Лабораторная работа 404_CRC

1.1 Интерфейс RS-232 (COM-порт)

Интерфейс RS-232, официально называемый "EIA/TIA–232–E", но более известный как интерфейс "COM-порта" (на логическом уровне Universal Asynchronous Receiver-Transmitter (UART)) ранее был одним из самых распространенных интерфейсов в компьютерной технике. Он и до сих пор встречается в системных блоках, несмотря на появление более скоростных и "интеллектуальных" интерфейсов, таких как USB и FireWire. У разработчиков аппаратуры UART интерфейс очень востребован благодаря простоте реализации протокола. Поэтому фирма FTDI разработала и выпускает серию микросхем переходников интерфейсов USB - UART и драйверов к ним.

Физический интерфейс RS-232 на материнских платах системных блоков персональных компьютеров (ПК) реализуется разъемом типа DB-9M.

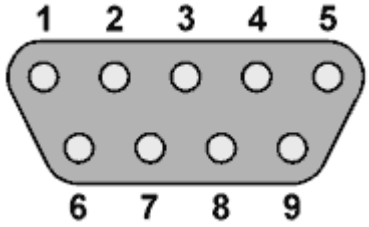

Рис.1 9-контактная вилка типа DB-9M COM порта компьютера

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

В данной работе изучается наиболее распространенный, простейший вариант асинхронного протокола 8 бит данных без дополнительного бита контроля четности. Каждый, передаваемый байт (8 бит), дополняется нулевым старт битом и заканчивается единичным стоп битом. Длительность старт бита равна длительности одного бита, а длительность стоп бита может составлять одну, полторы или две длительности бита. Данные передаются младшими битами вперед.

Временные диаграммы сигналов передатчика физического (RS232) и логического (UART) интерфейса COM порта приведены на рис.2. В паузе между передачами UTXD=1, а напряжение U_{TXD} на выводе TXD (DB-9M.3) примерно равно -10 В. Логическому нулю UTXD соответствует напряжение U_{TXD} = +10 В.

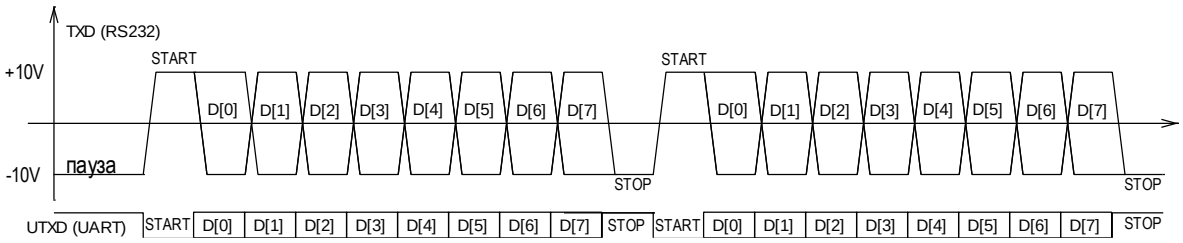

Рис.2 Сигналы физического (RS232) и логического (UART) интерфейса COM порта

Скорость обмена данными (BAUDRATE) задается в битах в секунду и выбирается из ряда стандартных значений (300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 430800,921600), но может быть и нестандартной, если поддерживаются обеими сторонами.

На макете NEXYS-2, на котором рекомендуется выполнение работы, на разъеме DB-9 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.

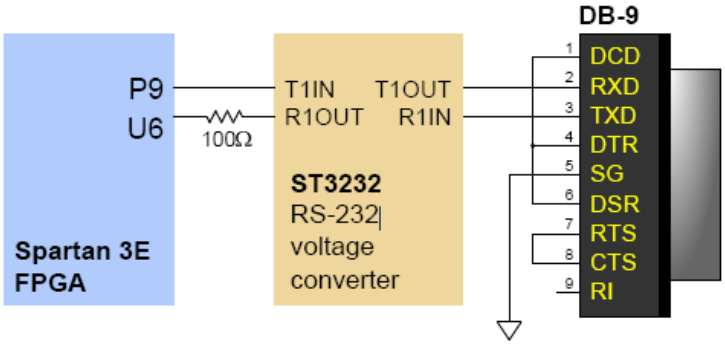

Рис.3 Схема соединений COM порта на макете NEXYS-2

Для защиты передаваемого пакета байт от ошибок к передаваемому пакету добавляется контрольный код CRC (Cyclic Redundancy Code) - 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 полином 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-16 необходим 16-ми разрядный регистр сдвига и 3 элемента XOR2 (исключающее ИЛИ) в цепях обратной связи. Схема вычислителя CRC-16 для порождающего полинома ($x^{16}+x^{15}+x^2+x^0$) приведена на рис.4, а вариант представления этой схемы для сдвига данных младшими битами вперед на Verilog-е в 1.2.

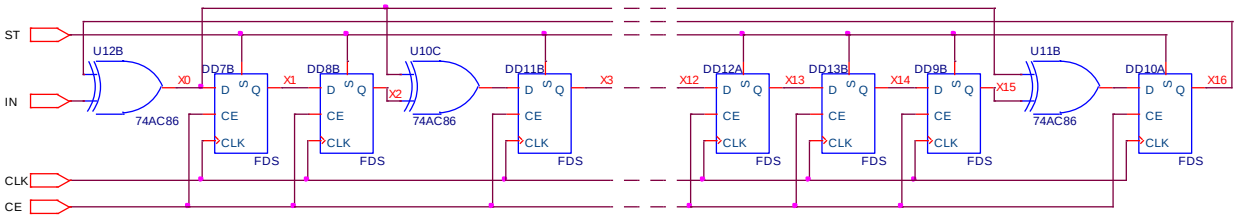

Рис.4 Схема вычислителя кода CRC-16 для порождающего полинома $x^{16}+x^{15}+x^2+x^0$

Схема вычислителя CRC кода не зависит от направления сдвига данных - младшими или старшими битами вперед. От направления сдвига зависит только интерпретация разрядов регистра crc[15:0].

При сдвиге данных младшими битами вперед x16=crc[0], x15=crc[1], x14=crc[2], … x2=crc[14], x1=crc[15], x0=IN^crc[0].

При сдвиге данных старшими битами вперед x16=crc[15], x15=crc[14], x14=crc[13], … x2=crc[1], x1=crc[0], x0=IN^crc[15].

