Team #3
Xie HE
A43011198
4/5/2013

Application Note

Signal Processing of ECG Signals

Introduction:

The signal processing in digital is what we are considering to implement to our ECG signals as an extra function after we finish the basic objective of the project, which is only to design, simulate, fabricate, test, and demonstrate an ECG demonstration board in analog. Based on the purpose of the project, we are trying to calculate BPM (beats per minutes) by programming an ECG processing algorithm in the Stellaris Microcontroller so that the user's heart beats will be coming up with their ECG signal on the LED screen. However, the programming work in the Stellaris Microcontroller is in C language, which I'm not familiar with based on the courses I've taken these years in ECE major. Therefore, I'm looking at a substituted way which I could use to write the algorithm rather than I directly write it in C language. The MATLAB has this function called code generator that is to convert the valid MATLAB code into the C code. Comparing to directly writing C code, writing a MATLAB code is much easier and more understandable.

The purpose of the note is to descript how to design an ECG processing algorithm in MATLAB and then how to convert a valid MATLAB code into a C code.

How it works:

The basic task of electrocardiogram (ECG) processing is R-peaks detection. There are some difficulties one can encounter in processing ECG: irregular distance between peaks, irregular peak form, presence of low-frequency component in ECG due to patient breathing etc. To solve the task the processing pipeline should contain particular stages to reduce influence of those factors. Present sample demonstrates such a pipeline. The aim is to show results of processing in main pipeline stages.

Let us have some digital ECG signal — that is our input data (**fig. 1**):

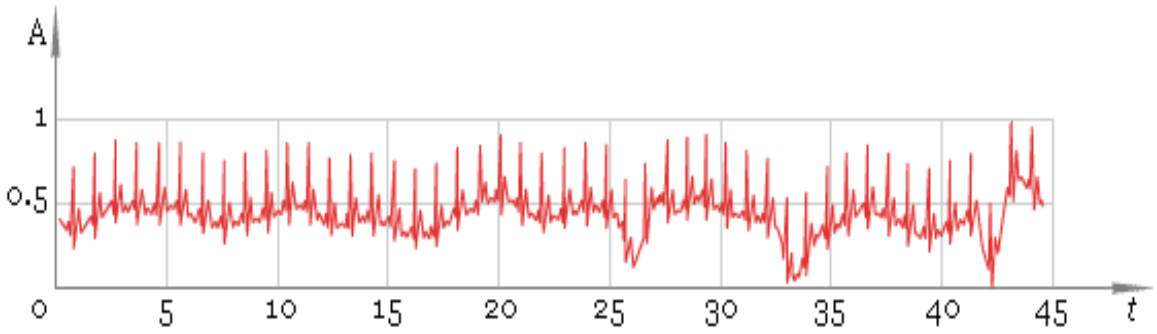


Fig. 1Original ECG

As one can see the ECG is uneven. Thus our first step is to straighten it. To say that in mathematical language, we should remove low-frequency component. The idea is to apply direct fast Fourier transforms — FFT, remove low frequencies and restore ECG with the help of inverse FFT. Here is the result of FFT processing — **fig. 2**.

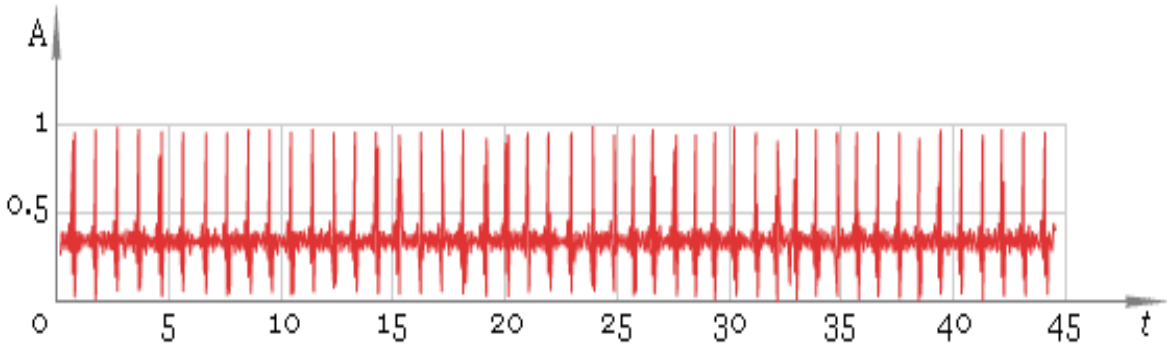


Fig. 2 FFT filtered ECG

Our second step is to find local maxima. To do that we use windowed filter that “sees” only maximum in his window and ignores all other values. On this step we use window of default size. As result we get fig. 3.

