Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Beágyazott Információs Rendszerek MSc. szakirány

Rendszerarchitektúrák Házi Feladat

AMBA APB – USRT periféria illesztő

Készítették:

Mező Dániel (NMNMMI)

Sipos-Takáts Bence László (T35NOC)

Konzulens:

Fehér Béla

Budapest, 2010.05.12.

AMBA-USRT specifikáció

Az USRT egy kétvezetékes, half-duplex, szinkron soros interfész. A két vezeték közül egy a vételnek (Receive - RxD), egy az adásnak (Transmit – TxD) van dedikálva. Ezen kívül adott egy órajel vezeték, mely a szinkron adásvételt biztosítja, azaz az adatvezetékek mintavételezési időpontjait.

Az AMBA protokoll család APB busza egy olyan interfész, mely alacsony energiaigényű és komplexitású perifériákra optimalizáltak. A következő jelvezetékekkel rendelkezik:

- PCLK: a busz órajele (felfutó él)
- PRESETn: reset jel (Aktív-LOW)
- 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). Mivel 8 bites a busz, és egy USRT üzenet ennyit továbbít*
- PRDATA: olvasandó adat (ha a PWRITE LOW). Ez is 8 bites
- PSTRB: íráskor ez adja meg, hogy melyik bájton történt változás. Mivel jelenleg 8 bites az adatbusz, ez egy egybites jelvezeték. Olvasáskor nem lehet aktív
- 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
- PPROT: védelem típusa. Három biten lehet állítani, hogy normál, privilegizált vagy védett a tranzakció típusa, és azt, hogy a tranzakció adat vagy instrukció elérése. Jelen esetben a komplexitás alacsony szintje miatt nincs szükség rá, ezért fixen normál, nem védett adat típusú

Ezen kívül még szükséges egy interrupt bekötése, de ez az AMBA-n kívül, közvetlenül a processzorba vezet. Ezzel lehet az USRT Receive IT-t lekezelni.

* Igaz a busz 32 bites, azonban a specifikációban nincs megszabva, csak maximális méret van megadva, tehát lehet ennél kevesebb is. Az USRT sebessége miatt nincs szükség a teljes szélességre, hiszen az AMBA busz így is jóval gyorsabb.

Vétel: ha engedélyezve van a TxD bemenet, egy 8 bites shiftregiszter mintavételezi és alakítja át párhuzamos jelre. Ha beérkezett egy üzenet, vagyis 8 bit, lementi egy regiszterbe. Ez a regiszter felelős a PREADY jel kiadásáért. Kimenete a PRDATA busz. Ezen felül a korábban említett RxD_IT jel is ki van vezetve, ez a jelzés, ha érkezett új adat.

A blokk még tartalmaz egy Paritásellenőrzőt, mely az Utasítás és állapotregiszter megfelelő bitjét állítja paritáshiba esetén 1-be. Ennek ellenőrzését a magas szintű programnak kell lekezelnie.

Adás: ha engedélyezve van a kimenet, akkor először az APB által az adatregiszterjébe írt adatot küldi ki bitenként (sorosan), majd az Utasítás és állapotregiszterben (CMD/ST) előre definiált, megfelelő paritás- és stopbit(ek)et. A küldés órajelét a Baud rate generátor állítja elő, mely értékét szintén a CMD/ST állítja be.

Órajel: ha az USRT master-ként viselkedik, akkor ő állítja elő az órajelet, ekkor engedélyezve van a buffer, azonban ha slave-ként viselkedik, akkor kívülről érkező órajel alapján mintavételez.

Regiszter engedélyezés: ha a periféria ki van választva, a PWRITE, PENABLE és a PADDR jelvezetékek segítségével a megfelelő regiszterek megfelelő memóriaterületeit engedélyezni lehet. Ez a blokk felel a címdekódolásért (CS – Chip Select)

Utasítás és állapotregiszter: ez kezeli le az USRT működését. Két bájttal lehet szabályozni az állapotát (utasítás); ezeket a bájtokat azonban vissza is lehet olvasni (állapot). Az, hogy olvasva vagy írva van, a CMDRW/RDn jeltől függ. A CMD/ST felel a kimenetek engedélyezéséért 4 biten, a paritás (nincs, páros, páratlan) és a stopbit (1 vagy 2) jellegéért 4 biten. Ezen kívül még a baud rate beállításáért (3 bit) és paritáshiba visszajelzéséért (1 bit).

Ezeken felül képes az USRT regisztereinek kódszó alapján való reset-elésre.

Felépítése:

Első kódszó (ADDR [31:30] = 00 címen)
Második kódszó (ADDR [31:30] = 01 címen)

