ELG 6163 - DSP Microprocessors, Software, and Applications

Final Report

Implementation of algorithms for QRS detection from ECG signals using TMS320C6713 processor platform

Geoffrey Green
Carleton Student # 100350275
geoffgreen@ieee.org

March 31, 2006

Introduction and Application of Interest

The electrocardiogram (ECG) provides a physician with a view of the heart's activity through electrical signals generated during the cardiac cycle, and measured with external electrodes. Its clinical importance in cardiology is well established, being used for example to determine heart rate, investigate abnormal heart rhythms, and causes of chest pain.

As shown in Figure 1, the most important ECG signal features in a single cardiac cycle are labelled (along with the physiological cause of that feature) [X]:

- "P" wave - due to depolarization of the atria
- "Q" wave - due to activation of the anterioseptal region of the ventricular myocardium
- "R" wave - due to depolarization of the ventricular myocardium
- "S" wave - due to activation of the posteriobasal portion of the ventricles
- "T" wave - due to rapid ventricular repolarization

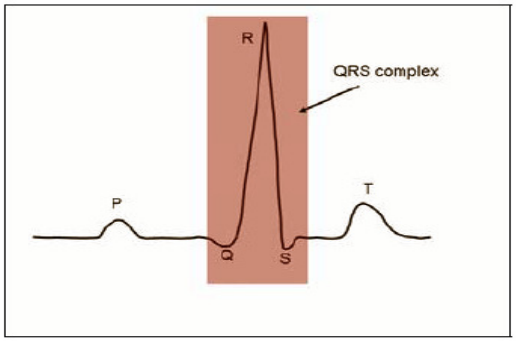

Figure 1: An "ideal" ECG signal from a healthy subject (time duration equivalent to one heartbeat). Key features, including the QRS complex, are identified.

Because the QRS complex is the major feature of an ECG, a great deal of clinical information can be derived from its features. Identification of this feature in an ECG is known in the literature as *QRS detection*, and it is a vital task in automated ECG analysis, portable arrhythmia monitoring, and many other applications [X]. Though trivial in an "ideal" ECG (as shown in Figure 1), the range in quality of real-world ECG signals obtained from a variety of subjects under different measurement conditions makes this task much more difficult (see Figure 2).

QRS detection in ECG signals has been the focus of a wide body of research for the last few decades. One finds in the literature a diverse collection of techniques drawn from many fields (an excellent review of which is given in [X]), including:

- methods based on discrete derivatives,
- methods based on digital filtering,
- methods based on a wavelet representation of the ECG signal,
- methods based on neural networks,
- methods based on adaptive filtering,
- methods based on genetic algorithms,
- methods based on maximum a posteriori (MAP) estimation.

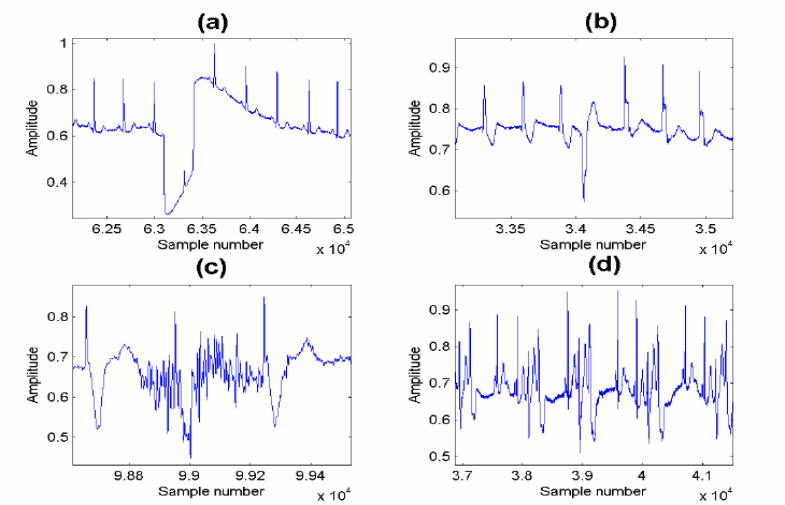

Figure 2: A variety of "real world" ECG signals (taken from the MIT/BIH Arrhythmia Database [X]) showing the effects of noise and time-varying morphology– a) Record 101, b) 102, c) 104, d) 106.