1.2 Схема вычислителя кода CRC-16 для сдвига данных младшими битами вперед

```
module CRC16_BL( input ce,        output reg [15:0] crc=0,//X[1:16]=1  0100 0000 0000 0010
                 input clk,       output wire ok_crc16,
                 input st,
                 input in);
assign ok_crc16 = (crc==16'h0000);
wire x0 = crc[0] ^ in ;
always @ (posedge clk) begin
crc <= st? 16'hFFFF : (x0 & ce)? (((crc^16'h4002)>>1) | 1<<15): ce? (crc>>1) : crc ;
end
endmodule
```

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 обязательно кратным 8, т.е. байты COM порта можно передавать и с битом контроля четности. Кроме того, и производящий полином может быть произвольным это и понятно, потому что добавление tx_CRC кода к передаваемому пакету эквивалентно по существу умножению передаваемого полинома на производящий полином, а вычисление CRC приемником эквивалентно делению умноженного полинома на такой же производящий полином поэтому очевидно, что остаток от деления т.е. rx_CRC=0. Тем не менее используются только некоторые производящие полиномы, которые дают максимальную защиту от ошибок. Используемые производящие полиномы CRC16:

- $x^{16}+x^{15}+x^2+1$ – CRC-16-IBM (Bisync, Modbus, USB, ANSI X3.28, многие другие; также известен как *CRC-16* и *CRC-16-ANSI*)),
- $x^{16}+x^{12}+x^5+1$ - CRC-16-CCITT (X.25, HDLC, XMODEM, Bluetooth, SD и др.),
- $x^{16}+x^{15}+x^{11}+x^9+x^8+x^7+ x^5+x^4+x^2+x+1$ - (CSCI DIF),
- $x^{16}+x^{13}+x^{12}+x^{11}+x^{10}+x^8+ x^6+x^5+x^2+x+1$ - (DNP, IEC 870, M-Bus)

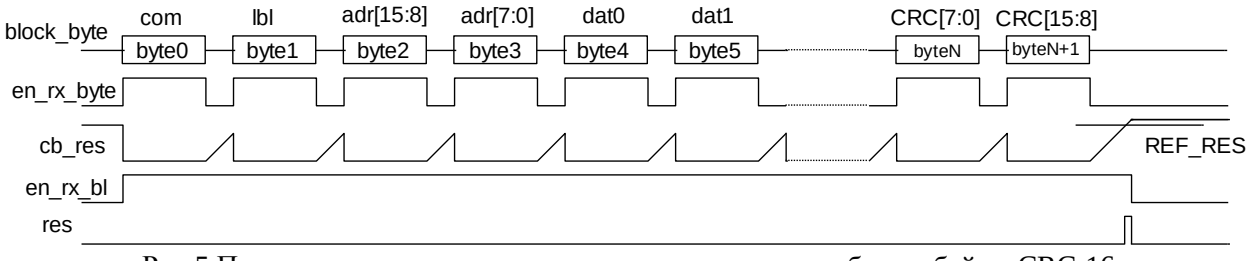

Рис.5 Пример временных диаграмм передачи и приема блока байт с CRC-16

В этом примере (рис.5) конец блока байт определяется по длительности паузы между байтами, т.к. предполагается, что на приемном конце число байт пакета неизвестно.

В данной работе предлагается следующее функциональное назначение байт передаваемого пакета.

Первый байт - *com* это код команды:
com=8’h00 – загрузить байт в регистр,
com=8’h80 – прочитать регистр,
com=8’h01 – загрузить байт в память,
com=8’h81 – прочитать память,

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 *dat*0, *dat*1,… - данные, предназначенные для записи. Для команд чтения 8’h80 и 8’h81 поля данных нет.

1.3 Схема модуля имитатора передатчика блока байт программы ComChange

```
`include "CONST.v"
module Sim_UTXD_CRC(
   input clk,            output wire UTXD,
   input [7:0] com,      output wire [7:0]tx_dat,
   input [7:0] lbl,      output reg [15:0]CRC=0,
   input [15:0] adr,     output reg en_tx=0,
   input st,             output reg [7:0]cb_byte=0);
parameter X=8'h11 ; //Для генератора данных
wire com_wr=(com==8'h00) | (com==8'h01); //Команда записи
//------Передатчик блока байт-----------------------------------
reg [7:0]dat=0 ;                    //Данные
reg [15:0] cb_tact ;                //Счетчик длительности такта (бита)
wire ce_tact = (cb_tact==`Nt) ;     //Tce_tact=1/BR
reg [3:0] cb_bit=0 ;                //Счетчик бит байта
reg [7:0] sr_dat=0 ;                //Регистр сдвига бит байта
wire T_start = ((cb_bit==0) & en_tx) ;          //Интервал старт бита
wire ce_sh=(cb_bit<9) & (cb_bit>0) & ce_tact;   //Разрешение сдвига бит байта
assign ce_stop = (cb_bit==9) & ce_tact ;        //Импульс конца байта
wire rep_st = st | (ce_stop & en_tx);           //Импульсы запуска передачи байт
assign UTXD = T_start? 0 : en_tx?  sr_dat[0] : 1 ;  //UTXD блока байт

wire [7:0]N_byte = com_wr? lbl+4 : 4 ;
wire T_com = (cb_byte==0) ;                 // Интервал команды
wire T_lbl = (cb_byte==1) ;                 // Интервал длинны блока
wire T_Hb_adr = (cb_byte==2) ;              // Интервал старшего байта адреса
wire T_Lb_adr = (cb_byte==3) ;              // Интервал младшего байта адреса
wire T_Lb_CRC = (cb_byte==N_byte);          // Интервал младшего байта CRC
wire T_Hb_CRC = (cb_byte==N_byte+1);        // Интервал старшего байта CRC
wire ce_dat = (ce_stop & cb_byte>=4) & !(T_Lb_CRC | T_Hb_CRC) & com_wr;
//--------Функциональное назначение байт
assign tx_dat=T_com? com :                  //Команда
              T_Lb_CRC? CRC[ 7:0] :         //Младший байт CRC
              T_Hb_CRC? CRC[15:8] :         //Старший байт CRC
              T_lbl? lbl :                  //Число байт данных
              T_Hb_adr? adr[15:8] :         //Старший байт адреса
```

```
                T_Lb_adr? adr[ 7:0] :       //Младший байт адреса :
                          dat ;             //Данные

wire ce_crc = ce_sh & !(T_Lb_CRC  |  T_Hb_CRC);
wire x0 = CRC[0] ^ sr_dat[0] ;

always @ (posedge clk) begin
cb_tact <= (st | ce_tact)? 1 : cb_tact+1;
cb_byte <= st? 0  : ce_stop? cb_byte+1 : cb_byte    ;
cb_bit <= rep_st? 0 : (ce_tact & en_tx)? cb_bit+1 :cb_bit ;
sr_dat <= (T_start & ce_tact)? tx_dat : ce_sh? sr_dat>>1 | 1<<7 : sr_dat ;
en_tx <= st? 1 : ((cb_byte==N_byte+1) & ce_stop)? 0 : en_tx ;
dat <= st? X : ce_dat? dat+X : dat ;
CRC <= st? `INIT_CRC : (x0 & ce_crc)? (((CRC^`XCRC16)>>1) | 1<<15): ce_crc? (CRC>>1) : CRC ;//16'h4002
end
endmodule
```

1.4 Модуль **CONST**

```
`define  Fclk        50000000     //50 MHz
`define  BR          115200       //BAUDRATE - скорость 115.2 kBOD
`define  Nt          `Fclk/`BR    //50000000/115200=434
`define  MY_ADR      16'h5800     //Базовый адрес
`define  XCRC16      16'h4002     //CRC (x^16 + x^15 + x^2 + 1)
`define  INIT_CRC    16'hFFFF     //Инициализация регистра CRC
```

