Университет ИТМО

Факультет программной инженерии
и компьютерной техники

Кафедра вычислительной техники

Системы ввода-вывода и периферийные устройства

Лабораторная работа №3

«Разработка высокоуровневой модели микропроцессорной системы»

Вариант №6

PmodJSTK

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

Преподаватель:
Быковский С. В.

Санкт-Петербург
2017 г.

Содержание

1 Описание работы **2**

2 Описание функциональности реализуемой системы **2**

3 Описание функций и регистровая карта разработанного контроллера периферийного устройства **2**

3.1 Регистровая карта 2

3.2 Временные диаграммы шины AHB и периферийного интерфейса 3

4 Листинги **3**

5 Временные диаграммы **9**

6 Отладочный вывод **9**

1. Описание работы

Разработать на языке Verilog синтезируемое описание контроллера интерфейса ввода-вывода (SPI) с интерфейсом AHB. Контроллер должен подключаться к шине AHB процессора MIPSfpga. Подключить разработанный контроллер к коммутатору шины в шаблоне проекта. Провести тестирование контроллера периферийного интерфейса с использованием имитатора ведущего устройства шины AHB. Получить временные диаграммы обмена по шине AHB и периферийному интерфейсу.

2. Описание функциональности реализуемой системы

Основная функция контроллера — управление обменом с PmodJSTK по интерфейсу SPI. Обмен ведется пятью байтами в связи с форматом данных для данного устройства. Контроллер подключен к коммутатору шины AHB процессорного ядра MIPSfpga. В качестве системной шины используется упрощенный стандарт AMBA 3 AHB-Lite. Роль ведущего системной шины в разрабатываемой системе играет процессорное ядро, роль ведомых - контроллеры ввода-вывода.

3. Описание функций и регистровая карта разработанного контроллера периферийного устройства

В качестве периферийного устройства используется джойстик PmodJSTK:

- Джойстик, передающий 10 битные значения координат X и Y, и значения кнопок;
- 6-контактный порт с интерфейсом SPI.

Name	Slice LUTs (63400)	Slice Registers (126800)
ahb_lite_matrix (mfp_ahb_lite_matrix)	140	223
gpio (mfp_ahb_gpio_slave)	14	51
spi (ahb_spi)	42	42
spi (spi_master_driver)	37	24
uart_tx (ahb_uart_tx)	84	127

Рис. 1: Данные о занятых ресурсах ПЛИС (Implemented Design -> Report Utilization)

3.1. Регистровая карта

Адрес	Название	Биты	Назначение
0x40001000	SA_ST	[0]	начало обмена с периферийным устройством
0x40001004	SA_SS	[0]	управление сигналом Slave Select
0x40001008	SA_RE	[0]	сигнал состояния периферийного контроллера
0x40001012	SA_DA	[7:0]	данные

