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

Rendszerarchitektúrák

Házi Feladat

*Dokumentáció*

**Periféria illesztő**

*AMBA APB 32bit/16MHz <-> SPI*

*Készítette:*

**Novoth Szilárd**

**Fehérvári Áron Botond**

**Tímár László**

*Konzulens:*

**Szántó Péter**

**2017.05.18.**

Tartalomjegyzék

[**1.** **Feladat ismertetése** 3](#_Toc483069689)

[**2.** **Specifikáció** 3](#_Toc483069690)

[2.1. AMBA APB 3](#_Toc483069691)

[2.2. SPI Interface 5](#_Toc483069692)

[2.3. Memória 5](#_Toc483069693)

[2.4. Teszt modul 6](#_Toc483069694)

[2.5. FIFO 7](#_Toc483069695)

[2.6. AMBA és az SPI kapcsolata 7](#_Toc483069696)

[**3.** **Blokkvázlat:** 7](#_Toc483069697)

[**4.** **Tesztelés, ábrák:** 8](#_Toc483069698)

[**5.** **Programkód – Verilog** 13](#_Toc483069699)

[5.1. spi.v 13](#_Toc483069700)

[5.2. spi\_interface.v 18](#_Toc483069701)

[5.3. amba.v 22](#_Toc483069702)

[5.4. top\_module.v 23](#_Toc483069703)

[5.5. alles\_test.v 25](#_Toc483069704)

[5.6. test\_bench 30](#_Toc483069705)

# **Feladat ismertetése**

A feladat egy AMBA APB buszra csatlakoztatható SPI kommunikációs modul megtervezése, melyet Verilog nyelven implementálunk. Megvalósításra került egy AMBA-busz illesztő modul, ami képes az amba felől érkező jeleket fogadni és azokat feldolgozni, egy SPI interface, ami a másik kimeneten levő kommunikációért felelős, két darab fifo, amik a két modul közti adatátvitelt biztosítják, valamint egy AMBA jeleket szimuláló teszt modul, ami a szükséges jeleket előállítja, amik a processzor felől jönnek. A megvalósítás során nem a teljes funkcionalitás megvalósítása volt a cél, hanem egy egyszerűbb, de a feladatnak eleget tevő működésű rendszer összeállítása.

# **Specifikáció**

## AMBA APB

A busz 16 MHz-es. A processzor felől a kapcsolatok a következők:

- PCLK: a busz órajele (felfutó él)

- PRESETn: reset jel

- PSELx: ezzel választható ki egy periféria. Minden egyes slave perifériának van saját ilyen jelvezetéke

- PENABLE: ez indikálja az másodlagos ciklust az APB-n. Íráskor az ún. elérési fázist biztosítja, azaz hogy az adatbuszon lévő adat érvényes. Amíg HIGH, addig nem változik a cím, adat és select jelek. Olvasáskor az az adat érvényes, mely a PENABLE HIGH állapotában kerülnek a buszra.

- PWRITE: ez jelzi a busz irányát. Ha HIGH, akkor írási, ha LOW, akkor olvasási ciklus

- PADDR: címvezeték. Ez biztosítja az egyes regiszterek kiválasztását, címzését. Max 32 bit

- PWDATA: ez az AMBA busz írandó adata (ha a PWRITE HIGH). 32 bit.

- PRDATA: olvasandó adat (ha a PWRITE LOW). 32 bit.

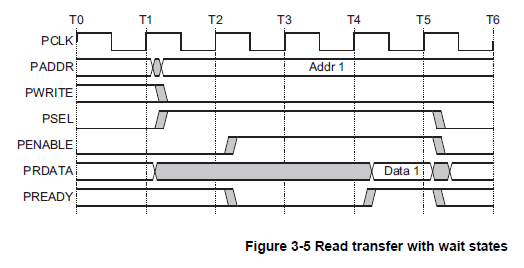
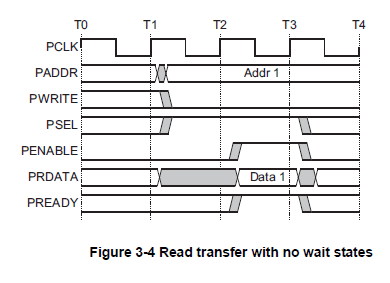
- PSTRB: A megvalósítás során nem használtuk.

- PREADY: ez a slave ready jele. Ezzel tudja jelezni, ha kész van, vagy wait-et kér.

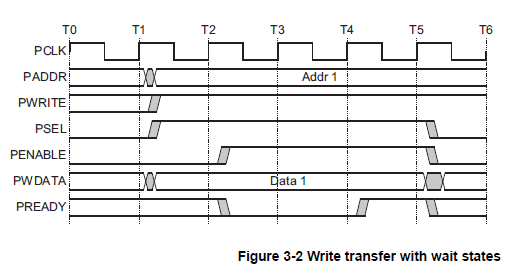
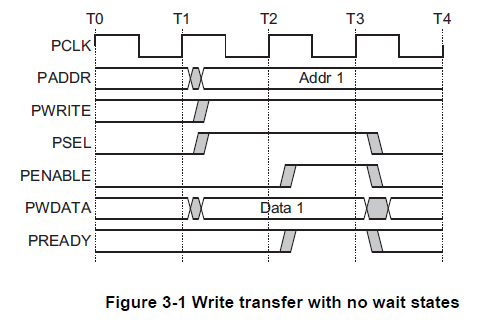
- PSLVERR: átviteli hiba jelzésére szolgál, de ne tudja mindegyik APB periféria lekezelni. A megvalósítás során nem használtuk.

- PPROT: védelem típusa. Három biten lehet állítani, hogy normál, privilegizált vagy védett a tranzakció típusa, és azt, hogy a tranzakció adat vagy instrukció elérése. A megvalósítás során nem használtuk.

Olvasási ciklus bemutatása:



Írási ciklus bemutatása:

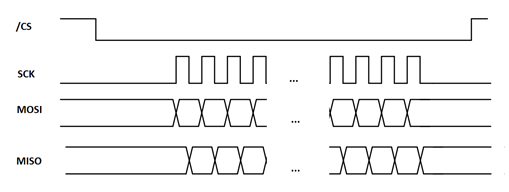


A megvalósítás során a fenti jelek alapján dolgoztunk.

## SPI Interface

A feladat SPI kommunikációt írt elő. Az SPI egy négyvezetékes kommunikáció a mi esetünkben. A CS jel felel a slave modul kiválasztásáért. Ebből lehetne több is, mi csak egy slave eszközt csatlakoztattunk. Az SCK jelen történik az órajel generálása. Ennek konfigurálhatónak kell lennie, hogy többféle eszköz is rácsatlakozhasson. A MOSI-n küldjük az adatot a slave-nak. A MISO-n fogadjuk az adatot a slave-tól.