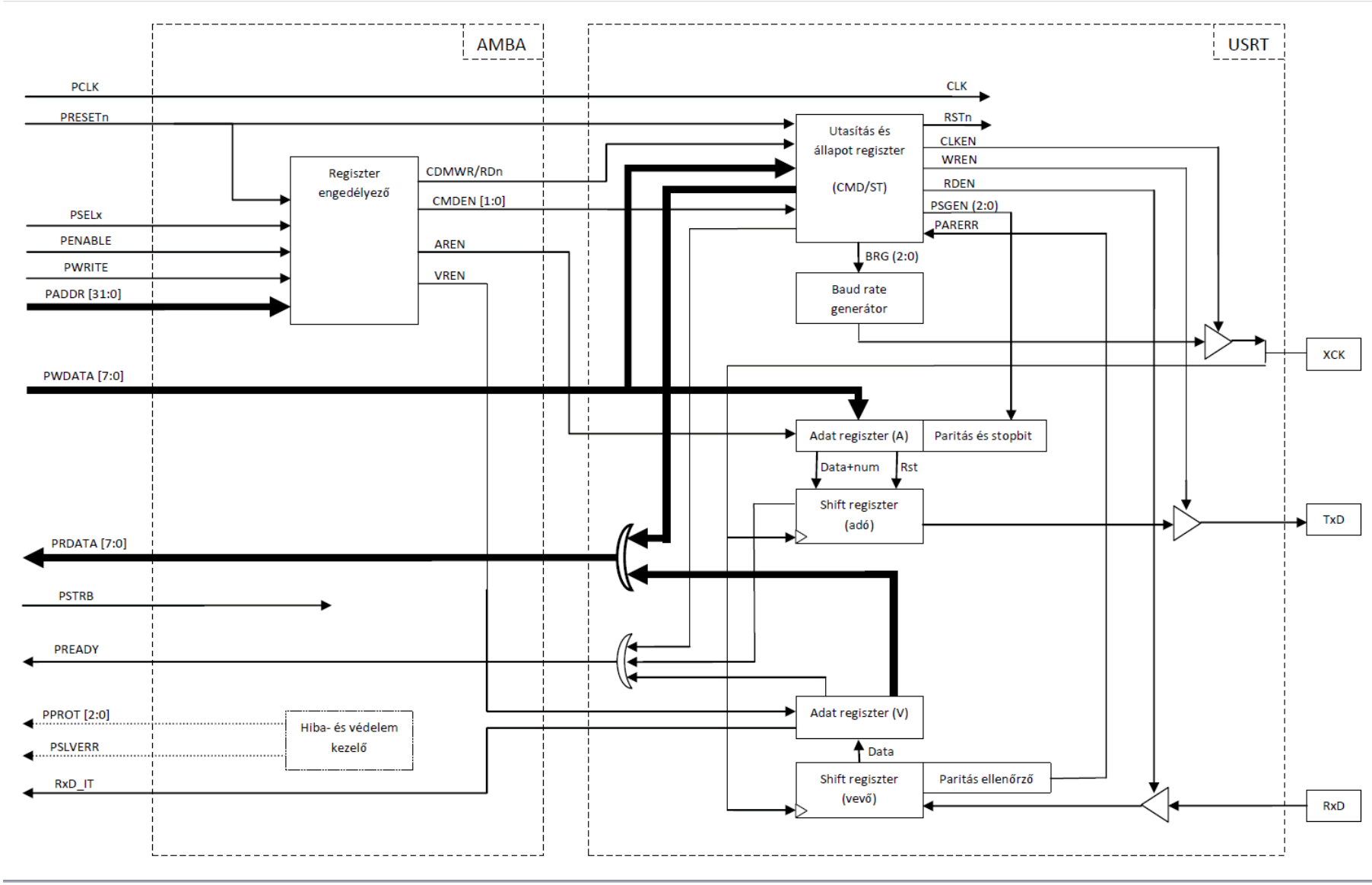
-	Stopbit	Páratlan	Páros	RDEn	WREn	-	ClkEn
-	-	-	Rst	Paritás	Baud	Baud	Baud

A stopbit-tel lehet megadni a stopbitek számát, értéke 0, ha egy stopbit van, 1, ha kettő. Balról jobbra haladva a következő két cella a paritásra vonatkozik, majd pedig a kimenetek buffereinek engedélyezői. Jobb oldalt pedig a korább említett órajel választás attól függően, ki a master.

Az Rst egybe állításával érhető el az USRT kódszavas resetelése. A paritás egy olvasható bit, mely 1 értékű, ha a vétel során paritáshiba lépett fel. Természetesen nem ha nincs beállítva paritás, ez az érték mindig 0. A következő és egyben utolsó 3 bittel lehet a baud rate-et beállítani, ezt jelen esetben 20MHz-es órajelhez méreteztük, de néhány konstans átírásával tetszőleges órajelre megvalósíthatók a szabványos értékek. Az üres helyek fenn vannak tartva későbbi fejlesztés lehetőségére, például ha új adat érkezik, és az felülírta a régi, egy bitet be lehet állítani.

Következő oldalon látható a rendszerterv blokkon szintjén. A tervezés ez alapján ment végbe, a programkód és az egységek összeköttetése ezen alapszik.

Látható, hogy két fő részből áll, egyik az AMBA, mely tartalmazza a címdekóder áramkört, másik az USRT, mely a periféria lekezelésére szolgál. Ebben található az állapot és utasításregiszter, a baud rate generátor, az adó és vevőegység, valamint a bufferek.


AMBA
USRT
PCLK
PRESETn
PSELx
PENABLE
PWRITE
PADDR [31:0]
PWDATA [7:0]
PRDATA [7:0]
PSTRB
PREADY
PPROT [2:0]
PSLVERR
RxD_IT
Regiszter engedélyező
CDMWR/RDn
CMDEN [1:0]
AREN
VREN
Hiba- és védelem kezelő
CLK
Utasítás és állapot regiszter
(CMD/ST)
RSTn
CLKEN
WREN
RDEN
PSGEN (2:0)
PARERR
BRG (2:0)
Baud rate generátor
XCK
Adat regiszter (A)
Paritás és stopbit
Data+num
Rst
Shift regiszter (adó)
TxD
Adat regiszter (V)
Data
Shift regiszter (vevő)
Paritás ellenőrző
RxD

Az elkészített programkód:

