



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

Logikai tervezés

Házi feladat

Programozható jelgenerátor

Kardos Bálint, ZI84PX
Murányi Péter, A74MW9
Konzulens: Dr. Fehér Béla

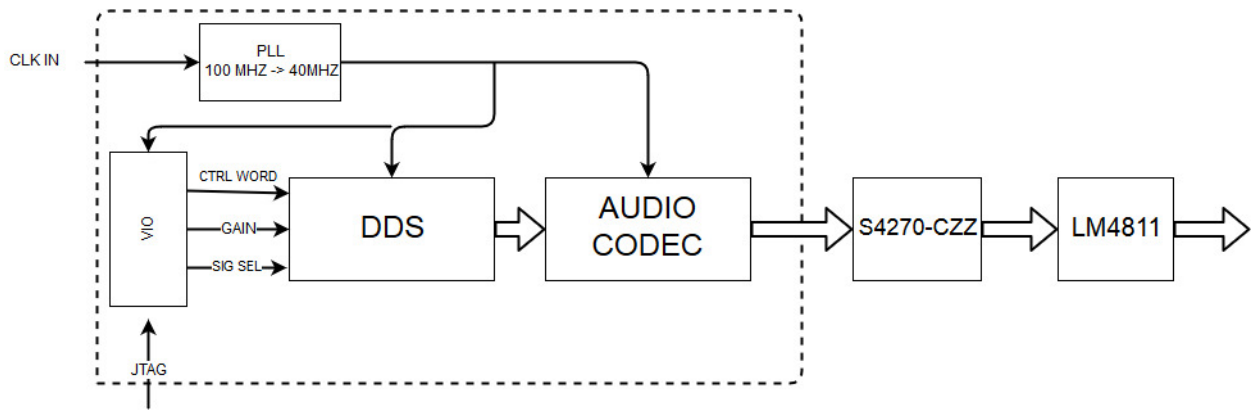
2018. május 26.

Tartalomjegyzék

1.	1
2.	1
2.1.	1
2.2.	1
2.3.	1
2.3.1.	1
2.3.2.	1
3.	1
3.1.	1
3.2.	1
4.	1
4.1.	1
4.2.	1
5.	1
5.1.	1
5.2.	1
5.2.1.	1
5.2.2.	1
5.3.	1
5.3.1.	1
5.4.	1
5.4.1.	1
6.	1
Hivatkozások	2

1. Feladat

2. Tervezés



1. ábra. Rendszer blokkdiagramja

3. Megvalósítás

3.1. DDS

A DDS modul tartalmazza a digitális jelet létrehozó hardvert. Az egész modul szinkron a rendszer 40MHz-s jelével, minden egység rendelkezik reset bemenettel, ami valamilyen kezdeti állapotba állítja azt. A modul belső funkciók szerint van szétosztva és megírva. A hullámformák párhuzamosan generálódnak, majd egy multiplexerrel választódik ki a megfelelő kimenet, ami egy szorzóval skálázódik.

Az első rész maga a NCO, ez verilogban nem több mint egy 24 bites regiszter amihez minden felfutó élen hozzáadjuk a vezérlő szavat. Az így kapott jel fogja a bemenetet képezni az összes többi alegységre. A számláló reset jellel nullázható.

A következő rész végzi a szinusz hullám generálását. A szinusz jel létrehozására egy LU-otT (LookUp Table) használunk. Ez a LUT tartalmazza a szinusz értékeit, mivel 24 bites a kimenet, így 24 bites előjeles formában. A LUT 1024 elemből áll, tehát 10 bites a címbemenete, és mivel minden extra bit felbontás 6 dB-t jelent a dinamika tartományban, így csak a szinusz egy negyede van eltárolva. Ahhoz hogy ebből az egy negyedből egy teljes szinuszt kapjunk, mind a LUT ki és bemenetét manipulálnunk kell, attól függően hogy a szinusznak éppen melyik negyedében vagyunk. Az NCO kimenetéhez egy dither jel van hozzáadva, hogy a kvantálási hibából adódó spektrumhibát korrigáljuk. A dither 12 bites, ugyanis úgy a legoptimálisabb, ha mérete megegyezik a kvantáláskor levágott bitek számával.

