



**Budapesti Műszaki és Gazdaságtudományi Egyetem**  
Villamosmérnöki és Informatikai Kar  
Méréstechnika és Információs Rendszerek Tanszék

# Rendszerarchitektúrák házi feladat

DOKUMENTÁCIÓ

*Készítette*  
Horváth Dániel  
Kalocsai Kristóf  
Uzseka Dániel

*Konzulens*  
Fehér Béla

2017. május 17.

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>2</b>
1.1. AMBA-APB . . . . .	2
1.2. I2C . . . . .	3
<b>2. Áttekintés</b>	<b>5</b>
2.1. A perifériaillesztő modul felépítése . . . . .	5
2.2. mod_apb . . . . .	5
2.3. mod_i2c . . . . .	6
<b>3. Tesztelés</b>	<b>9</b>
<b>Ábrák jegyzéke</b>	<b>11</b>
<b>Táblázatok jegyzéke</b>	<b>12</b>
<b>Irodalomjegyzék</b>	<b>13</b>
<b>Függelék</b>	<b>14</b>
F.1. mod_top.v . . . . .	14
F.2. macros.vh . . . . .	15
F.3. mod_APB.v . . . . .	16
F.4. mod_I2C.v . . . . .	17
F.5. tb_top.v . . . . .	19

# 1. fejezet

## Bevezetés

Féléves munkánk során egy perifériaillesztő modult valósítottunk meg System Verilog nyelven. Az illesztő az ARM<sup>TM</sup>Ltd. AMBA<sup>TM</sup>-APB rendszerbusza, és az I2C busz között teremt kapcsolatot. Ezen két buszt ismertetjük a továbbiakban.

### 1.1. AMBA-APB

Az Advanced Microcontroller Bus Architecture egy nyílt szabvány, amely chipen belüli kommunikációs összeköttetéseket definiál. Ennek a szabványnak része az Advanced Peripheral Bus (APB), amely alacsony sávszélességű, kis komplexitású, és minimális fogyasztású.

<b>PCLK</b>	Órajel, felfutó éle időzíti az összes átviteli ciklust.
<b>PRESETn</b>	Reset, aktív-alacsony.
<b>PADDR</b>	Címbusz, maximum 32 bit széles.
<b>PSELx</b>	Slave-select, minden egységhez tartozik egy. Kiválasztja az adott slave egységet.
<b>PENABLE</b>	Engedélyező jel, az átviteli ciklus második szakaszát jelzi.
<b>PWRITE</b>	Írányjelző, magas értéke írást, alacsony értéke olvasást jelent.
<b>PWDATA</b>	Adatbusz, maximum 32 bit széles, mindig a perifériabusz bridge hajtja.
<b>PRDATA</b>	Adatbusz, maximum 32 bit széles, a slave egység hajtja az olvasási ciklusban.

**1.1. táblázat.** Az APB busz jelei.

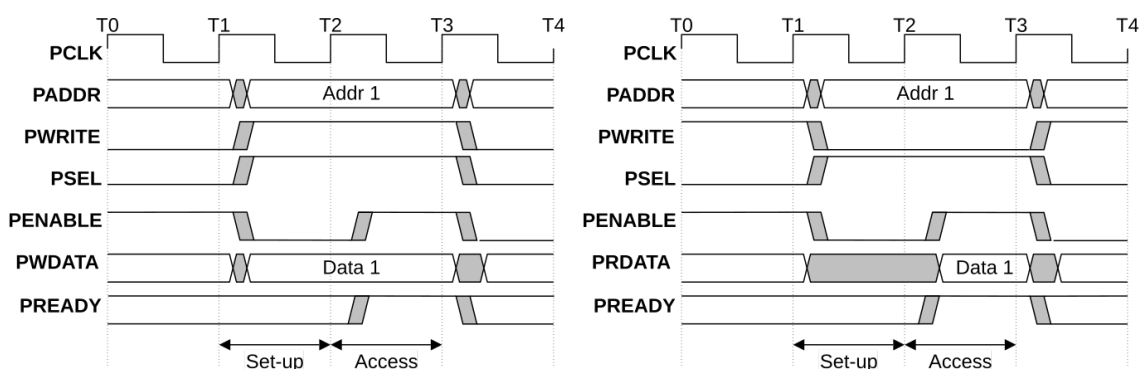
A busz jeleinek áttekintése után nézzünk egy írási ciklust a buszon. Minden átviteli ciklus két szakaszból áll, egy *setup* és egy *access* fázisból, ezen fázisok PCLK felfutó élére kezdődnek, ahogy az az 1.1 ábrán <sup>1</sup> megfigyelhető. A setup fázisban a híd eszköz (APB master) a PADDR buszra kiadja a címzett periféria címét, a PWRITE jelet az írásnak megfelelően magas értékűre

<sup>1</sup>Források:

[http://infocenter.arm.com/help/topic/com.arm.doc.ddi0367b/graphics/apb\\_write\\_transfer\\_no\\_wait\\_states.svg](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0367b/graphics/apb_write_transfer_no_wait_states.svg),  
[http://infocenter.arm.com/help/topic/com.arm.doc.ddi0367b/graphics/apb\\_read\\_transfer\\_no\\_wait\\_states.svg](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0367b/graphics/apb_read_transfer_no_wait_states.svg)

állítja. A slave-hez tartozó PSEL vonalat is magas értékre állítja be, továbbá a PWDATA buszra kapuzza az írni kívánt adatot. Ebben a fázisban PENABLE alacsony értékű, így a slave eszköznek lehetősége van felkészülni az átvitelre. PCLK következő felfutó élére a PENABLE vonalat logikai 1 értékre állítja, és ezzel a buszciklus végéig bezárólag a slave eszköz mintát vesz a PWDATA buszról, amivel lezárul az átvitel, PSEL, PWRITE és PENABLE visszaállításra kerül a master által.

