Лекция ИЦУ 13 ноября 2014 г.

1.1 Интерфейс RS-232 (COM-порт)

Интерфейс RS-232, официально называемый "EIA/TIA–232–E", но более известный как интерфейс "COM-порта", ранее был одним из самых распространенных интерфейсов в компьютерной технике. Он и до сих пор встречается в настольных компьютерах, несмотря на появление более скоростных и "интеллектуальных" интерфейсов, таких как USB и FireWare. К его достоинствам с точки зрения разработчиков аппаратуры можно отнести простоту реализации протокола. Физический интерфейс RS-232 реализуется разъемом типа DB-9M

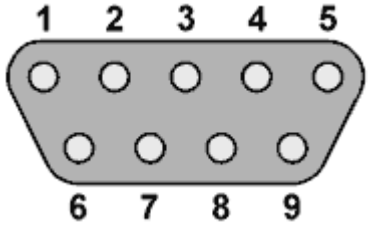

Рис.1 9-контактная вилка типа DB-9M COM порта компьютера

Таблица 1

№	Сигнал	Напра вление	Назначение
1	CD	Вход	Обнаружена несущая
2	RXD	Вход	Принимаемые данные
3	TXD	Выход	Передаваемые данные
4	DTR	Выход	Компьютер готов
5	GND	-	Общий провод
6	DSR	Вход	Устройство готово
7	RTS	Выход	Компьютер готов к передаче
8	CTS	Вход	Устройство готово к приему
9	RI	Вход	Обнаружен вызов

1.2 Электрические характеристики

Логические уровни передатчика: "0" – от +5 до +15 Вольт, "1" – от -5 до -15 Вольт.
Логические уровни приемника: "0" – выше +3 Вольт, "1" – ниже -3 Вольт.
Максимальная нагрузка на передатчик: входное сопротивление приемника не менее 3 кОм.

1.3 Описание основных сигналов интерфейса

CD – Устройство устанавливает этот сигнал, когда обнаруживает несущую в принимаемом сигнале. Обычно этот сигнал используется модемами, которые таким образом сообщают хосту об обнаружении работающего модема на другом конце линии.

RXD – Линия приема хостом данных от устройства. Подробно описана в разделе "Протокол обмена данными".

TXD – Линия передачи хостом данных к устройству. Подробно описана в разделе "Протокол обмена данными".

DTR – Хост устанавливает этот сигнал, когда готов к обмену данными. Фактически сигнал устанавливается при открытии порта коммуникационной программой и остается в этом состоянии все время, пока порт открыт.

DSR – Устройство устанавливает этот сигнал, когда включено и готово к обмену данными с хостом. Этот и предыдущий (DTR) сигналы должны быть установлены для обмена данными.

RTS – Хост устанавливает этот сигнал перед тем, как начать передачу данных устройству, а также сигнализирует о готовности к приему данных от устройства. Используется при аппаратном управлении обменом данными.

CTS – Устройство устанавливает этот сигнал в ответ на установку хостом предыдущего (RTS), когда готово принять данные (например, когда предыдущие присланные хостом данные переданы модемом в линию или есть свободное место в промежуточном буфере).

RI – Устройство (обычно модем) устанавливает этот сигнал при получении вызова от удаленной системы, например при приеме телефонного звонка, если модем настроен на прием звонков.

1.4 Протокол обмена данными

В протоколе RS-232 существуют два метода управления обменом данных: аппаратный и программный, а также два режима передачи: синхронный и асинхронный. Протокол позволяет использовать любой из методов управления совместно с любым режимом передачи. Также допускается работа без управления потоком, что подразумевает постоянную готовность компьютера и устройства к приему данных, когда связь установлена (сигналы DTR и DSR установлены).

Аппаратный метод управления реализуется с помощью сигналов RTS и CTS. Для передачи данных компьютер устанавливает сигнал RTS и ждет установки устройством сигнала CTS, после чего начинает передачу данных до тех пор, пока сигнал CTS установлен. Сигнал CTS проверяется компьютером непосредственно перед началом передачи очередного байта, поэтому байт, который уже начал передаваться, будет передан полностью независимо от значения CTS. В полудуплексном режиме обмена данными (устройство и компьютер передают данные по очереди, в полнодуплексном режиме они могут делать это одновременно) снятие сигнала RTS компьютером означает его переход в режим приема.

Программный метод управления заключается в передаче принимающей стороной специальных символов остановки (символ с кодом 0x13, называемый XOFF) и возобновления (символ с кодом 0x11, называемый XON) передачи. При получении данных символов передающая сторона должна соответственно остановить передачу или возобновить ее (при наличии данных, ожидающих передачи). Этот метод проще с точки зрения реализации аппаратуры, однако обеспечивает более медленную реакцию и соответственно требует заблаговременного извещения передатчика при уменьшении свободного места в приемном буфере до определенного предела.

Синхронный режим передачи подразумевает непрерывный обмен данными, когда биты следуют один за другим без дополнительных пауз с заданной скоростью. Этот режим COM-портом **не поддерживается.**

Асинхронный режим передачи состоит в том, что каждый байт данных (и бит контроля четности, в случае его наличия) "оборачивается" синхронизирующей последовательностью из одного нулевого старт-бита и одного или нескольких единичных стоп-битов.

Один из возможных алгоритмов работы приемника следующий:

1. Ожидать уровня "0" сигнала приема (RXD в случае компьютера, TXD в случае устройства).
2. Отсчитать половину длительности бита и проверить, что уровень сигнала все еще "0"
3. Отсчитать полную длительность бита и текущий уровень сигнала записать в младший бит данных (бит 0)
4. Повторить предыдущий пункт для всех остальных битов данных
5. Отсчитать полную длительность бита и текущий уровень сигнала использовать для проверки правильности приема с помощью контроля четности (см. далее)
6. Отсчитать полную длительность бита и убедиться, что текущий уровень сигнала "1".
7. Вернуться к ожиданию начала следующего байта данных (шаг 1)

Протокол имеет ряд переменных параметров, которые должны быть приняты одинаковыми на стороне приемника и на стороне передатчика для успешного обмена данными:

- **Скорость обмена данными** (VEL) задается в битах в секунду, определяя длительность одного бита, выбирается из ряда стандартных значений (300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 430800, 921600), но могут быть и нестандартными, если поддерживаются обеими сторонами;
- **Количество бит данных** может быть от 4 до 8;
- **Контроль четности** может быть четным ("even", когда общее число единичных битов в принятых данных, включая сам бит четности, должно быть четным), нечетным ("odd", когда общее число единичных битов в принятых данных, включая сам бит четности, должно быть четным) или вообще отсутствовать;

Длина стоп-бита может составлять одну, полторы или две длительности бита.

На макете NEXYS-2, на котором рекомендуется выполнение работы, через преобразователь уровней ST3232 выведены только два сигнала: RXD и TXD. Остальные выводы соединены перемычками так, чтобы имитировать готовность к приему и передаче обеих сторон.