A negyed meghatározására az NCO+dither kimenetének felső két bitjét használjuk. Ezekkel határozzuk meg hogy éppen milyen műveleteket kell végrehajtani a LUT címbemenetére adott adattal, illetve a kimeneti jellel. A LUT címbemenete a NCO+dither jel [21:12] tartománya. Első negyedbe nem végzünk semmilyen műveletet a LUT címbemenetére adott jellel, kimenetét se manipuláljuk. Második negyedben a kimenetet ugyan nem manipuláljuk, de a címbemenet

értékét nullából kivonva adjuk a LUT bemenetére, ezzel fordított sorrendben kapjuk meg a jeleket a kimeneten. Harmadik és negyedik negyedben a kimeneten is műveletet kell végezni. Ahhoz hogy a szinusznak a negatív felét megkapjuk a LUT kimeneti értékét nullából vonjuk ki. Mivel a LUT egy órajeles késéssel ad jelet a kimenetén, a kimenettel végzett műveletet meghatározó jelet egy órajellel késleltetni kell egy flip-flop segítségével.

A négyszögjel létrehozása nagyon egyszerű, mindössze az NCO MSB-jét kell venni és ha nulla akkor a kimenet 0x800000, ha egy akkor 0x7FFFFFFF.

A háromszög jel se bonyolult, szinuszhoz hasonlóan itt is négy negyedre bontjuk a jelet és a negyedek szerint állítjuk elő. Első negyed az NCO nyers kimenete [21:0] tartományban, kettővel megszorozva (balra shiftelés). Második és harmadik negyedben az NCO [21:0] értékét 0x800000-ből vonjuk ki hogy megkapjuk a lefutó éleket. Negyedik negyedben szintén nem manipuláljuk az NCO kimenetét, ugyanis a kettes komplementes számábrázolás miatt a negatív tartományban vagyunk alaptól, elég egyszerűen felfelé számolni.

A kiválasztott jelet skálázni kell a bemenetnek megfelelően, erre egy szorzót használunk. A skálázó érték egy 16 bites kettes komplementes szám, amiből 2 bit MSB az egész érték, a többi törtrész. A szorzás után a kapott értéket el kell tolnunk jobbra 14 bittel, ugyanis fixpontos szorzásnál a törtrészek száma összeadódik. Az innen kapott kimeneti jel lesz a DDS kimeneti jele.

3.2. LUT

A modul egy python szkripttel lett generálva. A LUT tárolja a szinusz generátorhoz szükséges értékeket. Egy negyedét tárol 1024 mintán. A címbemenetre adott értékkel egy case blokk kiválasztja a megfelelő értéket és megjeleníti a kimeneten. Ebben a case blokkban lévő elemek egy python szkripttel lettek generálva. A LUT úgy lett megírva hogy a szintézer felismerje és blokkram-ba helyezze el. Így az 1000 elem egy 16k-s BRAM egységet foglal el.

3.3. Dithering

A pszeudo véletlenszám generátort, nem mi írtuk, kész kódot használtunk. A leírása és használatának menete [ezen a linken](#) érhető el. A modul úgy lett konfigurálva, hogy egy 12 bites uniform eloszlású véletlen számot adjon a kimenetén minden órajelben.

3.4. Audio codec

A külső audio codec meghajtására a tárgy során gyakorlaton készített FIR szűrő feladatból vett kódot használjuk. Mivel 40MHz-vel van meghajtva a rendszer, így át kellett állítani a konfigurációs lábak állapotát, hogy single-speed üzemmódba működjön az eszköz. Ezen túl nem kellett módosítani a kódot. A kimenet mindkét csatornájára a DDS kimeneti jele van csatlakoztatva.

