Университет ИТМО

Факультет программной инженерии
и компьютерной техники

Кафедра вычислительной техники

Системы ввода-вывода и периферийные устройства

Лабораторная работа №1

«Разработка контроллера ввода/вывода»

Выполнили:
Милосердов А. О.
Калугин Ф. И.
Группа P3410

Преподаватель:
Быковский С. В.

Санкт-Петербург
2017 г.

Содержание

1. Описания задания

Для интерфейса SPI разработать модели контроллера ввода-вывода на уровне регистровых передач с использованием языка Verilog и в виде поведенческой модели с использованием языка SystemC.

Разработать тестовое окружение для каждой модели с целью проверки корректности работы контроллера. Провести тестирование обеих моделей, убедиться в корректности реализации и соответствии моделей друг другу.

2. Краткая характеристика интерфейса

SPI (Serial Peripheral Interface) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса. В SPI используются четыре цифровых сигнала:

- MOSI — выход ведущего, вход ведомого (англ. Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.
- MISO — вход ведущего, выход ведомого (англ. Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.
- SCLK — последовательный тактовый сигнал (англ. Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
- CS или SS — выбор микросхемы, выбор ведомого (англ. Chip Select, Slave Select).

Передача осуществляется пакетами. Длина пакета, как правило, составляет 1 байт (8 бит). Ведущее устройство инициирует цикл связи установкой низкого уровня на выводе выбора подчиненного устройства (SS) того устройства, с которым необходимо установить соединение.

Подлежащие передаче данные ведущее и ведомое устройства помещают в сдвиговые регистры. Передача данных осуществляется бит за битом от ведущего по линии MOSI и от ведомого по линии MISO. После передачи каждого пакета данных ведущее устройство, в целях синхронизации ведомого устройства, переводит линию SS в высокое состояние.

3. Описание разработанного контроллера

3.1. Описание режима работы и особенностей

Разработанный контроллер работает только в режиме Master, mode 0 (CPOL = 0, CPHA = 0). Частота синхросигнала в 4 раза меньше основной частоты тактового сигнала. Синхросигнал генерируется всё время активности контроллера (как, например, в ATE SPI). Работает на одном сдвиговом регистре.

3.2. Описание интерфейсов

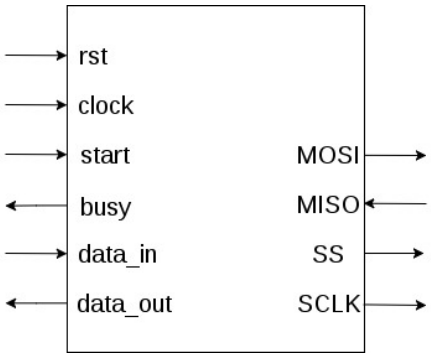

Рис. 1: Интерфейсы контроллера

На данный момент разработанный контроллер работает с одним ведомым устройством. Контроллер предоставляет следующие входы и выходы:

- rst – сигнал сброса состояния контроллера;
- clock – тактовый сигнал для работы контроллера:
- start – сигнал инициализации транзакции;
- busy – сигнал, означающий, что транзакция ещё не закончена;
- data_in – регистр, содержащий пакет на отправку;
- data_out – регистр для хранения полученного пакета от ведомого устройства.

4. Тестирование контроллера

В качестве тестирования было проведено две штатные транзакции и одна, прерванная сигналом сброса. Данные на отправку, сигнал инициализации транзакции и отправка данных по mosi осуществляется тестовым устройством.