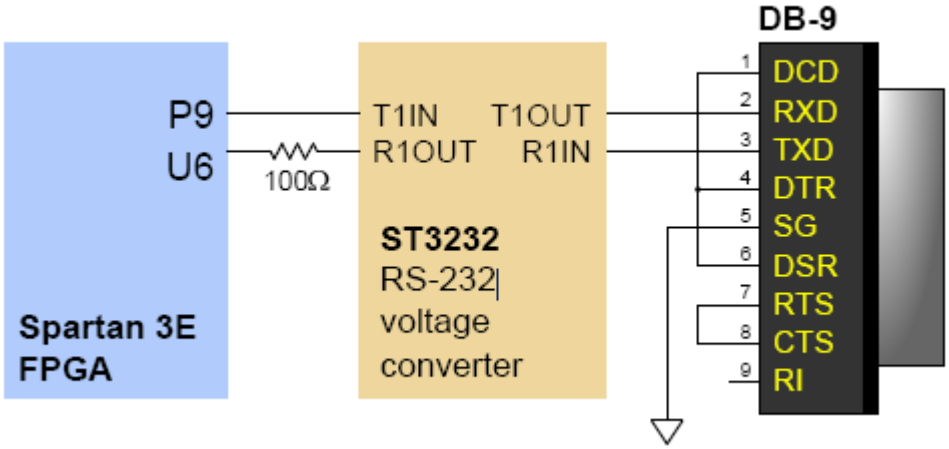

Рис.2 Схема соединений COM порта на макете NEXYS-2

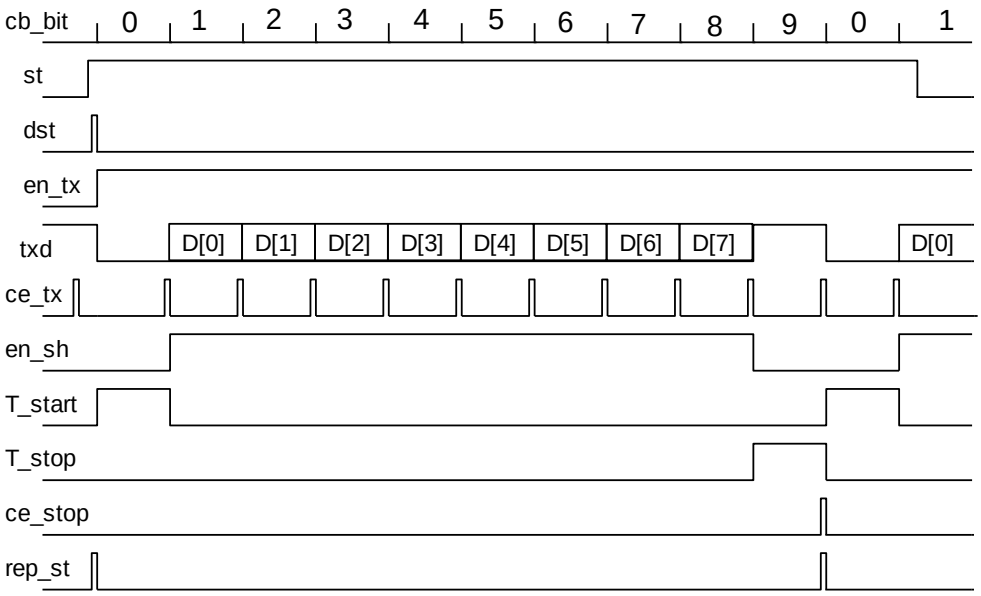

Рис.3 Временные диаграммы передатчика одного байта UTXD1B

1.4 Схема модуля передатчика одного байта UTXD1B

```
`define Fclk    50000000        //50MHz
`define VEL     6250000         // для моделирования
`define Nt      `Fclk/`VEL      // Nt=[54.25] = 54
module UTXD1B(      input clk,              output wire ce,
                    input st,               output reg en_tx=0,
                    input[7:0]dat,          output reg [3:0] cb_bit=0,
                                            output wire T_start,
                                            output wire en_sh,
                                            output wire T_stop,
                                            output wire ce_stop,
                                            output wire txd,
                                            output reg [7:0] sr_dat=0);
reg [7:0]cb_tact=0 ;
assign dst = st & !en_tx ;
assign ce = (cb_tact==`Nt) ;
assign T_start = ((cb_bit==0) & en_tx) ;
assign en_sh = (cb_bit<9) & (cb_bit>0);
assign T_stop = (cb_bit==9) ;
assign ce_stop = T_stop & ce ;
assign txd = T_start? 0 : en_tx? sr_dat[0] : 1 ;

always @ (posedge clk) begin
cb_tact <= (dst | ce)? 1 : cb_tact+1 ;
en_tx  <= st? 1 : ce_stop? 0 : en_tx ;
cb_bit <= dst? 0 : (ce & en_tx)? cb_bit+1 : cb_bit ;
sr_dat <= (T_start & ce)? dat : (en_sh & ce)? sr_dat>>1 | 1<<7  : sr_dat ;
end
endmodule
```

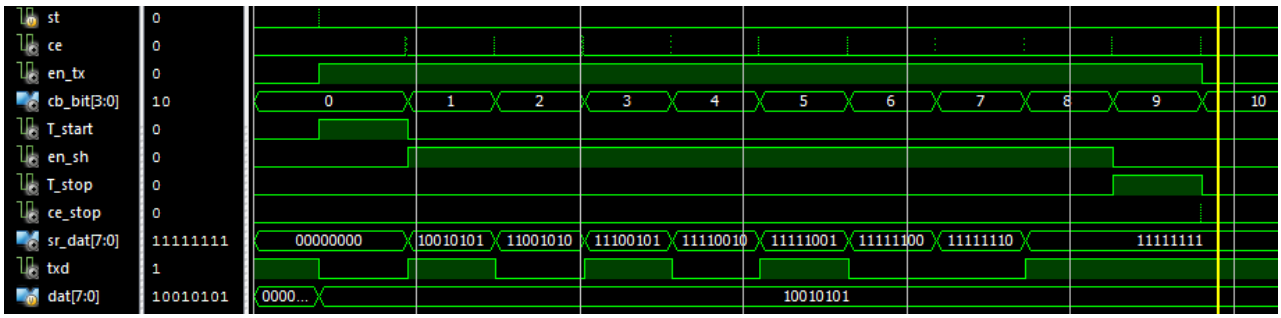

Рис.4 Пример моделирования модуля UTXD1B передатчика одного байта

В модуле UTXD1B число разрядов счетчика такта cb_tact должно соответствовать заданной скорости VEL. Например при VEL=115200 бод (Nt=50000000/115200=434) счетчик должен иметь 9 разрядов, а при скорости 300 бод -18 разрядов (Nt=50000000/300=166667). При моделировании можно для наглядности увеличить VEL до 6500000 (Nt=8).

Если для защиты передаваемого пакета байт от ошибок недостаточно бита контроля четности каждого байта, то к передаваемому пакету добавляется контрольный код CRC (Cyclic Redundancy Code (CRC) - циклический избыточный код). Основная идея алгоритма вычисления CRC состоит в представлении последовательности бит в виде полинома с двоичными коэффициентами и делении его на порождающий полином и использовании остатка от этого деления в качестве контрольного кода. Получив сообщение, приёмник должен выполнить аналогичное действие и сравнить полученный результат с принятым контрольным кодом. Сообщение считается достоверным, если выполняется это равенство. Наиболее часто используется полиномы *CRC*-8 ($x^8 + x^5 + x^4 + 1$) и *CRC*-16 ($x^{16} + x^{15} + x^2 + 1$, Bisync, Modbus, USB, ANSI X3.28).

Аппаратная реализация вычислителя CRC кода очень проста и эффективна. Например, для вычислителя CRC-8 необходим 8-ми разрядный регистр сдвига и 3 элемента XOR2 (исключающее ИЛИ) в цепях обратной связи. Схема вычислителя CRC-8 для порождающего полинома ($x^8 + x^5 + x^4 + 1$) приведена на рис.5, а вариант представления этой схемы и схемы вычисления CRC-16 на Verilog-е в П.1.5 и в П.1.6.

Эффективность аппаратной реализации состоит в том, что при передаче не надо располагать всеми битами передаваемого пакета, а при приеме не надо дожидаться, пока будет принят весь пакет, т.к. CRC код вычисляется на «ходу» по мере поступления передаваемых или принимаемых бит. Более того, если на приемном конце будет продолжаться вычисление CRC кода до конца, включая и присоединенный к передаваемому пакету CRC код, то при отсутствии ошибок результатом вычисления CRC кода будет ноль. Это очень удобно для контроля отсутствия ошибок в принятом пакете. Еще надо отметить, что число бит в пакете может быть произвольным, не кратным 8, т.е. байты COM порта можно передавать и с битом контроля четности.

