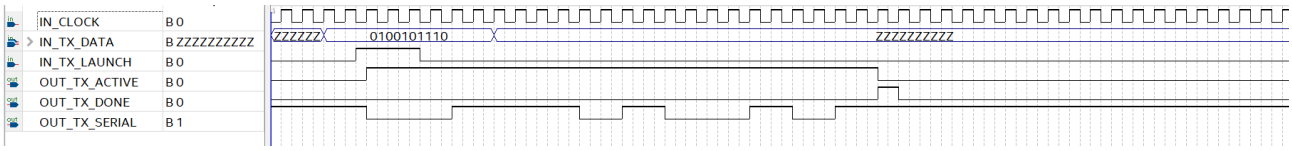

Рис. 32. Отправка пакета 0100101110 модулем TX при наборе параметров (16).

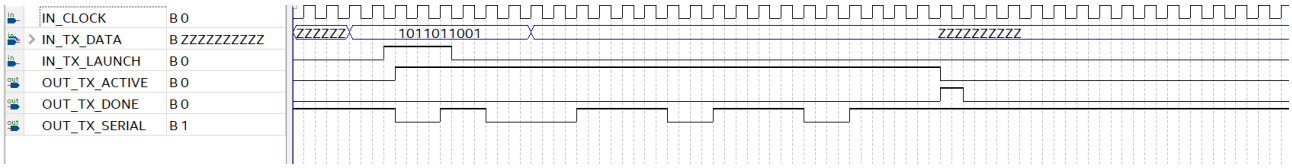

Рис. 33. Отправка пакета 1011011001 модулем TX при наборе параметров (16).

Набор параметров (17):

parameter PARITY	=0
parameter NUM_OF_DATA_BITS_IN_PACK	=10
parameter NUMBER_STOP_BITS	=2

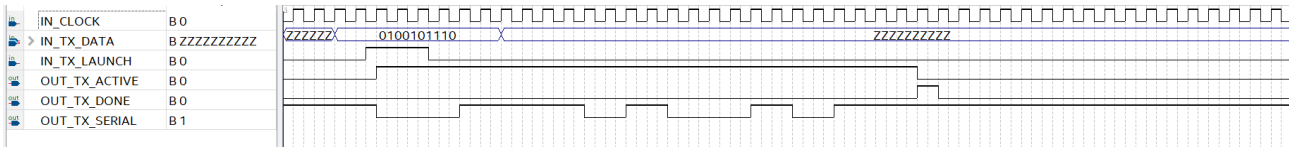

Рис. 34. Отправка пакета 0100101110 модулем TX при наборе параметров (17).

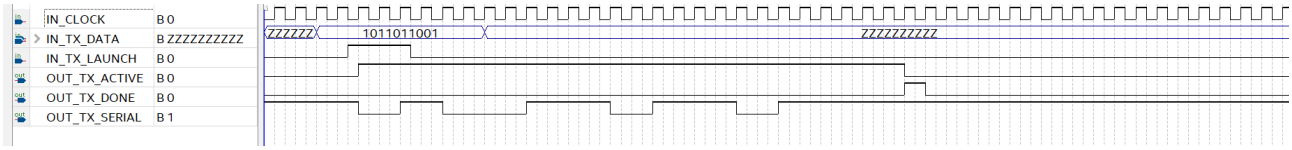

Рис. 35. Отправка пакета 1011011001 модулем TX при наборе параметров (17).

Набор параметров (18):

parameter PARITY	=1
parameter NUM_OF_DATA_BITS_IN_PACK	=10
parameter NUMBER_STOP_BITS	=1

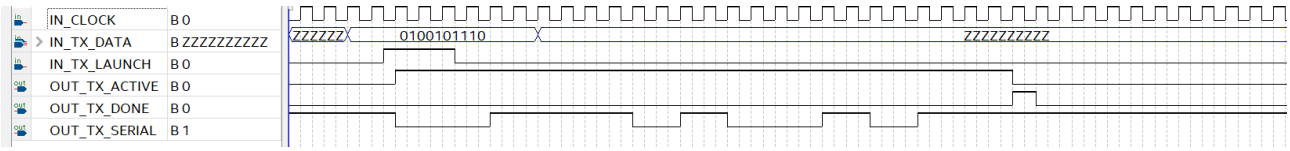

Рис. 36. Отправка пакета 0100101110 модулем TX при наборе параметров (18).

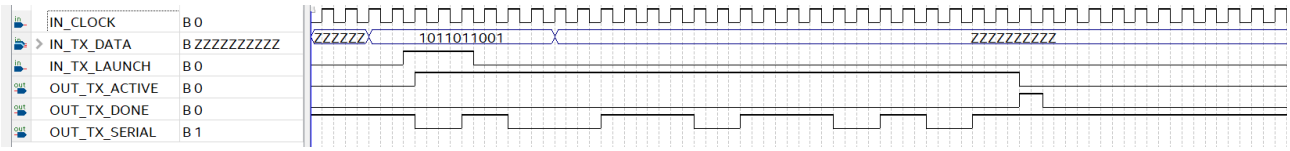

Рис. 37. Отправка пакета 1011011001 модулем TX при наборе параметров (18).

Набор параметров (19):

parameter PARITY	=2
parameter NUM_OF_DATA_BITS_IN_PACK	=10
parameter NUMBER_STOP_BITS	=1

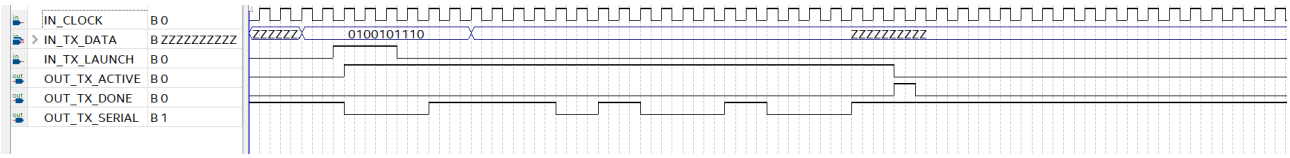

Рис. 38. Отправка пакета 0100101110 модулем TX при наборе параметров (19).

