Лабораторная работа Lab408

1. I2C интерфейс
(inter-IC или IIC (I2C) шина)

В шине интерфейса I2C используется всего два сигнальных проводника линий: SDA - данных и SCL - синхронизации. Выходные каскады устройств, подключенных к проводникам шины I2C, должны иметь: открытый сток, открытый коллектор или буфер с третьим состоянием (BUFE или DUFT). Высокий уровень сигнала на проводниках линий обеспечивается резисторами, соединяющими их с источником питания VCC (рис.1). Такое соединение реализует функцию монтажного “И” (по логическим единицам) или монтажного ”ИЛИ” (по логическим нулям), что обеспечивает двунаправленность шины. Линию синхронизации SCL, как правило, занимает только одно устройство, которое называют ведущим, а линию данных SDA «слушают» всегда и все, и занимать могут занимать тоже все, но только не одновременно, а строго по запросу ведущего.

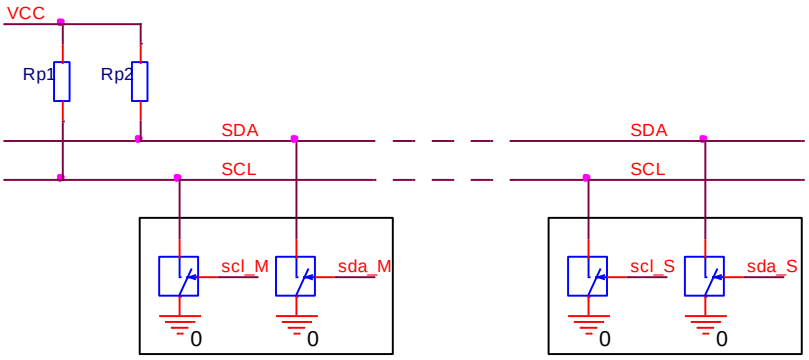

Рис.1 Соединение выходов устройств интерфейса I2C по схеме монтажного “И”

Процедура обмена начинается сигналом START. START это спад (negedge) сигнала SDA (переход из 1 в 0) при SCL=1 (рис.3,a). Этот переход воспринимается всеми устройствами, подключенными к шине как старт обмена.

Процедура обмена завершается сигналом STOP. STOP это фронт (posedge) SDA (переход из 0 в 1) также при SCL=1 (рис.3,b).

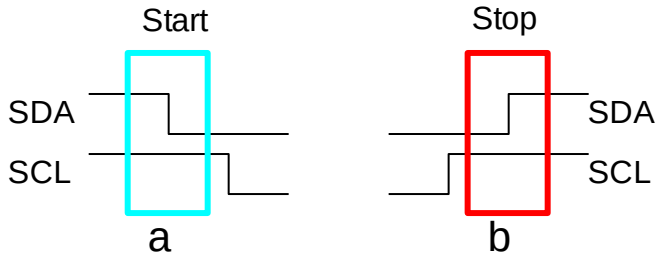

Рис.3 Старт и стоп обмена интерфейса I2C (ce_start=(spad_SDA & SCL), ce_stop=(front_SDA & SCL)).

Данные (SDA) действительны и должны оставаться постоянными при высоком уровне сигнала SCL=1. При низком уровне SCL=0 данные могут произвольно меняться. Таким образом между «стартом» и «стопом» изменения на линии SDA допускаются только при НИЗКОМ уровне сигнала на линии SCL=0 (рис.4).

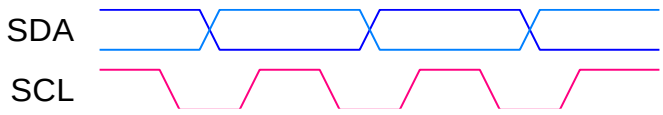

Рис.4 Правило изменения SDA в процессе обмена интерфейса I2C

После «старта», ведущий переводит SCL в НИЗКОЕ состояние и выставляет на линию SDA старший бит первого байта сообщения. Количество байт в сообщении не ограничено.

Каждое ведомое устройство распознается по уникальному адресу и может работать как приёмник, так и по запросу ведущего как передатчик данных.

Передача каждых 8 бит данных от передатчика к приемнику завершаются дополнительным девятым тактом АС при, котором ведомый выставляет низкий уровень сигнала на линии SDA, как признак успешного приема байта. Если при передаче первого байта ведущий не получает на 9-м такте АС подтверждения (низкого уровня), то он формирует сигнал СТОП, т.е. ограничивается передачей одного байта.

Каждое ведомое устройство, подключённое к шине, должно иметь уникальный адрес. Первый байт ведущего это адрес-команда, в котором старшие 7 бит - адрес ведомого, последний (младший) бит – команда: 0 – запись, 1 – чтение.

Ведомый не должен вырабатывать сигал подтверждения АС ни для первого байта ни для всех последующих байт обмена с другими ведомыми если первые 7 бит первого байта не совпадают с битами его адреса.

Ведущий должен знать структуру ведомого, к которому он обращается. Например, если в простейшем случае ведомый имеет только один 8-ми битный регистр, то по команде записи второй байт будет данными для ведомого, а по команде чтения второй байт будет данными от ведомого. На интервале приема байта данных от ведомого ведущий не должен занимать шину SDA (SDA_MASTER=1), но на такте АС должен давать подтверждение ведомому, что байт принят. Если ведомый имеет несколько регистров и они адресуемы, тогда первый байт от ведущего – адрес ведомого, второй байт - адрес младшего регистра, а остальные байты это данные регистров.

Если известно, что к шине подключен только один ведущий и ни у одного из ведомых нет необходимости замедлять скорость обмена, удерживая в 0 линию синхронизации SCL, то у ведущего для сигнала SCL можно использовать выходной буфер без третьего состояния (OBUF). Но лучше этого не делать потому, что это уменьшает допустимую емкость нагрузки шины I2C. Адрес имеет 7 разрядов поэтому к шине I2C, не считая ведущего, может быть подключено до 128 устройств. Каждое устройство нагружает шину емкостями выводов SDA и SCL. Например, даже при небольшой емкости 20pF 100 устройств нагрузят линии SDA и SCL емкостью по 2000pF.

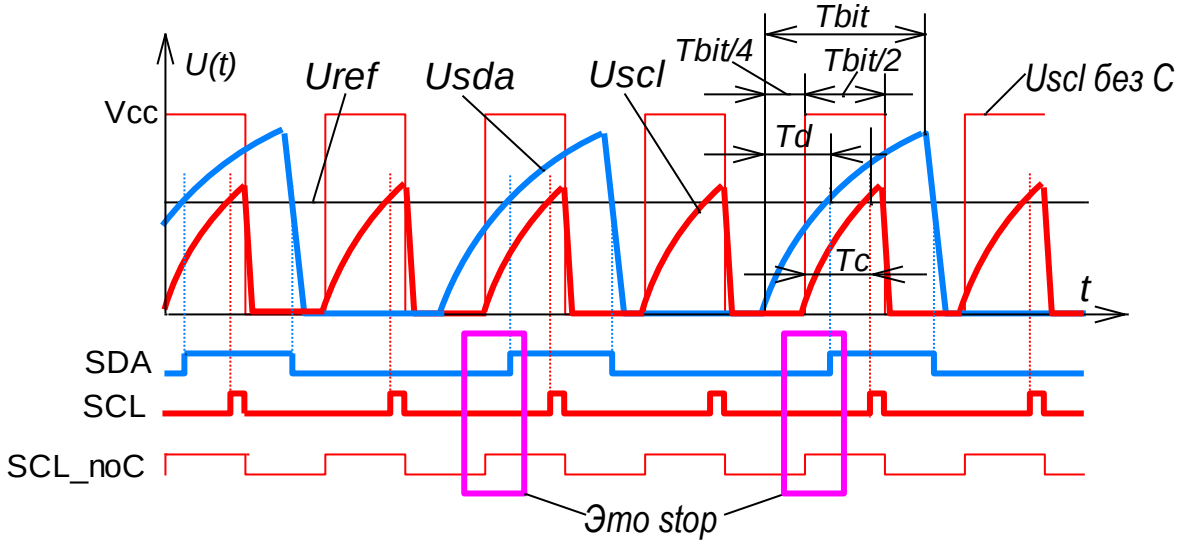

Рис.2 Временные диаграммы напряжений на линиях SDA и SCL

Стандартная скорость шины I2C 100kbit/s (Tbit=10us), но допустима в отдельных случаях и скорость 400kbit/s (Tbit=2.5us). Емкость С линии SCL через резистор R от источника Vcc должна зарядится до порога Uref компаратора за время меньшее, чем длительность импульса SCL, которая равна Tbit/2. При Uref=Vcc/2, R*C*ln2 должно быть меньше Tbit/2. Пусть R=1k, Tbit=2.5us тогда