3.5. VIO és PLL

A DDS konfigurálásához három jel szükséges. Az NCO szabályozó szava (frekvencia), skálázás mértéke, illetve a jelforma kiválasztó jel. Ezeket a bemeneteket nem lehetett volna egyszerűen megvalósítani a Logsys Kintex-7 kártyán, így JTAG-en keresztül VIO-val valósítjuk meg ezeket. Ennek segítségével a számítógéphez csatlakoztatva, Vivado-n keresztül tudjuk állítani ezeket az értékeket. A VIO modul a Vivado beépített IP varázslójával lett létrehozva.

Mivel a belső rendszer meghajtásához 100MHz a kimeneti 20Hz - 20kHz frekvenciatartományt figyelembe véve túl sok, alacsonyabb frekvenciát érdemes használni. Ez 40MHz-nak lett megválasztva, melyet egy PLL-el állítunk elő. Ezt a PLL-t szintén a Vivado IP varázslójával állítottuk elő.

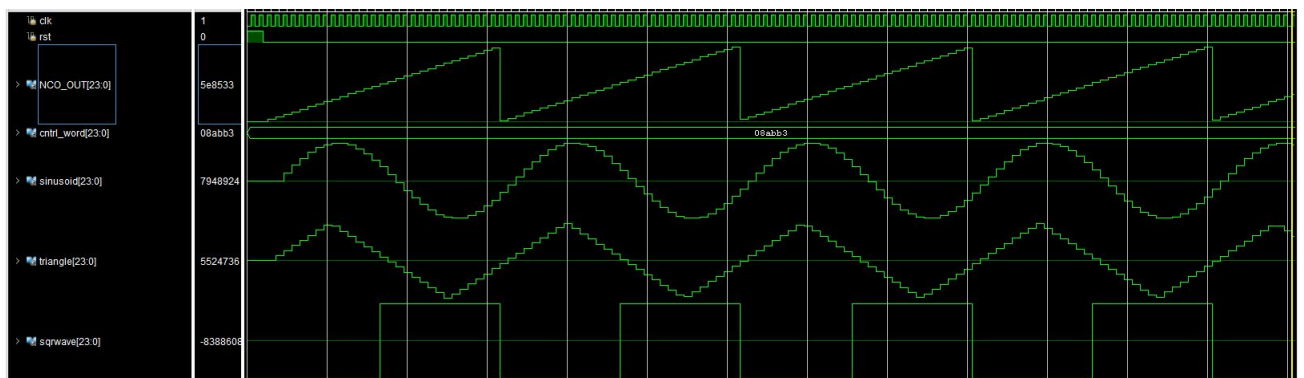
3.6. Toplevel

A toplevel csak a modulokat és összeköttetéseket tartalmazza. A bemeneti 100MHz a PLL bemenetét hajtja meg. Az összes modul ennek a PLL-nek a 40MHz-s kimenetével van meghajtva. A reset vonal egy külső dombbal és a PLL locked invertált kimenetével van ÉS kapcsolatban, hogy minden modul akkor induljon csak el ha már stabil a belső órajel.