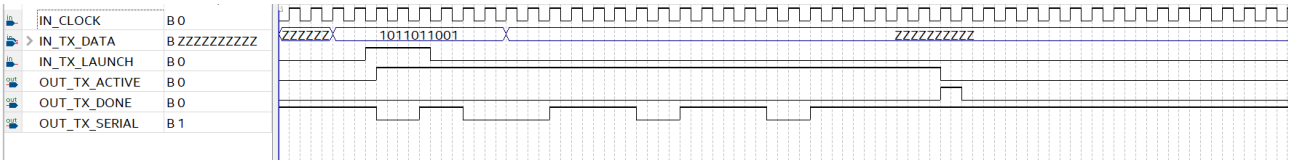

Рис. 39. Отправка пакета 1011011001 модулем TX при наборе параметров (19).

Посылка нескольких пакетов

Набор параметров (20):

parameter PARITY	=1
parameter NUM_OF_DATA_BITS_IN_PACK	=5
parameter NUMBER_STOP_BITS	=1

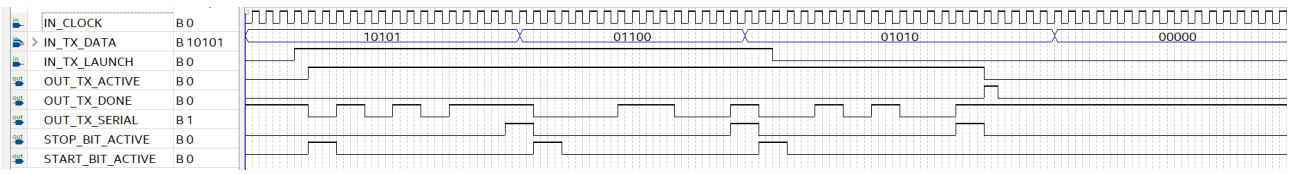

Рис. 40. Отправка пакетов 10101, 01100 и 01010 модулем TX при наборе параметров (20).

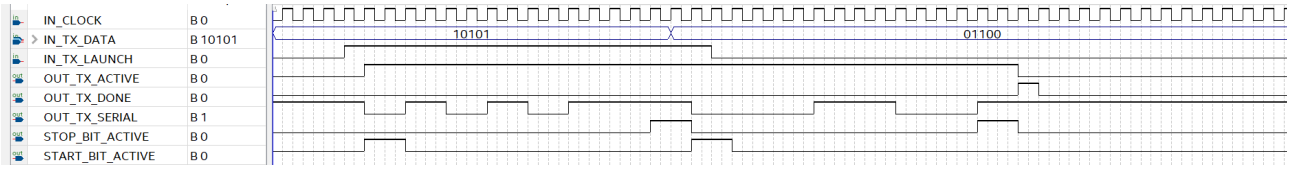

Рис. 41. Отправка пакетов 10101 и 01100 модулем TX при наборе параметров (20).

Набор параметров (21):

parameter PARITY =2
parameter NUM_OF_DATA_BITS_IN_PACK =5
parameter NUMBER_STOP_BITS =1

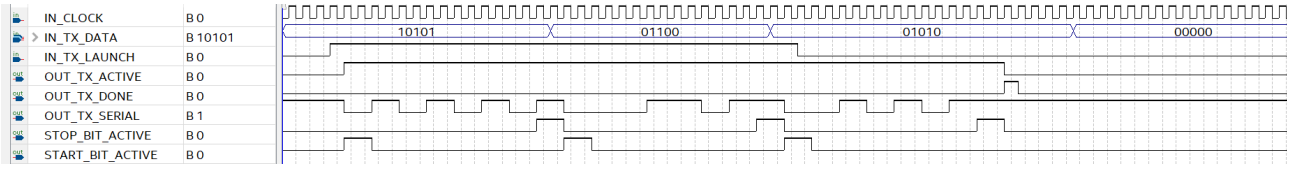

Рис. 42. Отправка пакетов 10101, 01100 и 01010 модулем TX при наборе параметров (21).

Модуль приемника RX на языке Verilog.

Таблица 9. Описание параметров модуля RX.

Параметр	Описание
UART_BAUD_RATE	Битрейт передачи данных по UART.
CLOCK_FREQUENCY	Частота тактового сигнала IN_CLOCK.
PARITY	Характеристика бита четности. 0-отсутствие бита четности, 1-проверка на четность, 2-проверка на нечетность.
CLKS_PER_BIT_LOG_2	Округленный до целого логарифм по основанию 2 отношения CLOCK_FREQUENCY/UART_BAUD_RATE .
NUM_OF_DATA_BITS_IN_PACK	Число информационных бит в одном пакете.
NUM_OF_DATA_BITS_IN_PACK_LOG_2	Округленный до целого логарифм по основанию 2 числа NUM_OF_DATA_BITS_IN_PACK.

Таблица 10. Описание входных портов модуля RX.

Входной порт	Описание

IN_CLOCK	Входной тактовый сигнал.
IN_RX_SERIAL	Входной сигнал RX.

Таблица 11. Описание выходных портов модуля RX.

Выходной порт	**Описание**
OUT_RX_DATA_READY	Кратковременный сигнал, объявляющий готовность принятого пакета данных, его [пакет] можно считывать.
OUT_RX_DATA	Параллельный сигнал- пакет принятых данных, который можно считывать по переднему фронту сигнала OUT_RX_DATA_READY.
OUT_RX_ERROR	На этом порту появляется высокий уровень сигнала на несколько тактов основного генератора IN_CLOCK при обнаружении ошибки передачи через бит четности или в случае некорректно принятого стоп бита.

Таблица 12. Описание регистров модуля RX.

Используемые регистры	**Описание**
REG_CLOCK_COUNT [CLKS_PER_BIT_LOG_2:0]	Счетчик формирования тайм-слотов
REG_BIT_INDEX [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]	Счетчик номера принимаемого бита
REG_STATE	Регистр, хранящий текущее состояние конечного автомата

Таблица 13. Описание состояний конечного автомата модуля RX.