3.2. Временные диаграммы шины AHB и периферийного интерфейса

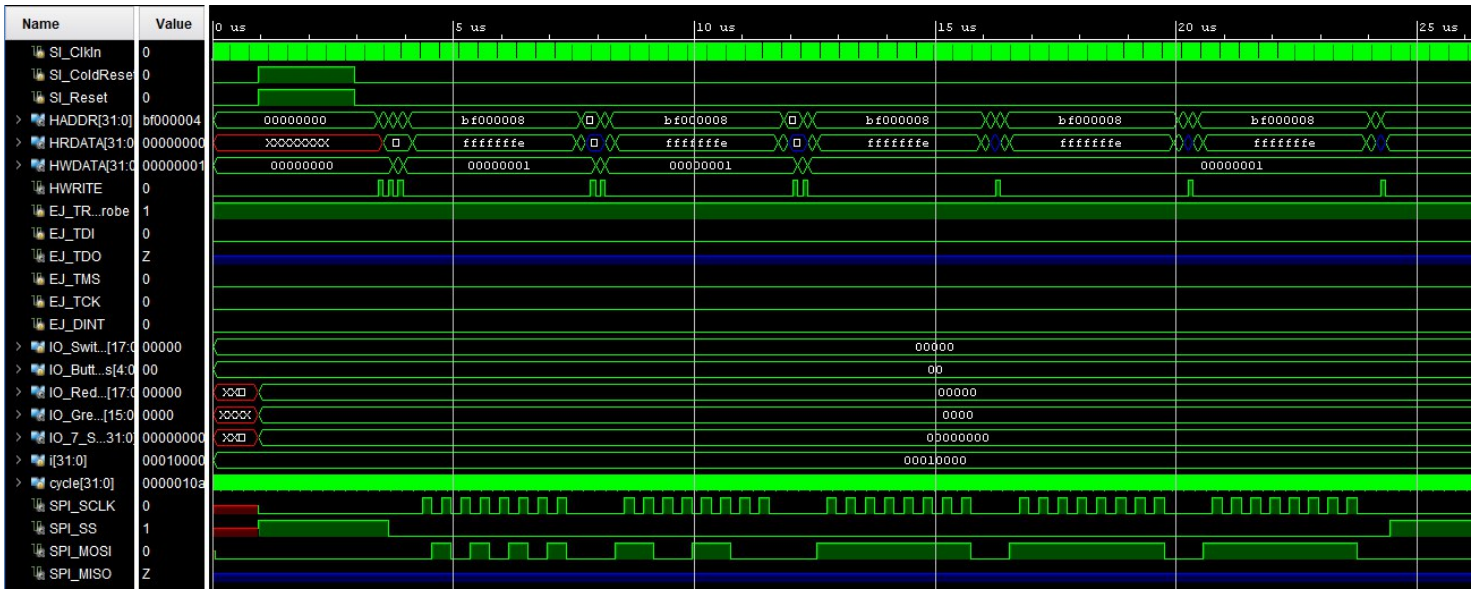

Рис. 2: Временная диаграмма обмена пятью байтами

4. Листинги

Листинг 1: ahb_spi.v

```
`timescale 100ps/1ps

module ahb_spi
(
    input               HCLK,
    input               HRESETn,
    input      [ 31: 0] HADDR,
    input      [ 31: 0] HWDATA,
    input               HWRITE,
    input               HSEL,
    output reg [ 31: 0] HRDATA,

    input               SPI_MISO,
    output              SPI_MOSI,
    output              SPI_SCLK,
    output reg          SPI_SS
);

    localparam START_REG_ADDR = 8'h0;
    localparam SS_REG_ADDR    = 8'h4;
    localparam READY_REG_ADDR = 8'h8;
    localparam DATA_REG_ADDR  = 8'h12;

    reg  [ 7:0] data_buf_w;
    wire [ 7:0] data_buf_r;
    reg         ss_flag;
    reg         start_flag;
    wire        ready_flag;

```

```
    spi_master_driver spi(
        .clk_i(HCLK),
        .rst_i(~HRESETn),

        .start_i(start_flag),
        .data_in_bi(data_buf_w),
        .busy_o(ready_flag),
        .data_out_bo(data_buf_r),

        .spi_miso_i(SPI_MISO),
        .spi_mosi_o(SPI_MOSI),
        .spi_sclk_o(SPI_SCLK),
        .spi_cs_i(ss_flag)
    );

    // AHB bus adapter
    reg  [ 11:0] HADDR_dly;
    reg          HWRITE_dly;
    reg          HSEL_dly;

    always @(posedge HCLK) begin
        HADDR_dly <= HADDR[11:0];
        HWRITE_dly <= HWRITE;
        HSEL_dly <= HSEL;
    end

    wire [ 11:0] reg_addr = HADDR_dly;

    always @(posedge HCLK or negedge HRESETn) begin
        if (~HRESETn) begin
            data_buf_w <= 8'b0;
            start_flag <= 1'b0;
            ss_flag    <= 1'b1;
        end
        else begin
            if (HSEL_dly) begin
                // Bus interface logic, write operation
                if (HWRITE_dly) begin
                    case (reg_addr)
                    DATA_REG_ADDR:  data_buf_w <= HWDATA[7:0];
                    START_REG_ADDR: start_flag <= HWDATA[0];
                    SS_REG_ADDR:    ss_flag    <= HWDATA[0];
                    default:; /* READY_REG_ADDR: do nothing */
                    endcase
                end
                // Reset start flag automatically after one tick
                if (!(HWRITE_dly && reg_addr == START_REG_ADDR)) begin
                    start_flag <= 1'b0;
                end
            end
        end
    end

    // Bus interface logic, data for read operation
    always @(*) begin
        SPI_SS = ss_flag;
```

```
        case (reg_addr)
            READY_REG_ADDR: HRDATA = ~ready_flag;
            DATA_REG_ADDR:  HRDATA = data_buf_r;
            default:        HRDATA = 32'b0;
        endcase
    end

endmodule
```

