**Baseband Lab\_2**

**4-QAM Transmitter**

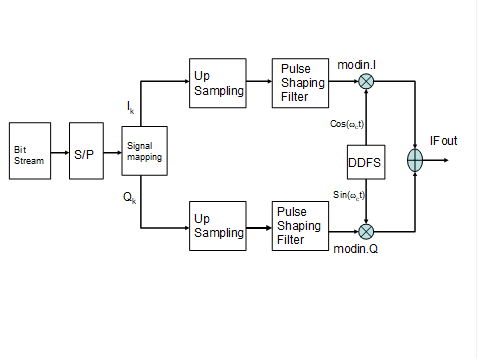
7109064351何宗諭

7110064486 林鼎鈞

7110064504 蘭松耀

7110064501 何明志

1.實驗原理 :



本實驗透過設計上圖中的submodule來完成一個4QAM的發射機，而其中的Pulse Shaping Filter將引用為Lab\_1的結果。

2. Matlab code :

clear;

bit=fix(randn(1,500)); %Generate rand 0 or 1 signal sequence and the length of signal sequence is 5000 point

bit=abs(bit);

for (i=1:length(bit))

if(bit(i)>1)

bit(i)=1;

end

end

% -----------------------------------------------------------

if(rem(length(bit),2)==0) %Part\_A S/P Converter

i1=1;

for(i=1:2:(length(bit)-1))

a1(i1,1:2)=bit(i:(i+1));

i1=i1+1;

end

else

i1=1;

for(i=1:2:length(bit)-2)

a1(i1,1:2)=bit(i:(i+1));

i1=i1+1;

end

end

for(i=1:length(a1)) %Part\_B Signal Mapping

if(a1(i,1:2)==[0 0])

a2(i,1:2)=[1 1];

elseif(a1(i,1:2)==[0 1])

a2(i,1:2)=[1 -1];

elseif(a1(i,1:2)==[1 0])

a2(i,1:2)=[-1 1];

elseif(a1(i,1:2)==[1 1])

a2(i,1:2)=[-1 -1];

end

end

% ---------------------------------------------------------------------

for(i=1:length(a2)) % Part\_C Up-sampling Interpolator

a3(i,:)=[a2(i,1) zeros(1,3) a2(i,2) zeros(1,3)];

end

intp\_num=4; % Copy Lab1 Pulse Shaping Filter

symbol\_interval=4;

r=0.5;

tap=2\*symbol\_interval\*intp\_num+1;

ti=-symbol\_interval:1/intp\_num:symbol\_interval;

for i=1:tap

t=ti(i);

if t==0

srrcc(i)=1-r+4\*r/pi;

elseif abs(t)==1/(4\*r)

srrcc(i)=r/sqrt(2)\*((1+2/pi)\*sin(pi/4/r)+(1-2/pi)\*cos(pi/4/r));

else

srrcc(i)=(sin(pi\*t\*(1-r))+4\*r\*t\*cos(pi\*t\*(1+r)))/(pi\*t\*(1-(4\*r\*t)^2));

end

end

for(i=1:length(a3)) % combinate m\_i1 and m\_q1 from matrix to a one row matrix

m\_i1(i,:)=[a3(i,1:4)];

m\_q1(i,:)=[a3(i,5:8)];

end

m\_i2=m\_i1'; % temp variable

m\_i=(m\_i2(:))'; % final we want Ik

m\_q2=m\_q1'; % temp variable

m\_q=(m\_q2(:))'; % final we want Qk

% -----------------------------------------------------------------------

modi=conv(srrcc,m\_i); % Part\_D DDFS

modq=conv(srrcc,m\_q);

txf\_now=0;

txf\_carrier\_freq=pi/2;

for(i=1:length(modi))

ifout(i)=modi(i).\*(cos(txf\_now))-modq(i).\*(sin(txf\_now));

s(i)=sin(txf\_now);

c(i)=cos(txf\_now);

txf\_now=txf\_now+txf\_carrier\_freq;

end

figure

subplot(2,1,1)