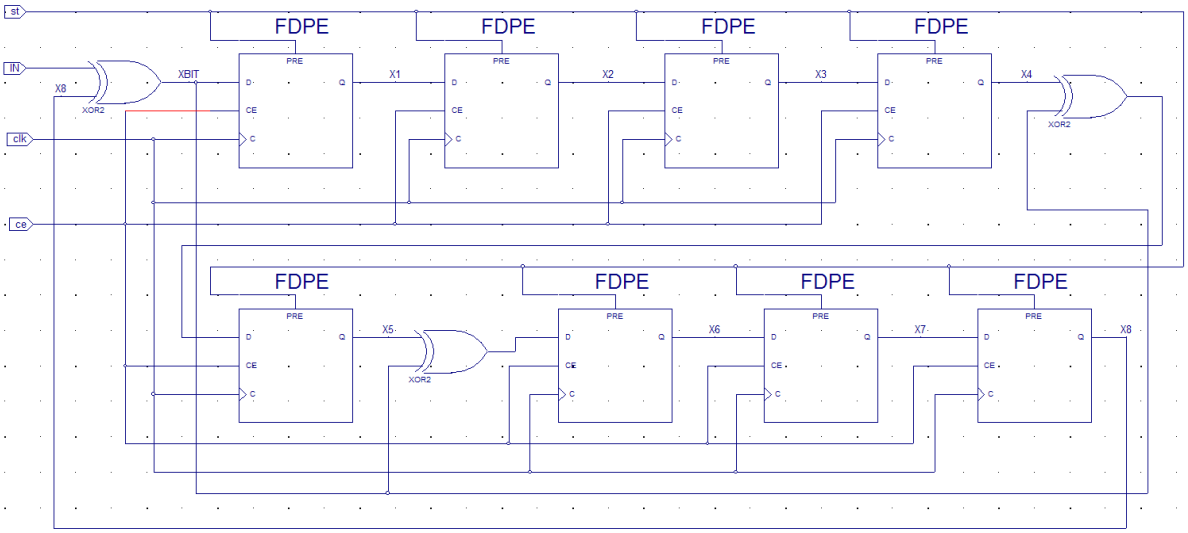

Рис.5 Схема вычислителя кода CRC-8 для порождающего полинома $x^8 + x^5 + x^4 + 1$

1.5 Схема вычислителя кода CRC-8 ($X^8 + X^5 + X^4 + 1$)

```
module CRC8_BL   (input st,        output reg[7:0] crc,// X[1:8] = 0001 1001
                  input ce,        output wire ok_crc8,
                  input c,
                  input in) ;
assign ok_crc8 = (crc==8'h00);
wire xbit = crc[0] ^ in ;
always @(posedge c) begin
crc <= st? 8'hFF : (xbit & ce)? (((crc^8'h18)>>1) | 1<<7): ce? (crc>>1)  : crc ;
end
endmodule
```

1.6 Схема модуля CRC-16 16 ($X^{16} + X^{15} + X^2 + 1$)

```
module CRC16_BL  (input ce,        output reg [15:0] crc,//X[1:16]=0100 0000 0000 0011
                  input clk,       output wire ok_crc16,
                  input st,
                  input in);
assign ok_crc16 = (crc==16'h0000);
wire xbit = crc[0] ^ in ;
always @ (posedge clk) begin
crc <= st? 16'hFFFF : (xbit & ce)? (((crc^16'h4002)>>1) | 1<<15): ce? (crc>>1) : crc ;
end
endmodule
```

На рис.6 приведен пример временных диаграмм передачи и приема блока байт с CRC-16. В этом примере конец блока байт (кадра) определяется по длительности паузы между байтами, т.к. предполагается, что на приемном конце число байт пакета неизвестно. Первый байт com это код команды, например:

com=8’h00 – загрузить байт в регистр,
com=8’h80 – прочитать регистр,
com=8’h01 – записать байт в память,
com=8’h81 – прочитать байт из памяти.

Второй байт lbl это длина блока данных, т.е. сколько байт прочитать или записать.
Третий adr[15:8] и четвертый adr[7:0] байты это адрес куда писать или откуда читать.
Последние два байта crc[7:0] и crc[15:8] это код CRC-16. В поле данных dat0, dat1,… - данные, предназначенные для записи. Для команд чтения 8’h80 и 8’h81 поля данных нет.

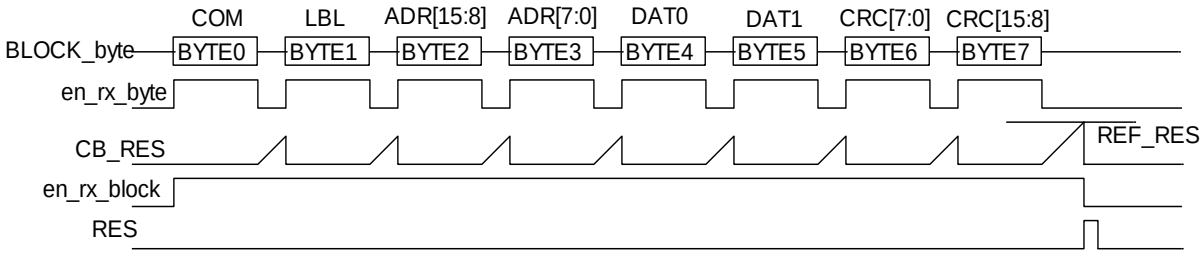

Рис.6 Пример временных диаграмм передачи и приема блока байт с CRC-16

1.7 Схема модуля передатчика блока байт с CRC-16

```
module UTXD_BL(  input st,              output wire txd_bl,
                 input [7:0] lbl,       output reg en_tx_bl,
                 input [7:0] dat,       output wire ce_stop,
                 input clk              output reg [8:0] cb_byte=0,
                                        output wire [15:0]crc);
wire txd, ce_tact, en_sh ;
assign txd_bl = en_tx_bl? txd : 1 ;
assign rep_start = st | ((cb_byte>1) & ce_stop) ;
wire T_hb_crc = (cb_byte==1) ;  wire T_lb_crc = (cb_byte==2) ;  wire T_dat = (cb_byte>2) ;
wire [7:0]dat_crc = T_hb_crc? crc[15:8] :  T_lb_crc? crc[7:0] : dat;
wire ce_crc = T_dat & ce_tact & en_sh ;

always @ (posedge clk) begin
cb_byte <= st? lbl+2 :  ce_stop? cb_byte-1 : cb_byte ;
en_tx_bl <= ((cb_byte==1) & ce_stop)? 0 : st? 1 : en_tx_bl ;
end
UTXD1B DD1 (     .clk(clk),       .ce_stop(ce_stop),
                 .dat(dat_crc),   .txd(txd),
                 .st(rep_start),  .ce(ce_tact),
                                  .en_sh(en_sh));
CRC_BL DD2 (     .ce(ce_crc),     .crc(crc),
                 .clk(clk),
                 .st(st),
                 .inp(txd));
endmodule
```

В передатчике блока байт **UTXD_BL_CRC** по сравнению с передатчиком одного байта **UTXD1B** добавлен вход lbl[7:0], на который задается число передаваемых байт пакета. Импульсом старта st с длительностью в один период сигнала синхронизации Tclk=20ns в счетчик числа байт cb_byte загружается число lbl+2, а после каждого переданного байта сигналом ct_stop декрементируется на 1. На интервале T_dat, пока cb_byte>2 на вход dat передатчика одного байта подаются внешние данные, при cb_byte=2 – младший байт crc[7:0], а при cb_byte=1 – старший байт crc[15:8]. Вычисление crc происходит только на интервале T_dat. Сигнал en_tx_bl запуска передатчика одного байта удерживается на высоком уровне до тех пор, пока не будут переданы все байты включая crc. Выходной порт dat_crc[7:0] нужен только для моделирования.