Листинг 2: ahb_feeder.v

```
`timescale 100ps/1ps

module ahb_feeder
(
    input               HCLK,
    output reg          HRESETn,
    output      [ 31: 0] HADDR,
    output      [  2: 0] HBURST,
    output               HMASTLOCK,
    output      [  3: 0] HPROT,
    output      [  2: 0] HSIZE,
    output      [  1: 0] HTRANS,
    output      [ 31: 0] HWDATA,
    output               HWRITE,
    input       [ 31: 0] HRDATA,
    input                HREADY,
    input                HRESP
);

    ahb_master ahb_master (
        .HCLK(HCLK),
        .HRESETn(HRESETn),
        .HADDR(HADDR),
        .HBURST(HBURST),
        .HMASTLOCK(HMASTLOCK),
        .HPROT(HPROT),
        .HSIZE(HSIZE),
        .HTRANS(HTRANS),
        .HWDATA(HWDATA),
        .HWRITE(HWRITE),
        .HRDATA(HRDATA),
        .HREADY(HREADY),
        .HRESP(HRESP)
    );

    reg  [31:0] data_buf;
    reg         spi_ready_flag;

    localparam UART_DATA_ADDR = 32'hbf400000; // register for data to transmit (only one byte is used)
    localparam UART_CTRL_ADDR = 32'hbf400004; // control regster
    localparam UART_DVDR_ADDR = 32'hbf400008; // clock divider register

    localparam SPI_START_ADDR = 32'hbf000000;
    localparam SPI_SS_ADDR    = 32'hbf000004;
    localparam SPI_READY_ADDR = 32'hbf000008;
```

```verilog
    localparam SPI_DATA_ADDR  = 32'hbf000012;

    initial begin
        // wait for end of reset
        repeat (35)  @(posedge HCLK);

        // Test PMODJSTK controller

        ahb_master.ahb_write(SPI_SS_ADDR, 1'b0);

        ahb_master.ahb_write(SPI_DATA_ADDR, 32'b10101010);
        ahb_master.ahb_write(SPI_START_ADDR, 1'b1);
        spi_ready_flag = 1'b0;
        while (spi_ready_flag == 1'b0) begin
            ahb_master.ahb_read(SPI_READY_ADDR, spi_ready_flag);
        end
        ahb_master.ahb_read(SPI_DATA_ADDR, data_buf);

        ahb_master.ahb_write(SPI_DATA_ADDR, 32'b00110011);
        ahb_master.ahb_write(SPI_START_ADDR, 1'b1);
        spi_ready_flag = 1'b0;
        while (spi_ready_flag == 1'b0) begin
            ahb_master.ahb_read(SPI_READY_ADDR, spi_ready_flag);
        end
        ahb_master.ahb_read(SPI_DATA_ADDR, data_buf);

        ahb_master.ahb_write(SPI_DATA_ADDR, 32'b11111111);
        ahb_master.ahb_write(SPI_START_ADDR, 1'b1);
        spi_ready_flag = 1'b0;
        while (spi_ready_flag == 1'b0) begin
            ahb_master.ahb_read(SPI_READY_ADDR, spi_ready_flag);
        end
        ahb_master.ahb_read(SPI_DATA_ADDR, data_buf);

        ahb_master.ahb_write(SPI_START_ADDR, 1'b1);
        spi_ready_flag = 1'b0;
        while (spi_ready_flag == 1'b0) begin
            ahb_master.ahb_read(SPI_READY_ADDR, spi_ready_flag);
        end
        ahb_master.ahb_read(SPI_DATA_ADDR, data_buf);

        ahb_master.ahb_write(SPI_START_ADDR, 1'b1);
        spi_ready_flag = 1'b0;
        while (spi_ready_flag == 1'b0) begin
            ahb_master.ahb_read(SPI_READY_ADDR, spi_ready_flag);
        end
        ahb_master.ahb_read(SPI_DATA_ADDR, data_buf);

        ahb_master.ahb_write(SPI_SS_ADDR, 1'b1);

    end

endmodule
```

