Mérési jegyzőkönyv a

A fejlesztő eszköz bemutatása

című laboratóriumi gyakorlat kijelölt részéről

A mérés helyszíne: BME I. ép. E504 laboratórium

A mérés időpontja: 2011.11.15.

A mérést végezték: Szepes Nóra KDE004
Demény Fruzsina Gyöngyi KDE004

A mérést vezető oktató neve: Gruber Gábor

A dokumentálásra kijelölt mérési feladatok

Feladat:

Soros adó egység (UART)

Realizálás: FPGA mérőpanelen

A soros adó egység az SW0..6 kapcsolókon binárisan beállított karakterkódot start és stop bittel keretezi, az előírásoktól függően paritásbittel is kiegészíti, és kiadja a panel soros portjának TXD vonalán, az RS232/V24 szabvány előírásainak megfelelően (lásd 3. mérés dokumentáció). (A Start bit 0 értékű, a Stop bit 1 értékű, a megfelelő jelszinteket az FPGA panelbe épített szintkonverter áramkör biztosítja.)

Az egység a BTN1 gomb megnyomásának hatására egyszer kiadja a kódot.

Az egység a BTN2 gomb megnyomásának hatására is kiadja a kódot, de a karaktert ismételten a vonalra adja mindaddig, amíg a gomb nyomva van. Az egyes karakterek adása közti szünet (Stop állapot) kb. 2 karakter hosszúságú.

Az adatátvitel paraméterei:

A karakterek 7 bitesek.

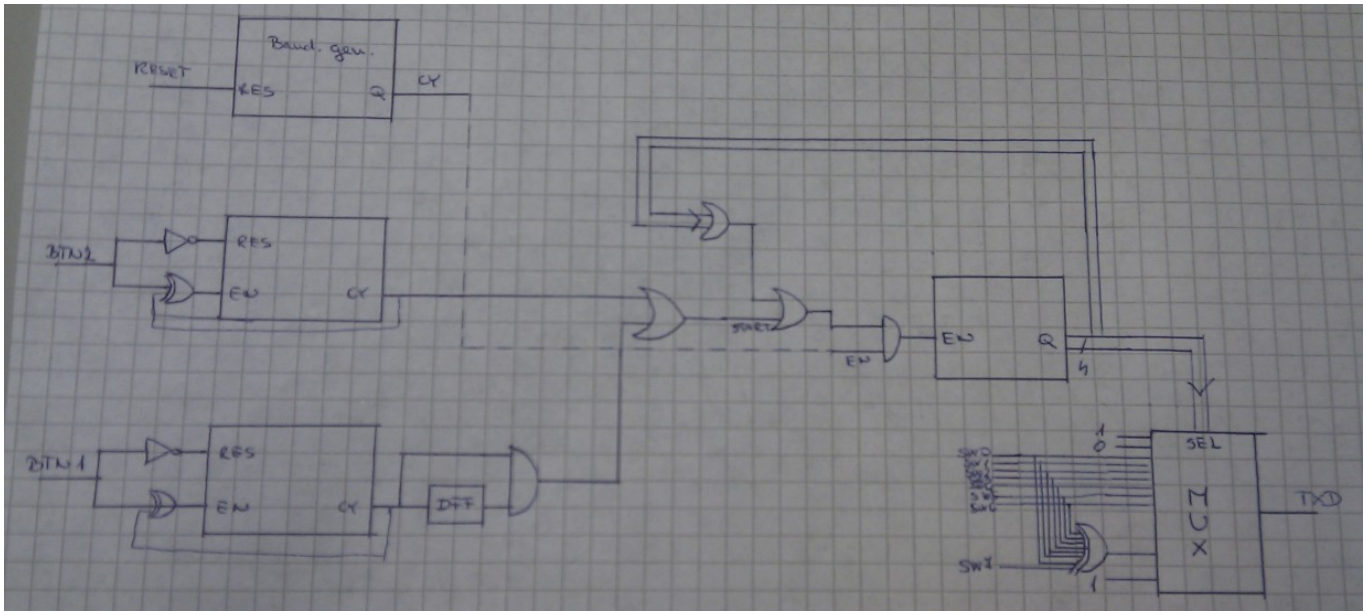
A stop bitek száma 2.

A jelváltási sebesség 4800 Baud.

Paritásbit van, típusa (páros vagy páratlan) az SW7 kapcsolóval választható.

Kötelező szinkron rendszer tervezése, azaz az összes FF órajel bemenetére az 50 MHz-es rendszerórajelet kell kötni!