Состояния конечного автомата	**Описание состояния**
STATE_WAIT	Состояние ожидания старт-бита от передатчика. Линия OUT_RX_DATA_READY держится в низком состоянии. Линия OUT_RX_ERROR тоже притянута к нулю. Если детектируется низкий уровень сигнала на IN_RX_SERIAL, то автомат меняет состояние на

	STATE_RX_START_BIT.
STATE_RX_START_BIT	Происходит умышленное ожидание в течении времени половины периода передачи одного бита, а затем на линии IN_RX_SERIAL проверяется состояние вновь. Если там все еще нуль, то счетчик тайм- слота обнуляется, а автомат переходит в другое состояние, а именно, STATE_RX_DATA_BITS.
STATE_RX_DATA_BITS	Два вложенных счетчика (счетчик тайм-слота и счетчик битов) изменяют свое состояние таким образом, что REG_BIT_INDEX увеличивается на 1, когда REG_CLOCK_COUNT делает полный проход, в конце каждого такого прохода считывается состояние с шины IN_RX_SERIAL и записывается в память. Когда REG_BIT_INDEX доходит до NUM_OF_DATA_BITS_IN_PACK, автомат обнуляет счетчики и в зависимости от значения PARITY совершает переход в другое состояние. Если PARITY==0, то автомат переходит в состояние STATE_RX_STOP_BIT, если же PARITY !=0, то следующее состояние — STATE_RX_PARITY_BIT.
STATE_RX_PARITY_BIT	В данном состоянии автомат проверяет посылку на четность и нечетность. Если четность/ нечетность не соответствует выбранному протоколу передачи, то в REG_RX_ERROR записывается 1.
STATE_RX_STOP_BIT	В данном состоянии автомат проверят стоп бит. Если на шине оказывается не высокий уровень сигнала, то детектируется ошибка, что реализуется через запись в регистр REG_RX_ERROR логической единицы. Если же сигнал на шине высокий, то выполняется переход

	автомата в состояние ожидания в связке со стандартными обслуживающими командами.

Описание функции sum_of_bits

Модуль RX содержит функцию подсчета суммы бит в пакете. Его присутствие в модуле упрощает понимание кода, выделяет функциональный блок за рамки поведенческого блока always. Задача функции sum_of_bits состоит в подсчете числа бит в пакете. Это требуется для проверки на четность/нечетность принятого пакета данных. Реализация основана на одном цикле, который делает обход всего принятого регистра как массива битов, попутно суммируя в буффер.

Код функции sum_of_bits

```
function [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] sum_of_bits;
        input [NUM_OF_DATA_BITS_IN_PACK-1:0]        value;
        reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]       sum;
        reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]       i;
        begin
                sum=0;
                for (i=0;i<NUM_OF_DATA_BITS_IN_PACK;i=i+1)
                        sum=sum+value[i];
                sum_of_bits=sum;
        end
endfunction
```

Код модуля приемника RX на языке Verilog

```
module UART_FPGA_RX
  #(
  parameter UART_BAUD_RATE=1,
  parameter CLOCK_FREQUENCY=4,
  parameter PARITY=2,
  parameter CLKS_PER_BIT_LOG_2=5,
  parameter NUM_OF_DATA_BITS_IN_PACK=10,
  parameter NUM_OF_DATA_BITS_IN_PACK_LOG_2=3
  )
  (
   input       IN_CLOCK,
   input       IN_RX_SERIAL,
   output reg  OUT_RX_DATA_READY,
   output reg  [NUM_OF_DATA_BITS_IN_PACK-1:0] OUT_RX_DATA=0,
   output reg  OUT_RX_ERROR=0
   );

  parameter CLKS_PER_BIT = CLOCK_FREQUENCY/UART_BAUD_RATE ;
  parameter STATE_WAIT        = 3'b000;
```

```
parameter STATE_RX_START_BIT = 3'b001;
parameter STATE_RX_DATA_BITS = 3'b010;/
parameter STATE_RX_STOP_BIT  = 3'b011;
parameter STATE_RX_PARITY_BIT= 3'b100;

reg [CLKS_PER_BIT_LOG_2:0]REG_CLOCK_COUNT                                   = 0;
reg [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0]    REG_BIT_INDEX                     = 0;
reg [2:0]    REG_STATE                                                      = 0;

always @(posedge IN_CLOCK)
begin
        case (REG_STATE)]
                STATE_WAIT:
                 begin
                        OUT_RX_DATA_READY                       <= 1'b0;
                        OUT_RX_ERROR                                    <=0;
                        if (IN_RX_SERIAL == 1'b0)
                                REG_STATE <= STATE_RX_START_BIT;
                 end
                 STATE_RX_START_BIT :
                begin
                        if (REG_CLOCK_COUNT == CLKS_PER_BIT/2-2)
                        begin
                                if (IN_RX_SERIAL == 1'b0)
                                begin
                                        REG_CLOCK_COUNT <= 0;
                                        REG_STATE    <= STATE_RX_DATA_BITS;
                                end
                                else
                                        REG_STATE <= STATE_WAIT;
                        end
                        else REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;

                end
                STATE_RX_DATA_BITS:
                begin
                        if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)
                                 REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
                        else
                        begin
                                REG_CLOCK_COUNT             <= 0;
                                OUT_RX_DATA[REG_BIT_INDEX] <= IN_RX_SERIAL;
                                if (REG_BIT_INDEX < NUM_OF_DATA_BITS_IN_PACK-1)
                                REG_BIT_INDEX <= REG_BIT_INDEX + 1;
                                else
                                begin
                                        REG_BIT_INDEX <= 0;
                                        if(PARITY!=0)
                                                REG_STATE   <= STATE_RX_PARITY_BIT;
                                        else
                                                REG_STATE   <= STATE_RX_STOP_BIT;
                                end
                        end
                end
                STATE_RX_PARITY_BIT:
                begin
                        if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)
                                REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
                        else
                        begin
                                REG_CLOCK_COUNT <= 0;
                                REG_STATE   <= STATE_RX_STOP_BIT;
                                case(PARITY)
```

```
                                        1:OUT_RX_ERROR<=((sum_of_bits(OUT_RX_DATA)
+IN_RX_SERIAL)%2==0) ?0:1;
                                        2:OUT_RX_ERROR<=((sum_of_bits(OUT_RX_DATA)
+IN_RX_SERIAL)%2==0)?1:0;
                                endcase
                        end
                end
                STATE_RX_STOP_BIT :
                begin
                        if (REG_CLOCK_COUNT < CLKS_PER_BIT-1)
                                REG_CLOCK_COUNT <= REG_CLOCK_COUNT + 1;
                        else
                        begin
                                if(IN_RX_SERIAL)
                                begin
                                        OUT_RX_DATA_READY      <= 1'b1;
                                        REG_CLOCK_COUNT                 <= 0;
                                        REG_STATE                       <= STATE_WAIT;
                                end
                                else
                                begin
                                        OUT_RX_ERROR                    <=1;
                                        REG_STATE                       <=STATE_WAIT;
                                        REG_CLOCK_COUNT                 <=0;
                                end
                        end
                end
                default :
                        REG_STATE <= STATE_WAIT;
        endcase
 end
 function [NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] sum_of_bits;
                input [NUM_OF_DATA_BITS_IN_PACK-1:0] value;
                reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] sum;
                reg[NUM_OF_DATA_BITS_IN_PACK_LOG_2:0] i;
                begin
                        sum=0;
                        for (i=0;i<NUM_OF_DATA_BITS_IN_PACK;i=i+1)
                                sum=sum+value[i];
                        sum_of_bits=sum;
                end
 endfunction
endmodule
```