Листинг 3: spi_master_driver.v

```
`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_cs_i,
    input             spi_miso_i,
    output reg        spi_mosi_o,
    output reg        spi_sclk_o
    );

    localparam CLK_NOPS = 1;

    reg   in_progress;
    reg   [2:0] counter;
    reg   [7:0] clk_div;
    reg   clk_div_pulse;
    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_IDLE         = 3; // wait one SCLK and swith to idle

    reg   [2:0] state;

    always @(posedge clk_i) begin
        if (rst_i) begin
            counter     <= 0;
            shiftreg    <= 0;
            bit_buffer  <= 0;
            in_progress <= 0;
            clk_div     <= 0;
            clk_div_pulse <= 0;
            state       <= STATE_IDLE;
        end
        else begin
            case (state)
                STATE_IDLE: begin
                    if (start_i) begin
                        in_progress = 1;
                        shiftreg <= data_in_bi;
                        state <= STATE_WAIT_SCLK_1;
                        clk_div <= 0;
```

```
                    end else begin
                        in_progress = 0;
                    end
                end
                STATE_WAIT_SCLK_1: begin
                    if (clk_div == CLK_NOPS) begin
                        bit_buffer <= spi_miso_i;
                        state <= STATE_WAIT_SCLK_0;
                        clk_div <= 0;
                        clk_div_pulse <= ~clk_div_pulse;
                    end else begin
                        clk_div <= clk_div + 1;
                    end
                end
                STATE_WAIT_SCLK_0: begin
                    if (clk_div == CLK_NOPS) begin
                        shiftreg <= { bit_buffer, shiftreg[7:1] };
                        if (counter == 7) begin
                            in_progress <= 0;
                            state <= STATE_WAIT_IDLE;
                            counter <= 0;
                        end else begin
                            state <= STATE_WAIT_SCLK_1;
                            counter <= counter + 1;
                        end
                        clk_div <= 0;
                        clk_div_pulse <= ~clk_div_pulse;
                    end else begin
                        clk_div <= clk_div + 1;
                    end
                end
                STATE_WAIT_IDLE: begin
                    if (clk_div == CLK_NOPS) begin
                        state <= STATE_IDLE;
                        clk_div <= 0;
                        clk_div_pulse <= 0;
                    end else begin
                        clk_div <= clk_div + 1;
                    end
                end
                default: begin
                    state <= STATE_IDLE;
                end
            endcase
        end
    end

    always @* begin
        busy_o      = (state != STATE_IDLE);
        data_out_bo = shiftreg;
        spi_sclk_o  = clk_div_pulse && !spi_cs_i;
        if (in_progress)
            spi_mosi_o  = shiftreg[0] && !spi_cs_i;
        else
            spi_mosi_o  = 0;
    end
```

```
113
114 endmodule
```