Open Source ECG Analysis Software Documentation

Patrick S. Hamilton
E.P. Limited

Table of Contents

1.0 Introduction

Computer analysis of ECG signals is common today. Devices on the market that analyze ECGs, such as patient monitors, stress test systems, and Holter analysis systems, do a good job of detecting beats and classifying arrhythmias. However, new companies are constantly emerging and applying new technologies, such as PDAs, in an effort to make smaller and cheaper ECG analysis systems. Other companies are using ECG gating to improve imaging of the heart. Each new company that produces a product that requires ECG analysis must implement their own ECG analysis software, duplicating the efforts of every other company. Similarly, researchers who need ECG analysis to explore new diagnostic methods, such as the analysis of heart rate variability, must also implement their own versions of basic ECG analysis functions. Thirty years of research on computer analysis of ECG signals has produced a great many methods for detecting beats and classifying ECG signals, but there is still a significant effort required to go from theory to implementation.

In an effort to reduce this industry and research wide duplication of effort, E.P. Limited, with support from the National Heart Lung and Blood Institute (NHLBI) of the National Institutes of Health (NIH), has been developing and is now releasing open source software for ECG analysis. We have developed C functions that implement the most basic ECG analysis operations, detection and classification of individual beats. By building on this open source software we hope that new companies will be able to bring reliable systems to market more quickly, and researchers will be able to spend more time exploring new diagnostic techniques rather than implementing beat detectors.

This is our fifth software release and includes C implementations of three variations of our beat detector and a C implementation of a beat classifier. The beat classifier and two of the beat detector implementations can easily be adapted for different sample rates. One beat detector has been written for implementation on processors with limited available memory. Finally, we have included some programs to facilitate the testing of beat detection and beat classification software.

In releasing this software as open source code, we hope that others will make improvements on it and make those improvements available. To be realistic, we do not expect that companies will make an effort to release modifications that they make to this software. On the other hand, we believe that this software presents significant opportunities for students and others in an academic environment to build on our work and make the results of their work available as open source code. With this in mind, we would like to suggest areas where our algorithms might be improved or extended. These areas represent opportunities for course projects, undergraduate theses, or Masters theses.

Some possible improvements or extensions of the beat detection algorithm are:

- The QRS detector only uses a single channel of information (on the MIT/BIH database the second channel is often too noisy to be of much help), can it be made more robust if it uses two or more ECG channels as input?
- The QRS detector has not been tested in noisier environments. Can performance be improved with noisier signals (as might occur during exercise or stress testing) by making stronger assumptions about rhythm?
- As is, the QRS detector has a variable, sometimes long, delay. How well can a QRS detector perform if the delay is restricted to be 0.25 s or less?
- The QRS detector filters were designed to be efficient, requiring only shift and add operations rather than multiplications of any precision. If processing power is not a significant problem, are there better filters that use less restricted filter coefficients?

Though the code for the QRS detection software has been developed and improved over the course of roughly 15 years, the beat classification software has only been under serious development for a little more than one year. The performance of the classification software is as good as or better than the performance reported for other algorithms on the same data [1-5], but it is still "tuned" to the data from the MIT/BIH and AHA databases. The following are some areas where the algorithm could be improved:

- Significant improvements could be made in the software's ability to recognize and compensate for noise.
- The algorithm only uses a limited set of features to recognize whether or not a beat is ventricular and completely ignores the T-wave region of the beat. A more complete feature set might allow more accurate classification of ventricular beats.
- The algorithm only uses information from a single channel of ECG data.
- The algorithm has only been tested on half hour records. Longer records will tend to use up the available beat templates with minor beat variations and noisy beats. We do not know how this will affect performance on a long record.
- The algorithm does not recognize Ventricular Fibrillation or Flutter (as on 207)

Further development of this software requires that the developer continually test the software against as large a database of ECG data as possible. It is not of much use to improve the performance on one record if it requires sacrificing performance on all the others. Luckily, it has never been easier or less expensive to test ECG analysis software on MIT/BIH data and data from supplemental databases. ECG data and open source software for accessing this data are available on line at www.physionet.org. In this release, we have provided two example programs (easytest and bxb) to facilitate testing beat detection and classification software with MIT/BIH formatted data.

The following documentation provides an overview of the software included in this release and documents the general theory of operation for the various software components. If anything seems unclear you can always refer directly to the source code. We would appreciate any improvements to, comments about, corrections to, or suggestions regarding this documentation or software. We will make an effort to test and incorporate all future improvements in the distribution software.

2.0 Software Overview

Our software implements the basic ECG analysis functions of beat detection and classification as C functions. This release includes three versions of our beat detector. Two are general-purpose beat detectors, where one represents a more efficient version with slightly different performance characteristics. The third beat detector is more computationally efficient, uses very little memory, and is embedded in a program that performs beat detection on a PIC processor. The beat detector functions are independent of the beat classification functions and may be used alone in applications that do not require beat classification.

2.1 File Listing

All the files required for detecting and classifying beats and this documentation should be included in the file osea121.zip (Open Source ECG Analysis 1.21). The code is written in C and the files are .cpp files so that our complier (Turbo C++ V 4.5) treats them as C++ files (the only C++ feature we actually use is the // comment). We wrote the Picqrs.c to be compiled with the CCS C compiler for PIC processors. The following is a list of the files that make up the open source ECG analysis software release:

QRS Detection Files:
Qrsdet.cpp	Original QRS detection functions.
Qrsdet2.cpp	Simplified and improved QRS detector functions.
Qrsfilt.cpp	Filter functions for QRS detection.
Qrsdet.h	General parameters definitions associated with QRS detection.
Picqrs.c	A PIC based QRS detector.

Beat Classification Functions:
Analbeat.cpp	Functions for extracting beat features.
Analbeat.h	Function prototypes associated with analbeat.cpp.
Bdac.cpp	API function for beat detection and classification.
Bdac.h	General parameters associated with the beat classification software.
Classify.cpp	Functions for classifying beats.
Match.cpp	Functions for matching beats to previously detected beat types.
Match.h	Function prototypes associated with match.cpp.
Noisechk.cpp	Functions for detecting low frequency noise.
Postclas.cpp	Functions associated with post classification**.**
Rythmchk.cpp	Functions for classifying R-to-R intervals.
Rythmchk.h	Function prototypes associated with rythmchk.h.

Test programs:
Easytest.cpp	Example program that creates MIT/BIH format annotation files from ECG data using the beat detection and classification software.
Bxb.cpp	Example program that compares annotation files to evaluate beat detection and classification performance.

Documentation:
Osea3.pdf	This document.

Database I/O:

Wfdb.lib   Library of database interface functions needed for easytest.cpp and bxb.cpp.

Wfdb.h   Header file for functions and structures in wfdb.lib

Ecgcodes.h   ECG classification code definitions.

2.2 File Dependencies

Fig. 1 illustrates the file dependencies for the beat detection and analysis files. **BDAC** contains the function *BeatDetectAndClassify*, which serves as the interface between input data and beat detection and classification. *BeatDetectAndClassify* takes sample-by-sample data as input and passes it to the QRS detection functions in **QRSDET** or **QRSDET2**. The QRS detection function uses filter functions in **QRSFILT**. When a beat is detected, *BeatDetectAndClassify* waits until the end of the beat, checks for low-frequency noise, and passes the beat, R-to-R interval, and noise level to the function *Classify* in the file **CLASSIFY**.

Classify decides whether a beat is a normal beat or a ventricular beat. *Classify* uses functions in **MATCH** to compare new beats to templates of previously recorded beats. Functions in **RYTHMCHK**, classify the beats R-to-R interval as either a normal R-to-R interval, a premature R-to-R interval, or an R-to-R interval that represents a compensatory pause. Functions in **ANALBEAT** determine the onset and end point of the QRS complex and the associated isoelectric level. Functions in **POSTCLAS** attempt to more accurately reclassify previous beats. Classify uses this information on whether the beat matches previous beats, whether or not the R-to-R interval appears premature, whether the beat is narrow or wide, and whether this type of beat has previously appeared to be ventricular or normal to classify the beat. Classify returns its classification to BDAC which, in turn, returns the classification to the calling program.

```
BDAC

    ├─   QRSDET or QRSDET2
    │        └─   QRSFILT
    ├─   NOISECHK
    └─   CLASSIFY

             ├─   MATCH
             ├─   RYTHMCHK
             ├─   ANALBEAT
             └─   POSTCLAS
```
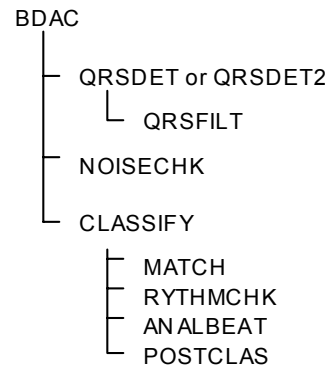
Fig. 1 Overview of ECG analysis software file and functional dependencies.

