Bibliography

[1] R. Van Nee and R. Prasad, “OFDM For Wireless Multimedia Communications", Artech House Boston, London, 2000.

[2] IEEE Standard 802.11a-1999, 30 December 1999.

[3] IEEE Standard 802.11b-1999, Supplement to 802.11 1999,Wireless LAN MAC and PHY specifications: Higher speed Physical Layer (PHY) extension in the 2.4 GHz band

[4] J. Tierney, C.M. Rader, and B. Gold, “A Digital Frequency Synthesizer,” IEEE Trans. Audio and Electro-Acoustics, vol. AU-19, no. 1, pp. 48-57, Mar. 1971.

[5] J. Vankka, “Methods of Mapping from Phase to Sine Amplitude in Direct Digital Synthesis,” IEEE Trans. Ultrasonics Ferroelectrics, and Frequency Control, vol.44, pp.526-534, March 1997.

[6] J.M.P. Langlois and D. Al-Khalili,” Hardware Optimized Direct Digital Frequency Synthesizer Architecture with 60 dBc Spectral Purity,” Circuits and Systems, 2002. ISCAS 2002. IEEE International Symposium, Vol. 5, pp.361-364, May 2002.

[7] K.I. Palomaki and J. Nittylahti,” A Low-Power, Memoryless Direct Digital Frequency Synthesizer Architecture,” Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium, Vol 2, pp.77-80 May 2003.

[8] A.N. Mohieldin, A.A Emira and E. Sanchez-Sinencio “A 100-MHz 8-mW ROM-less Quadrature Direct Digital Frequency Synthesizer,” IEEE Journal of Solid-State Circuits, Vol 37, pp. 1235 –1243, Oct. 2002.

[9] J.M.P. Langlois and D. Al-Khalili, ”ROM Size Reduction with Low Processing Cost for Direct Digital Frequency Synthesis,” Communications, Computers and signal Processing, 2001. PACRIM. 2001 IEEE Pacific Rim Conference, Vol 1,pp. 287 –290, Aug. 2001.

[10] D.Soudris, M. Kesoulis, C. Koukourlis, A. Thanailakis and S. Blionas, “Alternative Direct Digital Frequency Synthesizer Architectures with Reduced Memory Size,” Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium, Vol 2, pp.73-76 May 2003

[11] Y.S. Kim, S.H. Kim, K.H. Baek, S.Kim and S.M. Kang,” Multiple Trigonometric Approximation of Sine-Amplitude with Small ROM Size for Direct Digital Frequency Synthesizers,” VLSI Design, 2003. Proceedings. 16th International Conference on pp. 261-265, 4-8 Jan. 2003

[12] D.A. Sunderland, R.A. Strauch, S.S. Wharfield, H.T. Peterson, and C.R. Cole,“CMOS/SOS Frequency Synthesizer LSI Circuit for Spread Spectrum Communications,” IEEE Journal of Solid State Circuits, vol. SC-19, pp. 497-505, Aug 1984.

[13] H. T. Nicholas III et al., “The Optimization of Direct Digital Frequency Synthesizer Performance in the Presence of Finite Word Length Effects,” in Proc. 42nd Annu. Frequency Control Symp. , pp. 357–363, 1988.

[14] K.I. Palomaki and J. Nittylahti, “Methods to Improve the Performance of Quadrature Phase-to-Amplitude Conversion Based on Taylor Series Approximation,” in Proc. The 43rd IEEE Midwest Symposium on Circuits and Systems, Lansing, MI, USA, Aug. 8-11 2000

[15] A.Yamagishi et al., “A 2-V, 2-GHz Low-Power Direct Digital Frequency Synthesizer Chip-Set for Wireless Communication,” IEEE J. Solid-State Circuits, vol.33, pp.210-217, Feb.1998.

[16] A. M. Sodagar and G. R. Lahiji, “Mapping from Phase to Sine-Amplitude in Direct Digital Frequency Synthesizers using Parabolic Approximation,” in IEEE Transactions on Circuits and Systems-Ⅱ: Analog and Digital Signal Processing vol.47, pp.1452-1457. Dec. 2000

[17] A. Torosyan and A.N Willson,” Analysis of the Output Spectrum for Direct Digital Frequency Synthesizers in the Presence of Phase Truncation and Finite Arithmetic Precision,” Image and Signal Processing and Analysis, 2001. ISPA 2001. Proceedings of the 2nd International Symposium, pp. 458 –463, June 2001

[18] H. T. Nicholas and H. Samueli, “A Analysis of the Output Spectrum of Direct Digital Frequency Synthesizers in the Presence of Phase-Accumulator Truncation,” in Proc. 41st Annu. Frequency Control Symp. , pp. 495–502, 1987.