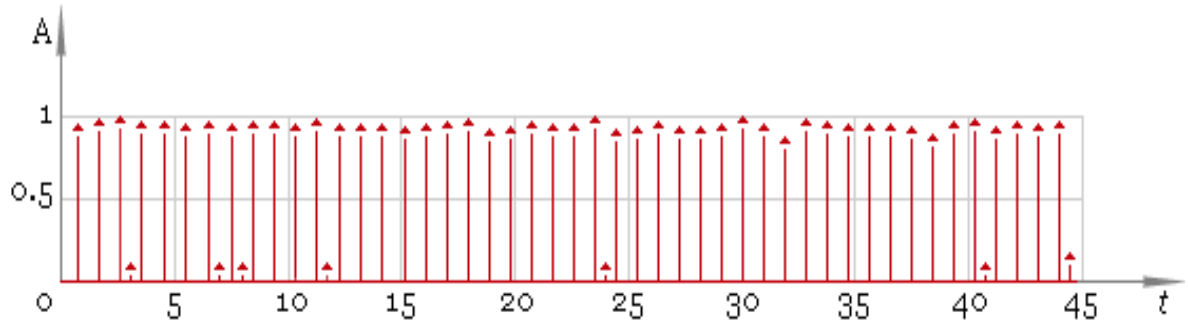


Fig. 3 Filtered ECG — first passes.

Now we should remove small values and preserve significant ones — **fig. 4**. Here we are using a threshold filter.

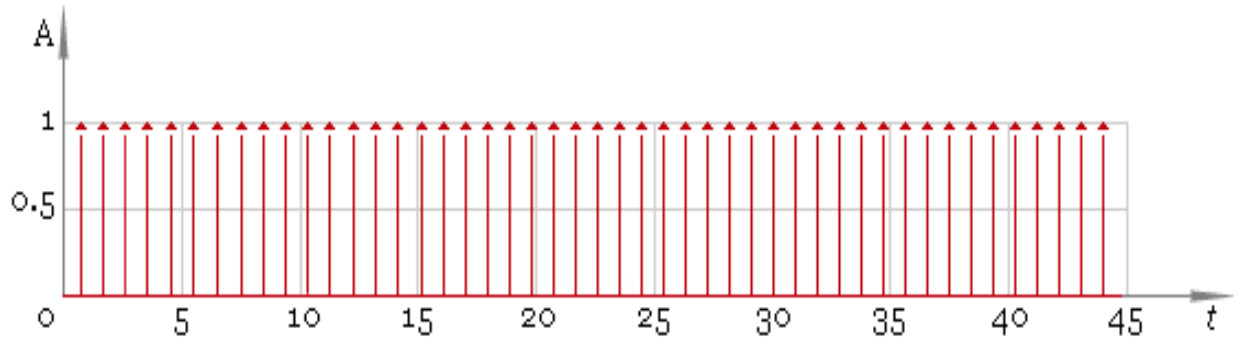


Fig. 4 Peaks.

In this case the result is good but in general case we cannot be sure we have all the peaks. So the next step is to adjust filter window size and repeat filtering — fig. 5. Compare the result with fig. 3 — now filtering quality is much better.