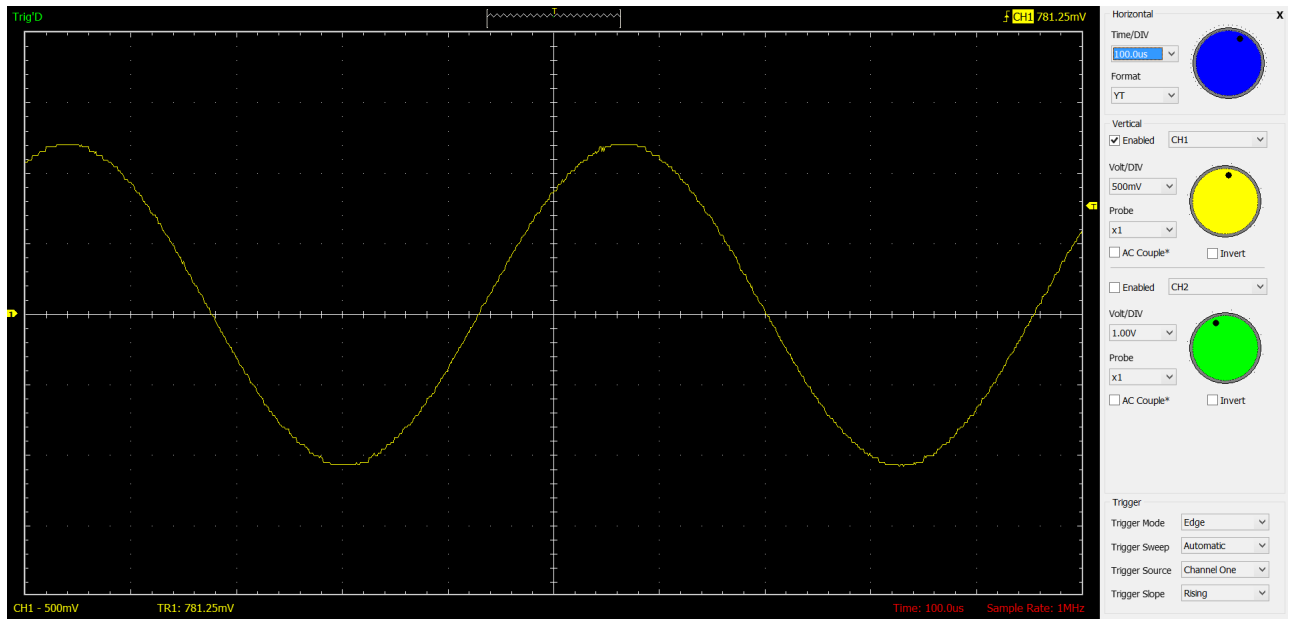
Mivel az audio codec kimenete egy állítható erősítőre van kötve, így a panelon egy gomb és tolókapcsoló rá lett kötve erre az erősítőre, hogy állítható legyen az erősítése.

