Appendice E

Listati Matlab

E.1 NCO

Listato E.1.1: NCO_Q.m

```
% DESCRIPTION : Implementa un NCO quantizzato, ne grafica i risultati sia nel
%               dominio del tempo che della frequenza e valuta l'SFDR

% Pulizia ambiente Matlab
clear all     ;
close all     ;
clc           ;
% Imposto la frequenza di clock e la frequenza richiesta in uscita dall'NCO
[f_out , f_clk] = imposta_frequenze_Q ;
% Imposto le caratteristiche dell'NCO
[n_bit_acc, n_bit_cordic, n_bit_dac, n_it_cordic, n_periods] = imposta_NCO_Q ;
% creo l'accumulatore il quale contiene i soli valori delle fasi che elaborati
%  alla frequenza di clock generano in uscita la frequenza desiderata
super_acc = crea_super_accumulatore_Q(f_out , f_clk , n_bit_acc, n_periods) ;
% le fasi vengono troncate da una rappresentazione con n_bit_acc bits ad una
%  rappresentazione con n_bit_cordic bits
super_acc_T = tronca_Q(super_acc , n_bit_cordic) ;
% vengono generati tramite l'algoritmo CORDIC i valori del seno e del coseno
% corrispondenti agli angoli presenti in super_acc
[coseno , seno] = crea_coseno_e_seno_Q(super_acc_T, n_bit_cordic, ...
                                       n_bit_dac, n_it_cordic) ;
% grafico un periodo del seno e del coseno calcolati con il CORDIC
figure ;
subplot(2,1,1); stairs(super_acc , coseno) ;
xlim( [ 0 , 6.2832 ] ); ylim( [ -1.01 , +1.01 ] );
subplot(2,1,2); stairs(super_acc , seno );
xlim( [ 0 , 6.2832 ] ); ylim( [ -1.01 , +1.01 ] );
% Rappresentazione nel dominio della frequenza del coseno generato dal CORDIC
visualizza_spettro_e_sfdr_p( coseno , f_clk );
% salva il coseno nel file coseno.dat
fid = fopen('coseno_Q.dat','w') ;
fprintf(fid,'%20.19f\n', coseno) ;
fclose(fid)                      ;
% salva il seno nel file seno.dat
fid = fopen('seno_Q.dat','w') ;
fprintf(fid,'%20.19f\n', seno) ;
fclose(fid)                    ;
```

Listato E.1.2: Imposta_frequenze_Q.m

```
% DESCRIPTION : consente d'impostare frequenza di clock e d'uscita dall'NCO

function [f_out,f_clk]=imposta_frequenze_Q

prompt = {'Frequenza desiderata in uscita (Hz) :',
          'Frequenza di clock (Hz)       :'};
title  = 'Caratteristiche NCO';
lines  = 1;
def    = {'41.25e6','165e6'};
```

```
answer = inputdlg(prompt,title,lines,def) ;
f_out  = str2double(answer(1));
f_clk  = str2double(answer(2));
```

Listato E.1.3: Imposta_NCO_Q.m

```
% DESCRIPTION : consente d'impostare le dimensioni dei bus dell'accumulatore,
%               del Cordic e del DAC

function [n_bit_acc,n_bit_cordic,n_bit_dac,n_it_cordic,n_periods]=imposta_NCO_Q

prompt     = {'N° di bit accumulatore di fase : ',
              'N° di bit per la fase in ingresso al Cordic : ',
              'N° di bit per seno e coseno prodotti dal Cordic : ',
              'N° di iterazioni del Cordic : ',
              'N° di periodi della funzione generata : '};
title      = 'Caratteristiche NCO';
lines      = 1;
def        = {'32','13','12','13','3000'};
answer     = inputdlg(prompt,title,lines,def) ;
% n° di bit in ingresso all'accumulatore di fase
n_bit_acc  = str2double(answer(1));
% n° di bit per la fase in ingresso al Cordic
n_bit_cordic = str2double(answer(2));
% n° di bit per seno e coseno prodotti dal Cordic
n_bit_dac  = str2double(answer(3));
% n° di iterazioni del Cordic
n_it_cordic = str2double(answer(4));
% n° di periodi generati
n_periods  = str2double(answer(5));
```

Listato E.1.4: Crea_super_accumulatore_Q.m

```
% DESCRIPTION : implementa un accumulatore contenente le fasi che lette
%               alla velocità del clock producono in uscita dal Cordic
%               la frequenza desiderata, può memorizzare più periodi

function super_acc = crea_super_accumulatore_Q(f_out,f_clk,n_bit_acc,n_periods)

% caratteristiche dell'accumulatore di fase
% massimo numero di fasi memorizzabili nell'accumulatore
range_acc = 2^n_bit_acc   ;
% minima frequenza generabile
f_res    = f_clk / range_acc ;
% minima fase memorizzabile
phi_res  = 2*pi / range_acc ;
% determina l'incremento di fase necessario ad ottenere f_out
f_word   = f_out / f_res  ;
% incremento di fase che alla frequenza di clock produce in uscita la
%   frequenza desiderata
delta_phi = f_word * phi_res ;
% dimensioni dell'accumulatore contenente 1 periodo con le sole fasi che danno
%  luogo alla corretta frequenza generata
dim_acc     = round(range_acc/f_word) ;
% massimo numero di fasi memorizzabili nel superaccumulatore
dim_super_acc = dim_acc * n_periods ;
% creazione del superaccumulatore
super_acc = 0 : 1 : dim_super_acc ;
super_acc = delta_phi * super_acc ;
```

Listato E.1.5: Tronca_Q.m

```
% DESCRIPTION : le fasi vengono troncate da una rappresentazione con
%               n_bit_acc bits ad una rappresentazione con n_bit_cordic bits

function super_acc_T = tronca_Q(super_acc , n_bit_cordic)

super_acc = rem(super_acc , 2*pi) ;
new_risoluzione_fase = 2*pi / 2^n_bit_cordic ;
super_acc_T = [fix(super_acc / new_risoluzione_fase)] * new_risoluzione_fase ;
```

Listato E.1.6: Crea_coseno_e_seno_Q.m

```
% DESCRIPTION : imposta le caratteristiche del CORDIC e calcola seno e
%               coseno per tutte le fasi presenti nel vettore super_acc

function [coseno , seno] = crea_coseno_e_seno_Q(super_acc_T, n_bit_cordic,...
                                                n_bit_dac, n_it)

% ottengo le dimensioni del super_accumulatore
dim_super_acc_T = size(super_acc_T , 2)     ;

% queste inizializzazioni velocizzano l'esecuzione del ciclo CORDIC
coseno = zeros(1,dim_super_acc_T) ;
seno  = zeros(1,dim_super_acc_T)  ;

new_capacity_fase  = 2^(n_bit_cordic - 1) ;
new_capacity_ampiezza = 2^(n_bit_dac - 1) ;

for j = 1 : dim_super_acc_T
 [coseno(j),seno(j)] = forward_cordic_Q(n_it,super_acc_T(j), ...
                     new_capacity_fase, new_capacity_ampiezza);
end ;
```

Listato E.1.7: Forward_cordic_Q.m

```
% DESCRIPTION : esegue N iterazioni dell'algoritmo Cordic per calcolare il
%               seno ed il coseno dell'angolo Theta che riceve in ingresso,
%               si parte dal valore 0.607252936517011 per il coseno in modo
%               da evitare la moltiplicazione finale ed evitare l'utilizzo
%               di un bit per il solo valore 1 del coseno che del seno.

% m=1 individua le coordinate circolari mentre Nit è il n° di iterazioni
function [coseno, seno] = forward_cordic(N_it, Theta, new_capacity_fase,...
                                         new_capacity_ampiezza);

  % per semplificare il CORDIC riportiamo l'angolo nel primo quadrante,
  % l'angolo effettivo potrà poi essere ricavato mediante multcos e multsin
  multcos = 1;
  multsin = 1;
  if ( Theta > pi/2 & Theta < pi)
     % l'angolo viene riportato dal 2° quadrante al 1°
     Theta  = pi/2 - rem(Theta,pi/2);
     multcos = -1;
  end
  if (Theta >= pi & Theta < (3/2)*pi)
     % l'angolo viene riportato dal 3° quadrante al 1°
     Theta  = rem(Theta,pi/2);
     multcos = -1;
     multsin = -1;
  end
  if (Theta >= (3/2)*pi & Theta < 2*pi)
     % l'angolo viene riportato dal 4° quadrante al 1°
     Theta  = pi/2 - rem(Theta,pi/2);
     multsin = -1;
  end
  % x ed y sono vettori aventi lunghezza pari al n° di iterazioni, contengono
  %  il coseno ed il seno dell'angolo inizialmente è memorizzato in z_attuale
  x_attuale = [floor(0.607252936517011* new_capacity_ampiezza)] / ...
                                         new_capacity_ampiezza ;
  y_attuale = 0      ;
  z_attuale = Theta  ;
  % calcola il seno ed il coseno mediante N_it iterazioni dell'algoritmo CORDIC
  for i = 0:(N_it-1)
     % calcola il fattore moltiplicativo mu per la p_esima iterazione
     if z_attuale <= 0
        mu = -1;
     else
        mu =  1;
     end
     S = 2^(-i) ;
     % Standard Cordic iteration, m = 1 per le coordinate circolari
     x_succ = x_attuale - [floor((mu * S * y_attuale) * ...
                          new_capacity_ampiezza)] / new_capacity_ampiezza;
     y_succ = y_attuale + [floor((mu * S * x_attuale) * ...
                          new_capacity_ampiezza)] / new_capacity_ampiezza;
     z_succ = z_attuale - [floor((mu * atan( S ))* ...
                          new_capacity_fase)] / new_capacity_fase;
     x_attuale = x_succ ; y_attuale = y_succ ;    z_attuale = z_succ ;
  end
  % Vengono restituiti il coseno ed il seno opportunamente scalati e riportati
  %   nelal giusto quadrante
  % new_capacity_ampiezza = 2^(12 - 1) ;
  coseno = [floor((multcos * x_attuale)*new_capacity_ampiezza)]/...
                                         new_capacity_ampiezza;
  seno  = [floor((multsin * y_attuale)*new_capacity_ampiezza)]/...
                                         new_capacity_ampiezza;
```

Listato E.1.8: Calcola_rotazioni_Cordic.m

```
% DESCRIPTION : Per ogni iterazione del Cordic calcola l'angolo di rotazione
%                e il fattore di scaling

% Pulizia ambiente Matlab
clear all     ;
close all     ;
clc           ;

disp('Valore degli angoli di rotazione a(n)') ;
% Calcolo dei valori degli a(n)
for i = 0:1:12
    a(i+1) = atan(2^(-i));
end ;
i = [1:1:13] ;
valore = [ i' , ((a)* 180 / pi)' ]
a_tot = sum((a)* 180 / pi)

% converto ora gli a(i) in complemento a due su 1 bit per il segno e dodici
% per la parte intera
n_bit = 13 ;
a_bin = dec2bin( (2^n_bit .* a(i) )/ (2*pi) )

disp('Valore dei fattori di scaling k(n)') ;
% Calcolo il valore degli scalamenti da effettuare ad ogni rotazione
k = sqrt(2) ;
valore = [ 0 , k]
for i = 1:1:13
    k = k*sqrt(1 + 2^(-2*i)) ;
    valore = [i , k]
end ;
```