Project Scope

In this course project, I focussed on the implementation of a QRS detection algorithm [X] (algorithm details are in the next section) on a programmable DSP. The platform that I used is the TMS320C6713 DSP Starter Kit (DSK) and its associated tools, with the following features [X]:

- floating point support
- advanced VLIW architecture
- up to eight 32-bit instructions executed each cycle
- on-board A/D and D/A converters
- Code Composer Studio IDE with C compiler, software pipeline support, profiler capability, debugging capabilities
- real time data exchange (RTDX) between target and host

The bulk of the development effort was done in Simulink (and Matlab). Using the tools below (see Figure 3), the Simulink model files were programmed onto the DSP.

- Real Time Workshop
- Plug-in for Code Composer Studio
- Signal Processing Toolbox / Signal Processing Blockset
- Texas Instruments (TI) Embedded Target for C6000

QRS Detection Algorithm Description

The first algorithm (Pan/Tompkins 1985) is summarized in the block diagram shown in Figure 3 (full details are given in [4]), as well as a series of plots showing intermediate results at various stages in the process. This algorithm uses a hybrid of several processing methods, drawing heavily on digital filtering techniques and sophisticated peak selection rules. Their QRS detector consists of three stages that are expanded on below:

1. Linear digital filtering
2. Nonlinear transformation
3. Decision rule algorithms

1. Linear Digital Filtering

This stage is intended to emphasize signal characteristics that are specific to the QRS complex while suppressing irrelevant information. This reduces the possibility of false positives.

a) A bandpass filtering of the ECG (for noise reduction) is done by cascading the following filters:
   - Low pass (IIR filter with 3dB frequency cutoff ~ 11 Hz):
   $$y[n] = 2y[n-1] - y[n-2] + x[n] - 2x[n-6] + x[n-12]$$
   - High pass (IIR filter with 3dB frequency cutoff ~ 5 Hz):
   $$y[n] = y[n-1] - \frac{1}{32}x[n] + x[n-16] - x[n-17] + \frac{1}{32}x[n-32]$$

b) A discrete derivative is computed in order to emphasize the high slope of the QRS complex:
   - Derivative :
   $$y[n] = \frac{1}{8}(2x[n] + x[n-1] - x[n-3] - 2x[n-4])$$

c) A moving window integrator (MWI) is used to ensure that a measure of QRS complex width is included in the processed signal (since R-wave slope information is not sufficient to isolate the QRS complex). A window size of 30 samples (determined empirically) allows the capture of a wide QRS complex with a single pulse in the MWI output. Equally important, it is sufficiently small to avoid the possibility of including T wave contributions.
   - Moving Window Integrator:
   $$y[n] = \frac{1}{30}(x[n] + x[n-1] + x[n-2] + \cdots + x[n-29])$$

   (Note: The nonlinear squaring operation (see below) is performed before the MWI filter is applied).

2. Nonlinear Transformation

The nonlinear step is a squaring operation, creating a positive-valued signal that emphasizes high frequencies (suitable for subsequent moving window integration).

3. Decision Rules

At this point in the algorithm, the preceding stages have produced a roughly pulse-shaped waveform at the output of the MWI (see Figure 3). The determination as to whether this pulse corresponds to a QRS complex (as opposed to a high-sloped T-wave or a noise artefact) is performed with an adaptive thresholding operation and other decision rules outlined below.

a) FIDUCIAL MARK - The waveform is first processed to produce a set of weighted unit samples at the location of the MWI maxima. This is done in order to localize the QRS complex to a single instant of time. The w[k] weighting is the maxima value.

$$y[n] = \sum_{\substack{k=MWI \\ peak \\ locations}} w[k]\delta[n-k]$$

b) THRESHOLDING - When analyzing the amplitude of the MWI output, the algorithm uses two threshold values (THR_SIG and THR_NOISE, appropriately initialized during a brief 2 second training phase) that continuously adapt to changing ECG signal quality. The first pass through y[n] uses these thresholds to classify the each non-zero sample (CURRENTPEAK) as either signal or noise:
   - If CURRENTPEAK > THR_SIG, that location is identified as a “QRS complex candidate” and the signal level (SIG_LEV) is updated:
   $$SIG\_LEV = 0.125 \cdot CURRENTPEAK + 0.875 \cdot SIG\_LEV$$