Egy SPI kommunikáció diagramja:

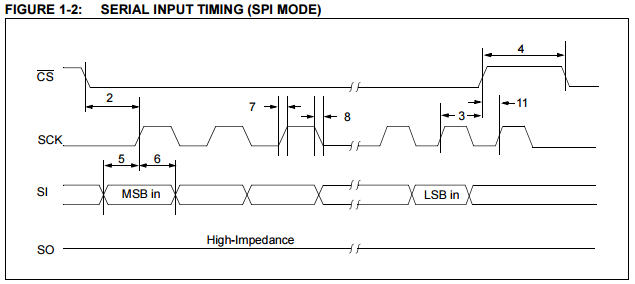


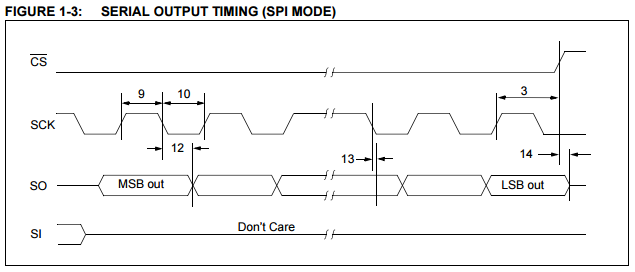
A memória más időzítési diagrammal rendelkezik.

## Memória

A feladat során kiválasztottunk egy memóriát, ami a MicroChip 23A1024[[1]](#footnote-1) nevű. 8 bit széles és 128k mély. Írási és olvasási sebessége 20 MHz. Tudja a szekvenciális írást, ami a folyamatos adatbeírást jelenti. A memória automatikusan növeli a címet ilyenkor. A memória 1 Mb-os. A honlapról letölthető hozzá egy verilog fájl, ami hozzáilleszthető a feladatunkhoz. A memória SPI kommunikációt használ, de az időzítése eltér a fent említettől, mert az SCK felfutó élére mintavételezi a jelet.

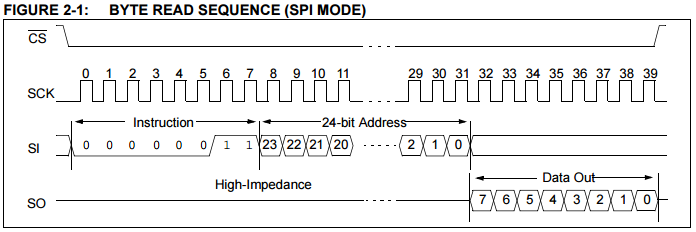
Időzítési diagram:



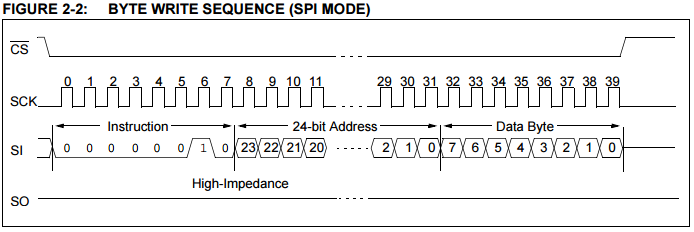


Memória olvasás:

Az ábra a bájtos írást/olvasást mutatja, de a sequence mód csak minimálisan különbözik.



Memória írás:



Ha írás van, akkor az utasítás 02h, ha olvasás van, akkor 03h. Utána 3 bájt cím következik, majd utána az adat bájtok.

## Teszt modul

A tesz modul, egy processzor szimulátor. A processzor által generált jeleket valósítja meg. Ezzel teszteli, hogy az AMBA slave jól működik-e.

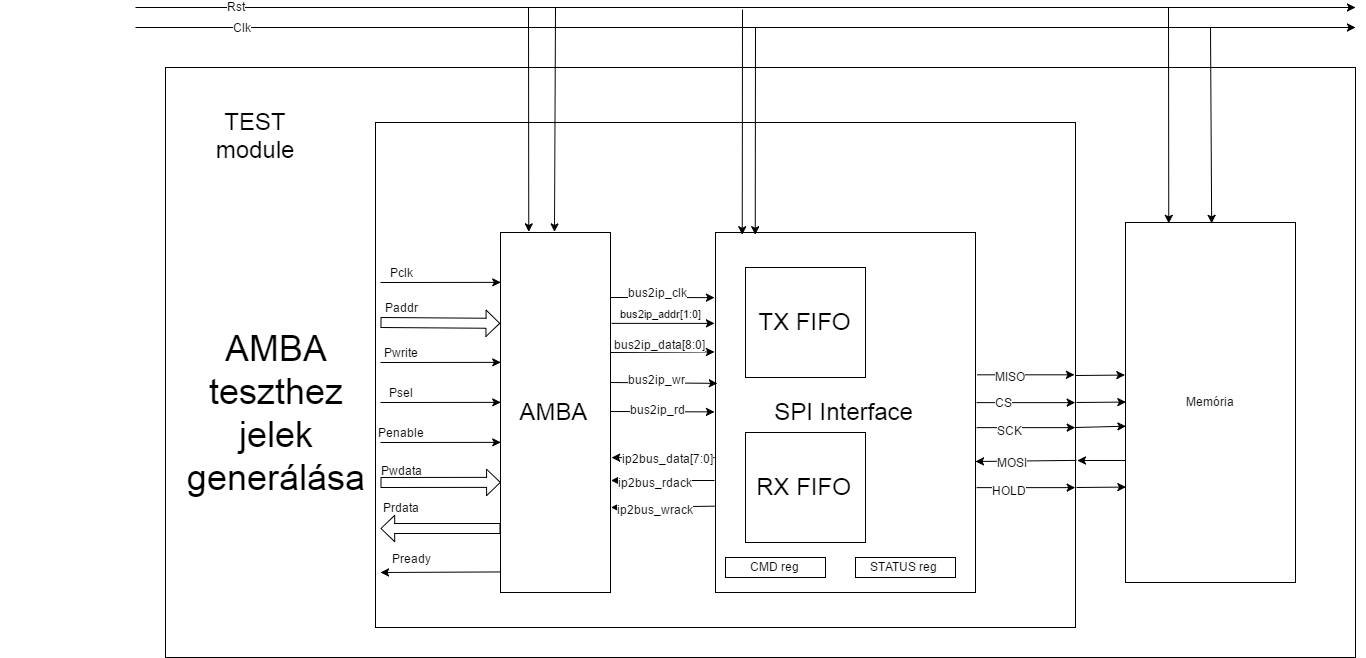
## FIFO

A FIFO-k a különböző sebességű kommunikációs egységek sebesség-különbségének az áthidalására vannak. Az AMBA nagy sebességgel tudja írni a TX FIFO-t, amiből az SPI kiveszi az adatokat és kiküldi. Amit a MOSI-n kap, azt az RX FIFO-ban eltárolja, majd azt az AMBA ki tudja olvasni. A TX és RX FIFO is 16 mély. A TX FIFO 9 bit az RX FIFO 8 bit széles.