4. Erőforrásigény

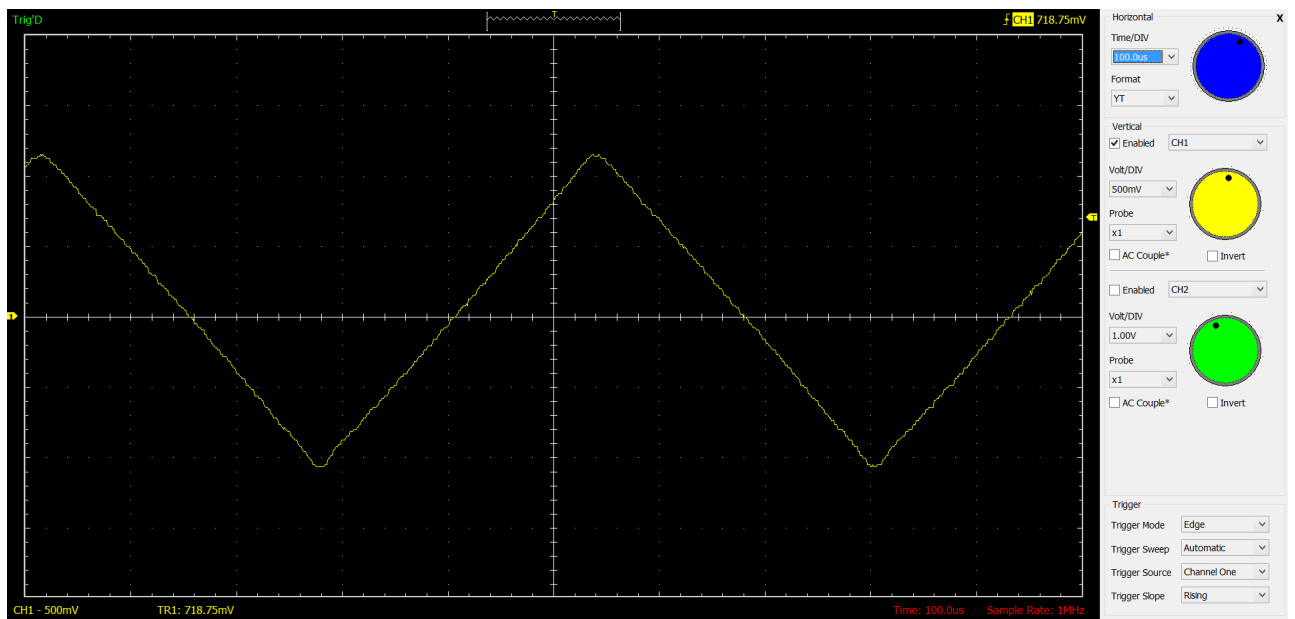
5. Szimuláció



2. ábra. DDS szimulációja



3. ábra. Szinuszos kimenet



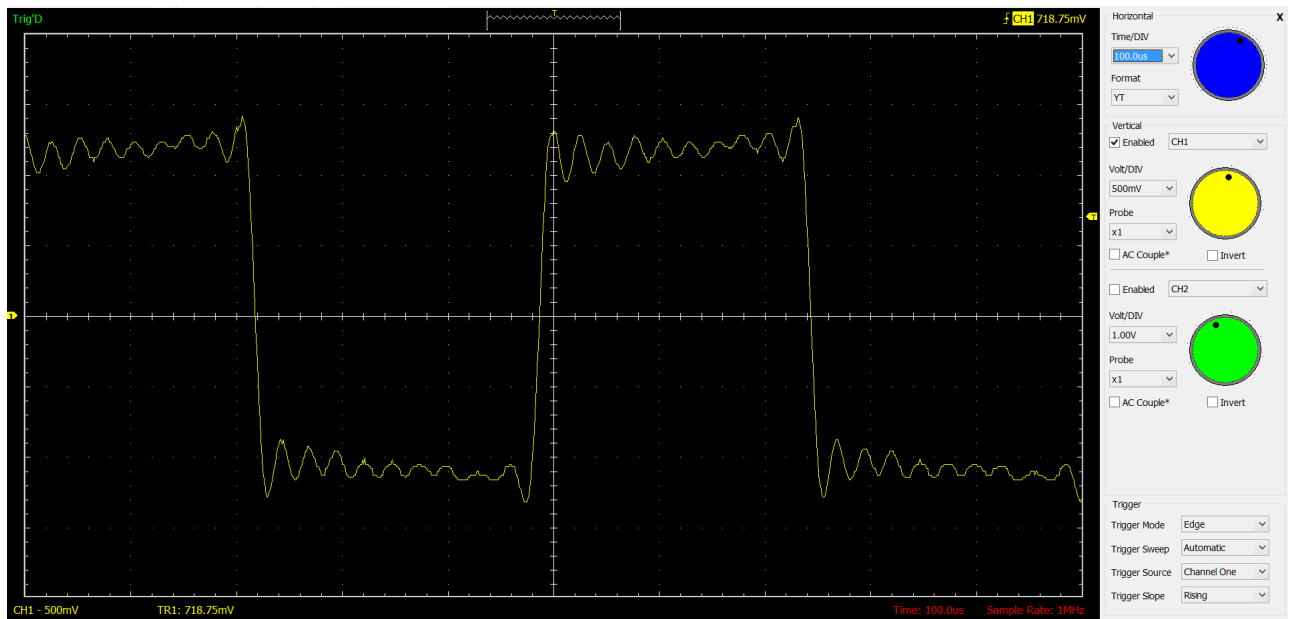
4. ábra. Háromszögjel kimenet

6. Éles teszt

7. Forráskód

7.1. DDS

```
'timescale 1ns / 1ps
```



5. ábra. Négyszögjel kimenet

```

module DDS (      //Reset,clock
    input clk,
    input rst,
    //Controll
    input [23:0] cntrl_word,
    input [1:0] wave_sel,
    input [14:0] attenn,
    //Output
    output [23:0] wave_out,
    //DEBUG
    output [23:0] nco_out,
    output [23:0] sin_out,
    output [23:0] tri_out,
    output [23:0] sqr_out,
    output [23:0] nco_dith,
    output [11:0] rnd

);

//NCO
reg [23:0] nco_cnt = 0;

always @ (posedge clk)
begin
    #1;
    if(rst)
        nco_cnt <= 0;
    else
        begin
            nco_cnt <= nco_cnt + cntrl_word;
        end
end