1.5 Схема модуля приемника блока байта с кодом CRC-16

```
`include "CONST.v"
module URXD_CRC_BL(
        input URXD,             output reg[7:0] com=0,          //Команда
        input clk,              output reg[7:0] lbl=0,          //Длина блока данных
                                output reg[15:0] wr_adr=0,      //Базовый адрес данных
                                output reg[7:0] rx_dat=0,       //Принятый байт
                                output reg en_rx_byte=0,        //Интервал приема байта
                                output reg en_rx_bl=0,          //Интервал приема блока байт
                                output reg[7:0] cb_byte=0,      //Счетчик принятых байт
                                output wire ok_rx_byte,         //Подтверждение приема байта
                                output wire ce_wr_dat,          //Разрешение записи данных
                                output wire res,                //Сброс в конце паузы
                                output reg [15:0] CRC=0,        //Контрольный код CRC
                                output wire ok_rx_bl );         //Конец приема блока байт

parameter REF_PAUESE =10 ; //Допустимая пауза между байтами
//--------Приемник строки байт-----------------------------
reg [11:0] cb_tact=0 ;                  //Счетчик длительности такта (бита)
wire ce_tact = (cb_tact==`Nt) ;         //Tce_tact=1/BAUDRATE
wire ce_bit = (cb_tact==(`Nt/2));       //Середина такта
reg [3:0]cb_bit=0 ;                     //Счетчик бит в кадре UART
reg [4:0]cb_res=0 ;                     //Счетчик паузы
reg RXD=0, tRXD=0 ; //
assign res = (cb_res>=REF_PAUESE) & ce_tact & en_rx_bl ;
wire com_wr = (com==8'h00) | (com==8'h01) ;
wire com_rd = (com==8'h80) | (com==8'h81) ;
```

```
//---Адреса регистров-------------------------
wire T_com = (cb_byte==0) ;          //Команда
wire T_lbl = (cb_byte==1) ;          //Длина блока
wire T_Hb_adr = (cb_byte==2) ;       //Старший байт адреса
wire T_Lb_adr = (cb_byte==3) ;       //Младший байт адреса
wire T_dat = (cb_byte>3) ;           //Интервал данных

wire dRXD = !RXD & tRXD ;            //"Спады" входного сигнала RXD
assign ok_rx_byte = (ce_bit & (cb_bit==9) & en_rx_byte & tRXD); //Успешный прием байта
wire st_rx_byte = dRXD & !en_rx_byte ;    //Старт приема очередного байта
wire st_rx_bl = dRXD & !en_rx_bl ;        //Старт приема блока байт
assign ok_rx_bl = res &  (CRC==0); /*Успешный прием блока байт (по паузе в 10 тактов между байтами)*/
wire T_sh = (cb_bit<9) & (cb_bit>0);      //Интервал данных байта
wire x0 = CRC[0] ^ RXD ;
wire ce_crc = ce_bit & T_sh ;
reg[7:0] cb_wr_byte=0 ;       //
assign ce_wr_dat = T_dat & ok_rx_byte & com_wr & (cb_wr_byte<lbl);

always @ (posedge clk) begin
RXD <= URXD ; tRXD <= RXD ;
cb_tact <= ((dRXD & !en_rx_byte) | ce_tact)? 1 : cb_tact+1;
cb_bit <= (st_rx_byte | ((cb_bit==9) & ce_tact))? 0 : (ce_tact & en_rx_byte)? cb_bit+1 : cb_bit ;
en_rx_byte <= (ce_bit & !RXD)? 1 : ((cb_bit==9) & ce_bit)? 0 : en_rx_byte ;
rx_dat <= (ce_bit & T_sh)? rx_dat>>1 | RXD<<7 : rx_dat ; //
cb_byte <= ok_rx_bl? 0 : ok_rx_byte? cb_byte+1 : cb_byte ;
cb_res <= en_rx_byte? 0 : (ce_tact)? cb_res+1 : cb_res ;// & en_rx_bl
en_rx_bl <= st_rx_byte? 1 : ok_rx_bl? 0 : en_rx_bl ;
CRC <= st_rx_bl? `INIT_CRC : (x0 & ce_crc)? (((CRC^`XCRC16)>>1) | 1<<15): ce_crc? (CRC>>1) : CRC ;//16'h4002
end
//---Загрузка регистров: команды и длиннны блока и адреса
always @ (posedge clk) begin
com <= (T_com & ok_rx_byte)? rx_dat : com ;
lbl <= (T_lbl & ok_rx_byte)? rx_dat : lbl ;
cb_wr_byte <= (T_com & ok_rx_byte)? 0 :
                  (T_lbl & com_rd & ok_rx_byte)? rx_dat :
                   ce_wr_dat? cb_wr_byte+1 :
                   cb_wr_byte ;
wr_adr[15:8] <=(T_Hb_adr & ok_rx_byte)? rx_dat :
                  (T_dat & ce_wr_dat & (wr_adr[7:0]==8'hFF))? wr_adr[15:8]+1 :
                  wr_adr[15:8] ;
wr_adr[ 7:0] <=(T_Lb_adr & ok_rx_byte)? rx_dat :
                  (T_dat & ce_wr_dat)? wr_adr[7:0]+1 :
                   wr_adr[7:0] ;
end
endmodule
```

Приемник байта построен по классическому алгоритму, т.е. решение о начале приема байта принимается с задержкой в половину длительности такта после первого спада входного сигнала RXD при RXD=0, а решение об успешном окончании приема байта

принимается на девятом такте при RXD=1. Значение каждого бита данных принимается в середине такта на интервале данных.

Сигнал ce_tact соответствует границам тактов, а сигнал ce_bit – серединам тактов (рис.6). Периоды сигналов ce_tact и ce_bit должны быть равны ожидаемой длительности бита принимаемых данных. Генератор сигналов ce_tact и ce_bit фазируется первым спадом сигнала RXD. Решение о конце приема байта принимается на девятом такте по сигналу ce_bit. Данные D[0],…D[7] задвигаются в регистр сдвига rx_dat[7:0] по сигналу ce_bit на интервале данных T_sh. Сигнал подтверждения правильности приема байта ok_rx_byte формируется на девятом такте при условии, что RXD=1.