[19] V.F Kroupa, ”Discrete Spurious Signals and Background Noise in Direct Digital Frequency Synthesizers,” in IEEE Proc. IFCS, pp242-250, 1993

[20] J. Vankka and K. Halonen, “Direct Digital Synthesizers Theory, Design and Applications," Kluwer Academic Publishers, 2001

[21] H. T. Nicholas and H. Samueli, “A 150-MHz Direct Digital Frequency Synthesizer in 1.25-um CMOS with –90dBc Spurious Performance,” IEEE Journal of Solid State Circuits, vol.26, pp. 1959–1969, 1991

Appendix

1. Matlab program for DDFS

(1).findrom.m

```
%*************************************************************************************/
%   M-file          :   findrom.m                                                    *
%   VERSION         :   1.0                                                          *
%   DATE            :   Jul. 1 2003                                                  *
%   AUTHOR          :   Cheng Hen Shan                                               *
%   DESCRIPTION     :   This m-file is used for the following purpose                *
%                       (1). Verify the algorithm of proposed DDFS                   *
%                       (2). Find value of coarse and fine ROM for the proposed DDFS *
%                       (3). Calculate the SFDR and plot the approximate sine wave form *
/*************************************************************************************/
parameter_store = 1;
for i=1:1024
    rel_amp(i) = sin(pi*i/(2*2^10)-0.0005);
    quarter_phase(i) = pi*i/(2*2^10)-0.0005;
    app_amp1(i) = fix(i/2);
    if(i<511)
        app_amp2(i)= fix(i/8);
    else
        app_amp2(i) = 128 - fix(i/8);
    end
end
app_amp = (app_amp1 + app_amp2)/512;
abits = 2;
bbits = 4;
cbits = 4;
%--------------------------------------------------------------------------------------*
if (parameter_store == 1)
    [fine_amp,fine_value,temp_error] = errortable(quarter_phase,app_amp,abits,bbits,cbits);
end
%--------------------------------------------------------------------------------------*
%           Use for calculating the coarse and fine error correct value                *
%--------------------------------------------------------------------------------------*
% ------------------------- Store the error correct table to file ---------------------*
```

```
if (parameter_store == 1)
     errortablesave(fine_value,temp_error,'simufloat','simuint','simuhex',9);
end
[fine_amp,fine_hex,coarse_hex,app_amp] = errorlookuptable1(app_amp,quarter_phase,abits,bbits,cbits);
% ---------------------------------- store the error correct table to file ---------------------------------------------------*
for i=1:4096
    rel_amp(i) = sin(pi*i/(2*2^10)-0.0005);
    phase(i) = pi*i/(2*2^10)-0.0005;
end
% -------------------------------- Calculate the second quantaure sin amplitude -----------------------------------------*
quarter_phase = phase(find(phase > 0.5*pi & phase <= pi));
quarter_phase = pi - quarter_phase;
for i = length(quarter_phase):-1:1
    sec_quarter_phase(length(quarter_phase)-i+1) = quarter_phase(i);
end
[temp_amp_quad2,fine_hex,coarse_hex,temp_second_quad] =
errorlookuptable1(app_amp,sec_quarter_phase,abits,bbits,cbits);
for i = length(temp_second_quad):-1:1
    amp_quad2(length(temp_second_quad)-i+1) = temp_amp_quad2(i);
    second_quad(length(temp_second_quad)-i+1) = temp_second_quad(i);
end
% ------------------------------ Calculate the third quantaure sin amplitude ---------------------------------------------*
quarter_phase = phase(find(phase > pi & phase <= 1.5*pi));
third_quarter_phase = quarter_phase - pi;
[amp_quad3,fine_hex,coarse_hex,third_quad] =
errorlookuptable1(app_amp,third_quarter_phase,abits,bbits,cbits);
amp_quad3 = -1.*amp_quad3;
third_quad = -1.*third_quad;
% ----------------------------- Calculate the forth quantaure sin amplitude ---------------------------------------------*
quarter_phase = phase(find(phase > 1.5*pi & phase <= 2*pi));
quarter_phase = 2*pi - quarter_phase ;
for i = length(quarter_phase):-1:1
    forth_quarter_phase(length(quarter_phase)-i+1) = quarter_phase(i);
end
[temp_amp_quad4,fine_hex,coarse_hex,temp_forth_quad] =
errorlookuptable1(app_amp,forth_quarter_phase,abits,bbits,cbits);
for i = length(temp_forth_quad):-1:1
    amp_quad4(length(temp_forth_quad)-i+1) = temp_amp_quad4(i);
    forth_quad(length(temp_forth_quad)-i+1) = temp_forth_quad(i);
end
amp_quad4 = -1.*amp_quad4;
forth_quad = -1.*forth_quad;
% ---------------------------------- Combine four quantaure sin amplitude -------------------------------------------*
roY = [ fine_amp amp_quad2 amp_quad3 amp_quad4];
approx_amp = [app_amp second_quad third_quad forth_quad];
% --------------------------------- Calculate the DFT to measure spur ------------------------------------------------*
iN = length( roY );
roPsd = abs( fft( roY ) );
fiFloor = -160;
%----------- Find the possible zeros and replace it with the fiFloor [dB] constant (10^(fiFloor/20)) ----------*
    Min = 10^(fiFloor/20);
    roPsd = (2/iN) .* roPsd;
    roPsd(1) = 1/2 * roPsd(1);        % Adjust the DC
    roPsd( find( roPsd < Min ) ) = Min;
    roPsd = 20 .* log10( roPsd );
%---------------------------------------------- Calculate the SFDR --------------------------------------------------*
    temp_psd = roPsd;
    [maxsign,index] = max(roPsd);
    temp_psd(index) = 20 * log10( Min );
    [max_noise,index] = max(temp_psd);
    temp_psd(index) = 20 * log10( Min );
    [max_noise,index] = max(temp_psd);
    spur = maxsign - max_noise;
    spur;
%---------------------------------------- Plot the approximate Sine wave for-----------------------------------------*
figure;
plot(phase/pi,rel_amp,'-',phase/pi, roY,'-');
xlabel('phase accumulator □');ylabel('Sine Amplitude');
figure;
plot(phase/pi,rel_amp,'-',phase/pi,approx_amp,'-');
xlabel('phase accumulator □');ylabel('Sine Amplitude');
```