Az olvasás ciklus a vezérlőjeleket tekintve csak PWRITE értékében tér el. Ha az APB master olvasási ciklust kezdeményez, a PWRITE jelet logikai alacsony értékre állítja a setup fázisban. Az access fázisban szintén kiadja a PENABLE jelet, mire a slave eszköz a megcímzett regiszterét a PRDATA buszra kapuzza. Az előzőekhez hasonlóan, a master vezérlőjeleket visszaveszi a ciklus befejeztével.



1.1. ábra. Egy írási és olvasási ciklus időzítési viszonyai az APB buszon.

## 1.2. I2C

Az Inter-Integrated Circuit (vagy I2C, I<sup>2</sup>C, esetleg IIC) egy multi-master, multi-slave, single-ended, félduplex soros kommunikációs busz. Két, nyitott kollektoros (open-drain) vezeték használ a kétirányú kommunikációhoz, ezek az SDA adatvonal és SCL órajelvezeték. Fizikai kialakításból adódóan a vezetékeket ellenállásokon keresztül magas logikai (3V3, 5V, 1V8 stb.) feszültségre kell felhúzni. A buszra csatlakozó eszközök a vezetékeket "kényszeríteni" tehát csak lefelé tudják. Ez teszi lehetővé a multi-master struktúrához szükséges kiválasztást és arbitrációt, továbbá mivel különálló kiválasztó jelek (slave-select) nem állnak rendelkezésre, így az üzeneteket címezni kell. Egy üzenet felépítése látható az 1.2 ábrán.<sup>2</sup>

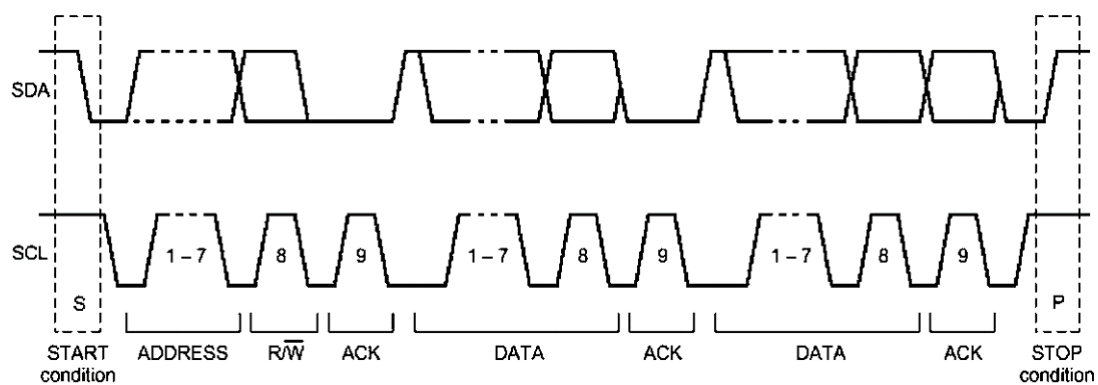
Az SCL órajelet mindig a master szolgáltatja a buszra csatlakozó eszközöknek.

Minden üzenet a START feltétellel (az SCL magas értéke mellett SDA lefutó éle) kezdődik és a STOP feltétellel (az SCL magas értéke mellett SDA felfutó éle) végződik. Ezek speciális feltételek, az üzenet belsejében nem fordulhatnak elő, mivel SDA csak SCL alacsony értéke mellett változhat.

A startbitet követi egy 7 bites címmező, amely a címzettet azonosítja, és egy R/W bit, amely írás esetén alacsony, olvasás esetén magas értékű. Ezután a küldő "elengedi" az adatbuszt, és a vevő, ha sikeres volt az átvitel, az adatbuszon a következő bit idejéig egy alacsony logikai értékű ACK nyugtázó jelet ad az SDA vonalra. Ha ez megtörtént, a master további 8 órajelciklus alatt egy byte-nyi adatot kapuz az adatbuszra. Újabb nyugtázás esetén a byte átvitele befejeződött, és sikeres. A masternek lehetősége van további byteokat küldeni, szintén

<sup>2</sup>Forrás:

<https://i2.wp.com/maxembedded.files.wordpress.com/2014/02/data-transfer-timing-diagram.png>