## AMBA és az SPI kapcsolata

Az AMBA és az SPI között találhatóak a FIFO-k. A TX FIFO-ba, az AMBA beleírja az adatot. A TX FIFO 9 bites. A 9. bit azt jelzi, hogy utasítást, vagy adatot tartalmaz. Az első bájt mindig azt mondja meg, hogy hány bájtot kell kiküldeni az SPI csatornán, és hogy az SPI-on olvasunk, vagy írunk. Utána következnek a kiküldendő bájtok. Ha újabb utasítás jön, akkor az SPI interface ezt érzékeli és új kommunikációt kezd. A TX FIFO-t az SPI folyamatosan üríti, amíg van benne adat. Az RX FIFO-ba az SPI beleírja a MOSI-n vett adatokat, ha az utasításban olvasás van. Az AMBA az RX FIFO-ból kiolvassa a bejött adatokat. Az SPI rendelkezik még egy COMMAND és egy STATUS regiszterrel is. A COMMAND regisztert az AMBA tudja írni és itt tudja beállítani az SPI sebességét. A STATUS regisztert tudja olvasni az AMBA. Itt tudja lekérdezni a FIFO-k állapotát. Ezt 4 biten állítja az SPI interface. TX FIFO Full, TX FIFO Empty, RX FIFO Full, RX FIFO Empty. A COMMAND regiszter alapján állítja az SPI az SCK órajel sebességét. Ezt csak akkor tudja változtatni, ha nincs aktív kommunikáció.

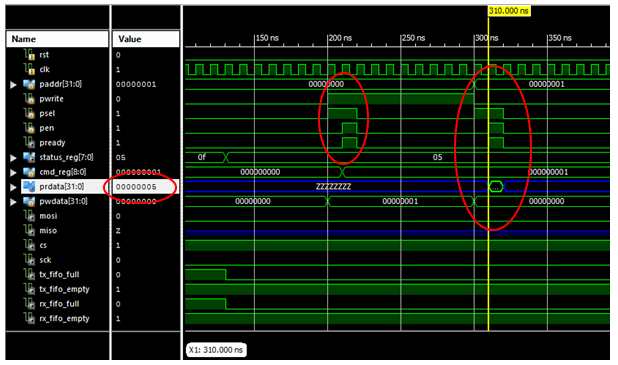
# **Blokkvázlat:**



# **Tesztelés, ábrák:**

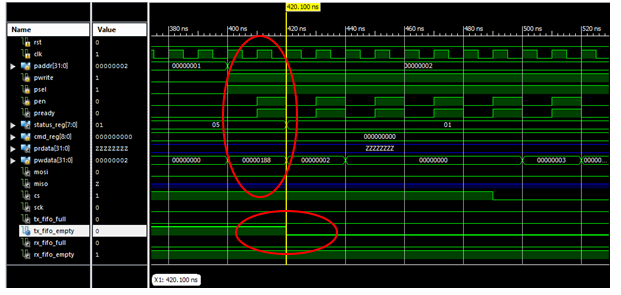
**CMD reg írás és a FIFO-k állapotának lekérdezése:**

Először a 00b címről olvasunk. Az AMBA buszon megjelenik a cím és az írni kívánt adat is. Először a pwdata 32’b1. Ez be is íródik a command regiszterbe. Utána kerül sor az status regiszter olvasására. Az a 01b címen van tárolva. A FIFO-k állapota a következő ~tx\_fifo\_full, tx\_fifo\_empty, ~rx\_fifo\_full és rx\_fifo\_empty. Tehát az utasítás, amit kiolvasunk 5h. Ez meg is jelenik az olvasás során a prdata kimeneten. A jelekből látható, hogy az írási és az olvasási ciklus két órajel alatt lezajlik. Az első órajel alatt a master beállítja a címet, psel jel kiválasztja a slave eszközt, a pwrite jel megmondja, hogy írni, vagy olvasni akarunk. A második órajel alatt a master generálja a pen jelet, amire a slave, ha készen van, akkor válaszol a pready jellel. Olvasásnál ilyenkor lesz a buszon érvényes adat.



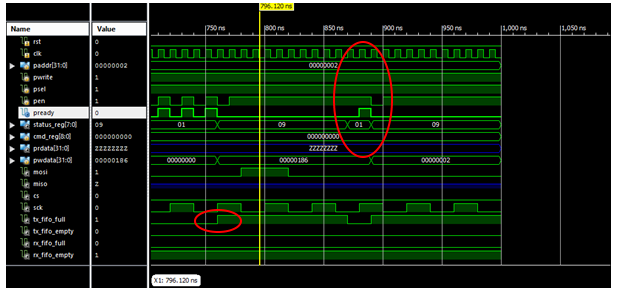
**Fifo-k feltöltése memória olvasás és írás utasítással:**

Az alábbi ábrán látható, hogy a 02b címre írunk, tehát a fifo-kat töltjük fel. Az előzőekben megismert kétciklusos írással töltjük fel a fifo-kat. Az első beírás után látható, hogy a tx\_fifo\_empty lemegy nullába, tehát az adat ténylegesen beíródott. A status regiszter is módosult, tehát az amba felé is látható, a fifo-k állapota. A kommunikáció során 32 adat kerül beírásra. miután ezeket kiküldte a master, a visszaérkező jeleket az rx fifo-ból olvassa majd.



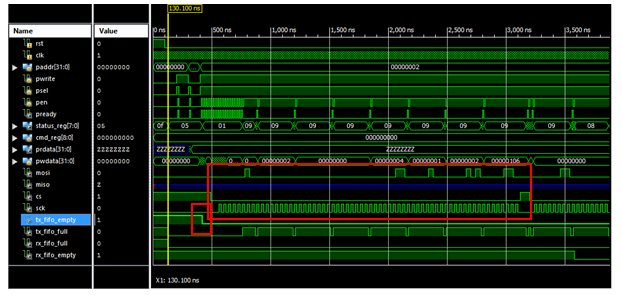
**Ha megtelt vár, amíg kiürül:**

A következő ábrán látható, hogy a slave kérhet wait állapotot, ilyenkor a master vár, amíg meg nem kapja a pready jelet. Az adatot nem tudja beírni, mert a fifo tele van. Miután a fifo nem lesz tele, a master megkapja a pready jelet és a következő adat kerül beírásra. Itt látható, hogy a slave eszköz megfelelően generálja a pready jelet. A master pedig megfelelően reagál.