(2). errortable.m

```
%*********************************************************************************/
%   M-file              :   errortable.m                                         *
%   VERSION             :   1.0                                                  *
%   DATE                :   Jul 20, 2003                                         *
%   AUTHOR              :   Cheng Hen Shan                                       *
%   DESCRIPTION         :   This m-file is used for the following purpose        *
%                           Subroutine was used for finding value of coarse and fine ROM for the   *
%                           proposed DDFS                                        *
/*********************************************************************************/
function [fine_signal,fine_value,temp_error] = errortable(quarterphase,approx_signal,abits,bbits,cbits)
% ---------------------------------- Calculate the first quantaure sin amplitude ----------------------------------------*
origin_amp = sin(quarterphase);
%--------------------------------------------- Coarse interval for phase ---------------------------------------------------*
coarsetheta = pi/(2^(abits+bbits+1));
%--------------------------------------------- Fine interval for phase e ---------------------------------------------------*
finetheta = coarsetheta/(2^cbits);
% --------------------------------- Calculate the coarse error amplitude ------------------------------------------------*
number_coarse = 2^(abits+bbits);
number_fine = 2^cbits;
temp_signal = [];
for i = 1: number_coarse
    range_index = find(quarterphase > (i-1)*coarsetheta & quarterphase <= i*coarsetheta);
    origin_amp = sin(quarterphase(find(quarterphase > (i-1)*coarsetheta & quarterphase <= i*coarsetheta)));
    temp_amp = approx_signal(find(quarterphase > (i-1)*coarsetheta & quarterphase <= i*coarsetheta));
    temp_error(i) = min(origin_amp - temp_amp);
    temp_amp = temp_amp + temp_error(i);
    temp_signal = [temp_signal    temp_amp];
end
gross_error_max = max(temp_error);
% --------------------------------- Calculate the fine error amplitude ---------------------------------------------------*
num_b = 2^bbits;
fine_signal = [];
for i = 1:2^abits
    for k = 1:2^bbits
        for j = 1:2^cbits
            origin_amp = sin(quarterphase(find(quarterphase >
(((i-1)*num_b+k-1)*coarsetheta+(j-1)*finetheta) & quarterphase
<=(((i-1)*num_b+k-1)*coarsetheta+j*finetheta))));
            temp_amp = temp_signal(find(quarterphase > (((i-1)*num_b+k-1)*coarsetheta+(j-1)*finetheta)
& quarterphase <= (((i-1)*num_b+k-1)*coarsetheta+j*finetheta)));
            temp_error2(k,j) = median(origin_amp - temp_amp);
        end
    end
    fine_value(i,:) = median(temp_error2,1);
    for k = 1:2^bbits
        for j = 1:2^cbits
            temp_amp = temp_signal(find(quarterphase > (((i-1)*num_b+k-1)*coarsetheta+(j-1)*finetheta)
& quarterphase <= (((i-1)*num_b+k-1)*coarsetheta+j*finetheta)));
            temp_amp = temp_amp + fine_value(i,j);
            fine_signal = [fine_signal temp_amp];
        end
    end
end
temp_error_max = max(fine_value);
fine_error_max = max(temp_error_max);
```

(3). errortablesave.m

```
%********************************************************************************/
%   M-file          :   errortablesave.m                                           *
%   VERSION         :   1.0                                                        *
%   DATE            :   Jul 20, 2003                                               *
%   AUTHOR          :   Cheng Hen Shan                                             *
%   DESCRIPTION     :   This m-file is used for the following purpose             *
%                       Subroutine was used for saving value of coarse and fine ROM for the   *
%                       proposed DDFS to file                                      *
/********************************************************************************/
function    errortablesave(fine_value,coarse_value,float_file,integ_file,hex_file,output_bit)
temp_coarse_max = max(coarse_value);
temp_error_max = max(fine_value);
temp_fine_max = max(temp_error_max);
i = 1;
% ------------------------------------- Calculate the bit save for Coarse ROM ---------------------------------------*
while temp_coarse_max < 1/(2^i),
      i = i   + 1;
```

```
end
save_coarse_bit = i-1;
% -------------------------------------- Calculate the bit save for Fine ROM ------------------------------------------*
i = 1;
while temp_fine_max < 1/(2^i),
          i = i    + 1;
end
save_fine_bit = i-1;
actual_bit = output_bit - save_coarse_bit;
% -------------------------------- Converter integral number of coarse correct ----------------------------------------*
coarse_int = floor(coarse_value*(2^save_coarse_bit)*(2^actual_bit)+0.5);
coarse_hex = dec2hex(coarse_int);
% -------------------------------- Converter integral number of coarse correct ----------------------------------------*
actual_bit = output_bit - save_fine_bit;
fine_int = floor(fine_value*(2^save_fine_bit)*(2^actual_bit)+0.5);
fine_hex = dec2hex(fine_int');
%------------------------------- Save to file for float error correct ROM table -----------------------------------------*
[coarse_row,coarse_col] = size(coarse_value);
[fine_row,fine_col] = size(fine_value);
sub_name1 = '_coarse';
sub_name2 = '_fine';
temp_name1 = [float_file sub_name1];
temp_name2 = [integ_file sub_name1];
temp_name3 = [float_file sub_name2];
temp_name4 = [integ_file sub_name2];
temp_name5 = [hex_file sub_name1];
temp_name6 = [hex_file sub_name2];
fid1 = fopen(temp_name1,'w');
fid2 = fopen(temp_name2,'w');
fid3 = fopen(temp_name3,'w');
fid4 = fopen(temp_name4,'w');
fid5 = fopen(temp_name5,'w');
fid6 = fopen(temp_name6,'w');
fprintf(fid1,'%f \n',coarse_value);
fprintf(fid2,'%d \n',coarse_int);
%------------------------------------------ First write row and col numbers --------------------------------------------*
fprintf(fid3,'%u %u\n',fine_row,fine_col);
fprintf(fid4,'%u %u\n',fine_row,fine_col);
fprintf(fid3,'%f \n',fine_value');
fprintf(fid4,'%d \n',fine_int');
[row,column] = size(coarse_hex);
for i = 1:row
    temp_string = coarse_hex(i,1:column);
    fprintf(fid5,'%s \n',temp_string);
end
[row,column] = size(fine_hex);
for i = 1:row
    temp_string = fine_hex(i,1:column);
    fprintf(fid6,'%s \n',temp_string);
end
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
fclose(fid5);
fclose(fid6);
```