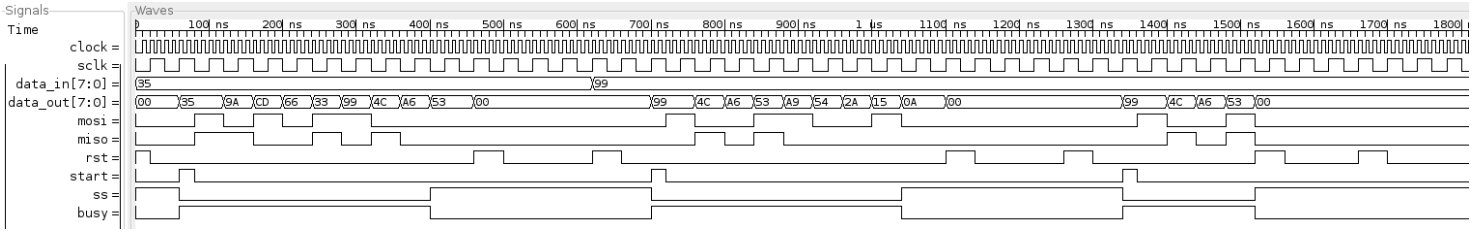

Рис. 2: Временные диаграммы SystemC

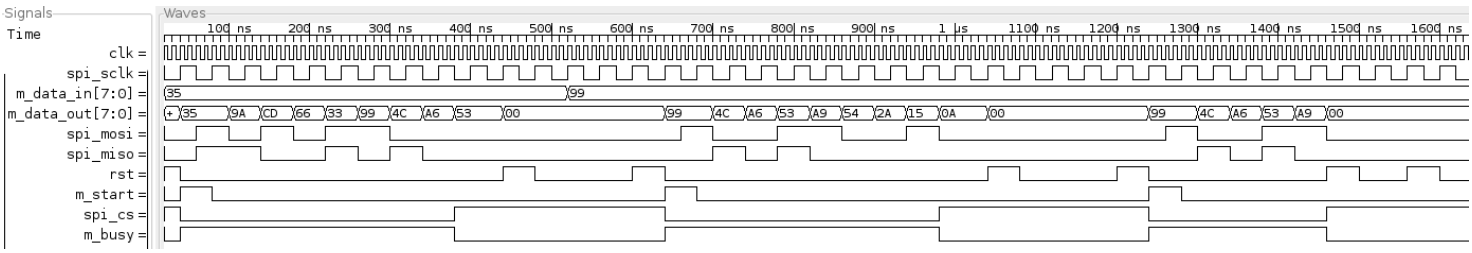

Рис. 3: Временные диаграммы Verilog

5. Исходные тексты

5.1. SystemC

Листинг 1: Объявление контроллера

```
/*
  SPI module, working in mode 0 (CPOL = 0, CPHA = 0), master only, transieves 8 bits in 8 cycles.
  Ports:
    clk   -- main input clock
    sclk  -- synchronous clock for interaction between SPI modules. 4 times slower than clk.
    rst   -- reset signal, edge sensitive
    busy  -- indicates transaction in progress.
    ss    -- slave select. Is low for 8 cycles, set to high on last positive sclk edge of transaction.
    start -- signal to start transaction, edge sensitive.

    data_in/data_out  -- mosi/miso registers
    mosi/miso -- master out/master in wires.
                 Set to high on falling edge of sclk,
                 read happens on falling edge of sclk,
                 write to data_in/data_out on rising edge of sclk.
*/

#include "systemc.h"
#include "clock.h"

SC_MODULE( spi ) {
  sc_in<sc_uint<8> > data_in;
  sc_out<sc_uint<8> > data_out;

  // Shift register
  sc_uint<8> shiftreg;

  sc_uint<3> ctr;

  sc_in<bool> clk, rst, start, miso;
  sc_out<bool> sclk, ss, mosi, busy;

  // SPI clock to generate sclk
  clock_gen clk_gen;

  // Flag for transaction start
  bool trans_start;

  // Indicate last bit transmission
  bool last;

  // Flag indicating first tick of transaction
  bool first;

  bool buf;

  void rx( );
  void tx( );
  void reset( );
  void end_transaction( );
  void loop( );
```

```
  SC_CTOR( spi ):
  clk( "CLK" ), rst( "RST" ), miso( "MISO" ), sclk( "SCLK" ),
  ss( "SS" ), mosi( "MOSI" ), clk_gen( "CLK_GEN" ) {

    // Clock generator for sclk
    clk_gen.clock( clk );
    clk_gen.qclk( sclk );

    trans_start = 0;

    SC_METHOD( loop );
    sensitive << sclk << rst.pos( ) << start.pos( );
  }
};
```