C<Tbit/2Rln2=2.5*10^{-6}/(2*10^{3}*0.693)=1803pF. Разряжается емкость линии ключом, замыкающим линию на «землю». Время разряда емкости линии током ключа тоже должно быть меньше Tbit/2 При типичном токе ключа 8mA порта I/O ПЛИС время разряда емкости 1803pF от 5V до 0 будет равно 1.127us, т.е. меньше Tbit/2 даже при максимальной скорости 400kbit/s. На рис.2 показано к чему приводят короткие фронты напряжения линии SCL.

2. Модуль ведущего интерфейса I2C

Модуль предназначен для записи или чтения адресуемых 8-ми битных регистров ведомого. Число регистров ведомого не превышает 256, т.е. адрес регистра имеет 8 бит. Для такой модели ведомого ведущий формирует кадр обмена из трех байт: адрес-команда, адрес регистра, данные ведущего или ведомого. Пример символа такого модуля приведен на рис.5.

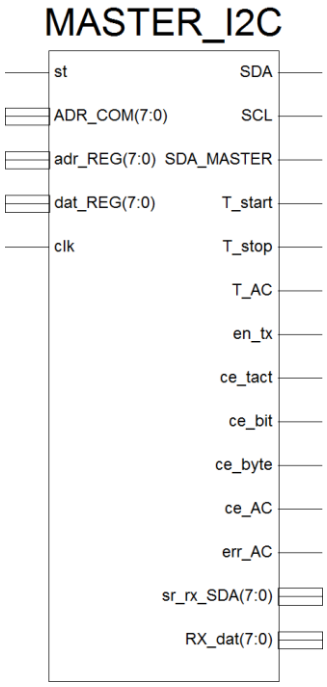

Рис.5 Символ модуля ведущего интерфейса I2C

В этом модуле, кроме необходимых выходных портов SCL, SDA и выходного буфера RX_dat[7:0], принятых данных от ведомого, имеются и другие выходные порты, предназначенные как для иллюстрации работы его внутренних устройств, так и для помощи проектирования и отладки схемы модуля ведомого.

Входы модуля MASTER_I2C:

- clk – сигнал синхронизации,
- st – импульс запуска передачи,
- ADR_COM[7:0] – адрес-команда (ADR_COM[7:1]-адрес ведомого, ADR_COM[0] – команда),
- adr_REG[7:0] – адрес регистра ведомого,
- dat_REG[7:0] – данные для ведомого.

Пример временных диаграмм передачи первого байта ведущим MASTER_I2C интерфейса I2C приведены на рис.6.

Выходные буферы с третьим состоянием BUFT ведущего и ведомого совместно с резистором R1 образуют функцию “И” логических сигналов SDA_MASTER и SDA_SLAVE для физического сигнала SDA (SDA=SDA_MASTER & SDA_SLAVE).

Если на 9-м такте T_AC_MASTER (cb_bit_MASTER=8) нет подтверждения (SDA=1), то ведущий формирует только 9 положительных импульсов сигнала синхронизации SCL и заканчивает передачу сигналом Stop.

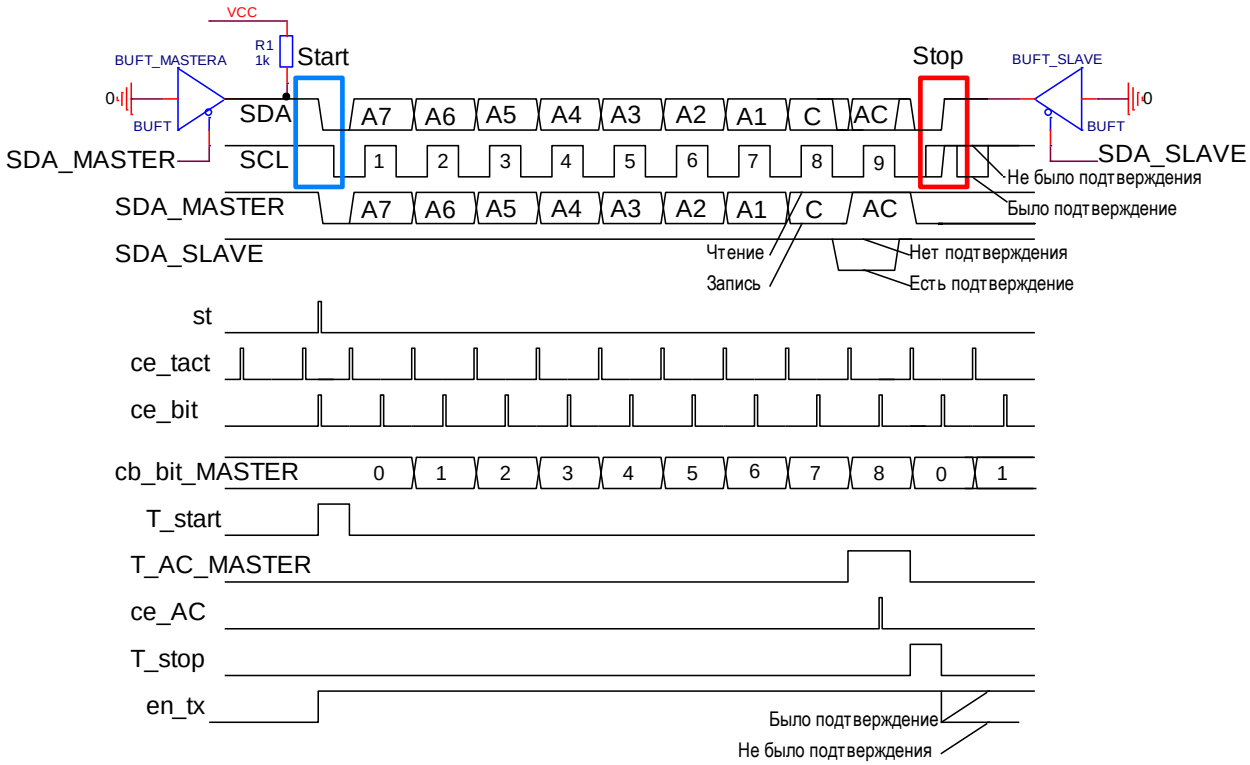

Рис.6 Временные диаграммы передачи первого байта (адрес-команда) ведущего (MASTER_I2C)