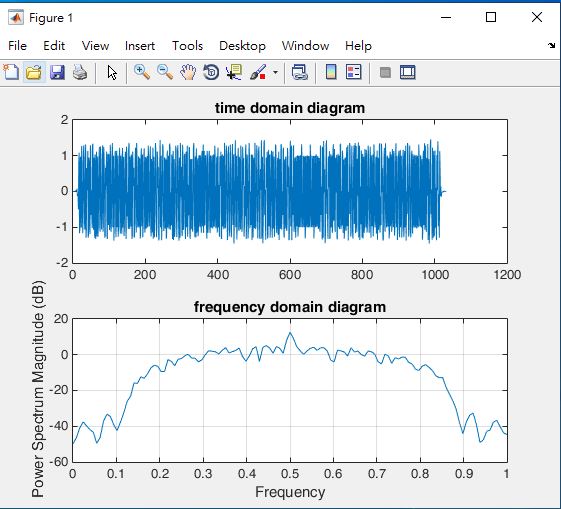
plot(ifout)

title('time domain diagram');

subplot(2,1,2)

psd(ifout)

title('frequency domain diagram');



產生的波型如上圖所示，上圖為時域的波型圖，下圖為對應的頻譜

3. Veriolg :

我們這組分別設計系統方塊圖的各個子電路，最後再透過Top module將各個submodule呼叫進來，以實現4QAM發射機的硬體實做。

以下展示各個電路的程式:

**(1) S/P Converter :**

module sp(

clk,

rst,

sp\_en,

sp\_in,

sp\_out

);

//

input clk, rst;

input sp\_in;

input sp\_en;

output reg [1:0] sp\_out;

//

reg [1:0] out\_temp;

reg flag; // asserted after first operation

reg ctrl;

// control signal

always@(posedge clk or posedge rst) begin

if(rst) begin

ctrl <= 0;

end

else if(sp\_en) begin

ctrl <= ~ctrl;

end

else begin

ctrl <= ctrl;

end

end

// parallel

always@(posedge clk or posedge rst) begin

if(rst) begin

out\_temp <= 0;

flag <= 0;

end

else if(sp\_en && ctrl == 0) begin

out\_temp <= sp\_in;

flag <= flag;

end

else if(ctrl == 1) begin

out\_temp <= out\_temp << 1;

out\_temp[0] <= sp\_in;

flag <= 1;

end

end

// output

always@(posedge clk or posedge rst) begin

if(rst) begin

sp\_out <= 0;

end

else if(ctrl == 0 && flag) begin

sp\_out <= out\_temp;

end

else begin

sp\_out <= 2'bzz;

end

end

endmodule

**(2) Signal Mapping :**

module mapping(

clk,

rst,

map\_en,

map\_in,

i\_out,

q\_out

);

input clk, rst;

input map\_en;

input [1:0] map\_in;

output [1:0] i\_out;

output [1:0] q\_out;

reg signed [1:0] i\_out;

reg signed [1:0] q\_out;

// 4\_QAM

always@(posedge clk or posedge rst) begin

if(rst) begin

i\_out <= 2'b00;

q\_out <= 2'b00;

end

else if(map\_en) begin

case(map\_in)

2'b00: begin

i\_out <= 2'b01;

q\_out <= 2'b01;

end

2'b01: begin

i\_out <= 2'b01;

q\_out <= 2'b11;

end

2'b10: begin

i\_out <= 2'b11;

q\_out <= 2'b01;

end

2'b11: begin

i\_out <= 2'b11;

q\_out <= 2'b11;

end

default: begin

i\_out <= i\_out;

q\_out <= q\_out;

end

endcase

end

else begin

i\_out <= 2'bzz;

q\_out <= 2'bzz;

end

end

endmodule

**(3) Up-sampling Interpolator :**

module up\_sample(

clk,

rst,

up\_en,

i\_in,

q\_in,

i\_up\_out,

q\_up\_out

);

input clk, rst;

input up\_en;

input signed [1:0] i\_in;

input signed [1:0] q\_in;

output [1:0] i\_up\_out;

output [1:0] q\_up\_out;

reg signed [1:0] i\_up\_out;

reg signed [1:0] q\_up\_out;

//

reg [1:0] samp\_cnt;

reg [1:0] buffer\_i, buffer\_q;

reg flag;

// sample counter

always@(posedge clk or posedge rst) begin

if(rst) begin

samp\_cnt <= 0;

end

else if(up\_en) begin

samp\_cnt <= samp\_cnt + 1;

end

else begin

samp\_cnt <= samp\_cnt;

end

end

// flag

always@(posedge clk or posedge rst) begin

if(rst) begin

flag <= 0;

end

else if(samp\_cnt == 0) begin

flag <= 1;

end

else begin

flag <= flag;

end

end

always@(posedge clk or posedge rst) begin

if(rst) begin

buffer\_i <= 0;

buffer\_q <= 0;

end

else if(samp\_cnt == 2'b10) begin

buffer\_i <= i\_in;

buffer\_q <= q\_in;

end

else begin

buffer\_i <= buffer\_i;

buffer\_q <= buffer\_q;

end

end

// output

always@(posedge clk or posedge rst) begin

if(rst) begin

i\_up\_out <= 2'b00;

q\_up\_out <= 2'b00;

end

else if(up\_en) begin

case(samp\_cnt)