(4). errorlookuptable.m

```
%*********************************************************************************/
%   M-file          :   errorlookuptable                                         *
%   VERSION         :   1.0                                                      *
%   DATE            :   Aug 4, 2003                                              *
%   AUTHOR          :   Cheng Hen Shan                                           *
%   DESCRIPTION     :   This m-file is used for the following purpose            *
%                       Subroutine was used for loading fine and coarse error correct from file   *
%                       for the proposed DDFS                                    *
/*********************************************************************************/
function [fine_signal,fine_hex,coarse_hex,approx_signal] = errorlookuptable(quarterphase,abits,bbits,cbits)
% ------------------------- Load fine and coarse error correct from table ---------------------------------------------*
[fine_value,coarse_value,fine_hex,coarse_hex] = loaderrortable('simufloat','simuint');
%------------------------ First approximation for sin function - one line approximation   -------------------------*
app_amp1 = 2*quarterphase/pi;
%------------------------ Second approximation for sin function - two line appraximation------------------------*
app_amp2 = quarterphase(find(quarterphase <= 0.25*pi))/(2*pi);
app_amp3 = 0.25 - quarterphase(find(quarterphase > 0.25*pi & quarterphase <= 0.5*pi))/(2*pi);
```

```
approx_signal = app_amp1 + [ app_amp2    app_amp3];
%---------------------------------------------------- Coarse interval for phase -------------------------------------------*
coarsetheta = pi/(2^(abits+bbits+1));
%---------------------------------------------------- Fine interval for phase ----------------------------------------------*
finetheta = coarsetheta/(2^cbits);
% -------------------------------------------- Calculate the coarse error amplitude --------------------------------------*
number_coarse = 2^(abits+bbits);
number_fine = 2^cbits;
temp_signal = [];
for i = 1: number_coarse
    range_index = find(quarterphase > (i-1)*coarsetheta & quarterphase <= i*coarsetheta);
    if(size(range_index) > 0)
        temp_amp = approx_signal(find(quarterphase > (i-1)*coarsetheta & quarterphase <=
i*coarsetheta));
        temp_amp = temp_amp + coarse_value(i);
        temp_signal = [temp_signal    temp_amp];
    end
end
gross_error_max = max(coarse_value);
% -------------------------------------------- Calculate the fine error amplitude ---------------------------------------*
num_b = 2^bbits;
fine_signal = [];
for i = 1:2^abits
    for k = 1:2^bbits
        for j = 1:2^cbits
            range_index = find(quarterphase > (((i-1)*num_b+k-1)*coarsetheta+(j-1)*finetheta) &
quarterphase <= (((i-1)*num_b+k-1)*coarsetheta+j*finetheta));
            if(size(range_index) > 0)
                temp_amp = temp_signal(find(quarterphase >
(((i-1)*num_b+k-1)*coarsetheta+(j-1)*finetheta) & quarterphase <=
(((i-1)*num_b+k-1)*coarsetheta+j*finetheta)));
                temp_amp = temp_amp + fine_value(i,j);
                fine_signal = [fine_signal temp_amp];
            end
        end
    end
end
temp_error_max = max(fine_value);
```

```
fine_error_max = max(temp_error_max);
```