2.3 Beat Detection and Classification API

There are only two functions required for using the beat detection and classification software, **BeatDetectAndClassify** and **ResetBDAC**. The first function provides the interface to the beat detection and classification software and the second allows the static variables required for beat detection and classification to be reset. If your application does not require beat classification, the function **QRSDet** can be used to interface directly to the QRS detection portion of the software.

The following are function definitions for **BeatDetectAndClassify**, **ResetBDAC**, and **QRSDet**. For details on using **BeatDetectAndClassify**, refer to easytest.cpp. We have included easytest.cpp in this software distribution as an example program that uses **BeatDetectAndClassify** to generate annotation files from MIT/BIH arrhythmia database records.

2.3.1 BeatDetectAndClassify

Syntax:

    int BeatDetectAndClassify(**int** *ecgSample*, **int** **beatType,* **int** **beatMatch*) ;

Description:

    BeatDetectAndClassify implements a beat detector and classifier. ECG samples are passed into *BeatDetectAndClassify* one sample at a time with *ecgSample*. When a beat has been detected and classified the detection delay is returned and the beat classification is returned through the pointer **beatType*. The index of the template that the beat was matched to is returned through the pointer **beatMatch* (used for debugging). The calling function must allocate space for *beatType* and *beatMatch*.

Returns:

    BeatDetectAndClassify returns 0 if no new beat has been detected and classified. If a beat has been classified, *BeatDetectAndClassify* returns the number of samples since the approximate location of the R-wave. If a beat has been classified, **beatType* is set to 1 (NORMAL), 5 (PVC), or 13 (UNKNOWN) to indicate the beat classification. The index of the previous beat template that the beat most closely matches is returned in **beatMatch*.

2.3.2 ResetBDAC

Syntax:

    void ResetBDAC(void) ;

Description:

    ResetBDAC resets static variables used in beat detection and classification. *ResetBDAC* should be called before beginning to process a new ECG record with *BeatDetectAndClassify*.

2.3.3 QRSDet

Syntax:

    int QRSDet(int *ecgSample*, int *init*) ;

Description:

    QRSDet implements a modified version of the QRS detection algorithm described in:

        Hamilton, Tompkins, W. J., "Quantitative investigation of QRS detection rules using the MIT/BIH arrhythmia database", IEEE Trans. Biomed. Eng., BME-33, pp. 1158-1165, 1987.

    Consecutive ECG samples are passed to *QRSDet* in *ecgSample*. *QRSDet* contains a number of static variables that it uses to adapt to different ECG signals. These variables can be reset by passing a non-zero value in *init*.

Returns:

    When a QRS complex is detected *QRSDet* returns the detection delay.

3.0 Beat Detection

Fig. 2 shows the basic operations of the beat detection algorithm. The beat detection algorithm can be broken down into two sections, the filters and the detection rules. Consequently, the implementation is broken down into two files **QRSFilt.cpp** which contains the filter implementations and **QRSDet.cpp** which contains the detection rule implementation. Our present release contains three versions of our QRS detector. All three versions use the same filters and differ primarily in the details of the detection rules and code implementations.

3.1 The Filters

We filter the signal to generate a windowed (time limited) estimate of the energy in the QRS frequency band. We do this by:

1. Low pass filtering,
2. High pass filtering,
3. Taking the derivative,
4. Taking the absolute value of the signal, and
5. Averaging the absolute value over an 80 ms window.

The final filter output produces what might be called a lump every time a QRS complex occurs. T-waves generally produce smaller lumps than QRS complexes.

ECG → QRSDet.cpp → QRSFilt.cpp → Low-pass Filter (~16 Hz cutoff) → High-pass Filter (~ 8 Hz cutoff) → |d[ ]/dt| → 80 ms Moving Average → Peak Detection → Detection Rules → QRS Detections

Fig. 2 : Beat detection operations

These operations are implemented with the functions *lpfilt*, *hpfilt*, *deriv1* or *deriv2*, *abs*, and *mvwint* in **QRSFilt.cpp**. The high pass, low pass, and derivative combine to produce a bandpass filter with a pass band from 5 to 11 Hz, roughly the bandwidth that contains most of the energy in the QRS complex. The theory and implementation of these filters are detailed in [6], [7], and *Biomedical Digital Signal Processing* [8]. However, there are errors in the filter descriptions in [6]. In [6] and [7], the filtered signal was squared rather than rectified. This operation caused the QRS detector to be somewhat gain sensitive. In this implementation we have used the absolute value, reducing the gain sensitivity and slightly improving the performance of the algorithm.

The averaging window was chosen to be roughly the width of a typical QRS complex. In the original algorithm this window was 150 ms wide to allow for the wide QRS complexes produced by Premature Ventricular Contractions (PVCs). Since then, it has been shown that a narrower window produces better results [9].

3.2 Detection Rules

After the signal has been filtered, *QRSDet* detects peaks in the signal. Each time a peak is detected it is classified as either a QRS complex or noise, or it is saved for later classification. The
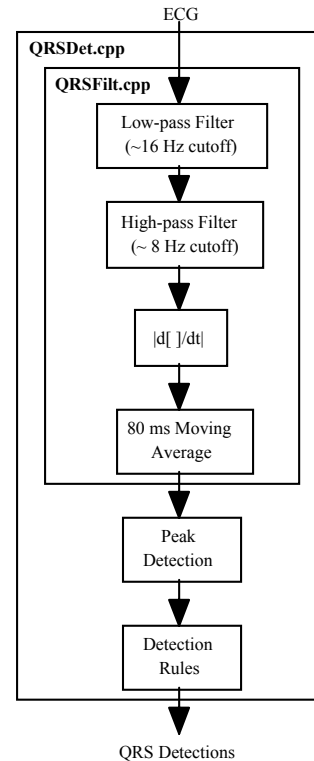
6

algorithm uses the peak height, peak location (relative to the last QRS peak), and maximum derivative to classify peaks.

The following is an outline of the basic detection rules for the algorithm.

1. Ignore all peaks that precede or follow larger peaks by less than 200 ms.
2. If a peak occurs, check to see whether the raw signal contained both positive and negative slopes. If not, the peak represents a baseline shift.
3. If the peak occurred within 360 ms of a previous detection check to see if the maximum derivative in the raw signal was at least half the maximum derivative of the previous detection. If not, the peak is assumed to be a T-wave.
4. If the peak is larger than the detection threshold call it a QRS complex, otherwise call it noise.
5. If no QRS has been detected within 1.5 R-to-R intervals, there was a peak that was larger than half the detection threshold, and the peak followed the preceding detection by at least 360 ms, classify that peak as a QRS complex.

The rules as outlined above are implemented in QRSDET and detailed in [6] and [7]. However, in exploring simpler implementations, we found that the using slope to discriminate T-waves from QRS complexes did not improve performance, so this feature (rule 3) of the algorithm has been eliminated in QRSDET2 and the PIC implementatation.

3.3 Threshold Estimation

The detection threshold used in 4 and 5 above is calculated using estimates of the QRS peak and noise peak heights. Every time a peak is classified as a QRS complex, it is added to a buffer containing the eight most recent QRS peaks. Every time a peak occurs that is not classified as a QRS complex, it is added to a buffer containing the eight most recent non-QRS peaks (noise peaks). The detection threshold is set between the mean or median of the noise peak and QRS peak buffers according to the formula:

$$\text{Detection\_Threshold} = \text{Average\_Noise\_Peak} + TH*(\text{Average\_QRS\_Peak} - \text{Average\_Noise\_Peak})$$

where *TH* is the threshold coefficient. Similarly, the R-to-R interval estimate used in 5 is calculated as the median or mean of the last eight R-to-R intervals.

Originally, we estimated average QRS peak values, noise peak values, and average R-to-R intervals using the median of the last eight values. When we recently implemented a simpler QRS detector using the mean of the previous eight values rather than the median, we found that using mean estimates improved the QRS detector performance. However, when coupled with the beat classification software, using means to estimate average peak values and R-to-R intervals produced a slight decrease in the performance of the beat classifier (12 more incorrect classifications on the MIT/BIH data and 5 on the AHA data). In this release, the functions in QRSDET use median and functions in QRSDET2 use mean for average peak values and R-to-R interval estimation.

The beat detector must begin with some initial threshold estimate. In order to make an initial estimate, we detect the maximum peaks in eight consecutive 1-second intervals. These eight peaks are used as are initial eight values in the QRS peak buffer, we set the initial eight noise peaks to 0, and we set the initial threshold accordingly. We initially set the eight most recent R-to-R intervals to 1 second.