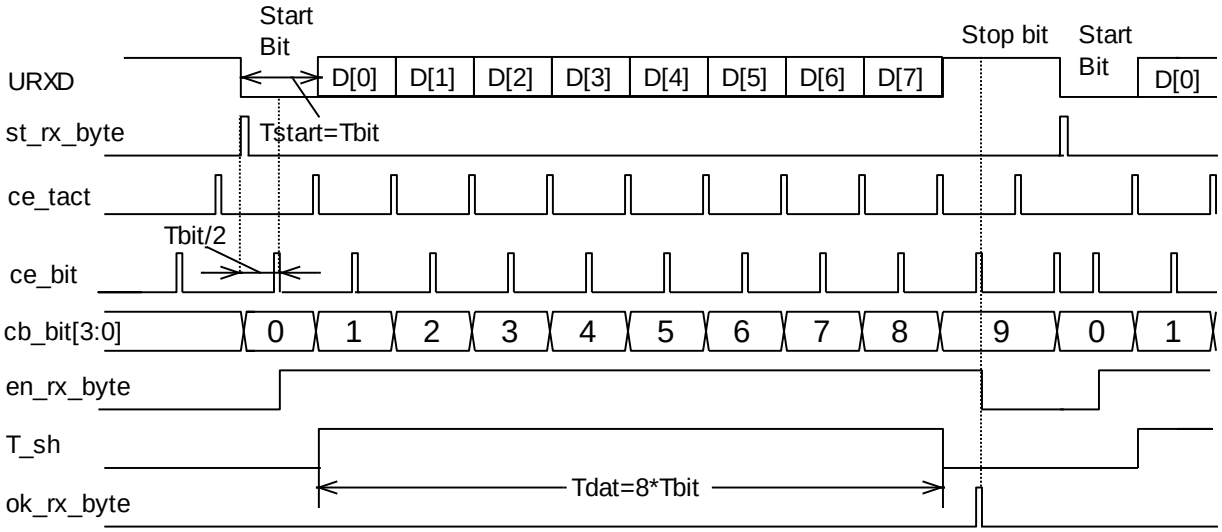

Рис.6 Временные диаграммы приема одного байта

В модуле URXD_CRC_BL число разрядов счетчика такта cb_tact также как и модуле передатчиков Sim_UTXD_CRC и RET_TXD_CRC_BL должно соответствовать заданной скорости VEL. Например, при BAUDRATE=115200 бод (Nt=50000000/115200=434) счетчик должен иметь 9 разрядов, а при скорости 1200 бод -16 разрядов (Nt=50000000/1200=41667).

Разрешение на запись данных ce_wr_dat дается только для команд записи (00 и 01) после приема 4-го байта на интервале данных. Инкремент адреса записи wr_adr разрешается на интервале данных также только для команд записи.

В регистр rx_lbl[7:0] при командах чтения загружается значение lbl, полученное от ComChange, а при командах записи число принятых байт данных.

1.6 Схема модуля передатчика ответа блока байт программе ComChange

```
`include "CONST.v"
module RET_TXD_CRC_BL(
        input clk,              output wire UTXD,           //Последовательные данные
        input st,               output reg en_tx=0,         //Интервал передачи блока байт
        input [7:0] com,        output wire [7:0] tx_dat,   //Передаваемые байты
        input [7:0] lbl,        output reg [15:0] CRC=15'hFFFF, //Контрольный CRC код
        input [15:0] adr,       output reg [7:0] cb_byte=0, //Счетчик передаваемых байт
        input [7:0] dat,        output reg[15:0] rd_adr=0); //Адрес передаваемых байт данных

wire com_rd=(com==8'h80) | (com==8'h81); //Команда чтения
wire com_wr=(com==8'h00) | (com==8'h01); //Команда записи
//------Передатчик блока байт-----------------------------------
reg [15:0] cb_tact ;                    //Счетчик длительности такта (бита)
wire ce_tact = (cb_tact==`Nt) ;         //Tce_tact=1/BAUDRATE
```

```
reg [3:0] cb_bit=0 ;                    //Счетчик бит байта
reg [7:0] sr_dat=0 ;                    //Регистр сдвига бит байта
wireT_start = ((cb_bit==0) & en_tx) ;           //Интервал старт бита
wire ce_sh=(cb_bit<9) & (cb_bit>0) & ce_tact;   //Разрешение сдвига бит байта
assign ce_stop = (cb_bit==9) & ce_tact ;        //Импульс конца байта
wire rep_st = st | (ce_stop & en_tx);           //Импульсы запуска передачи байт
assign UTXD = T_start? 0 :en_tx?  sr_dat[0] : 1 ;  //UTXD блока байт

wire [7:0]N_byte =    com_rd? lbl+4 :       //com, lbl, Hb_adr, Lb_adr, dat0,dat1,….
                      com_wr? 4 :           //com, lbl, adr[15:8], adr[7:0]
                              1 ;           //com («чужая» команда)
wire T_com = (cb_byte==0) ;                 // Интервал команды
wire T_lbl = (cb_byte==1) ;                 // Интервал длинны блока
wire T_Hb_adr = (cb_byte==2) ;              // Интервал старшего байта адреса
wire T_Lb_adr = (cb_byte==3) ;              // Интервал младшего байта адреса
wire T_Lb_CRC = (cb_byte==N_byte);          // Интервал младшего байта CRC
wire T_Hb_CRC = (cb_byte==N_byte+1);        // Интервал старшего байта CRC
wire ce_adr = (ce_stop & cb_byte>3) & !(T_Lb_CRC | T_Hb_CRC) ;

//--------Функциональное назначение байт ответа
assign tx_dat=T_com? com :              //Команда
              T_Lb_CRC? CRC[ 7:0] :     //Младший байт CRC
              T_Hb_CRC? CRC[15:8] :     //Старший байт CRC
              T_lbl? lbl :              //Число байт данных
              T_Hb_adr? adr[15:8] :     //Старший байт адреса
              T_Lb_adr? adr[ 7:0] :     //Младший байт адреса :
                        dat ;           //Данные
wire ce_crc = ce_sh& !(T_Lb_CRC | T_Hb_CRC);
wire x0 = CRC[0] ^ sr_dat[0] ;

always @ (posedge clk) begin
cb_tact <= (st | ce_tact)? 1 : cb_tact+1;
cb_byte <= st? 0  : ce_stop? cb_byte+1 : cb_byte    ;
cb_bit <= rep_st? 0 : (ce_tact & en_tx)? cb_bit+1 : cb_bit ;
sr_dat <= (T_start & ce_tact)? tx_dat : ce_sh? sr_dat>>1 | 1<<7 : sr_dat ;
en_tx <= st? 1 : ((cb_byte==N_byte+1) & ce_stop)? 0 : en_tx ;
rd_adr <= st? adr : ce_adr? rd_adr+1 : rd_adr ;
//-- Вычисление CRC передатчика----------
CRC <= st? `INIT_CRC : (x0 & ce_crc)? (((CRC^`XCRC16)>>1) | 1<<15): ce_crc? (CRC>>1) : CRC ;
end
endmodule
```

Модуль передатчика ответа для команд записи (8'h00 или 8'h01) 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 2 байта CRC.