- If THR_NOISE < CURRENTPEAK < THR_SIG, then that location is identified as a "noise peak" and the noise level (NOISE_LEV) is updated:

$$NOISE\_LEV = 0.125 \cdot CURRENTPEAK + 0.875 \cdot NOISE\_LEV$$

Based on new estimates of the signal and noise levels (SIG_LEV and NOISE_LEV, respectively) at that point in the ECG, the thresholds are adjusted as follows:

$$THR\_SIG = NOISE\_LEV + 0.25 \cdot (SIG\_LEV - NOISE\_LEV)$$

$$THR\_NOISE = 0.5 \cdot (THR\_SIG)$$

These adjustments lower the threshold gradually in signal segments that are deemed to be of poorer quality.

c) SEARCHBACK FOR MISSED QRS COMPLEXES - In the thresholding step above, if CURRENTPEAK < THR_SIG, the peak is deemed not to have resulted from a QRS complex. If however, an unreasonably long period has expired without an above-threshold peak, the algorithm will assume a QRS has been missed and perform a searchback. This limits the number of false negatives. The minimum time used to trigger a searchback is 1.66 times the current R peak to R peak time period (called the RR interval). This value has a physiological origin - the time value between adjacent heartbeats cannot change more quickly than this. The missed QRS complex is assumed to occur at the location of the highest peak in the interval that lies between THR_SIG and THR_NOISE. In this algorithm, the RR interval is calculated as an average of the last eight QRS locations in order to adapt to changing heart rate.

d) ELIMINATION OF MULTIPLE DETECTIONS WITHIN REFRACTORY PERIOD - It is impossible for a legitimate QRS complex to occur if it lies within 200ms after a previously detected one. This constraint is a physiological one – due to the refractory period during which ventricular depolarization cannot occur despite a stimulus[1]. As QRS complex candidates are generated, the algorithm eliminates such physically impossible events, thereby reducing false positives.

e) T WAVE DISCRIMINATION - Finally, if a QRS candidate occurs after the 200ms refractory period but within 360ms of the previous QRS, the algorithm determines whether this is a genuine QRS complex of the next heartbeat or an abnormally prominent T wave. This decision is based on the slope of the waveform at that position. A slope of less than one half that of the previous QRS complex is consistent with the slower changing behaviour of a T wave – otherwise, it becomes a QRS detection.

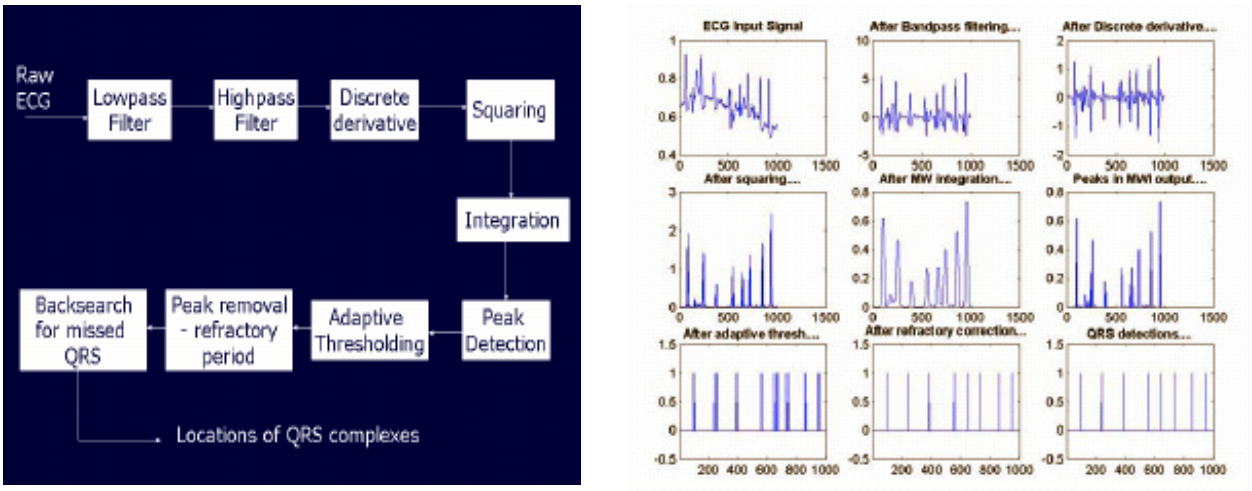

Figure 3: Stages of QRS Detection Algorithm (Pan/Tompkins 1985)