Listato E.1.9: Visualizza_spettro_e_SFDR_p.m

```
% DESCRIPTION : plotta la stima dello spettro e l'SFDR

function visualizza_spettro_e_SFDR_p( segnale , f_clk );

% ottengo le dimensioni del vettore segnale
dim_segnale = ( size(segnale , 2) )- 1 ;
% viene calcolata e plottata la densità spettrale di potenza
[Pxx , f] = pmtm(segnale , 8 , 4096 , f_clk) ;
Pxx_dB = 10*log10(Pxx) ;
figure;          plot(f , Pxx_dB - max(Pxx_dB) );
grid  ;          xlim([0 f_clk/2]) ;
title('Power spectral density MultiTaper');
xlabel('Frequency (Hz)'); ylabel('dB / Hz');
% viene calcolato e visualizzato il valore dello SFDR
valore_sfdr = calcola_sfdr(Pxx) ;
text(1e7,-50,['SFDR = ',num2str(valore_sfdr)],'FontSize',13);
```

Listato E.1.10: Calcola_SFDR.m

```
% DESCRIPTION : calcola l'SFDR del vettore passato in ingresso

function [valore_sfdr] = calcola_sfdr(Pxx)
% diff prende il vettore Pxx e costruisce un vettore i cui elementi sono
% la differenza tra elementi adiacenti del vettore Pxx, si ottiene quindi
% un vettore più corto di uno.
der = diff(Pxx);
% dal vettore precedente ottengo il vettore dei segni corrispondenti
signs=sign(der);
% viene inizializzato il ciclo che individua i massimi della PSD
signold = 0 ;    % contiene il valore del segno precedente
signnew = 0 ;    % contiene il valore del segno attuale
p = 1       ;    % puntatore per riempire il vettore dei massimi
% Siamo in presenza di un massimo se il segno della derivata nel punto
% precedente è negativo mentre il segno della derivata nel punto attuale
% è positivo, viene scandito il vettore dei segni alla ricerca di questa
% condizione e solo se verificata il massimo viene memorizzato
for k=1:(length(signs)-1)
    signold = signs(k) ;
```

```
20      signew = signs(k+1) ;
21      % individua la presenza di un massimo ed eventualmente lo memorizza
22      if (signold == 1 & signew == -1)
23        vettore_massimi(p) = Pxx(k+1) ;
24        p=p+1       ;
25      end
26    end
27    % ottengo posizione e valore del massimo assoluto
28    [max_assoluto , pos] = max(vettore_massimi);
29    % annullo il valore del massimo assoluto in modo da poter ricercare il
30    % primo massimo relativo
31    vettore_massimi(1 , pos) = 0;
32    % ottengo il valore del 1° massimo relativo
33    primo_max_relativo = max(vettore_massimi) ;
34    % restituisco l'SFDR
35    valore_sfdr = 10.*log10(max_assoluto)-10.*log10(primo_max_relativo);
```

Listato E.1.11: Calcola_incremento_fase_NCO.m

```
1   % DESCRIPTION : calcola la Frequency Word dell'NCO in 3 formati diversi
2
3   % Pulizia ambiente Matlab
4   clear all     ;
5   close all     ;
6   clc           ;
7
8   f_clk         = 165e6  ;
9   f_out         = 41.25e6 ;
10  n_bit_acc     = 32      ;
11  n_bit_cordic  = 13      ;
12
13  fw = round( (f_out * 2^n_bit_acc) / f_clk )
14
15  fw_hex = dec2hex(fw)
16  fw_bin = dec2bin(fw)
```

Listato E.1.12: Visualizza_spettro_NCO_VHDL.m

```
1   % DESCRIPTION : visualizza lo spettro del seno prodotto dall'NCO VHDL
2
3   % Pulizia ambiente Matlab
4   clear all     ;
5   close all     ;
6   clc           ;
7   % Imposto la frequenza di clock del sistema e quella prodotta dall'NCO
8   f_out = 40e6 ;
9   f_clk = 165e6 ;
10  % carica la sequenza filtrata dal file data_out_SRRC_I.dat
11  fid = fopen('sine_to_matlab.dat' , 'r') ;
12  seno_vhdl_dec = fscanf(fid,'%f')      ;
13  fclose(fid)                           ;
14  n_bit_dopo_virgola = 11                         ;
15  seno_vhdl = seno_vhdl_dec .* 2^(- n_bit_dopo_virgola) ;
16  % Rappresentazione nel dominio della frequenza del coseno generato dall'NCO
17  visualizza_spettro_e_sfdr_p( seno_vhdl , f_clk );
```

E.2 Creazione vettori di test

Listato E.2.1: CreaVettoriTest.m

```
1   % DESCRIPTION : Crea dei vettori NRZ e RZ da applicare al VHDL
2
3   % Pulizia ambiente Matlab
4   clear all ;
5   close all ;
6   clc       ;
7   % vengono impostate le caratteristiche del polifase
8   prompt    = {'N° bit Reset attivo :',
9                'Interpolazione :',
10               'Nome del file sorgente :' };
11  title     = 'Impostazione delle caratteristiche dei segnali di test';
```

```
lines     = 1;
def       = {'5' , '4', 'bit_tx.dat' };
answer    = inputdlg(prompt,title,lines,def) ;
n_bit_reset_alto = str2double(answer(1));
interp_rate      = str2double(answer(2));
nome_file_sorgente = char(answer(3));
% legge da un file i bit randomici da trasmettere
fid    = fopen( nome_file_sorgente , 'r') ;
bit_tx = fscanf(fid,'%f')              ;
fclose(fid)                            ;
% determinazione del numero di bit di test da applicare
n_bits2tx = length(bit_tx) ;
% creazione del segnale di reset
n_bit_dummy = 4*n_bit_reset_alto ;
reset = zeros(n_bits2tx/2, 1)      ;
% vengono aggiunti 10 bit 1 all'inizio del segnale di reset più altri 10
reset =[ones(1,n_bit_reset_alto) zeros(1,n_bit_dummy/4) reset']';
% salva su file il vettore di test RZ per la componente Q
fid = fopen('reset.dat','w') ;
fprintf(fid,'%d\n', reset) ;
fclose(fid)                ;
% vengono aggiunti dei bit di inizializzazione alla sequenza da trasmettere
bit_tx =[zeros(1,n_bit_dummy) bit_tx']';
% viene creato il vettore dei bit pari e quello dei bit dispari
bit_I = bit_tx(1 : 2 : n_bits2tx + n_bit_dummy)     ;
bit_Q = bit_tx(2 : 2 : n_bits2tx + n_bit_dummy)     ;
% salva su file il vettore di test RZ per la componente I
fid = fopen('data_in_SRRCxN_tx_I_rz.dat','w') ;
fprintf(fid,'%d\n', bit_I)                    ;
fclose(fid)                                   ;
% salva su file il vettore di test RZ per la componente Q
fid = fopen('data_in_SRRCxN_tx_Q_rz.dat','w') ;
fprintf(fid,'%d\n', bit_Q)                    ;
fclose(fid)                                   ;
% i bit vengono opportunamente mappati
data_in_SRRC_tx_I = -2*bit_I + 1              ;
data_in_SRRC_tx_Q = -2*bit_Q + 1              ;
% salva su file il vettore di test RZ per la componente I
fid = fopen('data_in_SRRCxN_tx_I_nrz.dat','w') ;
fprintf(fid,'%+d\n', data_in_SRRC_tx_I)   ;
fclose(fid)                               ;
% salva su file il vettore di test RZ per la componente Q
fid = fopen('data_in_SRRCxN_tx_Q_nrz.dat','w') ;
fprintf(fid,'%+d\n', data_in_SRRC_tx_Q)   ;
fclose(fid)                               ;
% si tiene conto del fatto che rate_sel è codificato su due bit
switch interp_rate
    case 3
        rate_sel_coded = 0 ;
    case 4
        rate_sel_coded = 1 ;
    case 6
        rate_sel_coded = 2 ;
    otherwise
            error('Impossible !!! ')
end
% creazione del segnale rate_sel per l'impostazione del data_rate
rate_sel = rate_sel_coded * ones(length(data_in_SRRC_tx_Q) , 1) ;
fid = fopen(strcat('rate_sel_x', num2str(interp_rate), '.dat'),'w');
fprintf(fid,'%+d\n', rate_sel) ;
fclose(fid) ;   disp('Vettori di test generati correttamente.') ;
```

Listato E.2.2: CreaSequenzaPatternGenerator.m

```
% DESCRIPTION : Ha in ingresso i file con le sequenze da applicare ai
%               rispettivi canali del pattern generator

% Pulizia ambiente Matlab
clear all ;
close all ;
clc       ;
fclose('all');
% vengono impostate le caratteristiche del polifase
prompt    = {'Tipo di clock (int , ext) :',
             'N° di segnali :',
             'N° di bit nella sequenza d''inizializzazione :',
             'Nome file di test da produrre :' };
title     = 'Caratteristiche dei vettori di test da applicare all''FPGA';
lines     = 1;
def       = {'ext' , '3', '10' , 'I_testVector_x3.dat'};
answer    = inputdlg(prompt,title,lines,def) ;
tipo_clock = char(answer(1));
n_segnali  = str2double(answer(2));
n_bit_start = str2double(answer(3));
nome_file_test = char(answer(4));
```

```
22  switch lower(tipo_clock)
23     case 'int',
24        % viene richiesto di impostare il valore del clock interno
25        answer    = inputdlg( {'frequenza del clock interno :'}, ...
26           'Impostazione del clock interno del Pattern Generator',1,{'40'}) ;
27        f_clk     = str2double(answer(1))*10^6;
28        clk_string = ['FORMAT:CLOCK INTERNAL,' num2str((1/f_clk)*10^9)'E-9'];
29     case 'ext',
30        % viene richiesto di impostare il range del clock esterno
31        clk_ext = questdlg('Selezionare range clock al Pattern Generator:',...
32                  'Selezione clock esterno', ...
33                  'f_clk_ext < 50MHz','50MHz < f_clk_ext < 100MHz', ...
34                  'f_clk_ext > 100MHz','f_clk_ext < 50MHz');
35        % viene ricavata la stringa da settare per il range esterno selezionato
36        switch clk_ext
37           case 'f_clk_ext < 50MHz', ...
38                 clk_string = 'FORMAT:CLOCK EXTERNAL, LEFifty';
39           case '50MHz < f_clk_ext < 100MHz', ...
40                 clk_string = 'FORMAT:CLOCK EXTERNAL, GTFifty';
41           case 'f_clk_ext < 50MHz', ...
42                 clk_string = 'FORMAT:CLOCK EXTERNAL, GTONe';
43        end
44  end
45  % si itera per acquisire i valori da assegnare ai diversi canali
46  for s = 1 : 1 : n_segnali
47     prompt    = {'Nome del segnale :',
48                  'N° di bit associati (multiplo di 4) :',
49                  'Nome file sorgente :' };
50     title     = ['Caratteristiche del segnale ' num2str(s) ];
51     lines     = 1;
52     def       = {'data_in_SRRCxN_tx_I','4','data_in_SRRCxN_tx_I_rz.dat'};
53     answer    = inputdlg(prompt,title,lines,def) ;
54     label(s)  = answer(1);
55     n_bit_label(s) = str2double(answer(2));
56     eing_file(s) = answer(3);
57  end
58  % vengono caricati da file i segnali di test da applicare
59  for s = 1 : n_segnali
60     fid_in(s)   = fopen( char(eing_file(s)) , 'r') ;
61     segnale(:,s) = fscanf(fid_in(s),'%d')     ;
62  end
63  % viene calcolata la lunghezza della stringa dummy costituita da tutti 0
64  n_cifreHEX_dummy = num2str( (40 - sum(n_bit_label) )/ 4 );
65  % scrivo su file le stringhe esadecimali calcolate
66  fid_out = fopen([tipo_clock 'Clk_' nome_file_test ],'w');
67  % aggiungo al file l'intestazione
68  fprintf(fid_out,'%s\n', 'ASCII 000000') ;
69  fprintf(fid_out,'%s\n', 'ASCDOWN') ;
70  fprintf(fid_out,'%s\n', 'FORMAT:MODE FULL') ;
71  fprintf(fid_out,'%s\n' , clk_string );
72  % vengono aggiunte le label
73  fprintf(fid_out,'%s', 'LABEL dummy, ') ;
74  fprintf(fid_out,'%s\n', num2str(40 - sum(n_bit_label) ));
75  for s = 1 : n_segnali
76     fprintf(fid_out,'%s', 'LABEL ') ;
77     fprintf(fid_out,'%s', char(label(s)) );
78     fprintf(fid_out,'%s', ', ') ;
79     fprintf(fid_out,'%s\n', num2str( n_bit_label(s) ));
80  end
81  fprintf(fid_out,'%s\n', 'VECTOR') ;
82  % si itera su tutta la lunghezza della sequenza d'inizializzazione
83  for row = 1 : 1 : n_bit_start
84     % viene stampata una riga del vettore di test finale
85     fprintf(fid_out,strcat('%0', n_cifreHEX_dummy ,'d'), 0 );
86     for s = 1 : n_segnali
87        fprintf(fid_out,'%s', ' ') ;
88        fprintf(fid_out, strcat('%0', num2str(n_bit_label(s) / 4 ), 'X'), ...
89                 segnale(row,s) );
90     end
91     fprintf(fid_out,'%s\n', '') ;
92  end
93  fprintf(fid_out,'%s\n', '*M') ;
94  % si itera su tutta la lunghezza dei vettori di test d'ingresso
95  for row = n_bit_start + 1 : 1 : length(segnale)
96     % viene stampata una riga del vettore di test finale
97     fprintf(fid_out,strcat('%0', n_cifreHEX_dummy ,'d'), 0 );
98     for s = 1 : n_segnali
99        fprintf(fid_out,'%s', ' ') ;
100       fprintf(fid_out, strcat('%0', num2str(n_bit_label(s) / 4 ), 'X'), ...
101                segnale(row,s) );
102    end
103    fprintf(fid_out,'%s\n', '') ;
104 end
105 % chiusura dei file contenenti i segnali di test da applicare
106 for s = 1 : n_segnali
107    fclose(fid_in(s));
108 end
109 % chiusura del file di test da applicare alla FPGA
110 fclose(fid_out);
111 % segnalazione di fine creazione del file
112 sound(wavread('drumroll.wav'));
```

```
113 disp('Vettore di test generato correttamente.')
```

