

Házi Feladat

Rendszerarchitektúrák

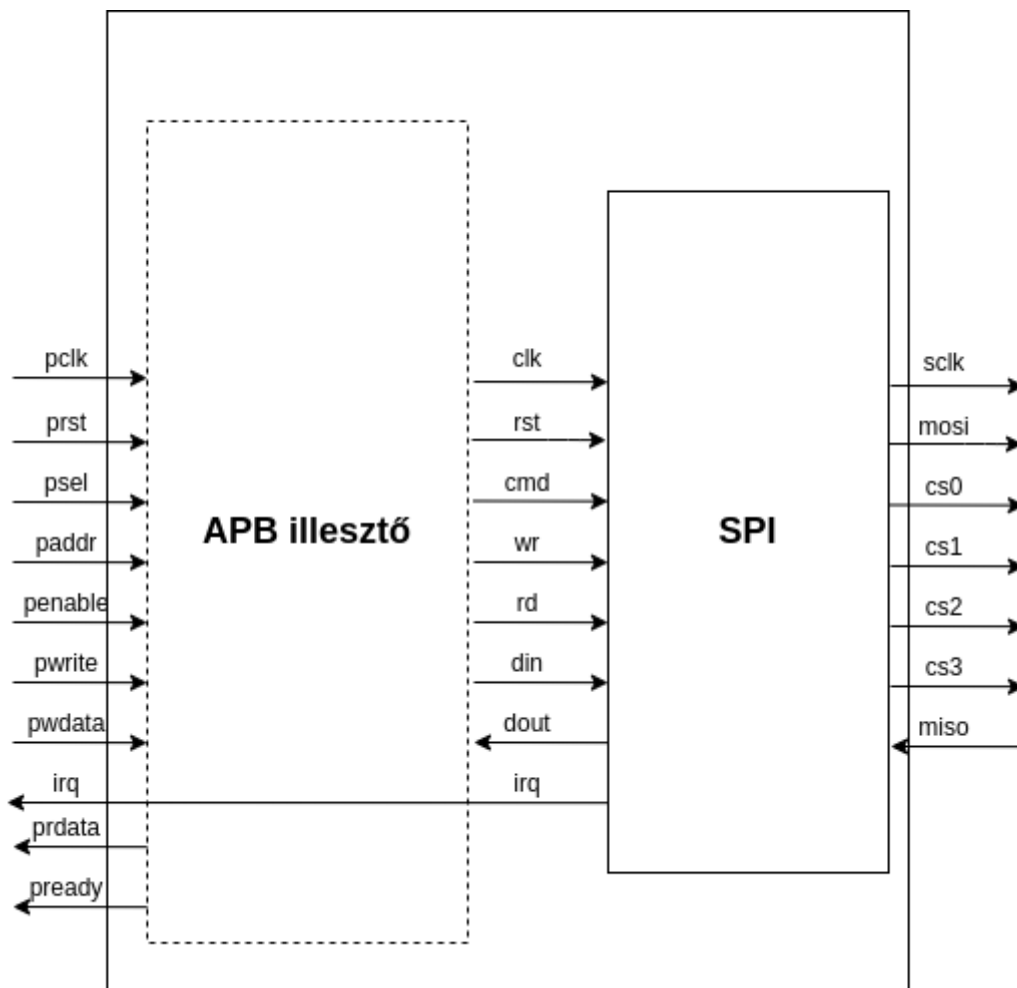
2020

Váradi Balázs – U6RZ8H

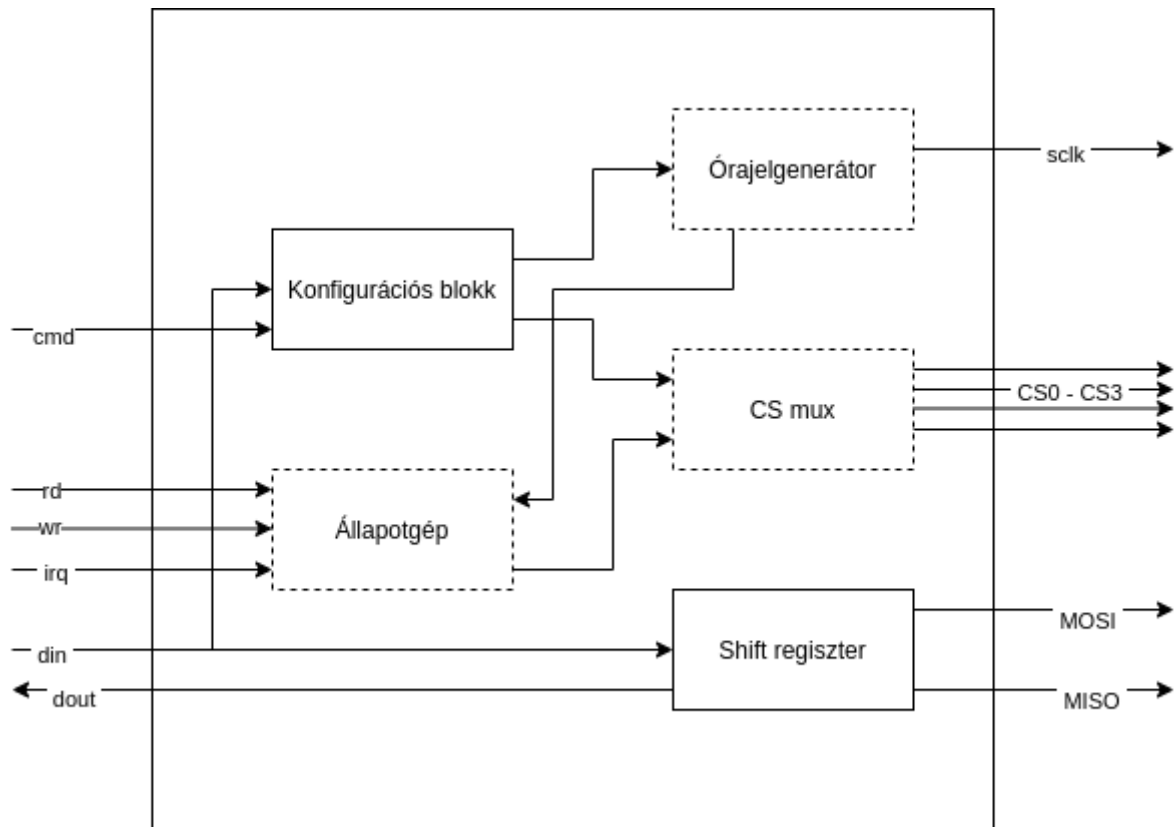
1. A periféria specifikációja:

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

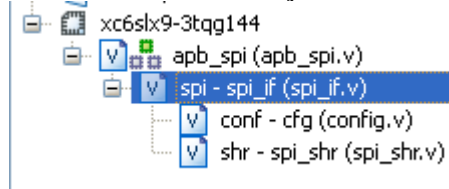


A top modul megvalósítja az APB illesztést, a megvalósított SPI interfészhez. Az APB illesztő egyszerűsége miatt a top modulban került megvalósításra.



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 periférián kiküldése
0x0002	Write	Konfiguráció beállítása

Konfiguráció

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

Órajel:

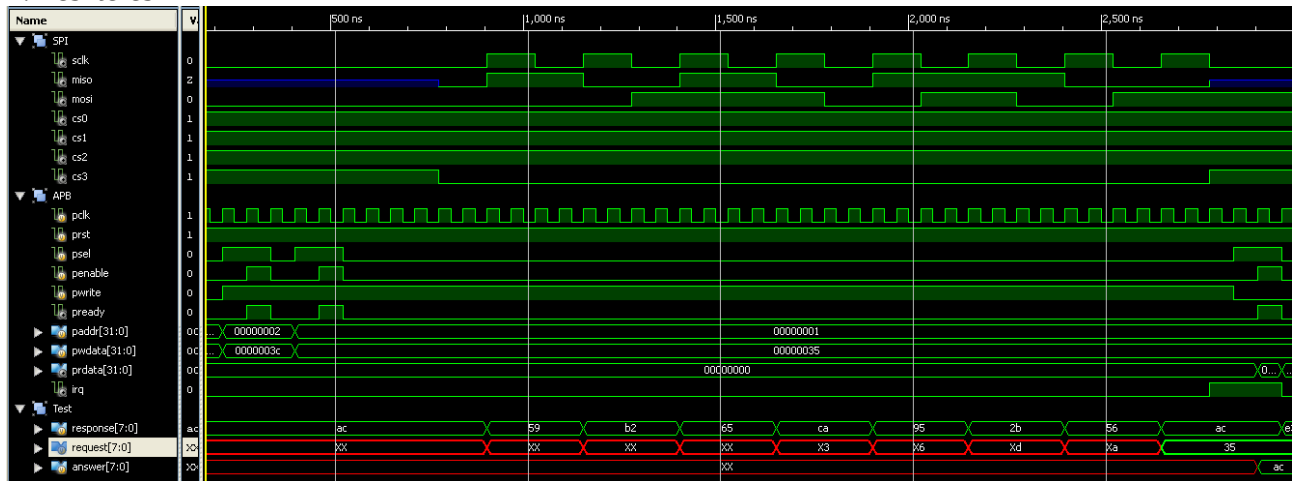
11 – 1MHz

10 – 2MHz

01 – 4MHZ

00 – 8MHz

4. Tesztelés

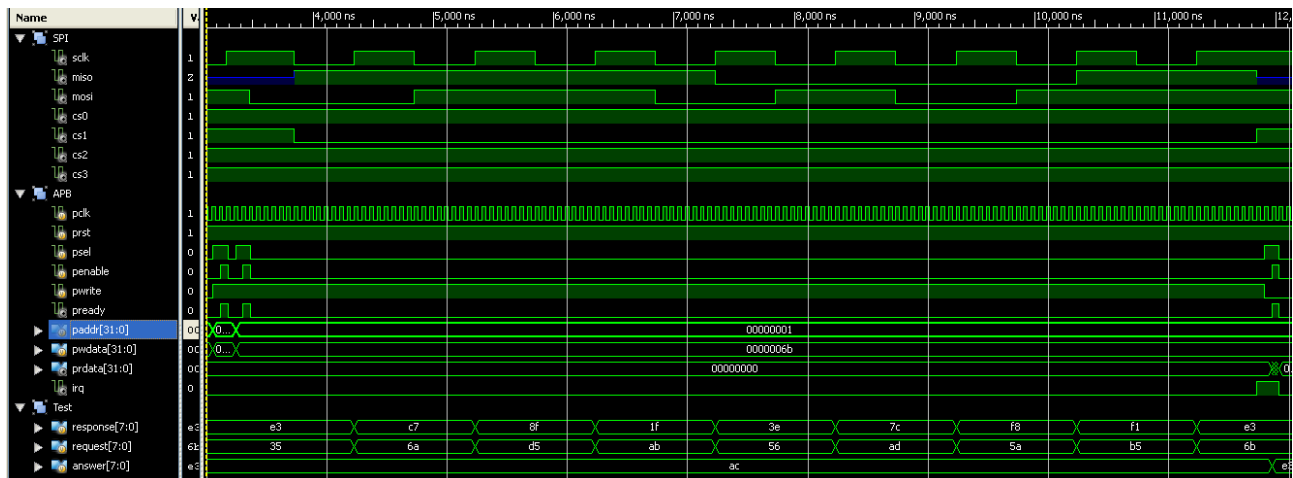


Ábrán megtekinthető:

0x3C konfiguráció beállítása, 4MHz, IRQ on, CS3, mód 0

0x35 perifériára írás.

Valamint az előző írás alatt beolvasott 0xAC érték kiolvasása.



Ugyanezen szimulációban később egy második művelet:

0x77 konfiguráció beállítása, 1MHz, IRQ on, CS1, mód 3

0x6B perifériára írás.

Valamint az előző írás alatt beolvasott 0xE3 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 [1:0] 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[6:5];
assign irq_en  = cfg_reg[4];
assign cs_sel  = cfg_reg[3:2];
assign mode    = cfg_reg[1:0];

endmodule
```

sbi_shr.v

```
`timescale 1ns / 1ps

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

reg [7:0] 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 [1:0] 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 | cmd)
        clock_cntr <= 0;
    else
        clock_cntr <= clock_cntr + 1;
end

assign clock = clock_cntr[clk_div];

always @(posedge clk)
begin
    if (rst | cmd)
        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 [1:0] state;
    parameter S_IDLE      = 2'b00;    // Idle
    parameter S_START     = 2'b01;    // Start transmission
    parameter S_TRANSMIT  = 2'b10;    // 1st byte on output ...
    parameter S_DONE      = 2'b11;    // ... 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 [31:0] paddr,
    input penable,
    input pwrite,
    input [31:0] pwidth,
    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 = pwidth[7:0];
assign rd = (~pwrite & psel & paddr == DATA_ADDR & penable);
assign prdata = rd ? {24'b0, dout[7:0]} : 32'b0;
assign wr = (pwrite & psel & paddr == DATA_ADDR & penable);
assign cmd = (pwrite & psel & paddr == CMD_ADDR & penable);
assign pready = psel & penable;

endmodule
```

tb_top.v:

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

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

    localparam CLK = 62.5; // 16 MHz

    // 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),
        .pdata(pdata),
        .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;
        pdata <= data;

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

        #CLK
        penable <= 0;
        psel <= 0;
    end
endmodule
```

```

end
endtask

reg [7:0] answer;

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

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

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

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

always @(posedge sclk) begin
    if (~cs1 | ~cs3) begin
        response <= {response[6:0], response[7]};
        request <= {request[6:0], mosi};
    end
end

assign miso = (cs1 & cs3) ? 1'bZ : response[0];

always #(CLK/2)
    pclk = ~pclk;

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

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

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

```

```
    response = 8'b11100011;
    // Add stimulus here
    #200;
    write(2, 8'b01110111); // clock 1MHz, IRQ enable, CS 1, mode 4
    #CLK;
    write(1, 8'b01101011);
    wait(irq)
    #CLK
    read();
end
endmodule
```