Testing Inputs

There are a couple of options for testing the algorithm with a real ECG signal – one is to use my own ECG (suitably recorded with a electrodes and a biopotential amplifier) or to use a pre-recorded one from a publicly available database. In this project, I opted for the second approach. The signals chosen were from the MIT/BIH Arrhythmia Database, a set of publicly available two-lead ECG records (11-bit data, sampling rate = 360 sps, 30 minutes each) that have been annotated by cardiologists to indicate the location of the "true" QRS detections [X].

After this signal had been read into Matlab, the "*sound*" command was used to play the signal, using the D/A converter of the computer's sound card. On the C6713 DSK board, that analog signal was subsequently digitized, then used as the input ECG signal for the DSP. The output of can be compared to the "true" QRS detections (in the annotations) to help identify whether or not the algorithm is working correctly. The setup is shown in Figure 4.

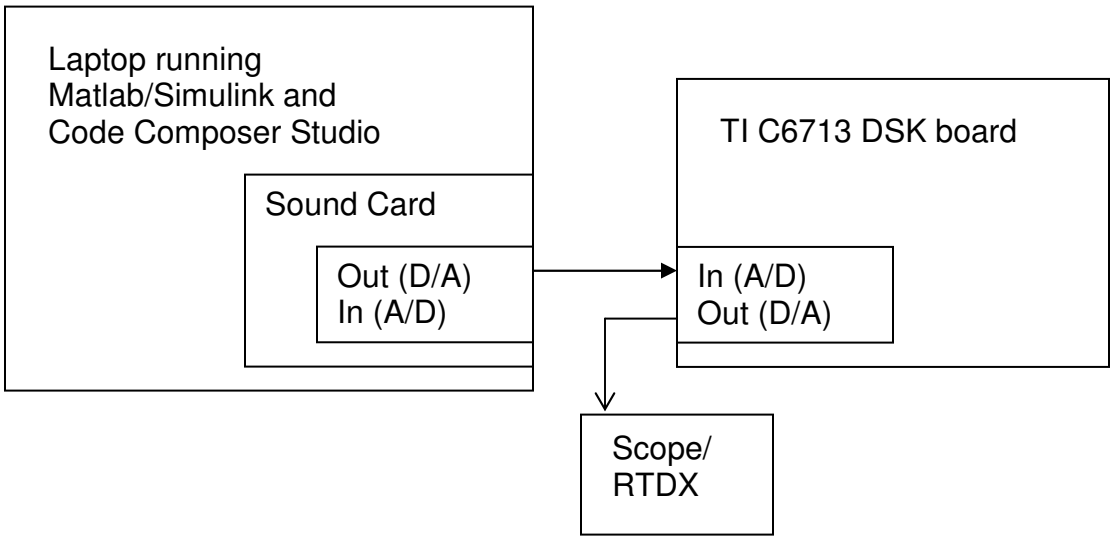

Figure 4: Testing setup

Project Workflow

1. Initial Testing of Algorithm (Simulink only)

The first stage in the development was to implement the algorithm as a Simulink model file, to verify its operation. A model file was constructed that used only blocks from the main Simulink library (i.e. no DSP blockset or TI blocks). The model file was constructed in a modular fashion, using subsystems as required. The full model file is provided in Appendix A.

At this stage, the input consisted of an ECG signal from the MIT/BIH database, which was read into the Matlab workspace using Physionet [X] tools. It was then incorporated into the model file using the "From workspace" block. This signal was originally sampled at 360Hz.

The model was verified by performing Simulink simulation and results from various algorithm stages (using the Scope block) are presented in Appendix A. The final signal consists of a single spike for each QRS complex in the ECG signal, which is the desired output.

2. Transfer of Algorithm to TI C6713 DSK

Having successfully implemented the QRS detection algorithm in Simulink, the next step was to convert the model file into one that could be used to generate running code on the TI C6713 P-DSP. The development process is outlined in Figure 5 and explained below.