**1.2. ábra.** Az I2C busz időzítési diagrammja.

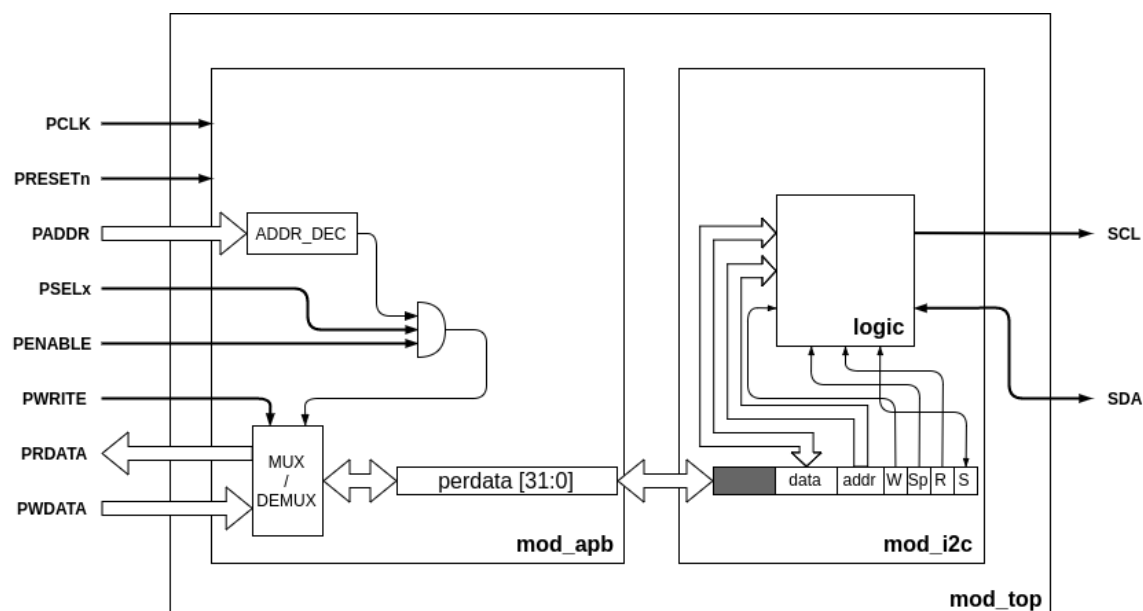
minden 8 bit adat után egy bit nyugtázással. Ha lezajlott a kívánt mennyiségű adat átvitele, a master kiadja a stopfeltételt, amely az üzenet végét jelenti.

## 2. fejezet

# Áttekintés

### 2.1. A perifériaillesztő modul felépítése

Mint az a 2.1 ábrán is látszik, modulunk (*mod\_top*) 2 almodult tartalmaz, a feladatkiírásnak megfelelően: egy *mod\_apb* modulból, ami közvetlenül az APB buszra csatlakozik, és egy *mod\_i2c* modulból, ami a soros kommunikáció vonalaival van összeköttetésben.



2.1. ábra. A modul magas szintű áttekintő ábrája.

Az APB modul a rendszerbusz vezérlőjeit dekódolja, és továbbítja a szükséges adatokat az I2C modulnak. Az I2C modul a rendelkezésére bocsátott adatokból lefolytatja soros kommunikációt.

A továbbiakban tekintsük át részletesebben az almodulok felépítését. A modulokhoz, és a későbbiekben a tesztekhez kapcsolódó forráskódok a dokumentum végén, a függelékben találhatóak.

### 2.2. mod\_apb

#### Ki és bemenetek

Mivel ez a modul közvetlenül az APB buszra csatlakozik, bemenetei megegyeznek a busz jeleivel, eltekintve a PREADY és PSLVERR jelektől, melyeket nem használunk. Továbbá tar-

talmaz két darab 32 bites ki illetve bemeneti portot, amely tartalmazza az I2C modul számára küldött, illetve attól fogadott összes releváns adatot. A pontos tartalmat lásd az I2C modul tárgyalásánál.

## Reset

A modul szinkron resetet valósít meg, az órajel felfutó élére mintát vesz a PRESETn jelből, amelynek alacsony értéke mellett nullára állítja a belső állapotregiszterét, illetve az APB és I2C felé menő kimeneteit.

## Állapotok

Ez az almodul 4 belső állapotot különböztet meg az APB vezérlőjelek alapján, ahol *X* az érdektelent (Don't Care), *1* a logikai magasat (illetve helyes címet), *0* pedig ennek ellenkezőjét jelöli. Az alábbi táblázat szemlélteti az állapotokat:

Állapot	PADDR	PSELx	PENABLE	PWRITE
IDLE	X	0	0	X
SETUP	X	1	0	X
READ	1	1	1	0
WRITE	1	1	1	1

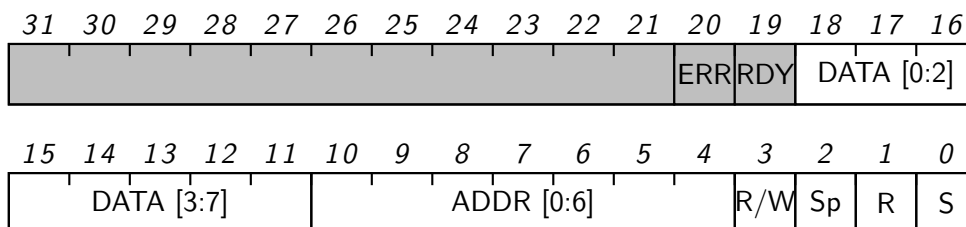
## Logika

A vezérlőjelek dekódolása alapján, READ állapotban az I2C modul felől kapott 32 bit széles értéket (*in\_perdata*) a PRDATA buszra kapuzza, WRITE állapotban a PWDATA busz tartalmát hozzárendeli az I2C felé tartó, *out\_perdata* kimenetéhez. IDLE állapotban nagyimpedanciát kapcsol a PRDATA buszra, hogy a rendszer többi egységét ne zavarja. Ez a logika feltételes értékadással került implementálásra.