Полученные результаты

Набор параметров (22):

```
parameter PARITY=0
parameter NUM_OF_DATA_BITS_IN_PACK=8
```

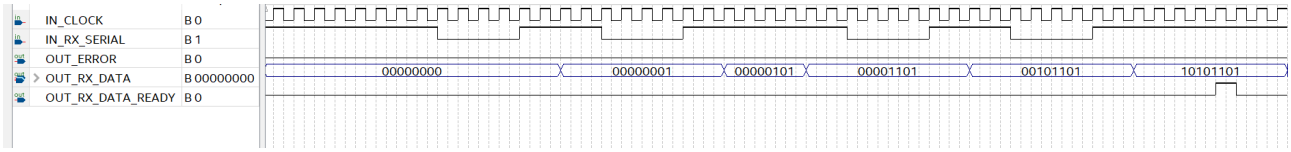

Рис. 43. Прием байта 10101101 модулем RX при наборе параметров (22).

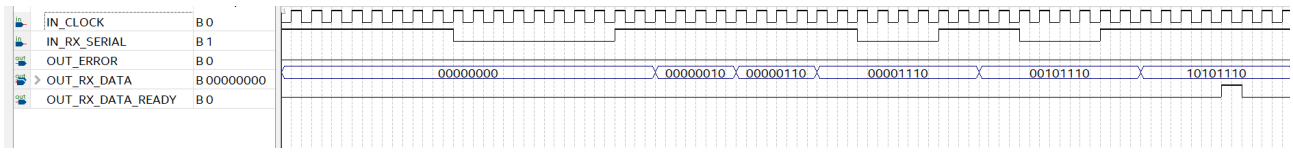

Рис. 44. Прием байта 10101110 модулем RX при наборе параметров (22).

Набор параметров (23):

parameter PARITY=1
parameter NUM_OF_DATA_BITS_IN_PACK=8

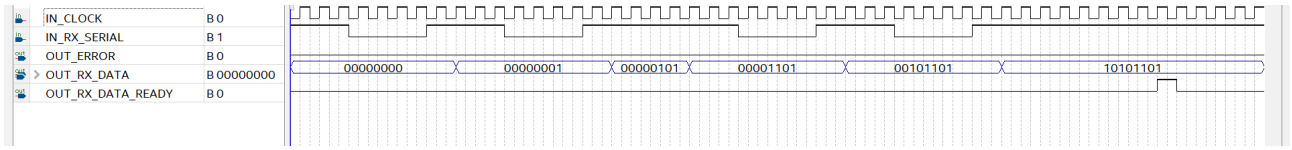

Рис. 45. Прием байта 10101101 модулем RX при наборе параметров (23).

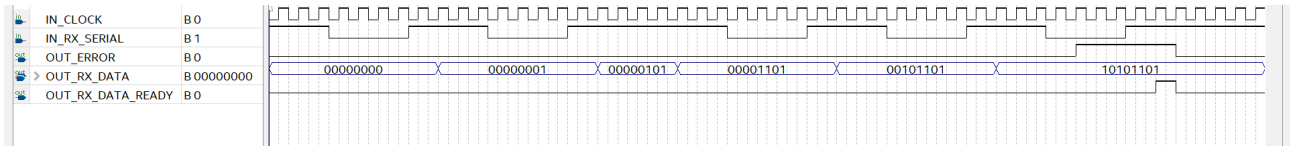

Рис. 46. Прием байта 10101101 модулем RX при наборе параметров (23). Детектирование ошибки в бите паритета.

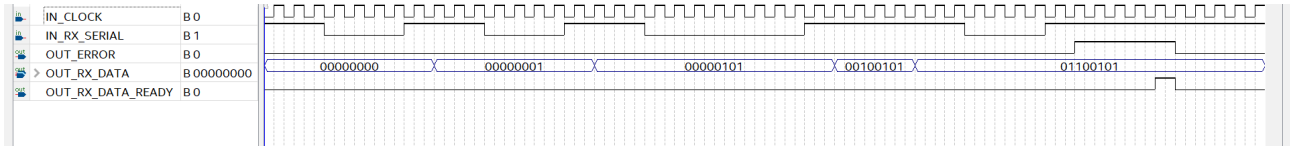

Рис. 47. Прием байта 01100101 модулем RX при наборе параметров (23). Детектирование ошибки в бите паритета.

Набор параметров (24):

parameter PARITY=2
parameter NUM_OF_DATA_BITS_IN_PACK=8

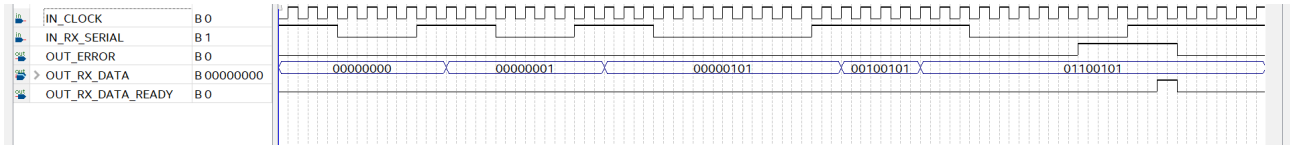

Рис. 48. Прием байта 01100101 модулем RX при наборе параметров (24). Детектирование ошибки в бите паритета.