E.3 Polifase

Creazione coefficienti

Listato E.3.1: CreaCoeffsFreqSamplScaled.m

```
% DESCRIPTION : calcola e confronta l'SRRC Matlab col FreqSampl e
%               condizionatamente scala i coefficienti

% pulizia ambiente Matlab
clear all ;
close all ;
clc       ;
% vengono impostate le caratteristiche del polifase
prompt           = {'Frequenza di clock :',
                    'Data Rate :',
                    'Roll Off :',
                    'N° coefficienti del FIR desiderati :',
                    'N° di bit dopo la virgola :',
                    'Scaling :' };
title            = 'Caratteristiche del polifase';
def              = {'165', '110', '0.35', '19', '11', 'Si'};
answer           = inputdlg(prompt, title, 1, def) ;
f_clk            = str2double(answer(1))*10^6;
symbol_rate      = str2double(answer(2))*10^6 / 2;
roll_off         = str2double(answer(3));
n_coeffs         = str2double(answer(4));
n_bit_dopo_virgola = str2double(answer(5));
Scaling          = answer(6);
% calcolo della interpolazione richiesta
SpS              = f_clk/symbol_rate ;
% carica da file la sequenza dei dati da filtrare
fid = fopen('data_in_SRRCxN_tx_I_nrz.dat' , 'r') ;
data_in_SRRCxN_tx_I = fscanf(fid,'%f')    ;
fclose(fid)                               ;

% ***************************** MATLAB SRRC *********************************
delay            = 3          ;
input_rate_SRRC  = symbol_rate ;
output_rate_SRRC = f_clk       ;
num_fir = rcosine(input_rate_SRRC,output_rate_SRRC,'fir/sqrt',roll_off,delay);
% viene filtrata la sequenza d'ingresso
data_out_SRRC_matlab = applica_polifase(data_in_SRRCxN_tx_I, SpS , num_fir);
% Power Spectral Density plotting
[Pyy_matlab , f_out] = pwelch(data_out_SRRC_matlab, [] , [] , 'onesided', ...
    length(data_out_SRRC_matlab) - 1 , f_clk) ;
Pyy_matlab_dB = 10*log10(Pyy_matlab) ;
figure ;          plot(f_out , Pyy_matlab_dB - max(Pyy_matlab_dB) );
grid   ;          xlim([0 f_clk/2]) ;    ylim([-70 0]);
legend(['SRRCx' num2str(SpS) ' Matlab ' [num2str(symbol_rate/10^6)] ' MSpS']);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');
figure ; impz(num_fir) ;

%  ***************************** FreqSampl SRRC ****************************
Ts      = 1/symbol_rate ; % tempo di simbolo
% risposta in frequenza nell'origine del filtro a radice di coseno rialzato
H(1) = sqrt(RaisedCosineResponse(0,roll_off,Ts));
% per la simmetria occorrono solo N/2 campioni della risposta in frequenza
for k = 1 : (n_coeffs - 1)/2,
    H(k + 1) = sqrt(RaisedCosineResponse(k*f_clk/n_coeffs, roll_off, Ts));
    H(n_coeffs - k + 1) = H(k + 1);
end
% mediante la trasformata di Fourier discreta inversa IDFT viene calcolata la
%  risposta all'impulso e traslata per garantire la causalità
for n = (-(n_coeffs - 1)/2) : ((n_coeffs - 1)/2)
    num_fir_FreqSampl(n + ((n_coeffs - 1)/2) +1) = H(0 + 1);
    for m = 1 : (n_coeffs - 1)/2,
        num_fir_FreqSampl(n + ((n_coeffs - 1)/2) +1) = ...
            num_fir_FreqSampl(n + ((n_coeffs-1)/2)+1) + ...
                2*H(m + 1)*cos(2*pi*m*n / n_coeffs);
    end
end
% calcolo il fattore di scaling e l'aggiunta al nome del file dei coefficienti
if strcmp(Scaling, 'No')
    % i coefficienti non scalati vengono salvati nel file
    fid    = fopen(strcat('SRRCx' , num2str(SpS) , '_FreqSampl.dat') , 'w');
    fprintf(fid,'%20.19f\n', num_fir_FreqSampl) ;
```

```
72      fclose(fid)                     ;
73      % la sequenza dati è applicata all'SRRC progettato campionando in frequenza
74      data_out_SRRC_FreqSampl = ...
75            applica_polifase(data_in_SRRCxN_tx_I, SpS , num_fir_FreqSampl);
76      % Power Spectral Density plotting
77      [Pyy_FreqSampl , f_out] = pwelch(data_out_SRRC_FreqSampl, [] , [] , ...
78                  'onesided', length(data_out_SRRC_FreqSampl) - 1 , f_clk) ;
79      Pyy_FreqSampl_dB = 10*log10(Pyy_FreqSampl) ;
80      figure ;       plot(f_out , Pyy_FreqSampl_dB - max(Pyy_FreqSampl_dB) );
81      grid  ;        xlim([0 f_clk/2]) ;    ylim([-70 0]);
82      legend(['SRRCx' num2str(SpS) ' FreqSampl ' ...
83                      [num2str(symbol_rate/10^6)] 'MSpS']);
84      xlabel('Frequency (Hz)'); ylabel('dB / Hz');
85      figure ; impz(num_fir_FreqSampl) ;
86  else
87      % n° di FIR che effettivamente costituiscono il polifase
88      n_fir       = SpS                       ;
89      % lunghezza massima dei FIR costituenti il polifase
90      max_dim_fir  = ceil( n_coeffs / n_fir) ;
91      % creo una matrice che in ogni riga ha i coefficienti
92      %  di uno dei rami del filtro SRRC polifase
93      % la inizializzo per velocizzare il riempimento
94      matrice_dei_fir = zeros(n_fir, max_dim_fir) ;
95      for i = 1 : n_fir           % riempio una riga alla volta
96          % estraggo i coefficenti dell'i_esimo fir
97          fir_i = num_fir_FreqSampl(i : n_fir : n_coeffs) ;
98          %figure ; freqz(fir_i) ; figure ; impz(fir_i) ;
99          % li copio nella matrice dei fir
100         matrice_dei_fir(i , 1:length(fir_i) )= fir_i ;
101     end
102     % sommo i moduli dei coefficienti di ogni FIR
103     for i = 1 : n_fir
104         modulo_i = sum(abs(matrice_dei_fir(i,:)));
105         vettore_moduli(i) = modulo_i ;
106     end
107     % calcolo il fattore di scaling
108     scaling_factor       = max(vettore_moduli) / 0.95 ;
109     % viene applicato il fattore di scaling
110     num_fir_FreqSampl_scaled = num_fir_FreqSampl / scaling_factor ;
111     % i coefficienti scalati vengono salvati nel file
112     fid    = fopen(strcat('SRRCx',num2str(SpS),'_FreqSampl_scaled.dat'),'w');
113     fprintf(fid,'%20.19f\n', num_fir_FreqSampl_scaled);
114     fclose(fid);
115     % la sequenza dati è applicata all'SRRC coi coefficienti scalati
116     data_out_SRRC_FreqSampl_scaled = applica_polifase(data_in_SRRCxN_tx_I, ...
117               SpS , num_fir_FreqSampl_scaled);
118     % Power Spectral Density plotting
119     [Pyy_FreqSampl_scaled , f_out] = pwelch(data_out_SRRC_FreqSampl_scaled,...
120       [],[], 'onesided', length(data_out_SRRC_FreqSampl_scaled) - 1 , f_clk);
121     Pyy_FreqSampl_scaled_dB = 10*log10(Pyy_FreqSampl_scaled) ;
122     figure; plot(f_out,Pyy_FreqSampl_scaled_dB-max(Pyy_FreqSampl_scaled_dB));
123     grid ; xlim([0 f_clk/2]) ;    ylim([-70 0]);
124     legend(['SRRCx' num2str(SpS) ' FreqSampl scaled ' ...
125                                 [num2str(symbol_rate/10^6)] ' MSpS' ]);
126     xlabel('Frequency (Hz)'); ylabel('dB / Hz');
127     figure ; impz(num_fir_FreqSampl_scaled) ;
128 end
```