Regiszter engedélyező:

```
module enable(
   input presetn,
   input pselx,
   input penable,
   input pwrite,
   input [31:30] paddr,
            input clock,
   output cmdwr,
   output reg [3:0] en
   );

assign cmdwr = pwrite;

always @ (posedge clock)

            if (presetn == 0) en = 4'b0;
            else
                        if (pselx == 1 && penable ==1)
                                    case (paddr [31:30])
                                                2'b00 : en = 4'b0001;
                                                2'b01 : en = 4'b0010;
                                                2'b10 : if (pwrite) en = 4'b0100;
                                                                        else       en = 4'b0;
                                                2'b11 : if (pwrite==1'b0) en = 4'b1000;
                                                                        else       en = 4'b0;
                                    endcase
                        else       en = 4'b0;


endmodule
```

Utasítás és állapotregiszer:

```
module cmd(
   input cmdwr,
   input [1:0] cmden,
   input presetn,
   input [7:0] data_in,
   input parerr,
          input clock,
   output reg [7:0] data_out,
   output reg rst,
   output reg clken,
   output reg wren,
   output reg rden,
   output reg [2:0] psgen,
          output wire xck,
          output reg ready
   );

reg [7:0] reg0 = 8'b0;
reg [7:0] reg1 = 8'b0;
reg [2:0] baud;
reg toready = 1'b0;
reg already = 1'b0;


always @ (posedge clock)
begin
          if (presetn == 0) begin                    //resetelés
                    reg0 <= 8'b00001101;
                    reg1 <= 8'b00000110;
            data_out = 8'b0;
                    rst <= 1'b0;
                    clken <= 1'b0;
                    wren <= 1'b0;
                    rden <= 1'b0;
                    psgen <= 4'b0100;
                    baud <= 3'b110;
                    ready = 1'b0;
                    already <= 1'b0;
                    end
          else      begin
                    case (cmden)                                   //regiszterblokkok írása, olvasása
                              2'b01 : begin
                                                  if (already == 1'b0) toready <= 1'b1;
                                                  if (cmdwr == 1) reg0 <= data_in;
                                                  else data_out = reg0;
                                                  end
                              2'b10 : begin
                                                  if (already == 1'b0) toready <= 1'b1;
                                                  if (cmdwr == 1) reg1 <= data_in;
                                                  else data_out = reg1;
                                                  end
                              default : begin
                                                            data_out = 8'b0;
                                                            already <= 1'b0;
```

```
                                                        end

                endcase                                                 //ready jel előállítása
                if (toready) begin
                        ready = 1'b1;
                        toready <= 1'b0;
                        already <= 1'b1;
                        end
                else begin
                        ready = 1'b0;
                        data_out = 8'b0;
                        end

                // regiszterek

                if (reg0 & 8'b00000001) clken <= 1;
                else clken <= 0;

                if (reg0 & 8'b00000100) wren <= 1;
                else wren <= 0;

                if (reg0 & 8'b00001000) rden <= 1;
                else rden <= 0;

                if (reg0 & 8'b00010000) psgen [0] <= 1;
                else psgen [0] <= 0;

                if (reg0 & 8'b00100000) psgen [1] <= 1;
                else psgen [1] <= 0;

                if (reg0 & 8'b01000000) psgen [2] <= 1;
                else psgen [2] <= 0;

                if (reg1 & 8'b00000001) baud [0] <= 1;
                else baud [0] <= 0;

                if (reg1 & 8'b00000010) baud [1] <= 1;
                else baud [1] <= 0;

                if (reg1 & 8'b00000100) baud [2] <= 1;
                else baud [2] <= 0;

                if (reg1 & 8'b00010000) rst <= 1'b1;
                else rst <= 1'b0;

                reg1[3] <= parerr;
                rst <= 1'b1;
        end

end

// Baudrate generátor

baudgen bdgen(
   .reset(rst),
   .baud_sel(baud),
   .clock(clock),
```

```
   .baud_clock(xck)
            );

Endmodule
```

Baud rate generátor:

```
module baudgen(
   input reset,
   input [2:0] baud_sel,
   input clock,
   output reg baud_clock
   );
//`define SYS_CLK 5'b10100  //20 -> system clock fr. in MHz

reg [12:0] div;
reg [12:0] count = 13'b0;

always @ (posedge clock)

begin

            if (reset == 0) begin
                        count <= 13'b0;
                        baud_clock <= 1'b0;
                        end
            else begin
                        case (baud_sel)
                                    3'b000 : div <= 13'd4167;
                                    3'b001 : div <= 13'd2084;
                                    3'b010 : div <= 13'd1042;
                                    3'b011 : div <= 13'd521;
                                    3'b100 : div <= 13'd348;
                                    3'b101 : div <= 13'd174;
                                    3'b110 : div <= 13'd10; //87;
                                    default : div <= 13'd87;
                        endcase

                        count <= count + 1;                                 // egyszerű leosztással állítjuk elő
                        if (count == (div -1)) begin                        // az megfelelő baud rate-t
                                    baud_clock <= baud_clock + 1'b1;
                                    count <= 0;
                        end
            end

end

endmodule
```

Bemeneti/ kimeneti buffer:

```
module en_buff(
        input clock,
        input en,
        input in,
        output reg out
   );

always @ (posedge clock)
        if (en == 1) out = in;
        else out = 1'bZ;

endmodule
```