## 2.3. mod\_i2c

### Ki és bemenetek

Ez a modul közvetlenül az I2C buszra csatlakozik, így rendelkezik egy kétirányú, háromállapotú SDA porttal, mely a nyitott-kollektoros működést valósítja meg, illetve egy SCL órajel kimenettel. Továbbá csatlakozik a mod\_apb modulhoz egy 32 bit széles regiszteren keresztül. Ez a regiszter tartalmaz minden információt a modul számára az I2C kommunikáció kezdeményezéséhez. A regiszter kiosztását és tartalmát lásd a 2.2 ábrán.



2.2. ábra. A kommunikációs regiszter kiosztása.

## Reset

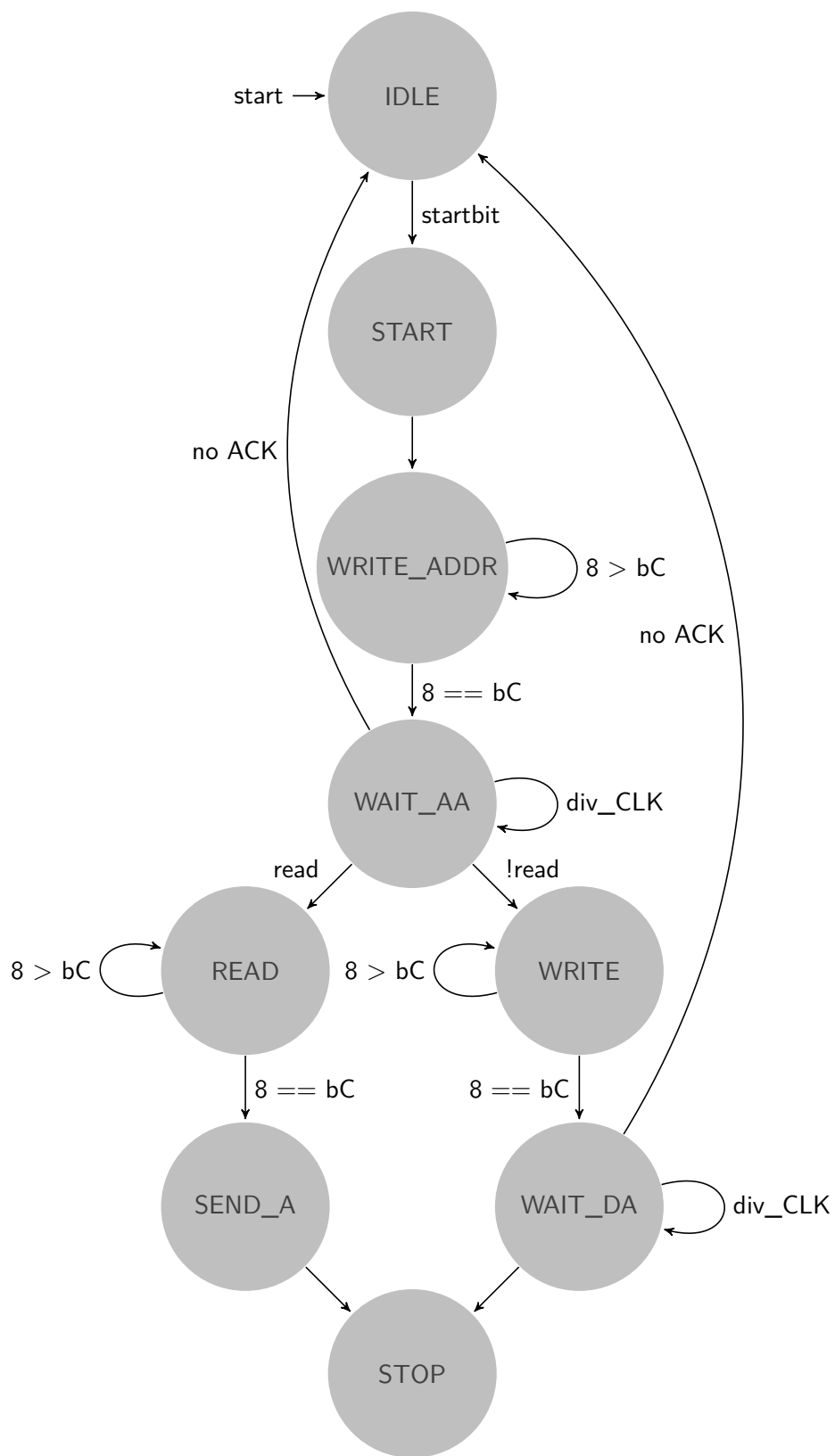
Ez az almodul is szinkron resetet valósít meg, az órajel felfutó élére vizsgálja a PRESETn vonalat és a regiszterének második bitjét ( regData[1] ). Ha ezek közül PRESETn-t 0-nak, vagy a bitet pedig 1-nek találja, elengedi az SDA és SCL vonalakat, és belső állapotváltozóit és számlálóit alaphelyzetbe állítja.

