EE202A/CS213A Homework 4 Report

Chenguang Shen and Kevin Ting
{pscggeforce, gwkin1989}@gmail.com

1. Summary

In this homework we mainly modified the signal generation (the Mbed M3) part of homework #3, and implemented an artificial ECG waveform synthesizer. We ported the Matlab code in [1] onto the Mbed M3 to make it generating ECG signal (with some probability to generate a PVC segment). Apart from the python program controlling the sample part at the M0, we implemented another python program to control the synthesis of ECG waveform at the M3. We also modified our plotting python program significantly for this assignment. The github repo is at https://github.com/chenguangshen/MbedECGGenerator.

2. System Organization

The organization of the software is shown in Figure 1:

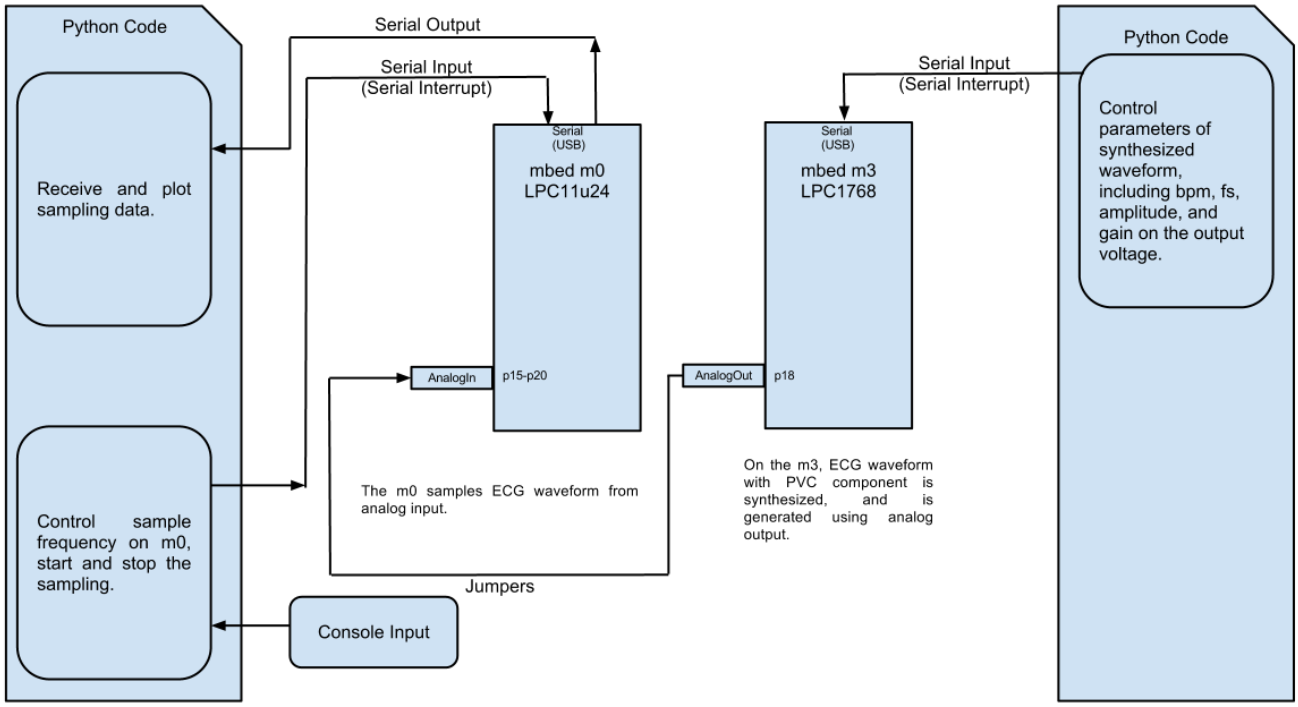

Figure 1: Software Organization

M3 Part:

We have ported the Matlab code for generating ECG waveform to the Mbed m3.

(1) Read parameters from serial input

The main() function on the m3 will register a serial interrupt function Rx_interrupt(). Upon each serial interrupt, the m3 will read bpm, fs, amplitude of the ECG waveform, and gain on the output voltage, from the serial port (data sent by Python). Then the function will call ecg_generate() function with the received parameters to synthesize an ECG waveform.