1.7 Схема моделирования URXD_CRC_BL

```
module Test_URXD_CRC_BL(
                    //--Выходы имитатора ComChange
    input clk,          output wire UTXD,           //Последовательные данные
    input st,           output wire [7:0] tx_dat,   //Передаваемые байты
```

```
        input [7:0] com,        output wire [15:0] tx_CRC,   //Контрольный код передатчика
        input [7:0] lbl,
        input [15:0] adr,
                //--Выходы приемника данных от ComChange
                        output wire[7:0] rx_com,      //Принятая команда
                        output wire[7:0] rx_lbl,      //Принятая длина блока данных
                        output wire[15:0] wr_adr,     //Базовый адрес данных
                        output wire[7:0] rx_dat,      //Принятый байт
                        output wire en_rx_byte,       //Интервал приема байта
                        output wire en_rx_bl,         //Интервал приема блока байт
                        output wire[7:0] cb_byte_rx, //Счетчик принятых байт
                        output wire ok_rx_byte,       //Подтверждение приема байта
                        output wire ce_wr_dat,        //Разрешение записи данных
                        output wire res,              //Сброс в конце паузы
                        output wire [15:0] rx_CRC,    //Контрольный код приемника
                        output wire ok_rx_bl,         //Конец приема блока байт
                //--Выходы передатчика ответов программе ComChange
                        output wire ret_UTXD,         //Ответные биты данных
                        output wire [7:0] ret_dat,    //Ответные байты
                        output wire [15:0] ret_CRC,   //Ответный контрольный CRC код
                        output wire ret_en_tx);       //Интервал ответа
        //---Имитатор данных от передатчика программы ComChange
Sim_UTXD_CRC DD1 (
                .clk(clk),      .UTXD(UTXD),      //Последовательные данные
                .st(st),        .tx_dat(tx_dat),  //Передаваемые байты
                .com(com),      .CRC(tx_CRC),     //Контрольный CRC код
                .lbl(lbl),
                .adr(adr));
        //---Приемник команд и данных от ComChange
URXD_CRC_BL DD2 (
                .URXD(UTXD),    .com(rx_com),       //Команда
                .clk(clk),      .lbl(rx_lbl),       //Длина блока данных
                                .wr_adr(wr_adr),    //Базовый адрес данных
                                .rx_dat(rx_dat),    //Принятый байт
                                .en_rx_byte(en_rx_byte),//Интервал приема байта
                                .en_rx_bl(en_rx_bl),  //Интервал приема блока байт
                                .cb_byte(cb_byte_rx),//Счетчик принятых байт
                                .ok_rx_byte(ok_rx_byte),//Подтверждение приема байта
                                .ce_wr_dat(ce_wr_dat),//Разрешение записи данных
                                .res(res),          //Сброс в конце паузы
                                .CRC(rx_CRC),       //Контрольный код CRC
                                .ok_rx_bl(ok_rx_bl) );//Конец приема блока байт
        //---Передатчик ответа программе ComChange
wire [15:0]rd_adr ;
wire [7:0]test_dat = rd_adr[7:0]+8'hD0 ;
RET_TXD_CRC_BL DD3 (
                .clk(clk),              .UTXD(ret_UTXD),  //Ответные биты данных
                .st(ok_rx_bl),          .tx_dat(ret_dat),  //Ответные байты
                .com(rx_com),           .CRC(ret_CRC),     //Ответный контрольный CRC код
                .lbl(rx_lbl),           .rd_adr(rd_adr),   //Адрес ответных данных
                .adr(wr_adr),           .en_tx(ret_en_tx),  //Интервал ответа
                .dat(test_dat));
```

endmodule

В этой схеме для простоты и наглядности в качестве данных test_dat модулю RET_TXD_CRC_BL передатчика ответа подставляется младший байт адреса rd_adr[7:0] плюс 8’hD0.

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 генератор сигнала синхронизации clk и значения входных сигналов, т.е. заполнить блок initial.

1.8 Пример к заданию на моделирование схемы Test_URXD_CRC_BL

```
always begin clk=0; #10; clk=1; #10; end //Генератор сигнала синхронизации
        initial begin
                st = 0;  com = 0;       lbl = 0;        adr = 0;
#10000;         st = 1;  com = 8'h00;  lbl = 8'h04;    adr = 16'h5800;//adr=MY_ADR
#20;            st = 0;  com = 8'h00;  lbl = 8'h04;    adr = 16'h5800;// adr=MY_ADR

//#10000;       st = 1;  com = 8'h80;  lbl = 8'h04;    adr = 16'h5800;// adr=MY_ADR
//#20;          st = 0;  com = 8'h80;  lbl = 8'h04;    adr = 16'h5800;// adr=MY_ADR
                end
```

В Process Properties логического симулятора Simulate Behavioral Model надо установить подходящее время моделирования. Например, для lbl=4 и скорости 115.2 kBod достаточно Simulation Run Time = 1500us, (4+2)+(4+4+2)+1 байт по 10*Ttact=170*1/115.2kHz = 1476 us).