It is possible (but rare) for a string of large ventricular beats to cause the detection thresholds of the beat detector to increase so much that low amplitude normal beats that follow are ignored.

In practice this only occurred on one AHA record and none of the MIT/BIH records. To avoid this problem,when no beats are detected for eight seconds, the detection thresholds are reset in the same way that they were initialized.

3.4 Detection Delay

When a beat is detected by *QRSDet*, *QRSDet* returns the number of samples that have occurred since QRS complex occurred. *QRSDet* does this because the detection delay will vary. If a peak is detected using rule 4 above, the detection delay will be roughly the sum of the filter delays, the moving window integration width, and the 200 ms delay required for rule 1 for a total delay of 395 ms. If the peak is found using rule 5 (also called a search back detection), the delay will be this delay plus half the average R-to-R interval. Consequently, the detection delay can easily vary, depending on heart rate and detection rule, from 395 ms to as much as 1 second.

Many ECG monitoring applications require shorter detection delays than our QRS detector. The relatively long, variable, detection delay can be a problem for some applications. For example, the heart beat indication for a patient monitor should coincide more closely (roughly 0.25 s) with the pulse that can be felt at the wrist.

The algorithm will produce a fixed delay if the search back detections are eliminated by simply not reporting detections that result from search back. If this is done and the detection threshold is adjusted, the algorithm will still have QRS detection sensitivities and positive predictivities near 99.7%.

With the search back detections disabled the algorithm will have a fixed detection delay of 395 ms. The majority of this delay occurs because after each peak is detected the algorithm waits 200 ms to insure that the peak is not followed by a larger peak within the QRS refractory period. Earlier versions of the algorithm simply ignored all peaks for 200 ms following a QRS detection. This allowed large P-waves (primarily on record 108) to be detected as QRS complexes, and the following QRS complexes were ignored because they occurred within the 200 ms refractory blanking period.

In order to revert to simple 200 ms blanking following a QRS detection, include the statement:

#define POST_BLANK

at the top of qrsdet.h.

Ignoring search back detections and reverting to simple 200 ms blanking following a QRS detection reduces the detection delay to 195 ms at the expense of reduced sensitivity and positive predictivity. With the detection threshold adjusted to minimize the number of false detections, the sensitivity drops to 99.2% and the positive predictivity drops to 99.5%. A better approach to the design of a short detection delay algorithm would be to use the full algorithm to set thresholds for a simple, short delay detection scheme.

3.5 A PIC16F877 QRS detector

We were curious about whether we could fit our QRS detector into a microcontroller with limited resources such as the PIC16F877. Our algorithm was originally developed [6] to run on a Z80 processor running at 4 MHz, the filters are implemented so that they only require shift and add operations rather than multiplications of any precision, so we had little doubt whether the algorithm would be efficient enough to run in realtime. Lack of RAM in the microcontroller turned out to be a more critical limitation, necessitating some algorithm modifications. In the process of investigating the effect of simplifying the QRS detector, we discovered a simplified

version of the QRS detector actually performed better than the previous version of the detector. We incorporated these simplifications into a QRS detector that is not specific to the PIC16F877 in QRSDET2.CPP.

3.5.1 Algorithm Modifications for PIC implementation.

The PIC16F877 can only access 368 bytes of on board RAM. Because of this limited amount of RAM there is no memory space in the 16F877 for the buffer of previous derivative values necessary for distinguishing T-waves. After removing the T-wave discrimination, we discovered that the beat detector performance on the MIT/BIH and AHA databases actually improved as long as noise detections in the T-wave range (between 200 and 360 ms following a QRS) were not used as search back candidates.

We also found that performance improved when the algorithm used the mean QRS peak, noise peak, and RR interval rather than the median of these values for threshold calculation and search back. Though it was possible to implement an efficient running median, it was much easier to implement mean calculations on the 16F877. We had not examined the assumption that median was better than mean for some time, and, apparently, algorithm changes that we have made in the interim have made median calculation unnecessary.

Finally, we modified the filter and QRS detector implementations to make the PIC based QRS detector more efficient. Array operations are not executed very efficiently with the 16F877 and the CCS C compiler, and we found that the code executed much faster when we avoided the use of arrays whenever possible.

3.5.2 Implementation and Testing

We implemented a version of our QRS detector on a PIC 16F877 processor running at 3.6864 MHz. The source code for this implementation is contained in PICQRS.C. We input ECG samples to the PIC using its asynchronous serial port at 19.2 kbaud, and the QRS detector returned detections using the same serial port. This allowed us to directly test the PIC-based detector with data from the MIT/BIH arrhythmia database. Implementing a useful hardware beat detector would require substituting A-to-D input from an ECG amplifier for the serial input data.

When implemented with the CCS PCM C compiler (version 3.093) the QRS detector program required 290 bytes of RAM (79% of available RAM) and 2721 bytes of ROM (33% of available ROM).

We were able to transmit ECG data and receive output from the PIC QRS detector at an average rate of 631 samples per second. Considering that the processed data was actually sampled at 200 samples per second, this indicates that the 16F877 was able to process ECG data three times faster than real time. However, this processing rate was probably more limited by our transmission strategy than by processing time. To allow Windows to run while we were running data through the processor we transmitted bursts of 35 samples every 50 ms. The typical processing time for a single ECG sample was 1.09 to 1.18 ms. So, a 16F877 running at 3.6864 MHz and sampling data every 5 ms. would only use approximately 23% of its processing power for beat detection.

The following tables list our test results using the PIC based algorithm. Rather than test the PIC based algorithm directly on the PIC processor (which would have required roughly eight hours for the MIT/BIH data), we ran the PIC C code on a PC. Before testing we ran the PIC and the PC side by side on select records to confirm that the two implementations produced identical beat detections.

The performance of the PIC based detector is comparable to that of other variations of our QRS detector, achieving a sensitivity and positive predictivity of 99.80 on the MIT/BIH ECG

data. The algorithm produced a significant number of false detections on record 7210 of the AHA database. The annotations on this record place the detection marks early on the beat, while our algorithm places detection marks on the end of the beat. Because the beats are wide, a significant number of late detections produced both false negative and false positive detections. Disregarding these detections, the performance of the algorithm was in line with our previous algorithm versions.

3.6 Beat Detection and EC13

The Association for the Advancement of Medical Instrumentation (AAMI) has developed a standard for the performance of cardiac monitors, heart rate meters, and alarms. This standard is known as ANSI/AAMI EC13:1992 (American National Standard: Cardiac Monitors, Heart Rate Meters, and Alarms) and is available from AAMI (www.aami.org). EC13 addresses a wide range of issues such as patient electrode connections, devise labeling, defibrillator overload protection, and display bandwidth. A portion of the standard specifies minimum performance standards for QRS detection in these devices.

While our original beat detector worked well on the MIT/BIH database and the AHA database, until recently, we had never checked to see whether it met all the requirements of EC-13. When we tested the algorithm against the EC13 requirements, we found that the algorithm required a few adjustments, but these adjustments did not significantly affect the database performance of the algorithm.

3.6.1 EC13 QRS Detector Requirements

The EC13 standard specifies a representative QRS and T-wave waveform and requires that QRS detectors reliably detect:
- QRS complexes with amplitudes ranging from 0.5 to 5 mV and widths ranging from 70 to 120 ms;
- QRS complexes in the presence of 100 uV peak-to-peak 60 Hz noise; and
- QRS complexes in the presence of 0.1-Hz, 4 mV peak-to-peak triangle waves.

EC13 specifies that QRS detectors should not detect:
- QRS complexes with amplitudes of less than 0.15 mV, and
- 1 mV QRS complexes with widths less than 10 ms.

Additionally, EC13 specifies that QRS detectors should accurately detect QRS complexes at rates as high as 250 beats-per-minute (bpm), and should not report heart rates lower than 250 bpm when rates as high as 300 bpm occur. Finally, manufactures should test the detectors tolerance for tall T-waves and report the amplitude of the tallest T-wave, following a 1mV, 100 ms QRS complex, that does not produce a false detection.

3.6.2 EC13 Testing

We generated a set of MIT/BIH database files with the test waveforms specified by EC13, and tested our beat detector on these waveforms. These files are:

Ec13_1 – EC13_27:	Simulated ECGs with amplitudes from 0.5 to 5 mV, widths from 70 to 120 ms, and rates from 30 to 200 bpm.
EC13_28:	Simulated ECG at 300 bpm.
EC13_29:	Simulated ECG with QRS amplitude of 0.5 mV, QRS width of 100 ms, rate of 80 bpm, and 60 Hz noise with a p-p of 400 uV.

