```
function [rec_bits, ht, z_signal] = MatchedFilter(T_sq,E_bit,fs,y_signal,type)
%
% Inputs:
%   T_sq_dur:   Duration of the square pulse in seconds
%   E_bit:      Total energy in all samples of one square pulse
%   fs:         Sampling frequency
%   y_square:   Sequence of samples which correspond to the square pulses
%               of the input bits to be detected
%   type:       Type of bit coding, 'unipolar' or 'bipolar' (default is
%               'unipolar')
% Outputs:
%   rec_bits:   The sequence of bits decoded by the matched filter
%               corresponding to the input y_square
%   ht:         The impulse response of the matched filter
%   z_square:   The output of the convolution operation between the input
%               sequence y_square and the impulse response h_t
%
% This function implements the matched fiilter receiver for a square pulse
% shape. The function takes as input y_square, which contains the samples
% corresponding to the sequence of square pulse shapes of the input bits.
% The operation of the function is the matched filter operation:
%
%         1- It generates ht, the impulse response of the appropriate
%         matched filter
%         2- It performs a convolution operation between ht and the input
%         sequence y_square. The output of this operation is stored in a
%         variable called z_square
%         3- From the variable z_square, the function makes a decision on
%         the value of each input bit.
%
% Notes and hints:
%       - The dimensions of ht should be equal to N_sq, which is the length
%       of the square pulse used in the generation of the input sequence.
%       - The dimensions of z_square should be equal to the expected length
%       of the output of a convolution operation between two input
%       sequences: y_square and ht.
%       - From z_square (which should be a long vector, longer than the
%       expected number of input bits), the function should decide the
%       value of each input bit and store those in the variable rec_bits.
%       Note therefore that rec_bits should have a smaller length than
%       z_square.
%       - If type is not specified, it is assumed to be 'unipolar'.

if nargin < 5
    type = 'unipolar';
end

Ts = 1/fs;

N_sq = round(T_sq/Ts);          % Length of the square pulse used for pulse shaping
N_y_signal = length(y_signal);  % Length of the input sequence

N_bits = 0;
%%% WRITE YOUR CODE HERE
% Knowing the length of the square pulse and the length of the input
% sequence of pulses corresponding to the input bits, compute the number of
% input bits and store it in N_bits.
N_bits = round(N_y_signal / N_sq);
%%%

ht = [];
z_signal = [];
rec_bits = [];
switch type
```

```
case ('bipolar')
    %%% WRITE YOUR CODE HERE
    % Compute the MF impulse response ht, and the MF output signal
    % z_signal for the bipolar encoding case
    amp_bi = sqrt(E_bit/N_sq);
    xo = amp_bi*ones(1, N_sq);
    x1 = -amp_bi*ones(1, N_sq);

    ht = xo - x1;
    z_signal = conv(y_signal, ht);
    %%%

    %%% WRITE YOUR CODE HERE
    % Implement the decision part of the receiver with bipolar encoding
    % which uses z_signal to decide the values of the input bits
    E1 = E_bit/2;
    E2 = E_bit/2;
    Vth = (E1 - E2) / 2;
    rec_bits = zeros(1, N_bits);
    for i = 1:N_bits
       % the sampling time here can be chosen arbitrary as the MF is
       % designed for any chioce of Ts
       if z_signal(i*N_sq) >= Vth
           rec_bits(i) = 1;
       else
           rec_bits(i) = 0;
       end
    end

    %%%

case ('unipolar')


    %%% WRITE YOUR CODE HERE
    % Part 2-a: Compute the MF impulse response ht, and the MF output
    % signal z_signal for the unipolar encoding case
    amp_uni = sqrt((2*E_bit)/N_sq);
    xo = amp_uni*ones(1, N_sq);
    x1 = zeros(1, N_sq);

    ht = xo - x1;
    z_signal = conv(y_signal, ht);

    %%%

    %%% WRITE YOUR CODE HERE
    % Pat 2-b: Implement the decision part of the receiver with
    % unipolar encoding which uses z_signal to decide the values of the
    % input bits
    E1 = 2*E_bit;
    E2 = 0;
    Vth = (E1 - E2) / 2;
    rec_bits = zeros(1, N_bits);
    for i = 1:N_bits
       % the sampling time here can be chosen arbitrary as the MF is
       % designed for any chioce of Ts
       if z_signal(i*N_sq) >= Vth
           rec_bits(i) = 1;
       else
           rec_bits(i) = 0;
       end
    end

    %%%
```

end