A feladathoz készített **<u>blokkvázlat</u>**:



A soros adó egység elkészítésekor első feladat a BTN1 és a BTN2 gombok megnyomásának észleléséhez szükséges pergésmentesítés, ezeket számlálók illetve a BTN1 esetében még D Flipfloppal oldottuk meg a késleltetést.

<u>BTN1 pergésmentesítése</u>:

```
module btn1_pergesmentesito(
   input btn,
        input clk,
        input en,
   output btn1_en
   );
        wire rst = ~btn;
        wire carry;
        wire engedelyezes;
        reg[15:0] Q;

        //assign carry=(Q==1000);
        assign carry=(Q==4);
        assign engedelyezes=((btn^carry)&en);

        //belso szamlalo
        always@(posedge clk)
                begin
                        if(rst) Q<=16'b0;
                        else            if(engedelyezes)
Q<=Q+1;
                end

                reg dff;

                //DFF
                always@(posedge clk)
                begin
                        if(en)   dff<=carry;
                end

                assign btn1_en=(dff&~carry);
endmodule
```

BTN2 pergésmentesítése:

```
module btn2_pergesmentesito(
  input btn,
        input clk,
        input en,
   output btn2_en
   );

        wire rst = ~btn;
        wire engedelyezes;
        reg[15:0] Q;

//      assign btn2_en=(Q==1000);
        assign btn2_en=(Q==4);
                                                        assign
engedelyezes=((btn^btn2_en)&en);

        //belso szamlalo
        always@(posedge clk)
        begin
                if(rst) Q<=16'b0;
                else            if(engedelyezes)
Q<=Q+1;
        end

endmodule
```

A gombok pergésmentesítése után a kimenetet belevezetjük egy **bitcounter**-be, amely egy olyan számláló, mely 0-10-ig számol, ezzel állítjuk be a multiplexer SEL jelét, hogy éppen hányadik bitet adja ki.

```
module bitcounter(
   input start,
   input en,
        input clk,
        input rst,
   output [3:0] out
   );
wire engedelyezes;
wire nemnulla;
reg[3:0] Q;

assign nemnulla =(Q>4'b0);
assign      engedelyezes      =      (en&(start|
nemnulla));

always@(posedge clk)
begin
        if(rst|(en&Q==10)) Q<=4'b0;
        else if(engedelyezes) Q<=Q+1;
end

assign out = Q;

endmodule
```

A 11 bitet be kell állítani, ebből az 0. és a 10. bit STOP bitek, amelyek értéke mindig 1, míg az 1. bit a START bit, melynek értéke mindig 0. A 11 bitből a 2., 3., 4., 5., 6., 7. és 8. bitekre kötjük az első 7 switch kapcsolót, mellyel a kiadandó karaktert állítjuk be, a 9. bit pedig a paritásbit, melyet a nyolcadik kapcsolóval választunk úgy, hogy XOR kapuba kötjük az első hét kapcsoló állását, és hogy a paritásbit beállítható legyen, pluszba bekötjük a kapuba a 8. switch kapcsolót is.

```
module bitgenerator(
   input [7:0] sw,
   output [10:0] out
   );
        assign out[0]=1; // STOP bit - elsõ
        assign out[1]=0; //START bit
        assign out[2]=sw[0]; //elsõ kapcsoló állása
        assign out[3]=sw[1]; //második kapcsoló állása
        assign out[4]=sw[2]; //harmadik kapcsoló állása
        assign out[5]=sw[3]; //negyedik kapcsoló állása
        assign out[6]=sw[4]; //ötödik kapcsoló állása
        assign out[7]=sw[5]; //hatodik kapcsoló állása
        assign out[8]=sw[6]; //hetedik kapcsoló állása
           assign    out[9]=sw[0]^sw[1]^sw[2]^sw[3]^sw[4]^sw[5]^sw[6]^sw[7];    //    nyolcadik
kapcsolóval (SW7) tudjuk választani a paritást
   //a paritást bithez pedig összeszámoljuk az 1-eseket, és ha páratlan számú 1es van, akkor lesz az
1.
        //Egyszerûsítésként az összes kapcsoló állását összeXORoltam.
        assign out[10]=1; // STOP bit - második
endmodule
```

A **Baudrate generátor** az egész rendszer engedélyező jele. Az 10416 pedig úgy jött ki, hogy az 50 000 000-t elosztottuk az előírt 4800-al, vagyis ha a számláló a 10417. órajelben van, akkor engedélyezi a rendszert, és közben nullázza a számlálót.

```
module baud_gen(
   input clk,
   input rst,
   output cy
   );

reg[15:0] Q;
//assign cy =(Q==16'd10416);
assign cy=(Q==16'd3); //szimlulációhoz, hogy
lássuk mi történik

always@(posedge clk)
begin
        if(rst|cy) Q<=16'd0;
        else Q<=Q+1;
end

endmodule
```