```

```

end

//Dither
wire [11:0] rnd_gen;
wire [23:0] dith_out = nco_cnt + rnd_gen[11:0];

LFSR_Plus dither( .clk(clk),
                  .n_reset(~rst),
                  .enable(1'b1),
                  .g_noise_out(rnd_gen));

//SIN gen
//Használt regiszterek,összeköttetések
wire [1:0] quad_sel = dith_out[23:22]; //A negyed meghatározására az NCO felső 2 bitjét használjuk
reg quad_sel_z1, quad_sel_z2;
reg [9:0] LUT_address;
wire [22:0] sin_lut_out;
reg signed [23:0] sin_signal = 0;

//LUT be és kimenetének manipulálása
always @ (posedge clk)
begin
    #1;
    if(rst)
        begin
            LUT_address <= 0;
            sin_signal <= 0;
            quad_sel_z1 <= 0;
            quad_sel_z2 <= 0;
        end
    else
        begin
            //LUT kimenete 2 órajellel késleltetve van
            quad_sel_z1 <= quad_sel[1];
            quad_sel_z2 <= quad_sel_z1;
            //Címbevitel kezelése
            if(quad_sel[0]) //2. és 4. negyed
                begin
                    LUT_address <= 10'h3FF - dith_out[21:12];
                end
            else //1. és 3. negyed
                begin
                    LUT_address <= dith_out[21:12];
                end
            //LUT kimenet műveletei
            if(quad_sel_z1) // 2. és 3. negyed
                begin
                    sin_signal <= 0 - sin_lut_out;
                end
            else //1. és 2. negyed
                begin
                    sin_signal <= sin_lut_out;
                end
        end
    end
end

```



```

end

    //LUT példányosítása
    SIN_LUT singen( .clk(clk),
                    .address(LUT_address),
                    .sin_out(sin_lut_out));

    //COMP (SQR Wave)
    wire [23:0] sqr_signal;
    assign sqr_signal = (nco_cnt[23])? 24'h7FFFFFFF: 24'h800000;

    //TRI Wave
    reg signed [23:0] triangle_out = 0;
    wire [23:0] nco_tri_in = nco_cnt;

    always @ (posedge clk)
    begin
        #1;
        case(nco_tri_in[23:22])
            2'b00 : triangle_out[23:1] <= nco_tri_in;
            2'b01 : triangle_out[23:1] <= 24'h800000 - nco_tri_in;
            2'b10 : triangle_out[23:1] <= 24'h800000 - nco_tri_in;
            2'b11 : triangle_out[23:1] <= nco_tri_in;
        endcase
    end

    //Output select
    wire signed [23:0] selected_wave;
    assign selected_wave = (wave_sel == 2'b0)? sin_out: (wave_sel == 2'b01)? tri_out: (wave_sel == 2'b10) ? sqr_out :

    //Attenuation
    reg signed [39:0] multiply_out = 0;
    wire signed [16:0] attenuation;

    assign attenuation = {1'b0,attenn};

    always @ (posedge clk)
    begin
        #1;
        if(rst)
            begin
                multiply_out <= 0;
            end
        else
            begin
                multiply_out <= selected_wave * attenuation;
            end
    end

    assign wave_out = multiply_out[38:14];

    //Debug connections

```

```

    assign nco_out = nco_cnt;
    assign sin_out = sin_signal;
    assign tri_out = triangle_out;
    assign sqr_out = sqr_signal;
    assign nco_dith = dith_out;
    assign rnd = rnd_gen;

endmodule