Listato E.3.2: applica_polifase.m

```
1   % DESCRIPTION : Applica il filtraggio secondo lo schema polifase
2
3   function data_out_polifase=applica_polifase(data_in_polifase,SpS,coefficienti)
4
5   % n° di FIR che costituiscono il polifase
6   n_fir        = SpS                      ;
7   n_symb       = length(data_in_polifase)  ;
8   % n° di coefficienti del filtro da realizzare
9   n_coefficienti = length(coefficienti)    ;
10  % lunghezza massima dei FIR costituenti il polifase
11  max_dim_fir  = ceil( n_coefficienti / n_fir) ;
12  % creo una matrice che in ogni riga ha i coefficienti di uno dei rami del
13  %  filtro SRRC polifase
14  matrice_dei_fir = zeros(n_fir, max_dim_fir)  ;
15  for i = 1 : n_fir       % riempio una riga alla volta
16      % estraggo i coefficenti dell'i_esimo fir
17      fir_i = coefficienti(i : n_fir : n_coefficienti) ;
18      %figure ; freqz(fir_i) ; figure ; impz(fir_i) ;
19      % li copio nella matrice dei fir
20      matrice_dei_fir(i , 1:length(fir_i) )= fir_i ;
21  end
22  % applico la sequenza dati ad ogni fir
23  matrice_fir_out = zeros( n_fir , n_symb );
24  for i = 1 : n_fir
25      matrice_fir_out(i,:)=filter(matrice_dei_fir(i,:),1,data_in_polifase');
```

```
end
% creo un vettore con le uscite dal filtro prese come le prenderebbe un
% multiplexer questo vettore in pratica è il risultato dell'interpolazione
data_out_polifase = zeros( 1 , n_symb * n_fir );
k = 0 ;                 % è un puntatore per scandire il vettore da riempire
for colonna = 1 : n_symb    % considero un simbolo alla volta
   for riga = 1 : n_fir     % ed un fir alla volta
      k = k + 1 ;
      data_out_polifase(k) = matrice_fir_out(riga , colonna);
   end
end
```

Listato E.3.3: RaisedCosineResponse.m

```
% DESCRIPTION : Calcola la risposta in frequenza del filtro a coseno rialzato

function [y] = RaisedCosineResponse(f,roll_off,T)
if (abs(f) > ((1+roll_off)/(2*T))),
   y = 0;
elseif (abs(f) > ((1-roll_off)/(2*T))),
   y = (T/2)*(1+cos((pi*T/roll_off)*(abs(f)-(1-roll_off)/(2*T))));
else
   y = T;
end
```

Listato E.3.4: CreaROM.m

```
% DESCRIPTION : genera i valori da inserire nella ROM del polifase mediante
%               il costrutto VHDL case oppure il Constant oppure genera un
%                file in formato .coe adatto ad essere fornito ad un Core
%                  Xilinx che implementa una BlockRAM

% Pulizia ambiente Matlab
clear all ;
close all ;
clc       ;
% vengono impostate le caratteristiche del polifase
prompt           = {'Interpolazione :',
                    'N° di bit dopo la virgola :'};
title            = 'Caratteristiche del polifase';
def              = {'3', '11'};
answer           = inputdlg(prompt, title, 1, def) ;
SpS              = str2double(answer(1));
n_bit_dopo_virgola = str2double(answer(2));
% scelta tra l'implementazione della ROM con il CASE o con una costante
implementazione  = questdlg('Scegliere il codice VHDL da generare ', ...
       'Selezione implementazione ROM','Case','Costante','BlockRAM','Case');
% carico da file il vettore contenente i coefficienti del filtro gia scalati
fid    = fopen(strcat('SRRCx',num2str(SpS),'_FreqSampl_scaled.dat'),'r');
num_fir = fscanf(fid,'%f')                    ;
num_fir = num_fir'                            ;
fclose(fid)                                   ;
% n° di FIR che costituiscono il polifase
n_fir       = SpS                             ;
% n° di coefficienti del filtro da realizzare
n_coefficienti = length(num_fir)              ;
% lunghezza massima dei FIR costituenti il polifase
max_dim_fir  = ceil( n_coefficienti / n_fir) ;
% matrice che in ogni riga ha i coefficienti di uno dei rami del filtro SRRC
% inizializzo per velocizzare il riempimento
matrice_dei_fir = zeros(n_fir, max_dim_fir) ;
for i = 1 : n_fir        % riempio una riga alla volta
   % estraggo i coefficienti dell'i_esimo fir
   fir_i = num_fir(i : n_fir : n_coefficienti) ;
   % li copio nella matrice dei fir
   matrice_dei_fir(i , 1:length(fir_i) )= fir_i ;
end
% indirizzo della ROM da inserire nel file
rom_addr = 0 ;
% viene aperto il file nel quale memorizzare i valori della ROM
fid = fopen(strcat('SRRCx',num2str(SpS),'ROM_',implementazione,'.dat'),'w');
switch(implementazione)
   case 'BlockRAM',
      % viene richiesto se generare una RAM zero padded
      RamPadded   = questdlg('Generare una RAM zero padded ', ...
                             'Selezione tipo ROM','Si','No','Si');

      % sintassi per il file .coe da fornire al CORE BlockRAM
      fprintf(fid,'%s\n', 'MEMORY_INITIALIZATION_RADIX=16;') ;
      fprintf(fid,'%s\n', 'MEMORY_INITIALIZATION_VECTOR=') ;
```

```
54  end
55  % itero sui fir che costituiscono il polifase
56  %for i = 1 : n_fir
57  for i = n_fir : -1 : 1
58      % viene scritta sul file l'istruzione per l'assegnamento di default
59      % per i valori non assegnati
60      switch(implementazione)
61          case {'Case','Costante'},
62              % aggiungo al file il commento che indica il numero del FIR
63              fprintf(fid,'%s\n', ' ') ;
64              fprintf(fid,'%s', '    -- somme per il FIR ') ;
65              %fprintf(fid,'%s\n', num2str(i)) ;
66              fprintf(fid,'%s\n', num2str(n_fir - i)) ;
67      end
68      % calcolo le somme da inserire nel file per il filtro i_esimo
69      % itero su tutte le combinazioni d'ingresso
70      for eing = 0 : 1 : 2^max_dim_fir - 1
71          % la sequenza d'ingresso attuale viene convertita in formato rz +1,0
72          eing_rz = bitget(eing,max_dim_fir:-1:1) ;
73          % la sequenza d'ingresso attuale viene convertita in formato nrz +1,-1
74          eing_nrz = ( -2 .* eing_rz )+ 1 ;
75          % vengono sommati i coefficienti pesati del filtro i_esimo
76          somma   = sum( eing_nrz .* matrice_dei_fir(i,:) )  ;
77          % le somme vengono convertite in interi e arrotondate in modo da poter
78          %  esser elaborate dal VHDL
79          somma_int = round( somma * 2^n_bit_dopo_virgola );
80          % calcolo l'intero in complemento a due corrispondente alla somma
81          if sign(somma_int) == -1 % coefficiente negativo
82            somma_int_C2 = 2^(n_bit_dopo_virgola+1) - abs(somma_int); %2^N - |x|
83          else                     % coefficiente positivo
84            somma_int_C2 = abs(somma_int) ; % |x|
85          end
86          % aggiungo al file l'i_esimo valore della somma del fir
87          switch(implementazione)
88              case 'Case',
89                  % sintassi per il costrutto VHDL Case
90                  fprintf(fid,'%s', '       when ') ;
91                  fprintf(fid,'%3d', rom_addr );
92                  fprintf(fid,'%s', ' => SRRC_out <= X"') ;
93                  fprintf(fid,'%03X', somma_int_C2) ;
94                  fprintf(fid,'%s\n', '";') ;
95              case 'Costante',
96                  % sintassi per il costrutto VHDL Constant
97                  fprintf(fid,'%s', '       ROM_value''(X"') ;
98                  fprintf(fid,'%03X', somma_int_C2) ;
99                  fprintf(fid,'%s', '") , -- ') ;
100                 fprintf(fid,'%3d', rom_addr) ;
101                 fprintf(fid,'%s\n', ' ') ;
102             case 'BlockRAM',
103                 % sintassi per il file .coe da fornire al CORE BlockRAM
104                 fprintf(fid,'%03X', somma_int_C2) ;
105                 fprintf(fid,'%s\n', ', ') ;
106             otherwise, disp('Unknown method.')
107         end
108         % punto alla successiva cella di memoria da riempire
109         rom_addr = rom_addr + 1 ;
110     end
111 end
112 % viene scritta sul file l'istruzione per l'assegnamento di default
113 %  per i valori non assegnati
114 switch(implementazione)
115     case 'Case',
116         % sintassi per il costrutto VHDL Case
117         fprintf(fid,'%s\n', '       when OTHERS => SRRC_out <= X"000";');
118     case 'Costante',
119         % sintassi per il costrutto VHDL Constant
120         fprintf(fid,'%s\n', ' ') ;
121         fprintf(fid,'%s\n', '       OTHERS => ROM_value''(X"000") );');
122     case 'BlockRAM',
123         % se la RAM è padded aggiungo degli 0 che la occupano tutta in questo
124         % modo si possono mettere in un'unica RAM tutti e tre gli SRRC
125         if strcmp(RamPadded , 'Si')
126             % per l'indirizzamento considero la ROM di dimensioni maggiori
127             for addr = rom_addr : 1 : 2^(max_dim_fir + round(log2(6))) - 1
128                 fprintf(fid,'%03X', 0) ;
129                 fprintf(fid,'%s\n', ', ') ;
130             end
131         end
132         fprintf(fid,'%s\n', ';') ;
133     otherwise, disp('Unknown method.')
134 end
135 % viene chiuso il file contenente i valori delle somme da porre nella ROM
136 fclose(fid) ;       disp('Generato file valori ROM .') ;
```