EC13_30:	ECG with a QRS amplitude of 0.5 and width of 100 ms and 4 mV p-p triangle wave with a frequency of 0.1 Hz.
EC13_31-EC13_39:	ECG with amplitude of 0.15 mV; widths of 70, 100 and 120 ms; and rates of 30, 80, and 200 bpm.
EC13_40,EC13_41:	ECG with a QRS amplitude of 1 mV and a QRS width of 10 us at 30 and 200 bpm.
EC42:	ECG with QRS amplitude of 1 mV, QRS width of 100 ms, and a T-wave that increases from 0.2 to 1.2 mV in amplitude.

These files are available on in a separate download from our source code. We simulated these signals at a sample rate of 1000 Hz to facilitate down sampling to lower sample rates, and tested our algorithm at a sample rate of 200 Hz.

3.6.3 Algorithm Modifications for EC13

We ran the simplified algorithm (QRSDET2) on the EC13 test records. The algorithm had no trouble reliably detecting all the QRS complexes at rates of up to 200 bpm. The algorithm passed the drift tolerance test (triangle wave noise) and the 60 Hz noise tolerance tests without modification. In fact the algorithm tolerates 60 Hz noise with peak-to-peak amplitudes as large as 1 mV. Because the tall T-wave test allows the T-wave to gradually increase from 0.2 mV to 1.2 mV, the algorithm adapts to the changes in T-wave amplitude and ignores simulated T-waves as large as 1.2 mV. The algorithm, however, required modifications to reliably detect QRS complexes at 300 bpm. The algorithm also required modifications to avoid detecting low amplitude and narrow QRS complexes.

At 300 bpm, there is only 200 ms between QRS complexes, so beats fall within the blanking window of surrounding beats. The algorithm would detect one QRS complex, but ignore the next because it occurred within 200 ms, thus skipping every other beat. The resulting heart rate would be reported as 150 bpm, violating the EC13 specification. To avoid this problem, we reduced the blanking interval from 200 ms to 195 ms.

Our original algorithm had no lower limit on acceptable beat amplitudes. EC13 specifies a lower limit on beat amplitudes to prevent ECG monitors from reporting beat detections when only noise is present but the patient's heart is not beating. Similarly, narrow pulses, as might be produced by a pacemaker are attenuated by the beat detection filters, but they can still produce large enough lumps in the filtered signal to cause beat detections. The 150 mV QRS, 80 ms, QRS complexes produced peaks in the filtered signal of 4 with an input resolution of 5 uV/LSB. The 10 ms, 1 mV QRS complexes produced peaks in the filtered signal of 7 with an input resolution of 5 uV/LSB. We set the lower detection threshold limit to 7 so that the algorithm would ignore these signals.

After modifying the algorithm to pass the EC13 requirements we adjusted the detection threshold coefficient to minimize the total false negative and false positive detections on the MIT/BIH database. With a threshold coefficient of 0.3125, the algorithm had a sensitivity of 99.77% and a positive predictivity of 99.79%. Without modification, the algorithm's sensitivity and positive predictivity were 99.81% and 99.81%, only slightly better than the performance for the EC13 conforming algorithm.

3.7 Beat Detection Performance

When applied to channel 1 of the MIT/BIH arrhythmia database with the data resampled at 200 samples-per-second and a 5-minute interval for initialization, the detector produces 235 false positive detections and 176 false negative detections. This represents a sensitivity of 99.74% and

a positive predictivity of 99.80% . Similarly, the beat detector had a sensitivity of 99.74% and a positive predictivity of 99.76% when tested on the AHA ECG database. When QRS detection is performed as part of beat classification, the results are slightly different because, as part of beat classification, the beat classification software makes a more careful estimate of the fiducial mark or R-wave location.

This QRS detector has been tuned to 200 sample-per-second data from the MIT/BIH database. How it performs on other ECG databases depends on the database, but it should work very well under normal conditions. The algorithm has not been rigorously tested under the type of noise conditions that might occur during a stress test. It is my experience that under noisy conditions, algorithms perform better if they rely more heavily on the assumption that the R-to-R interval is regular. Because this assumption is not true for many of the MIT/BIH database tapes, this algorithm only makes loose assumptions about regularity.

4.0 Beat Classification

When a beat is detected it is classified as either a Premature Ventricular Contraction (PVC), a normal beat, or an unknown beat type. As mentioned previously a PVC represents any beat of ventricular origin, a normal beat is any contraction of non ventricular origin, and the unknown classification is used when the beat classifier has not been running long enough to make a decision.

Our beat classification strategy is probably the same as that used for most commercial arrhythmia detection systems. When a beat is detected it is characterized by a number of features such as width, amplitude, and R-to-R interval. The beat is also compared to previously detected beat types. If the beat closely matches a previously classified beat type, the beat takes on the classification of its matching type. Otherwise, the beat is classified based on its beat features, the features of the beat type that the algorithm judges to be the dominant normal type, and the general signal conditions. The following documentation details the beat features used for classification, our methods for matching beats to previously detected beat types, our rules for selecting the dominant normal beat types, and the classification rules that we use.

4.1 Beat And Signal Features

Our beat classification algorithm only uses a small number of features to characterize individual beats. These include width, rhythm classification, similarity to the dominant beat shape, and noise content. The width and rhythm classification are considered relative to the width and rhythm classification that applies to the dominant beat type. Beat widths and polarities are estimated by the function **AnalyzeBeat** in analbeat.cpp, rhythm classification is handled by the function **RhythmChk** in rythmchk.cpp, low-frequency noise is estimated by the function **NoiseCheck** in noisechk.cpp, high-frequency noise estimation is handled by **HFNoiseCheck** in classify.cpp, and shape similarity is handled by either **DomCompare** or **DomCompare2** in match.cpp

4.1.1 Beat width

Beat width refers the width of the QRS complex.   Width is a fundamental feature used for classifying beats. Normal beats tend to be narrower (less than 100 ms) than ventricular beats, and ventricular beats tend to be wider.

We estimate the QRS onset and offset points two different ways. First we search out from the fiducial point (detection point from the beat detector) until we find a region that precedes the QRS complex and a region that follows the QRS complex where at least 5 samples are within a 100 uV range. This gives us a first estimate of the QRS onset and offset. We then estimate QRS onset and offset based on the slopes of the QRS complex.

In obtaining a slope based estimate, we first search the QRS complex for maximum and minimum slopes. We use ¼ of the smaller magnitude of the maximum positive and maximum negative slopes as a slope threshold. Searching out from the center of the QRS complex we estimate onset and offset points as points where at least three samples have slopes with magnitudes that are less than the slope threshold.

If the isoelectric onset and offset estimates are within 25 ms of the slope based onset and offset estimates, we use the isoelectric onset and offset estimates, otherwise we use the slope based onset and offset estimates. Finally, if the offset is significantly lower that the onset, we search for a maximum slope between the offset point and the end of the beat and move the offset to a point where the slope is half that value.

4.1.2 Rhythm Classification

During a regular rhythm, ventricular beats often occur prematurely. Actually, the coupling interval between a normal beat and a ventricular beat tends to be fixed for ventricular beats that originate from a given site. If the normal R-to-R interval is less than the coupling interval, the ventricular beats do not tend to occur because the normal contraction occurs first. If the normal R-to-R interval is longer than the coupling interval, the ventricular beat occurs earlier than a normal beat would have occurred.

As a feature of each beat, we characterize the interval from the previous beat to the present beat as either NORMAL, PVC, or UNKNOWN. A classification of NORMAL indicates that the beat was either part of a regular rhythm, or it was a normal beat that followed a ventricular beat. A rhythm classification of PVC indicates that the beat was either a premature contraction following a normal contraction or a ventricular beat that is part of a string of ventricular beats. A rhythm classification of UNKNOWN indicates that the present rhythm is so irregular that it is unclear whether the present beat is premature or not.

In the function that classifies R-to-R intervals, intervals are actually characterized as NN for two consecutive normal beats, NV for a normal beat followed by a ventricular beat, VN for a ventricular beat followed by a normal beat, VV for a pair of ventricular beats, and QQ for an interval where it is not clear from the rhythm what either beat represents. NN and VN intervals are represent NORMAL intervals, NV and VV intervals represent PVC intervals, and QQ intervals represent UNKNOWN intervals.

We consider a string of beats to be part of a regular rhythm when at least four intervals occur where each pair of consecutive intervals matches. For rhythm classification we consider that two intervals match when they are within 12.5% of the average of the two intervals.

We consider an interval to be an NV interval when the interval follows an NN interval and is less than 75% of the preceding interval.

Following an NV interval, the next interval is classified as VV if it matches the previous NV interval. The next interval is classified as VN if it is longer than the previous interval.