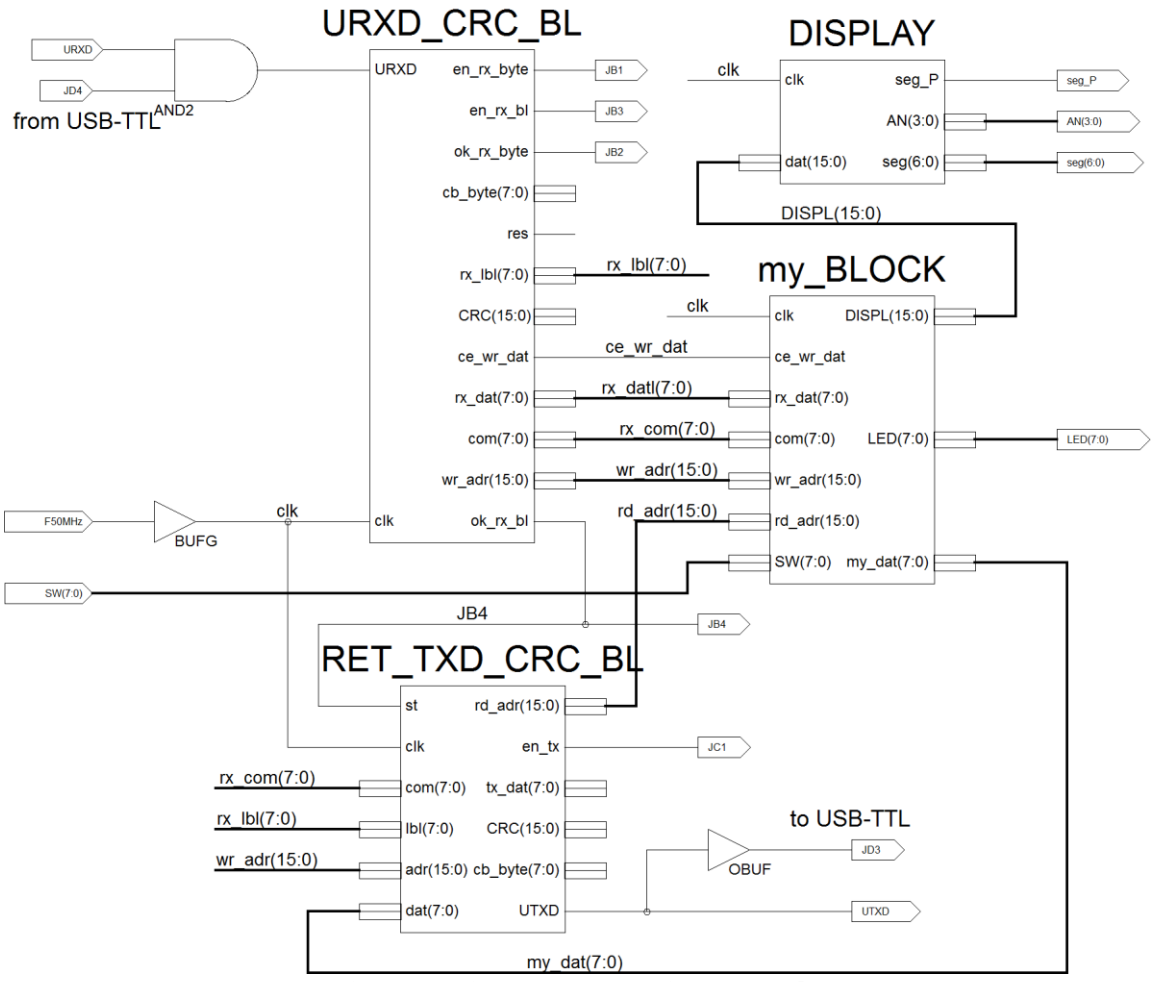

Рис.7 Схема лабораторной работы S_Lab404crc

В этой схеме предусмотрены 2 варианта связи макета NEXYS-2 с COM портом ПК.

- DB-9 ПК с DB-9 макета через кабель «Удлинитель COM порта».
- Порт USB ПК через переходник USB-TTL с выводами JD3 (UTXD), JD4 (URXD) JD4(GND) макета.

В состав схемы рис.7 входит модуль my_BLOCK, который предстоит спроектировать самостоятельно. В этом модуле должно быть 3 8-ми битных регистра: LED[7:0], DISPL[15:8] и DISPL[7:0]. Загружаться эти регистры должны программой ComChange данными модуля URXD_CRC_BL по заданному варианту базового адреса (см. таблицу 1). Например, для загрузки LED и DISPL данными 11, 22, 33 при MY_ADR=16'hA400 от ComChange должна быть получена последовательность байт: 00 03 A4 00 11 22 33. Два байта CRC кода ComChange добавит автоматически, если при настройке в окне «Режим контроля» было выбрано «Контроль CRC».

Модуль my_BLOCK должен обеспечить возможность чтения программой ComChange регистров LED, DISPL и входа SW по тому же базовому адресу. Данные регистров и входа SW должны выдаваться на выход my_dat[7:0] в соответствии с адресом rd_adr модуля RET_TXD_CRC_BL. Например, для чтения 4-х байт (трех регистров и SW) от ComChange модулем URXD_CRC_BL должна быть получена следующая последовательность байт: 80 04 A4 00.

В модуле my_BLOCK должен быть и модуль блочной памяти BMEM_256x8 (см. приложение 5.1). Загружаться BMEM_256x8 должен программой ComChange данными модуля URXD_CRC_BL по заданному варианту базового адреса. Например, для загрузки данными: 11,22,33,44,55 при MY_ADR=16'hA400 от ComChange должна быть получена последовательность байт: 01 05 A4 00 11 22 33 44 55.

При чтении блочной памяти (com=8'h81) выходные данные BMEM_256x8 DO[7:0] должны выдаваться на выход my_dat[7:0] в соответствии с адресом rd_adr модуля RET_TXD_CRC_BL. Например, для чтения 5-и ячеек от ComChange модулем URXD_CRC_BL должна быть получена следующая последовательность байт: 81 05 A4 00.

Таблица 1

№	Скорость BR (BAUDRATE)	Базовый адрес регистров и блока памяти MY_ADR
1	115200	16'h1800
2	57600	16'h2800
3	19200	16'h3500
4	9600	16'h4900
5	4800	16'h5800
6	38400	16'h6E00
7	9600	16'h7700
8	19200	16'h8A00
9	115200	16'h9600
10	57600	16'hA500
11	115200	16'hB800
12	4800	16'hC000
13	9600	16'hD400
14	115200	16'hE300
15	19200	16'hFC00
16	4800	16'h0100
17	19200	16'h8400
18	57600	16'hA200
19	9600	16'h4600
20	115200	16'hA400

2. Задание к допуску (стоимость 2)

2.1 Начертить временные диаграммы TXD(RS232) и UTXD(UART) (рис.2).
2.2 Начертить схему вычислителя кода CRC-16 (рис.4).
2.3 Начертить временные диаграммы передачи и приема блока байт с CRC-16 (рис.5).
2.4 Начертить схему лабораторной работы S_Lab404crc (рис.7).
2.5 Начертить структурную схему модуля **my_BLOCK** (рис.8).