1.8 Схема модуля передатчика ответа на команду

```
module TXD_RET_BL(input clk,            output wire TXD,
                  input st,             output reg[15:0]ret_adr,
                  input [ 7:0] com,     output wire en_tx_bl,
                  input [ 7:0] lbl,
                  input [15:0] adr,
                  input [ 7:0] ret_dat );
reg [2:0]cb_tx_byte ;
wire ce_stop ;
wire COM_wr = (com==8'h00)|(com==8'h01) ;
wire COM_rd = (com==8'h80)|(com==8'h81) ;

always @ (posedge clk) begin
cb_tx_byte <= st? 0  : (ce_stop & (cb_tx_byte<4) )? cb_tx_byte+1 : cb_tx_byte  ;
ret_adr    <= st? adr: (ce_stop & (cb_tx_byte==4))? ret_adr+1    : ret_adr ;
end

wire[7:0]tx_lbl = COM_wr? 4 :
                  COM_rd? lbl + 4 : 1 ;

wire[7:0]tx_dat = (cb_tx_byte==0)? com :
                  (cb_tx_byte==1)? lbl :
                  (cb_tx_byte==2)? adr[15:8] :
                  (cb_tx_byte==3)? adr[ 7:0] : ret_dat ;

UTXD_BL_CRC DD1 (   .clk(clk),          .txd_bl(TXD),
                    .dat(tx_dat),       .ce_stop(ce_stop),
                    .st(st),            .en_tx_bl(en_tx_bl),
                    .lbl(tx_lbl) );
endmodule
```

Модуль передатчика ответа на команду TXD_RET_BL включает в себя модуль передатчика блока байт UTXD_BL_CRC и имеет два дополнительных входных порта com[7:0] и adr[15:0]. Для команд записи (8'h00 или 8'h01) он возвращает, без учета crc только 4 байта : com[7:0], lbl[7:0], adr[15:8] и adr[7:0]. Для команд чтения (8'h80 или 8'h81) он возвращает эти же 4 байта и поле данных, состоящее из lbl байт данных. Если же первый байт не совпадает ни с одной из команд, то возвращается только этот байт и crc. На рис.7 приведен пример временных диаграмм ответа на команду чтения двух байт.

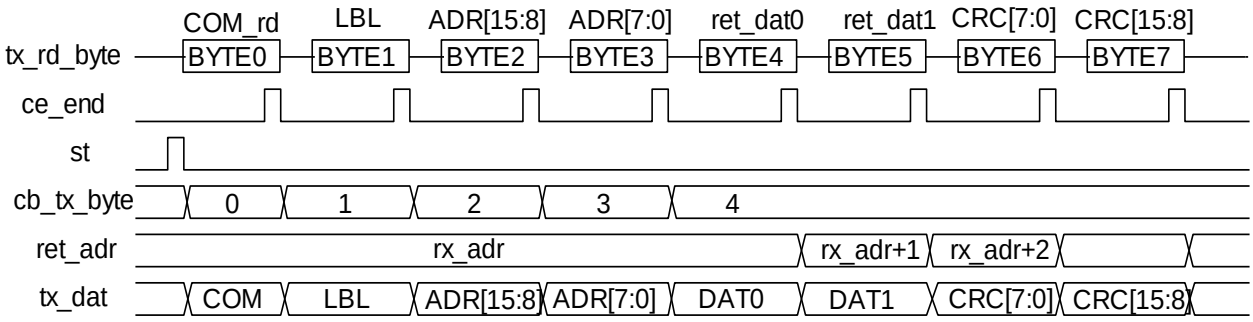

Рис.7 Пример временных диаграмм ответа на команду чтения двух байт

1.9 Схема модуля точечного приемника одного байта

```
`define rx_clk 50000000 //50Mhz
`define rx_vel  6250000 //для моделирования
`define rx_Nt `rx_clk/`rx_vel //8  для моделирования
module URXD1B_D(input in,          output reg en_rx_byte=0,
                input clk,         output reg [7:0] sr_dat=0,
                                   output wire ok_rx_byte,
                                   output wire start_rx,
                                   output wire en_sh,
                                   output wire ce,
                                   output wire cet,
                                   output wire rx_bit);
reg tin=0, ttin=0;
reg [3:0]cb_bit=0 ; reg [8:0]cb_tact=0 ;
assign ce = (cb_tact==`rx_Nt-1) ;
wire din = !tin & ttin ;
wire T_stop = (cb_bit==9) ;
assign start_rx = din & !en_rx_byte ;
assign en_sh = (cb_bit>0) & (cb_bit<9);
assign cet = (cb_tact==(`rx_Nt/2)-1);
assign rx_bit = tin ;
assign ok_rx_byte = (cet & T_stop & en_rx_byte & tin);

always @ (posedge clk) begin
tin <= in ; ttin <= tin ;
cb_tact <= (start_rx | ce)? 0 : cb_tact+1;
en_rx_byte <= (T_stop & cet)? 0 : (cet & !tin)? 1: en_rx_byte ;
cb_bit <=  (start_rx | (T_stop & cet))? 0 : (ce & en_rx_byte)? cb_bit+1 : cb_bit ;
sr_dat <= start_rx? 0 : (cet & en_sh)? sr_dat >>1 | tin<<7 : sr_dat ;//in
end
endmodule
```

Приемник байта можно строить по классическому алгоритму, что и делается в UART приемниках микроконтроллеров, т.е. принимать решение о значении очередного бита в середине такта. Схема реализации упрощенного варианта такого приемника из элементов ПЛИС приведена в П.1.8.

В модуле URXD1B_D число разрядов счетчика такта cb_tact также как и модуле передатчика UTXD1B должно соответствовать заданной скорости VEL.

Сигнал ce соответствует границам тактов, а сигнал cet – серединам тактов. Периоды сигналов ce и cet должны быть равны ожидаемой длительности бита принимаемых данных. Генератор сигналов ce и cet фазируется первым спадом сигнала in. Решение о конце приема байта принимается на девятом такте по сигналу cet. Данные D[0],…D[7] задвигаются в регистр сдвига sr_dat[7:0] по сигналу cet на интервале данных en_sh. Сигнал подтверждения правильности приема байта формируется на девятом такте при условии, что in=1.

