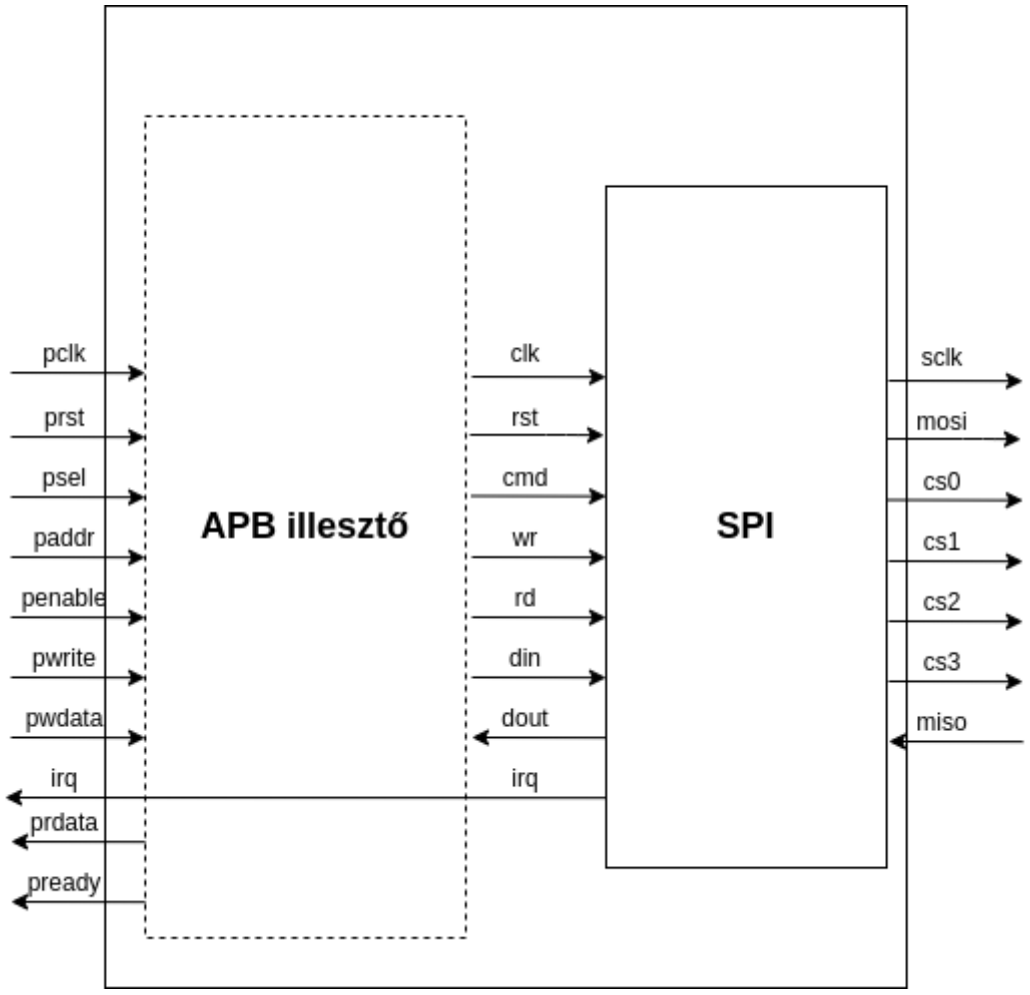
Házi Feladat
Rendszerarchitektúrák
2020

Váradi Balázs – U6RZ8H

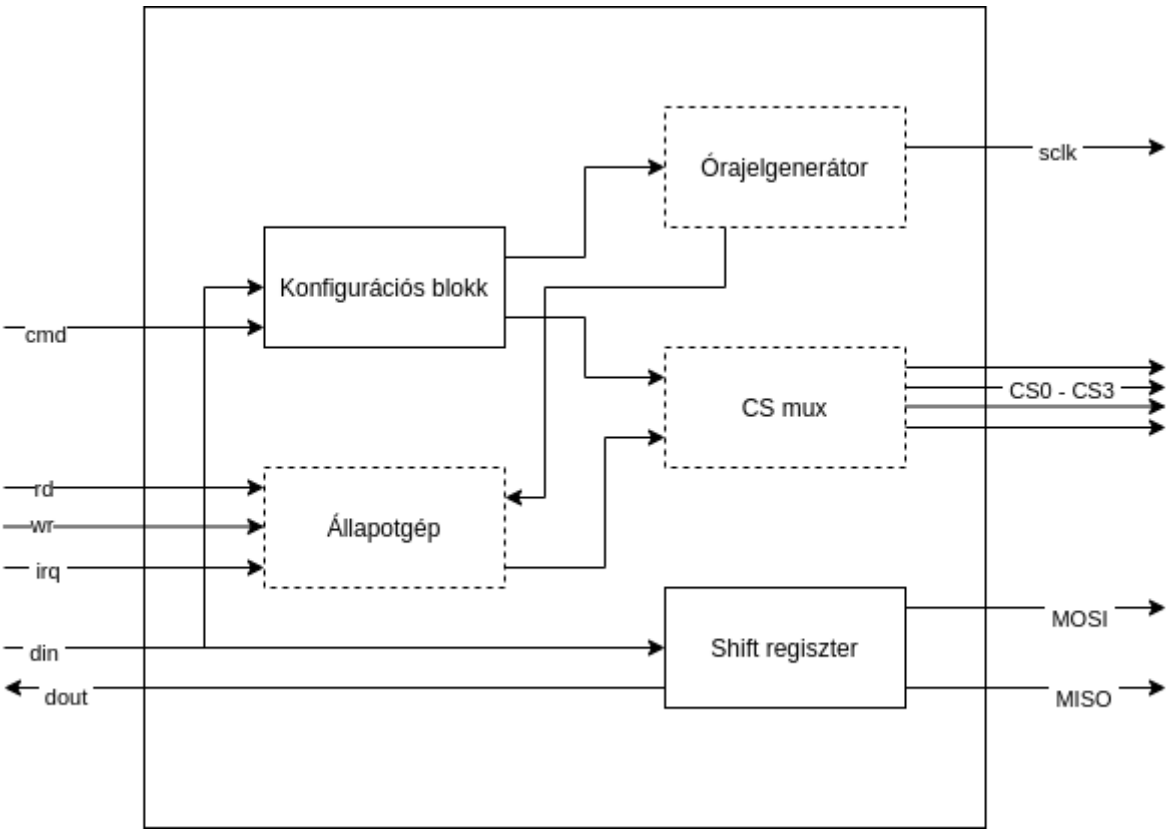
1. A periféria specifikációja:

A specifikációnak megfelelően a periféria 8 bites SPI master interfészt valósít meg, támogatja a 4 üzemmódot 1 és 8 MHz között, valamint 4 konfigurálható CSn jellel és engedélyezhető IRQ-val rendelkezik.

2. A rendszer terve

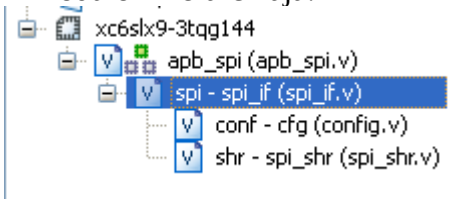


A top modul megvalósítja az APB illesztést, a megvalósított SPI interfészhez. Az APB illesztő egyszerűsége miatt a top modulban került megvalósításra.



A SPI modul megvalósítja az órajel és a CSn jelek előállítását, valamint kezeli az interfész állapotait, ezen felül magába foglalja az SPI shift regisztert és az konfigurációs blokkot.

A modulok hierarchiája:



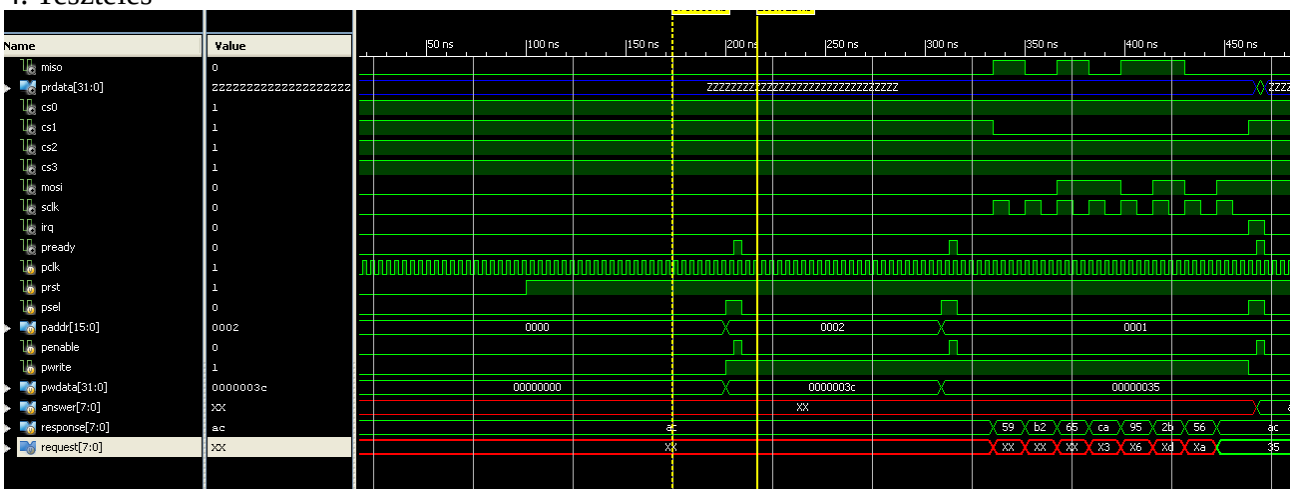
3. Memóriatérkép

Cím	Művelet	Tartalom
0x0001	Read	Fogadott SPI adat kiolvasása és IRQ törlése
0x0001	Write	Megadott adat SPI perifférián kiküldése
0x0002	Write	Konfiguráció beállítása

Konfiguráció

Bit	7	6	5	4	3	2	1	0
Opció	Órajel osztó			IRQ	Csn		CPOL	CPHA

4. Tesztelés



Ábrán megtekinthető:
0x3c konfiguráció beállítása,
0x35 perifériára írás.
Valamint az előző írás alatt beolvasott 0xac érték kiolvasása.

Forráskód:
A teljes forráskód elérhető a https://github.com/netpok/ambaspi címen.

config.v:

```
`timescale 1ns / 1ps

module cfg(
     input clk,
     input rst,

    input [7:0] din,
    input cmd,
    output [2:0] clk_div,
    output irq_en,
    output [1:0] cs_sel,
    output [1:0] mode
    );

reg [7:0] cfg_reg;

always @ (posedge clk)
begin
    if (rst)
        cfg_reg <= 0;
    else if (cmd)
       cfg_reg <= din;
end

assign clk_div = cfg_reg[7:5];
assign irq_en  = cfg_reg[4];
assign cs_sel  = cfg_reg[3:2];
assign mode    = cfg_reg[1:0];

endmodule
```

```
`timescale 1ns / 1ps

module spi_shr(
    input clk,
    input rst,
    input load,
    input shift,
    input in,
    input [7:0] din,
    input out,
    input [7:0] dout
    );
```

spi_shr.v

```
reg [8:0] shr;

always @ (posedge clk)
begin
    if (rst)
        shr <= 0;
    else if (load)
        shr <= din;
    else if (shift)
        shr <= {shr[6:0], in};
    else
        shr <= shr;
end

assign dout = shr;
assign out = shr[7];

endmodule
```

spi_if.v

```
`timescale 1ns / 1ps
module spi_if(
    input clk,
    input rst,

    input miso,
    output sclk,
    output mosi,
    output cs0,
    output cs1,
    output cs2,
    output cs3,

   input [7:0] din,
   input cmd,
   input rd,
   input wr,
   output [7:0] dout,
   output irq
   );

reg csn;
wire shift;

wire [2:0] clk_div;
wire irq_en;
wire [1:0] cs_sel;
wire [1:0] mode;

wire clock;
wire clock_rising;
reg [3:0] clock_cntr;
reg clock_last;

always @(posedge clk)
begin
    if (rst)
        clock_cntr <= 0;
    else
        clock_cntr <= clock_cntr + 1;
end

assign clock = clock_cntr[clk_div];

always @(posedge clk)
begin
    if (rst)
        clock_last <= 0;
    else
        clock_last <= clock;
end

assign clock_rising = ({clock, clock_last} == 2'b10);


reg [2:0] bit_cntr;

cfg conf (
    .clk(clk),
    .rst(rst),
    .din(din),
    .cmd(cmd),
```

```
    .clk_div(clk_div),
    .irq_en(irq_en),
    .cs_sel(cs_sel),
    .mode(mode)
);

spi_shr shr (
    .clk(clk),
    .rst(rst),
    .load(wr),
    .shift(shift),
    .din(din),
    .in(miso),
    .dout(dout),
    .out(mosi)
);

reg [3:0] state;
parameter S_IDLE     = 4'b0000;   // Idle
parameter S_START    = 4'b0001;   // Start transmission
parameter S_TRANSMIT = 4'b0010;   // 1st byte on output ...
parameter S_DONE     = 4'b0011;   // ... 8th byte on output

always @ (posedge clk)
begin
    if (rst | rd) begin
        state <= S_IDLE;
        bit_cntr <= 0;
        csn <= 0;
    end
    else if (wr) begin
        state <= S_START;
        bit_cntr <= bit_cntr;
        csn <= 0;
    end
    else if (clock_rising & state == S_START) begin
        csn <= 1;
        bit_cntr <= 7;
        state <= S_TRANSMIT;
    end
    else if (clock_rising & state == S_TRANSMIT & bit_cntr == 0) begin
        csn <= 0;
        bit_cntr <= bit_cntr;
        state <= S_DONE;
    end
    else if (clock_rising & state == S_TRANSMIT) begin
        csn <= csn;
        bit_cntr <= bit_cntr - 1;
        state <= state;
    end
    else begin
        csn <= csn;
        bit_cntr <= bit_cntr;
        state <= state;
    end
end

assign sclk = ((clock_last ^ ~mode[0]) & state == S_TRANSMIT) ^ mode[1];
assign shift = (clock_rising & state == S_TRANSMIT);
assign irq = (irq_en & state == S_DONE);
assign cs0 = ~((cs_sel == 0) & csn);
assign cs1 = ~((cs_sel == 1) & csn);
assign cs2 = ~((cs_sel == 2) & csn);
assign cs3 = ~((cs_sel == 3) & csn);
```

```
endmodule
```