És végül a top level-ben összeállítjuk a programot, amelyben a multiplexert is megírtjuk.

```
module top_level(
        input clk,
        input rst,
        input btn1,
        input btn2,
        input[7:0] sw,
        output kimenet,
        //output[10:0] bitgen_dbg,
        //output[3:0] dbgki,
        //output startki,
        //output btn1enki,
        //output btn2enki
        //teszteléshez kellettek a fentiek
   );

        wire en;
        baud_gen baud_generator(.clk(clk), .rst(rst), .cy(en)); //létrehozzuk az engedélyezõ jelet
//.bennevan(belerakom)

        wire[10:0] swk_allasa;          //sw-k állásának "leolvasása"
        bitgenerator bitgen(.sw(sw), .out(swk_allasa));

        wire start;
        wire btn1_en;
        wire btn2_en;

        btn1_pergesmentesito btn1_p(.btn(btn1), .clk(clk), .en(en), .btn1_en(btn1_en));
        btn2_pergesmentesito btn2_p(.btn(btn2), .clk(clk), .en(en), .btn2_en(btn2_en));

        assign start=(btn1_en||btn2_en); //létrehozzuk a start jelet
        //assign startki=start;
        //assign btn1enki=btn1_en;
        //assign btn2enki=btn2_en;

        wire[3:0] mpx_select;
        bitcounter bitcntr(.start(start), .en(en), .clk(clk), .rst(rst), .out(mpx_select));

        //multiplexer
        assign kimenet=swk_allasa[mpx_select];
        //assign bitgen_dbg=swk_allasa;
        //assign dbgki=mpx_select;


endmodule
```

Az előzőleg leírt modulokon kívül, el kellett végezni a lábkiosztás beállításait. Beállítottuk az órajelet, és a reset gombot, továbbá a BTN1, BTN2 gombokat, mellyel a karakterkiadást irányítjuk. Majd a switcheket kötöttük be, amelyen az adatbiteket és a paritásbitet állíthatjuk be, végül pedig az UART-vevőre rákötöttük a kimenetünket.

```
NET "clk" LOC = "T9"; # órajel, 50MHz
NET "rst" LOC = "M13"; # reset gomb - btn0

//gombok
NET "btn1" LOC = "M14";
NET "btn2" LOC = "L13";

//switchek
NET "sw[0]" LOC = "F12"; # adatbit, lsb
NET "sw[1]" LOC = "G12"; # adatbit
NET "sw[2]" LOC = "H14"; # adatbit
NET "sw[3]" LOC = "H13"; # adatbit
NET "sw[4]" LOC = "J14"; # adatbit
NET "sw[5]" LOC = "J13"; # adatbit
NET "sw[6]" LOC = "K14"; # adatbit, msb
NET "sw[7]" LOC = "K13"; # paritás kapcsoló

//uart vevő
NET "kimenet" LOC = "R13"; //  PC_TXD
```