Our rhythm classification function also recognizes VN and NV intervals when they are part of a bigeminal rhythm (ventricular beats alternating with normal beats). The classification function assumes that bigeminy is occurring when over six intervals alternate intervals match but consecutive intervals do not match. The shorter intervals are then classified as NV and the longer intervals are classified as VN.

Finally, the rhythm classification function recognizes the rhythm pattern NNVNNNV when the previous interval was classified as QQ. This pattern is characterized two matching NN intervals, two matching NV intervals that are shorter than the NN intervals, and two VN intervals that are longer than the NN intervals.

The rhythm classification could be extended to recognize other complex rhythm patterns in the QQ state (such as NNVNNV), this would allow more rapid initialization of the rhythm classification when PVCs are occurring frequently, but it has the potential to produce false rhythm classifications during irregular rhythms such as atrial fibrillation of flutter.

4.1.3 Low Frequency Noise

We try to characterize the lower frequency noise by examining the signal between the end of the T-wave of the previous beat and the beginning of the P-wave of the beat being classified. Low frequency noise generally results from movements that deform the electrode or the skin around the electrode.

We estimate the end of the T-wave as the beginning of the first isoelectric segment that follows the fiducial point by at least 300 ms.. We estimate the beginning of the P-wave as the first

isoelectric segment that is at least 250 ms before the fiducial mark. If we find an isoelectric segment at 250 ms, we search inward from there to find the isoelectric region that is closest to the onset of the QRS complex.

If there is space between the T-wave of the previous beat and the P-wave of the present beat, the noise in the segment is characterized by the maximum and minimum signal values divided by the segment length. If the segment is longer that 250 ms the noise measurement is made using only the 250 ms that precedes the P-wave.

4.1.4 High Frequency Noise

High-frequency noise is generally produced by muscle contractions close to the electrodes. We characterize the high-frequency noise surrounding a QRS complex beat by the maximum signal level of a bandpass filtered version of the beat outside the QRS complex.

4.2 Beat Matching

The file match.cpp contains the functions that handle matching beats to previously detected beats. Functions in match also handle storage of beat features that are associated with different beat types.

4.2.1 Beat Comparison

Beat matching is done by comparing beats sample point by sample point after aligning the beats and removing any constant difference between beats. In the present implementation, we represent beats with 100 sample long buffers of data that have been down sampled from 200 samples-per-second to 100 samples-per-second by averaging consecutive samples. The detection point as indicated by the QRS detector is 400 ms (FIDMARK) from the beginning of the buffer. Though we represent beats with 1000 ms sample buffers, we only compare 300 ms (MATCH_LENGTH) of data centered on the detection point.

The functions **CompareBeats** and **CompareBeats2** compare pairs of beats to each other. In **CompareBeats** the second beat is scaled so that its amplitude (max – min) matches the first beat. **CompareBeats** then calculates the point-by-point sums and absolute differences for the two beats for five different relative shifts and saves the sum and difference for the relative shift that produced the smallest total difference. We use the ratio of the sum of the absolute differences to the sum of the absolute sums as indication of the similarity between beats. The similarity metric is also divided by the MATCH_LENGTH so that it is insensitive to changes in MATCH_LENGTH.

CompareBeats2 differs only slightly from **CompareBeats**. **CompareBeats2** does not scale the second beat before comparing beats. In this way the similarity measure returned by **CompareBeats** is blind to amplitude differences between beats, but the similarity measure returned by CompareBeats2 takes amplitude differences into account.

4.2.2 Rules for Matching Beats

The rules for deciding whether a beat matches any of the previously encountered beat types are implemented in **Classify**. The function **BestMorphMatch** (morph is short for morphology here) compares a new beat to templates of all the preceding beats and returns the index of the template that best matches the new beat. **BestMorphMatch** initially compares the new beat to all templates using **CompareBeats**. If the beat closely matches two different templates, **BestMorphMatch** matches the new beat to the template which produced the best match with **CompareBeats2**.

BestMorphMatch also returns the two similarity indexes for the two beats. If the match indexes are small enough and the beat is not premature, the new beat is considered to be of the same type template indicated.

When a beat matches a template the function **UpdateBeatTypes** is called to average the new beat into the template. **UpdateBeatTypes** performs an iterative average by adding 7/8ths of the template to $1/8^{th}$ of the new beat. If, however, the new beat is the only second beat, half the template is added to half the new beat. I tried combining 3/4ths of the template with $1/4^{th}$ of the new beat for the third and fourth beats, but this hurt performance of the algorithm.

UpdateBeatTypes also calculates the beat width and polarity for the new average template and stores the type of beat that preceded the new beat.

If either match index is too large or the beat is premature and the amplitude sensitive match is too large, the new beat is considered to be a new beat type and the new beat is used to start a new beat template. However, if the beat is judged to be both noisy and regular a new beat type is not initialized because we assume that it is just a normal beat that would have matched if it weren't for the noisy conditions.

4.2.3 Combining Similar Templates and Eliminating Templates

Memory limits can produce problems for beat classification algorithms that use template matching. Low levels of noise can cause multiple beat types and templates to be created for what are actually beats of the same type. This can quickly use up whatever space was allocated for beat template storage. If space is allocated for a large number of templates, comparing beats to a large number of possible matches can significantly eat into available processing time.

In an effort to reduce the number of redundant templates that are created, **BestMorphMatch** also detects when two templates are similar enough that they can be combined to reduce memory usage. If a new beat closely matches two different beat types, **BestMorphMatch** compares those two types to each other. If the two types match closely enough, they are combined, freeing up memory for a new template type.

If the maximum number of beat types has already been detected and a new beat is detected that does not match any of the beat types, the new beat is substituted for the beat type that has been detected the fewest number of times. If two beats types have been detected an equal number of times, the new beat is substituted for the beat type that was least recently detected.

4.3 Dominant Normal Beat Selection

New beats are classified as NORMAL or PVC largely based on how they compare to the dominant normal beat type. Dominant beat selection is handled primarily by the function **DomMonitor** in classify.cpp. **DomMonitor** keeps track of the beat types, rhythm classifications, and beat classifications for the most recent 180 beats.

In deciding which is the dominant normal beat, DomMonitor uses a simplified set of rules to classify beats. The rules that DomMonitor uses are biased in favor of misclassifying some normal beats in order to be sure that ventricular beats are not classified as normal. DomMonitor classifies beats as normal if:

- The rhythm is NORMAL, the beat width is less than 120 ms, and there have been at least two in a row of this type; or
- The previous beat was within the P-R interval of this beat and the one before that was the same type (this rule allows for accidental detections of P-waves on 108).

DomMonitor chooses the dominant normal type as the beat type with the most beats classified as normal out of the last 180 beats. If there aren't any beat types that have had at least

half of their beats classified as normal in the last 180 beats, or the most frequently occurring beat type in the last 180 beats occurred at least twice as often as the most frequently occurring normal beat type, DomMonitor selects as dominant the type that has occurred most frequently since the beginning of the record.

4.4 Dominant Rhythm

DomMonitor also characterizes the general rhythm. If the rhythm is regular, than the premature occurrence of a beat indicates that the beat is abnormal. However, if the dominant rhythm is irregular (as occurs during atrial fibrillation of flutter) than the relative prematurity of a beat does not tell us much about whether or not the beat is normal. DomMonitor classifies the dominant beat type as regular if at least 60 (IRREG_RR_LIMIT) of the last 180 (DM_BUFFER_LENGTH) R-to-R intervals were classified as UNKNOWN (QQ).

4.5 Beat Classification

Beat classification is handled by the function **Classify** in classify.cpp. After a beat has been matched (or not matched) to a beat type it is classified as NORMAL, PVC, or UNKNOWN. If the type that the beat was matched to has been classified as NORMAL or PVC, the beat takes on the classification of its type. If the beat type has not been classified, but the three of the last four or six of the last eight of this type were classified as PVCs by post classification, then the beat is classified as a PVC. Otherwise, the beat is classified according to its features relative to the features of the dominant normal beat.

4.5.1 Type Classification

When a beat is matched to a type that does not have a classification yet, **Classify** attempts to classify the type. A type is classified as NORMAL if:

- There have been at least four in a row and the width for the QRS is less than 20 ms wider than the dominant type; or
- No dominant type has yet been found and there have been at least six in a row; or
- The beats of this type are NORMAL rhythm beats in bigeminy.

Types are only classified as PVC if they occur as the premature beats in a bigeminal rhythm.

If a beat type has occurred at least 10 times in the last 180 beats, none of the beats appear to be normal (regular with width less than 130 ms), and the type has previously been set to NORMAL, **DomMonitor** resets the type classification to UNKNOWN.