Test VHDL

Listato E.3.5: PolyphasePSDVHDLvsPSDMatlab.m

```
% DESCRIPTION : Confronta lo spettro della sequenza filtrata tramite il VHDL
%                con quello della sequenza filtrata tramite Matlab

% Pulizia ambiente Matlab
clear all ;
close all ;
clc       ;
% ************** IMPOSTAZIONE CARATTERISTICHE DEL MODULATORE **************
% vengono impostate le caratteristiche del polifase
prompt           = {'Frequenza di clock :',
                    'Symbol Rate :',
                    'N° di bit dopo la virgola :'};
title            = 'Caratteristiche del polifase';
def              = {'40', '40/3', '11'};
answer           = inputdlg(prompt, title, 1, def) ;
f_clk            = str2double(answer(1))*10^6;
symbol_rate      = eval( char(answer(2)) )*10^6;
n_bit_dopo_virgola = str2double(answer(3));
% viene determinato il valore dell'interpolazione
SpS       = f_clk / symbol_rate    ;
% selezione del file sorgente tra quello prodotto dalla simulazione semplice e
%  quello della back-annotata
file_simulazione = questdlg('Selezionare il nome del file sorgente VHDL :',...
                'Selezione simulazione back-annotata o semplice',...
                'data_out_SRRCxN_tx_I.dat','TB_data_out_SRRCxN_tx_I.dat',...
                'data_out_SRRCxN_tx_I.dat');

% ****  VIENE APPLICATA LA DECOMPOSIZIONE POLIFASE MATLAB DI RIFERIMENTO ****
% carica da file la sequenza dei dati da filtrare
fid = fopen('data_in_SRRCxN_tx_I_nrz.dat' , 'r') ;
data_in_SRRCxN_tx_I = fscanf(fid,'%f')    ;
fclose(fid)                               ;
% carica da file i coefficienti del filtro polifase
fid    = fopen( strcat('SRRCx', num2str(SpS), '_FreqSampl_scaled.dat'),'r');
num_fir = fscanf(fid,'%f')                ;
num_fir = num_fir'                        ;
fclose(fid)                               ;
% applica il filtraggio polifase Matlab
data_out_SRRCxN_tx_I = applica_polifase(data_in_SRRCxN_tx_I, SpS , num_fir);
% con il metodo di Welch viene calcolato lo spettro della sequenza filtrata
%  mediante il polifase MATLAB
[Pyy_Matlab , f_out] = pwelch(data_out_SRRCxN_tx_I, [] , [] , 'onesided',...
        length(data_out_SRRCxN_tx_I) - 1 , f_clk) ;
Pyy_dB_Matlab = 10*log10(Pyy_Matlab) ;
figure ;          plot(f_out , Pyy_dB_Matlab - max(Pyy_dB_Matlab) );
grid  ;           xlim([0 f_clk/2]) ;           ylim([-70 0]);
legend(['Out SRRCx',num2str(SpS),' Matlab ',num2str(symbol_rate/1e6),' MSpS']);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

% ************** VIENE APPLICATA LA DECOMPOSIZIONE POLIFASE VHDL *********
% carica da file la sequenza filtrata dal polifase VHDL
fid = fopen(file_simulazione , 'r') ;
   % viene scartato il primo numero che è un overflow
   string_unuseful = fscanf(fid, '%s' , 1);
   % vengono letti un numero di campioni pari a quelli del polifase Matlab
   data_out_SRRCxN_tx_I_dec=fscanf(fid,'%f',length(data_in_SRRCxN_tx_I)*SpS);
fclose(fid);
data_out_SRRCxN_tx_I_VHDL=data_out_SRRCxN_tx_I_dec.*2^(- n_bit_dopo_virgola);
% viene prodotta una stringa da aggiungere alla label nel caso di simulazione
%  back-annotata
if strcmp(file_simulazione , 'TB_data_out_SRRCxN_tx_I.dat')
   tipo_simulazione = 'B.A. ' ;
else
   tipo_simulazione = '' ;
end
% con il metodo di Welch viene calcolato lo spettro della sequenza filtrata
%  dal polifase VHDL
[Pyy_VHDL , f_out]=pwelch(data_out_SRRCxN_tx_I_VHDL, [] , [] , 'onesided',...
                    length(data_out_SRRCxN_tx_I_VHDL) - 1 , f_clk) ;
Pyy_dB_VHDL = 10*log10(Pyy_VHDL) ;
figure ;          plot(f_out , Pyy_dB_VHDL - max(Pyy_dB_VHDL) , '-r');
grid  ;           xlim([0 f_clk/2]) ;          ylim([-70 0]);
legend(['Out SRRCx',num2str(SpS),' VHDL ',tipo_simulazione,...
       num2str(symbol_rate/1e6),' MSpS']);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

% ***************************** CONFRONTO SPETTRALE ********************
% visualizzazione contemporanea dei due spettri
figure ;
plot(f_out,Pyy_dB_Matlab-max(Pyy_dB_Matlab),'-b',f_out,...
                          Pyy_dB_VHDL-max(Pyy_dB_VHDL),'-r');
grid  ;           xlim([0 f_clk/2]) ;          ylim([-70 0]);
legend(['Out SRRCx', num2str(SpS), ' Matlab ' , num2str(symbol_rate/1e6),...
```

```
84          ' MSpS' ],['Out SRRCx', num2str(SpS),' VHDL ',tipo_simulazione,...
85         num2str(symbol_rate/1e6) , ' MSpS' ] );
86  xlabel('Frequency (Hz)'); ylabel('dB / Hz');
87  % ****************************** CONFRONTO TEMPORALE ********************
88  % n_bits = length(data_out_SRRCxN_tx_I);
89  % N_bit_sfasamento = 8 ;
90  % [data_out_SRRCxN_tx_I((N_bit_sfasamento+1):n_bits)',...
91  %   data_out_SRRCxN_tx_I_VHDL(1:(n_bits - N_bit_sfasamento))]
```

Listato E.3.6: applica_polifase.m

vedi listato(E.3.2)

Test FPGA

Listato E.3.7: VisualizzaPSDPolifaseVHDLFPGAout.m

```
1   % DESCRIPTION : carica da file la sequenza dati filtrata tramite l'FPGA e
2   %               ne fa l'analisi spettrale
3
4   % Pulizia ambiente Matlab
5   clear all ;
6   close all ;
7   clc     ;
8   % ************ IMPOSTAZIONE CARATTERISTICHE DEL MODULATORE ************
9   % vengono impostate le caratteristiche del polifase
10  prompt           = {'Frequenza di clock :',
11                      'Symbol Rate :',
12                      'N° di bit dopo la virgola :',
13                      'N° di campioni da considerare :'};
14  title            = 'Caratteristiche del polifase';
15  def              = {'40', '40/3', '11', '18000'};
16  answer           = inputdlg(prompt, title, 1, def) ;
17  f_clk            = str2double(answer(1))*10^6;
18  symbol_rate      = eval( char(answer(2)) )*10^6;
19  n_bit_dopo_virgola = str2double(answer(3));
20  n_values         = str2double(answer(4));
21  % viene determinato il valore dell'interpolazione
22  SpS          = f_clk / symbol_rate     ;
23  % ****  VIENE APPLICATA LA DECOMPOSIZIONE POLIFASE MATLAB DI RIFERIMENTO ****
24  % carica da file la sequenza dei dati da filtrare
25  fid = fopen('data_in_SRRCxN_tx_I_nrz.dat' , 'r') ;
26  data_in_SRRCxN_tx_I = fscanf(fid,'%f')    ;
27  fclose(fid)                               ;
28  % carica da file i coefficienti del filtro polifase
29  fid    = fopen( strcat('SRRCx',num2str(SpS),'_FreqSampl_scaled.dat'),'r');
30  num_fir = fscanf(fid,'%f')                ;
31  num_fir = num_fir'                        ;
32  fclose(fid)                               ;
33  % applica il filtraggio polifase Matlab
34  data_out_SRRCxN_tx_I = applica_polifase(data_in_SRRCxN_tx_I, SpS , num_fir);
35  % con il metodo di Welch viene calcolato lo spettro della sequenza
36  %   filtrata dal polifase MATLAB
37  [Pyy_Matlab , f_out] = pwelch(data_out_SRRCxN_tx_I, [] , [] , 'onesided',...
38                                length(data_out_SRRCxN_tx_I) - 1 , f_clk) ;
39  Pyy_dB_Matlab = 10*log10(Pyy_Matlab) ;
40  figure ;          plot(f_out , Pyy_dB_Matlab - max(Pyy_dB_Matlab) , '-b');
41  grid   ;          xlim([0 f_clk/2]) ;          ylim([-70 0]);
42  legend(['Out SRRCx', num2str(SpS), ' Matlab ' , ...
43           num2str(symbol_rate/1e6) , ' MSpS' ]);
44  xlabel('Frequency (Hz)'); ylabel('dB / Hz');
45  % ************ ELABORAZIONE DELLA SEQUENZA FILTRATA DALL'FPGA *************
46  % carica la sequenza proveniente dall'analizzatore di stati logici
47  fid = fopen('file_out.txt' , 'r');
48      % vengono scartate le prime 7 stringhe
49      string_unuseful = fscanf(fid, '%s' , 7);
50      % viene prelevata la stringa che indica il formato
51      formato = fscanf(fid, '%s' , 1) ;
52      % viene scartata una stringa con tutti -
```

```
53      string_unuseful = fscanf(fid, '%s' , 1);
54      % i dati vengono letti dal file nel giusto formato
55      switch(formato)
56         case 'Hex',
57            value_int_C2 = fscanf(fid, '%x' , n_values) ;
58         case 'Decimal',
59            value_int_C2 = fscanf(fid, '%f' , n_values) ;
60      end
61   fclose(fid)                                ;
62   % converte l'intero unsigned in complemento a due in un intero con segno
63   value_int = zeros(n_values,1);
64   for i = 1 : 1 : n_values
65      if bitget( value_int_C2(i) , 12) == 1 % valore negativo
66         % x_C2 = 2^N - |x|
67         value_int(i) = -( 2^(n_bit_dopo_virgola+1) - value_int_C2(i) );
68      else                                  % coefficiente positivo
69         value_int(i) = value_int_C2(i) ; % x_C2 = |x|
70      end
71   end
72   % converte da intero con segno a numero reale con segno
73   value_real = value_int .* 2^(- n_bit_dopo_virgola);
74   data_out_SRRCxN_tx_I_FPGA = value_real ;
75   % con il metodo di Welch viene calcolato lo spettro della sequenza
76   %  filtrata dal polifase VHDL e plottata
77   [Pyy_FPGA , f_out] = pwelch(data_out_SRRCxN_tx_I_FPGA, [] , [] , ...
78          'onesided', length(data_out_SRRCxN_tx_I_FPGA) - 1 , f_clk) ;
79   Pyy_dB_FPGA = 10*log10(Pyy_FPGA) ;
80   figure ;          plot(f_out , Pyy_dB_FPGA - max(Pyy_dB_FPGA) , '-r' );
81   grid   ;          xlim([0 f_clk/2]) ;          ylim([-70 0]);
82   legend(['Out SRRCx', num2str(SpS), ' FPGA ' , ...
83           num2str(symbol_rate/1e6), ' MSpS' ]);
84   xlabel('Frequency (Hz)'); ylabel('dB / Hz');
85
86   % ************** CONFRONTO SPETTRALE *********************************
87   % visualizzazione contemporanea dei due spettri
88   figure ;
89   plot(f_out , Pyy_dB_Matlab - max(Pyy_dB_Matlab) , '-b' , f_out , ...
90                                Pyy_dB_FPGA - max(Pyy_dB_FPGA) , '-r');
91   grid   ;          xlim([0 f_clk/2]) ;          ylim([-70 0]);
92   legend(['Out SRRCx', num2str(SpS), ' Matlab ' , num2str(symbol_rate/1e6),...
93           ' MSpS' ], ['Out SRRCx', num2str(SpS), ' FPGA ' , ...
94          num2str(symbol_rate/1e6) , ' MSpS' ] );
95   xlabel('Frequency (Hz)'); ylabel('dB / Hz');
96   % ******************* CONFRONTO TEMPORALE ****************************
97   % n_bits = length(data_out_SRRCxN_tx_I);
98   % N_bit_sfasamento = 8 ;
99   % [data_out_SRRCxN_tx_I((N_bit_sfasamento+1):n_bits)' , ...
100  %           data_out_SRRCxN_tx_I_FPGA(1:(n_bits - N_bit_sfasamento))]
```

Listato E.3.8: applica_polifase.m

vedi listato(E.3.2)

E.4 Modulatore

Tabella e grafico BER

Listato E.4.1: CreaTabellaBER.m

```
1   % MODULE NAME  : CreaTabellaBER.m
2   % DESCRIPTION : Crea una tabella ed un grafico del BER in funzione di Eb/N0
3
4   % Pulizia ambiente Matlab
5   clear all ;
6   close all ;
7   clc     ;
8   % ************************************************************************
```

```
% ***************** IMPOSTAZIONE CARATTERISTICHE MISURA ****************
% **********************************************************************
prompt           = {'Frequenza di clock :',
                    'Data Rate :',
                    'Eb/No minimo (dB) :',
                    'Eb/No massimo (dB) :',
                    'N° di ripetizioni per ogni valore di Eb/No :'};
title            = 'Impostazione della simulazione Modem QPSK';
def              = {'165', '110', '2', '6', '2'};
answer           = inputdlg(prompt, title, 1, def) ;
f_clk            = str2double(answer(1))*10^6;
data_rate        = str2double(answer(2))*10^6;
EbNo_dB_min      = str2double(answer(3));
EbNo_dB_max      = str2double(answer(4));
Repeat           = str2double(answer(5));

% messaggio per rassicurare l'operatore, qualcosa si muove anche se non sembra
disp('...elaborazione della tabella dei BER in corso')
disp(' ')
disp(' ')
% iterazione sui diversi valori di EbNo_dB
for EbNo_dB = EbNo_dB_min : 1 : EbNo_dB_max
    EbNo       = 10.^(EbNo_dB/10)                                    ;
    expBER     = 0.5 .* erfc(sqrt(EbNo))                             ;
    expBER_vect(EbNo_dB + 1 - EbNo_dB_min) = expBER                  ;
    n_bits     = 60 * 12 .* round( [1 ./ expBER'] ./ 12)             ;
    if n_bits < 12000
      n_bits = 12000 ;
    end
    for i = 1:1:Repeat
       [n_bit_errati , BER] = calcolaBER(n_bits , EbNo_dB, f_clk , data_rate) ;
      temp_BER(i)       = BER                                        ;
    end
    BER_vect(EbNo_dB + 1 - EbNo_dB_min) = ( sum(temp_BER) )/ Repeat ;
end
% Viene plottato il BER teorico affiancato al BER sperimentale
EbNo_dB = [EbNo_dB_min : 1 : EbNo_dB_max] ;
semilogy(EbNo_dB(:), BER_vect, 'b-', EbNo_dB(:), expBER_vect, 'r-');
legend('BER sperimentale ','BER Teorico',0); grid on;
xlabel('EbNo (dB)');       ylabel('BER');
disp('    Eb/No BER teorico BER sperimentale')
    [EbNo_dB' expBER_vect' BER_vect']
```