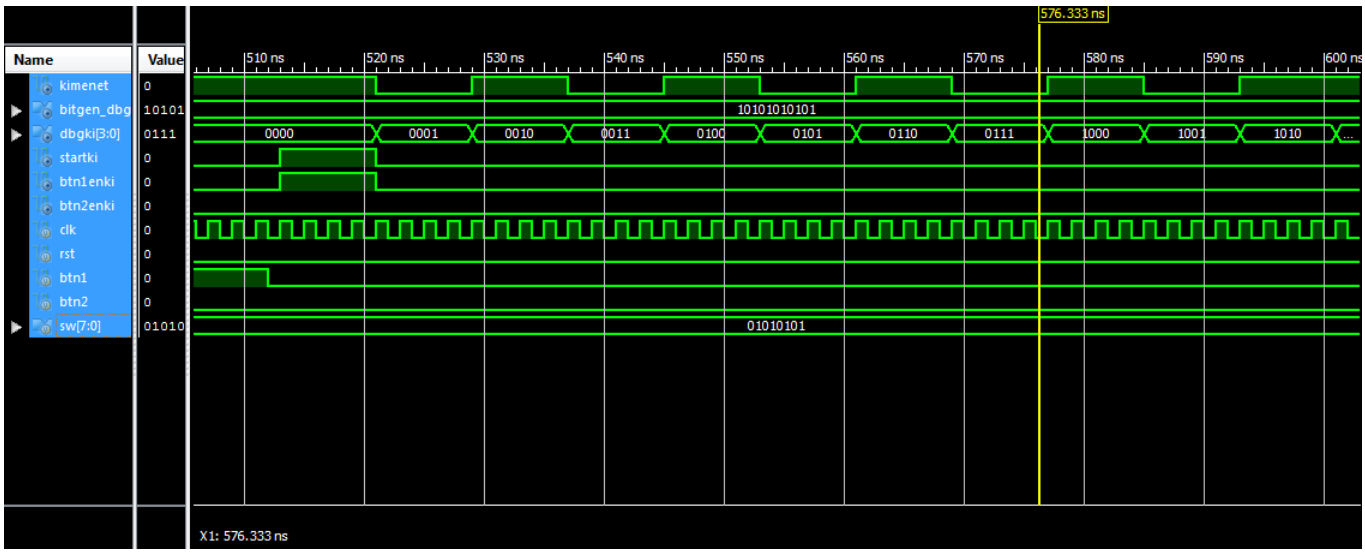
Ezen kívül még a logikai analizátor lábkiosztását is megcsináltuk.

```
NET "LANAL<33>"   LOC = "C10"; #CLK2
NET "LANAL<32>"   LOC = "C15"; #CLK1

NET "LANAL<31>"   LOC = "T3";  #D31
NET "LANAL<30>"   LOC = "E10"; #D30
NET "LANAL<29>"   LOC = "N11"; #D29
NET "LANAL<28>"   LOC = "C11"; #D28
NET "LANAL<27>"   LOC = "P10"; #D27
NET "LANAL<26>"   LOC = "D11"; #D26
NET "LANAL<25>"   LOC = "R10"; #D25
NET "LANAL<24>"   LOC = "C12"; #D24
NET "LANAL<23>"   LOC = "T7";  #D23
NET "LANAL<22>"   LOC = "D12"; #D22
NET "LANAL<21>"   LOC = "R7";  #D21
NET "LANAL<20>"   LOC = "E11"; #D20
NET "LANAL<19>"   LOC = "N6";  #D19
NET "LANAL<18>"   LOC = "B16"; #D18
NET "LANAL<17>"   LOC = "M6";  #D17
NET "LANAL<16>"   LOC = "R3"; #D16
NET "LANAL<15>"   LOC = "C16"; #D15
NET "LANAL<14>"   LOC = "D15"; #D14
NET "LANAL<13>"   LOC = "D16"; #D13
NET "LANAL<12>"   LOC = "E15"; #D12
NET "LANAL<11>"   LOC = "E16"; #D11
NET "LANAL<10>"   LOC = "F15"; #D10
NET "LANAL<9>"    LOC = "G15"; #D9
NET "LANAL<8>"    LOC = "G16"; #D8
NET "LANAL<7>"    LOC = "H15"; #D7
NET "LANAL<6>"    LOC = "H16"; #D6
NET "LANAL<5>"    LOC = "J16"; #D5
NET "LANAL<4>"    LOC = "K16"; #D4
NET "LANAL<3>"    LOC = "K15"; #D3
NET "LANAL<2>"    LOC = "L15"; #D2
NET "LANAL<1>"    LOC = "N9";  #D1
NET "LANAL<0>"    LOC = "M11"; #D0
```