Листинг 2: Логика контроллера

```
// SPI, mode 0, only master mode, transieve from 0th bit to 7th

// Ports are assigned using write and local vars are assigned using `=` just for readability.
#include "spi.h"

// SPI receive
void spi::rx( ) {
  //shiftreg[7] = miso.read( );
  buf = miso.read( );
//cout << "@" << sc_time_stamp( ) << " Receive miso/ctr: " << miso.read( ) << "/" << ctr << endl;
}

// SPI transmit
void spi::tx( ) {
  mosi.write( shiftreg[0] );
}

// On reset end transaction end reset output data
void spi::reset( ) {
  end_transaction( );
  data_out.write( 0 );

}

/* On transaction end:
    MOSI -- Low
    SS -- High
    Busy -- Low

    Reset counter, disable toggler and set last bit flag to 0
*/
void spi::end_transaction( ) {
  ss.write( 1 );
  ctr = 0;
  busy.write( 0 );

  mosi.write( 0 );

```

```
  trans_start = 0;
  last = 0;
  first = 1;
}

// Main SPI loop
void spi::loop( ) {

  trans_start = trans_start | ( start.read( ) & !busy.read( ) );

  if( rst ) {
    reset( );
    return;
  }

  // Assign shiftreg to input packet
  if( first && sclk && trans_start ) {
    shiftreg = data_in.read( );
  }

  // Main logic on every sclk tick
  if( sclk ) {
    if( trans_start ) {
      if( first ) {
        data_out.write( shiftreg );
        busy.write( 1 );
        ss.write( 0 );
      }
      rx( );
    }

  } else if( busy ) {


    if( !first ) {
      shiftreg = shiftreg >> 1;
      shiftreg[7] = buf;
    }

    tx( );

    if( !last && !first ) {
      ctr++;

    } else if( last ) {
      rx( );
      end_transaction( );
    }

    first = 0;

    if( ctr == 7 ) last = 1;

    data_out.write( shiftreg );
  }

```

```
}
```

Листинг 3: Тестовое устройство

```
#include "test_spi.h"

void wait_fall( ) {
  wait( 40, SC_NS );
}

void test_spi::test_send( uint8_t in, uint8_t out, bool reset ) {

  data_in.write( in );

  rst.write( 1 );

  wait( );
  rst.write( 0 );
  wait( );

  start.write( 1 );

  wait( 20, SC_NS );

  start.write( 0 );

  msg = out;

  for( counter = 0; counter < 8; counter++ ) {
    if( counter >= 4 && reset ) {
      miso.write( 0 );
      rst.write( 1 );
      wait_fall( );
      rst.write( 0 );
      wait_fall( );
      break;
    } else {
      miso.write( msg << 7 & 0xFF );
      msg = msg >> 1;
      wait_fall( );
    }
  }

  wait( );
  wait( );

  // Reset data_out
  rst.write( 1 );
  wait( );
  rst.write( 0 );

  for ( int i = 0; i < 3; ++i ) {
    wait( );
  }

```

```
}

void test_spi::demo_send( ) {

  test_send( 0b00110101, 0b01010011, false );
  test_send( 0b10011001, 0b00001010, false );
  test_send( 0b10011001, 0b00001010, true );

  sc_stop( );
}
```

5.2. Verilog

Листинг 4: Ведущий контроллера

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Simple SPI master controller with CPOL=0, CPHA=0
////////////////////////////////////////////////////////////////////////////////

module spi_master_driver(
    input             clk_i,
    input             rst_i,

    // system interface
    input             start_i,     // signal to start transaction
    input       [7:0] data_in_bi,  // data that master will write to slave
    output reg        busy_o,      // transaction is being processed
    output reg  [7:0] data_out_bo, // data recevied from slave in last transaction