4.5.2 Post Classification

Post classification attempts to more accurately classify beats types and rhythms by using information from the following beat. It is easier to detect a premature R-to-R interval followed by a compensatory pause than it is to decide whether a given interval is premature. A running record of the post classifications for the last eight beats of each beat type is used to determine a post classification for that beat type. This classification may change whenever a new beat of a given type is detected.

A beat is considered by post classification to have a PVC type rhythm if the preceding R-to-R interval is less than 87.5% of the R-to-R interval that preceded it and the following R-to-R interval is 12.5% larger than the R-to-R interval that preceded the interval that preceded the beat. If a beat

has a PVC type rhythm and is preceded and followed by the dominant beat type, it is post classified as a PVC. Also, if a beat is significantly different in shape (amplitude sensitive similarity index > 2.5) and it is preceded and followed by the dominant beat type, it is classified as a PVC.

4.5.3 Beat Classification

If a beat is does not match a previously classified type it is classified according to a set of rules that are part of the function **TempClass**. The rules in TempClass are arranged in an if-else structure as follows:

- If no dominant beat type has been identified, classify the beat as UNKNOWN;
- If the dominant rhythm is normal, the dominant beat type doesn't vary much, this beat is premature and looks sufficiently different than the dominant beat classify as PVC;
- Else, if the beat is sufficiently narrow (< 90 ms), classify as NORMAL;
- Else, if the beat cannot be matched to any previously morphology and it is not premature, consider it NORMAL (probably noisy);
- Else, if the maximum number of beat types have been stored, the beat is not regular or premature and only one beat of this morphology has been seen, call it normal (probably noisy);
- Else, if this beat looks like the dominant beat and the rhythm is regular, call it NORMAL;
- Else, if the post classification rhythm is normal for this type and its shape is close to the dominant shape, classify as NORMAL;
- Else, if the beat is not premature, it looks similar to the dominant beat type, and the dominant beat type is variable (noisy), classify as NORMAL;
- Else, if this beat is significantly different from the dominant beat, there have previously been matching beats, the post rhythm classification of this type is PVC, and the dominant rhythm is regular, classify as PVC.
- Else, if the beat is wide (> 110 ms), wider than the dominant beat (at least 40 ms), doesn't appear to be noisy, and matches a previous type, classify it as a PVC.
- Else, if the dominant rhythm is regular and this beat is premature then classify as PVC.
- Else, if the beat is regular and the dominant rhythm is regular call the beat NORMAL.
- Else, if the beat is wider than the dominant normal beat type, wider than 100 ms , and significantly different from the dominant beat, call it a PVC.
- Else, if the beat is sufficiently narrow (< 100 ms), call it NORMAL.
- Else, if the beat isn't much wider (< 20 ms) than the dominant beat classify it as NORMAL.
- Else, if the beat is similar to the dominant beat, call it NORMAL.
- Else, if the beat is noisy, we can't trust our width estimate and we have no useful rhythm information, so guess NORMAL.
- Else, classify it as a PVC.

When a rule indicates that a beat or beat type should be similar to normal, this means that the amplitude sensitive similarity index for the beat compared to the dominant beat is must be below a given threshold.

5.0 Adapting Beat Detection and Classification for Different Sample Rates

As of release 1.2, two *#define* statements allow the beat detection and beat classification functions to be changed to accommodate different sample rates. The SAMPLE_RATE constant defined in qrsdet.h specifies the base sample rate for the software. This is the rate at which data should be passed to **QRSDet** or **BeatDetectAndClassify**. The BEAT_SAMPLE_RATE constant defined in bdac.h specifies the sample rate for the beat templates used for beat classification.

In our original implementation, the base sample rate was set to 200 samples-per-second and the beat sample rate was set to 100 samples-per-second. Comparing beats at a lower temporal resolution reduced the processing time required for classification without significantly changing the classification performance. We down sampled beats before classifying them by averaging pairs of beat samples. Down sampling is taken care of by the function **DownSampleBeat** in bdac.cpp.

If you wished, for example, to run the beat detection at 250 samples-per-second and the beat classification at 125 samples per second, you would change SAMPLE_RATE to 250 and BEAT_SAMPLE_RATE to 125. Because the beat sample rate is half the base sample rate, DownSampleBeat would not need to be changed. If, on the other hand, you wanted to run the beat detector at 1000 samples-per-second because that was the base sample rate for your system, but you still wanted to classify beats using 100 samples-per-second beat templates, you would have to change SAMPLE_RATE to 1000, leave BEAT_SAMPLE_RATE at 100, and modifiy **DownSampleBeat** to average 10 consecutive samples of the input beat to produce a single sample of the output beat.

5.1 Performance at Different Sample Rates

We have designed the software so that all filter lengths and time related constants scale with sample rate changes so that the constants and filter lengths are equivalent to the constants and filter lengths in the original 200/100 samples-per-second implementation. Ideally, implementations at other sample rates would perform the same as the 200/100 samples-per-second implementation, but round off approximations in constants and filter lengths result in slight differences in performance. Table 1 lists the sensitivities and positive predictivities for beat detection and beat classification on the MIT/BIH database and the AHA database (through 7210) for three combinations of base rate and beat rate. Table 2 lists the sensitivities and positive predictivities for beat detection alone on a wider range of sample rates. Performance differences only seem significant when the base sample rate is dropped as low as 100 or 125 SPS.

TABLE 1: Beat Detector and Classifier Performance at Different Sample Rates									
		MIT/BIH Database				**AHA Database**			
Base Rate	**Beat Rate**	**QRS Sens.**	**QRS +Pred.**	**PVC Sens.**	**PVC +Pred.**	**QRS Sens.**	**QRS +Pred**	**PVC Sens.**	**PVC +Pred.**
200	100	0.9975	0.9981	0.9371	0.9664	0.9977	0.9979	0.9343	0.9787
250	125	0.9974	0.9981	0.9359	0.9597	0.9975	0.9979	0.9287	0.9784
300	150	0.9974	0.9979	0.9303	0.9665	0.9976	0.9979	0.9284	0.9833

TABLE 2: Beat Detector Performance at Different Sample Rates		
	MIT/BIH Database	**AHA Database**

Sample Rate	QRS Sens.	QRS +Pred.	QRS Sens.	QRS +Pred.
100	0.996856	0.997905	0.995839	0.996423
125	0.997426	0.998257	0.99666	0.996788
150	0.997458	0.998016	0.997429	0.997652
175	0.997601	0.998093	0.997119	0.997268
200	0.997426	0.998071	0.997397	0.997588
225	0.997228	0.997994	0.997268	0.997769
250	0.997502	0.99806	0.997087	0.9973
300	0.99736	0.998016	0.99745	0.997897
325	0.997448	0.997874	0.997578	0.997684
360	0.997535	0.998038	0.997503	0.997865

5.1 Optimization at Different Sample Rates

In generalizing the QRS detection filters in qrsfilt.cpp, we substituted division operations for some operations that had been implemented as shifts. In a processor with limited power, this will slow the execution of the filter functions. In **lpfilt**, the division in:

$$output = y0 / ((LPBUFFER\_LGTH*LPBUFFER\_LGTH)/4);$$

only serves to scale the filter output, substituting the shift operation that provides the closest approximation to this operation will speed the operation up without a significant change in performance. Similarly, in **hpfilt** the operation:

$$z = data[halfPtr] - (y / HPBUFFER\_LGTH);$$

can be speeded up by choosing HPBUFFER_LGTH to be the closest power of 2 to the optimal buffer length. This would allow the division to be implemented with a shift operation.

The algorithm may be tuned for better performance with different base frequencies by experimenting with different values for the many constants used in classification. At a given base frequency, there may be constant values and filter lengths that were not possible to use with a base sampling frequency of 200 samples-per-second, but it is hard to say whether this type of tuning actually represents an improvement in the algorithm performance.

6.0 Easytest & Bxb

We have included the source code for two applications that we have used to test our beat detection and classification software, easytest.cpp and bxb.cpp.

Easytest uses the beat detection and classification functions to analyze records from either the MIT/BIH database or the AHA database and creates an annotation record for each file containing the beat detections and classifications for that record. The annotation files produced by easytest have the extension ".ate" (atest) and are copied into the same directory as the database files.

Bxb.cpp is a modified version of the bxb.c distributed as a utility application with the WFDB library on physionet.org. The original bxb.c uses a command line interface and compares beat-by-beat annotations in two annotation files. Our version of bxb does not take any input and compares ".ate" (generated by easytest) and ".atr" annotation files for all records in the MIT/BIH database or 35 minute files through 7210 in the AHA database.

Bxb creates two output files, adstat.txt and testrpt.txt. Adstat.txt contains a summary listing of the results for each file in the format:

<record>　　<QRS TP>　<QRS FN>　<QRS FP>　<PVC TP>　<PVC FN>　<PVC FP>

testrpt.txt contains a more detailed record of the types of detection errors for each record.