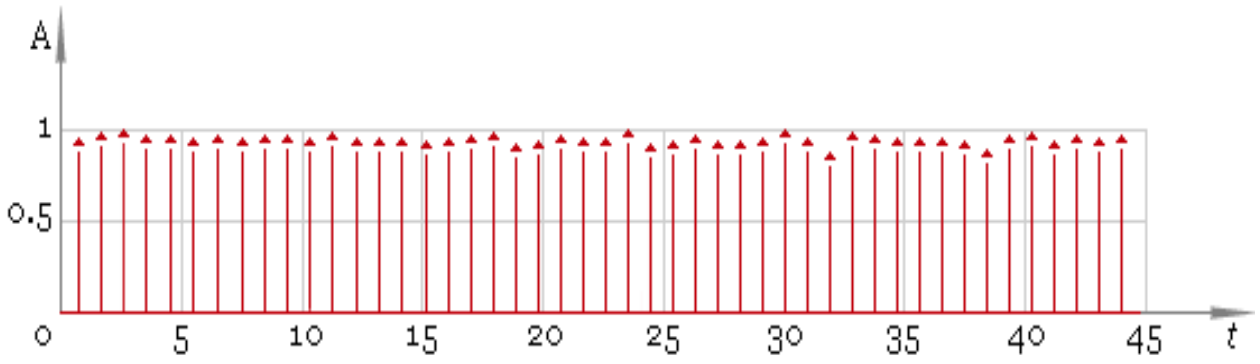


Fig. 5 Filtered ECG — second pass.

Now we are ready to get the final result and here it is: **fig. 6**.

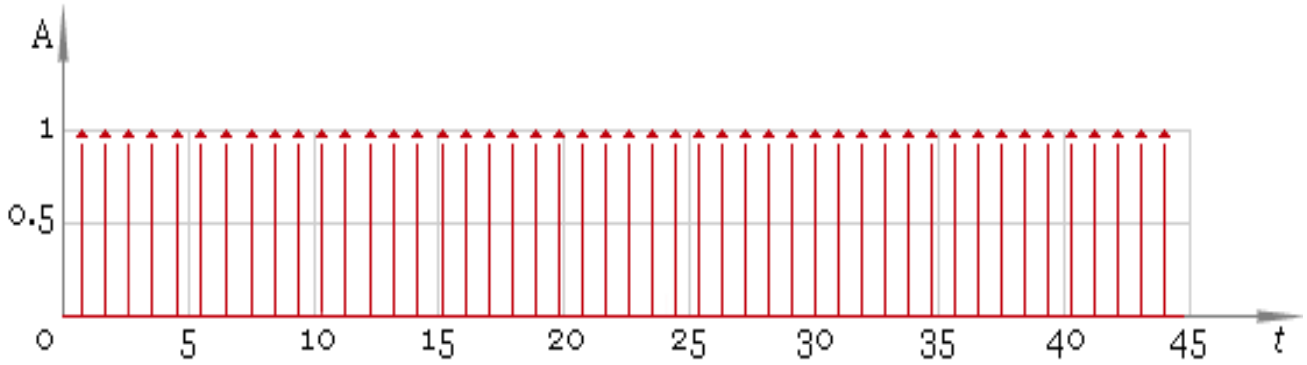


Fig. 6 Peaks — final result.

Example:

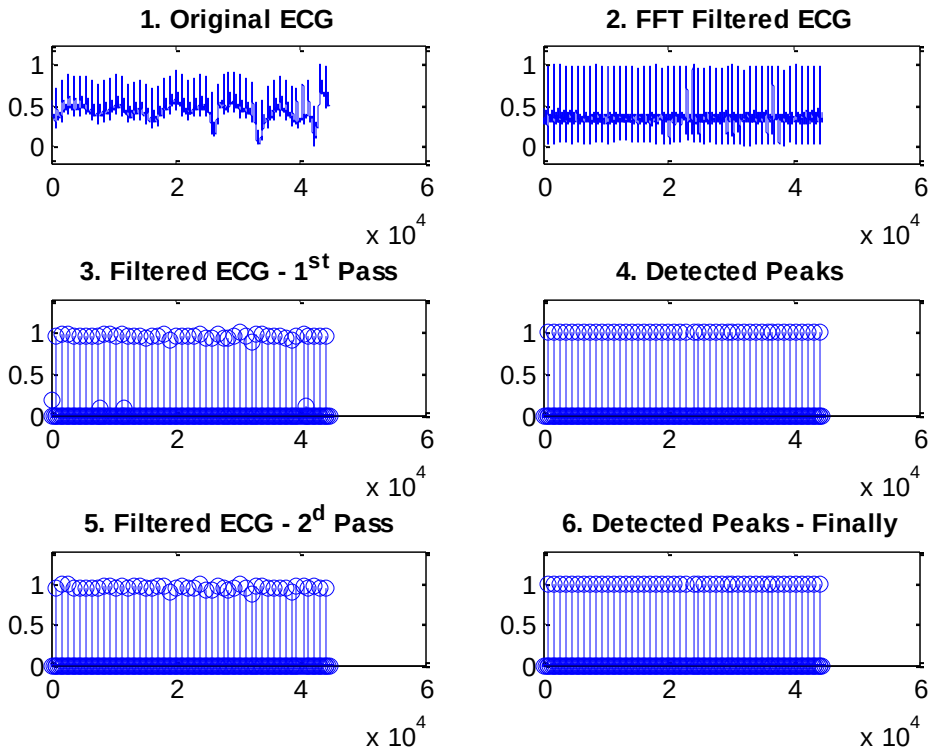
Here is a sample MATLAB code demonstrating the work above.

```
25
26      %   We are processing two data samples to demonstrate two different situations
27 -    for demo = 1:2:3
28          %   Clear our variables
29
30
31 -        clear ecg samplingrate corrected filtered1 peaks1 filtered2 peaks2 fresult
32          %   Load data sample
33 -        switch(demo)
34 -            case 1,
35 -                plotname = 'Sample 1';
36 -                load ecgdemodata1;
37 -            case 3,
38 -                plotname = 'Sample 2';
39 -                load ecgdemodata2;
40 -        end
```

