Electrocardiogram Analyzer

Final Project Report
December 8, 2006
E155: Microprocessor - Based Systems

Nick Evans and Nancy Yu

Abstract

The final project entails using two PICs and FPGAs to analyze an electrocardiogram signal. The system finds the inflection points in the sampled ECG signal and determines the time intervals between them. These values are used to calculate the lengths of the PR Interval, QRS Complex, and QT Interval. If any of these values falls outside of a given appropriate range, an alert will be triggered. Since this system was works, it will eventually be incorporated as a subsystem into the 2006-2007 CIMIT Clinic project.

Introduction

The final project analyzes an electrocardiogram signal using two PICs, FPGAs, and transceivers. This system was designed for the 2006-2007 CIMIT Clinic project and will be integrated into that project in the near future. Through this system, the lengths of the PR Interval, QRS Complex, and QT Interval can be determined. The system will find the locations in the ECG signal where the slope changes signs in addition to the time intervals between these inflection points – start, P, Q, R, S, T, and end, as shown in Figure 1. With these results, the desired intervals can be concluded. When these calculated values fall outside of the acceptable range, an LED is programmed to turn on. The results of the ECG analyzer will be sent through transceivers, via wireless, to another location.

Background

An electrocardiogram signal (ECG or EKG) is a measure of the electrical signals controlling the activities in the heart. An ideal and actual signal is shown in Figure 1. An ideal signal shows segments of no voltage change between peaks. Occasionally, if a person's heart rate is slow enough, these segments will appear. However, actual signals usually do not have those segments. Examples of real ECG signals are documented in APPENDIX F. An ECG can detect abnormally slow, fast, and irregular heart rhythms in addition to other cardiac conditions.