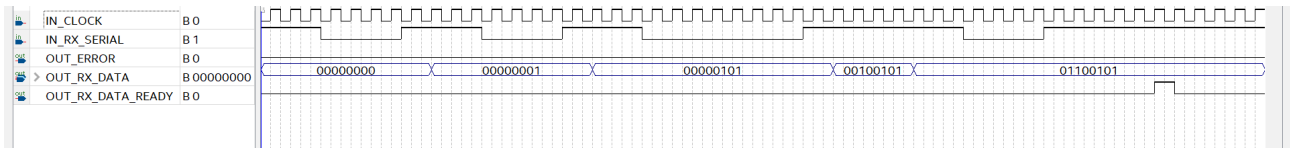

Рис. 49. Прием байта 01100101 модулем RX при наборе параметров (24).

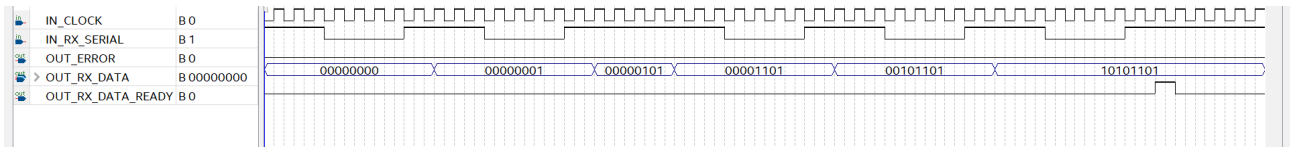

Рис. 50. Прием байта 10101101 модулем RX при наборе параметров (24).

Набор параметров (25):

parameter PARITY=0
parameter NUM_OF_DATA_BITS_IN_PACK=5

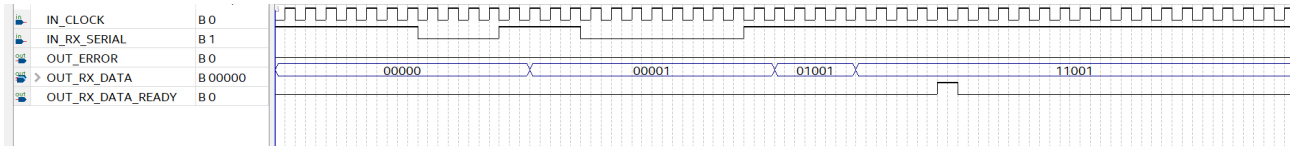

Рис. 51. Прием пакета 11001 модулем RX при наборе параметров (25).

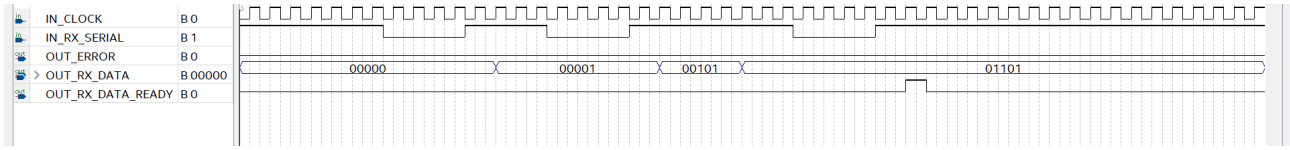

Рис. 52. Прием пакета 01101 модулем RX при наборе параметров (25).

Набор параметров (26):

parameter PARITY=1
parameter NUM_OF_DATA_BITS_IN_PACK=5

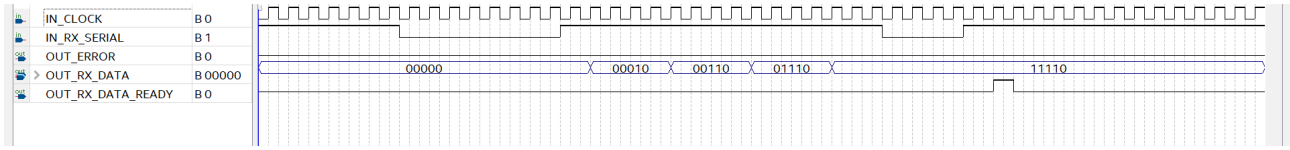

Рис. 53. Прием пакета 11110 модулем RX при наборе параметров (26).

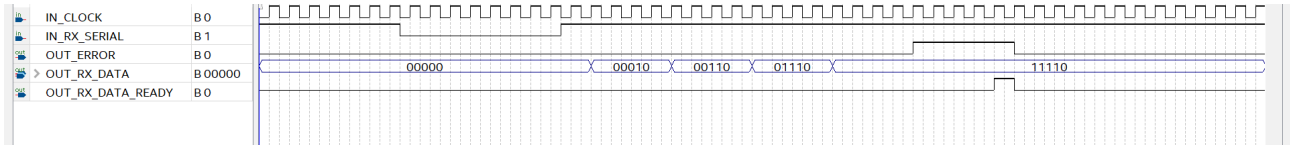

Рис. 54. Прием пакета 11110 модулем RX при наборе параметров (26).
Детектирование ошибки в бите паритета.

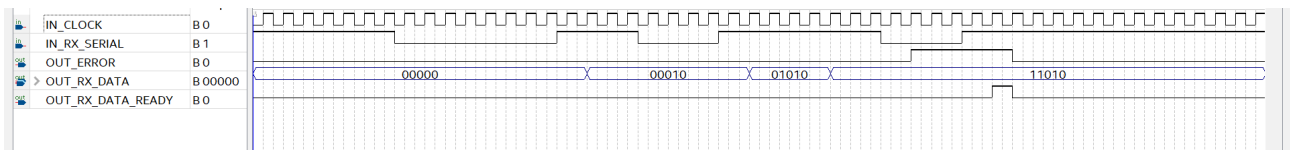

Рис. 55. Прием пакета 11010 модулем RX при наборе параметров (26).