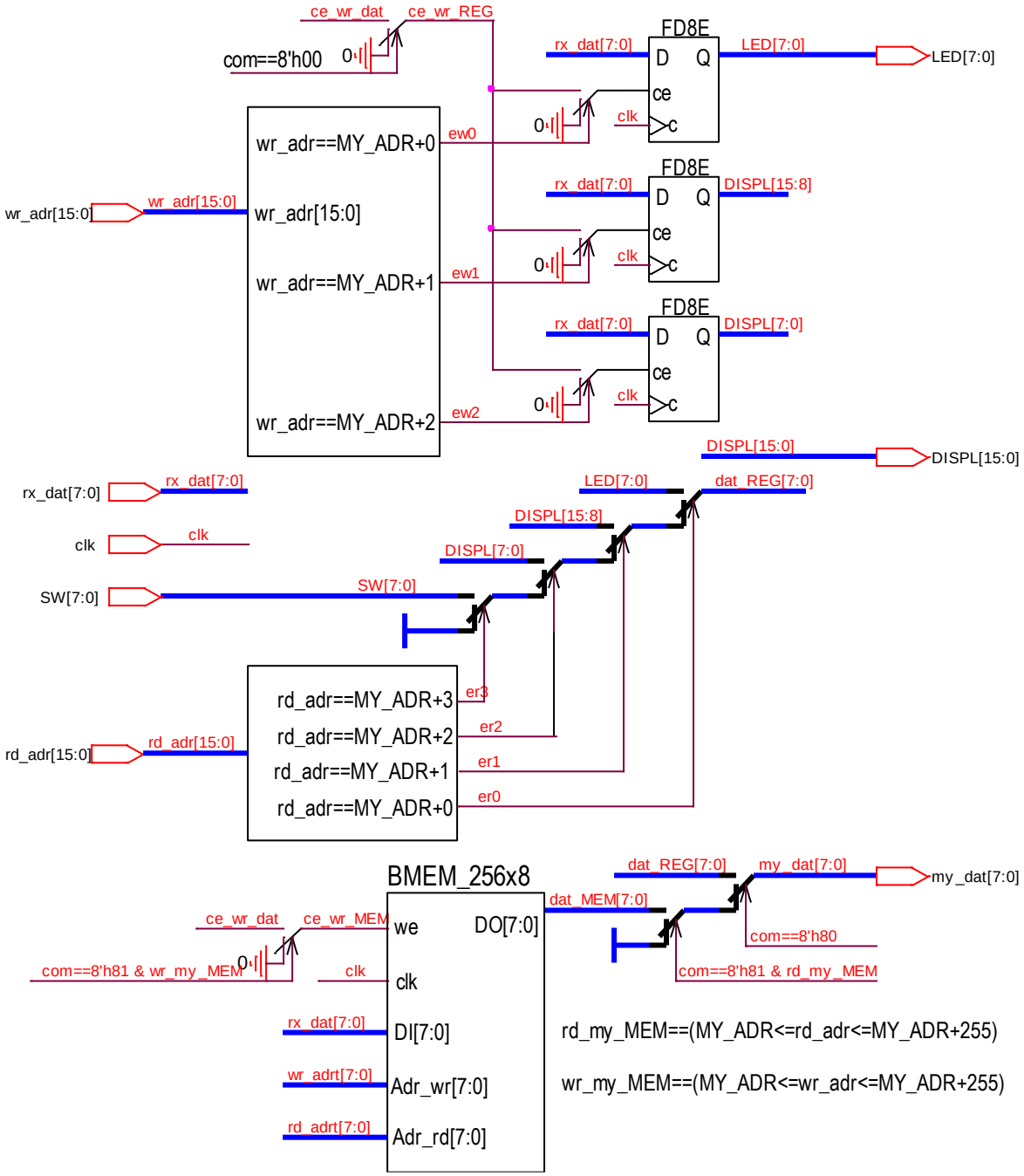

Рис.8 Пример структурной схемы модуля my_BLOCK

3. Задание к выполнению работы (стоимость 4)

Создать проект с именем Lab404crc для ПЛИС, используемой в макете NEXYS-2.

При создании каждого модуля проверить синтаксис (Check Syntax), выполнить синтез (Sinthesize-XST). Исправить ошибки (Errors), проверить существенность предупреждений (Warnings).

3.1 Создать модуль CONST.v (1.3), установить параметры заданного варианта (таблица 1).

3.2 Создать модуль Sim_UTXD_CRC имитатора передатчика ComChange (1.4).

3.3 Создать модуль URXD_CRC_BL приемника блока байт (1.5).

3.4 Создать модуль RET_TXD_CRC_BL передатчика ответа программе ComChange (1.6).

3.5 Создать модуль схемы Test_URXD_CRC_BL (1.7).

3.6 Провести моделирование схемы Test_URXD_CRC_BL для заданного базового адреса и двух команд com=8'h00 и com==8'h80. Зарисовать эскизы временных диаграмм сигналов:

- передатчика имитатора ComChange - tx_dat, tx_CRC,
- приемника – rx_lbl, wr_adr, en_rx_byte, en_rx_bl, cb_byte_rx, rx_CRC,
- передатчика ответа – ret_en_tx, ret_cb_byte, ret_dat, ret_CRC.

4. Задание к сдаче работы (стоимость 4)

4.1 Создать модуль блочной памяти **BMEM_256x8** (приложение 5.1).

4.2 Создать модуль семи сегментного индикатора DISPLAY (приложение 5.2)

4.2 Составить на Verilog-е схему модуля my_BLOCK (см. рис.8) Создать модуль my_BLOCK. При необходимости провести моделирование его работы.

4.3 Создать символы модулей входящих в состав схемы рис.7.

4.4 Составить лист **S_SchLab404crc** схемы лабораторной работы. Разместить на листе схемы необходимые символы, соединить их связями в соответствии с рис.7. (См. также альтернативный вариант схемы на Verilog-е **V_SchLab404crc**, приложение 5.3).

4.5 Создать для схемы лабораторной работы Implementation Constraints File S_SchLab404crc.ucf или V_SchLab404crc.ucf (приложение 5.4).

4.6 Создать конфигурацию составленной схемы загрузить в макет. Отладить при необходимости работу схемы макета. Продемонстрировать работу макета.

5. Приложения

5.1 Схема модуля блочной памяти

```
module BMEM_256x8 (     input clk,              output reg [7-1:0]DO,
                        input we,
                        input [7:0] DI,
                        input [7:0] Adr_wr,
                        input [7:0] Adr_rd);

reg  [7:0] MEM [255:0] ; //блочная память  8 x 256 bit.
always @ (posedge clk) begin
MEM[Adr_wr]<= we? DI :MEM[Adr_wr]; //Запись в память
DO <= MEM[Adr_rd];
end
```

```
endmodule
```

5.2 Модуль семи сегментного индикатора (дисплея)

```
module DISPLAY(
                input clk,              output wire[3:0] AN,  //Аноды
                input[15:0]dat,         output wire[6:0] seg,  //Сегменты
                                        output wire seg_P);    //Точка
parameter Fclk=50000 ;        //50000 kHz
parameter F1kHz=1 ;           //1 kHz
wire [1:0]ptr_P=2'b10 ;       //Точка в центре
reg [15:0] cb_1ms =0 ;
wire ce = (cb_1ms==Fclk/F1kHz) ;
//---Генератор ce------------
always @ (posedge clk) begin
cb_1ms <= ce? 1 : cb_1ms+1 ;
end
//---------------------------------------
reg [1:0]cb_an=0 ; //Счетчик анодов
always @ (posedge clk) if (ce) begin
cb_an <= cb_an+1 ;
end
//-------Переключатель анодов-------------
assign  AN =    (cb_an==0)?  4'b1110 : //включение цифры 0 (младшей)
                (cb_an==1)?  4'b1101 : //включение цифры 1
                (cb_an==2)?  4'b1011 : //включение цифры 2
                             4'b0111 ; //включение цифры 3 (старшей)
//-------Переключатель тетрад (HEX цифр)-------------
wire[3:0] dig =(cb_an==0)? dat[3:0]:
                (cb_an==1)? dat[7:4]:
                (cb_an==2)? dat[11:8]: dat[15:12];
//-------Семи сегментный дешифратор----------
                             //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
                (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
//-------Указатель точки-------
assign seg_P = !(ptr_P == cb_an) ;
```

```
endmodule
```