2.1 Схема модуля ведущего интерфейса I2C на языке Verilog

```
module MASTER_I2C (inout wire SDA,     // Физический сигнал SDA мастера
    input st,                  output reg SCL=1,      // Сигнал SCL мастера
    input clk,                 output reg SDA_MASTER=1,   //Логический сигнал SDA мастера
    input [7:0]ADR_COM,        output reg T_start=0,      // Старт передачи
    input [7:0]adr_REG,        output reg T_stop=0,       // Стоп передачи
    input [7:0]dat_REG,        output reg[3:0]cb_bit=0,  // Счетчик бит
                               output wire T_AC,          // Такт подтверждения
                               output reg en_tx=0,        // Разрешение передачи
                               output wire ce_tact,       // Границы тактов
                               output wire ce_bit,        // Середины тактов
                               output wire ce_byte,       //Границы байт
                               output wire ce_AC,         //Строб такта T_AC
                               output reg err_AC=0,           //Триггер подтверждения
                               output reg [2:0]cb_byte=0,   //Счетчик байт
                               output reg[7:0]sr_rx_SDA=0,//Регистр сдвига принимаемых данных
                               output reg[7:0] RX_dat=0);  //Регистр данных от ведомого

PULLUP DA1(SDA); //PULLUP Резистор

//----- T-буфер--(T=0 O=I, T=1 SDA=O=Z)-----------------
```

```
BUFT DD1 (     .I(1'b0),       .O(SDA),
               .T(SDA_MASTER));

parameter Fclk=50000000 ;  //Fclk =50 000 kHz
parameter Fvel= 1250000 ;   //Fvel = 1 250 kHz
parameter N4vel=Fclk/(4*Fvel);//50000000/(4*1250000)=10
parameter N_byte = 3 ;        //Число байт (адрес ведомого, адрес регистра, данные)

reg [10:0]cb_ce=4*N4vel ;
assign ce_tact =(cb_ce==1*N4vel) ;        //10
assign ce_bit = (cb_ce==3*N4vel) & en_tx ; //30
reg[7:0]sr_tx_SDA=8'h00 ;           //Регистр сдвига передаваемых данных

assign T_AC= (cb_bit==8) ;          //Контрольный такт
wire T_dat = en_tx & !T_start & !T_stop & !T_AC;
assign ce_AC= T_AC & ce_bit;          //Строб контроля AC
assign ce_byte = ce_tact & T_AC ;     //ce_byte

wire R_W = ADR_COM[0] ;             //1-чтение, 0-запись
reg rep_st = 0;
//Интевал передачи данных
wire [7:0]TX_dat =     (cb_byte==0)? ADR_COM :          //Адрес_команда
            (cb_byte==1)? adr_REG :             //Адрес регистра
            ((cb_byte==2) & !R_W)? dat_REG : 8'hFF ; //Данные регистра

always @ (posedge clk) begin
cb_ce <= st? 3*N4vel : (cb_ce==1)? 4*N4vel : cb_ce-1 ; // 3*N4vel-задержка первого ce_bit от st
T_start <= st? 1 : ce_tact? 0 : T_start ;
cb_bit <= (st | ce_byte)? 0 : (ce_tact & en_tx & !T_start)? cb_bit+1 : cb_bit ;

T_stop <= ce_byte & ((cb_byte==N_byte-1) | err_AC)? 1 : ce_bit? 0 : T_stop ;
en_tx <= st? 1 : (T_stop & ce_bit)? 0 : en_tx ;
SCL <= (cb_ce>2*N4vel) | !en_tx ;
SDA_MASTER <= (T_start | T_stop)? 0 : en_tx? (sr_tx_SDA[7] | T_AC) : 1 ;

sr_tx_SDA <= rep_st? TX_dat : (ce_tact & T_dat)? sr_tx_SDA<<1 | 1'b1 : sr_tx_SDA ;
cb_byte <= st? 0 : (ce_byte & en_tx)? cb_byte+1 : cb_byte ;
err_AC <= st? 0 : (ce_AC & SDA)? 1 : err_AC ; //1- нет подтверждения (AC_SLAVE=1)
rep_st = (st | (ce_byte & en_tx)) ;
//Последовательный прием данных от ведомого на интервале третьего байта
sr_rx_SDA <= ((cb_byte==N_byte-1) & ce_bit & T_dat)? sr_rx_SDA<<1 | SDA : sr_rx_SDA ;
RX_dat <= ((cb_byte==N_byte-1) & ce_byte & R_W)? sr_rx_SDA : RX_dat ;//N_byte-1
end
endmodule
```

Для Fvel=400 кБод получается небольшая не симметрия длительностей 0 ($T0_{SCL}$) и 1 ($T1_{SCL}$) сигнала SCL, $T0_{SCL} = 62*T_{clk}$, $T1_{SCL} = 63*T_{clk}$.

Компонент DA1 PULLUP резистор (PULLUP DA1(SDA);) необходимо использовать при моделировании работы модуля MASTER_I2C.

3. Модуль ведомого SLAVE_I2C

В модуле ведомого SLAVE_I2C должен быть блок 8-ми битных регистров. Схема такого блока, реализованного на основе блока «слайсовой» памяти приведена ниже.

3.1 Модуль блока регистров REG_BL

```
module REG_BL (    input clk,     output wire [7:0]dat_REG, //Выходные данные
          input we,     //Разрешение записи
          input [7:0] DI,       //Входные данные
          input [7:0] Adr_wr,    //Адрес записи
          input [7:0] Adr_rd);    //Адрес чтения

reg [7:0]MEM[255:0] ;    //Модуль памяти
assign  dat_REG = MEM[Adr_rd]; //Слайсовая память (256 регистров)
always @ (posedge clk) begin
MEM[Adr_wr] <= we?  DI : MEM[Adr_wr] ; //Запись при we=1 по фронту clk
end
endmodule
```

В этом модуле всего 256 регистров (8-ми битных ячеек памяти), но ведомый должен выполнять запись и чтение только в заданное число N_REG регистров начиная с базового адреса BASE_ADR (см. Табл. 1).

На рис.7 приведены временные диаграммы сигналов ведомого на интервале первого байта

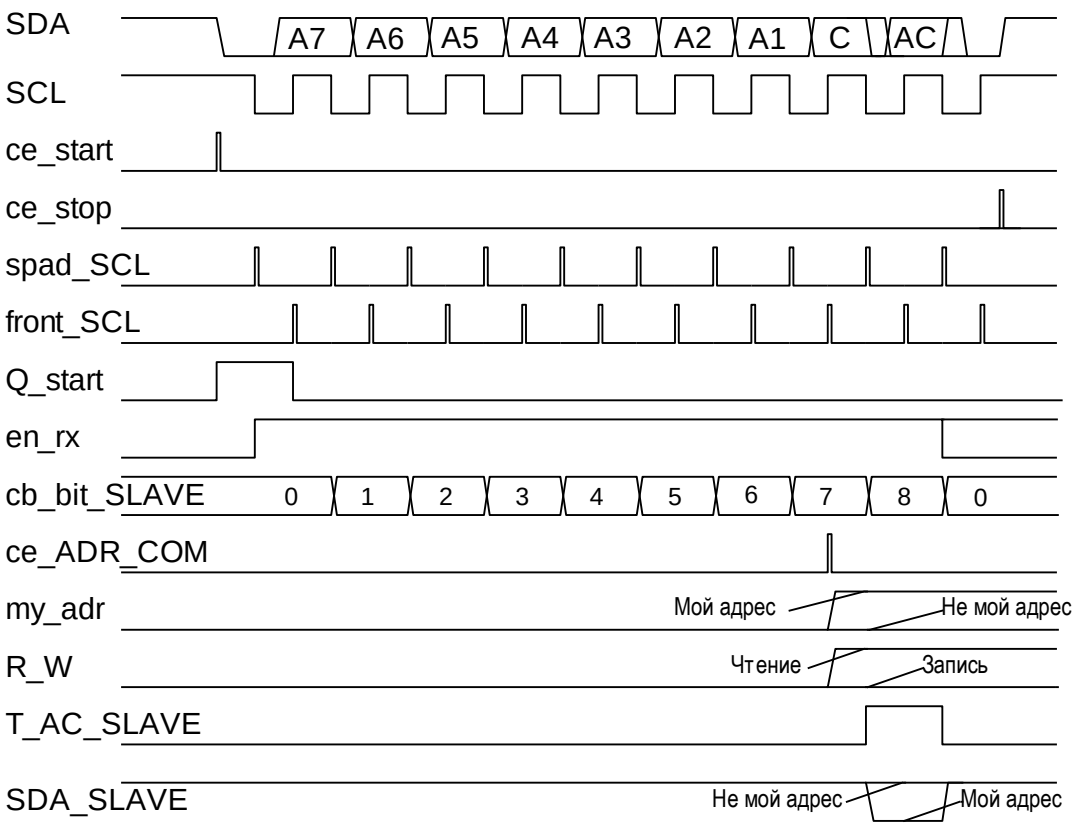

Рис.7 Временные диаграммы ведомого (SLAVE_I2C)

При создании и отладке модуля SLAVE_I2C можно ориентироваться на эти временные диаграммы. Например, для сигналов front_SCL, spad_SCL, ce_start и ce_stop можно предложить приведенную ниже схему на Verilog-e (2.3).