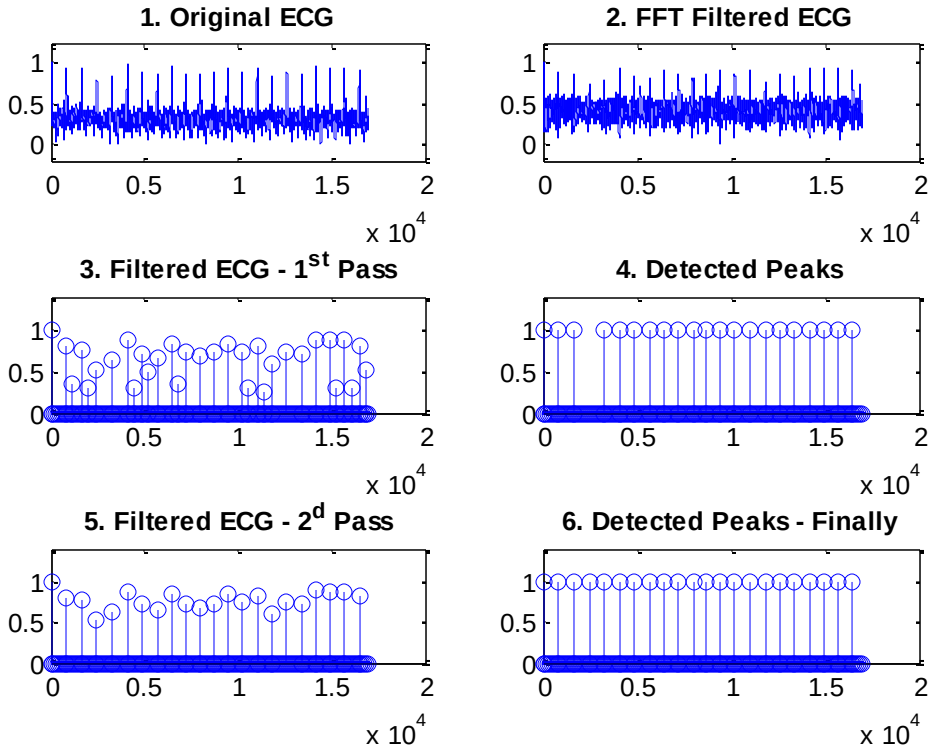
```
%   Remove lower frequencies
fresult=fft(ecg);
fresult(1 : round(length(fresult)*5/samplingrate))=0;
fresult(end - round(length(fresult)*5/samplingrate) : end)=0;
corrected=real(ifft(fresult));
%   Filter - first pass
WinSize = floor(samplingrate * 571 / 1000);
if rem(WinSize,2)==0
    WinSize = WinSize+1;
end
filtered1=ecgdemowinmax(corrected, WinSize);
%   Scale ecg
peaks1=filtered1/(max(filtered1)/7);
%   Filter by threshold filter
for data = 1:1:length(peaks1)
    if peaks1(data) < 4
        peaks1(data) = 0;
    else
        peaks1(data)=1;
    end
end
positions=find(peaks1);
distance=positions(2)-positions(1);
for data=1:1:length(positions)-1
    if positions(data+1)-positions(data)<distance
        distance=positions(data+1)-positions(data);
    end
end
% Optimize filter window size
QRdistance=floor(0.04*samplingrate);
if rem(QRdistance,2)==0
    QRdistance=QRdistance+1;
end
WinSize=2*distance-QRdistance;
% Filter - second pass
filtered2=ecgdemowinmax(corrected, WinSize);
peaks2=filtered2;
for data=1:1:length(peaks2)
    if peaks2(data)<4
        peaks2(data)=0;
    else
        peaks2(data)=1;
    end
end
%   Create figure - stages of processing
figure(demo); set(demo, 'Name', strcat(plotname, ' - Processing Stages'));
%   Original input ECG data
subplot(3, 2, 1); plot((ecg-min(ecg))/(max(ecg)-min(ecg)));
title('\bf1. Original ECG'); ylim([-0.2 1.2]);
%   ECG with removed low-frequency component
subplot(3, 2, 2); plot((corrected-min(corrected))/(max(corrected)-min(corrected)));
title('\bf2. FFT Filtered ECG'); ylim([-0.2 1.2]);
%   Filtered ECG (1-st pass) - filter has default window size
subplot(3, 2, 3); stem((filtered1-min(filtered1))/(max(filtered1)-min(filtered1)));
title('\bf3. Filtered ECG - 1^{st} Pass'); ylim([0 1.4]);
%   Detected peaks in filtered ECG
subplot(3, 2, 4); stem(peaks1);
title('\bf4. Detected Peaks'); ylim([0 1.4]);
%   Filtered ECG (2-d pass) - now filter has optimized window size
```

```
100 -       subplot(3, 2, 5); stem((filtered2-min(filtered2))/(max(filtered2)-min(filtered2)));
101 -       title('\bf5. Filtered ECG - 2^d Pass'); ylim([0 1.4]);
102         %   Detected peaks - final result
103 -       subplot(3, 2, 6); stem(peaks2);
104 -       title('\bf6. Detected Peaks - Finally'); ylim([0 1.4]);
105         %   Create figure - result
106 -       figure(demo+1); set(demo+1, 'Name', strcat(plotname, ' - Result'));
107         %   Plotting ECG in green
108 -       plot((ecg-min(ecg))/(max(ecg)-min(ecg)), '-g'); title('\bf Comparative ECG R-Peak Detection Plot');
109         %   Show peaks in the same picture
110 -       hold on
111         %   Stemming peaks in dashed black
112 -       stem(peaks2'.*((ecg-min(ecg))/(max(ecg)-min(ecg)))', ':k');
113         %   Hold off the figure
114 -       hold off
115 -   end
```

Result:



This is the result of one of two sample input signals. For this signal, it has a significant lower frequency noise curving the original signal. Comparing with fig. 2 next to it, we can see that signal is strengthen obviously after FFT and a high pass filter.



For this input signal, instead of having a much lower frequency noise, it has many close frequency noises. And if we compare it with fig.3 below it, we can see that the impulses in its spectrum are kind of mess.

For both two different original signals above, I get the final spectrums with detected peaks only. Furthermore, by using another peak detection algorithm, the heart beats can be easily recorded on this final spectrum.