Listato E.4.2: calcolaBER.m

```
% MODULE NAME  : Modulatore QPSK con applica_polifase SRRC.m
% DESCRIPTION : Funzione per la misura del BER
% DATE              : 19-10-2001

function [n_bit_errati_Matlab , BER_Matlab] = ...
                BER_sperimentale(n_bits , EbNo_dB , f_clk , data_rate)

%%%%%%%%%%%%%%%%%%%%%%%% IMPOSTAZIONE MODULATORE %%%%%%%%%%%%%%%%%%%%%%%%%
symbol_rate = data_rate / 2  ;
SpS         = f_clk / symbol_rate ;
n_symb      = n_bits / 2       ;
n_sample    = n_symb * SpS     ;

% crea un vettore di bits randomici
bit_tx = randint(n_bits, 1, [0 1] );
n_bits_tx = length(bit_tx) ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MODULAZIONE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% carica da file i coefficienti del filtro polifase
fid    = fopen( strcat('SRRCx', num2str(SpS), '_FreqSampl_scaled.dat'),'r');
num_fir = fscanf(fid,'%f')                ;
num_fir = num_fir'                        ;
fclose(fid)                               ;
% viene creato il vettore dei bit pari e quello dei bit dispari
bit_tx_I = bit_tx(1 : 2 : n_bits_tx)      ;
bit_tx_Q = bit_tx(2 : 2 : n_bits_tx)      ;
% si passa dalla codifica RZ alla codifica NRZ
data_in_SRRCxN_tx_I_Matlab = -2*bit_tx_I + 1 ;
data_in_SRRCxN_tx_Q_Matlab = -2*bit_tx_Q + 1 ;
% applica il filtraggio polifase Matlab
data_out_SRRCxN_tx_I_Matlab = applica_polifase(data_in_SRRCxN_tx_I_Matlab,...
                                SpS , num_fir);
data_out_SRRCxN_tx_Q_Matlab = applica_polifase(data_in_SRRCxN_tx_Q_Matlab,...
                                SpS , num_fir);
% Vengono generati i campioni di seno e coseno considerando f_IF = f_clk / 4
% n° di campioni da generare per le portanti
n_sample = length(data_out_SRRCxN_tx_I_Matlab);
% periodo di campionamento
t_clk   = 1 / f_clk                    ;
% istanti di campionamento
```

```
t       = 0 : t_clk : (n_sample-1)*t_clk ;
% incrementi di fase
theta   = 2*pi*(f_clk/4)*t              ;
% generazione di un coseno
coseno  = [cos(theta)]                  ;
% generazione di un seno
  seno  = [sin(theta)]                  ;
data_out_QPSK_Mod_Matlab = [coseno .* data_out_SRRCxN_tx_I_Matlab] - ...
                           [seno .* data_out_SRRCxN_tx_Q_Matlab] ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% EFFETTO DEL CANALE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data_in_QPSK_DeMod_Matlab = awgn(data_out_QPSK_Mod_Matlab, ...
          EbNo_dB + 10*log10(2) -10*log10(0.5.*SpS), 'measured', [], 'dB');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULAZIONE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% il segnale in banda traslata viene suddiviso nelle componenti in banda base
data_in_SRRCxN_rx_I_Matlab = coseno .* data_in_QPSK_DeMod_Matlab ;
data_in_SRRCxN_rx_Q_Matlab = -seno .* data_in_QPSK_DeMod_Matlab ;
% applico il filtro adattato ad entrambe le componenti
data_out_SRRCxN_rx_I_Matlab = filter(num_fir, 1 , data_in_SRRCxN_rx_I_Matlab);
data_out_SRRCxN_rx_Q_Matlab = filter(num_fir, 1 , data_in_SRRCxN_rx_Q_Matlab);
data_I_Q_rx_Matlab = ...
            [data_out_SRRCxN_rx_I_Matlab' data_out_SRRCxN_rx_Q_Matlab'];
% viene effettuato uno sfasamento, una decimazione ed applicato il decisore
symbols_Matlab = demodmap(data_I_Q_rx_Matlab , [symbol_rate 1] , f_clk , ...
                    'qask' , 4) ;
% l'uscita del decisore viene convertita in formato NRZ
for n = 1:length(symbols_Matlab)
    switch symbols_Matlab(n)
       case 0
          symbols_I_Matlab(n) = 1 ; symbols_Q_Matlab(n) = 1 ;
       case 1
          symbols_I_Matlab(n) = -1 ; symbols_Q_Matlab(n) = 1 ;
       case 2
          symbols_I_Matlab(n) = 1 ; symbols_Q_Matlab(n) = -1 ;
       case 3
          symbols_I_Matlab(n) = -1 ; symbols_Q_Matlab(n) = -1 ;
    end
end
% si passa dalla codifica NRZ alla codifica RZ
bit_rx_I_Matlab = ( symbols_I_Matlab - 1) / (-2) ;
bit_rx_Q_Matlab = ( symbols_Q_Matlab - 1) / (-2) ;
% dai simboli si riottiene la sequenza composta da bit I e Q alternati
bit_rx_Matlab = 4*ones(1 , n_bits_tx) ;
bit_rx_Matlab(1 : 2 : n_bits_tx) = bit_rx_I_Matlab ;
bit_rx_Matlab(2 : 2 : n_bits_tx) = bit_rx_Q_Matlab ;
% viene compensato lo sfasamento tra la sequenza trasmessa e quella ricevuta
N_bit_sfasamento_Matlab = 12 ;
[n_bit_errati_Matlab BER_Matlab] = ...
   biterr(bit_tx(1:(n_bits_tx - N_bit_sfasamento_Matlab)), ...
     bit_rx_Matlab( (N_bit_sfasamento_Matlab + 1):n_bits_tx)') ;
```

Listato E.4.3: applica_polifase.m

vedi listato(E.3.2)