(2) Synthesize ECG waveform

The ecg_generate() function synthesizes the ECG waveform, mainly following the algorithm used by the Matlab code in [1]. The duration of the waveform is computed by (bpm / 60) * fs, and only one period of the waveform is synthesized. The waveform is stored in a buffer[] array. Note that, in order to save memory space, we didn't save all the values of the ECG waveform. Instead, we only stored those non-zero values. In fact, there are only two non-zero parts inside the ECG waveform. The first part consists of three lines (Q-R, R-?, ?-S), and the second part is the T-wave. The remaining part in the ECG waveform are all zeros. In the extreme case (fs=5000, bpm=60), there are only some 1400 non-zero data points within one period of the ECG waveform.

We used four pointers to indicate the start and end of the two non-zero parts, respectively. The same approach will be used for the PVC segment.

(3) Output the ECG waveform with PVC segment

After the ECG waveform is synthesized, the output_signal() function will be attached to a Ticker, for periodical analog output. Note that the value of the ECG waveform is multiplied by the gain, and is rescaled to [0..3.3] (since no negative voltage can be generated by mbed, a 0.1 offset is added to each value in the buffer[] array).

As the interval of the ticker is set according to the frequency of the ECG waveform (bpm / 60), the ECG waveform will be generated at the frequency. Here, we also considered the random PVC component. After each period of the ECG waveform is sent through analog output, the output_signal() function will generate a random number [0..1] to see if we need to add a PVC segment here. Here we set the probability of PVC component to be 20%:

(a) If we need to add a PVC segment here, the output_signal() function will first be detached from the ticker to stop the further output of the ECG waveform. Then the pvc_generate() function will be called to synthesize a PVC segment, and store in a PVC buffer. Finally, a new output_signal() function will be attached to the ticker for sending the PVC buffer through analog output.

The PVC segment is generated with random amplitude, and doesn't have the deadspace 0s.

(b) If we don't need to add a PVC segment here, the same output_signal() function will be attached to the ticker, continuing to send ECG buffer through analog output.

We can see that in our case, after each period of ECG waveform/PVC waveform, output_signal() function will be detached from the ticker, and a new one will be attached. Since we are dealing with low frequency here (bpm is at most ~140), this will not cause discontinuity in the output waveform.

One important issue is how to generate random number in a timely fashion. The method we are using is analog input. The result of the read() function of analog input is a floating number with 6 digits after the floating point. We are using the last two digits as a random number, since they are random enough according to our test.

M0 Part:

We changed our M0 code significantly for this assignment. On the M0 side, we use read_u16() instead of read() which increases throughput. We fill a buffer of length 100 with unsigned integers and then proceed to send each of them over as two 8 bit character by bitwise shifting. Then we call a wait function with a frequency that can be dynamically changed by the user. The frequency variable is a volatile global variable and, in our main loop, we read incoming serial data via getc and scanf to set the freq variable to whatever the user designates. Furthermore, there is another global variable 'trig' that is by default, false, and is set to true whenever the frontend app wants to receive a new block of data. The sample function, includes an if (trig == true) statement that encapsulates the read_u16() function. Therefore, data is only read and sent when trig is true. Furthermore, after each block is sent, trig is reset to be false, so it has to wait for another request from the frontend before any data is sent. Trig is set to be true every time the serial port of the M0 receives a 't' character.

Python GUI:

We stuck with the same GUI layout, but removed the pins option since we only have to sample with 1 pin. We changed the frequency text box to a period text box since we are actually changing the period by change the 'freq' variable in the M0. On every redraw timer, we enter GetSample() which starts out by flushing the serial port. Then we send a 't' through the serial port to tell the mbed to send the frontend a block of data. We then read 200 bytes (100 times 2 for 2 bytes) and unpack it using struct.unpack(). We then store it in a numpy array and scale the value to be between 0.0 and 3.3 instead of 0 to 2^16. Then we write the array to a text file so we can check to see if we receive all the values we send. Finally, we rotate a length 1000 buffer by the the length of the smaller buffer, replace the end of the large buffer by the entire 100 point buffer, and then draw the entire 1000 point buffer. This gives a rolling-effect to the plotter. We also built a simple Python GUI to send ECG synthesis parameters to the m3.

4. Reference

[1] http://www.physionet.org/physiotools/matlab/ECGwaveGen/

[2] Ruha, Antti and Seppo Nissila, "A Real-Time Microprocessor QRS Detector System with a 1-ms Timing Accuracy for the Measurement of Ambulatory HRV", IEEE Trans. Biomed. Eng. Vol. 44, No. 3, 1997.