Набор параметров (27):

parameter PARITY=2
parameter NUM_OF_DATA_BITS_IN_PACK=5

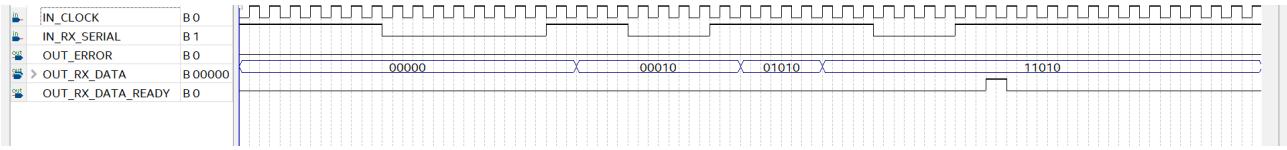

Рис. 56. Прием пакета 11010 модулем RX при наборе параметров (27).

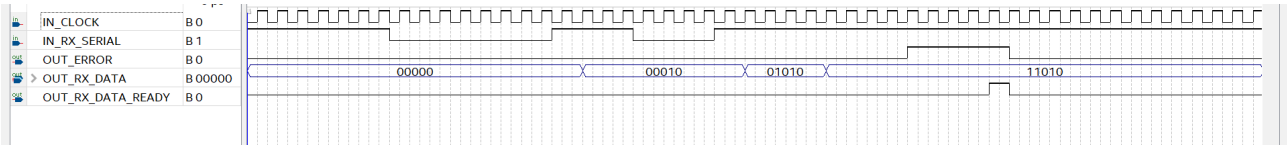

Рис. 57. Прием пакета 11010 модулем RX при наборе параметров (27). Детектирование ошибки в бите паритета.

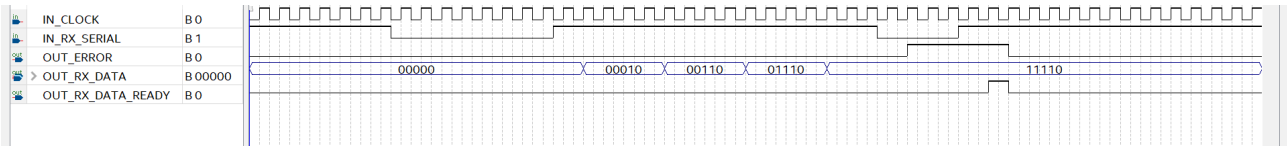

Рис. 58. Прием пакета 11110 модулем RX при наборе параметров (27). Детектирование ошибки в бите паритета.

Набор параметров (28):

parameter PARITY=0
parameter NUM_OF_DATA_BITS_IN_PACK=5

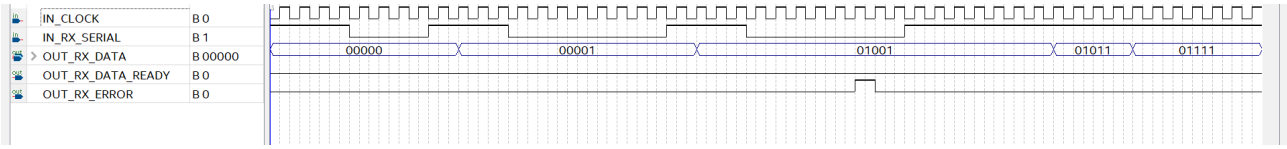

Рис. 59. Прием пакета 01111 модулем RX при наборе параметров (28). Тест на некорректный стоп-бит.

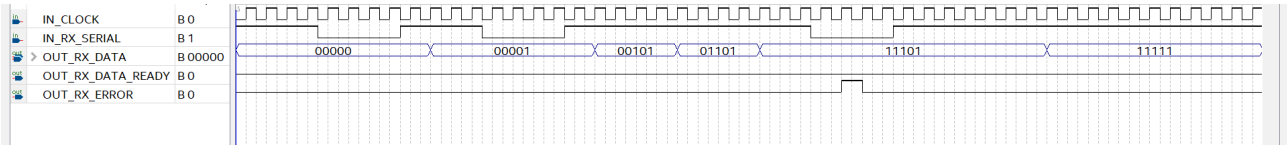

Рис. 60. Прием пакета 11111 модулем RX при наборе параметров (28). Тест на некорректный стоп-бит.

Набор параметров (29):

parameter PARITY=1
parameter NUM_OF_DATA_BITS_IN_PACK=5

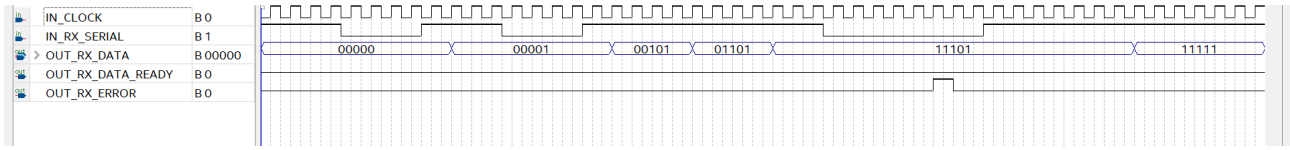

Рис. 61. Прием пакета 11111 модулем RX при наборе параметров (29). Тест на некорректный стоп-бит.

Набор параметров (30):

parameter PARITY=0
parameter NUM_OF_DATA_BITS_IN_PACK=10

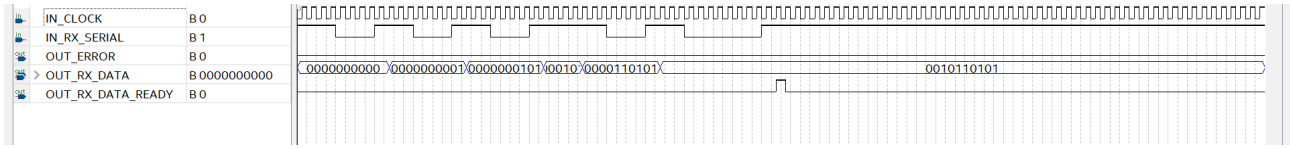

Рис. 62. Прием пакета 0010110101 модулем RX при наборе параметров (30).