    // SPI interface
    input             spi_miso_i,
    output reg        spi_mosi_o,
    output reg        spi_sclk_o = 0,
    output reg        spi_cs_o
    );

    reg   [3:0] counter;
    reg   clk_div2 = 1;
    reg   mosi_enable;
    reg   [7:0] shiftreg;
    reg         bit_buffer;

    localparam STATE_IDLE              = 0; // wait for transaction begin
    localparam STATE_WAIT_SCLK_1       = 1; // wait for SCLK to become 1
    localparam STATE_WAIT_SCLK_0       = 2; // wait for SCLK to become 0
    localparam STATE_WAIT_SCLK_0_START = 3; // wait for SCLK to become 0, no shift


    reg   [2:0] state;

    always @(posedge clk_i) begin
        if (clk_div2) begin
            spi_sclk_o = !spi_sclk_o;
        end
        clk_div2 <= !clk_div2;
```

```
        if (rst_i) begin
            counter     <= 0;
            shiftreg    <= 0;
            bit_buffer  <= 0;
            data_out_bo <= 0;
            spi_mosi_o  <= 0;
            state       <= STATE_IDLE;
            mosi_enable <= 0;
        end
        else begin
            case (state)
                STATE_IDLE: begin
                    if (start_i) begin
                        shiftreg <= data_in_bi;
                        mosi_enable <= 0;
                        state <= STATE_WAIT_SCLK_0_START;
                    end
                end
                STATE_WAIT_SCLK_1: begin
                    if (spi_sclk_o == 1) begin
                        bit_buffer <= spi_miso_i;
                        state <= STATE_WAIT_SCLK_0;
                    end
                end
                STATE_WAIT_SCLK_0: begin
                    if (spi_sclk_o == 0) begin
                        shiftreg <= { bit_buffer, shiftreg[7:1] };
                        state <= STATE_WAIT_SCLK_1;
                        if (counter == 7) begin
                            mosi_enable <= 0;
                            state <= STATE_IDLE;
                            counter <= 0;
                            mosi_enable <= 0;
                        end else begin
                            counter <= counter + 1;
                        end
                    end
                end
                STATE_WAIT_SCLK_0_START: begin
                    if (spi_sclk_o == 0) begin
                        mosi_enable <= 1;
                        state <= STATE_WAIT_SCLK_1;
                    end
                end
                default: begin
                    state <= STATE_IDLE;
                end
            endcase
        end
    end

    always @* begin
        busy_o = (state != STATE_IDLE);
        spi_cs_o = (state == STATE_IDLE);

        data_out_bo = shiftreg;
```

```
98
99          spi_mosi_o = shiftreg[0] && mosi_enable;
100     end
101
102 endmodule
```

Листинг 5: Ведомый контроллер

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////
3   // Simple SPI slave controller with CPOL=0 CPHA=0
4   //////////////////////////////////////////////////////////////////////////////
5
6   module spi_slave_driver(
7       input            clk_i,
8       input            rst_i,
9
10      // system interface
11      input      [7:0] data_in_bi,  // data that master can read from slave
12      output reg       ready_o,     // transaction is not processed now
13      output reg [7:0] data_out_bo, // data written to slave in last transaction
14
15      // SPI iterface
16      output reg       spi_miso_o,
17      input            spi_mosi_i,
18      input            spi_sclk_i,
19      input            spi_cs_i
20      );
21
22      reg   [3:0] counter;
23      reg   [7:0] shiftreg;
24      reg         bit_buffer;
25      reg         miso_enable;
26      localparam STATE_IDLE = 0; // wait for transaction begin
27      localparam STATE_WAIT_SCLK_1 = 1; // wait for SCLK to become 1
28      localparam STATE_WAIT_SCLK_0 = 2; // wait for SCLK to become 0
29      localparam STATE_WAIT_SCLK_0_START = 3; // wait for SCLK to become 0
30
31      reg   [2:0] state;
32
33      always @(posedge rst_i, posedge clk_i) begin
34          if (rst_i == 1) begin
35              shiftreg <= 0;
36              bit_buffer <= 0;
37
38              data_out_bo <= 0;
39              counter     <= 0;
40              spi_miso_o <= 0;
41              miso_enable <= 0;
42              state <= STATE_IDLE;
43          end
44          else begin
45              if (spi_cs_i) begin
46                  state <= STATE_IDLE;
47              end
48              else begin
```

```
                case (state)
                    STATE_IDLE: begin
                        if (!spi_cs_i) begin
                            miso_enable <= 0;
                            counter     <= 0;
                            state <= STATE_WAIT_SCLK_0_START;
                        end
                    end
                    STATE_WAIT_SCLK_1: begin
                        if (spi_sclk_i) begin
                            bit_buffer <= spi_mosi_i;

                            state <= STATE_WAIT_SCLK_0;
                        end
                    end
                    STATE_WAIT_SCLK_0: begin
                        if (spi_sclk_i == 0) begin
                            shiftreg <= { bit_buffer, shiftreg[7:1] };
                            miso_enable <= 1;
                            state <= STATE_WAIT_SCLK_1;
                            if (counter == 7)
                                miso_enable <= 0;
                            counter <= counter + 1;
                        end
                    end
                    STATE_WAIT_SCLK_0_START: begin
                        if (spi_sclk_i == 0) begin
                            shiftreg <= data_in_bi;
                            miso_enable <= 1;
                            state <= STATE_WAIT_SCLK_1;
                        end
                    end
                    default: begin
                        state <= STATE_IDLE;
                    end
                endcase
            end
        end
    end

    always @* begin
        ready_o = (state == STATE_IDLE);
        data_out_bo = shiftreg;

        spi_miso_o = shiftreg[0] && miso_enable && !spi_cs_i;
    end

endmodule
```