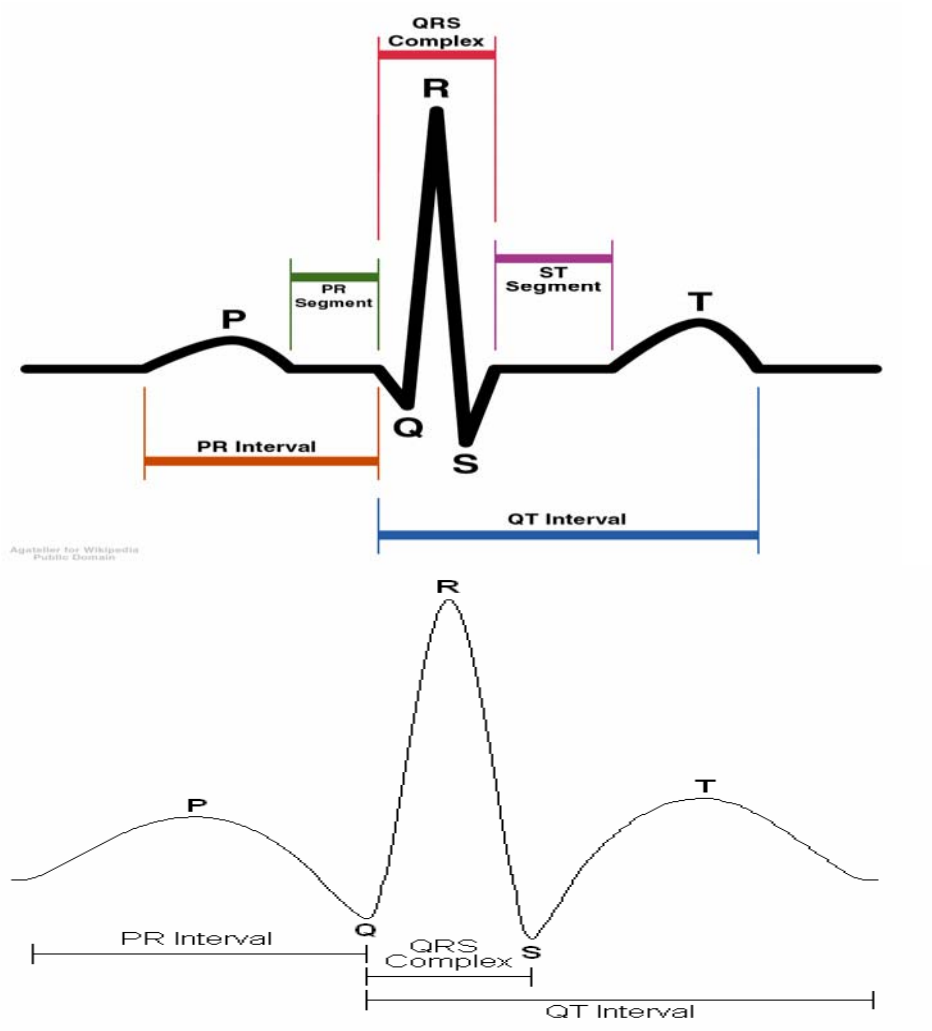


Figure 1: Top – Ideal and Bottom - Realistic Electrocardiogram Signal

Overall Procedures

Since the ECG signal has both positive and negative voltage, in order for the analog to digital (A/D) converter to accurately capture the signal, the entirety of the signal must be positive and in the range 0V to 3.3V. Once the input is in an appropriate form to be read, a timer will be designed to interrupt the program to sample at 100 Hz using the A/D converter. The number of samples per second is quite variable. The only real limitations on the number of samples per second are the max speed of the clock on the PIC and the amount of memory on the PIC.

To analyze the signal sampled and extract the desired information, the program needs to find all inflections points in a period and the time period between each inflection, distinguish between a signal and noise, determine the start and end of a complete cycle, calculate the PR and QT intervals and QRS complex, trigger alarm if the intervals are not reasonable, and send results to a wireless chip. The overall implementation is illustrated in Figure 2.

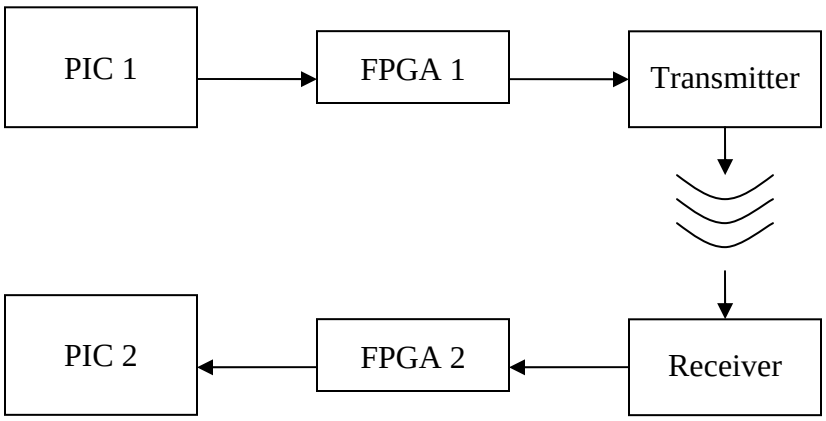


Figure 2: System Block Diagram – The main ECG signal processing is done in PIC 1 and transferred to FPGA 1 to be sent through the Transmitter. The FPGA 2 processes what was received by the Receiver and passes the information to PIC 2 for display.

The outline in APPENDIX A also gives a good overview of the steps needed to be taken for this project.

Current Implementations

Currently, everything proposed, with the exception of the trigger detection and wireless reception, works. By examining several signals we determined that the signal tended to range from -1.06V to 1.8V. In order to make sure that the signal falls inside the detectable range but not outside, a 1.2V DC offset will be applied. Since the mechanism

that sends data into the system can add the needed DC offset to the input, the signal being sampled can easily be manipulated into the appropriate values. Then, to ensure that the data is sampled at a constant rate, the PIC timer will be used to trigger an interrupt and enable the A/D converter to check the input every 1/100 of a second. Since the sample frequency is 100 Hz, and the pre-scale is set to 256, the timer will be set to 0xFE79. This results in the timer interrupting every 100,000 clock cycles which, with the PIC we are using, equals 1/100 of a second. At the beginning of the interrupt portion of the code the timer is reset and restarted right away in order to keep the sampling rate as accurate as possible. This timer/interrupt routine runs 200 times, once for every data point sampled. In the interrupt, the timer as well as the interrupt is reset and the A/D converter is enabled. Each ECG signal is about 0.76 seconds long so we decided to use 100 samples per second giving us about 76 data points per signal would be high enough resolution. Since 100 data points per period seems reasonable, the sample frequency needs to be 100 Hz. In order to guarantee the presence of a full signal cycle, two seconds of data will be collected and transferred from the ADRES registers into an array with size 200.