```

7.2. Szinuszos LUT

```

`timescale 1ns / 1ps

//1024 elemű  $\frac{1}{2}$  LUT egy szinuszos első  $\frac{1}{2}$  negyedű  $\frac{1}{2}$  tartalmazza

module SIN_LUT( input clk,
                input [9:0] address,
                output reg [22:0] sin_out
                );

    //Address latching
    reg [9:0] address_reg;
    always @ (posedge clk)
    begin
        address_reg <= address;
    end

    //LUT
    always @*
    begin
        case(address_reg)
            10'h000 : sin_out = 23'h000000;
            10'h001 : sin_out = 23'h003244;
            10'h002 : sin_out = 23'h006488;
            10'h003 : sin_out = 23'h0096cc;
            10'h004 : sin_out = 23'h00c910;
            10'h005 : sin_out = 23'h00fb53;
            .
            .
            .
            .
            .
            10'h3fc : sin_out = 23'h7fff62;
            10'h3fd : sin_out = 23'h7fffa7;
            10'h3fe : sin_out = 23'h7fffd9;
            10'h3ff : sin_out = 23'h7ffff6;
        endcase
    end

endmodule

```

7.3. DDS toplevel

```

`timescale 1ns / 1ps

```

```

module DDS_t1( //CLK in
    input clk,
    input rst,
    //Codec wires
    output        codec_m0,
    output        codec_m1,
    output        codec_i2s,
    output        codec_mdiv1,
    output        codec_mdiv2,
    output        codec_rstn,
    output        codec_mclk,
    output        codec_lrclk,
    output        codec_sclk,
    output        codec_sdin,
    input         codec_sdout,
    //Volume
    input         bt3,
    input         sw0,
    output        vol_clk,
    output        vol_ud
);

assign vol_clk = bt3;
assign vol_ud  = sw0;

wire clk_pll_out;
wire reset_internal;
wire pll_lock;
//CLK generation
clk_wiz_0 clock_pll(.clk_in1(clk),
                    .reset(rst),
                    .locked(pll_lock),
                    .clk_out1(clk_pll_out)
                    );

//RESET
assign reset_internal = ~pll_lock | rst;
//VIO
wire [23:0] cntrl_word;
wire [1:0] signal_sel;
wire [14:0] gain_set;
vio_DDS vio(.clk(clk_pll_out),
            .probe_out0(cntrl_word),
            .probe_out1(signal_sel),
            .probe_out2(gain_set));

//DDS
wire [23:0] wave_out;
DDS dds_int(.clk(clk_pll_out),
            .rst(reset_internal),
            .cntrl_word(cntrl_word),
            .wave_sel(signal_sel),
            .attenn(gain_set),
            .wave_out(wave_out),
            .nco_out(),

```

```

        .sin_out(),
        .tri_out(),
        .sqr_out(),
        .nco_dith(),
        .rnd());

//AUDIO CODEC
codec_if codec( .clk(clk_pll_out),
               .rst(reset_internal),
               //IO connections
               .codec_m0      (codec_m0),
               .codec_m1      (codec_m1),
               .codec_i2s      (codec_i2s),
               .codec_mdiv1    (codec_mdiv1),
               .codec_mdiv2    (codec_mdiv2),
               .codec_rstn     (codec_rstn),
               .codec_mclk     (codec_mclk),
               .codec_lrclk    (codec_lrclk),
               .codec_sclk     (codec_sclk),
               .codec_sdin     (codec_sdin),
               .codec_sdout    (codec_sdout),
               //We don't care about outputs
               .aud_dout_vld   (),
               .aud_dout       (),
               //Inputs, same on both channel
               .aud_din_vld    (2'b11),
               .aud_din_ack    (),
               .aud_din0       (wave_out),
               .aud_din1       (wave_out)
               );

endmodule

```

Hivatkozások

- [1] 2018. május 26.
- [2] *Dither generátor*
<https://eewiki.net/pages/viewpage.action?pageId=16351401>
2018. május 26.
- [3] 2018. május 26.
- [4] 2018. május 26.
- [5]
- [6] 2018. május 26.