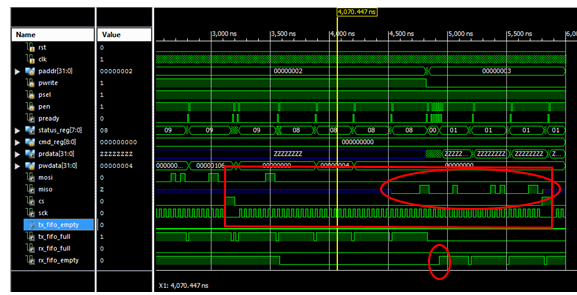
**Ha a tx fifo nem üres, elkezdődik az spi kommunikáció:**

A következő ábrán látható, hogy miután a tyx\_fifo már nem üres, ezt az SPI interface észleli és elkezdi kiküldeni a fifo-ból az adatokat a memóriának. Az első utasítás mindig az, hogy hány darab bájtot akarunk kiküldeni. Ha a fifo kiürül, de a megfelelő számú bájt még nem lett kiküldve, akkor a CS még alacsony szinten marad és nem szakad meg a kommunikáció. A memória nem is tudná értelmezni, ha a kommunikáció közben menne magasba a CS. A memória vár egy bájtnyi utasítást 02h, hogy írni, vagy olvasni akarunk. Utána következik 3 bájt cím. Jelenleg a 0 címre kezdjük az írást. Utána következnek az adatok 03h 04h 05h 06h. Ezek a cím automatikus inkrementálásával kerülnek beírásra. Miután kiment az összes bájt a kommunikáció lezárul és a CS magas állapotba kerül.



**Második egy memória olvasás:**

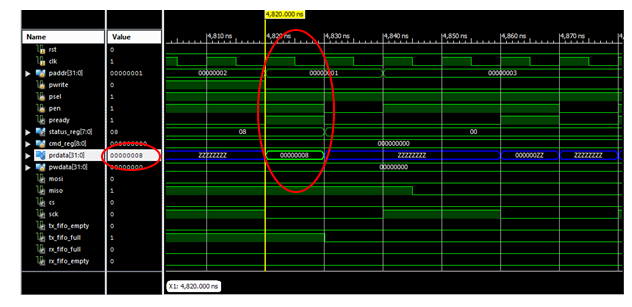
A memória írás után, visszaolvassuk ugyanarról a címről az adatokat, hogy ellenőrizzük, hogy megfelelően lettek-e beírva a memóriába. Az első bájt megint egy darabszám, hogy mennyit akarunk olvasni, és hogy olvasni akarunk. Utána az SPI elkezdi a fifo tartalmát kiküldeni. Az rx fifo-ban ilyenkor eltárolódik minden bejövő adat, ami a miso-n van. Az első kiküldött bájt megint, egy utasítás 03h, hogy olvasni akarunk. Utána a cím bájtok jönnek. Megint a 0-ás címtől kezdve olvasunk. Az utasítás és a cím kiküldése alatt is eltároljuk a bejövő adatokat, habár ezek érvénytelen értékek (ZZ). A slave eszköztől függ, hogy milyen adat jön vissza. A mi esetünkben az első négy bájt fölösleges, de ez slave eszközönként változik. Utána még négy bájtot kiküldünk. Ezek természetesen bármik lehetnek, mert olvassuk a memóriát, ezért 0-ákat küldünk ki. Mialatt a nullák kimennek, látható, hogy a memória már érvényes adatot küldd ki a kimenetén. Az rx\_fifo\_empty jel mutatja, hogy van-e már adat benne. Miután a kommunikáció lezajlott a CS magas állapotba kerül és indul a következő kommunikáció.



**Miután kiírta az összes utasítást a tx-fifo-ba, olvasási ciklusok következnek a buszon**

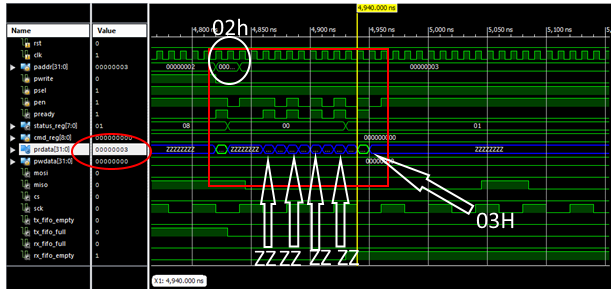
**Először a status regisztert olvassa:**

Az amba ezalatt elkezdi olvasni a status regisztert ebben látható, hogy milyen állapotban vannak a fifo-k. Ez az érték 8h, tehát a tx\_fifo tele van, de az rx fifo nem üres. Ezért lehet olvasni belőle az adatokat.

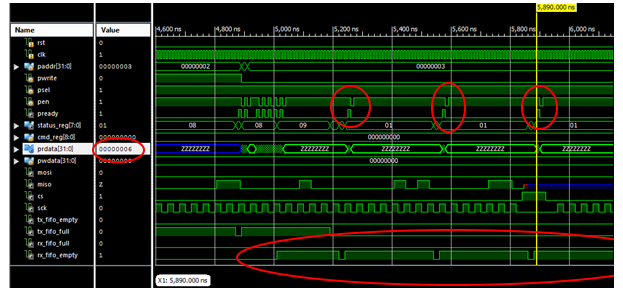


**Utána az rx fifo-t olvassa. Ha kiürült, vár az új adatra.**

Az rx\_fifo a 03h címen található. Az ábrán látható, hogy az amba megcímzi a buszt, majd kiolvassa az adatokat. Az első négy adat érvénytelen, de a következő már a 03h bájt, ami az első beírt adat a memóriába. Utána kiürült az rx\_fifo, tehát még nem zajlott le a teljes kommunikáció az SPI-on, azért nincs még meg, de a busz kiadja az olvasási parancsot és utána várja az adatot.