Modification of model file

It was not possible to simply use the model file developed above. Indeed, transferring this model file to one which was "C6713-friendly" took most of the development effort. Some issues that were encountered included:

- Introducing blocks from the TI Embedded Target for C6000 library, such as target board type, reset blocks, etc.
- Incorporating the DSK's A/D and D/A converters in place of software scope blocks. This stage, while conceptually simple, was complicated by several software incompatibility issues (see Discussion section)
- Switching from Simulink main library blocks to DSP blockset blocks for most of the components including digital filters, delay blocks, data type conversion, rate transitions, and switches. It was not immediately obvious which of the blocks needed to be changed – in some cases, this was done quite easily but in other cases, it required a significant level of troubleshooting.
- Incorporating the RTDX blocks into the model file so that it would be possible to see the data being returned from the DSP board. Again, this stage should have been simple but was complicated by software incompatibility issues.

Setting up Configuration Parameters

There are several configuration parameters that are associated with the Simulink model file, specifying such things as model solvers, simulation diagnostics, and several Real-Time Workshop parameters such as hardware target board, compiler/linker options, byte ordering, etc. For the most part, these were easy to determine (based on an example provided in the TI documentation). A full list of these can be by selecting Simulation -> Configuration Parameters in the model file included with this report.

Generating Code

After modifying the model file, it was necessary to get the algorithm code into a form so that it could be programmed on the P-DSP. This was done with Real-time Workshop, a Matlab toolbox that converts the Simulink .mdl (model) file into C code. For my project, the following source files were generated by Real-time Workshop.

This was essentially a “push button” stage that went quite easily (since the bulk of the development effort was spent generating the model file). Investigation of the C source files that were generated revealed that there was a substantial amount of overhead but the crux of the algorithm seemed to be quite efficient, especially for the digital filtering stages.

Building Executable and Sending to C6713 DSK board

Having produced the required C files (along with an associated CCS project (.pjt) file), the Real-time Workshop build process (in association with the Matlab Link for Code Composer Studio (CCS) toolbox) then launches the CCS package. In CCS, the code is compiled (convert C to assembly code), assembled (convert assembly code to machine code) and linked in order to produce an executable (COFF format - *.out* file) which is then downloaded to the C6713 board and executed.