Adómodul:

```
module ado(
   input [7:0] dinka,
        input clock,
   input en,
        input rst,
        input [2:0] psgen,
        input xck,
   output out,
        output ready
        );

wire [11:0] data;
wire [3:0] num;
wire shr_rst;

dregtr_psgen register(
        .din(dinka),
        .clock(clock),
   .en(en),
        .rst(rst),
        .psgen(psgen),
   .dout(data),
        .num(num),
        .shr_rst(shr_rst)
        );

shiftregps shiftregisterps(
        .clock(clock),
        .reset(rst),
        .shr_rst(shr_rst),
        .xck(xck),
   .data_in(data),
        .num(num),
   .out(out),
```

```
        .ready(ready)
  );


Endmodule



module dregtr_psgen(
  input [7:0] din,
        input clock,
  input en,
        input rst,
        input [2:0] psgen,
  output reg [11:0] dout,
        output reg [3:0] num,
        output reg shr_rst
  );

reg [11:0] inner_reg = 11'b11;
reg parity = 1'b0;
reg already = 1'b1;
reg [2:0] psgen_or = 3'b0;
reg [7:0] din_temp = 8'b0;

always @ (posedge en)
begin
        if (rst) begin                          //Üzenet előkészítése engedélyezésnél, paritás
                                                kiszámítása
                        inner_reg[11] <= 1'b0;
                        din_temp <= din;
                        parity <= inner_reg[10] + inner_reg[9] + inner_reg[8] + inner_reg[7] + inner_reg[6] +
inner_reg[5] + inner_reg[4] + inner_reg[3];
                        psgen_or = psgen;
                end
        else begin
                        din_temp <= 8'b0;
                        parity <= 1'b0;
                        psgen_or = 3'b0;
                end
end

always @ (posedge clock)
begin
        if (rst == 1'b0)
                begin
                        dout <= 11'b11;
                        num <= 4'b0;
                        shr_rst <= 1'b0;
                        already = 1'b1;
                end
        else begin
                case (psgen_or[1:0])                    //Paritástól és stopbitek számától
függően üzenet elkészítése
                        2'b01 :                 //és üzenet hosszának beállítása -> ez //kerül
                                                átadásra a SHR-nek
                                begin
```

```
                                        inner_reg[2] <= parity;
                                        if (psgen_or[2]) num <= 4'b1100;
                                        else num <= 4'b1011;
                                end
                        2'b10 :
                                begin
                                        inner_reg[2] <= parity + 1;
                                        if (psgen_or[2]) num <= 4'b1100;
                                        else num <= 4'b1011;
                                end
                        default :
                                begin
                                        inner_reg[2] <= 1'b1;
                                        if (psgen_or[2]) num <= 4'b1011;
                                        else num <= 4'b1010;
                                end
                endcase
                inner_reg[10:3] <= din_temp;
                dout <= inner_reg;
                if (en) begin                                   //Shiftregiszter resetelése
                                if (already) begin
                                        shr_rst <= 1'b1;
                                        already = 1'b0;
                                        end
                                end
                else already = 1'b1;
                if (shr_rst) shr_rst <=1'b0;
        end
end


endmodule


module shiftregps(
   input clock,
           input reset,
           input shr_rst,
           input xck,
   input [11:0] data_in,
           input [3:0] num,
   output reg out,
           output reg ready
   );

reg [3:0] count = 4'b0;
reg already = 1'b0;
reg shr_rst_start = 1'b0;
reg transmit =1'b0;
reg shr_rst_reset = 1'b0;

always @ (posedge clock)                        //Ready jel elkészítése
        if (reset) begin
                        if (count == num && already == 1'b0 && num != 0) begin
                                ready <= 1'b1;
                                already <= 1'b1;
                                end
```

```
                        if (ready) ready <= 1'b0;
                        if (shr_rst) begin
                                already <= 1'b0;
                                shr_rst_start = 1'b1;
                                end
                        if (shr_rst_reset) shr_rst_start = 1'b0;
                end
        else ready <= 1'b0;

always @ (posedge xck)                  //Shiftregiszter -> annyi adatot ad ki, amennyit
begin                                           //neki paraméteresen átadunk
        if (shr_rst_start) begin
                count <= 4'b0;
                shr_rst_reset = 1'b1;
                transmit <= 1'b1;
        end
        else begin
                        shr_rst_reset = 1'b0;
                        if (reset) begin
                                        if ((count < num) && transmit) begin
                                        count <= count + 1'b1;
                                        out <= data_in[4'd11 - count];
                                        end
                                        else begin
                                                out <=1'b1;
                                                transmit <=1'b0;
                                        end
                                end
                        else begin
                                count <= 4'b0;
                                out <= 1'b1;
                                transmit <=1'b0;
                                end
                        end
end

endmodule
```

Vevőmodul:

```
module vevo(
   input vren,
            input rden,
            input clock,
            input rst,
            input [2:0] psgen,
            input xck,
   input rxd,
   output ready,
   output [7:0] par_out,
   output IT,
            output parerr
   );

wire [7:0] data;
wire ser_in;

datareg_rec datareg_rc(
   .vren(vren),
   .data_in(data),
            .clock(clock),
            .rst(rst),
   .data_out(par_out),
   .ready(ready)
            );

shiftregsp shrsp(
            .psgen(psgen),
            .xck(xck),
            .clock(clock),
   .s_in(ser_in),
   .rst(rst),
   .p_out(data),
   .it(IT),
            .parerr(parerr)
   );

en_buff ebuffrxd(
            .clock(clock),
            .en(rden),
            .in(rxd),
            .out(ser_in)
   );

Endmodule


module datareg_rec(
   input vren,
   input [7:0] data_in,
            input clock,
            input rst,
   output reg [7:0] data_out,
   output reg ready
```

```
   );

reg already = 1'b0;
always @ (posedge clock)

        begin
                if (rst == 0) begin
                        data_out = 8'b0;
                        ready = 1'b0;
                        end
                else if (vren) begin                            //Egyszerű adattároló és ready jel elkészítő
                                if (already==1'b0)
                                        begin
                                                data_out = data_in;
                                                ready = 1'b1;
                                                already = 1'b1;
                                        end
                                else
                                        begin
                                                data_out = 8'b0;
                                                ready = 1'b0;
                                        end
                                end
                else already = 1'b0;
                end

endmodule

module shiftregsp(
   input [2:0] psgen,
           input xck,
           input clock,
   input s_in,
   input rst,
   output reg [7:0] p_out,
   output reg it,
           output reg parerr
   );

reg [3:0] count = 4'b0;
reg [7:0] temp = 8'b0;
reg prev = 1'b0;
reg in_use = 1'b0;
reg par = 1'b0;
reg par_ch = 1'b0;
reg [3:0] bitmax = 4'b1010;

always @ (negedge xck)
begin
        if (rst)
                begin
                        prev <= s_in;

                        //1->0 átmenet detektálása, azaz üzenetkezdés

                        if ((in_use == 1'b0) & (prev == 1'b1) & (s_in == 1'b0)) begin
                                count <= 4'b0;
```