Once all 200 points have been collected, the slope between each point can easily be calculated. Because the interval between each point is constant and only the magnitude in relation to the other points is important, the slope is simply be the difference in magnitudes of the first and second point. Each slope is calculated and sorted into positive or negative sign. Calculating one slope at a time can save memory from not storing unnecessary values as well as easily filter through some of the noise. Because noise can add unexpected and false results to the signal, they must be removed. Noise will therefore be small slope change intervals. Thus, a counter will count the number of points in the

intervals and append short ones to the appropriate actual intervals. Since the previous slope is stored into memory, the two slopes can be compared for similar slope signs. If the slopes have the same sign – both positive or both negative – the interval count is incremented by one. If they are not consistent, a temporary counter will keep track of how long the discrepancy lasts. The algorithm is set so that noise should only be two intervals long. Once the temporary counter passes that value, a new interval will start with the count stored in the temporary counter, while the old interval is recorded. This way, small noise will be caught while finding the intervals.

A second level of noise filter has also been incorporated. A change in slope sign is considered as a result of noise if the interval between changes is short enough not to be significant. Cases to consider are:

1. The first inflection point occurs too early to determine any information
2. The last inflection point is too short to be useful
3. A small discontinuity in the slope due to noise in signal
4. Two short consecutive slope sign changes
    a. The first slope sign change interval is larger than the second
    b. The second slope sign change interval is larger than the first
    c. The two slope sign change intervals are equal
5. Consistent oscillation of short slope sign change intervals

The noise interference will be removed by adding the false inflection point segment to either the previous or following interval. An algorithm has been developed which determines which inflections are caused by noise and which ones are legitimate cardiac signals. The noise induced inflections are analyzed by looking at how they relate to the segments on either side. The noise induced segments are then absorbed into the segments on either side depending on the results of the analysis. After all these calculations, an array of time intervals between each significant inflection point will be available.

A full cycle of the ECG signal can be considered from one point R, which has the greatest magnitude of the whole sign, to the next point R. Thus, using the array of sample data points and the list of inflection points, the highest magnitude point in the sample data can be determined. Since the desired intervals include multiple inflection points, certain values must be added together. Starting from point R, QRS Complex is simply the addition of two calculated intervals. QT is total interval of the QRS Complex and the next two intervals. PR is the addition of the two intervals before point R. Once these values are determined, they need to be compared to acceptable ECG interval ranges, which can be set by the user. If the intervals in the sampled signal are inappropriate, an alert needs to be set. Therefore, certain LEDs will light up according to which alarm was triggered. The entirety of the PIC code, along with enough comments to understand each section, is included in APPENDIX B.

Once the PIC has determined the values of the QRS, QT, and PR intervals it passes these values on to the FPGA. Because the A/D converter used port A, only port B, C, D and E were available. Ports C and D can both handle 8 bits while port B only has the capacity for 6 bits and port E only 3 bits. This presented a problem because the three results that needed to be transferred to the FPGA were 8 bit values. But because the QRS Complex would never use all 8 bits, this problem can be circumvented by using port B to send the QRS complex value. Port B would be more than appropriate since the QRS value should never be as high as 63, the limitation of a 6 bit number.

Thus, the following Table 1 shows the ports and their outputs:

PORT	Input / Output	Value
A	Input	A/D converter
B	Output	QRS Complex Interval
C	Output	QT Interval
D	Output	PR Interval

Table 1: Values in each port

The FPGA takes in these values and concatenates them in order to transmit them serially through the wireless transmitter. The majority of the code for the wireless transmission portion of this project was taken from D. Chan's MicroToys project on the E155 website. Some relatively minor changes were made to expand the capacity of the transmitter. The initial design was set up to take in values from switches and transmit an 8 bit number to the receiver. This project's supplemental code modified this slightly by expanding the number of bits to 24 in order to transmit all of the information simultaneously. In order to get the transceivers to work, it is simply a matter of modifying the code included in APPENDIX C for the specific task and following the transmitter and receiver schematics in APPENDIX D.

The FPGA then transmits the string of binary bits through a pin on the Harris board. The information is transmitted by sending the top bit of the concatenated data and shifting all of the bits left by one, which cuts the top bit off. This process will repeat until all the data is sent. Over a period of several cycles the entirety of the data is transmitted through a single pin on the Harris board.