Easytest requires the following files to be compiled and linked together:

> Easytest.cpp
> Bdac.cpp
> Classify.cpp
> Rytmchk.cpp
> Noisechk.cpp
> Match.cpp
> Postclas.cpp
> Analbeat.cpp
> Qrsfilt.cpp
> Qrsdet.cpp
> Wflib.lib

Wflib.lib is a library of functions for accessing the database files. I used Turbo C++ to compile this library from the WFDB source code available on www.physionet.org. You may wish to compile your own WFDB library from more recent versions of the source code. Bxb only requires the files bxb.cpp and wflib.lib.

Both easytest and bxb were written to run on a PC under windows and look for MIT/BIH database files in the directory "c:\mitdb\". Easytest and bxb look for AHA database files in the directory "c:\AHA Database\." You may either create these directories and copy the database files into them, or you can change the define statements at the top of easytest and bxb to the names the directories containing database files on your system. Note that the directory names in the define statement need double backslashes because they are C strings, and the "c:\AHA Database\" directory is represented as "c:\\AHAdat~1\\" because of the way windows represents directory names containing spaces and more than eight characters.

Easytest and bxb are configured to analyze the MIT/BIH data. If you comment out the line:

#define MITDB

in easytest and bxb and recompile, easytest and bxb will analyze the AHA data.

References

[1] Moody, G.B., Mark, R.G., "Development and evaluation of a 2-lead ECG analysis program," Computers in Cardiology. 1982; 39-44.

[2] Forbes, A.D., Helfenbein, E.D., Heumann, J.M., Jimison, H.B., Lindauer, J.M., Platt, J.S., "Ambulatory arrhythmia analysis: a dual-channel, Baysian approach," Computers in Cardiology. 1985; 373-375.

[3] Hu, YH., Palreaddy, S., Tompkins, W.J., "A patient-adaptable ECG beat classifier using a mixture of experts approach," IEEE Trans. Biomed Eng., BME-44, pp. 891-899, 1997.

[4] Wieben, O., Tompkins W.J., Afonso, V.X., "Classification of PVCS with a fuzzy logic system," Proceedings-19th International Conference-IEEE/EMBS, pp. 65-67, 1997.

[5] Afonso, V.X., Wieben, O., Tompkins, W.J., Nguyen, T.Q., Luo, S., "Filter bank-based ECG beat classification," Proceedings-19th International Conference-IEEE/EMBS, pp. 97-100, 1997.

[6] J. Pan and W.J. Tompkins, "A real-time QRS detection algorithm," IEEE Trans. Biomed. Eng., vol. BME-32, pp. 230-236, 1985.

[7] P.S. Hamilton and W.J. Tompkins, "Quantitative investigation of QRS detection rules using the MIT/BIH arrhythmia database," IEEE Trans. Biomed Eng., vol. BME-33, pp. 1157-1165, 1986.

[8] Tompkins WJ (ed.). *Biomedical Digital Signal Processing: C-Language Examples and Laboratory Experiments for the IBM PC.* 1993. Englewood Cliffs, NJ: PTR Prentice Hall.

[9] J. L. Urrusti and W. J. Tompkins, "Performance evaluation of an ECG QRS complex detection algorithm," *Proc. Annual International Conference of the IEEE Engineering inMedicine and Biology Society*, pp. 800–801, 1993.

Appendix A: Database Results

Table A1 presents the QRS detection and beat classification performance for the open source ECG analysis software on individual records from the MIT/BIH arrhythmia database. The base sample rate was 200 Hz and the beat sample rate was 100 Hz for the results shown in A1. QTP, QFN, and QFP represent QRS complex True Positive, False Negative, and False Positive detections. Similarly VTP, VFN, and VFP represent True Positive, False Negative and False Positive detections for classification of ventricular beats.

Though ventricular classification results are shown for records that include paced beats (102, 104, 107, and 217) the totals for ventricular classifications do not include these records. Also, though the beat detector only requires eight seconds to initialize and the classifier generally only requires six normal beat detections before beginning beat classification, results in TABLE A1 allowed for a 5 minute learning period.

TABLE A1: MIT/BIH Arrhythmia Database Results

Record	QTP	QFN	QFP	VTP	VFN	VFP
100	1901	1	0	1	0	0
101	1523	0	2	0	0	2
102	1820	1	0	3	0	0
103	1728	1	0	0	0	0
104	1856	1	2	0	1	2
105	2153	2	42	18	11	38
106	1694	2	0	455	5	1
107	1783	1	0	49	8	0
108	1475	5	71	4	9	11
109	2098	1	0	28	4	0
111	1775	1	0	1	0	4
112	2110	1	0	0	0	0
113	1505	1	0	0	0	0
114	1602	2	2	30	0	5
115	1636	1	0	0	0	0
116	1996	21	4	97	1	2
117	1283	1	0	0	0	0
118	1916	0	0	11	2	0
119	1660	1	0	364	0	0
121	1558	2	0	1	0	0
122	2053	1	0	0	0	0
123	1266	3	0	0	3	0
124	1365	2	0	33	14	0
200	2164	4	1	669	31	2
201	1479	42	0	196	2	41
202	1866	5	0	13	2	12
203	2449	32	19	253	120	32
205	2199	2	0	62	3	0

Record						
207	1591	1	2	93	16	3
208	2415	22	5	803	21	2
209	2518	1	1	0	1	5
210	2184	20	3	140	25	5
212	2284	1	0	0	0	0
213	2697	3	0	184	11	3
214	1877	1	1	200	12	0
215	2794	1	0	128	3	1
217	1841	4	1	119	21	0
219	1772	1	0	45	6	0
220	1693	1	0	0	0	0
221	2016	4	0	313	3	0
222	2108	8	12	0	0	10
223	2197	2	0	403	52	3
228	1702	1	13	298	4	2
230	1858	1	0	1	0	0
231	1277	1	0	0	0	0
232	1485	0	2	0	0	5
233	2559	2	0	679	13	4
234	2291	0	0	3	0	0
TOTAL:	**91072**	**212**	**183**	**5526**	**374**	**193**

Table A2 lists the beat detection and beat classification results for the non-paced records in the first 70 records of the AHA ECG database. These results were obtained using a base sample rate of 200 Hz and a beat sample rate of 100 Hz.

TABLE A2: AHA ECG Database Results

Record	QTP	QFN	QFP	VTP	VFN	VFP
1201	1621	1	0	0	0	0
1202	2594	1	1	0	0	1
1203	2180	0	1	0	0	0
1204	2950	24	8	0	0	38
1205	2552	1	1	0	0	116
1206	2121	1	1	0	0	0
1207	1535	0	1	0	0	0
1208	2447	1	0	0	0	**0**
1209	3580	2	2	0	0	**16**
1210	1987	5	87	0	0	25
2201	2874	1	1	73	0	0
2203	2412	1	1	13	0	0
2204	3412	98	34	37	1	13
2205	1627	0	13	58	1	5
2206	1611	1	1	269	0	3
2207	3286	1	1	276	0	0

2208	2853	1	0	308	0	0
2209	2413	1	2	144	0	0
2210	2534	2	4	79	0	5
3201	2173	2	2	26	0	0
3202	2942	1	1	59	0	0
3203	1949	0	2	30	5	3
3204	1876	1	1	72	7	0
3205	1781	2	1	6	8	4
3206	3244	1	1	113	0	0
3207	2322	2	1	25	2	0
3208	2423	1	0	115	0	0
3209	2583	1	1	61	1	0
3210	2449	22	1	81	0	1
4201	1931	1	1	441	0	0
4202	2370	6	2	106	14	1
4203	2581	0	1	21	453	0
4204	2257	0	3	109	0	2
4205	1439	12	1	134	12	0
4206	1946	0	2	156	0	5
4207	3516	2	3	639	3	1
4208	1876	1	1	25	0	0
4209	2366	5	0	823	4	1
4210	2905	1	1	586	98	2
5201	2079	177	1	57	178	5
5202	2349	2	1	161	3	3
5203	2371	1	0	4	0	0
5204	2286	1	1	360	0	0
5205	1805	0	1	314	2	0
5206	2067	0	3	44	1	1
5207	2921	2	1	46	0	0
5208	1844	1	1	34	1	0
5209	2163	1	1	11	0	0
5210	2015	1	1	355	0	3
6201	2499	1	0	41	5	0
6202	1952	1	1	235	2	4
6203	2696	2	1	153	4	3
6204	2249	1	1	127	9	0
6205	2293	1	1	204	0	0
6206	2775	7	1	357	3	2
6207	2043	1	0	463	0	0
6208	2357	1	1	47	4	0
6209	2495	5	6	743	7	5
6210	3287	2	0	382	12	1
7201	3158	4	2	638	12	0
7202	2127	0	0	217	2	0

7203	2531	1	1	139	31	1
7204	1925	1	1	40	2	0
7205	2436	1	6	143	25	3
7206	3107	1	1	1882	2	0
7207	2339	1	5	93	0	5
7208	1558	7	1	34	7	0
7209	2863	33	11	1590	48	10
7210	1741	13	2	16	2	5
TOTAL:	**93518**	**473**	**238**	**13815**	**971**	**293**

Appendix B: Database Results with QRSdet2.