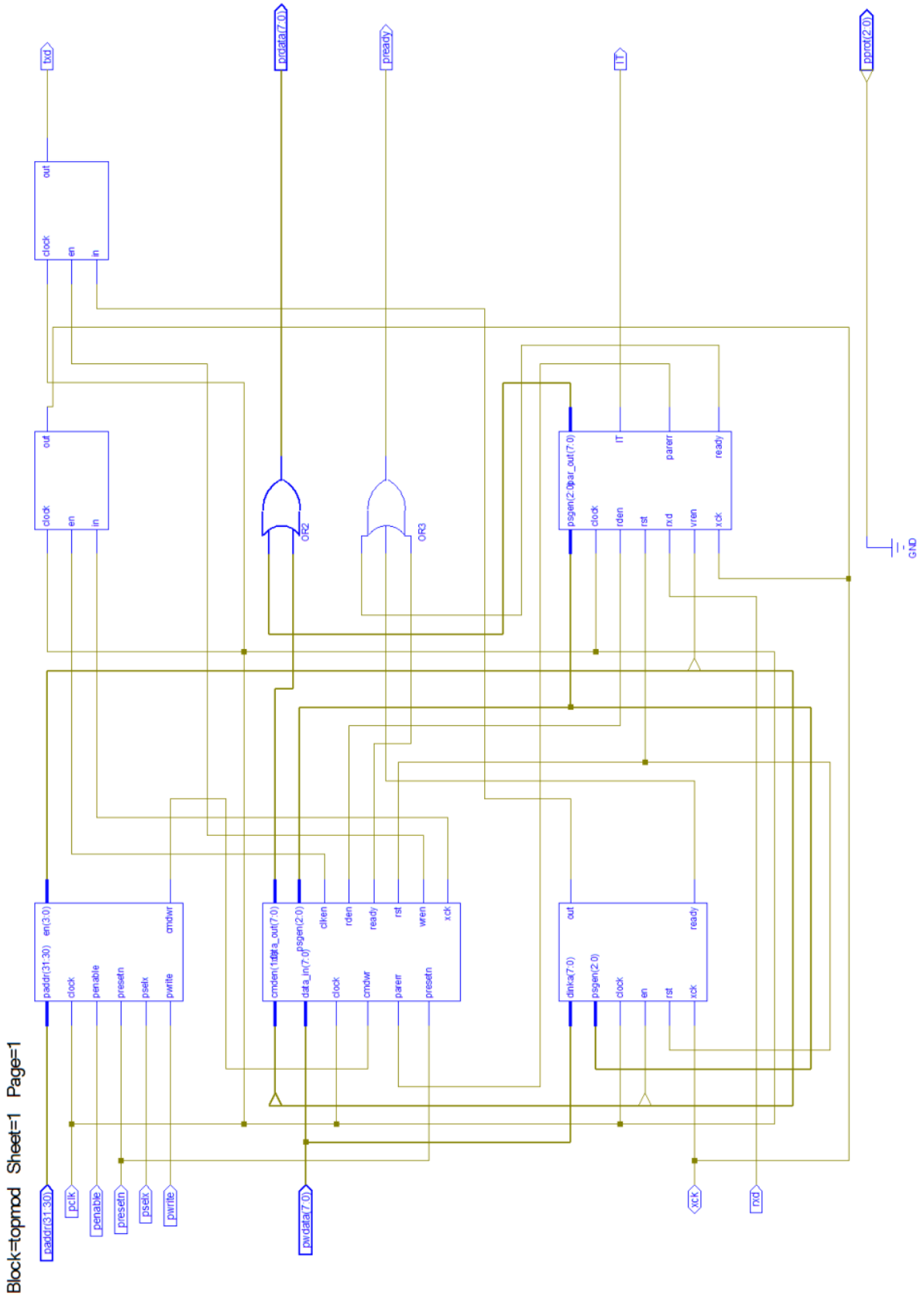
```
                        par <= 1'b0;
                        par_ch <=1'b0;
                        in_use <= 1'b1;
                        temp <= 8'b0;
                        end
                else if (in_use) begin
                        if (count < 4'b1000) begin
                                temp <= {temp, s_in};                   // Adat összeállítása
                                par_ch <= par_ch + s_in; end
                        else if (count == 4'b1000)
                                par <= s_in;
                        else if (count == bitmax) begin
                                in_use <= 1'b0;
                                it <= 1'b1;
                                p_out <= temp;
                                case (psgen[1:0])                       // Paritás ellenőrzése
                                        2'b01: if ((par_ch ^ par) == 1'b0) parerr <= 1'b1; // ptl
                                                        else parerr <= 1'b0;
                                        2'b10: if (par_ch ^ par) parerr <= 1'b1; // paros
                                                        else parerr <= 1'b0;
                                        default: parerr <= 1'b0;
                                endcase
                        end
                end

                if (in_use) count <= count + 1'b1;
                else it <= 1'b0;
        end
    else
        begin                                                   //Resetelés
                count <= 4'b0;
                temp <= 8'b0;
                prev <= 1'b0;
                in_use <= 1'b0;
                par <= 1'b0;
                par_ch <= 1'b0;
                it <=0;
                parerr <=0;
                p_out <=0;
        end
end

always @ (posedge clock)        //Stop és paritásbitnek megfelelő maximális bitszám
    case (psgen)
        3'b000: bitmax[3:0] <= 4'b1010;
        3'b001: bitmax[3:0] <= 4'b1001;
        3'b010: bitmax[3:0] <= 4'b1001;
        3'b100: bitmax[3:0] <= 4'b1001;
        3'b101: bitmax[3:0] <= 4'b1010;
        3'b110: bitmax[3:0] <= 4'b1010;
        default : bitmax[3:0] <= 4'b1001;
    endcase

endmodule
```