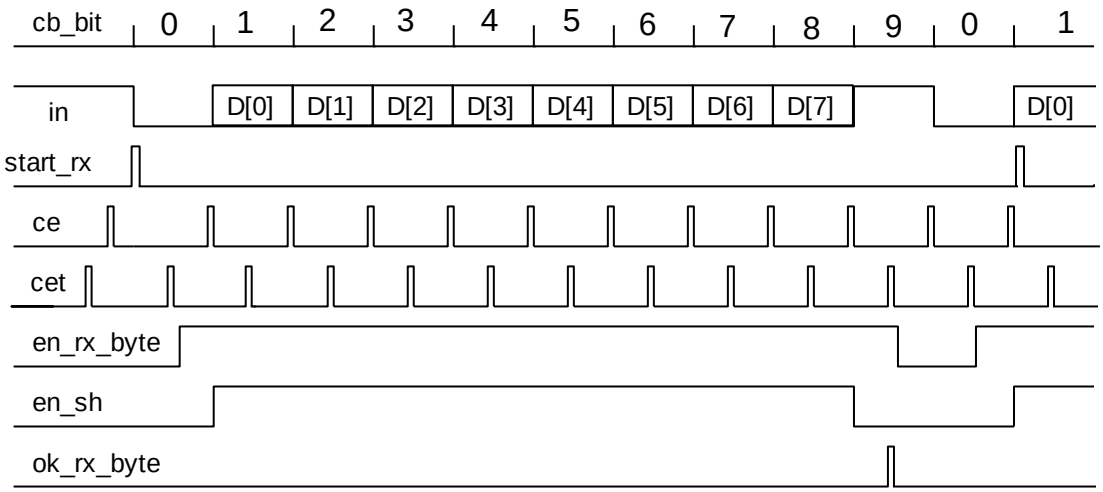

Рис.7 Временные диаграммы точечного приемника одного байта URXD1B_D

1.10 Схема модуля интегрирующего приемника одного байта

```
`define rx_clk 50000000       //50Mhz
`define rx_vel  6250000       // для моделирования
`define rx_Nt `rx_clk/`rx_vel // rx_Nt=8  для моделирования
module URXD1B_I( input in,          output reg en_rx_byte=0,
                 input clk,         output reg [7:0] sr_dat=0,
                                    output wire ok_rx_byte,
                                    output wire ce,
                                    output wire en_sh,
                                    output wire rx_bit,
                                    output wire start_rx);
reg tin=0, ttin=0, T_start=0;
reg [3:0]cb_bit=0 ; reg [8:0]cb_INT=0 ; reg [8:0]cb_tact=0 ;
assign ce = (cb_tact==`rx_Nt-1) ;
assign start_rx = !tin & ttin & !en_rx_byte ;
assign en_sh = (cb_bit>0) & (cb_bit<9);
wire T_stop = (cb_bit==9)    ;
assign rx_bit = (cb_INT>=`rx_Nt/2) ;
assign ok_rx_byte = (ce & T_stop & en_rx_byte & rx_bit);

always @ (posedge clk) begin
tin <= in ; ttin <= tin ;
cb_tact <= (start_rx | ce)? 0 : cb_tact+1 ;
T_start <= start_rx? 1 : ce? 0 : T_start ;
en_rx_byte <= (T_start & rx_bit & ce)? 1 : (T_stop & ce)? 0 : en_rx_byte ;
cb_bit <= (start_rx | (T_stop & ce))? 0 : (ce & (en_rx_byte | T_start))? cb_bit+1 : cb_bit ;
sr_dat <= start_rx? 0 : (ce & en_sh)? sr_dat >>1 | rx_bit<<7 : sr_dat ;
cb_INT <= (ce | start_rx)? 0 : ((T_start & !tin) | (en_rx_byte & tin))? cb_INT+1 : cb_INT ;
end
endmodule
```

В интегрирующем приемнике решение о значении очередного бита принимается в конце такта по результату интегрирования. Для этого используется отдельный счетчик cb_INT (интегратор) сигналов синхронизации clk, который сигналом ce сбрасывается на границах тактов в 0, а счет разрешается, если in=1. Максимальное число в конце такта на счетчике cb_INT может достигать rx_Nt=rx_clk/rx_vel. Если число на счетчике cb_INT превышает некоторый порог, например rx_Nt/2, то принимается решение, что бит на этом

интервале равен 1,т.е. в регистр сдвига приемника «задвигается» сигнал с выхода компаратора rx_bit = (cb_INT>=`rx_Nt/2).

В модуле URXD1B_I не только число разрядов счетчика такта cb_tact, но и число разрядов счетчика интегратора cb_INT должно соответствовать заданной скорости rx_vel. Например при rx_vel =9600 бод (Nt= 50000000/9600=5208) эти счетчики должны иметь по 13 разрядов.

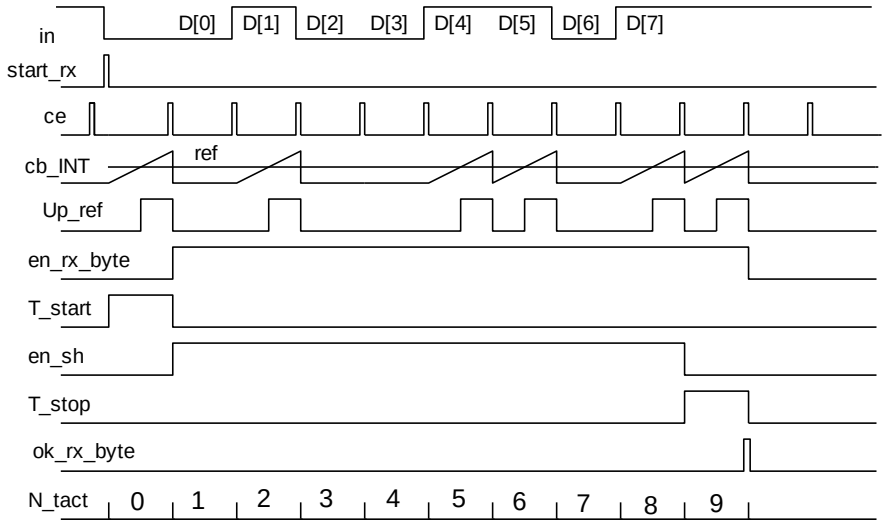

Рис.8 Временные диаграммы интегрирующего приемника одного байта URXD1B_I

1.11 Схема модуля приемника блока байт с кодом CRC-16

```
module URXD_BL_CRC(   input in,        output wire[7:0] dat_BUF,
                      input clk,       output reg [7:0]cb_rd_BUF=0,
                                       output wire ce_dat_BUF,
                                       output reg en_rd_BUF=0,
                                       output wire Stop_rd_BUF,
                                       output wire [15:0] rx_crc);

reg [7:0]cb_rx_byte=0 ;  reg en_rx_block=0, ten_rd_BUF=0 ;
reg [7:0]cb_res=0 ;  reg [3:0]cb_ce_rd_BUF=0 ;
wire [7:0]rx_dat;
wire ce_rx, en_sh, start_rx_byte, ok_rx_byte, en_rx_byte ;//
wire start_rx_block = start_rx_byte & !en_rx_block ;
wire ce_crc = en_sh & ce_rx;
wire ce_rd_BUF = (cb_ce_rd_BUF==15) ;
wire res = (cb_res==5) & ce_rx ;
wire ok_crc = (rx_crc==0) ;
wire ok_rx_bl = res & ok_crc ;
wire end_rd_BUF      = ce_rd_BUF & (cb_rd_BUF==cb_rx_byte-1) ;
assign ce_dat_BUF = en_rd_BUF & ce_rd_BUF ;
assign Stop_rd_BUF = ce_rd_BUF & !en_rd_BUF & ten_rd_BUF ;

always @ (posedge clk) begin
cb_ce_rd_BUF <= ok_rx_bl? 0 : cb_ce_rd_BUF + 1 ;
en_rd_BUF <= (end_rd_BUF | start_rx_block)? 0 : ok_rx_bl? 1 : en_rd_BUF ;
cb_rd_BUF <= ok_rx_bl? 0 : (ce_rd_BUF & en_rd_BUF)? cb_rd_BUF+1 : cb_rd_BUF ;
cb_rx_byte <= start_rx_block? 0 : ok_rx_byte? cb_rx_byte+1 : cb_rx_byte ;
cb_res <= en_rx_byte? 0 : (ce_rx & en_rx_block)? cb_res+1 : cb_res ;
en_rx_block <= res? 0 : start_rx_block? 1 : en_rx_block ;
```

```
ten_rd_BUF <= ce_rd_BUF? en_rd_BUF : ten_rd_BUF ;
end

URXD1B_D DD1(   .in(in),                    .en_rx_byte(en_rx_byte),
                .clk(clk),                  .sr_dat(rx_dat),
                .ok_rx_byte(ok_rx_byte),    .cet(ce_rx),
                .en_sh(en_sh),
                .start_rx(start_rx_byte),
                .rx_bit(rx_bit));

CRC_BL DD2 (    .ce(ce_crc),        .crc(rx_crc),
                .clk(clk),
                .st(start_rx_block),
                .in(rx_bit) );

BUF_BMEM DD3 (  .clk(clk),          .DO(dat_BUF),
                .we(ok_rx_byte),
                .DI(rx_dat),
                .Adr_wr(cb_rx_byte),
                .Adr_rd(cb_rd_BUF));
endmodule
```

В модуле **URXD_BL_CRC** принимаемые байты «складываются» в буфер, т.е. в модуле блочной или слайсовой памяти и только после подтверждения сигналом ok_crc считываются из буфера и используются в соответствии с командой и ее параметрами. Порт rx_crc[15:0] на выходе модуля нужен только для моделирования.