(5). loaderrortable.m

```
%*********************************************************************************/
%   M-file              :    loaderrortable.m                                           *
%   VERSION             :    1.0                                                         *
%   DATE                :    Aug 5, 2003                                                 *
%   AUTHOR              :    Cheng Hen Shan                                              *
%   DESCRIPTION         :    This m-file is used for the following purpose              *
%                            Subroutine was used for loading fine and coarse error correct from file   *
%                            for the proposed DDFS, called by errorlookuptable.m        *
/*********************************************************************************/
function [fine_value,coarse_value,fine_hex,coarse_hex] = loaderrortable(float_file,integ_file)
%---------------------------------- Load float error correct ROM table from file -------------------------------------*
sub_name1 = '_coarse';
sub_name2 = '_fine';
temp_name1 = [float_file sub_name1];
temp_name2 = [integ_file sub_name1];
temp_name3 = [float_file sub_name2];
temp_name4 = [integ_file sub_name2];
fid1 = fopen(temp_name1,'r');
fid2 = fopen(temp_name2,'r');
fid3 = fopen(temp_name3,'r');
fid4 = fopen(temp_name4,'r');
%-------------------------------------------- Load coarse error correct value    ----------------------------------------*
coarse_value = fscanf(fid1,'%f',inf);
coarse_int = fscanf(fid2,'%d',inf);
coarse_hex = dec2hex(coarse_int);
%------------------------------------------- Load fine error correct value --------------------------------------------*
fine_unm_float = fscanf(fid3,'%u %u\n',2);
fine_unm_int = fscanf(fid4,'%u %u\n',2);
temp_value = fscanf(fid3,'%f',inf);
temp_value = reshape(temp_value,fine_unm_float(2),fine_unm_float(1));
fine_value = temp_value';
temp_int = fscanf(fid4,'%d',inf);
temp_int = reshape(temp_int,fine_unm_int(2),fine_unm_int(1));
fine_int = temp_int';
```

```
fine_hex = dec2hex(fine_int');
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
```

2.Verilog code for digital modulator

(1). modulator.v

```
/*********************************************************************************/
// MODULE          :    modeset.v                                                 *
// FILE NAME       :    modeset.v                                                 *
// VERSION         :    1.0                                                       *
// DATE            :    OCT 1 2003                                                *
// AUTHOR          :    Cheng Hen Shan                                            *
// DESCRIPTION:         This module is used for setting modulation type          *
//                      That is FSK DFSK BPSK QPSK according to the input parameter  *
/*********************************************************************************/
// DEFINES
`define QUATER_PHASE_OUT_BITS    9          // amplitude out bits for quarter phase
`define PHASE_OUTPUT_BITS        10         // amplitude out bits for full phase
`define ACCUMU_BITS              20         // number of   bits for accumulator
`define PHASE_TRUN_BITS           8         // number of   bits for phase truncation
`define PHASE_INPUT_BITS         10         // number of   bits for accumulator

// MODULE
module modulator(freqoffset,mode,data,symboltime,Clock, Reset, FCW,sinout,cosout,adcclk );
input       [11:0]                      freqoffset;
input       [1:0]                       mode;
input       [1:0]                       data;
input       [7:0]                       symboltime;
input                                   Clock, Reset;
input       [`ACCUMU_BITS-1:0]          FCW;
output      [`PHASE_OUTPUT_BITS-1:0]    sinout, cosout;
output                                  adcclk;

// SIGNAL DECLARATIONS
wire        [`ACCUMU_BITS-1:0]          FCWOUT;
wire        [1:0]                        sinphasebit,cosphasebit;
wire                                     sinmsbbit,cosmsbit,loadbit;
wire                                     sinms2bit,cosms2bit;
wire        [`PHASE_OUTPUT_BITS-1:0]     sinout_wire, cosout_wire;
reg         [`PHASE_OUTPUT_BITS-1:0]     sinout, cosout;
// MAIN CODE
// Instantiate
DDFS sinddfs(
        .Clock(Clock),
        .Reset(Reset),
        .phaseoffset(sinphasebit),
        .phaseload(loadbit),
        .FCW(FCWOUT),
        .ddfs_out(sinout_wire),
        .msbbit(sinmsbbit),
        .ms2bit(sinms2bit)
         );
DDFS cosddfs(
        .Clock(Clock),
        .Reset(Reset),
        .phaseoffset(cosphasebit),
        .phaseload(loadbit),
        .FCW(FCWOUT),
        .ddfs_out(cosout_wire),
        .msbbit(cosmsbbit),
        .ms2bit(cosms2bit)
         );

Modeset modeset(
        .freqoffset(freqoffset),
        .mode(mode),
        .data(data),
        .symboltime(symboltime),
        .msbbit(sinmsbbit),
        .msb2bit(sinms2bit),
        .Clock(Clock),
        .Reset(Reset),
        .FCW(FCW),
        .load_flag_pulse(loadbit),
        .sinphaseoff(sinphasebit),
        .cosphaseoff(cosphasebit),
        .FCWOUT(FCWOUT)
         );
assign adcclk = Clock;
always    @(posedge adcclk)
begin
    cosout <= cosout_wire ;
    sinout <= sinout_wire ;
end
endmodule           // end modeset
```