## Állapotok

### Logika

A modul állapotátmenetei, és mechanizmusa látható a 2.3 ábrán. A 2.2 ábrán látható start bit (**S**) 1 értékbe állításakor elindul a soros kommunikáció. A sebességet tároló bit (**Sp**) 1-re állításával az I2C kommunikáció órajele 400 kHz-es, míg 0-ra állításával 100 kHz-es frekvenciával rendelkezik. Az írás/olvasás bit (**R/W**) 1 értéke esetén a perifériától való olvasás valósul meg, míg fordított esetben ennek az ellenkezője. Amint a kommunikáció elindul, a modul a készenlét jelzőbitjét (**RDY**) 0-ra állítja, ezzel jelezve, (ha a felhasználó lekérdezi, kiolvassa az APB buszon keresztül) hogy átvitel van folyamatban. Hasonlóképpen, ha hiba történik az átvitelben (nem érkezik ACK jel), azt az **ERR** bittel jelzi a modul.



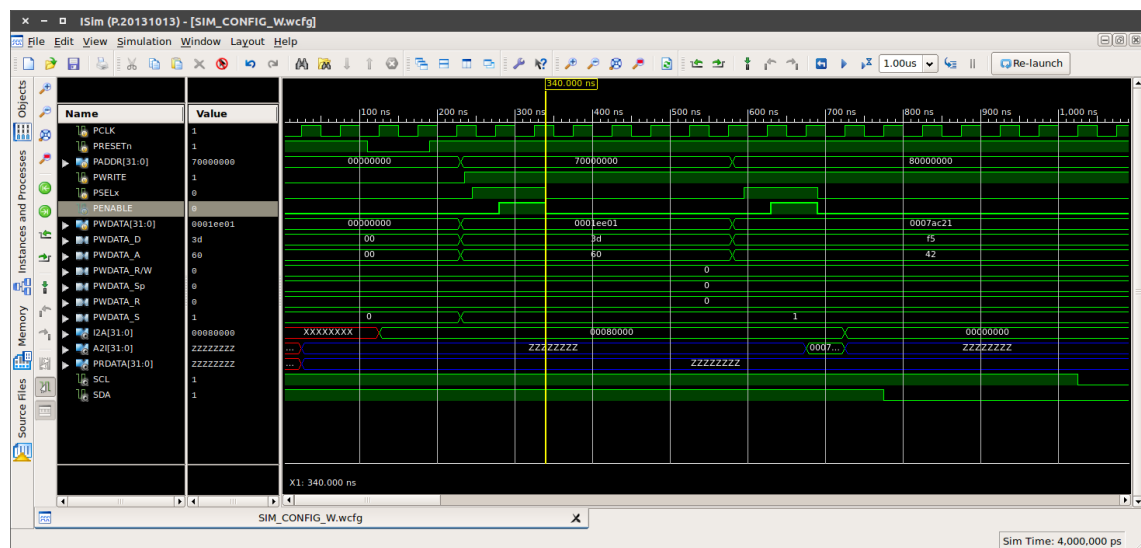


**2.3. ábra.** Az I2C modul állapotátmenetei

## 3. fejezet

# Tesztelés

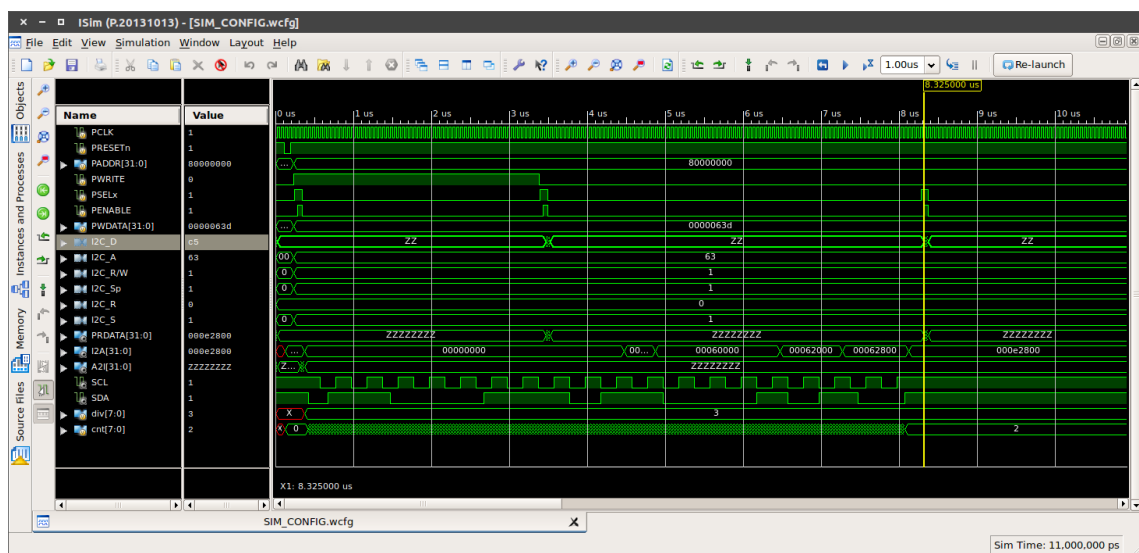
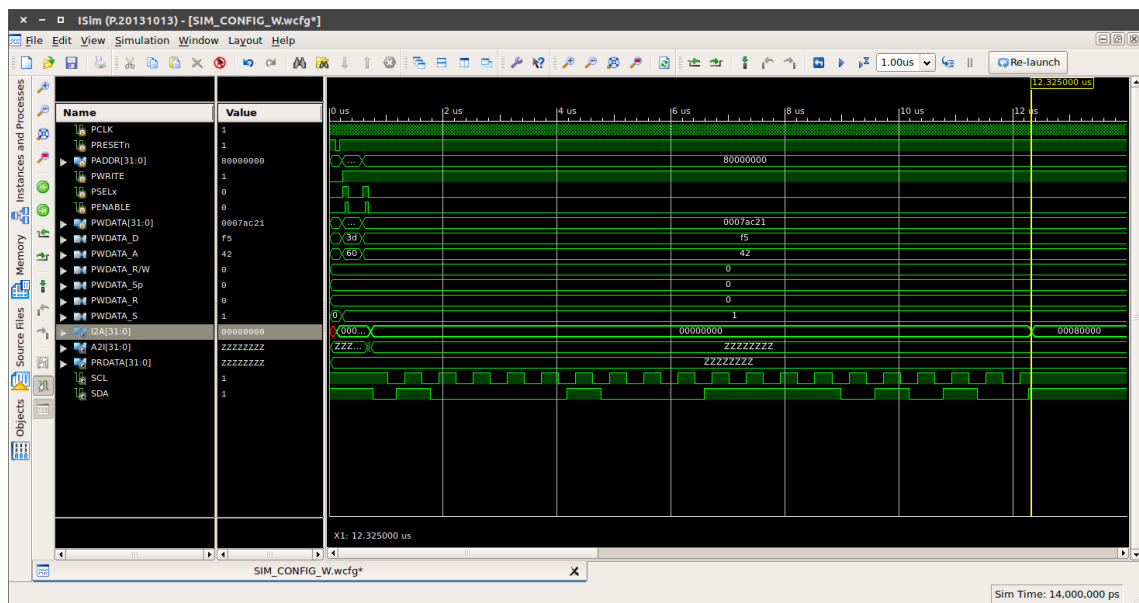
A 3.1 ábrán látható egy szimulációs hullámforma, amely a helyes APB vezérlőjel dekódolást hivatott demonstrálni. A kurzorral megjelölt hely előtti PCLK felfutóélre már minden szükséges vezérlő és engedélyezőjel ki volt adva az APB master részéről, az APB modul mégsem küld semmiféle adatot az I2C modul felé, mivel a PADDR buszon kiadott cím nem egyezik a perifériánk 0x80000000 címével. Viszont az ábrán 675ns - nál megfigyelhető egy helyes címre történő írás, így az A2I vezetéken megjelenik PWDATA tartalma, amit az I2C modul fel is dolgoz, és kezdi is a kommunikációt.