Test VHDL

Listato E.4.4: QPSKModemPSDeBERVHDLvsMatlab.m

```
% DESCRIPTION : Confronta lo spettro ed il BER della sequenza modulata
%                    tramite il VHDL con quello Matlab

% Pulizia ambiente Matlab
clear all ;
close all ;
clc       ;
% ***************************************************************************
% ***************** IMPOSTAZIONE CARATTERISTICHE DEL MODEM ******************
% ***************************************************************************
prompt           = {'Frequenza di clock :',
```

```
                    'Symbol Rate :',
                    'N° di bit dopo la virgola :'
                    'Eb/No (dB)'  };
title             = 'Impostazione della simulazione Modem QPSK';
def               = {'40', '40/3', '11', '6'};
answer            = inputdlg(prompt, title, 1, def) ;
f_clk             = str2double(answer(1))*10^6;
symbol_rate       = eval( char(answer(2)) )*10^6;
n_bit_dopo_virgola = str2double(answer(3));
EbNo_dB           = str2double(answer(4));
EbNo              = 10^(EbNo_dB/10);
% viene determinato il valore dell'interpolazione
SpS       = f_clk / symbol_rate    ;
% selezione del file sorgente tra quello prodotto dalla simulazione semplice e
%   quello della back-annotata
file_simulazione = questdlg('Selezionare il nome del file sorgente VHDL :',...
       'Selezione simulazione back-annotata o semplice', ...
       'data_out_QPSK_modulator.dat','TB_data_out_QPSK_modulator.dat',...
       'data_out_QPSK_modulator.dat');

% ************************************************************************
% ******************** MODEM QPSK MATLAB DI RIFERIMENTO ********************
% ************************************************************************
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MODULAZIONE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% carica da file i coefficienti del filtro polifase
fid    = fopen( strcat('SRRCx', num2str(SpS), '_FreqSampl_scaled.dat'),'r');
num_fir = fscanf(fid,'%f')                ;
num_fir = num_fir'                        ;
fclose(fid)                               ;
% carica da file la sequenza randomica da modulare
fid    = fopen('bit_tx.dat' , 'r')        ;
bit_tx = fscanf(fid,'%f')                 ;
fclose(fid)                               ;
n_bits_tx = length(bit_tx) ;
% viene creato il vettore dei bit pari e quello dei bit dispari
bit_tx_I = bit_tx(1 : 2 : n_bits_tx)      ;
bit_tx_Q = bit_tx(2 : 2 : n_bits_tx)      ;
% si passa dalla codifica RZ alla codifica NRZ
data_in_SRRCxN_tx_I_Matlab = -2*bit_tx_I + 1 ;
data_in_SRRCxN_tx_Q_Matlab = -2*bit_tx_Q + 1 ;
% applica il filtraggio polifase Matlab
data_out_SRRCxN_tx_I_Matlab = applica_polifase(data_in_SRRCxN_tx_I_Matlab,...
                                SpS , num_fir);
data_out_SRRCxN_tx_Q_Matlab = applica_polifase(data_in_SRRCxN_tx_Q_Matlab,...
                                SpS , num_fir);
% Vengono generati i campioni di seno e coseno considerando f_IF = f_clk / 4
% n° di campioni da generare per le portanti
n_sample = length(data_out_SRRCxN_tx_I_Matlab);
% periodo di campionamento
t_clk   = 1 / f_clk                       ;
% istanti di campionamento
t       = 0 : t_clk : (n_sample-1)*t_clk ;
% incrementi di fase
theta   = 2*pi*(f_clk/4)*t                ;
% generazione di un coseno
coseno  = [cos(theta)]                    ;
% generazione di un seno
  seno  = [sin(theta)]                    ;

 data_out_QPSK_Mod_Matlab = [coseno .* data_out_SRRCxN_tx_I_Matlab] - ...
                            [seno .* data_out_SRRCxN_tx_Q_Matlab] ;
% spettro del segnale QPSK modulato tramite Matlab
[Pyy_Mod_Matlab , f_out] = pwelch(data_out_QPSK_Mod_Matlab, [] , [] , ...
             'onesided', length(data_out_QPSK_Mod_Matlab) - 1 , f_clk) ;
Pyy_Mod_Matlab_dB = 10*log10(Pyy_Mod_Matlab) ;
figure ;      plot(f_out , Pyy_Mod_Matlab_dB - max(Pyy_Mod_Matlab_dB) );
grid   ;      xlim([0 f_clk/2]) ;          ylim([-60 0]);
legend(['Out QPSK Modem Matlab ' , num2str(2*symbol_rate/1e6) , ' MbpS'], 3);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% EFFETTO DEL CANALE %%%%%%%%%%%%%%%%%%%%%%%%%%%
data_in_QPSK_DeMod_Matlab = awgn(data_out_QPSK_Mod_Matlab, ...
          EbNo_dB + 10*log10(2) -10*log10(0.5.*SpS), 'measured', [], 'dB');
% viene calcolata e plottata la densità spettrale di potenza del segnale
%  QPSK affetto da rumore
[Pyy_DeMod_in_Matlab , f_out] = pwelch(data_in_QPSK_DeMod_Matlab,[],[], ...
             'onesided', length(data_in_QPSK_DeMod_Matlab) - 1 , f_clk) ;
Pyy_DeMod_in_Matlab_dB = 10*log10(Pyy_DeMod_in_Matlab) ;
figure ; plot(f_out, Pyy_DeMod_in_Matlab_dB - max(Pyy_DeMod_in_Matlab_dB));
grid  ; xlim([0 f_clk/2]);
legend(['In QPSK DeMod Matlab ' , num2str(2*symbol_rate/1e6) , ' MbpS'], 3);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULAZIONE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% il segnale in banda traslata viene suddiviso nelle componenti in banda base
data_in_SRRCxN_rx_I_Matlab = coseno .* data_in_QPSK_DeMod_Matlab ;
data_in_SRRCxN_rx_Q_Matlab = -seno .* data_in_QPSK_DeMod_Matlab ;
% applico il filtro adattato ad entrambe le componenti
data_out_SRRCxN_rx_I_Matlab = filter(num_fir, 1 , data_in_SRRCxN_rx_I_Matlab);
data_out_SRRCxN_rx_Q_Matlab = filter(num_fir, 1 , data_in_SRRCxN_rx_Q_Matlab);
```

```
% diagramma ad occhio della sequenza I ricevuta
eyediagram(data_out_SRRCxN_rx_I_Matlab(601:2400) , SpS , 1/symbol_rate , 2);
data_I_Q_rx_Matlab = ...
            [data_out_SRRCxN_rx_I_Matlab' data_out_SRRCxN_rx_Q_Matlab'];
% scatterplot del segnale ricevuto
scatterplot(data_I_Q_rx_Matlab , SpS , 2) ;
% viene effettuato uno sfasamento, una decimazione ed applicato il decisore
symbols_Matlab = demodmap(data_I_Q_rx_Matlab , [symbol_rate 1] , f_clk , ...
                    'qask' , 4) ;
% l'uscita del decisore viene convertita in formato NRZ
for n = 1:length(symbols_Matlab)
    switch symbols_Matlab(n)
       case 0
          symbols_I_Matlab(n) = 1 ; symbols_Q_Matlab(n) = 1 ;
       case 1
          symbols_I_Matlab(n) = -1 ; symbols_Q_Matlab(n) = 1 ;
       case 2
          symbols_I_Matlab(n) = 1 ; symbols_Q_Matlab(n) = -1 ;
       case 3
          symbols_I_Matlab(n) = -1 ; symbols_Q_Matlab(n) = -1 ;
    end
end
% si passa dalla codifica NRZ alla codifica RZ
bit_rx_I_Matlab = ( symbols_I_Matlab - 1) / (-2) ;
bit_rx_Q_Matlab = ( symbols_Q_Matlab - 1) / (-2) ;
% dai simboli si riottiene la sequenza composta da bit I e Q alternati
bit_rx_Matlab = 4*ones(1 , n_bits_tx) ;
bit_rx_Matlab(1 : 2 : n_bits_tx) = bit_rx_I_Matlab ;
bit_rx_Matlab(2 : 2 : n_bits_tx) = bit_rx_Q_Matlab ;
% viene compensato lo sfasamento tra la sequenza trasmessa e quella ricevuta
N_bit_sfasamento_Matlab = 12 ;
[n_bit_errati_Matlab BER_Matlab] = ...
   biterr(bit_tx(1:(n_bits_tx - N_bit_sfasamento_Matlab)), ...
    bit_rx_Matlab( (N_bit_sfasamento_Matlab + 1):n_bits_tx)') ;
disp(['          Risultati Modulatore Matlab -> Demodulatore Matlab ']);
disp(['N° di bit di sfasamento : ' num2str(N_bit_sfasamento_Matlab)]);
disp(['N° di bit errati    : ' num2str(n_bit_errati_Matlab)]);
disp(['BER                 : ' num2str(BER_Matlab)]);
% viene calcolato il valore dello sfasamento per il quale si ha la minore BER
for N_bit_sfasamento_Matlab = 1:1:15
   [n_bit_errati_Matlab(N_bit_sfasamento_Matlab) ...
         BER_Matlab(N_bit_sfasamento_Matlab)] = ...
         biterr(bit_tx(1:(n_bits_tx - N_bit_sfasamento_Matlab)) , ...
         bit_rx_Matlab( (N_bit_sfasamento_Matlab + 1):n_bits_tx)');
   disp(' ');
   disp(['N° di bit di sfasamento : ' num2str(N_bit_sfasamento_Matlab)]);
   disp(['N° di bit errati    : ' ...
                num2str(n_bit_errati_Matlab(N_bit_sfasamento_Matlab))]);
   disp(['BER                 : ' ...
                num2str(BER_Matlab(N_bit_sfasamento_Matlab))]);
end
[ber_Matlab , N_bit_sfasamento_Matlab] = min(BER_Matlab) ;
disp(' ');
disp(' ');
disp(['BER minima    : ' num2str(ber_Matlab)]);
disp(['N° di bit errati : ' ...
           num2str(n_bit_errati_Matlab(N_bit_sfasamento_Matlab))]);
disp(['Sfasamento    : ' num2str(N_bit_sfasamento_Matlab)]);
% Visualizza il valore teorico per il BER
theoretical_BER = 0.5 .* erfc(sqrt(EbNo)) ;
disp(' ');
disp(['BER teorico         : ' num2str(theoretical_BER)]);
disp(' ');

% ***************************************************************************
% ************** MODULATORE QPSK VHDL E DEMODULATORE MATLAB ******************
% ***************************************************************************
% carica da file la sequenza modulata tramite il VHDL
fid = fopen(file_simulazione , 'r') ;
   % viene scartato il primo numero che è un overflow
   string_unuseful = fscanf(fid, '%s' , 1);
   % vengono letti un numero di campioni pari a quelli del polifase Matlab
   data_out_QPSK_Mod_dec = fscanf(fid,'%f', ...
                              length(data_in_SRRCxN_tx_I_Matlab) * SpS);
fclose(fid);
data_out_QPSK_Mod_VHDL = data_out_QPSK_Mod_dec .* 2^(- n_bit_dopo_virgola);
% viene prodotta una stringa da aggiungere alla label nel caso di simulazione
%   back-annotata
if strcmp(file_simulazione , 'TB_data_out_QPSK_modulator.dat')
   tipo_simulazione = 'B.A. ' ;
else
   tipo_simulazione = '' ;
end
% spettro del segnale QPSK modulato tramite il VHDL
[Pyy_Mod_VHDL , f_out] = pwelch(data_out_QPSK_Mod_VHDL,[],[],'onesided',...
                         length(data_out_QPSK_Mod_VHDL) - 1 , f_clk) ;
Pyy_Mod_VHDL_dB = 10*log10(Pyy_Mod_VHDL) ;
figure ;     plot(f_out , Pyy_Mod_VHDL_dB - max(Pyy_Mod_VHDL_dB) , 'r');
grid  ;      xlim([0 f_clk/2]) ;          ylim([-60 0]);
legend(['Out QPSK Modem VHDL ' , tipo_simulazione , ...
```

```
                          num2str(2*symbol_rate/1e6) , ' MbpS'], 3);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% EFFETTO DEL CANALE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
data_in_QPSK_DeMod_VHDL = awgn(data_out_QPSK_Mod_VHDL, EbNo_dB +10*log10(2)...
                           -10*log10(0.5.*SpS), 'measured', [], 'dB');
% viene calcolata e plottata la densità spettrale di potenza del segnale QPSK
%      affetto da rumore
[Pyy_DeMod_in_VHDL , f_out] = pwelch(data_in_QPSK_DeMod_VHDL, [] , [] , ...
          'onesided', length(data_in_QPSK_DeMod_VHDL) - 1 , f_clk) ;
Pyy_DeMod_in_VHDL_dB = 10*log10(Pyy_DeMod_in_VHDL) ;
figure ; plot(f_out , Pyy_DeMod_in_VHDL_dB-max(Pyy_DeMod_in_VHDL_dB),'-r') ;
grid  ;          xlim([0 f_clk/2])          ;
legend(['In QPSK DeMod VHDL ' , num2str(2*symbol_rate/1e6) , ' MbpS'], 3) ;
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DEMODULAZIONE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROCEDURA PER L'ALLINEAMENTO DEL DEMODULATORE :
% Il demodulatore è suscettibile a tre variabili :
% quadrante_iniziale , offset_demodmap , N_bit_sfasamento_VHDL
% a) togliere il commento dal ciclo FOR che itera su N_bit_sfasamento_VHDL
% b) assegnare quadrante_iniziale = 0
% c) far partire la simulazione, alla fine si ha il minimo BER e il
%      corrispondente N_bit_sfasamento_VHDL
% d) fare lo stesso variando per i valori 1, 2, 3 di quadrante_iniziale
% e) fissando il valore minimo per quadrante_iniziale e N_bit_sfasamento_VHDL
%      ricercare il minimo di offset_demodmap

% Vengono generati i campioni di seno e coseno considerando f_IF = f_clk / 4
quadrante_iniziale = 1 ;
% incrementi di fase
theta_rx_VHDL = 2*pi*(f_clk/4)*t + quadrante_iniziale*pi/2 ;
coseno_rx_VHDL = [cos(theta_rx_VHDL)] ; % generazione del coseno
  seno_rx_VHDL = [sin(theta_rx_VHDL)] ; % generazione del seno
% il segnale in banda traslata viene suddiviso nelle componenti in banda base
data_in_SRRCxN_rx_I_VHDL = coseno_rx_VHDL .* data_in_QPSK_DeMod_VHDL' ;
data_in_SRRCxN_rx_Q_VHDL = -seno_rx_VHDL .* data_in_QPSK_DeMod_VHDL' ;
% applico il filtro adattato ad entrambe le componenti
data_out_SRRCxN_rx_I_VHDL = filter(num_fir, 1 , data_in_SRRCxN_rx_I_VHDL);
data_out_SRRCxN_rx_Q_VHDL = filter(num_fir, 1 , data_in_SRRCxN_rx_Q_VHDL);
% diagramma ad occhio della sequenza I ricevuta
%eyediagram(data_out_SRRCxN_rx_I_VHDL(601:2400),SpS,1/symbol_rate,3,'-r');
data_I_Q_rx_VHDL = [data_out_SRRCxN_rx_I_VHDL' data_out_SRRCxN_rx_Q_VHDL'];
% scatterplot del segnale ricevuto
%scatterplot(data_I_Q_rx_VHDL , SpS , 3) ;
% viene effettuato uno sfasamento, una decimazione ed applicato il decisore
offset_demodmap = 1 ;
symbols_VHDL = demodmap(data_I_Q_rx_VHDL , [symbol_rate offset_demodmap] ,...
               f_clk , 'qask' , 4) ;
% l'uscita del decisore viene convertita in formato NRZ
for n = 1:length(symbols_VHDL)
    switch symbols_VHDL(n)
       case 0
          symbols_I_VHDL(n) = 1 ; symbols_Q_VHDL(n) = 1 ;
       case 1
          symbols_I_VHDL(n) = -1 ; symbols_Q_VHDL(n) = 1 ;
       case 2
          symbols_I_VHDL(n) = 1 ; symbols_Q_VHDL(n) = -1 ;
       case 3
          symbols_I_VHDL(n) = -1 ; symbols_Q_VHDL(n) = -1 ;
    end
end
% si passa dalla codifica NRZ alla codifica RZ
bit_rx_I_VHDL = ( symbols_I_VHDL - 1) / (-2) ;
bit_rx_Q_VHDL = ( symbols_Q_VHDL - 1) / (-2) ;
% dai simboli si riottiene la sequenza composta da bit I e Q alternati
bit_rx_VHDL = 4*ones(1 , n_bits_tx) ;
bit_rx_VHDL(1 : 2 : n_bits_tx) = bit_rx_I_VHDL ;
bit_rx_VHDL(2 : 2 : n_bits_tx) = bit_rx_Q_VHDL ;
% viene compensato lo sfasamento tra la sequenza trasmessa e quella ricevuta
N_bit_sfasamento_VHDL = 16 ;
[n_bit_errati_VHDL BER_VHDL] = ...
       biterr(bit_tx(1:(n_bits_tx - N_bit_sfasamento_VHDL)) , ...
       bit_rx_VHDL( (N_bit_sfasamento_VHDL + 1):n_bits_tx)') ;
disp(['          Risultati Modulatore VHDL -> Demodulatore Matlab ']);
disp(['N° di bit di sfasamento : ' num2str(N_bit_sfasamento_VHDL)]);
disp(['N° di bit errati    : ' num2str(n_bit_errati_VHDL)]);
disp(['BER                 : ' num2str(BER_VHDL)]);
% viene calcolato il valore dello sfasamento per il quale si ha la minore BER
for N_bit_sfasamento_VHDL = 1:1:30
 [n_bit_errati_VHDL(N_bit_sfasamento_VHDL) BER_VHDL(N_bit_sfasamento_VHDL)]...
    = biterr(bit_tx(1:(n_bits_tx - N_bit_sfasamento_VHDL)) , ...
    bit_rx_VHDL( (N_bit_sfasamento_VHDL + 1):n_bits_tx)');
 disp(' ');
 disp(['N° di bit di sfasamento : ' num2str(N_bit_sfasamento_VHDL)]);
 disp(['N° di bit errati    : ' ...
                      num2str(n_bit_errati_VHDL(N_bit_sfasamento_VHDL))]);
 disp(['BER                 : ' num2str(BER_VHDL(N_bit_sfasamento_VHDL))]);
end
[ber_VHDL , N_bit_sfasamento_VHDL] = min(BER_VHDL) ;
```

```
282 disp(' ');
283 disp(' ');
284 disp(['BER minima    : 'num2str(ber_VHDL)]);
285 disp(['N° di bit errati : 'num2str(n_bit_errati_VHDL(N_bit_sfasamento_VHDL))]);
286 disp(['Sfasamento    : 'num2str(N_bit_sfasamento_VHDL)]);
287 % Visualizza il valore teorico per il BER
288 theoretical_BER = 0.5 .* erfc(sqrt(EbNo)) ;
289 disp(' ');
290 disp(['BER teorico        : ' num2str(theoretical_BER)]);
291 disp(' ');
292
293 % ************************************************************************
294 % ************************* CONFRONTO MODULATORI ************************
295 % ************************************************************************
296 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CONFRONTO SPETTRALE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
297 % visualizzazione contemporanea dei due spettri
298 figure ;      plot(f_out , Pyy_Mod_Matlab_dB - max(Pyy_Mod_Matlab_dB),...
299               'b' , f_out , Pyy_Mod_VHDL_dB - max(Pyy_Mod_VHDL_dB) , 'r');
300 grid  ;           xlim([0 f_clk/2]) ;          ylim([-60 0]);
301 legend(['Out QPSK Modem Matlab ' , num2str(2*symbol_rate/1e6) , ' MbpS' ],...
302       ['Out QPSK Modem VHDL ',tipo_simulazione,num2str(2*symbol_rate/1e6),...
303        ' MbpS' ] , 3);
304 xlabel('Frequency (Hz)'); ylabel('dB / Hz');
305 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CONFRONTO TEMPORALE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
306 % n_bits = 1000;
307 % N_bit_sfasamento = 7 ;
308 % [data_out_QPSK_Mod_Matlab(1:(n_bits - N_bit_sfasamento))' , ...
309 %         data_out_QPSK_Mod_VHDL((N_bit_sfasamento+1):n_bits)]
```