(2). ddfs.v

```
/*************************************************************************************/
// MODULE          :     ddfs.v                                                      *
// FILE NAME       :     ddfs.v                                                      *
// VERSION         :     1.0                                                         *
// DATE            :     Sep 1 2003                                                  *
// AUTHOR          :     Cheng Hen Shan                                              *
// DESCRIPTION:          This module is for SCMF(sine/cosine mapping function)       *
//                       This module defines a direct digit frequency synthesizer that is used for   *
//                       generating the sin wave according to the frequency control word             *
/*************************************************************************************/
// DEFINES
`define QUATER_PHASE_OUT_BITS     9          // amplitude out bits for quarter phase
`define PHASE_OUTPUT_BITS        10          // amplitude out bits for full phase
`define ACCUMU_BITS              20          // number of   bits for accumulator
`define PHASE_TRUN_BITS           8          // number of   bits for phase truncation
`define PHASE_INPUT_BITS         10          // number of   bits for accumulator
// MODULE
module DDFS(Clock, Reset, phaseoffset, phaseload, FCW, ddfs_out,msbbit,ms2bit);
input                                       Clock, Reset;
input        [1:0]                          phaseoffset;
input                                       phaseload;
input        [`ACCUMU_BITS-1:0]             FCW;
output       [`PHASE_OUTPUT_BITS-1:0]       ddfs_out;
output                                                  msbbit,ms2bit ;   // MSB bit for phase address
// SIGNAL DECLARATIONS
wire          [`ACCUMU_BITS-1:0]                        Acc_addr_Out;
wire          [`QUATER_PHASE_OUT_BITS-1:0]              quarter_ddfs_Out;
wire          [`PHASE_OUTPUT_BITS-1:0]                  ddfs_out;
wire          [`PHASE_INPUT_BITS-1:0]                   phase_input;
// Instantiate
Accumulator Accumu (
              .Clock(Clock),
              .Reset(Reset),
              .FCW(FCW),
              .phaseload(phaseload),
              .phaseoffset(phaseoffset),
              .Acc_Out(Acc_addr_Out)
              );
PhaseToAmp amplitude (
              .phase(phase_input),
              .Amp_out(quarter_ddfs_Out)
              );
assign ddfs_out =
       (Acc_addr_Out[`ACCUMU_BITS-1]) ?{1'b1 ,~quarter_ddfs_Out}:{1'b0 ,quarter_ddfs_Out};
assign phase_input =
       (Acc_addr_Out[`ACCUMU_BITS-2]) ?~Acc_addr_Out[`ACCUMU_BITS-3:`PHASE_TRUN_BITS]:
       Acc_addr_Out[`ACCUMU_BITS-3:`PHASE_TRUN_BITS];
assign msbbit = Acc_addr_Out[`ACCUMU_BITS-1];
assign ms2bit = Acc_addr_Out[`ACCUMU_BITS-2];
endmodule          // end DDFS
```

(3). modeset.v

```
/*************************************************************************************/
// MODULE        :     modeset.v                                                     *
// FILE NAME     :     modeset.v                                                     *
// VERSION       :     1.0                                                           *
// DATE          :     OCT 1 2003                                                    *
// AUTHOR        :     Cheng Hen Shan                                                *
// DESCRIPTION:        This module is used for setting modulation                    *
//                     type that is FSK DFSK BPSK QPSK according to the input parameter  *
/*************************************************************************************/
```

```
// DEFINES
`define     QUATER_PHASE_OUT_BITS       9           // amplitude out bits for quarter phase
`define     PHASE_OUTPUT_BITS           10          // amplitude out bits for full phase
`define     ACCUMU_BITS                 20          // number of   bits for accumulator
`define     PHASE_TRUN_BITS             8           // number of   bits for phase truncation
`define     PHASE_INPUT_BITS            10          // number of   bits for accumulator
`define     FSKMODE                     2'b00       // FSK mode
`define     DFSKMODE                    2'b01       // DFSK mode
`define     BPSKMODE                    2'b10       // BPSK mode
`define     QPSKMODE                    2'b11       // QPSK mode
// MODULE
module Modeset(freqoffset,mode,data,symboltime,msbbit,msb2bit,Clock, Reset, FCW,
                 load_flag_pulse, sinphaseoff, cosphaseoff, FCWOUT );
input       [11:0]                  freqoffset;
input       [1:0]                   mode;
input       [1:0]                   data;
input       [7:0]                   symboltime;
input                               msbbit,msb2bit,Clock, Reset;
input       [`ACCUMU_BITS-1:0]      FCW;
output                              load_flag_pulse;
output [    1:0]                    sinphaseoff,cosphaseoff;
output      [`ACCUMU_BITS-1:0]      FCWOUT;
// SIGNAL DECLARATIONS
wire        [`PHASE_INPUT_BITS-1:0] phase_input;
reg         [`ACCUMU_BITS-1:0]      FCWOUT;
reg         [1:0]                   prev_data;
reg         [7:0]                   prev_symbol_time;
reg                                 load_flag;
reg         [7:0]                   count;
reg         [1:0]                   sinphaseoff,cosphaseoff;
reg                                 load_pulse,init_load,msb_pulse;
wire                                msb2_pulse_check ,delay_load;
reg                                 load_flag_pulse ,rst,set;
// MAIN CODE
always @(Reset)
begin
  rst = 0;
  set = 1;
```

```
end

always @(posedge Clock)
begin
    load_pulse <= load_flag;
    msb_pulse <= msb2bit;
    load_flag_pulse <= delay_load;

    if (Reset)    // reset
    begin
          init_load <= set;    // reset output to zero
          FCWOUT <= rst;
          count <= rst;
          load_flag <= rst;
          sinphaseoff <= rst;
          cosphaseoff <= set;
          prev_data <= rst;
    end
    else if(delay_load || init_load)
    begin
          FCWOUT <= FCW;
          case (mode)
                  `FSKMODE    :
                    begin
                             if(prev_data[0] == 1'b0)
                                      FCWOUT <= FCW;
                             else
                                      FCWOUT <= FCW + {8'b0, freqoffset};
                    sinphaseoff <= 2'b00;
                    cosphaseoff <= 2'b01;
                    end
   `DFSKMODE :
     begin
           case(prev_data)
                 2'b00 : FCWOUT <= FCW;
                 2'b01 : FCWOUT <= FCW + {8'b0, freqoffset>>8};
                 2'b10 : FCWOUT <= FCW + {8'b0, freqoffset>>3};
                 2'b11 : FCWOUT <= FCW + {8'b0, freqoffset};
    endcase
        sinphaseoff <= 2'b00;
         cosphaseoff <= 2'b01;
end
`BPSKMODE :
 begin
    if(prev_data[0] == 1'b0)
    begin
          sinphaseoff <= 2'b00;
          cosphaseoff <= 2'b01;
    end
    else
    begin
          sinphaseoff <= 2'b10;
          cosphaseoff <= 2'b11;
    end
          end
`QPSKMODE :
 begin
    case(prev_data)
        2'b00 :
        begin
            sinphaseoff <= 2'b00;
            cosphaseoff <= 2'b00;
        end
        2'b01 :
        begin
            sinphaseoff <= 2'b10;
            cosphaseoff <= 2'b00;
        end
        2'b10 :
        begin
            sinphaseoff <= 2'b00;
            cosphaseoff <= 2'b10;
        end
        2'b11 :
        begin
            sinphaseoff <= 2'b10;
```

```
           cosphaseoff <= 2'b10;
        end
     endcase
  end
   //default :
   //begin
   //end
   endcase
   init_load <= 1'b0;
 end
 else if(msb2_pulse_check)
 begin
 case (mode)
`BPSKMODE ,`QPSKMODE :
 begin
   if(prev_data[0] == 1'b0)
       if((~msbbit)&&(~msb2bit))
             begin
                 if(count == prev_symbol_time)
                 begin
                    count <= 8'b00000000;
                    load_flag <= 1'b1;
                    prev_data <= data;
                    prev_symbol_time <= symboltime;
                 end
                 else
                 begin
                     count <= count + 1;
                     load_flag <= 1'b0;
                 end
              end
              else
                 load_flag <= 1'b0;
    else
if(msbbit&&(~msb2bit))
             begin
                if(count == prev_symbol_time)
                begin
```

```
                  count <= 8'b00000000;
                  load_flag <= 1'b1;
                  prev_data <= data;
            prev_symbol_time <= symboltime;
               end
               else
               begin
                  count <= count + 1;
                  load_flag <= 1'b0;
              end
            end
            else
              load_flag <= 1'b0;
      end

      default :
      begin
         if((~msbbit)&&(~msb2bit))
         begin
            if(count == prev_symbol_time)
            begin
               count <= 8'b00000000;
               load_flag <= 1'b1;
               prev_data <= data;
         prev_symbol_time <= symboltime;
            end
            else
            begin
               count <= count + 1;
               load_flag <= 1'b0;
            end
         end
         else
            load_flag <= 1'b0;
      end
endcase
 end
else
```

```
    init_load <= 1'b0;
end
assign delay_load = load_flag & ~load_pulse;
assign msb2_pulse_check = ~msb2bit & msb_pulse;
endmodule           // end modeset
```