3.1. ábra. APB írás ciklus

A szimulációt tovább futtatva megfigyelhetünk egy rendben lezajló I2C írásműveletet a 3.2 ábrán. A szimulációhoz a slave felől érkező ACK jeleket kézzel, a szimulációs fájlokban szereplő módon adtuk ki, hogy igazolhassuk a modul megfelelő működését. A fájl természetesen megtalálható a függelékben, A hullámforma egy byte-nyi 0xF5 adat 0x42 címre történő írását ábrázolja.

A következő, 3.3 ábrán egy I2C olvasási ciklust szimuláltunk. Az ehhez tartozó szimulációs fájl szintén megtalálható a függelékben : Megfigyelhető egy lekérdezés az átvitel közben, de az olvasott adat ekkor **RDY** bit hiányában érvénytelen. Még egy APB olvasás látható a hullámformán, ezúttal már az átvitel lezajlása után, mikor is megfigyelhető az I2A, illetve a PRDATA buszokon az SDA vonalra kézzel felvezetett 0xC5 adat.



# Ábrák jegyzéke

1.1. Egy írási és olvasási ciklus időzítési viszonyai az APB buszon. . . . .	3
1.2. Az I2C busz időzítési diagrammja. . . . .	4
2.1. A modul magas szintű áttekintő ábrája. . . . .	5
2.2. A kommunikációs regiszter kiosztása. . . . .	6
2.3. Az I2C modul állapotátmenetei . . . . .	8
3.1. APB írás ciklus . . . . .	9
3.2. I2C írás ciklus . . . . .	10
3.3. I2C olvasás ciklus . . . . .	10