TABLE B1: QRSDet2 MIT/BIH Arrhythmia Database Results

Record	QTP	QFN	QFP	VTP	VFN	VFP
100	1901	1	0	1	0	0
101	1523	0	1	0	0	1
102	1820	1	0	3	0	0
103	1728	1	0	0	0	0
104	1855	2	3	0	1	3
105	2152	3	47	17	12	38
106	1695	1	1	457	3	1
107	1784	0	0	49	8	0
108	1473	7	53	9	4	20
109	2098	1	0	28	4	0
111	1775	1	0	1	0	4
112	2110	1	0	0	0	0
113	1505	1	0	0	0	0
114	1599	5	2	30	0	5
115	1636	1	0	0	0	0
116	1997	20	1	97	1	1
117	1283	1	0	0	0	0
118	1916	0	1	11	2	0
119	1660	1	0	364	0	0
121	1558	2	0	1	0	0
122	2053	1	0	0	0	0
123	1269	0	0	3	0	0
124	1366	1	0	34	13	0
200	2165	3	2	665	35	2
201	1494	27	0	197	1	59
202	1866	5	0	13	2	12
203	2450	31	14	250	123	26
205	2199	2	0	62	3	0
207	1591	1	0	102	7	3
208	2423	14	6	816	8	3
209	2518	1	2	0	1	5
210	2158	46	3	121	44	5
212	2284	1	0	0	0	0
213	2698	2	0	185	10	3
214	1877	1	1	200	12	0
215	2794	1	0	128	3	1

217	1842	3	0	121	19	0
219	1773	0	0	45	6	0
220	1693	1	0	0	0	0
221	2016	4	0	313	3	0
222	2108	8	42	0	0	15
223	2197	2	0	403	52	3
228	1702	1	5	298	4	2
230	1858	1	0	1	0	0
231	1277	1	0	0	0	0
232	1485	0	4	0	0	6
233	2559	2	0	679	13	4
234	2291	0	0	3	0	0
TOTAL:	**91074**	**210**	**188**	**5534**	**366**	**219**

TABLE B2: QRSDet2 AHA Database Results.

Record	QTP	QFN	QFP	VTP	VFN	VFP
1201	1621	1	1	0	0	1
1202	2594	1	1	0	0	1
1203	2180	0	1	0	0	0
1204	2947	27	4	0	0	37
1205	2553	0	1	0	0	117
1206	2121	1	1	0	0	0
1207	1535	0	1	0	0	0
1208	2447	1	0	0	0	**0**
1209	3580	2	3	0	0	**10**
1210	1983	9	122	0	0	40
2201	2874	1	1	73	0	0
2203	2412	1	1	13	0	0
2204	3402	108	39	37	1	11
2205	1627	0	15	58	1	5
2206	1611	1	1	269	0	3
2207	3286	1	1	276	0	0
2208	2853	1	0	308	0	0
2209	2413	1	2	144	0	0
2210	2534	2	5	79	0	6
3201	2173	2	2	26	0	0
3202	2942	1	1	59	0	0
3203	1949	0	3	30	5	3
3204	1876	1	1	72	7	0
3205	1781	2	1	6	8	4
3206	3244	1	1	113	0	0
3207	2322	2	2	25	2	1
3208	2423	1	0	115	0	0

3209	2582	2	2	61	1	1
3210	2449	22	2	81	0	2
4201	1931	1	1	441	0	0
4202	2371	5	4	108	12	2
4203	2581	0	1	21	453	0
4204	2257	0	2	109	0	1
4205	1439	12	1	134	12	2
4206	1946	0	9	156	0	11
4207	3516	2	4	639	3	1
4208	1876	1	3	25	0	1
4209	2366	5	0	823	4	1
4210	2905	1	1	586	98	2
5201	2072	184	1	51	184	5
5202	2349	2	1	161	3	3
5203	2371	1	0	4	0	0
5204	2286	1	1	360	0	0
5205	1805	0	1	314	2	0
5206	2067	0	7	44	1	5
5207	2921	2	1	46	0	0
5208	1844	1	1	34	1	0
5209	2163	1	1	11	0	0
5210	2015	1	1	355	0	1
6201	2499	1	0	41	5	0
6202	1952	1	1	235	2	4
6203	2696	2	1	153	4	3
6204	2249	1	1	127	9	0
6205	2293	1	1	204	0	0
6206	2775	7	0	357	3	1
6207	2042	2	0	462	1	0
6208	2357	1	1	47	4	0
6209	2495	5	6	742	8	4
6210	3288	1	0	382	12	1
7201	3159	3	2	641	9	0
7202	2127	0	0	217	2	0
7203	2531	1	1	139	31	1
7204	1925	1	1	40	2	0
7205	2436	1	10	152	16	2
7206	3107	1	1	1882	2	0
7207	2339	1	6	93	0	6
7208	1564	1	1	40	1	0
7209	2884	12	11	1612	26	10
7210	1752	2	1	16	2	162
TOTAL:	**93502**	**457**	**301**	**13849**	**937**	**471**

Appendix C: PIC 16F877 QRS detector results

TABLE C1: PIC QRS Detector MIT/BIH database results.

Record	QTP	QFN	QFP
100	1901	1	0
101	1523	0	2
102	1820	1	0
103	1728	1	0
104	1849	8	2
105	2149	6	52
106	1695	1	0
107	1782	2	0
108	1474	6	71
109	2098	1	0
111	1775	1	1
112	2110	1	0
113	1505	1	0
114	1602	2	1
115	1636	1	0
116	1993	24	2
117	1283	1	0
118	1916	0	0
119	1661	0	0
121	1558	2	0
122	2053	1	0
123	1269	0	0
124	1366	1	0
200	2164	4	1
201	1494	27	0
202	1866	5	0
203	2457	24	19
205	2199	2	0
207	1592	0	2
208	2425	12	6
209	2518	1	3
210	2193	11	3
212	2284	1	0
213	2699	1	0
214	1877	1	1
215	2794	1	0

217	1841	4	1
219	1773	0	0
220	1693	1	0
221	2020	0	0
222	2099	17	5
223	2198	1	0
228	1702	1	6
230	1859	0	0
231	1277	1	0
232	1485	0	2
233	2559	2	0
234	2291	0	0
TOTAL:	**91105**	**179**	**180**

TABLE C2: PIC QRS Detector AHA database results.

Record	QTP	QFN	QFP
1201	1621	1	1
1202	2594	1	1
1203	2180	0	1
1204	2956	18	15
1205	2553	0	1
1206	2121	1	1
1207	1535	0	1
1208	2448	0	0
1209	3580	2	2
1210	1989	3	63
2201	2874	1	1
2203	2413	0	4
2204	3401	109	53
2205	1627	0	10
2206	1612	0	1
2207	3286	1	1
2208	2845	9	9
2209	2414	0	2
2210	2534	2	27
3201	2173	2	2
3202	2942	1	1
3203	1949	0	3
3204	1877	0	1
3205	1781	2	1
3206	3244	1	1
3207	2322	2	5
3208	2423	1	0
3209	2583	1	1

3210	2449	22	3
4201	1931	1	1
4202	2368	8	2
4203	2581	0	1
4204	2257	0	3
4205	1439	12	1
4206	1945	1	2
4207	3516	2	4
4208	1877	0	3
4209	2366	5	1
4210	2906	0	1
5201	2083	173	3
5202	2350	1	1
5203	2372	0	0
5204	2286	1	1
5205	1805	0	1
5206	2067	0	2
5207	2921	2	1
5208	1844	1	1
5209	2163	1	1
5210	2016	0	1
6201	2499	1	1
6202	1952	1	1
6203	2696	2	1
6204	2250	0	1
6205	2293	1	1
6206	2775	7	1
6207	2043	1	0
6208	2358	0	1
6209	2497	3	4
6210	3288	1	0
7201	3160	2	2
7202	2127	0	0
7203	2532	0	1
7204	1925	1	1
7205	2436	1	5
7206	3107	1	2
7207	2339	1	5
7208	1560	5	1
7209	2789	107	108
7210	1752	2	3
TOTAL:	**93512**	**525**	**382**