(4). PhaseToAmp.v

```
/*****************************************************************************************/
// MODULE         :     PhaseToAmp.v                                                     *
// FILE NAME      :     PhaseToAmp.v                                                     *
// VERSION        :     1.0                                                              *
// DATE           :     Sep 1 2003                                                       *
// AUTHOR         :     Cheng Hen Shan                                                   *
// DESCRIPTION:         This module is for phase to amplitude                            *
//                      by approximation method using coarse rom and fine rom            *
//                      to compensate the approximation value                            *
/*****************************************************************************************/
// DEFINES
`define ADDRESS_BITS                 10          // Number of addition operations
`define COARSE_OUTPUT_BITS           6           // out bits for COARSE ROM
`define COARSE_ADDRESS_BITS          6           // address bits for COARSE ROM
`define FINE_OUTPUT_BITS             3           // out bits for FINE ROM
`define AMP_OUT_BITS                 9           // Bit width of amplitude output
`define A_ADDR_BITS                  2           // number of   bits for A address
`define B_ADDR_BITS                  4           // number of   bits for B address
`define C_ADDR_BITS                  4           // number of   bits for C address
`define OUT_DIFF_BIT                 `COARSE_OUTPUT_BITS-`FINE_OUTPUT_BITS
                                                 // Difference of bits for C address
`define COARSE_START_BIT             `ADDRESS_BITS-`A_ADDR_BITS-`B_ADDR_BITS
                                                 // Difference of bits for C address
`define FINE_START_BIT          `ADDRESS_BITS-`A_ADDR_BITS
//   MODULE DEFINITION
module PhaseToAmp(Clock,phase,Amp_out);
input                                            Clock;
input         [`ADDRESS_BITS-1:0]                phase;
output        [`AMP_OUT_BITS-1:0]                Amp_out;
```

```
// SIGNAL DECLARATIONS
wire        [`COARSE_OUTPUT_BITS-1:0]       coarse_rom_out;
wire        [`FINE_OUTPUT_BITS-1:0]         fine_rom_out;
wire        [`AMP_OUT_BITS-1:0]             Amp_out;
wire        [`COARSE_OUTPUT_BITS-1:0]       Rom_out;
wire        [6:0]                           multi_out;
// Instantiate the adder
coarserom coarse_rom(
                        .address(phase[`ADDRESS_BITS-1:`COARSE_START_BIT]),
                        .q(coarse_rom_out)
                        );
  finerom fine_rom(
            .address({phase[`ADDRESS_BITS-1:`FINE_START_BIT],phase[`C_ADDR_BITS-1:0]}),
            .q(fine_rom_out)
                );
 assign Rom_out = coarse_rom_out + {3'b0,fine_rom_out};
assign multi_out[6:0] = (phase[`ADDRESS_BITS-1]) ? ~phase[9:3]:phase[9:3];
assign Amp_out =    phase[9:1]+ {2'b0,multi_out[6:0]}+{3'b0,Rom_out[5:0]};
endmodule           // end PhaseToAmp
```