# Táblázatok jegyzéke

1.1. Az APB busz jelei. . . . .	2
---------------------------------	---

# Irodalomjegyzék

# Függelék

## F.1. mod\_top.v

```
'timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    07:42:55 04/07/2017
// Design Name:
// Module Name:    mod_top
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 – File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "macros.vh"

module mod_top
(
    input                PCLK,          // The rising edge of PCLK is used to
        time all transfers on the APB.
    input                PRESETn,       // The APB bus reset signal is active LOW
        and this signal will normally be connected directly to the system bus reset
        signal.
    input  ['addrWidth-1:0]  PADDR,      // This is the APB address bus,
        which may be up to 32-bits wide and is driven by the peripheral bus
        bridge unit.
    input                PSELx,         // A signal from the secondary decoder,
        within the peripheral bus bridge unit, to each peripheral bus slave x.
        This signal indicates that the slave device is selected and a data
        transfer is required. There is a PSELx signal for each bus slave.
    input                PENABLE,       // This strobe signal is used to time all
        accesses on the peripheral bus. The enable signal is used to indicate
        the second cycle of an APB transfer. The rising edge of PENABLE occurs
        in the middle of the APB transfer.
    input                PWRITE,        // When HIGH this signal indicates an APB
        write access and when LOW a read access.
    output  ['dataWidth-1:0]  PRDATA,    // The read data bus is driven by
        the selected slave during read cycles (when PWRITE is LOW). The read
        data bus can be up to 32-bits wide.
    input  ['dataWidth-1:0]  PWDATA     // The write data bus is driven
        by the peripheral bus bridge unit during write cycles (when PWRITE is
        HIGH). The write data bus can be up to 32-bits wide.
);

// apb_mod
```

```

// Outputs
wire ['dataWidth-1:0] prdata;
wire startbit;
wire resetbit;
wire it_enable;
wire ['dataWidth-1:0] per_addr; //azert datawith nem pedig addrwith mert ez
    az i2c periferiaira vonatkozik
wire ['dataWidth-1:0] per_data;

apb_mod amp_instance (
    .clk(PCLK),
    .reset(PRESETn),
    .addr(PADDR),
    .pwwdata(PWDATA),
    .prdata(PRDATA),
    .pwrite(PWRITE),
    .psel(PSELx),
    .penable(PENABLE),
    .startbit(startbit),
    .resetbit(resetbit),
    .it_enable(it_enable),
    .per_addr(per_addr),
    .per_data(per_data)
);

// Instantiate the module
mod_i2C instance_name (
    .SDA(SDA),
    .SCL(SCL),
    .command(command),
    .address(address),
    .data(data),
    .clk(clk),
    .rst(rst),
    .ready(ready)
);

/*reg [31:0] cntr;

always @ (posedge PCLK) begin
    if (!PRESETn)
        cntr <= 0;
    else if (PENABLE)
        if (PSELx)
            cntr <= cntr + 1;
        else
            cntr <= cntr - 1;
end*/

assign PRDATA = prdata;

endmodule

```

## F.2. macros.vh

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    08:52:53 04/29/2017
// Design Name:
// Module Name:    macros
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created

```



```

// Additional Comments:
//
/////////////////////////////////////////////////////////////////
`ifndef macros_vh
`define macros_vh

`define addrWidth 8
`define dataWidth 32

`endif

```

### F.3. mod\_APB.v

```

`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:45:43 04/28/2017
// Design Name:
// Module Name:    apb_mod
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
/////////////////////////////////////////////////////////////////
`include "macros.vh"

module apb_mod
(
    input clk,
    input reset,
    input ['addrWidth-1:0] addr,
    input ['dataWidth-1:0] pwrdata,
    output reg ['dataWidth:0] prdata,
    input pwrite,
    input psel,
    input penable,
    output reg startbit,
    output reg resetbit,
    output reg it_enable,
    output reg ['dataWidth-1:0] per_addr,
    output reg ['dataWidth-1:0] per_data
);

/*reg [7:0] clk_counter;
initial clk_counter = 0;*/

reg [1:0] apb_status;
integer i;

reg ['dataWidth-1:0] mem ['addrWidth-1:0];

always @ (posedge clk or negedge reset) begin
    /*clk_counter <= clk_counter + 1;
    prdata <= clk_counter;*/
    if (reset == 0) begin
        apb_status = 0;
        prdata <= 0;
        for (i=0; i<='addrWidth; i=i+1) begin
            mem[i] = 0;
        end
    end
end

```

```

else
    if (psel == 0)
        apb_status = 0;
    else
        if (penable == 0)
            apb_status = 1;
        else
            if (pwrite == 0)
                apb_status = 2;
            else
                apb_status = 3;

    if (apb_status == 2)
        prdata = mem[addr];
    else if (apb_status == 3)
        mem[addr] = pwdata;

    startbit = mem[0];
    resetbit = mem[1];
    it_enable = mem[2];
    per_addr = mem[3];
    per_data = mem[4];

/*    $display("Status is: %d", apb_status);
    //$display("int: %d", addr_int);

    for (i=0; i<addrWidth; i=i+1)
        $display("Mem %d: %d", i, mem[i]);
    $display("-----");*/

    $display("startbit: %d", startbit);
    $display("resetbit: %d", resetbit);
    $display("it_enable: %d", it_enable);
    $display("per_addr: %d", per_addr);
    $display("per_data: %d", per_data);
    $display("-----");

end

endmodule

```

## F.4. mod\_I2C.v

```

`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:43:24 04/27/2017
// Design Name:
// Module Name:    mod_I2C
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
module mod_I2C(
    inout          SDA,          // I2C Data
    output          SCL,          // I2C clock

    input [3:0]     command,
        //0: start
        //1:
        //2: reset periphery

```