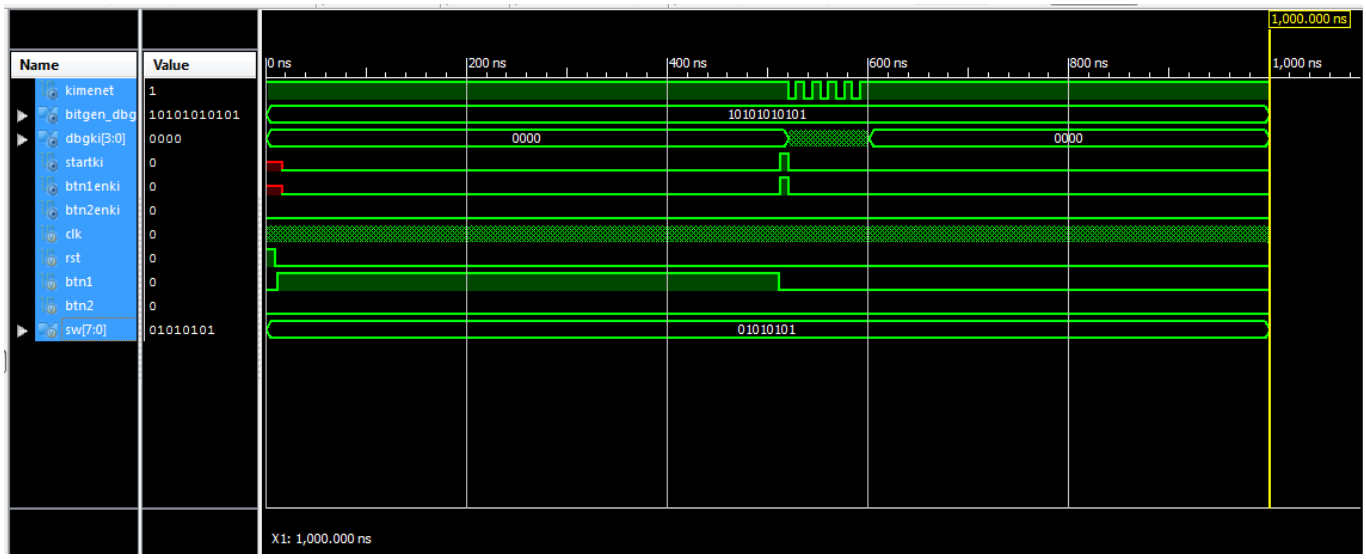
Ezután elvégeztünk néhány szimulációt a program tesztelésére.

Először a **BTN1** gombot teszteltük egy megnyomásra, és nyomva tartásra is.

Elsőnek teszteltük normál működésre, amikor a gombot egyszer, rövid időre nyomjuk meg. Működése megfelelő, a gomb megnyomására kiadja a startjelet és az engedélyezés is megtörténik, ezután egyszer kikerül a switch-eken beállított karakter a kimenetre.



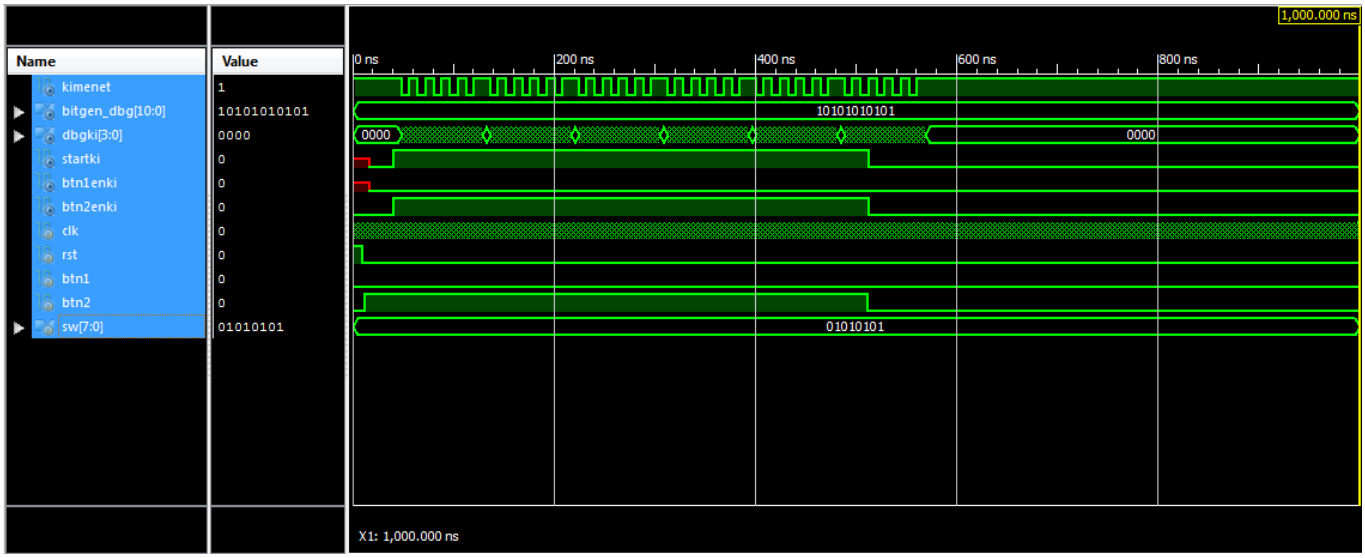
Ezután hosszabb idei nyomva tartásra teszteltük, látható, hogy a BTN1 gomb még így is csak egyszer adja ki a karaktert. Tehát a működésben nincs hiba, amíg el nem engedjük és nyomjuk meg újra, addig nem küldi ki a karaktert újra.