Converting MATLAB code into C Code:

In MATLAB, a codegen function is used to generate C/C++ code from MATLAB code. When you include a codegen function in your MATLAB code, the MATLAB will automatically check your code all the time that whether the present context in your code is valid to be converted to a C/C++ code. A sign on right top corner indicates the state of your code. If

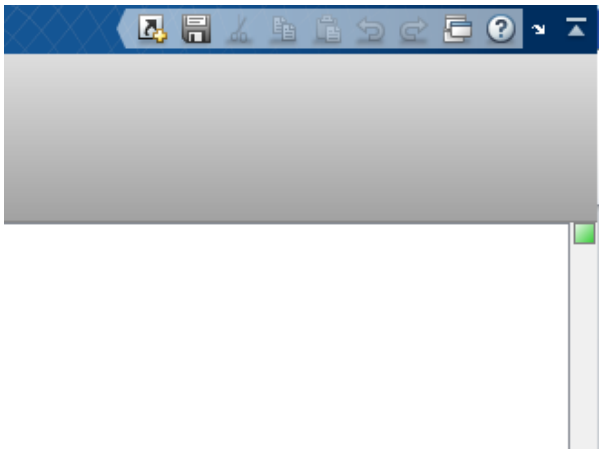
there is no warning found, it will turn green, otherwise it will become red and in this case, there much be certain instructions in your MATLAB are invalid to be converted.

Example:

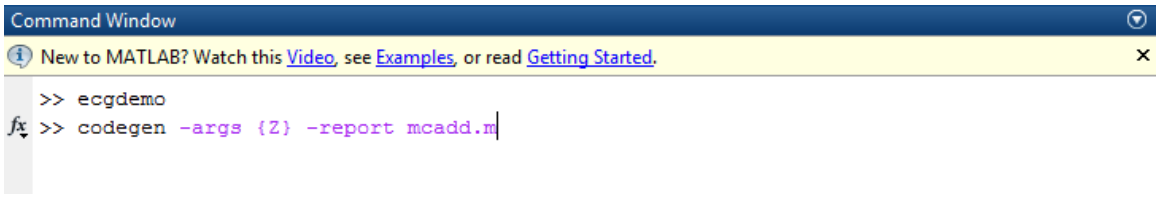
Typing %#codegen will activate the check mode.

```
1   function Hd1 = mcadd()
2   %#codegen
3
4 -  Fc    = 0.4;
5 -  N = 100;   % FIR filter order
6 -  Hf = fdesign.lowpass('N,Fc',N,Fc);
7 -  Hd1 = design(Hf,'window','window',@hamming,'SystemObject',true);
```

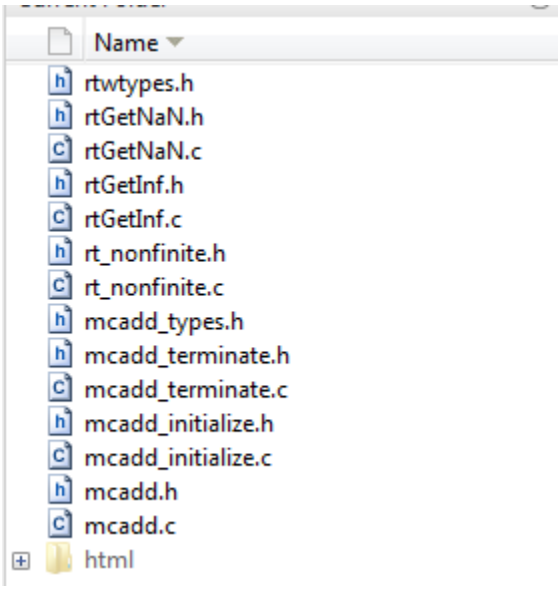
The sign indicates the state (green rectangle)



The generating command in command window



The generated C code



Conclusion:

Comparing to directly writing a complex c code with hundreds or even thousands of lines, writing a MATLAB code in MATLAB program will make the work much easier, especially for the work such as signal process. Once you get your MATLAB finished and check there is no warning, convert it to c code and get started to use it!