5.3 Схема лабораторной работы на Verilog-е

```
module V_SchLab404crc (
   input URXD,       output wire UTXD,    //UTXD
   input JD4,        output wire JD3,     //UTXD (USB_TTL)
   input F50MHz,     output wire [3:0] AN,//Аноды
   input [7:0] SW,   output wire [6:0] seg, //Сегменты
                     output wire seg_P,   //Точка
                     output wire [7:0] LED,//Светодиоды
                     output wire JB1,     //en_rx_byte
                     output wire JB2,     //ok_rx_byte
                     output wire JB3,     //en_rx_bl
                     output wire JB4,     //ok_rx_bl
                     output wire JC1);    //en_tx

wire clk ;
assign JD3=UTXD ; //--Дополнительный выход UTXD для USB_TTL

//--Глобальный буфер сигнала синхронизации----
BUFG DD1 (.I(F50MHz),    .O(clk));

//--UART приемник с контролем CRC-16
wire Inp_RXD = URXD & JD4 ;
wire [7:0]rx_com ; wire[7:0]rx_lbl ; wire [7:0]rx_dat ;
wire [15:0] wr_adr ;
wire ce_wr_dat, ok_rx_bl, ok_rx_byte, en_rx_bl ;
URXD_CRC_BL DD2 (
        .URXD(Inp_RXD),   .com(rx_com),          //Команда
        .clk(clk),        .rx_lbl(rx_lbl),       //Длина блока данных
                          .wr_adr(wr_adr),       //Базовый адрес данных
                          .rx_dat(rx_dat),       //Принятый байт
                          .ce_wr_dat(ce_wr_dat), //Разрешение записи данных
                          .en_rx_byte(en_rx_byte), //Интервал приема байта
                          .en_rx_bl(en_rx_bl),   //Интервал приема блока байт
                          .ok_rx_byte(ok_rx_byte), //Подтверждение приема байта
                          .ok_rx_bl(ok_rx_bl));  //Конец приема блока байт


//--UART передатчик с контролем CRC-16
wire [15:0] rd_adr ;  wire [7:0]my_dat ; wire en_tx ;
RET_TXD_CRC_BL DD3 (
                .clk(clk),          .UTXD(UTXD),     //Последовательные данные
                .st(ok_rx_bl),      .en_tx(en_tx),   //Интервал передачи блока байт
                .com(rx_com),       .rd_adr(rd_adr), //Адрес передаваемых байт данных
                .lbl(rx_lbl),
                .adr(wr_adr),
                .dat(my_dat));

//--Мой блок----------
wire [15:0]disp_dat ;
```

```
my_BLOCK DD4 (
                .clk(clk),                                      .DISPL(disp_dat),
                .ce_wr_dat(ce_wr_dat),          .LED(LED),
                .rx_dat(rx_dat),                        .my_dat(my_dat),
                .com(rx_com),
                .wr_adr(wr_adr),
                .rd_adr(rd_adr),
                .SW(SW));
//---Семи сегментный индикатор---------
DISPLAY DD5 (.clk(clk),         .AN(AN),        //Аноды
                .dat(disp_dat),         .seg(seg),      //Сегменты
                                        .seg_P(seg_P));//Точка

assign JB1=en_rx_byte ;
assign JB2=ok_rx_byte ;
assign JB3=en_rx_bl ;
assign JB4=ok_rx_bl ;
assign JC1=en_tx ;

endmodule
```

5.4 Список выводов ПЛИС макета Nexys2

```
NET "F50MHz"  LOC = "B8"  ; #F50MHz
#---- Аноды светодиодов сегментов индикатора
NET "AN<0>"  LOC = "F17"  ; #AN0
NET "AN<1>"  LOC = "H17"  ; #AN1
NET "AN<2>"  LOC = "C18"  ; #AN2
NET "AN<3>"  LOC = "F15"  ; #AN3
#--- Катоды светодиодов сегментов индикатора
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_P"  LOC = "C17"  ; #CP точка
#---Кнопки
#NET "BTN0"  LOC = "B18"  ; #BTN3
#NET "BTN1"  LOC = "D18"  ; #BTN2
#NET "BTN2"  LOC = "E18"  ; #BTN1
#NET "BTN3"  LOC = "H13"  ; #BTN0
#---Светодиоды
NET "LED<0>"  LOC = "J14" ; #LD0
NET "LED<1>"  LOC = "J15" ; #LD1
NET "LED<2>"  LOC = "K15" ; #LD2
NET "LED<3>"  LOC = "K14" ; #LD3
NET "LED<4>"  LOC = "E17" ; # LD4
NET "LED<5>"  LOC = "P15" ; #LD5
NET "LED<6>"  LOC = "F4"   ; #LD6
NET "LED<7>"  LOC = "R4"   ; #LD7
```

```
#---Переключатели
NET "SW<0>"  LOC = "G18"  ; #SWT0
NET "SW<1>"  LOC = "H18"  ; #ADR[1]
NET "SW<2>"  LOC = "K18"  ; #ADR[2]
NET "SW<3>"  LOC = "K17"  ; #ADR[3]
NET "SW<4>"  LOC = "L14"  ; #ADR[4]
NET "SW<5>"  LOC = "L13"  ; #ADR[5]
NET "SW<6>"  LOC = "N17"  ; #ADR[6]
NET "SW<7>"  LOC = "R17"  ; #ADR[7]
#----COM порт
NET "URXD"  LOC = "U6"  ; #TXD U6
NET "UTXD"  LOC = "P9"  ; #TXD P9
#---Выводы порта JA
#NET "JA1"  LOC = "L15"  ;#T_start
#NET "JA2"  LOC = "K12"  ;#T_AC
#NET "JA3"  LOC = "L17"  ;#en_tx
#NET "JA4"  LOC = "M15"  ;#T_stop
#NET "JA7"   LOC = "K13"  ;#en_tx_bl
#NET "JA8"   LOC = "L16"  ;#Pin8
#NET "JA9"  LOC = "M14"  ;#Pin9
#NET "JA10" LOC = "M16"  ;#Pin10
#---Выводы порта JB
NET "JB1"  LOC = "M13" ; #en_rx_byte
NET "JB2"  LOC = "R18" ; #ok_rx_byte
NET "JB3"  LOC = "R15" ; #en_rx_bl
NET "JB4"  LOC = "T17" ; #ok_rx_bl
#NET "JB7"  LOC = "P17"  ; #Pin7
#NET "JB8"   LOC = "R16"  ; #Pin8
#NET "JB9"  LOC = "T18"  ; #Pin9
#NET " JB10" LOC = "U18"  ; #Pin10
#---Выводы порта JC
NET "JC1" LOC = "G15"   ; #en_tx
#NET "JC2" LOC = "J16"   ; #
#NET "JC3"  LOC = "G13"  ; #Pin3
#NET "JC4"  LOC = "H16"  ; #Pin4
#NET "JC7" LOC = "H15"   ; #
#NET "JC8" LOC = "F14"   ; #
#NET "JC9" LOC = "G16"   ; #
#NET "JC10" LOC = "J12"  ; #
#---Выводы порта JD
#NET "JD1"  LOC = "J13"  ; #Pin1
#NET "JD2"  LOC = "M18"  ; #Pin2
NET "JD3"  LOC = "N18"   ; #TXD (USB-COM)
NET "JD4"  LOC = "P18"  |  PULLUP  ; #RXD (USB-COM)
#NET "JD7"   LOC = "K14"  ; #LED0
#NET "JD8"   LOC = "K15"  ; #LED1
#NET "JD9"  LOC = "J15"  ; #LED2
#NET "JD10" LOC = "J14"  ; #LED3
```