The transmitter itself is very simple; the transmitting pin on the transceiver is connected to the outputting pin on the Harris board, which allows for easy serial transmission. The receiver works effectively the same way. All of the serially received

data comes in through a single reception pin and then sent to the second FPGA in the order it was received from the transmitter. Ideally the FPGA on the second board would have taken the data, disassembled it, and assigned portions of it to specific pin outputs. Then that information would be passed to the second PIC, again through the use of Ports B, C, and D, to be compared to acceptable values However, the second FPGA was unfortunately not registering the sent information. Because of this, the comparison and alarm triggering process had to be moved to the first FPGA.

Main Problems Encountered

A big issue in this project was the limitation on memory space. Initially, the arrays were holding int values. However, changing the array type into char, much more memory spaces opened up. Since each value in the array only needed eight bits, char is an appropriate format.

When the QRS, QT and PR results were not within the appropriate ranges, a LED would be triggered to light up. However, the LEDs did not turn on when triggered. Because of the limited available time, this problem was not fixed. This problem was probably due to either the code or the method of method of implementation.

Unfortunately, the team was also unable to get the second FPGA to recognize the incoming signal. The transmitting FPGA sends a synchronization signal designed to force the two FPGAs to synchronize. This is mean to ensure that the second FPGA will receive all of the transmitted data. Next, a start sequence from the first FPGA is sent to signify that the following transmitted bits are to be taken in by the second FPGA. While there was not enough time to adequately diagnose the problems with the second FPGA, the

team believes that there was a problem in getting the two FPGAs to synchronize correctly. This would prevent the second FPGA from correctly taking in values from the receiver.

Conclusion

Even though the right LED did not light up when it was triggered and the second FPGA could not register the transmitted information, the rest of the project was completed. Therefore, most of the proposed functions worked.

This project was able to receive an input signal, capture and store at least two full ECG signals, find all the inflection points in the data, determine the span between them, calculate the desired QRS, QT and PR intervals, compare them to acceptable values, and send those results through transceivers.

References

Chen, D. “Wireless RF Transceiver.” Mar. 2005.

<http://www4.hmc.edu:8001/Engineering/microtoys/Wireless%20RF/MicroToys%20Wireless%20RF.pdf>