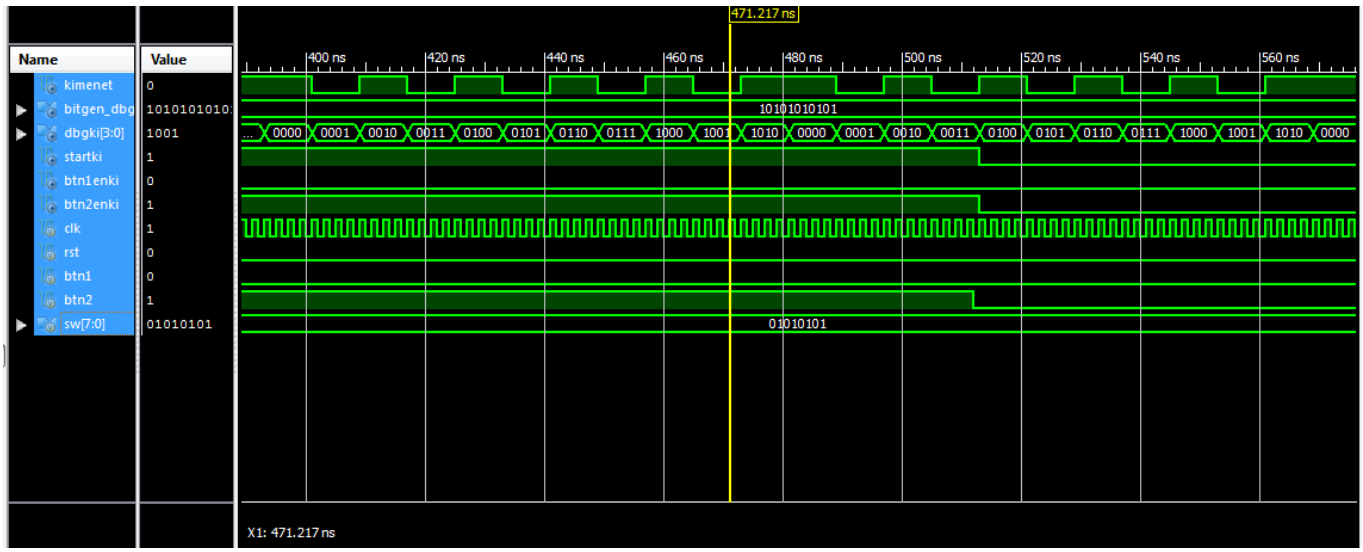
```
module proba1;

	// Inputs
	reg clk;
	reg rst;
	reg btn1;
	reg btn2;
	reg [7:0] sw;

	// Outputs
	wire kimenet;
	wire[10:0] bitgen_dbg;
	wire[3:0] dbgki;
	wire startki;
	wire btn1enki;
	wire btn2enki;

	// Instantiate the Unit Under Test (UUT)
	top_level uut (
		.clk(clk),
		.rst(rst),
		.btn1(btn1),
		.btn2(btn2),
		.sw(sw),
		.kimenet(kimenet),
		.bitgen_dbg(bitgen_dbg),
		.dbgki(dbgki),
		.startki(startki),
		.btn1enki(btn1enki),
		.btn2enki(btn2enki)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		rst = 1;
		btn1 = 0;
		btn2 = 0;
		sw = 8'b01010101;

		// Wait 100 ns for global reset to
finish
		#10;
			rst=0;
		#2;
			btn1=1;
		#500;
			btn1=0;


		// Add stimulus here

	end
    always #1 clk=~clk;
endmodule
```

Másodszorra a **BTN2** gombra indítottuk el a szimulációt.

Következőnek teszteltük a BTN2 gomb működését. A BTN2 gomb nyomva tartása esetén folyamatosan ki kell adnia a karaktert, mindaddig, amíg a gomb nyomva van. Látszik, hogy a karakter bitjeit folyamatosan kiadja, 2 stopbitnyi szünetet tartva közöttük. Amikor a gombot elengedjük, a karakter bitjeinek kiadását nem szakítja meg, még befejezi a karakter további bitjeinek a kimenetre küldését, majd ezután vár a következő megnyomásra.