Листинг 6: Тестовое окружение

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// SPI controllers testbench
////////////////////////////////////////////////////////////////////////////////

```

```
module spi_drivers_tb(
    );

    localparam CLK_HALFPERIOD = 5;

    reg         clk, rst;

    reg         m_start;
    reg   [7:0] m_data_in;
    wire        m_busy;
    wire  [7:0] m_data_out;

    reg   [7:0] s_data_in;
    wire        s_ready;
    wire  [7:0] s_data_out;

    wire        spi_miso;
    wire        spi_mosi;
    wire        spi_sclk;
    wire        spi_cs;

    spi_master_driver dut_m(
        .clk_i(clk),
        .rst_i(rst),

        .start_i(m_start),
        .data_in_bi(m_data_in),
        .busy_o(m_busy),
        .data_out_bo(m_data_out),

        .spi_miso_i(spi_miso),
        .spi_mosi_o(spi_mosi),
        .spi_sclk_o(spi_sclk),
        .spi_cs_o(spi_cs)
    );

    spi_slave_driver dut_s(
        .clk_i(clk),
        .rst_i(rst),

        .data_in_bi(s_data_in),
        .ready_o(s_ready),
        .data_out_bo(s_data_out),

        .spi_miso_o(spi_miso),
        .spi_mosi_i(spi_mosi),
        .spi_sclk_i(spi_sclk),
        .spi_cs_i(spi_cs)
    );

    always #(CLK_HALFPERIOD) clk = !clk;

    initial begin
        clk = 1;
        rst = 1;
        m_start = 0;
```

```
62
63          m_data_in = 8'b00110101;
64          s_data_in = 8'b01010011;
65
66          #20;
67          $dumpfile ("spi_verilog.vcd");
68          $dumpvars;
69
70          #20;
71
72          rst = 0;
73          m_start = 1;
74          #40;
75          m_start = 0;
76          #360;
77          rst = 1;
78          #40;
79          rst = 0;
80          #40;
81          m_data_in = 8'b10011001;
82          s_data_in = 8'b00001010;
83          #80;
84          rst = 1;
85          #40;
86
87          rst = 0;
88          m_start = 1;
89          #40;
90          m_start = 0;
91          #360;
92          rst = 1;
93          #40;
94          rst = 0;
95          #40;
96          m_data_in = 8'b10011001;
97          s_data_in = 8'b00001010;
98          #80;
99          rst = 1;
100         #40;
101
102         rst = 0;
103         m_start = 1;
104         #40;
105         m_start = 0;
106         #180;
107         rst = 1;
108         #40;
109         rst = 0;
110         #60;
111         rst = 1;
112         #40;
113         rst = 0;
114
115         #1000;
116                 $finish;
117     end
```

```
118
119 endmodule
```