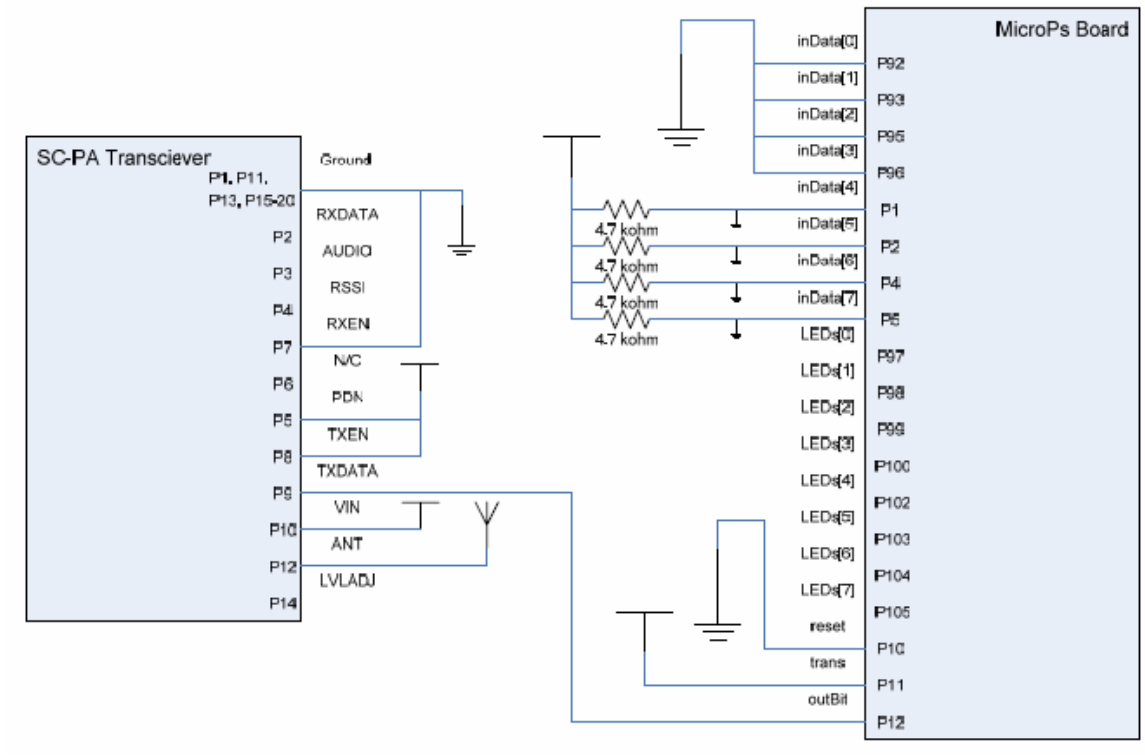
APPENDIX A: Procedure Outline

1.) Make sure input data is acceptable
    a. DC offset input – either on sawtooth sample or real ECG signal
2.) Take in input (Change dynamic signal to static data points) – 100 Hz
    a. A/D converter
        i. Use 1/100 timer for interrupt and sample once *200
        ii. Sample data for 2 seconds (guarantees at least two R peaks)
    b. Store each data point from sample
        i. Store into an array
3.) Analyze the signal
    a. Find inflection points
        i. Find slope for two points
            1. Point 2 – Point 1
        ii. Count interval for one slope sign
            1. Determine sign of slope
            2. Compare sign with previous slope sign
                a. Same: increment interval by one
                b. Different: increment temp, check next slope
            3. Store interval and repeat through all points
    b. Filter noise
        i. Check for small inconsistencies or short intervals
        ii. Append short intervals with previous or following intervals
    c. Determine one complete signal
        i. Find all max inflection points
            1. Compare magnitude of inflection points
        ii. Shift point toward middle of sample data if not already
    d. Add appropriate values together to find desired periods
    e. Compare calculated periods against acceptable range
        i. If test failed – trigger alarm
4.) Report results wirelessly
    a. PIC 1 to FPGA 1 to transmitter
    b. Receiver to FPGA 2 to PIC 2

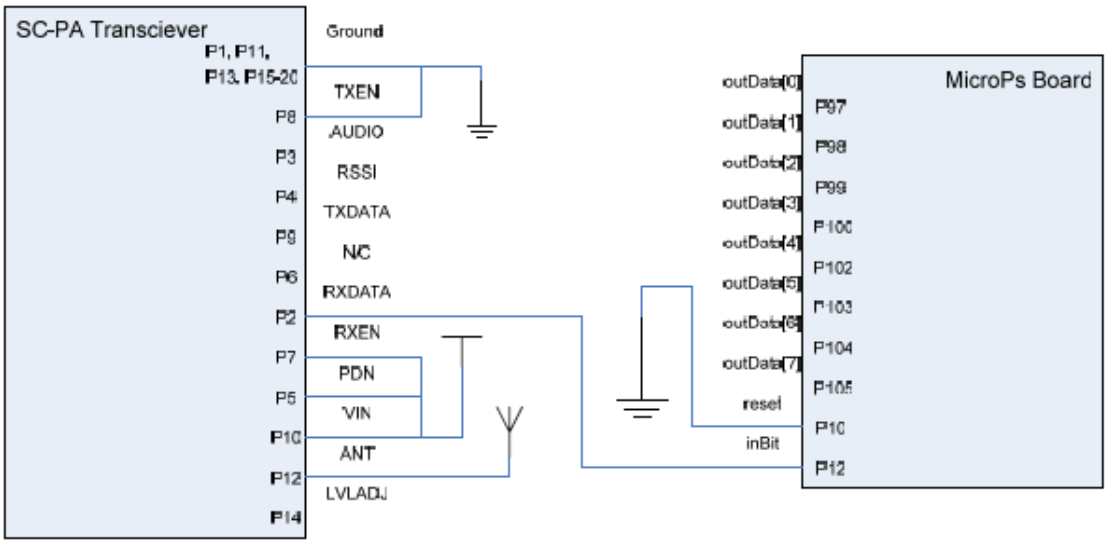
APPENDIX B: PIC Code

APPENDIX C: FPGA Code

APPENDIX D: Transceiver Schematics



Transmitter Schematic



Receiver Schematic

APPENDIX E: Program Schematics

APPENDIX F: ECG Real Sample Signals