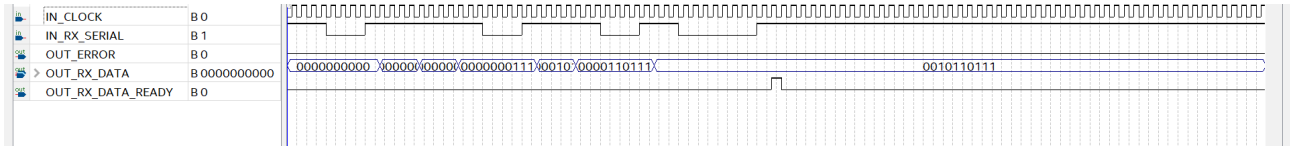

Рис. 63. Прием пакета 0010110111 модулем RX при наборе параметров (30).

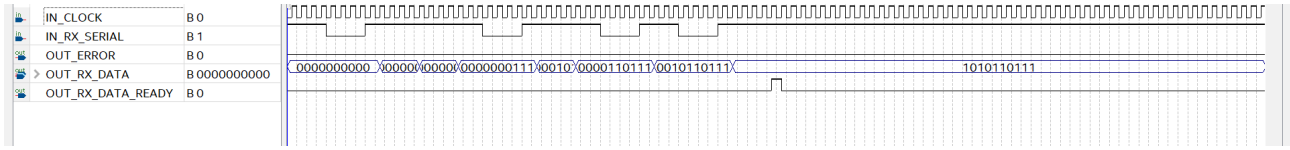

Рис. 64. Прием пакета 1010110111 модулем RX при наборе параметров (31).

Набор параметров (31):

parameter PARITY=1
parameter NUM_OF_DATA_BITS_IN_PACK=10

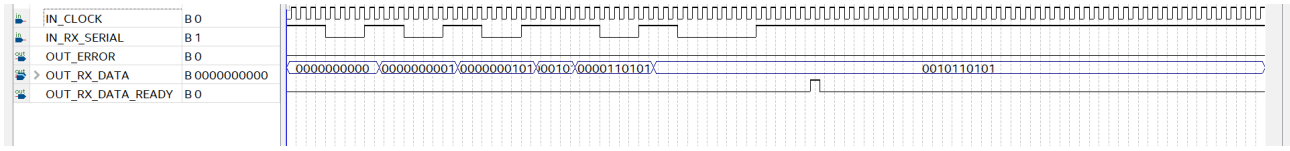

Рис. 65. Прием пакета 0010110101 модулем RX при наборе параметров (31).

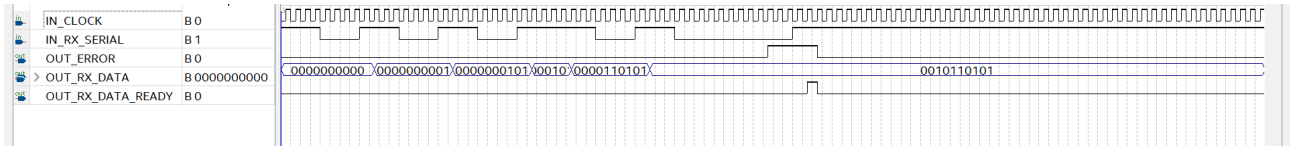

Рис. 66. Прием пакета 0010110101 модулем RX при наборе параметров (31). Детектирование ошибки в бит паритета.

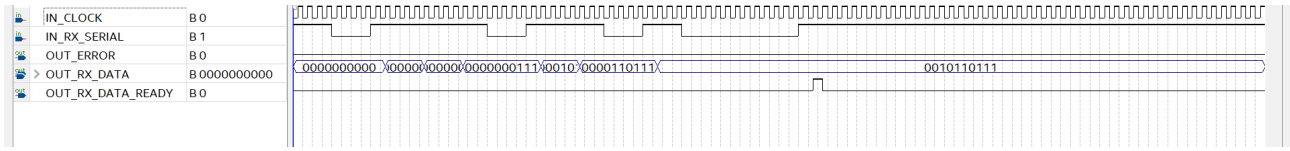

Рис. 67. Прием пакета 0010110111 модулем RX при наборе параметров (31).

Набор параметров (32):

parameter PARITY=2
parameter NUM_OF_DATA_BITS_IN_PACK=10

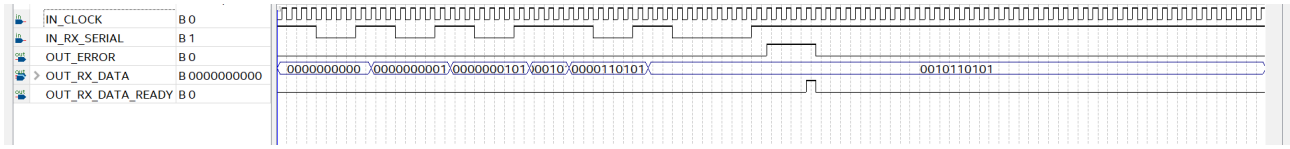

Рис. 68. Прием пакета 0010110101 модулем RX при наборе параметров (32). Детектирование ошибки в бите паритета.

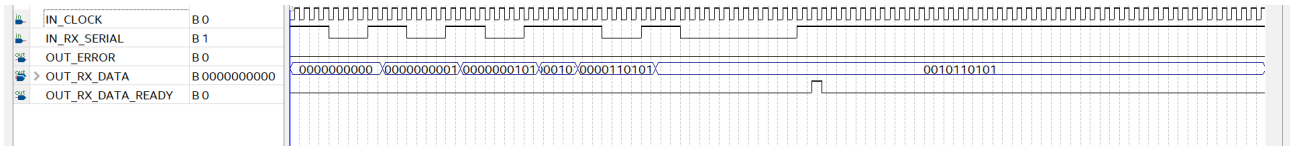

Рис. 69. Прием пакета 0010110101 модулем RX при наборе параметров (32).

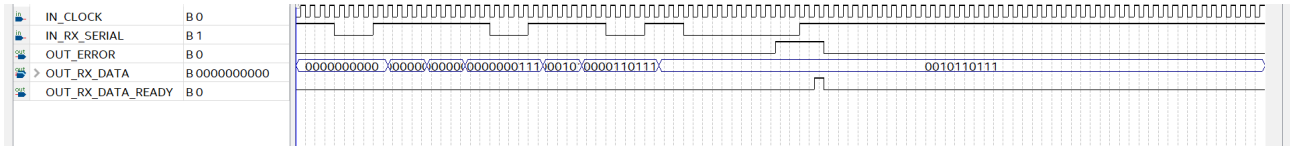

Рис. 70. Прием пакета 0010110111 модулем RX при наборе параметров (32). Детектирование ошибки в бит паритета.