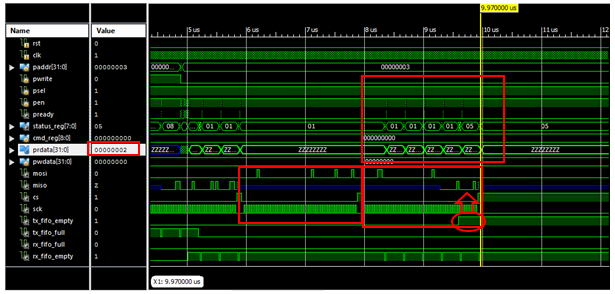


Itt látható az SPI kommunikáció is, amint a memória folyamatosan küldi az adatot a miso-n Miután egy bájtnyi adat beérkezett, az eltárolódik az rx\_fifoban. Az amba kiolvassa azt és megjelenik a buszon az érvényes adat. A következő kiolvasott értékek a 04h, 05h és a 06h.



**Új írás olvasás ciklus:**

Miután lezajlott a memória olvasása, az amba a tx\_fifoban újabb írási és olvasási utasításokat tárolt. A következő SPI kommunikáció ugyan úgy zajlik le, mint az a fenti példákban már láthattuk. Most a 4h címre fogunk írni a memóriában a 01h és a 02h bájtokat egymás után. Az SPI kommunikáció rövidebb lesz, mert kevesebb adatot kell kiküldeni. Látható az ábrán, hogy a tx\_fifo kiürül, de az SPI kommunikáció még nem áll le. Ennek oka, hogy miután az utolsó bájtot is kiolvastuk a fifo-ból, azt még ki kell küldeni. Tehát a megvalósítás során figyelni kellett arra, hogy a fifo üressége nem jelenti azt, hogy nem kell kiküldeni semmit az SPI-on. Miután az utolsó bájt is kiküldésre került, az SPI kommunikáció leáll. Ezalatt az amba busz folyamatosan olvassa az rx\_fifo-t Látható, hogy az utoljára kiolvasott bájt a 02h, amit ténylegesen beírtunk a memóriába.



A megvalósítás során azt a könnyítést kaptuk, hogy az SPI kommunikáció sebességét nem kell konfigurálhatóvá tenni. Ez a funkció végül megvalósításra került, és működik is, de a teszteket már nem futattuk le, így arról nem készült idődiagram.

# **Programkód – Verilog**

## spi.v

`timescale 1ns / 1ps

module spiv2(

input clk,

input rst,

output cs,

output sck,

input miso,

output mosi,

output [7:0] dout, //miso-n vett adat

input [8:0] din, //mosi-n küldendő adat

input [1:0] freq, //sck frekvenciáját meghatározó érték

input tx\_fifo\_empty,

input rx\_fifo\_full,

output tx\_fifo\_rd,

output rx\_fifo\_wr

);

reg rd\_wr;

//az sck frekvenciájának beállítása

// kommunikáció alatt nem lehetséges

reg [1:0] frq;

always@(posedge clk)

if (cs\_delayed)

frq<=freq;

reg [4:0] cntr;

always@ (posedge clk)

if(rst)

cntr <= 0;

else

cntr <= cntr+1'b1;

////////

/\*

frq = 00 -> /2 órajel osztás

frg = 01 -> /4 órajel osztás

frg = 10 -> /8 órajel osztás

frg = 11 -> /16 órajel osztás

\*/

// SPI órajel

wire sck\_temp;

assign sck\_temp = (frq[1]) ? (frq[0] ? cntr[4] : cntr[3]) : (frq[0] ? cntr[2] : cntr[1]);

// SPI órajel felfutó él lefutó él

reg sck\_reg;

always@(posedge clk)

if(rst)

sck\_reg <= 0;

else

sck\_reg <= sck\_temp;

wire sck\_rise;

wire sck\_fall;

assign sck\_fall = ((sck\_reg == 1 && sck\_temp == 0) & ~cs\_delayed[2] & (datapos != 4'b0000));

assign sck\_rise = ((sck\_reg == 0 && sck\_temp == 1) & ~cs\_delayed[2] );

reg delaysck;

always@(posedge clk)

delaysck <= sck\_temp;

`define IDLE 4'b0000

`define COMMAND 4'b0001

`define DATA 4'b0010

`define READY 4'b0011

`define WAIT 4'b0100

reg [1:0]delayed\_mosi;

always@ (posedge clk)

delayed\_mosi <= { delayed\_mosi[0] , (`DATA == status | `WAIT == status)? temp\_mosi[7] : 1'b1};

assign mosi = (datapos != 4'b0)? delayed\_mosi[1] : 1'b0;

assign sck = ((`DATA == status)? delaysck : 1'b0) & ~cs\_delayed[2] & ~( datapos[3:0] == 4'b0000 ) & (datacntr != 8'b0);

reg [8:0] cs\_delayed;

always@ (posedge clk)

cs\_delayed <= { cs\_delayed[7:0] ,(`DATA == status)? 1'b0 : 1'b1};

assign cs = (`DATA == status | `WAIT == status) ? ((datacntr != 8'b0)? 1'b0 : 1'b1 ) : 1'b1;

reg [3:0] status;

reg [7:0] datacntr;

reg [7:0] mosiout;

reg [3:0] datapos;

always@ (posedge clk)

if(rst)

datacntr <= 0;

else

if(status == `COMMAND & din[8] == 1'b1)

datacntr <= din[6:0];

else

if(status == `DATA)

if(datapos == 4'b1000 & sck\_rise)

datacntr <= datacntr - 1'b1;

always@ (posedge clk)

if (rst)

rd\_wr <= 0;

else if(status == `COMMAND & din[8] == 1'b1)

rd\_wr <= din[7];

reg tx\_fifo\_rd\_reg;

always @ (posedge clk)

if (rst)

tx\_fifo\_rd\_reg <= 0;

else

begin

if(tx\_fifo\_rd\_reg == 1'b1 & tx\_fifo\_empty)

tx\_fifo\_rd\_reg <= 1;

else if(tx\_fifo\_rd\_reg == 1'b1 & ~tx\_fifo\_empty)

tx\_fifo\_rd\_reg <= 0;

else if(tx\_fifo\_empty & status == `DATA & datapos == 4'b1000 & sck\_fall & datacntr != 8'h01)

tx\_fifo\_rd\_reg <= 1;

else

begin

if(~tx\_fifo\_empty)

if(status == `COMMAND)

tx\_fifo\_rd\_reg <= 1'b1;

else if(datapos == 4'b1000 & sck\_fall & datacntr != 8'h01)

tx\_fifo\_rd\_reg <= 1'b1;

else if(datapos == 4'b0 & status == `WAIT)

tx\_fifo\_rd\_reg <= 1'b1;