Listato E.4.5: applica_polifase.m

vedi listato(E.3.2)

Test FPGA

Listato E.4.6: QPSKModemPSDFPGAvsPSDMatlab.m

```
1  % DESCRIPTION : Confronta lo spettro della sequenza modulata tramite il VHDL
2  %                con quello della sequenza modulata tramite Matlab
3
4  % Pulizia ambiente Matlab
5  clear all ;
6  close all ;
7  clc     ;
8  % ************** IMPOSTAZIONE CARATTERISTICHE DEL MODULATORE *************
9  % vengono impostate le caratteristiche del modulatore
10 prompt           = {'Frequenza di clock :',
11                     'Symbol Rate :',
12                     'N° di bit dopo la virgola :',
13                     'N° di campioni da considerare :'};
14 title            = 'Caratteristiche del modulatore';
15 def              = {'40', '40/3', '11','18000'};
16 answer           = inputdlg(prompt, title, 1, def) ;
17 f_clk            = str2double(answer(1))*10^6;
18 symbol_rate      = eval( char(answer(2)) )*10^6;
19 n_bit_dopo_virgola = str2double(answer(3));
20 n_values         = str2double(answer(4));
21 % viene determinato il valore dell'interpolazione
22 SpS       = f_clk / symbol_rate    ;
23
24 % ********* VIENE APPLICATA LA MODULAZIONE QPSK MATLAB DI RIFERIMENTO ******
25 % carica da file la sequenza dei dati da filtrare sul ramo I
26 fid = fopen('data_in_SRRCxN_tx_I_nrz.dat' , 'r') ;
27 data_in_SRRCxN_tx_I = fscanf(fid,'%f')    ;
28 fclose(fid)                               ;
29 % carica da file la sequenza dei dati da filtrare sul ramo Q
30 fid = fopen('data_in_SRRCxN_tx_Q_nrz.dat' , 'r') ;
31 data_in_SRRCxN_tx_Q = fscanf(fid,'%f')    ;
32 fclose(fid)                               ;
```

```
% carica da file i coefficienti del filtro polifase
fid    = fopen( strcat('SRRCx', num2str(SpS), '_FreqSampl_scaled.dat'),'r');
num_fir = fscanf(fid,'%f')                    ;
num_fir = num_fir'                            ;
fclose(fid)                                   ;
% applica il filtraggio polifase Matlab
data_out_SRRCxN_tx_I = applica_polifase(data_in_SRRCxN_tx_I, SpS , num_fir);
data_out_SRRCxN_tx_Q = applica_polifase(data_in_SRRCxN_tx_Q, SpS , num_fir);
% Vengono generati i campioni di seno e coseno considerando che f_IF = f_clk/4
% n° di campioni da generare per le portanti
n_sample = length(data_out_SRRCxN_tx_I) ;
% periodo di campionamento
t_clk   = 1 / f_clk                           ;
% istanti di campionamento
t       = 0 : t_clk : (n_sample-1)*t_clk ;
% incrementi di fase per generare 41.25 MHz
theta   = 2*pi*(f_clk/4)*t                    ;
% generazione del coseno
coseno  = [cos(theta)]                        ;
% generazione del seno
  seno  = [sin(theta)]                        ;
data_out_QPSK_modulator = [coseno .* data_out_SRRCxN_tx_I] - ...
                                [seno   .* data_out_SRRCxN_tx_Q] ;
% viene calcolata e plottata la densità spettrale di potenza
[Pyy_Matlab , f_out] = pwelch(data_out_QPSK_modulator, [] , [] , ...
            'onesided', length(data_out_QPSK_modulator) - 1 , f_clk) ;
Pyy_dB_Matlab = 10*log10(Pyy_Matlab) ;
figure ;          plot(f_out , Pyy_dB_Matlab - max(Pyy_dB_Matlab) );
grid   ;          xlim([0 f_clk/2]) ;           ylim([-60 0]);
legend(['Out QPSK Modem Matlab ' , num2str(2*symbol_rate/1e6) , ' MbpS'], 3);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

% *************** VIENE APPLICATA LA MODULAZIONE QPSK VHDL **************
% ************ ELABORAZIONE DELLA SEQUENZA FILTRATA DALL'FPGA ************
% carica la sequenza proveniente dall'analizzatore di stati logici
fid = fopen('file_out.txt' , 'r');
    % vengono scartate le prime 7 stringhe
    string_unuseful = fscanf(fid, '%s' , 7);
    % viene prelevata la stringa che indica il formato
    formato = fscanf(fid, '%s' , 1) ;
    % viene scartata una stringa con tutti -
    string_unuseful = fscanf(fid, '%s' , 1);
    % i dati vengono letti dal file nel giusto formato
    switch(formato)
        case 'Hex',
            value_int_C2 = fscanf(fid, '%x' , n_values) ;
        case 'Decimal',
            value_int_C2 = fscanf(fid, '%f' , n_values) ;
    end
fclose(fid)                                   ;
% converte l'intero unsigned in complemento a due in un intero con segno
value_int = zeros(n_values,1);
for i = 1 : 1 : n_values
   if bitget( value_int_C2(i) , 12) == 1 % valore negativo
      % x_C2 = 2^N - |x|
      value_int(i) = -( 2^(n_bit_dopo_virgola+1) - value_int_C2(i) );
   else                                  % coefficiente positivo
      value_int(i) = value_int_C2(i) ; % x_C2 = |x|
   end
end
% converte da intero con segno a numero reale con segno
value_real = value_int .* 2^(- n_bit_dopo_virgola);
data_out_QPSK_modulator_FPGA = value_real ;
% viene calcolato lo spettro della sequenza modulata dal VHDL
[Pyy_FPGA , f_out] = pwelch(data_out_QPSK_modulator_FPGA, [] , [] , ...
            'onesided', length(data_out_QPSK_modulator_FPGA) - 1 , f_clk) ;
Pyy_dB_FPGA = 10*log10(Pyy_FPGA) ;
figure ;          plot(f_out , Pyy_dB_FPGA - max(Pyy_dB_FPGA) , '-r') ;
grid   ;          xlim([0 f_clk/2]) ;           ylim([-60 0])  ;
legend(['Out QPSK Modem FPGA ' , num2str(symbol_rate*2/1e6) , ' MbpS'], 3) ;
xlabel('Frequency (Hz)'); ylabel('dB / Hz');

% ************************** CONFRONTO SPETTRALE ************************
% visualizzazione contemporanea dei due spettri
figure ;          plot(f_out , Pyy_dB_Matlab - max(Pyy_dB_Matlab),'-b',...
                       f_out , Pyy_dB_FPGA - max(Pyy_dB_FPGA) , '-r');
grid   ;          xlim([0 f_clk/2]) ;           ylim([-60 0]);
legend(['Out QPSK Modem Matlab ' , num2str(symbol_rate*2/1e6) , ' MbpS' ] , ...
       ['Out QPSK Modem FPGA ' , num2str(symbol_rate*2/1e6) , ' MbpS'' ] , 3);
xlabel('Frequency (Hz)'); ylabel('dB / Hz');
% ************************** CONFRONTO TEMPORALE ************************
% n_bits = length(data_out_QPSK_modulator);
% n_bits = 12000;
% N_bit_sfasamento = 0 ;
% [data_out_QPSK_modulator(1:(n_bits - N_bit_sfasamento))' , ...
%       data_out_QPSK_modulator_FPGA((N_bit_sfasamento+1):n_bits)]
```

Listato E.4.7: applica_polifase.m

vedi listato(E.3.2)