3.2 Схема формирования сигналов front_SCL, spad_SCL, ce_start и ce_stop на Verilog-e

```
reg tSDA=0, ttSDA=0, tSCL=0, ttSCL=0; //D-триггеры
wire spad_SDA = !tSDA &  ttSDA ;   wire front_SDA = tSDA & !ttSDA ;
wire ce_start = spad_SDA & SCL ;    wire ce_stop = front_SDA & SCL ;
wire spad_SCL = !tSCL &  ttSCL ;    wire front_SCL = tSCL & !ttSCL ;
always @ (posedge clk) begin
tSDA <= SDA ;    ttSDA <= tSDA; //Задержка на Tclk
tSCL <= SCL ;    ttSCL <= tSCL ; // Задержка на Tclk
end
```

Для сигнала SDA_SLAVE можно, например, предложить следующую логическую функцию:

```
wire SDA_SLAVE = !(T_AC & my_adr & (cb_byte==1)) &
                 !(T_AC & my_adr & my_reg) &
                 !(R_W & my_reg & (cb_byte>=2) & !sr_tx[7]) ;
```

Старший бит sr_tx[7] регистра sr_tx передатчика ведомого в конце кадра должен быть равен 1. Для этого необходимо на интервале в его младший бит вдвигать 1, а загружать в него данные модуля REG_BL по стробу tce_adr_REG с задержкой на Tclk по отношению к стробу записи ce_adr_REG адреса регистра (ce_adr_REG=(ok_rx_byte & (cb_byte==1))), где ok_rx_byte=(cb_bit==7) & spad_SCL.

Например:

```
sr_tx <= (tce_adr_REG & R_W)? dat_REG :
         ((cb_byte==2) & R_W & !T_AC & spad_SCL)? sr_tx<<1 | 1'b1: sr_tx ;
```

Регистр my_adr должен устанавливаться в 1 по стробу (ce_ADR_COM = (ok_rx_byte & (cb_byte==0))) при условии, что sr_rx[7:1] первого принятого байта (cb_byte==0) совпадает с adr_SLAVE, а регистр команды W_R должен принимать значение sr_rx[0].

Регистр my_reg должен устанавливаться в 1 по стробу tce_adr_REG при условии, что принятые данные sr_rx второго байта (cb_byte==1), т.е адрес регистра, укладывается в заданный интервал адресов: (sr_rx>=`BASE_ADR) & (sr_rx<`BASE_ADR+`N_REG).

Регистры my_adr, R_W, my_reg можно после окончания сеанса не сбрасывать и для наглядности выводить на светодиоды: LED7=my_adr, LED1=my_reg, LRD0=R_W.

3.3 Модуль параметров **CONST.v**

```
`define Fclk 50000000      //50 MHz
//---Для COM порта-----------------
`define UART_vel  115200   //115.2kBOD
`define UART_Nt  `Fclk/`UART_vel //434
//---Для I2C-----------------------
`define Fvel 1250000       //Скорость обмена бит/сек (из таблицы 1)
`define N4vel `Fclk/(4*`Fvel)    //50000000/(4*1250000)=10
`define BASE_ADR 8'hE0    //Базовый адрес регистров ведомого (из таблицы 1)
`define N_REG 8     //Число регистров ведомого (из таблицы 1)
```

4. Задание к допуску (стоимость 2)

4.1 Начертить в тетради временные диаграммы модуля MASTER_I2C (рис.6).
4.2 Переписать в тетрадь схему (2.1) модуля MASTER_I2C.
4.3 Начертить в тетради временные диаграммы модуля SLAVE_I2C (рис.7).
4.4 Переписать модуль CONST.v с данными своего варианта из таблицы 1.

Таблица 1

№ варианта	Скорость Fvel [bit/s]	Адрес ведомого adr_SLAVE SW[7:1] [BIN]	Базовый адрес блока регистров BASE_ADR [HEX]	Число регистров ведомого N_REG [DEC]
1	125000	0000001	8’h10	16
2	625000	0000010	8’h 20	32
3	500000	0000011	8’h 30	64
4	400000	0000100	8’h 40	128
5	250000	0000101	8’h 50	64
6	125000	0000110	8’h 60	32
7	62500	0000111	8’h 70	16
8	50000	0001000	8’h 80	8
9	250000	0001001	8’h 90	16
10	400000	0001010	8’h A0	32
11	50000	0001011	8’h B0	64
12	625000	0001100	8’h C0	32
13	1250000	0001101	8’h D0	16
14	625000	0001110	8’h E0	8
15	500000	0001111	8’h F0	4
16	400000	0010000	8’h 10	14
17	250000	0010001	8’h 20	32
18	125000	0010010	8’h 30	64
19	62500	0010011	8’h 40	16
20	12500	0010100	8’h 50	8

5. Задание к выполнению (стоимость 5)

В папке FRTK создать папку со своим именем (только латинские символы). Далее в этой папке в системе ISE Design Suite 14.4 создать проект с именем Lab408, для ПЛИС используемой в макете (для NEXYS2: Spartan3E, XC3S500E, FG320, XST (VHDL/Verilog), Isim(VHDL/Verilog), Store all values).

В режиме Design в окне Hierarchy или окне Source создать (New Source) заданный модуль (Verilog Module). При создании модуля можно сделать его пустым, т.е. не задавать ни входы, ни выходы, а в полученную заготовку вставить готовый текст схемы модуля из методички или вписать его самостоятельно. Сделать его главным в проекте (Set as Top Module). Проверить синтаксис веденного текста схемы (Check Syntax). Выполнить Synthesize-XST. Исправить возможные ошибки (Errors), обратить внимание на предупреждения (Warnings).

5.1 Создать модуль **MASTER_I2C.** Выполнить Synthesize-XST. Исправить возможные ошибки.

5.2 Создать для модуля **MASTER_I2C** задание на моделирование (Verilog Test Fixture).

5.2.1 Содержательная часть задания на моделирование (Verilog Test Fixture)

```
always begin clk=0 ; #10 clk=1; #10 ; end //Сигнал синхронизации (период Tclk=20 нс)
initial begin
   st = 0;ADR_COM = 8'h00;  adr_REG = 8'h00;  dat_REG = 8'h00;//Исходное состояние
#5000;    st = 1;ADR_COM = 8'h??;  adr_REG = 8'h??;  dat_ REG = 8'h??;//Импульс
#20;   st = 0;ADR_COM = 8'h??;  adr_REG = 8'h??;  dat_ REG = 8'h??;//запуска Tst=Tclk
end
```

Здесь вместо “??” надо подставить значения ADR_COM и adr_REG соответствующие заданному варианту параметров (см. табл.1), а dat_ REG может быть произвольным, например, номер варианта задания.

5.3 Провести моделирование модуля (2.1) **MASTER_I2C.** Зарисовать полученные временные диаграммы.