end

end

assign tx\_fifo\_rd = tx\_fifo\_rd\_reg;

reg delayed\_dout;

always@ (posedge clk)

delayed\_dout <= (datapos == 4'b1000 & sck\_rise) ? 1'b1 : 1'b0;

reg rx\_fifo\_wr\_reg;

always@ (posedge clk)

if(rst)

rx\_fifo\_wr\_reg <= 0;

else if(rx\_fifo\_wr\_reg)

rx\_fifo\_wr\_reg <= 0;

else

if(~rd\_wr)

begin

if(delayed\_dout)

rx\_fifo\_wr\_reg <= 1;

end

assign rx\_fifo\_wr = rx\_fifo\_wr\_reg & ~rd\_wr;

reg [7:0] tx\_fifo\_rd\_delay;

always@ (posedge clk)

tx\_fifo\_rd\_delay <= { tx\_fifo\_rd\_delay[6:0] , tx\_fifo\_rd };

wire test;

assign test = ~(|tx\_fifo\_rd\_delay);

always@ (posedge clk)

if(rst)

datapos <= 0;

else

if(cs)

datapos <= 0;

else

begin

if(sck\_rise)

if(datapos == 4'b1000 & ~tx\_fifo\_empty)

datapos <= 4'b0001;

else if(datapos == 4'b1000 & tx\_fifo\_empty & (|tx\_fifo\_rd\_delay))

datapos <= 4'b1;

else if(datapos == 4'b1000 & tx\_fifo\_empty)

datapos <= 4'b0;

else

datapos <= datapos + 1'b1;

end

always@ (posedge clk)

if (rst)

begin

status <= `IDLE;

end

else

case(status)

`IDLE : if(~tx\_fifo\_empty)

status <= `READY;

`READY : if(din[8] == 1 | datacntr == 8'b0)

status <= `COMMAND;

else

status <= `DATA;

`COMMAND : if(din[8] == 1)

begin

status <= `DATA;

rd\_wr <= din[7];

end

else

status <= `DATA;

`DATA : begin

if(datacntr == 8'b0)

status <= `IDLE;

else if(datapos == 4'b1000 & datacntr != 8'b0 & tx\_fifo\_empty & sck\_rise & tx\_fifo\_rd)

status <= `WAIT;

else status <= `DATA;

end

`WAIT : if(~tx\_fifo\_empty) status <= `DATA;

endcase

reg [8:0] temp\_mosi;

reg [8:0] temp\_miso;

always@ (posedge clk)

if(rst)

begin

temp\_mosi <= 0;

temp\_miso <= 0;

end

else

begin

if(datapos == 4'b0)

temp\_mosi <= {1'b0, din[7:0]};

if(datapos == 4'b1000 & ~tx\_fifo\_empty & sck\_rise)

temp\_mosi <= {din[7:0], 1'b0};

else

begin

if(sck\_reg == 0 & sck\_temp == 1)

temp\_mosi <= {temp\_mosi[7:0] , 1'b0};

end

if(sck\_fall)

temp\_miso <= {temp\_miso[7:0] , miso};

end

assign dout = (rst)? (8'b0) : (datapos == 4'b1000 & sck\_rise)? temp\_miso[7:0] : dout;

endmodule

## spi\_interface.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 13:21:18 05/09/2017

// Design Name:

// Module Name: spi\_interface

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module spi\_interface(

//amba felol

input bus2ip\_clk,

input rst,

input [8:0] bus2ip\_data,

input [1:0] bus2ip\_addr,

input bus2ip\_wr,

input bus2ip\_rd,

output [7:0] ip2bus\_data,

output ip2bus\_rdack,

output ip2bus\_wrack,

//memory fele

input mosi,

output HOLD\_ENABLE,

output miso,

output sck,

output cs

);

assign HOLD\_ENABLE = 1'b1;

wire clk;

assign clk = bus2ip\_clk;

reg [8:0] cmd\_reg;

wire [7:0] status\_reg;

reg [1:0] state;

reg [7:0] tempip2bus\_data;

reg t\_ip2bus\_wrack;

reg t\_ip2bus\_rdack;

assign status\_reg = {4'b0,tx\_fifo\_full,tx\_fifo\_empty,rx\_fifo\_full,rx\_fifo\_empty};

reg [1:0]rdack\_help;

always@ (posedge clk)

if (rst)

begin

cmd\_reg <= 0;

state <= 0;

t\_ip2bus\_wrack <= 0;

t\_ip2bus\_rdack <= 0;

tx\_fifo\_datain <= 0;

tx\_fifo\_wr <= 0;

rx\_fifo\_rd <= 0;

end

else if (state == 2'b00)

begin

if(bus2ip\_wr)

begin

if(bus2ip\_addr == 2'b00)

begin

cmd\_reg <= bus2ip\_data;

t\_ip2bus\_wrack <= 1;

state <= 2'b01;

end

else if(bus2ip\_addr == 2'b10)

begin

if(tx\_fifo\_full)

begin

t\_ip2bus\_wrack <= 0;

end

else

begin

state <= 2'b01;

tx\_fifo\_datain <= bus2ip\_data;

tx\_fifo\_wr <= 1;

t\_ip2bus\_wrack <= 1;

end

end

else if(bus2ip\_addr == 2'b01)

begin

state <= 2'b01;

t\_ip2bus\_wrack <= 1;

end

else if(bus2ip\_addr == 2'b11)

begin

state <= 2'b01;

t\_ip2bus\_wrack <= 1;

end

end

else if(bus2ip\_rd)

begin

if(bus2ip\_addr == 2'b00)

begin

state <= 2'b11;

t\_ip2bus\_rdack <= 1;

end

else if(bus2ip\_addr == 2'b01)

begin

tempip2bus\_data <= status\_reg;

state <= 2'b11;

t\_ip2bus\_rdack <= 1;

end

else if(bus2ip\_addr == 2'b10)

begin

state <= 2'b11;

t\_ip2bus\_rdack <= 1;

end

else if(bus2ip\_addr == 2'b11)

begin

if(~rx\_fifo\_empty)

begin

state <= 2'b11;

t\_ip2bus\_rdack <= 1;

rx\_fifo\_rd <= 1;

tempip2bus\_data <= rx\_fifo\_dataout;

rdack\_help <= {rdack\_help[0], 1'b1};

end

else

begin

t\_ip2bus\_rdack <= 0;

end

end

end

end

else

begin

if(tx\_fifo\_wr) begin tx\_fifo\_wr <= 0; state <= 0; end

if(rx\_fifo\_rd) begin rx\_fifo\_rd <= 0;end

if(state == 2'b11)

