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 23.

Tartalomjegyzék

1.

2. Megvalósítás

2.1. DDS

2.2. Dithering

2.3. Toplevel

3. Forráskód

3.1. DDS

```
`timescale 1ns / 1ps



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

    //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 felsõ 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;
            //Címbemenet 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 mûveletei
            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
    //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'h7FFFFF: 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] attennuation;

assign attennuation = {1'b0,attenn};

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

3.2. DDS toplevel

```
`timescale 1ns / 1ps

module DDS_tl(  //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 23.

[2]
2018. május 23.

[3]
2018. május 23.

[4]
2018. május 23.

[5]

[6]
2018. május 23.