5.4 Составить схему модуля SLAVE_I2C. Провести моделирование его работы.

При составлении и отладке схемы модуля SLAVE_I2C можно последовательно вводить и отлаживать устройства, временные диаграммы сигналов которых приведены на рис.8.

Источником сигналов SCL и SDA для отлаживаемого модуля SLAVE_I2C должен быть модуль MASTER_I2C.

5.4.1 Пример схемы **Test_MASTER_SLAVE_I2C** для совместного моделирования модулей MASTER_I2C и SLAVE_I2C.

```
module Test_MASTER_SLAVE_I2C(
        inout wire SDA,                          //Физический сигнал SDA
    //---------Сигналы модуля ведущего MASTER_I2C
    input clk,              output wire SCL,     //Сигнал SCL мастера
    input[7:0]ADR_COM,      output wire en_tx,   //Регистр разрешения передачи
    input[7:0]adr_REG,      output wire T_AC,    //Такт подтверждения приема байта
    input[7:0]dat_REG,      output wire ce_byte_tx,      //Конец байта
    input st,               output wire T_start,         //Старт передачи
                            output wire [3:0]cb_bit_tx,  //Счетчик бит ведущего
                            output wire [2:0]cb_byte_tx, //Счетчик байт
                            output wire [7:0]RX_dat,     //Регистр данных от ведомого
                            output wire [7:0]sr_rx_SDA,//Рег. Сдвига принимаемых данных
                            output wire T_stop,          //Стоп передачи
      //---------Сигналы модуля ведомого SLAVE_I2C
    input[7:1]Adr_SLAVE, output wire ce_start_rx,    //Старт приема
                            output wire SDA_SLAVE,   //Данные SLAVE-а
                            output wire en_rx,       //Интервал приема
                            output wire R_W,         //Команда (чтение/запись)
                            output wire [3:0]cb_bit_rx,  //Счетчик бит
```

```
                output wire ce_stop_rx,        //Крнец сеанса приема
                output wire [7:0]sr_rx,        //Регистр сдвига приема
                output wire [7:0]sr_tx,        //Регистр сдвига передачи
                output wire ok_rx_byte,        //Конец приема байта
                output wire AC_rx,             //Бит подтвеждения
                output wire [2:0]cb_byte_rx,//Счетчик байт
                output wire[7:0]dat_MEM,   //Данные модуля памяти
                output wire my_adr,            //Мой адрес
                output wire my_reg);           //Мой регистр

MASTER_I2C DD1 (   .SDA(SDA),//Физический сигнал SDA мастера
                   .SCL(SCL),//Физический сигнал SCL мастера
    .st(st),       .RX_dat(RX_dat),        //Регистр данных от ведомого
    .clk(clk),     .en_tx(en_tx),          //Регистр разрешения передачи
    .ADR_COM(ADR_COM),  .T_AC(T_AC),   //Бит подтверждения приема байта

    .adr_REG(adr_REG),  .cb_bit(cb_bit_tx),    //Счетчик бит
    .dat_REG(dat_REG),  .cb_byte(cb_byte_tx),  //Счетчик байт
                   .T_start(T_start),          //Старт передачи
                   .T_stop(T_stop),            //Стоп передачи
                   .ce_byte(ce_byte_tx),       //Конец байта
                   .sr_rx_SDA(sr_rx_SDA));//Рег. Сдвига принимаемых данных

SLAVE_I2C DD2 ( .SDA(SDA),//Физический сигнал SDA ведомого
    .SCL(SCL),     .ce_start(ce_start_rx),
    .clk(clk),     .en_rx(en_rx),
    .Adr_SLAVE(Adr_SLAVE),.R_W(R_W),
                   .SDA_SLAVE(SDA_SLAVE),
                   .cb_bit(cb_bit_rx),
                   .ce_stop(ce_stop_rx),
                   .sr_rx(sr_rx),
                   .sr_tx(sr_tx),
                   .ok_rx_byte(ok_rx_byte),
                   .T_AC(AC_rx),
                   .cb_byte(cb_byte_rx),
                   .my_adr(my_adr),
                   .my_reg(my_reg),
                   .dat_MEM(dat_MEM));
endmodule
```

Эта схема соответствует первым шагам проектирования модуля ведомого, а именно созданию устройств формирования сигналов: ce_start, ce_stop, front_SCL, spad_SCL,… Остальные порты схемы **Test_ MASTER_SLAVE_I2C,** модуля **SLAVE_I2C**, а также некоторые порты модуля **MASTER_I2C** временно закомментированы.

5.4.2 Пример содержательной части (Verilog Test Fixture) задания **tf_Test_Sch_Lab408** на моделирование схемы **Test_Sch_Lab408**

```
always begin clk=0 ; #10 clk=1; #10 ; end
    initial begin
            st = 0;     Adr_SLAVE=7'h00;   ADR_COM=8'h00;    adr_REG=8'h00;    dat_REG=8'h00;
        //Сеанс записи байта 8'hA5 по адресу 8'hE2
#100; st = 1;     Adr_SLAVE=7'h40;   ADR_COM=8'h80;    adr_REG=8'hE2;   dat_REG=8'hA5;
#20;   st = 0;
        //Сеанс чтения байта по адресу 8'hE2
#350000;st = 1; Adr_SLAVE=7'h40;  ADR_COM=8'h81;    adr_REG=8'hE2;   dat_REG=8'hA5;
#20;        st = 0;
    end
```

Задержка повторного запуска должна быть больше длительности передачи трех байт. Например, для скорости Fvel=100 kbit/s Tdel=#350000 (350 мкс). В этом задании при первом запуске ведущий записывает байт в блок регистров ведомого, а при втором запуске считывает его.

5.5 Зарисовать временные диаграммы сигналов отлаженного модуля SLAVE_I2C.

6. Задание к сдаче работы (стоимость 3)

6.1 Создать символ модуля MASTER_I2C (Dsign Utilites – Create Schematic Simbol).

6.2 Создать символ созданного и отлаженного модуля SLAVE_I2C.

6.3 Создать модуль и символ ADR_COM_DAT_BL загрузки данных для MASTER_I2C через COM порт ПК программой ComChange (см. приложения 7.1 (6.1.1, 6.1.2, 6.1.3), 6.5).

6.4 Создать модуль и символ TXD_RET_BL чтения данных модулей MASTER_I2C и SLAVE_I2C через COM порт ПК программой ComChange (см. приложения 7.2(7.2.1, 7.2.2)).

6.5 Создать модуль и символ Display отображения данных на семи сегментном светодиодном индикаторе макета NEXYS2 (см. приложение 7.3).

6.6 Создать лист схемы (Schematic) S_Sch_Lab408 или модуль V_Sch_Lab408. Из созданных модулей (для V_Sch_Lab408) или символов ADR_COM_DAT_BL (для S_Sch_Lab408), TXD_RET_BL, MASTER_I2C, SLAVE_I2C и Display составить модуль (приложение 7.6) или лист схемы (рис. 8) лабораторной работы. Для составленной схемы S_Sch_Lab408 или V_Sch_Lab408 создать (Implementation Constraints File) файл Sch_Lab408.ucf (см. приложение 7.4).

6.7 Соединить макет с USB и COM портом ПК. Соединить перемычками SCL и SDA ведущего и ведомого (JB1<->JC1, JB2<->JC2). Создать файл конфигурации. Загрузить в макет. При помощи программы ComChange (см. приложение 6.5) продемонстрировать работу макета в соответствии с заданным вариантом задания.

6.8 Подключить к макету осциллограф. Получить и сохранить осциллограммы сигналов SCL, SDA. Для внешнего запуска развертки осциллографа можно, например, использовать сигнал en_tx или en_rx, выведенные на порты JA3 и JC7 макета.