1.12 Схема модуля блока памяти

```
`define m 8
`define N 256
`define n 8
module BUF_BMEM( input clk,     output reg [`m-1:0]DO,
                 input we,
                 input [`m-1:0] DI,
                 input [`n-1:0] Adr_wr,
                 input [`n-1:0] Adr_rd);

reg  [`m-1:0] MEM [`N-1:0] ; //блочная память  m x N bit.
always @ (posedge clk) begin
MEM[Adr_wr]<= we? DI : MEM[Adr_wr];//Запись в память
//if (we) MEM[Adr_wr] <= DI ;
DO <= MEM[Adr_rd];
end
endmodule
```

1.13 Память на «LUT- ах»

```
`define h 8
`define N 512
`define n 9
module BUF_SMEM (input clk,             output wire [`h-1:0]DO,
                 input we,
                 input [`h-1:0] DI,
                 input [`n-1:0] Adr_wr,
                 input [`n-1:0] Adr_rd );
```

```
reg  [`h-1:0] MEM [`N-1:0] ; //Память на LUT-ах m x N bit.
assign DO = MEM[Adr_rd];
always @ (posedge clk) begin
MEM[Adr_wr]<= we? DI : MEM[Adr_wr];//Запись в память
end
endmodule
```

1.14 Схема модуля приемника кода команды, длины блока и данных

```
module URXD_COM_LBL_ADR_CRC
            ( input in,              output reg[7:0] com,
              input clk,             output reg[7:0] lbl,
                                     output reg[15:0] adr,
                                     output wire [7:0] dat_BUF,
                                     output wire en_wr_dat,
                                     output wire re_tx,
                                     output wire en_rx_byte,
                                     output wire ok_rx_byte );
wire ce_dat_BUF, Stop_rd_BUF, en_rd_BUF ; assign re_tx = Stop_rd_BUF ;
wire [7:0]cb_rd_BUF ;
wire com_wr = (com==8'h00) | (com==8'h01) ;
wire com_rd = (com==8'h80) | (com==8'h81) ;
wire my_com = com_wr | com_rd ;
wire T_com = (cb_rd_BUF==0) ;  wire T_lbl = (cb_rd_BUF==1) ;
wire T_Hadr = (cb_rd_BUF==2) ;  wire T_Ladr = (cb_rd_BUF==3) ;
assign T_dat = (cb_rd_BUF>3) & en_rd_BUF & !(lbl==0) & com_wr;
assign en_wr_dat = T_dat & ce_dat_BUF ;

always @ (posedge clk) if (ce_dat_BUF) begin
com <= T_com? dat_BUF : com;
end

always @ (posedge clk) if (ce_dat_BUF & my_com) begin
lbl <= T_com? 0 : T_lbl? dat_BUF : T_dat? lbl-1 : lbl ;
adr[15:8] <= T_Hadr? dat_BUF : adr[15:8];
adr[ 7:0] <= T_Ladr? dat_BUF : T_dat? adr[7:0]+1 : adr[7:0] ;
end

URXD_BL_CRC DD1 (     .in(in),          .dat_BUF(dat_BUF),
                      .clk(clk),        .cb_rd_BUF(cb_rd_BUF),
                                        .ce_dat_BUF(ce_dat_BUF),
                                        .en_rd_BUF(en_rd_BUF),
                                        .Stop_rd_BUF(Stop_rd_BUF) );
endmodule
```

В модуле **URXD_COM_LBL_ADR_CRC** считываемые из буфера модуля URXD_BL_CRC байты загружаются последовательно в регистры: com[7:0], lbl[7:0], adr[15:0], а данные dat_BUF, если это необходимо, загружаются в регистры или блок памяти пользователя (макета). При записи данных в регистры или в блок памяти адрес adr[15:0] автоматически инкрементируется.

1.15 Схема моделирования двух вариантов (URXD1B_D, URXD1B_I) приёмников одного байта

```
module Sch1_Lab404(input tx_clk,          output wire txd,
```

```
                    input st,               output wire [3:0]cb_bit_tx,
                    input[7:0]tx_dat,       output wire en_rx_byte_D,
                    input rx_clk,           output wire [7:0]sr_dat_D,
                                            output wire ok_rx_byte_D,
                                            output wire start_rx_D,
                                            output wire en_sh_D,
                                            output wire ce_D,
                                            output wire cet_D,
                                            output wire rx_bit_D,

                                            output wire en_rx_byte_I,
                                            output wire [7:0]sr_dat_I,
                                            output wire ok_rx_byte_I,
                                            output wire start_rx_I,
                                            output wire en_sh_I,
                                            output wire ce_I,
                                            output wire cet_I,
                                            output wire rx_bit_I );

UTXD1B DD1      ( .clk(tx_clk),           .txd(txd),
                  .dat(tx_dat),           .cb_bit(cb_bit_tx),
                  .st(st)) ;

URXD1B_D DD2 (    .in(txd),               .en_rx_byte(en_rx_byte_D),
                  .clk(rx_clk),           .sr_dat(sr_dat_D),
                                          .ok_rx_byte(ok_rx_byte_D),
                                          .start_rx(start_rx_D),
                                          .en_sh(en_sh_D),
                                          .ce(ce_D),
                                          .cet(cet_D),
                                          .rx_bit(rx_bit_D) );

URXD1B_I DD3    ( .in(txd),               .en_rx_byte(en_rx_byte_I),
                  .clk(rx_clk),           .sr_dat(sr_dat_I),
                                          .ok_rx_byte(ok_rx_byte_I),
                                          .start_rx(start_rx_I),
                                          .en_sh(en_sh_I),
                                          .ce(ce_I),
                                          .cet(cet_I),
                                          .rx_bit(rx_bit_I) );
endmodule
```

В этой схеме используются три модуля: модуль передатчика одного байта UTXD1B и два варианта модулей приемников одного байта URXD1B_D и URXD1B_I. Выходной сигнал txd передатчика UTXD1B подается на входы in приемников, а также для контроля и на выходной порт txd схемы Sch1_Lab404. Выведены также для сравнения и все выходные сигналы обоих приемников.

Для моделирования отдельного модуля или всей схемы удобно создавать Verilog Test Fixture. При этом система проектирования автоматически создает текстовое описание всех портов модулей и схемы, а также связей между ними. Остается только дописать генераторы сигналов синхронизации и параметры входных сигналов, т.е. заполнить блок initial .

1.16 К заданию на моделирование схемы Sch1_LAB404

```
always begin tx_clk = 1'b0; #10   tx_clk = 1'b1; #10;   end //        PERIOD = 20
always begin rx_clk = 1'b0; #10.4 rx_clk = 1'b1; #10.4;  end //      PERIOD = 20.8
//always begin rx_clk = 1'b0; #9.7 rx_clk = 1'b1; #9.7;  end //       PERIOD = 19.4
initial begin
                tx_dat = 0 ;     st  = 0 ;
#100;           tx_dat = 11;     st = 1;
#1600;          tx_dat = 22;     st = 1;
#1600;          tx_dat = 33;     st = 1;
#1600;          tx_dat = 44;     st = 1;
#100 ;          tx_dat = 55;     st = 0;
end
```

В этом примере по старту моделирования tx_dat=0 и st=0. Далее через 100ns: tx_dat=11, st=1, затем через каждые 1600ns (длительность одного байта при VEL=6250000) tx_dat принимает новое значение, а st в течение 4800ns удерживается в 1, что обеспечивает передаче данных трех байт. В Process Properties логического симулятора Simulate Behavioral Model надо установить подходящее (Simulation Run Time) время моделирования, например 6000ns.