apb_spi.v:

```
`timescale 1ns / 1ps
module apb_spi(
    input pclk,
    input prst,
    input psel,
    input [15:0] paddr,
    input penable,
    input pwrite,
    input [31:0] pwdata,
    output [31:0] prdata,
     output pready,
    output cs0,
    output cs1,
    output cs2,
    output cs3,
    input miso,
    output mosi,
    output sclk,
    output irq
    );

wire wr;
wire cmd;
wire rd;
wire [7:0] din;
wire [7:0] dout;

spi_if spi (
    .clk(pclk),
    .rst(~prst),
    .cs0(cs0),
    .cs1(cs1),
    .cs2(cs2),
    .cs3(cs3),
    .miso(miso),
    .mosi(mosi),
    .sclk(sclk),
    .irq(irq),
    .din(din),
    .dout(dout),
    .cmd(cmd),
    .rd(rd),
    .wr(wr)
);

parameter DATA_ADDR = 1;
parameter CMD_ADDR = 2;

assign din = pwdata[7:0];
assign rd = (~pwrite & psel & paddr == DATA_ADDR & penable);
assign prdata = rd ? {24'b0, dout[7:0]} : 32'bZ;
assign wr = (pwrite & psel & paddr == DATA_ADDR & penable);
assign cmd = (pwrite & psel & paddr == DATA_ADDR & penable);
assign pready = psel & penable;

endmodule
```

tb_top.v:

```
`timescale 1ns / 1ps
module tb_top;

    // Inputs
    reg pclk;
    reg prst;
    reg psel;
    reg [15:0] paddr;
    reg penable;
    reg pwrite;
    reg [31:0] pwdata;
    wire miso;

    // Outputs
    wire [31:0] prdata;
    wire cs0;
    wire cs1;
    wire cs2;
    wire cs3;
    wire mosi;
    wire sclk;
    wire irq;
    wire pready;

    // Instantiate the Unit Under Test (UUT)
    apb_spi uut (
        .pclk(pclk),
        .prst(prst),
        .psel(psel),
        .paddr(paddr),
        .penable(penable),
        .pwrite(pwrite),
        .pwdata(pwdata),
        .prdata(prdata),
        .pready(pready),
        .cs0(cs0),
        .cs1(cs1),
        .cs2(cs2),
        .cs3(cs3),
        .miso(miso),
        .mosi(mosi),
        .sclk(sclk),
        .irq(irq)
    );

    task write(input [7:0] addr, input [7:0] data);
    begin
        wait(pclk);
        paddr <= addr;
        pwrite <= 1;
        psel <= 1;
        pwdata <= data;

        #4
        penable <= 1;
        wait(pready);

        #4
        penable <= 0;
        psel <= 0;
    end
    endtask
```

```
    reg [7:0] answer;

    task read;
    begin
        wait(pclk);
        paddr <= 1;
        pwrite <= 0;
        psel <= 1;

        #4
        penable <= 1;
        wait(pready);
        answer <= prdata[7:0];

        #4
        penable <= 0;
        psel <= 0;
        end
    endtask

    reg [7:0] response;
    reg [7:0] request;

    always @(posedge sclk) begin
        if (~cs1) begin
            response <= {response[6:0], response[7]};
            request <= {request[6:0], mosi};
        end
    end

    assign miso = response[0];

    always #2
        pclk = ~pclk;


    initial begin
        // Initialize Inputs
        pclk = 0;
        prst = 0;
        psel = 0;
        paddr = 0;
        penable = 0;
        pwrite = 0;
        pwdata = 0;
        response = 8'b10101100;

        // Wait 100 ns for global reset to finish
        #100;
        prst = 1;

        // Add stimulus here
        #100;
        write(2, 8'b00111100); // Prescaler 2 (4MHz), IRQ enable, CS 4, mode 0
        #100;
        write(1, 8'b00110101);
        wait(irq)
        read();
    end

endmodule
```