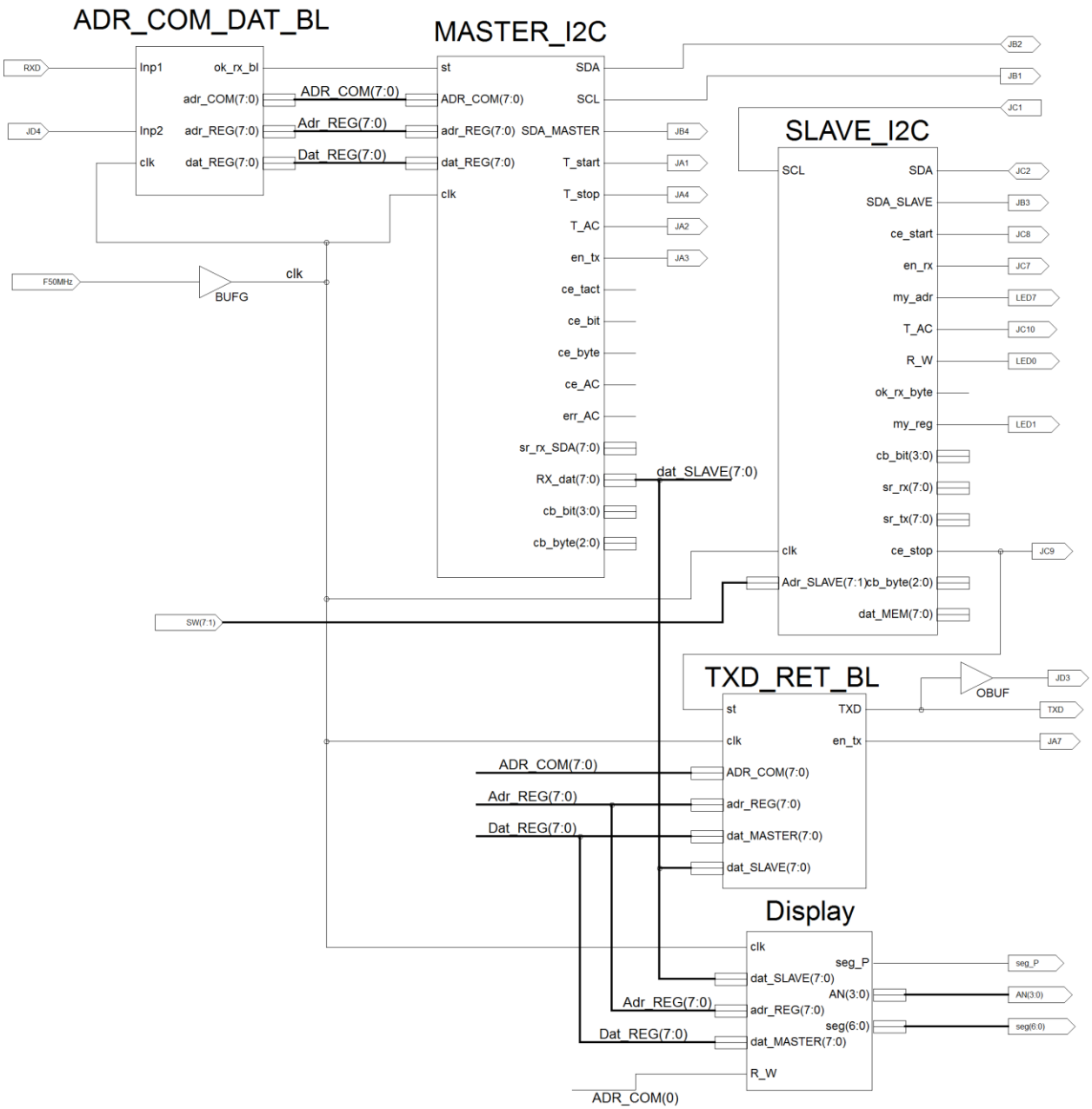

Рис. 8 Лист схемы лабораторной работы Lab408

В этой схеме адрес ведомого задается переключателями SW[7:1] макета.

Сеанс связи запускается импульсом st с выхода модуля ADR_COM_DAT-BL после поступления данных ADR_COM, Adr_REG, Dat_REG от COM порта ПК.

Связь SCL и SDA модулей ведущего и ведомого осуществляется внешними проводными перемычками (JB1<->JC1, JB2<->JC2).

В приложении 7.6 приведен текстовый вариант этой схемы, написанный на языке Verilog.

7. Приложения

7.1 Модуль **ADR_COM_DAT_BL** загрузки данных для MASTER_I2C через COM порт ПК программой ComChange

Этот модуль имеет два входа для приема последовательных данных COM порта ПК:

- In1 – от входа RXD макета MEXYS2 с выхода TXD DSUB-9 COM порта ПК,
- In2 – или с выхода TXD преобразователя интерфейса “USB to TTL”.

При использовании “USB to TTL” надо в свойствах ПК (Диспетчер устройств/Порты (COM и LPT)) узнать его номер COM порта и установить этот номер в настройках программы ComChange.

Модуль **ADR_COM_DAT_BL** принимает из COM порта ПК три байта предназначенные для модуля MASTER_I2C:

- первый байт – ADR_COM (адрес/команда),
- второй байт – Adr_REG (адрес регистра),
- третий байт – Dat_REG (данные для ведомого).

Сигнал ok_rx_bl окончания приема блока байт используется для запуска модуля MASTER_I2C после окончания приема байт из COM порта ПК.

```
`include "CONST.v"
module ADR_COM_DAT_BL(
        input Inp1,     output reg [7:0] adr_COM=0, //Адрес.команда
        input Inp2,     output reg [7:0] adr_REG=0, //Адрес регистра
        input clk,      output reg [7:0] dat_REG=0, //Данные регистра
                    output wire ok_rx_bl);   //
wire Inp=Inp1 & Inp2 ;
//--------Приемник строки байт------------------------------
reg [11:0] cb_tact ;              //Счетчик длительности такта (бита)
wire ce_tact = (cb_tact==`UART_Nt) ;   //Tce_tact=1/UARTvel
wire ce_bit = (cb_tact==(`UART_Nt/2));//Середина такта
reg [3:0]cb_bit=0 ;               //Счетчик бит в кадре UART
reg [7:0]cb_res=0 ;          //Счетчик паузы
reg [7:0]cb_byte=0 ;         //Счетчик принятых байт
reg [7:0]rx_dat=0 ;          //Принятый байт
reg en_rx_byte=0, en_rx_bl=0 ;
reg RXD=0, tRXD=0 ; //

wire dRXD = !RXD & tRXD ;       //"Спады" входного сигнала RXD
wire ok_rx_byte = (ce_bit & (cb_bit==9) & en_rx_byte & tRXD); //Успешный прием байта
wire start_rx_byte = dRXD & !en_rx_byte ;     //Старт приема очередного байта
assign ok_rx_bl = (cb_res==10) &  ce_tact ; /*Успешный прием блока байт (по паузе в 10
тактов между байтами)*/
wire T_dat = (cb_bit<9) & (cb_bit>0);
```

```
always @ (posedge clk) begin
RXD <= Inp ; tRXD <= RXD ;
cb_tact <= ((dRXD & !en_rx_byte) | ce_tact)? 1 : cb_tact+1;
cb_bit <= (start_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_dat)? 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 & en_rx_bl)? cb_res+1 : cb_res ;
en_rx_bl <= start_rx_byte? 1 : ok_rx_bl? 0 : en_rx_bl ;
end
//---Адреса регистров-------------------------
wire T_adr_COM = (cb_byte==0) ;
wire T_adr_REG = (cb_byte==1) ;
wire T_dat_REG = (cb_byte==2) ;
//---Загрузка регистров адреса, команды и данных
always @ (posedge clk) begin
adr_COM <= (T_adr_COM & ok_rx_byte)? rx_dat : adr_COM ;
adr_REG <= (T_adr_REG & ok_rx_byte)? rx_dat : adr_REG ;
dat_REG <= (T_dat_REG & ok_rx_byte)? rx_dat : dat_REG ;
end
endmodule
```