```

        //3: speed
input [7:0] address, //7 bits address + read/write bit
input [7:0] data, //1 byte data
input clk, //16MHz clk
input rst,

output reg ready
);

reg rSDA = 1;
reg rSCL = 1;

reg i2c_clk;

reg [3:0] states = 0;
reg [3:0] IDLE = 0;
reg [3:0] START = 1;
reg [3:0] STOP = 2;
reg [3:0] WRITE_ADDR = 3;
reg [3:0] READ = 4;
reg [3:0] WRITE = 5;
reg [3:0] WAIT_ADDR_ACK = 6;
reg [3:0] WAIT_DATA_ACK = 7;
reg [3:0] SEND_ACK = 8;

reg SPEED_100kBPS = 0;
reg SPEED_400kBPS = 1;

reg [3:0] byteCounter = 0;

reg [7:0] div;
reg [7:0] cnt;

//initial begin
// for (i=0; i<=addrWidth; i=i+1) begin
//     mem[i] = 0;
// end
//end

always @(posedge clk)
begin
// if (rst == 1) //reset
//     state <= 1;

case (states)
IDLE :
begin
if (1 == command[0]) //start
begin
ready <= 0; //i2c is not ready for another
communication

//set the speed of the communication
if (command[3] == SPEED_100kBPS)
begin
div <= 2; //16Mbps to 100kbps (x2) - 80
end
else
begin
div <= 1; //16Mbps to 400kbps (x2) - 20
end

cnt <= 0;
states <= START;

end

START :
begin
rSDA = 0; //pull down the wire

```

```

        if (cnt == div) //divided clock -> toggle the scl
        begin
            cnt <= 0;

            rSCL <= ~rSCL; //pull scl down

            states = WRITE_ADDR; //go to the next state

        end
        else
        begin
            cnt <= cnt + 1;
        end
    end

WRITE_ADDR :
begin
    if (cnt == div) //divided clock -> toggle the scl
    begin
        cnt <= 0;

        rSCL <= ~rSCL; //pull scl down

    end
    else
    begin
        cnt <= cnt + 1;
    end
    end
    if (i2c_clk_haf)
    begin
        if (0 == scl_monitor)
        begin
            end
        end
    end
end

endcase

$display("state: %d",states);

end

// Open Drain assignment
assign SDA = rSDA ? 1'bz : 1'b0;
assign SCL = rSCL;
//assign SCL = rSCL ? 1'bz : 1'b0;
// assign i2c_clk = (cnt == div);

endmodule

```

## F.5. tb\_top.v

```

`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    08:42:21 04/07/2017
// Design Name:    mod_top
// Module Name:    F:/RSZA/HW/tb_top.v
// Project Name:   HW
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: mod_top
//
// Dependencies:
//

```

```

// Revision:
// Revision 0.01 – File Created
// Additional Comments:
//
////////////////////////////////////

`include "macros.vh"

module tb_top();
    // Inputs
    reg PCLK;
    reg PRESETn;
    reg ['addrWidth-1:0] PADDR;
    reg PSELx;
    reg PENABLE;
    reg PWRITE;
    reg ['dataWidth-1:0] PWDATA;

    // Outputs
    wire ['dataWidth-1:0] PRDATA;

    // Instantiate the Unit Under Test (UUT)
    mod_top uut (
        .PCLK(PCLK),
        .PRESETn(PRESETn),
        .PADDR(PADDR),
        .PSELx(PSELx),
        .PENABLE(PENABLE),
        .PWRITE(PWRITE),
        .PRDATA(PRDATA),
        .PWDATA(PWDATA)
    );

    initial begin
        // Initialize Inputs
        PCLK = 0;
        PRESETn = 1;
        PADDR = 0;
        PSELx = 0;
        PENABLE = 0;
        PWRITE = 0;
        PWDATA = 0;

        // Wait 100 ns for global reset to finish
        #50;

        // Add stimulus here

        //reset
        #10 PRESETn = 0;
        #80 PRESETn = 1;

        // 1-es (resetbit) cimbe 1 bersa
        #40 PADDR = 1; PWDATA = 1;
        #5 PWRITE = 1;
        #10 PSELx = 1;
        #25 //penable csak kovetkezo ciklusban!
        #10 PENABLE = 1;
        #50

        //to idle
        #10 PENABLE = 0; PSELx = 0;

        #40

        //
        // 4-es (per_data) cimbe 144 bersa
        #10 PADDR = 4;
        #10 PWDATA = 144;
        #10 PSELx = 1;
        #10 PENABLE = 1;
    end

```

```

#40

//to idle
#10 PENABLE = 0;
#10 PSELx = 0;

#40

//kilvasas
#10 PWRITE = 0;
#10 PSELx = 1;
#10 PENABLE = 1;

#100

//to idle
#10 PENABLE = 0;
#10 PSELx = 0;

#50

#100 PRESETn = 1;

end

always begin
    #25 PCLK = ~PCLK;
end

endmodule

```