2'b00: begin

if(flag == 0) begin

i\_up\_out <= i\_in;

q\_up\_out <= q\_in;

end

else begin

i\_up\_out <= buffer\_i;

q\_up\_out <= buffer\_q;

end

end

2'b01: begin

i\_up\_out <= 2'b00;

q\_up\_out <= 2'b00;

end

2'b10: begin

i\_up\_out <= 2'b00;

q\_up\_out <= 2'b00;

end

2'b11: begin

i\_up\_out <= 2'b00;

q\_up\_out <= 2'b00;

end

default: begin

i\_up\_out <= i\_up\_out;

q\_up\_out <= q\_up\_out;

end

endcase

end

else begin

i\_up\_out <= i\_up\_out;

q\_up\_out <= q\_up\_out;

end

end

endmodule

**(4) Top Module :**

module Four\_QAM(

clk,

rst,

data\_in,

data\_en,

IFout

);

input clk, rst;

input data\_en;

input data\_in;

output reg [15:0] IFout;

//

wire [1:0] sp\_out;

wire [1:0] i\_map, q\_map;

wire [1:0] i\_up, q\_up;

wire [13:0] i\_srrc\_in, q\_srrc\_in;

wire [15:0] I\_out, Q\_out;

// enable

reg map\_en;

reg up\_en;

reg ddfs\_en;

// control flag

reg [1:0] ddfs\_flag;

// process mode

reg [2:0] process;

// enable state

parameter MAP = 3'b011; // waiting three cycles for sp\_out

parameter SAMPLE = 3'b100;

parameter DDFS = 3'b101;

// DDFS state

parameter COS = 2'b00;

parameter SIN = 2'b01;

parameter N\_COS = 2'b10;

parameter N\_SIN = 2'b11;

// serial\_to\_parallel converter

sp converter(.clk(clk), .rst(rst), .sp\_in(data\_in), .sp\_en(data\_en), .sp\_out(sp\_out));

// signal\_mapping

mapping symbol(.clk(clk), .rst(rst), .map\_en(map\_en), .map\_in(sp\_out), .i\_out(i\_map), .q\_out(q\_map));

// up\_sampling

up\_sample sample(.clk(clk), .rst(rst), .up\_en(up\_en), .i\_in(i\_map), .q\_in(q\_map), .i\_up\_out(i\_up), .q\_up\_out(q\_up));

assign i\_srrc\_in = {i\_up,12'd0};

assign q\_srrc\_in = {q\_up,12'd0};

// pulse shaping filter

srrc\_csd i\_mod(.clk(clk), .rst(rst), .data\_in(i\_srrc\_in), .data\_out(I\_out));

srrc\_csd q\_mod(.clk(clk), .rst(rst), .data\_in(q\_srrc\_in), .data\_out(Q\_out));

//===========================================//

// CONTROL€UNIT //

//===========================================//

always@(posedge clk or posedge rst) begin

if(rst) begin

process <= 0;

end

else begin

process <= process + 1;

end

end

// enable signal

always@(posedge clk or posedge rst) begin

if(rst) begin

map\_en <= 0;

up\_en <= 0;

ddfs\_en <= 0;

end

else if(data\_en) begin

case(process)

MAP: begin

map\_en <= 1;

up\_en <= up\_en;

ddfs\_en <= ddfs\_en;

end

SAMPLE: begin

up\_en <= 1;

map\_en <= map\_en;

ddfs\_en <= ddfs\_en;

end

DDFS: begin

ddfs\_en <= 1;

map\_en <= map\_en;

up\_en <= up\_en;

end

default: begin

map\_en <= map\_en;

up\_en <= up\_en;

ddfs\_en <= ddfs\_en;

end

endcase

end

else begin

map\_en <= map\_en;

up\_en <= up\_en;

ddfs\_en <= ddfs\_en;

end

end

// ddfs\_flag

always@(posedge clk or posedge rst) begin

if(rst) begin

ddfs\_flag <= 0;

end

else if(ddfs\_en == 1) begin

ddfs\_flag <= ddfs\_flag;

end

end

always@(posedge clk or posedge rst) begin

if (rst) begin

IFout <= 0;

end

else begin

case(ddfs\_flag)