1.17 Схема моделирования передатчика и приемника блока байт с CRC-16

```
module Sch2_Lab404(input tx_clk,          output wire txd_bl,
                   input st,              output wire en_tx_bl,
                   input[7:0]tx_dat,      output wire [7:0]cb_byte_tx,
                   input[7:0]lbl,         output wire ce_stop_tx,
                   input rx_clk,          output wire [15:0]crc_tx,
                                          output wire [7:0] tx_dat_crc,
                                          output wire [7:0]dat_BUF,
                                          output wire [7:0]cb_rd_BUF,
                                          output wire ce_dat_BUF,
                                          output wire en_rd_BUF,
                                          output wire Stop_rd_BUF,
                                          output wire [15:0]crc_rx );

UTXD_BL_CRC DD1 (   .st(st),          .txd_bl(txd_bl),
                    .lbl(lbl),        .en_tx_bl(en_tx_bl),
                    .dat(tx_dat),     .cb_byte(cb_byte_tx),
                    .clk(tx_clk),     .ce_stop(ce_stop_tx),
                                      .crc(crc_tx),
                                      .dat_crc(tx_dat_crc)) ;

URXD_BL_CRC DD2 (   .in(txd_bl),      .dat_BUF(dat_BUF),
                    .clk(rx_clk),     .cb_rd_BUF(cb_rd_BUF),
                                      .ce_dat_BUF(ce_dat_BUF),
                                      .en_rd_BUF(en_rd_BUF),
                                      .Stop_rd_BUF(Stop_rd_BUF),
                                      .rx_crc(crc_rx));
endmodule
```

В этой схеме используются только два модуля: модуль передатчика блока байт с CRC кодом UTXD_BL_CRC и модуль приемника блока байт с CRC кодом URXD_BL_CRC. Выходной сигнал txd_bl передатчика UTXD_BL_CRC подается на вход

in приемника, а также для контроля и на выходной порт txd_bl схемы Sch2_Lab404. Выведены также и для сравнения и все выходные сигналы обоих модулей.

1.18 К заданию на моделирование схемы Sch2_LAB404

```
always begin tx_clk = 1'b0; #10   tx_clk = 1'b1; #10;    end //        PERIOD = 20
always begin rx_clk = 1'b0; #9.7 rx_clk = 1'b1; #9.7;  end //        PERIOD = 19.4

initial begin
                tx_dat = 0 ;     st  = 0 ;        lbl = 0 ;
#100;           tx_dat = 11;     st = 1;          lbl = 3 ;
#20;            tx_dat = 11;     st = 0;          lbl = 3 ;
#1600;          tx_dat = 22;     st = 0;          lbl = 3 ;
#1600;          tx_dat = 33;     st = 0;          lbl = 3 ;
#1600;          tx_dat = 44;     st = 0;          lbl = 3 ;
#1600;          tx_dat = 55;     st = 0;          lbl = 3 ;
#100 ;          tx_dat = 55;     st = 0;          lbl = 3 ;
end
```

В initial блоке этого задания длительность импульса st запуска передатчика равна одному периоду сигнала синхронизации tx_clk блока UTXD_BL_CRC. Для этой схемы необходимо установить большее время моделирования, например, 10000ns.

Таблица 1

№	VEL	Объём памяти SIZE_MEM	Базовый адрес блока памяти BASE_ADR_MEM	Базовый адрес регистров BASE_ADR_REG
1	115200	BMEM 8x4096	16’h1000	16’h1000
2	57600	BMEM 8x2048	16’h2000	16’h2000
3	19200	SMEM 8x128	16’h3000	16’h3000
4	9600	SMEM 8x64	16’h4000	16’h4000
5	4800	BMEM 8x512	16’h5000	16’h5000
6	38400	SMEM 8x32	16’h6000	16’h6000
7	9600	BMEM 8x1024	16’h7000	16’h7000
8	19200	SMEM 8x16	16’h8000	16’h8000
9	115200	BMEM 8x8196	16’h9000	16’h9000
10	57600	SMEM 8x256	16’hA000	16’hA000
11	115200	BMEM 8x512	16’hB000	16’hB000
12	4800	SMEM 8x32	16’hC000	16’hC000
13	9600	SMEM 8x64	16’hD000	16’hD000
14	115200	BMEM 8x512	16’hE000	16’hE000
15	19200	BMEM 8x4096	16’hF000	16’hF000
16	4800	BMEM 8x256	16’h0000	16’h0000
17	19200	SMEM 8x16	16’h8000	16’h8000
18	57600	SMEM 8x256	16’hA000	16’hA000
19	9600	SMEM 8x64	16’h4000	16’h4000
20	115200	BMEM 8x4096	16’h1000	16’h1000

2.Задание к допуску

2.1.1 Переписать в тетрадь схему модуля UTXD1B передатчика одного байта.
2.1.2 Начертить в тетради эскизы временных диаграмм сигналов передатчика одного байта (рис.3).

2.2 Переписать в тетрадь схему модуля генератора CRC-16.
2.3 Переписать в тетрадь схему модуля UTXD_BL_CRC передатчика блока байт с CRC-16.
2.4 Переписать в тетрадь схему модуля передатчика TXD_RET_BL ответа на команду.
2.5 Переписать в тетрадь схему модуля URXD1B_D точечного приемника одного байта.
2.6 Начертить в тетради временные диаграммы сигналов модуля URXD1B_D.
2.7 Переписать в тетрадь схему модуля URXD1B_I интегрирующего приемника одного байта.
2.8 Начертить в тетради временные диаграммы сигналов модуля URXD1B_I.
2.9 Переписать в тетрадь схему модуля URXD_BL_CRC приемника блока байт с CRC-16.
2.10 Переписать в тетрадь схему Sch1_Lab404.
2.11 Переписать в тетрадь схему Sch2_Lab404.

3. Задание к выполнению работы

Создать проект с именем Lab404 для ПЛИС, используемой в макете NEXYS или NEXYS-2. Если на ПК установлен Modelsim и в окне Properties проекта в строке Simulator - выбрать Modelsim и в дальнейшем моделирование рекомендуется проводить для Simulate Post Translate Verilog Model. Если на ПК нет Modelsim, то выбрать ISim (VHDL/Verilog), а во всех модулях, которые будут моделироваться, все регистры проинициализировать, например, приравнять к 0.

3.1 Создать модуль UTXD1B передатчика одного байта. Провести моделирование его работы при VEL= 6250000. Для какого либо значения dat[7:0] зарисовать временные диаграммы сигналов модуля.

3.2.1 Создать модули URXD1B_D и URXD1B_I приемников одного байта.

3.2.2 Создать модуль схемы Sch1_Lab404.

3.2.3 Создать модуль tf_Sch1 (Verilog Test Fixture) для схемы Sch1_Lab404.

3.2.4 Провести моделирование приемников одного байта URXD1B_D и URXD1B_I в схеме Sch1_Lab404 при различных значениях периода сигнала синхронизации rx_clk приемников. Определить допустимый диапазон периода rx_clk для каждого приемника. Зарисовать эскизы временных диаграмм сигналов.

3.3.1 Создать модули UTXD_BL_CRC и URXD_BL_CRC и модуль схемы Sch2_Lab404.

3.3.2 Создать модуль tf_Sch2 (Verilog Test Fixture) для схемы Sch2_Lab404.

3.3.3 При допустимом значении периода rx_clk провести моделирование модулей UTXD_BL_CRC и URXD_BL_CRC в схеме Sch2_Lab404. Зарисовать эскизы временных диаграмм сигналов.

4. Задание к сдаче работы

4.1 Составить схему модуля заданного в Таблице 1 варианта памяти.
4.2 Составить схему содержащую 4 16-и битных регистра: Test0_reg[15:0], Test1_reg[15:0], Test2_reg[15:0], Test3_reg[15:0], состояния которых при помощи модуля **DISPLAY** могут отображаться на 7-ми сегментном индикаторе макета в зависимости от положения переключателей SWT[1:0]. Светодиоды макета подключить к одному из регистров. Положение точки индикатора должно определяться SWT [1:0].
4.3 Составить схему записи и чтения данных в модуль памяти и в регистры Test0_reg – Test3_reg при помощи программы ComChange по заданному в Таблице 1 базовому адресу. По одному из адресов выводить состояние SWT[7:0] и BTN[3:0].
4.4 Создать конфигурацию составленной схемы загрузить в макет. Отладить при необходимости работу схемы макета. Продемонстрировать работу макета.