A BTN2 gombot hosszabb idei lenyomásra is leteszteltük, melyen hasonló eredményt kaptunk, mint az első tesztelésnél.



```
module proba1;

	// Inputs
	reg clk;
	reg rst;
	reg btn1;
	reg btn2;
	reg [7:0] sw;

	// Outputs
	wire kimenet;
	wire[10:0] bitgen_dbg;
	wire[3:0] dbgki;
	wire startki;
	wire btn1enki;
	wire btn2enki;

	// Instantiate the Unit Under Test (UUT)
	top_level uut (
		.clk(clk),
		.rst(rst),
		.btn1(btn1),
		.btn2(btn2),
		.sw(sw),
		.kimenet(kimenet),
		.bitgen_dbg(bitgen_dbg),
		.dbgki(dbgki),
		.startki(startki),
		.btn1enki(btn1enki),
		.btn2enki(btn2enki)
	);

	initial begin
		// Initialize Inputs
		clk = 0;
		rst = 1;
		btn1 = 0;
		btn2 = 0;
		sw = 8'b01010101;

		// Wait 100 ns for global reset to finish
		#10;
			rst=0;
		#2;
			btn2=1;
		#500;
			btn2=0;


		// Add stimulus here

	end
    always #1 clk=~clk;
endmodule
```

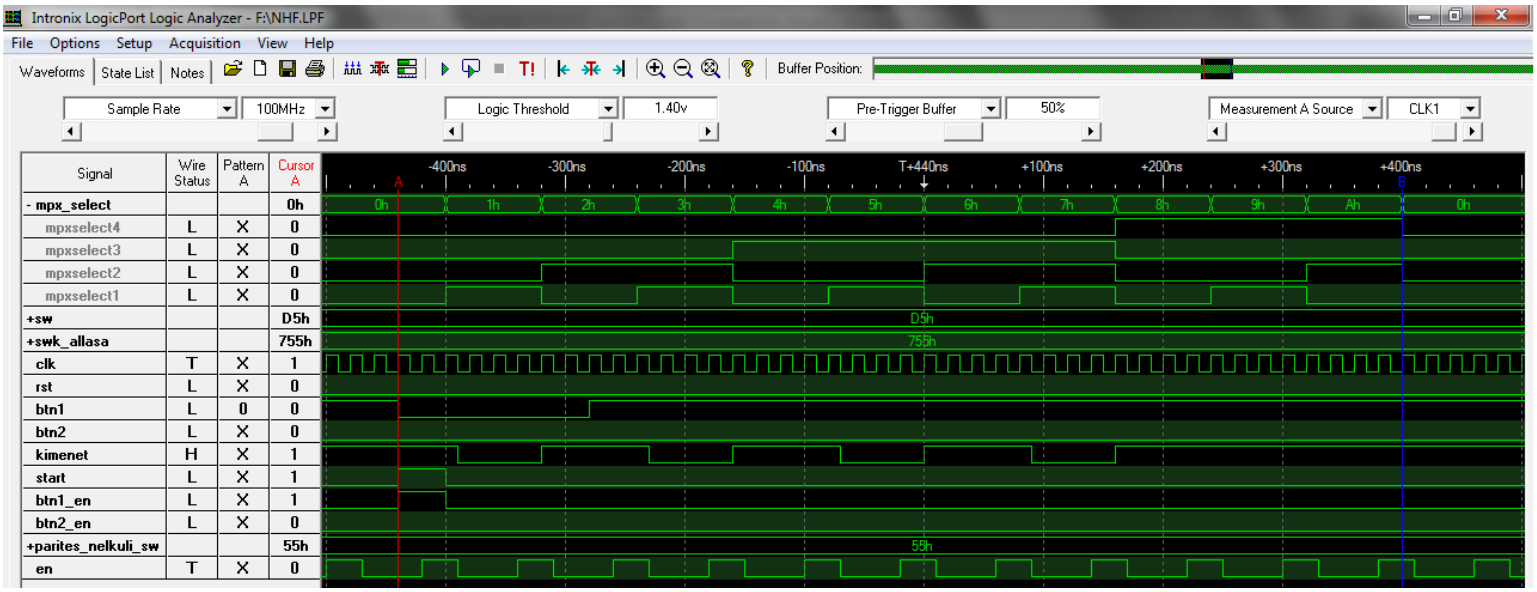
Működés vizsgálata FPGA és TVSIO segítségével

Kikommenteztük a test fixture-höz szükséges debug kimeneteket, illetve a baud generátornál visszaállítottuk 4800-ra a baud-ot. Így már tudtunk használható jelet venni:

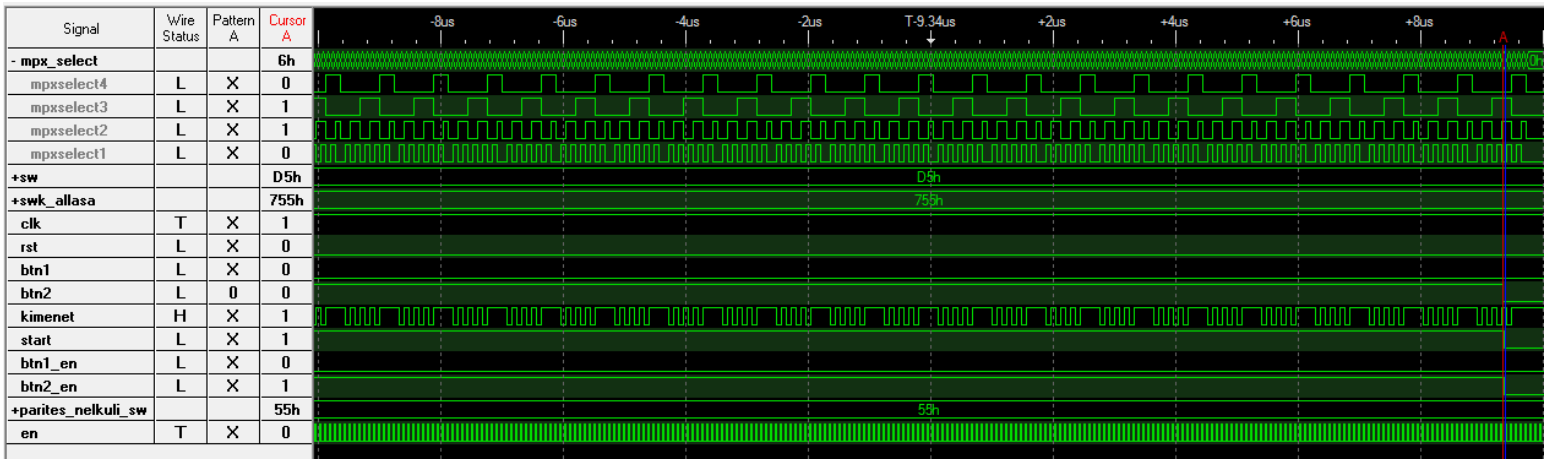
Switchek beállítása: 1010101, paritásbit pedig páratlan, mert az SW7-et is 1-re állítottuk be. Ez hexadecimálisan 55, vagyis jó eredményt kaptunk.

Lehet látni, hogy akkor is működik, ha csak a BTN1-et nyomjuk le, vagyis ilyenkor csak 1x küldi ki a beállított értéket, míg a fenti képen a BTN2-t lenyomva tartva sokkal többször adja ki a jelet.

Logikai analizátoros tesztelés



A teszteléshez át kellett állítani a baud rate-t, így sokkal több engedélyező jelet kap a rendszer. Triggerfeltételnek beírtuk, hogy a btn1 az legyen 0, és a tesztelésnél mielőtt elindítottuk az analizátor mintavételezését, folyamatosan nyomva tartottuk a btn1 gombot. Miután elengedtük látszik, hogy mind a start, mind a btn1_en jelek egy-re váltottak (piros kurzor jelöli), és a rendszer megkezdte a kiküldést. Lehet látni, hogy a multiplexer minden egyes jelet kiválaszt, ennek hatására a kimenet megváltozik, kiadja a biteket, majd be is fejezi a küldést, mivel a btn1 lenyomására csak egyszer küldi ki a beállított értékeket.



A btn2 tesztelésénél is azt a triggerfeltételt állítottuk be, hogy addig mintavételezzen, amíg az 0 nem lesz. Látszik, hogy amíg nyomva tartjuk a gombot, folyamatosan kiküldi a kimenetre a jeleket, majd ha elengedtük, az elkezdett kiküldést befejezi, majd nem is küld ki több jelet.

Javítás a kódban

Észrevettük, hogy a gombot viszonylag sokáig kell lenyomnunk ahhoz, hogy megjelenjen a jel a kimeneten, ennek az volt az oka, hogy a baud rate generátor kimenetét is rákötöttük a pergésmentesítőkben lévő számlálókra, vagyis csak akkor számoltak, ha a rendszer en-je 1 volt. Ennyit módosítottunk a kódon:

```
assign engedelyezes=(btn^carry);
 //assign engedelyezes=((btn^carry)&en);
```

Ezen kívül kivettük a paritásbitet a laborvezető utasítására, mert az elméletileg nem volt rossz, de helyes működését nem tudtuk megmagyarázni.

```
//assign out[9]=sw[0]^sw[1]^sw[2]^sw[3]^sw[4]^sw[5]^sw[6]^sw[7];
assign out[9]=1; // STOP bit - második
```