begin

t\_ip2bus\_rdack <= 0;

state <= 0;

end

if(~bus2ip\_wr & state == 2'b01)

begin

t\_ip2bus\_wrack <= 0;

state <= 0;

end

end

wire t\_ip\_rd\_reg;

assign t\_ip\_rd\_reg = (bus2ip\_addr == 2'b11)? ((t\_ip2bus\_rdack)? 1'b1 : 1'b0) : (1'b0);

reg delay\_rdack;

always@( \* )

delay\_rdack <= t\_ip\_rd\_reg;

reg delay\_rdack2;

always@(posedge clk)

delay\_rdack2 <= delay\_rdack;

reg delay\_rdack3;

always@(posedge clk)

delay\_rdack3 <= delay\_rdack2;

reg delay\_rdack4;

always@(posedge clk)

delay\_rdack4 <= delay\_rdack3;

assign ip2bus\_data = (bus2ip\_addr == 2'b11)? rx\_fifo\_dataout[7:0] : status\_reg[7:0];

assign ip2bus\_wrack = t\_ip2bus\_wrack;

assign ip2bus\_rdack = (bus2ip\_addr == 2'b11)? delay\_rdack3 : t\_ip2bus\_rdack;

wire tx\_fifo\_full;

reg [8:0] tx\_fifo\_datain;

wire [8:0] tx\_fifo\_dataout;

reg tx\_fifo\_wr;

wire tx\_fifo\_rd;

wire tx\_fifo\_empty;

wire rx\_fifo\_full;

wire [7:0] rx\_fifo\_datain;

wire [7:0] rx\_fifo\_dataout;

wire rx\_fifo\_wr;

reg rx\_fifo\_rd;

wire rx\_fifo\_empty;

wire sck;

tx\_fifo tx\_fifo (

.rst(rst),

.clk(clk),

.din(tx\_fifo\_datain),

.dout(tx\_fifo\_dataout),

.wr\_en(tx\_fifo\_wr),

.full(tx\_fifo\_full),

.rd\_en(tx\_fifo\_rd),

.empty(tx\_fifo\_empty)

);

rx\_fifo rx\_fifo (

.rst(rst),

.clk(clk),

.din(rx\_fifo\_datain),

.dout(rx\_fifo\_dataout),

.wr\_en(rx\_fifo\_wr),

.full(rx\_fifo\_full),

.rd\_en(rx\_fifo\_rd),

.empty(rx\_fifo\_empty)

);

spiv2 spi(

.clk(clk),

.rst(rst),

.cs(cs),

.sck(sck),

.miso(miso),

.mosi(mosi),

.dout(rx\_fifo\_datain), //miso-n vett adat

.din(tx\_fifo\_dataout), //mosi-n küldendő adat

.freq(cmd\_reg [1:0] ), //sck frekvenciáját meghatározó érték

.tx\_fifo\_empty(tx\_fifo\_empty),

.rx\_fifo\_full(rx\_fifo\_full),

.tx\_fifo\_rd(tx\_fifo\_rd),

.rx\_fifo\_wr(rx\_fifo\_wr)

);

endmodule

## amba.v

module amba(

input clk,

input rst,

input [31:0] addr,

input pwrite,

input psel,

input pen,

input [31:0] pwdata,

output [31:0] prdata,

output pready,

output bus2ip\_clk,

output [1:0] bus2ip\_addr,

output [8:0] bus2ip\_data,

output bus2ip\_wr,

output bus2ip\_rd,

input [7:0] ip2bus\_data,

input ip2bus\_rdack,

input ip2bus\_wrack

);

assign bus2ip\_clk = clk;

assign bus2ip\_addr = addr[1:0];

assign bus2ip\_data = pwdata[8:0];

assign bus2ip\_wr = pwrite & psel;

assign bus2ip\_rd = ~pwrite & psel;

reg [1:0] cntr;

always@ ( \* )

if (rst)

cntr <= 2'b0;

else

begin

if(psel & ~pen) cntr <= 2'b01;

if(psel & pen) cntr <= 2'b10;

if(~psel & ~pen) cntr <= 2'b00;

end

assign pready = (cntr == 2'b10)? (ip2bus\_wrack | ip2bus\_rdack) : 0 ;

assign prdata = (~pwrite & pready & psel & cntr == 2'b10) ? {24'b0 , ip2bus\_data} : 32'bz;

endmodule

## top\_module.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 10:50:29 05/09/2017

// Design Name:

// Module Name: top\_module

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module top\_module(

input rst,

input clk,

//memoria

input miso,

output mosi,

output sck,

output cs,

output HOLD\_ENABLE,

//amba

input pclk,

input [31:0] paddr,

input pwrite,

input psel,

input pen,

input [31:0] pwdata,

input [31:0] prdata,

output pready

);

wire bus2ip\_clk;

wire [1:0] bus2ip\_addr;

wire [8:0] bus2ip\_data;

wire bus2ip\_wr;

wire bus2ip\_rd;

wire [7:0] ip2bus\_data;

wire ip2bus\_rdack;

wire ip2bus\_wrack;

amba amba(

.clk(clk),

.rst(rst),

.addr(paddr),

.pwrite(pwrite),

.psel(psel),

.pen(pen),

.pwdata(pwdata),

.prdata(prdata),

.pready(pready),

.bus2ip\_clk(bus2ip\_clk),

.bus2ip\_addr(bus2ip\_addr),

.bus2ip\_data(bus2ip\_data),

.bus2ip\_wr(bus2ip\_wr),

.bus2ip\_rd(bus2ip\_rd),

.ip2bus\_data(ip2bus\_data),

.ip2bus\_rdack(ip2bus\_rdack),

.ip2bus\_wrack(ip2bus\_wrack)

);

spi\_interface spi\_interface(

.bus2ip\_clk(bus2ip\_clk),

.rst(rst),

.bus2ip\_data(bus2ip\_data),

.bus2ip\_addr(bus2ip\_addr),

.bus2ip\_wr(bus2ip\_wr),

.bus2ip\_rd(bus2ip\_rd),

.ip2bus\_data(ip2bus\_data),

.ip2bus\_rdack(ip2bus\_rdack),

.ip2bus\_wrack(ip2bus\_wrack),

//memory fele

.mosi(mosi),

.HOLD\_ENABLE(HOLD\_ENABLE),

.miso(miso),

.sck(sck),

.cs(cs)

);

endmodule

## alles\_test.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 09:35:18 05/15/2017

// Design Name:

// Module Name: alles\_test

// Project Name:

// Target Devices:

// Tool versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module alles\_test(

input rst,

input clk,

input enable,

input [2:0] controll

);

reg [31:0] address [0:3];

reg [31:0] data [0:31];

initial begin

address[0] = 32'h0000\_0000;

address[1] = 32'h0000\_0001;

address[2] = 32'h0000\_0002;

address[3] = 32'h0000\_0003;

data[0] = 32'h0000\_0188;

data[1] = 32'h0000\_0002;

data[2] = 32'h0000\_0000;

data[3] = 32'h0000\_0000;

data[4] = 32'h0000\_0000;

data[5] = 32'h0000\_0003;

data[6] = 32'h0000\_0004;

data[7] = 32'h0000\_0005;

data[8] = 32'h0000\_0006;

data[9] = 32'h0000\_0108;

data[10] = 32'h0000\_0003;

data[11] = 32'h0000\_0000;

data[12] = 32'h0000\_0000;

data[13] = 32'h0000\_0000;

data[14] = 32'h0000\_0000;

data[15] = 32'h0000\_0000;

data[16] = 32'h0000\_0000;

data[17] = 32'h0000\_0000;

data[18] = 32'h0000\_0186;

data[19] = 32'h0000\_0002;

data[20] = 32'h0000\_0000;

data[21] = 32'h0000\_0000;

data[22] = 32'h0000\_0004;

data[23] = 32'h0000\_0001;

data[24] = 32'h0000\_0002;

data[25] = 32'h0000\_0106;

data[26] = 32'h0000\_0003;

data[27] = 32'h0000\_0000;

data[28] = 32'h0000\_0000;

data[29] = 32'h0000\_0004;

data[30] = 32'h0000\_0000;

data[31] = 32'h0000\_0000;

end

`define IDLE 3'b000

`define WRITE\_CMD 3'b001

`define RD\_STATUS 3'b010

`define WR\_DATA 3'b011

`define RD\_DATA 3'b100

wire miso;

reg [31:0] paddr;

reg pwrite;

reg psel;

reg pen;

reg [31:0] pwdata;

wire [31:0] prdata;

// Outputs

wire mosi;

wire sck;

wire cs;

wire HOLD\_ENABLE;

wire pready;

// Instantiate the Unit Under Test (UUT)

top\_module uut (

.rst(rst),

.clk(clk),

.miso(miso),

.mosi(mosi),

.sck(sck),

.cs(cs),

.HOLD\_ENABLE(HOLD\_ENABLE),

.pclk(clk),

.paddr(paddr),

.pwrite(pwrite),

.psel(psel),

.pen(pen),

.pwdata(pwdata),

.prdata(prdata),

.pready(pready)

);

M23A1024 memoria(

.SI\_SIO0(mosi),

.SO\_SIO1(miso),

.SCK(sck),

.CS\_N(cs),

.SIO2(),

.HOLD\_N\_SIO3(HOLD\_ENABLE),

.RESET()

);

//////////////////////////AMBA regiszter vezérlő

reg [4:0] cntr;

always@ ( \* )

case(controll)

`IDLE :

begin

pwrite <= 0;

paddr <= address[0];

pwdata <= 32'h0000\_0000;

end

`WRITE\_CMD :

begin

pwrite <= 1;

paddr <= address[0];

pwdata <= 32'h0000\_0000;

end

`RD\_STATUS :

begin

pwrite <= 0;

paddr <= address[1];

pwdata <= 32'b0000\_0000;

end

`WR\_DATA :

begin

pwrite <= 1;

paddr <= address[2];

pwdata <= data[cntr];

end

`RD\_DATA :

begin

pwrite <= 0;

paddr <= address[3];

pwdata <= 32'b0000\_0000;

end

endcase

///////////////////////AMBA állapotgép

`define AMBAIDLE 2'b00

`define AMBASETUP 2'b01

`define AMBAACCESS 2'b10

/////////////////////AMBA állapotgép

reg [1:0] ambastate;

always@ ( \* )

case(ambastate)

`AMBAIDLE :

begin

psel <= 0;

pen <= 0;

end

`AMBASETUP :

begin

psel <= 1;

pen <= 0;

end

`AMBAACCESS :

begin

psel <= 1;

pen <= 1;

end

endcase

always@ (posedge clk)

if(rst)

begin

ambastate <= `AMBAIDLE;

cntr <= 0;

end

else

begin

if(enable)

ambastate <= `AMBASETUP;

if(ambastate == `AMBASETUP)

ambastate <= `AMBAACCESS;

if(ambastate == `AMBAACCESS)

begin

if(~pready)

ambastate <= `AMBAACCESS;

else if(pready & enable)

begin

ambastate <= `AMBASETUP;

if(pwrite & controll == `WR\_DATA)

cntr <= cntr+1;

end

else if(pready & ~enable)

begin

ambastate <= `AMBAIDLE;

if(pwrite & controll == `WR\_DATA)

cntr <= cntr+1;

end

end

end

////////////////////////////////AMBA állapotgép vége

endmodule

## test\_bench

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 11:07:13 05/17/2017

// Design Name: alles\_test

// Module Name: C:/gitproject/rarchhazi/spi\_rom\_proba/test\_top\_mmodule\_alles.v

// Project Name: spi\_rom\_proba

// Target Device:

// Tool versions:

// Description:

//

// Verilog Test Fixture created by ISE for module: alles\_test

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

////////////////////////////////////////////////////////////////////////////////

module test\_top\_mmodule\_alles;

// Inputs

reg rst;

reg clk;

reg enable;

reg [2:0] controll;

// Instantiate the Unit Under Test (UUT)

alles\_test uut (

.rst(rst),

.clk(clk),

.enable(enable),

.controll(controll)

);

`define IDLE 3'b000

`define WRITE\_CMD 3'b001

`define RD\_STATUS 3'b010

`define WR\_DATA 3'b011

`define RD\_DATA 3'b100

initial begin

// Initialize Inputs

rst = 1;

clk = 1;

enable = 0;

controll = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

rst = 0;

#100;

controll = `WRITE\_CMD;

enable = 1;

#80;

enable = 0;

#70;

controll = `RD\_STATUS;

enable = 1;

#80;

enable = 0;

#40;

controll = `WR\_DATA;

enable = 1;

#4420;

enable = 0;

controll = `RD\_STATUS;

enable = 1;

#20;

enable = 0;

//#20;

controll = `RD\_DATA;

enable = 1;

#40;

#800000;

end

always#5 clk <= ~clk;

endmodule

1. http://www.microchip.com/wwwproducts/en/23A1024 [↑](#footnote-ref-1)