5. Приложение

5.1 Модуль индикатора (дисплея)

```
module DISPLAY    (     input clk,              output wire [3:0] AN,
                        input [15:0]DAT,        output wire[7:0] SEG,
                        input [1:0]PTR,         output wire ce1ms);

wire [3:0]Dig; wire [1:0]Adr_dig ;

Gen_4An      DD1(  .clk(clk),      .q(Adr_dig),
                   .ce(ce1ms),     .An(AN));

MUX16_4      DD2 (  .DI(DAT),      .DO(Dig),
                    .s(Adr_dig));

D7seg        DD3 (  .dig(Dig),              .seg(SEG[6:0]));

Gen_P        DD4 (  .adr_An(Adr_dig),     .seg_P(SEG[7]),
                    .ptr(PTR) );

Gen_ce1ms  DD5 (.clk(clk), .ce_1ms(ce1ms));
endmodule
```

5.2 Модуль генератора сигналов анодов индикатора

```
module Gen_4An  (  input clk,       output reg [1:0] q = 0,//Счетчик номера анода
                   input ce,        output wire [3:0] An );
assign    An = (q==0)? 4'b1110 ://включение цифры 0 (младшей)
               (q==1)? 4'b1101 ://включение цифры 1
               (q==2)? 4'b1011 ://включение цифры 2
                       4'b0111 ;//включение цифры 3 (старшей)
always @ (posedge clk) if (ce) begin
q <= q+1 ;
end
endmodule
```

5.3 Модуль мультиплексора динамической индикации

```
module MUX16_4(   input [15:0] DI, output [3:0] DO,
                  input [1:0] s);

assign  DO =   (s==0)? DI[3:0]:
               (s==1)? DI[7:4]:
               (s==2)? DI[11:8]:
                       DI[15:12];
endmodule
```

5.4 модуль дешифратора семи сегментного индикатора

```
module D7seg(input [3:0] dig,         output wire [6:0] seg);
//                                gfedcba
assign seg =   (dig==0)?  7'b1000000 ://0      a
               (dig==1)?  7'b1111001 ://1    f     b
               (dig==2)?  7'b0100100 ://2      g
               (dig==3)?  7'b0110000 ://3    e     c
               (dig==4)?  7'b0011001 ://4      d     h
               (dig==5)?  7'b0010010 ://5
               (dig==6)?  7'b0000010 ://6
```

```
                (dig==7)?  7'b1111000 ://7
                (dig==8)?  7'b0000000 ://8
                (dig==9)?  7'b0010000 ://9
                (dig==10)? 7'b0001000 ://A
                (dig==11)? 7'b0000011 ://b
                (dig==12)? 7'b1000110 ://C
                (dig==13)? 7'b0100001 ://d
                (dig==14)? 7'b0000110 ://E
                           7'b0001110 ;//F
endmodule
```

5.5 Модуль сигнала ce_1ms

```
module Gen_ce1ms(  input clk,//Сигнал синхронизации
                                    output wire ce_1ms); //1 милисекунда
parameter Fclk  =50000000 ;  //Частота генератора синхронизации
parameter F1kHz =1000 ;      //Частота 1 кГц
reg[16:0]cb_ms = 0  ;        //Счетчик миллисекунд
assign ce_1ms = (cb_ms==1) ;// 1 милисекунда

always @(posedge clk) begin
cb_ms <= ce_1ms? (Fclk/F1kHz) : cb_ms-1 ;      //Счет миллисекунд
end
endmodule
```

5.6 Модуль генератора точки

```
module Gen_P (      input [1:0] ptr,          output wire seg_P,
                    input [1:0] s );
assign seg_P = !(ptr==s) ;
endmodule
```

5.7 Список выводов ПЛИС макета Nexys2

```
NET "an<0>"  LOC = "F17"  ; #AN0
NET "an<1>"  LOC = "H17"  ; #AN1
NET "an<2>"  LOC = "C18"  ; #AN2
NET "an<3>"  LOC = "F15"  ; #AN3

NET "BTN<0>"  LOC = "B18"  ; #BTN3
NET "BTN<1>"  LOC = "D18"  ; #BTN2
NET "BTN<2>"  LOC = "E18"  ; #BTN1
NET "BTN<3>"  LOC = "H13"  ; #BTN0

NET "F50MHz"  LOC = "B8"  ; #F50MHz

NET "ld<1>"  LOC = "J15"  ; #LD1
NET "ld<2>"  LOC = "K15" ; #LD2
NET "ld<3>"  LOC = "K14" ; #LD3
NET "ld<4>"  LOC = "E17"  ; #LD4
NET "ld<5>"  LOC = "P15"  ; #LD5
NET "ld<6>"  LOC = "F4"    ; #LD6
NET "ld<7>"  LOC = "R4"    ; #LD7
NET "RXD"   LOC = "U6"    ; #TXD U6

NET "seg<0>"  LOC = "L18"  ; #CA
NET "seg<1>"  LOC = "F18"  ; #CB
NET "seg<2>"  LOC = "D17"  ; #CC
```

```
NET "seg<3>"  LOC = "D16"  ; #CD
NET "seg<4>"  LOC = "G14"  ; #CE
NET "seg<5>"  LOC = "J17"    ; #CF
NET "seg<6>"  LOC = "H14"  ; #CG
NET "seg<7>"  LOC = "C17"  ; #CP

NET "SWT<0>"  LOC = "G18"  ; #SWT0
NET "SWT<1>"  LOC = "H18"  ; #SWT1
NET "SWT<2>"  LOC = "K18"  ; #SWT2
NET "SWT<3>"  LOC = "K17"  ; #SWT3
NET "SWT<4>"  LOC = "L14"  ; #SWT4
NET "SWT<5>"  LOC = "L13"  ; #SWT5
NET "SWT<6>"  LOC = "N17"  ; #SWT6
NET "SWT<7>"  LOC = "R17"  ; #SWT7
NET "TXD"  LOC = "P9"  ; #TXD P9

#NET "JA1"  LOC = "L15"  ; #Pin1
#NET "JA2"  LOC = "K12"  ;#Pin2
#NET "JA3"  LOC = "L17"   ;#Pin3
#NET "JA4"  LOC = "M15"  ; #Pin4
#NET "JA7"   LOC = "K13"  ;#Pin7
#NET "JA8"   LOC = "L16"  ;#Pin8
#NET "JA9"  LOC = "M14"   ;#Pin9
#NET "JA10" LOC = "M16"  ;#Pin10

#NET " JB1"  LOC = "M13"  ;# Pin1
#NET " JB2"  LOC = "R18"  ;# Pin2
#NET " JB3"  LOC = "R15"  ;# Pin3
#NET " JB4"  LOC = "T17"  ;# Pin4
#NET "JB7"  LOC = "P17"    ; #Pin7
#NET "JB8"   LOC = "R16"  ;#Pin8
#NET "JB9"  LOC = "T18"    ;#Pin9
#NET " JB10" LOC = "U18"  ;#Pin10

#NET "JC1" LOC = "G15"   ;#Pin1
#NET "JC2" LOC = "J16"    ;#Pin2
#NET "JC3"  LOC = "G13"   ;#Pin3
#NET "JC4"  LOC = "H16"   ;#Pin4
#NET "JC7" LOC  = "H15"   ;#Pin7
#NET "JC8" LOC  = "F14"    ;#Pin8
#NET "JC9" LOC  = "G16"    ;#Pin9
#NET "JC10" LOC= "J12"     ;#Pin10

#NET "JD1"  LOC = "J13"    ;#Pin1
#NET "JD2"  LOC = "M18"  ;#Pin2
#NET "JD3"  LOC = "N18"   ;#Pin3
#NET "JD4"  LOC = "P18"    ;#Pin4
#NET "JD7"   LOC = "K14"   ;#LD3
#NET "JD8"   LOC = "K15"   ;#LD3
#NET "JD9"  LOC  = "J15"     ;#LD3
#NET "JD10" LOC = "J14"     ;#LD3
```