7.2 Модуль **TXD_RET_BL** чтения данных модулей MASTER_I2C и SLAVE_I2C через COM порт ПК программой ComChange

Этот модуль по сигналу ce_end модуля SLAVE_I2C возвращает в COM порт ПК байты модуля ведущего MASTER_I2C:

- первый байт – ADR_COM (адрес/команда),
- второй байт – adr_REG (адрес регистра),
- третий байт:
  ADR_COM[0]=0 - dat_REG (данные мастера для ведомого),
  ADR_COM[0]=1 - dat_SLAVE (данные от ведомого).

```
`include "CONST.v"
module TXD_RET_BL(          output wire [7:0]tx_dat,
      input clk,          output wire TXD, //Последовательные данные
      input st,         output reg en_tx =0, //Интервал передачи
      input [7:0] ADR_COM,    //Адрес.команда
      input [7:0] adr_REG,      //Адрес регистра
      input [7:0] dat_MASTER, //Данные мастера
      input [7:0] dat_SLAVE ); //Данные ведомого

parameter N_byte = 3 ; //Число байт
//------Передатчик ответа-----------------------------------
```

```
reg [8:0] cb_tact ; //Счетчик длительности такта (бита)
wire ce_tact = (cb_tact==`UART_Nt) ;       //Tce_tact=1/UARTvel
reg [3:0] cb_bit=0 ;                  //Счетчик бит байта
reg [7:0] sr_dat=0 ;                  //Регистр сдвига бит байта
reg [2:0] cb_byte =0;                    //Счетчик передаваемых байт
wire  T_start = ((cb_bit==0) & en_tx) ;       //Интервал старт бита
wire  T_dat = (cb_bit<9) & (cb_bit>0);       //Интервал 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 TXD = T_start? 0 : en_tx?  sr_dat[0] : 1 ;   //TXD блока байт

//--------Функциональное назначение байт ответа
assign tx_dat =(cb_byte==0)? ADR_COM :     //Команда
       (cb_byte==1)? adr_REG :     //Адрес регистра
                ((cb_byte==2) & !ADR_COM[0])? dat_MASTER :  //Данные мастера
       ((cb_byte==2) &  ADR_COM[0])? dat_SLAVE : 8'hFF ;   //Данные ведомого
always @ (posedge clk) begin
cb_tact <= (st & !en_tx | 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 : (en_sh & ce_tact)? sr_dat>>1 | 1<<7 : sr_dat ;
en_tx <= st? 1 : ((cb_byte==N_byte-1) & ce_stop)? 0 : en_tx ;
end
endmodule
```

7.3 Модуль Display отображения данных на семи сегментном светодиодном индикаторе макета NEXYS2

Этот модуль отображает на левых двух цифрах индикатора адрес регистра, а на правых двух цифрах: по команде записи - данные для ведомого (Dat_REG), а по команде чтения – данные от ведомого (Dat_SLAVE).

```
module Display(
        input clk,          output wire[3:0] AN,    //Аноды
        input[7:0]adr_REG,       output wire[6:0] seg,    //Сегменты
        input[7:0]dat_MASTER,    output wire seg_P,   //Точка
        input[7:0]dat_SLAVE,   //Данные ведомого
        input R_W);         //Команда

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) ;

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)
//-------Мультиплексор данных для индикатора
wire [15:0] dat = R_W? {adr_REG,dat_SLAVE} : {adr_REG,dat_MASTER} ;
//-------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];
//-------Семисегментный 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
```

7.4 Размещение портов схем Sch_Lab408 на выводах ПЛИС xc3s500e-5fg32
(не используемые выводы ПЛИС закомментированы символом #)

```
#----Сигнал генератора синхронизации
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
#---Кнопки
#NET "BTN0"  LOC = "B18"  ; #BTN3
#NET "BTN1"  LOC = "D18"  ; #BTN2
#NET "BTN2"  LOC = "E18"  ; #BTN1
#NET "BTN3"  LOC = "H13"  ; #BTN0
#---Светодиоды
NET "LED0"  LOC = "J14" ; #LD0
NET "LED1"  LOC = "J15" ; #LD1
#NET "LED<2>"  LOC = "K15" ; #LD2
#NET "LED<3>"  LOC = "K14" ; #LD3
#NET "LED<4>"  LOC = "E17" ; #
#NET "LED<5>"  LOC = "P15" ; #
 #NET "LED<6>"  LOC = "F4"  ; #
NET "LED7"  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 "RXD"  LOC = "U6"  ; #TXD U6
NET "TXD"  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" | PULLUP ; #SCL MASTER
NET "JB2"  LOC = "R18" ; #SDA_MASTER
NET "JB3"  LOC = "R15" ; #SDA_SLAVE
NET "JB4"  LOC = "T17" ; #SDA_MASTER
#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"   ; #SCL SLAVE
NET "JC2" LOC = "J16"   ; #SDA_SLAVE
#NET "JC3"  LOC = "G13"  ; #Pin3
#NET "JC4"  LOC = "H16"  ; #Pin4
NET "JC7" LOC = "H15"   ; #en_rx
NET "JC8" LOC = "F14"   ; #ce_start
NET "JC9" LOC = "G16"   ; #ce_stop
NET "JC10" LOC = "J12"  ; #T_AC
#---Выводы порта JD
#NET "JD1"  LOC = "J13"  ; #Pin1
#NET "JD2"  LOC = "M18"  ; #Pin2
NET "JD3"  LOC = "N18"   ; #TXD (USB-COM)
NET "JD4"  LOC = "P18"   ; #RXD (USB-COM)
#NET "JD7"   LOC = "K14"  ; #LED0
#NET "JD8"   LOC = "K15"  ; #LED1
#NET "JD9"  LOC = "J15"  ; #LED2
#NET "JD10" LOC = "J14"  ; #LED3
```

7.5 Программа **ComChange**

Эта программа выводит в COM порт с заданным номером и с заданной скоростью строку байт. К введенной строке байт программа может добавлять два байта контрольного кода CRC-16. В модулях ADR_COM_DAT и TXD_RET_BL контрольный код CRC не используется.

При вводе воспринимаются только HEX цифры, пробелы между байтами вводятся автоматически. Параметры вывода устанавливаются в окне «Настройка». Обмен запускается кнопкой «Обмен»