Topmodul:

```
module topmod(
   input pclk,
   input presetn,
   input pselx,
   input penable,
   input pwrite,
   input [31:30] paddr,
   input [7:0] pwdata,
            input rxd,
   output [7:0] prdata,
   output pready,
   output [2:0] pprot,
   output txd,
            output IT,
   inout xck
   );

assign pprot = 3'b0;

wire cmdwr;
wire [3:0] all_en;
wire [1:0] cmden;
wire aren;
wire vren;
wire parerr;
wire rstn;
wire clken;
wire wren;
wire rden;
wire [2:0] psgen;
wire cmd_ready;
wire aready;
wire vready;
wire [7:0] cmd_data_out;
wire xck_out;
wire [7:0] rec_data_out;
wire txwire;
wire rxwire;

enable reg_enable(
   .presetn(presetn),
   .pselx(pselx),
            .penable(penable),
   .pwrite(pwrite),
   .paddr(paddr),
            .clock(pclk),
   .cmdwr(cmdwr),
            .en(all_en)
   );

assign cmden = all_en [1:0];
assign aren = all_en [2];
assign vren = all_en [3];
cmd cmdst(
```

```
   .cmdwr(cmdwr),
   .cmden(cmden),
   .presetn(presetn),
   .data_in(pwdata),
   .parerr(parerr),
            .clock(pclk),
   .data_out(cmd_data_out),
   .rst(rstn),
   .clken(clken),
   .wren(wren),
   .rden(rden),
   .psgen(psgen),
            .xck(xck_out),
            .ready(cmd_ready)
   );

en_buff ebuffpclk(
            .clock(pclk),
            .en(clken),
            .in(xck_out),
            .out(xck)
   );

ado adomodul(
            .dinka(pwdata),
            .clock(pclk),
   .en(aren),
            .rst(rstn),
            .psgen(psgen),
            .xck(xck),
   .out(txwire),
            .ready(aready)
            );

en_buff ebufftxd(
            .clock(pclk),
            .en(wren),
            .in(txwire),
            .out(txd)
   );

vevo vevomodul(
   .vren(vren),
            .rden(rden),
            .clock(pclk),
            .rst(rstn),
            .psgen(psgen),
            .xck(xck),
   .rxd(rxd),
   .ready(vready),
   .par_out(rec_data_out),
   .IT(IT),
            .parerr(parerr)
   );

assign prdata = cmd_data_out | rec_data_out;
assign pready = (cmd_ready | aready | vready);
```

Ellenőrzésképp a program által generált RTL kapcsolási rajz:



Tesztelés:

Egyenként teszteltük minden egyes modult, készítettünk többféle idődiagramot, valamint elvégeztünk számos bemeneti kombináció rekonstruálását. Ez azért volt hasznos, mivel így könnyebb volt egyesével, egyszerűbb szinten debug-olni, így mikor a topmodult teszteltük, jó eséllyel gondoltuk, hogy a modulok összerakva is működni fognak.

Teszteléshez verilogban task-okat írtunk: egyet a busz írásának, egyet olvasásának szimulálására, egyet pedig az USRT receive portjához. Erre illeszettünk testbench-et, mellyel a szimuláció végbement, közben folyamatosan figyeltük a jelek, buszok változását. A taskok a specifikációkhoz igazodnak, mind az AMBA, mind az USRT részről.