COS: begin

IFout <= I\_out;

end

SIN: begin

IFout <= Q\_out;

end

N\_COS: begin

IFout <= ~(I\_out) + 1;

end

N\_SIN: begin

IFout <= ~(Q\_out) + 1;

end

default: begin

IFout <= IFout;

end

endcase

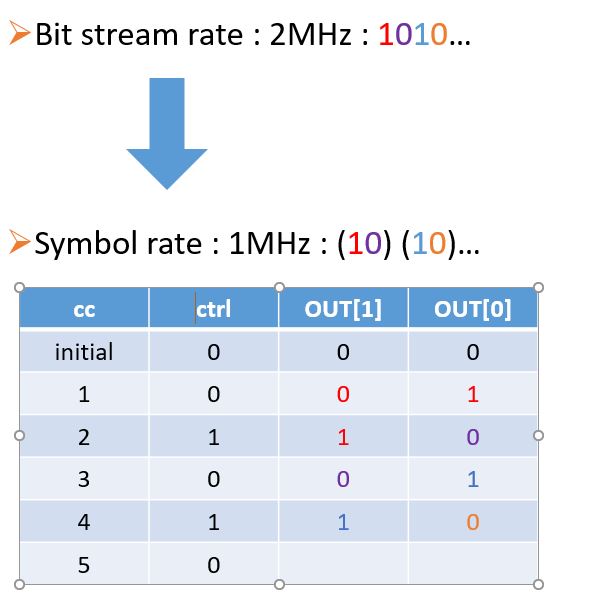
end

end

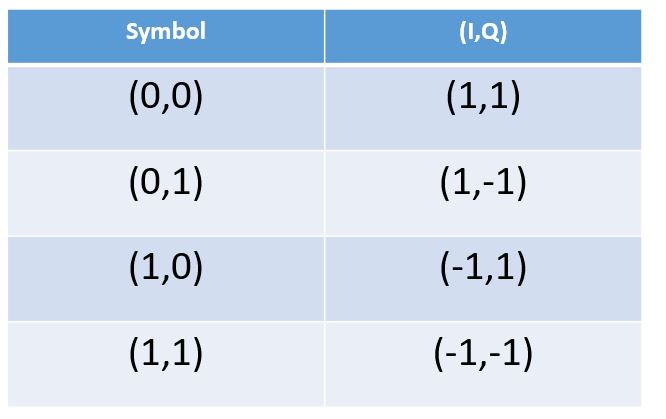
endmodule

4. 電路功能描述 :

(1) sp.v : 利用ctrl訊號來控制data\_in移位以及輸出，ctrl訊號一個clock cycle會變一次。



(2) mapping.v : 利用verilog中的case語法，製作多工器來決定星座圖上的對應輸出。



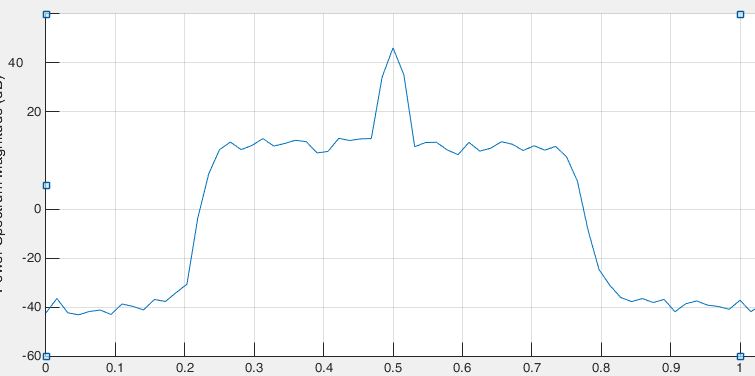
(3) up\_sample.v : 設計一個計數器，並透過此計數器來控制upsample。此計數器為2 bits的計數器，故每四個clock cycle會循環一次，當計數器在第一個clock cycle時會輸出由(2)送進來的訊號，而在第2~4的clock cycle會輸出0，藉此來達到提升4倍sample rate的效果。

(4) Four\_QAM.v :

此段程式會呼叫前面所設計子程式，並且執行DDFS block。

我們在主程式的後半部設計一個control unit，來控制其中的子程式的運作，例如第二段子程式會等待第一段子程式執行完才被觸發，以此類推，來確保輸出結果是正確的。

5. FPGA :



上圖為FPGA模擬的結果

可以看出在0.5的附近有一段突起。

**參考資料:**

1.范志鵬老師,“基頻通訊積體電路設計與實驗＂教材, 國立

中興大學 電機系。