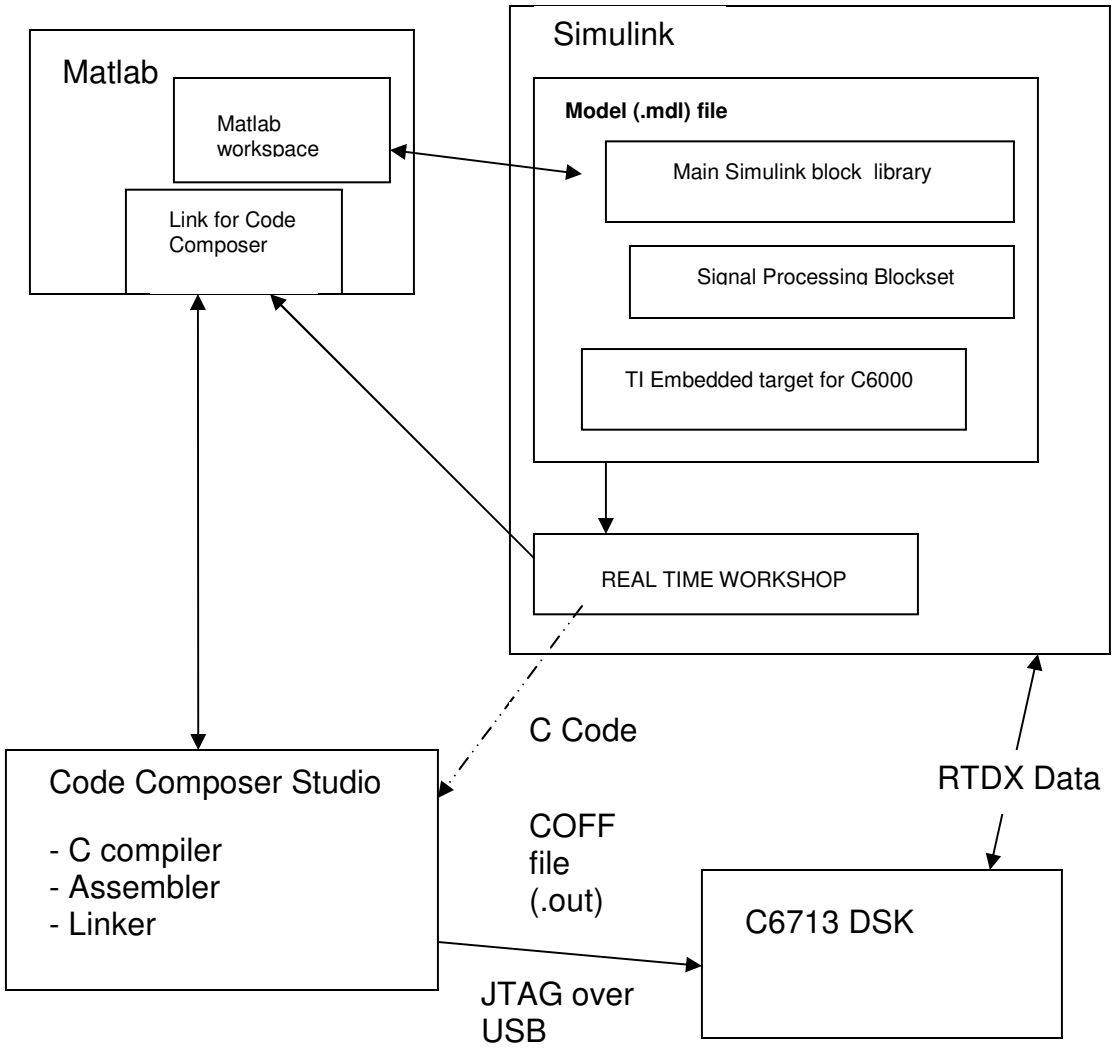

Figure 5: Overall development process using TI C6713 target with Simulink/Matlab and associated toolboxes.

Discussion of Problems Resolved / Outstanding Issues / Future Work (preliminary)

- There was an incompatibility between the versions of Code Composer Studio (2.2) and Matlab (7.0.1) that I was using that prevented any RTDX data from being sent from the DSP to Matlab for debugging/algorithm validation. This issue was a significant setback and it is document in the following URL:
  http://www.mathworks.com/support/solutions/data/1-Q28IB.html?solution=1-Q28I
  This problem was resolved by upgrading to Matlab 7.0.4.

- Right now, the algorithm implementation easily works at 8000Hz. It would be interesting to increase the sampling frequency to see when overrun condition appears (i.e. when the output can't keep up with the processing to maintain real-time operation).
- The feedback loop for adaptive thresholding works perfectly during Simulink simulation, but not during Real-time Workshop code generation.
- The "T-wave discrimination" and "Searchback for Missing QRS" parts of the QRS detection algorithm have not been implemented yet. In the interests of moving ahead with the P-DSP part of the project, a simplified version of the algorithm (that still performs QRS detection) was used.
- I have still not been able to successfully interface the A/D converter due to another incompatibility between Code Composer 2.2 and Matlab 7.0.4. To workaround this, I have stored the ECG input sample in RAM on the P-DSP.
- I had no time to investigate the wavelet-based QRS detection algorithm (Kadambe et al.1999) in the original project proposal.
- A detailed profiling during code execution (this can be performed at both the Simulink level and the Code Composer level) would be interesting to possibly isolate areas where the efficiency of the implementation can be improved.

References

[X] Rangayyan, R. M. Biomedical Signal Analysis – A Case-Study Approach, IEEE Press/Wiley, New York, NY, 2002.

[X] B-U. Köhler, C. Hennig, and R. Orglmeister, "The Principles of QRS Software Detection" *IEEE Eng. Med. Biol. Mag.*, pp. 42-57, 2002.

[X] PhysioNet, http://www.physionet.org

[X] J. Pan and W.T Tompkins, "A Real-Time QRS Detection Algorithm" *IEEE Trans. Biomed. Eng.* vol 32, pp.230-236, 1985.

[X] S. Kadambe, R. Murray, G.F. Boudreaux-Bartels, "Wavelet Transform-Based QRS Complex Detector", *IEEE Trans. Biomed. Eng.* vol. 46, pp.838-848, 1999.

[X] Texas Instruments, http://focus.ti.com/docs/toolsw/folders/print/tmdsdsk6713.html

[X] The Mathworks, http://www.mathworks.com