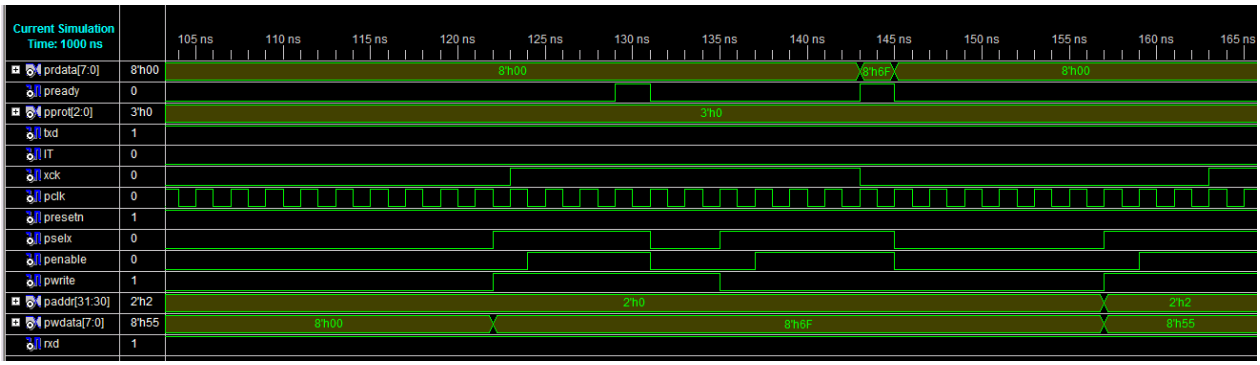
Testbench:

```
module topmodteszt2;

	// Inputs
	reg pclk;
	reg presetn;
	reg pselx;
	reg penable;
	reg pwrite;
	reg [31:30] paddr;
	reg [7:0] pwdata;
	reg rxd;

	// Outputs
	wire [7:0] prdata;
	wire pready;
	wire [2:0] pprot;
	wire txd;
	wire IT;

	// Bidirs
	wire xck;

	// Instantiate the Unit Under Test (UUT)
	topmod uut (
		.pclk(pclk),
		.presetn(presetn),
		.pselx(pselx),
		.penable(penable),
		.pwrite(pwrite),
		.paddr(paddr),
		.pwdata(pwdata),
		.rxd(rxd),
		.prdata(prdata),
		.pready(pready),
		.pprot(pprot),
		.txd(txd),
		.IT(IT),
		.xck(xck)
```

```
	);


task bus_read(input [31:30] paddr1);			//Busz olvasás taskja
begin
#2 paddr <= paddr1;
pwrite <= 0;
pselx <= 1;

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

#2 pselx <= 0;
penable <=0;
#10;
end
endtask

task bus_write(input [31:30] paddr1, input [7:0] pwdata1);		//Busz írás taskja
begin

#2 paddr <= paddr1;
pwrite <= 1;
pwdata <= pwdata1;
pselx <= 1;

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

#2 penable <= 0;
pselx <= 0;
#2;
end
endtask

task usrt_write(input [8:0] usrt_d);				//USRT írás taskja
begin
	@ (negedge xck) begin wait(xck);   rxd = 1'b0;          end
	@ (negedge xck) begin wait(xck);   rxd = usrt_d[8]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[7]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[6]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[5]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[4]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[3]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[2]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[1]; end
	@ (negedge xck) begin wait(xck);           rxd = usrt_d[0]; end
	@ (negedge xck) begin wait(xck);           rxd = 1'b1; end
end
endtask

	initial begin
		// Initialize Inputs
		pclk = 0;
		presetn = 0;
		pselx = 0;
		penable = 0;
```

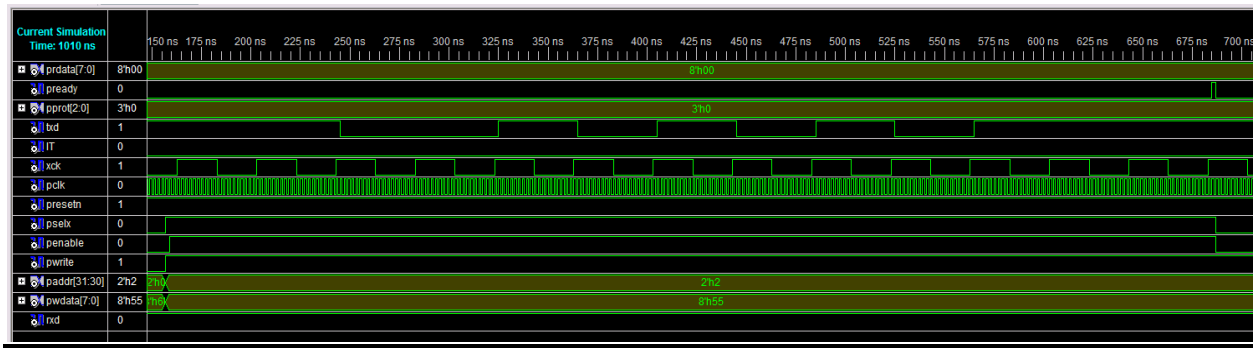
```
                pwrite = 0;
                paddr = 0;
                pwdata = 0;
                rxd = 1;
                #20
                presetn = 1;

                #100;                                   //Egyszerű példatesztprogram
                bus_write(0, 8'h6F);
                bus_read(0);
                bus_write(2, 8'h55);
                usrt_write(9'b010101011);
                wait(IT) bus_read(3);
                bus_read(1);
        end
    always #1 pclk = ~pclk;

endmodule
```

Busz írás és adat visszaolvasása az állapotregiszterből:



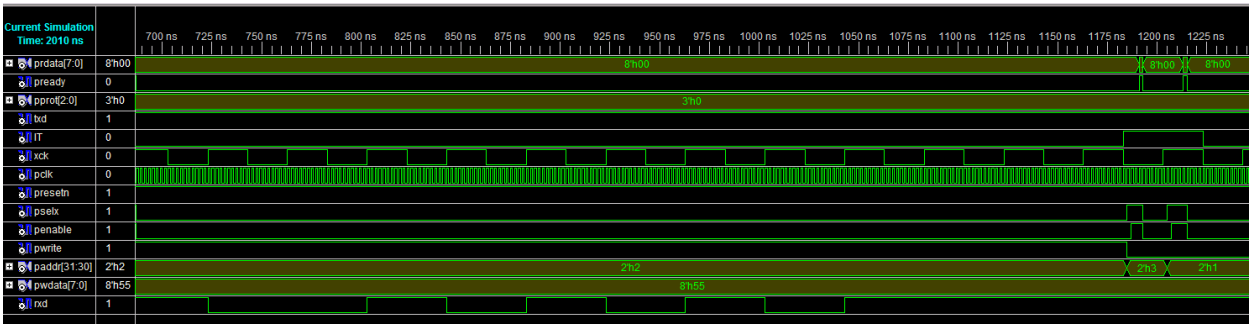
Ezen az ábrán látható, hogy írásnál a megfelelő jelek kiadása után érkezik egy ready, mely az írás sikerességét mutatja. A beírt adat 8'h6F volt, melyet utána egy olvasási ciklusban kiolvasunk ugyanarról a címről.