(5). Accumulator.v

```
/*************************************************************************************/
// MODULE        :     Accumulato.v                                                 *
// FILE NAME     :     Accumulato.v                                                 *
// VERSION       :     1.0                                                          *
// DATE          :     Sep 1 2003                                                   *
// AUTHOR        :     Cheng Hen Shan                                               *
// DESCRIPTION:        This module defines an accumulator with synchronous clk and reset inputs.   *
//                     When the adder is synchronously reset, the outputs go to zero and proceed add  *
//                     at each clk period
/*************************************************************************************/
// DEFINES
`define ACC_BITS               20            // number of    bits for accumulator
module    Accumulator (Clock, Reset, FCW, phaseload, phaseoffset, Acc_Out);
input                                    Clock, Reset;
input         [`ACC_BITS-1:0]            FCW;
input                                    phaseload;
input       [1:0]                   phaseoffset;
output      [`ACC_BITS-1:0]         Acc_Out;
reg         [`ACC_BITS-1:0]         Acc_Out;
reg         [`ACC_BITS-1:0]         Acc_In;
reg                                 init_load;
always   @(phaseoffset)
begin
     case(phaseoffset)
        2'b00 : Acc_In <= `ACC_BITS'b0000_0000_0000_0000_0000;
        2'b01 : Acc_In <= `ACC_BITS'b0100_0000_0000_0000_0000;
        2'b10 : Acc_In <= `ACC_BITS'b1000_0000_0000_0000_0000;
        2'b11 : Acc_In <= `ACC_BITS'b1100_0000_0000_0000_0000;
     endcase
end

always   @(posedge Clock)
begin
   if (Reset)    // reset
   begin
       Acc_Out <= 0;    // reset output to zero
       init_load <= 1'b1;
   end
   else if(phaseload || init_load)
   begin
       Acc_Out <= Acc_In;
       init_load <= 1'b0;
   end
   else             // ADD input Data_In to output
       Acc_Out <= Acc_Out + FCW;
end
endmodule
```

(6). coarserom.v

```
/*************************************************************************************/
// MODULE        :    coarserom.v                                                  *
// FILE NAME     :    coarserom.v                                                  *
// VERSION       :    WM1.0                                                        *
// DATE          :    Feb 1 2004                                                   *
// AUTHOR        :    megafunction wizard: %LPM_ROM%                               *
// DESCRIPTION:       megafunction generated by Quartet software for Rom
/*************************************************************************************/
module coarserom (address, q);
 input      [5:0]           address;
 output     [5:0]           q;
 wire       [5:0]           sub_wire0;
 wire       [5:0]           q = sub_wire0[5:0];
 lpm_rom    lpm_rom_component (.address (address), .q (sub_wire0));
 defparam
         lpm_rom_component.lpm_width = 6,
         lpm_rom_component.lpm_widthad = 6,
         lpm_rom_component.lpm_address_control = "UNREGISTERED",
         lpm_rom_component.lpm_outdata = "UNREGISTERED",
         lpm_rom_component.lpm_file = "simuhex_coarse.mif";
endmodule
```

(7). finerom.v

```
/*************************************************************************************/
// MODULE        :    finerom.v                                                    *
// FILE NAME     :    finerom.v                                                    *
// VERSION       :    WM1.0                                                        *
// DATE          :    Feb 1 2004                                                   *
// AUTHOR        :    megafunction wizard: %LPM_ROM%                               *
// DESCRIPTION:       megafunction generated by Quartet software for Rom
/*************************************************************************************/
module finerom (address, q);
 input      [5:0]      address;
 output     [2:0]      q;
 wire       [2:0]      sub_wire0;
 wire         [2:0]    q = sub_wire0[2:0];
 lpm_rom    lpm_rom_component (.address (address), .q (sub_wire0));
 defparam
        lpm_rom_component.lpm_width = 3,
        lpm_rom_component.lpm_widthad = 6,
        lpm_rom_component.lpm_address_control = "UNREGISTERED",
        lpm_rom_component.lpm_outdata = "UNREGISTERED",
        lpm_rom_component.lpm_file = "simuhex_fine.mif";
endmodule
```

Autobiography

姓名	：	鄭恒杉	
出生年月日	：	1963 年 10 月 8 日	
學歷	：	高雄中學	1982 畢業
		臺灣大學機械系	1986 畢業
		臺灣大學機械研究所	1988 畢業
		交通大學電機資訊學院電子光電組	2004 畢業
經歷	：	工研院機械所	1988 – 1995
		立衛科技	1996 – 1999
		宇慶科技	1999 – 2001
		世紀民生	2001 – 2004