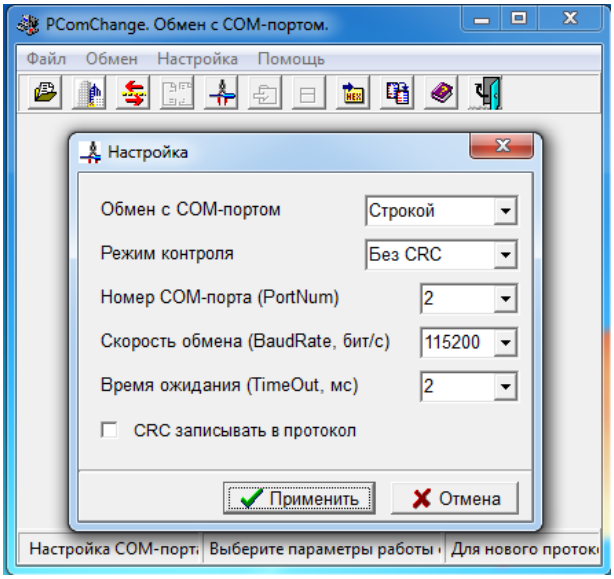

Рис. 9 Настройка ComChange

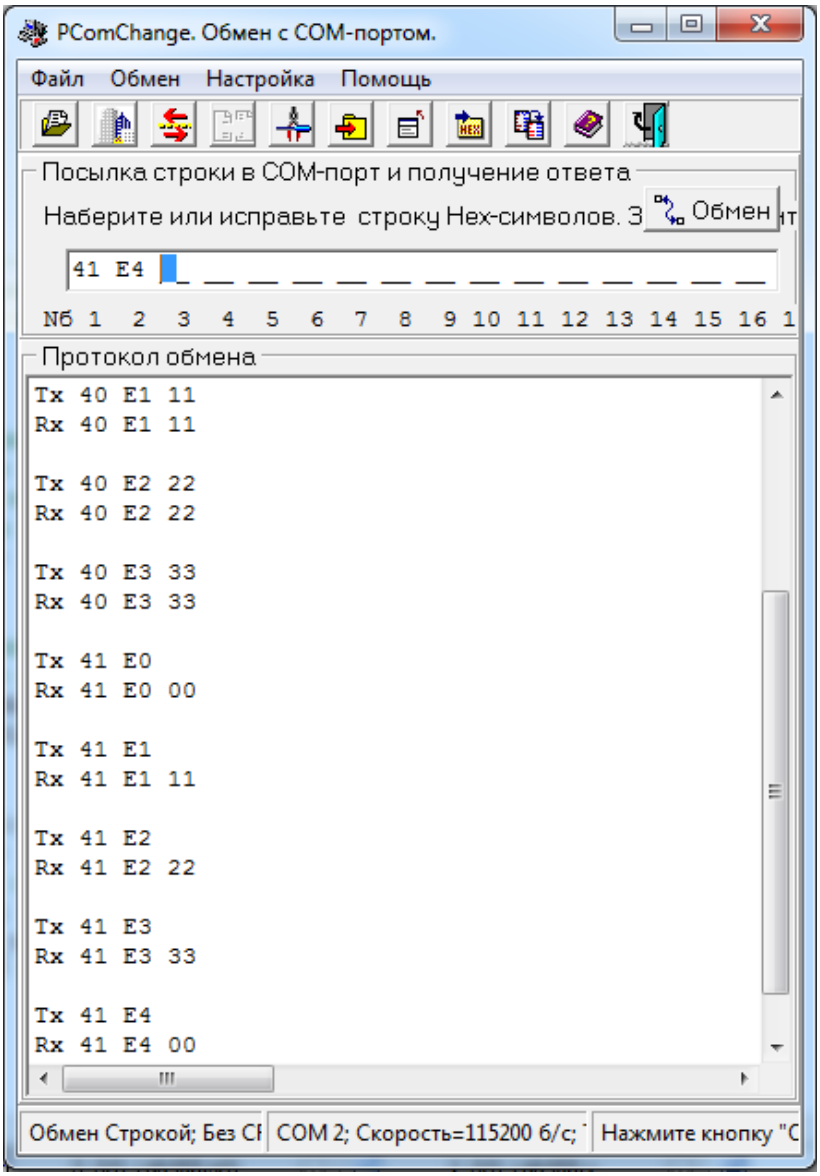

Рис. 10 Пример диалогов ComChange и макета с Lab_408

Передаваемая последовательность байт отображается в стоке “Tx”. После вывода строки (кнопка «Обмен») ComChange ожидает ответа и если ответа долго (>TimeOut) нет, выводится сообщение “Rx Нет ответа”. Если ответ есть, то в ответной строке Rx выводятся принятые байты.

В этом примере в первых трех сеансах обмена: 40 E1 11, 40 E2 22, 40 E3 33 ведомому с адресом 40 его трем регистрам с адресами E1, E2, E3 записываются данные 11, 22, 33.

В следующих 5 сеансах обмена считываются данные 5 регистров ведомого с адресами E0, E1, E2, E3, E5.

7.6 Текстовый вариант схемы лабораторной работы Lab_408

```
module V_Sch_Lab408(  inout JB2,  //SDA MASTER
                      inout JC2,  //SDA_SLAVE
        input JC1,      output wire JB1,  //SCL MASTER/SLAVE
        input F50MHz,
//--------------------------USB COM port
        input JD4,      output wire JD3,
//--------------------------RS232 COM port
        input RXD,     output wire TXD,
        input[7:1]SW                  //Адрес/команда
                    output wire LED0,   //R_W
                    output wire LED1,   //my_reg
                    output wire LED7,   //my_adr
//-----------------------------------------DISPLAY--------
                    output wire [3:0]AN,    //Аноды дисплея
                    output wire [6:0]seg,    //Сегменты дисплея
                    output wire seg_P,   //Точка
//-------------------------------------------------------------------------------------------
                    output wire JA1,  //T_start
                    output wire JA2,  //T_AC
                    output wire JA3,  //en_tx
                    output wire JA4,  //T_stop
                    output wire JA7,  //en_tx_bl

                    output wire JB3,  //SDA_SLAVE
                    output wire JB4,  //SDA_MASTER

                    output wire JC7,  //en_rx
                    output wire JC8,  //ce_start
                    output wire JC9,  //ce_stop
                    output wire JC10 );  //T_AC

wire clk, st ;
wire [7:0]ADR_COM;      wire [7:0]adr_REG; wire[7:0]dat_REG ;
wire [7:0]dat_SLAVE ;     wire [2:0]N_byte ;
```

```
assign JD3=TXD ; //USB-COM_port
//---Глобальный буфер сигнала синхронизации
BUFGDLL DD1 (.I(F50MHz),      .O(clk));

//---Модуль приема данных от ПК через COM порт
ADR_COM_DAT_BL DD2 (  .clk(clk), .adr_COM(ADR_COM),
                .Inp1(RXD),.adr_REG(adr_REG),
                .Inp2(JD4),  .dat_REG(dat_REG),
                        .ok_rx_bl(st));
//---Модуль I2C мастера
MASTER_I2C DD3 (          .SDA(JB2),//Физический сигнал SDA мастера
    .clk(clk), .SCL(JB1),  //Сигнал SCL мастера
    .ADR_COM(ADR_COM),   .RX_dat(dat_SLAVE),//Регистр данных от ведомого
    .adr_REG(adr_REG),      .SDA_MASTER(JB4),    //Логический сигнал SDA мастера
    .dat_REG(dat_REG),      .T_start(JA1),        //Старт передачи
    .st(st),                .T_AC(JA2),          //Такт подтверждения приема байта
                    .en_tx(JA3),     //Регистр разрешения передачи
                    .T_stop(JA4));       //Стоп передачи
//---Модуль I2C ведомого
SLAVE_I2C DD4 (   .SDA(JC2), //Физический сигнал SDA ведомого
        .SCL(JC1),       .ce_start(JC8),
        .clk(clk),      .en_rx(JC7),
        .Adr_SLAVE(SW),    .SDA_SLAVE(JB3),
                    .T_AC(JC10),
                    .R_W(LED0),
                    .my_reg(LED1),
                    .my_adr(LED7),
                    .ce_stop(JC9) );
//---Модуль чтения данных модулей MASTER_I2C и SLAVE_I2C через COM порт ПК
//---программой ComChange
TXD_RET_BL DD5(.clk(clk),           .TXD(TXD),
            .st(JC9),          .en_tx (JA7),
            .ADR_COM(ADR_COM),
            .adr_REG(adr_REG),
            .dat_MASTER(dat_REG),
            .dat_SLAVE(dat_SLAVE) );
//---Модуль семи сегментного индикатора
Display DD6 (.clk(clk),             .AN(AN),   //Аноды
        .adr_REG(adr_REG),        .seg(seg),    //Сегменты
        .dat_MASTER(dat_REG),     .seg_P(seg_P),//Точка
        .dat_SLAVE(dat_SLAVE),
        .R_W(ADR_COM[0]));             //Команда
endmodule
```