Модуль приемника-передатчика TX_RX на языке Verilog.

В роли законченного функционального узла UART выступают совмещенные приемник и передатчик. Это позволяет завернуть два раздельных модуля в одну абстракцию, упростить работу с UART. Код приемника-передатчика описан ниже. Он содержит описанные ранее порты входа/выхода, регистры, параметры и модули. (Названия могут незначительно отличаться).

```
module UART_TX_RX_MODULE
#(
  parameter UART_BAUD_RATE                          =       1,
  parameter CLOCK_FREQUENCY                         =       4,
  parameter PARITY                                  =       2,
  parameter NUM_OF_DATA_BITS_IN_PACK                =       8,
  parameter NUMBER_STOP_BITS                        =       2
)
(
  input                                             IN_CLOCK,
  input                                             IN_TX_LAUNCH,
  input [NUM_OF_DATA_BITS_IN_PACK-1:0]              IN_TX_DATA,

  output                                            OUT_TX_ACTIVE,
  output                                            OUT_TX_DONE,
  output                                            OUT_TX_STOP_BIT_ACTIVE,
  output                                            OUT_TX_START_BIT_ACTIVE,
  output                                            OUT_RX_DATA_READY,
  output   [NUM_OF_DATA_BITS_IN_PACK-1:0]           OUT_RX_DATA,
  output                                            OUT_RX_ERROR,


   input                                            IN_RX_SERIAL,
   output                                           OUT_TX_SERIAL
);

        localparam NUM_OF_DATA_BITS_IN_PACK_LOG_2=$clog2(NUM_OF_DATA_BITS_IN_PACK) ;
        localparam
CLKS_PER_BIT_LOG_2=$clog2(NUMBER_STOP_BITS*CLOCK_FREQUENCY/UART_BAUD_RATE);

        UART_FPGA_TX #(
                .UART_BAUD_RATE(UART_BAUD_RATE),
                .CLOCK_FREQUENCY(CLOCK_FREQUENCY),
                .PARITY(PARITY),
                .CLKS_PER_BIT_LOG_2(CLKS_PER_BIT_LOG_2),
                .NUM_OF_DATA_BITS_IN_PACK(NUM_OF_DATA_BITS_IN_PACK),
                .NUM_OF_DATA_BITS_IN_PACK_LOG_2(NUM_OF_DATA_BITS_IN_PACK_LOG_2),
                .NUMBER_STOP_BITS(NUMBER_STOP_BITS)
        )
        TX
        (
                .IN_CLOCK(IN_CLOCK),
                .IN_TX_LAUNCH(IN_TX_LAUNCH),
                .IN_TX_DATA(IN_TX_DATA),
                .OUT_TX_ACTIVE(OUT_TX_ACTIVE),
                .OUT_TX_SERIAL(OUT_TX_SERIAL),
                .OUT_TX_DONE(OUT_TX_DONE),
                .OUT_TX_STOP_BIT_ACTIVE(OUT_TX_STOP_BIT_ACTIVE),
                .OUT_TX_START_BIT_ACTIVE(OUT_TX_START_BIT_ACTIVE)
        );

        UART_FPGA_RX #(
                .UART_BAUD_RATE(UART_BAUD_RATE),
                .CLOCK_FREQUENCY(CLOCK_FREQUENCY),
                .PARITY(PARITY),
                .CLKS_PER_BIT_LOG_2(CLKS_PER_BIT_LOG_2),
                .NUM_OF_DATA_BITS_IN_PACK(NUM_OF_DATA_BITS_IN_PACK),
                .NUM_OF_DATA_BITS_IN_PACK_LOG_2(NUM_OF_DATA_BITS_IN_PACK_LOG_2)
        )
        RX
        (
```

```
        .IN_CLOCK(IN_CLOCK),
        .IN_RX_SERIAL(IN_RX_SERIAL),
        .OUT_RX_DATA_READY(OUT_RX_DATA_READY),
        .OUT_RX_DATA(OUT_RX_DATA),
        .OUT_RX_ERROR(OUT_RX_ERROR)
    );

endmodule
```

Полученные результаты

Для получения объемных и информативных результатов был выбран путь написания тестбенчей под моделирование в среде Modelsim, это позволило смоделировать взаимодействие двух устройств через UART в рамках виртуального модельного стенда. Временные диаграммы находятся во вложениях к КП.

Модуль приемника-передатчика массивов TX_RX на языке Verilog.

Зачастую отправке подлежат сразу несколько (счетное количество) пакетов, было принято решение написать модуль, которые обладает буфером на отправку и прием , гибкость устройства была достигнута за счет глубокой параметризации, в частности, объема буфера. Важно то, что данные хранятся не в RAM, а в специально выделенных регистрах, что позволяет добиться очень высокой скорости взаимодействия.

За отправку и прием пакетов данных в рамках одного модуля отвечают разные подсистемы, развернутые в виде поведенческих конструкций. Важно заметить, что передатчик не обошелся без внедрения в него конечного автомата, а приемник его не содержит.

Передача массива пакетов осуществляется следующим образом. Сперва защелкивается комплексный вектор данных при детектировании на шине LAUNCH высокого уровня сигнала, после чего вектор декомпозируется последовательно, из него вычленяются пакеты данных по мере их отправки и получения от передатчика UART сигнала об окончании передачи и о передачи стоп бита.

Прием данных осуществляется в поведенческом блоке, в список чувствительности которого входят передние фронты OUT_RX_DATA_READY и CLEAR_RX_BUFFER. По переднему фронту первого сигнала происходит инкрементирование счетчика пакетов и запись в выходной комплексный вектор принятого пакета согласно его номеру. По второму фронту идет очистка «буфера» и счетчика пакетов.

Тестирование модуля приемника-передатчика массива проходило через написание тестбенчей, в которых проверялись общие и частные возможности и случаи поведения. Выпуск в виде временных диаграмм и кода программы находится в приложении к КП.