Adat kiküldése USRT-re:

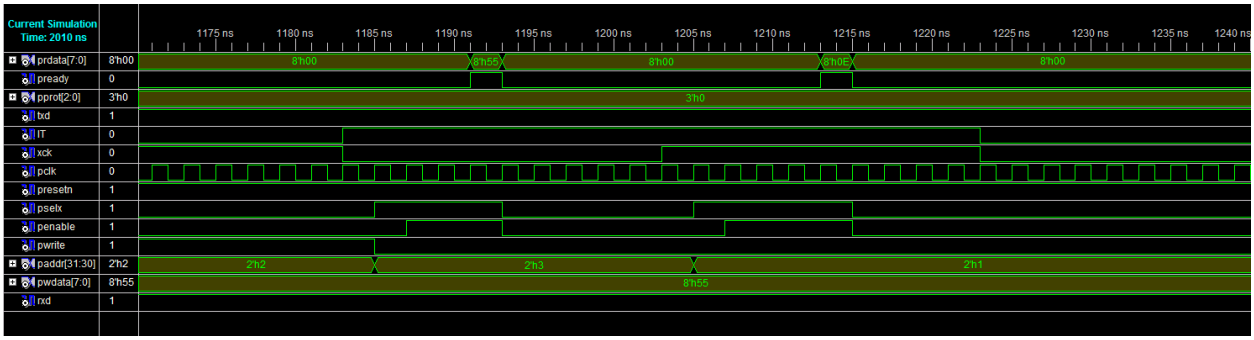


Itt a 8'h55-ös üzenetet küldjük ki, 0 paritásbittel. Ha kiment az adat, megjelenik a ready jel és befejeződik a ciklus.

Beérkező adat USRT-re:

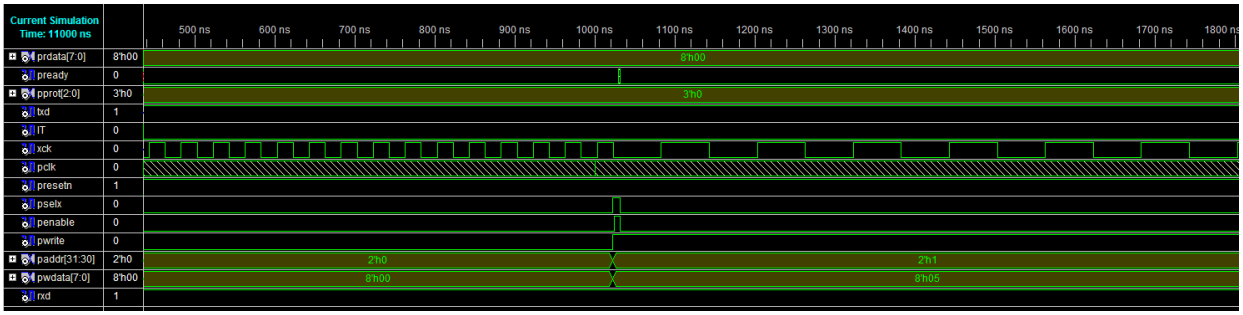


Beküldünk adatot, direkt rossz paritással. Ha megérkezik az interrupt, kiolvassuk az adatot a regiszterből, majd ready-vel jelzünk, hogy az adat kiolvasva. A bal oldali rész kinagyítva a következő ábrán látható. Ez is jól szemlélteti, hogy mennyivel gyorsabb a busz sebessége, mint a leggyorsabb USRT.



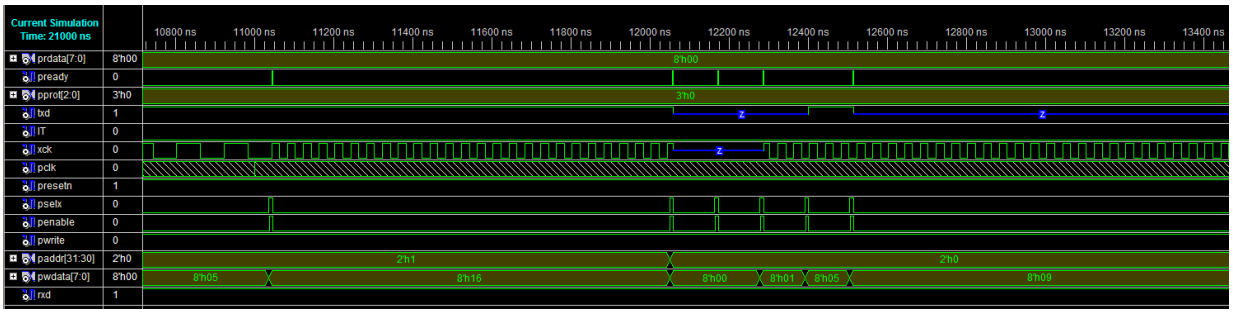
Az első kiolvasás a jel értéke (55 hexa), a második pedig az állapotregiszter második kódszava, amely a paritáshiba értékével rendelkezik.

USRT órajelének átállítása:



Az állapotregiszter megfelelő kódszóval átállítja a baud rate-et.

Kimeneti bufferek tiltása/engedélyezése:



Itt ismét az állapotregiszter átírásával tudjuk a kimeneti buffereket változtatni. Először mindegyiket letiltjuk, majd engedélyezzük először az órajelet, utána a TXD-t. (Az ábrán nem látszik, de a befelé irányuló RXD-t is megváltoztattuk.)