INT J COMPUT COMMUN, ISSN 1841-9836
8(4):548-559, August, 2013.

Multithreaded Application for Real-Time Visualization of ECG Signal Waveforms and their Spectrums

S.S. Ilić, A.Č. Žorić, P. Spalević, Lj. Lazić

Siniša Ilić, Aleksandar Č. Žorić, Petar Spalević
University of Pristina in Kosovska Mitrovica
Faculty of Technical Sciences
Computer Sciences and Informatics Department
Serbia, 38220 Kosovska Mitrovica, Kneza Miloša 7
sinisa.ilic@pr.ac.rs

Ljubomir Lazić
State University of Novi Pazar
Serbia, 36300 Novi Pazar, Vuka Karadžića bb
llazic@np.ac.rs

Abstract: By using concept of virtual instrumentation, signals from human body can be digitized and transferred to computer for further processing. Software in a computer enables use of modern tools for digital signal processing that can be improved easily with emergence of new knowledge and with increasing of computer performances. Presenting the ECG signal in both: Time and Time-Frequency domains enables to cardiologist to obtain more reliable diagnosis.
In order to present simultaneously the waveform and spectrogram of ECG signal in the real time we use Fast Fourier and Discrete Wavelet transform in the multithreading environment of a standard personal computer. The synchronization of accessing the signal data by threads according to the principle one thread a time is performed by controlling the state of event type variables.
Keywords: Biomedical signal processing, Fast Fourier transforms, Multithreading, Real time systems, Wavelet transforms

1 Introduction

The standard data acquisition devices are powerful, relatively expensive and designed to perform one or more specific tasks defined by the vendor. However, the user generally cannot extend or customize them. Virtual instruments, by virtue of being PC-based, inherently take advantage of the benefits from the latest technology incorporated into off-the-shelf PCs.

Use of PC in so called virtual or personal instrumentation [1], [2] development enables realization of a new generation of superior devices. Virtual instrument consists of two sub-devices:

1. Acquisition hardware device that collects data and transform it in the form suitable for reading by PC and
2. Computer with software (user interface) device that reads digital data, process it and presents results.

Once the data has appeared at the input of computer in digital form, it can be stored and used for different types of processing as many times as needed by using different or same software. As the central role in the virtual instrumentation concept is played by the software [1], [3], the costs of development, maintenance and reconfiguration of instruments are reduced significantly. These devices can be connected to the internet, they offer easy modifications according to the users needs, it is possible to store large amount of data at a cheap price, and these data can be retrieved quickly.

By using of PC-based ECG device that is built upon the principles of virtual instrumentation, we can monitor changes of bio-potentials at the skin of the human body caused by heart activity ECG. Some of the tools for processing of ECG signal that can be implemented in this device are: segmentation (recognition and separation of ECG signal parts) [4], [5], measuring of heart rate variability [6], compression of signal for storage purposes [7], etc.

The analysis of ECG signal in Time-Frequency domain can help cardiologist to have a better look to the signal from a different angle. The presence of some frequency components in appropriate segments of ECG signals can help cardiologists in decision making [8] - [10]. The mathematical tools that can be used for this purpose are: Short Time Fourier Transform (STFT), Discrete Wavelet Transform (DWT) and Continual Wavelet Transform (CWT). It would be very useful for the users to monitor ECG signal in Time-Frequency domain in real time.

The need for real-time processing of medical images becomes higher and some implementations are presented in the following papers [11] [13]. In order to enable processing in real-time, faster algorithms [14], multithreading [15], modularity [16] and other improvements are needed.

We have come to idea to plot in real time simultaneously ECG signal in both Time and Time-Frequency domain using STFT and DWT. The idea is realized by development of multithreading application in software part of our virtual instrument [18].

2 The PC-ECG Device

The PC-based ECG device is described in [9], [17], but the most important features will be presented in this paper too. The main user requirements, suggested by cardiologists, were the following functional and control system performances:

- selection of one of twelve leads for real time monitoring and recording by the descriptive and intuitive GUI,
- recording of waveforms for each of twelve leads during a time interval of six seconds,
- connection to the database of patients,
- filtering of patient body signal from the public power network frequency,
- possibility to select, measure the selection time duration and zoom the selection,
- printed report generation (waveforms of 12 leads and cardiologist diagnosis),
- easily portable and no additional power supply (plug into PC and use),
- 100 % patient's body insulation.

2.1 Hardware part of the Device

The hardware part of the system is designed by using Microchip microcontroller PIC16F876 and USB communication port of a PC is selected as interface for bilateral communication between acquisition hardware and a PC. The communication between acquisition hardware and a PC can be obtained: 1) by using standard USB cable; in this case PC is power supplier of the acquisition hardware and 2) by using MaxStream XBee RF module; in this case the device is powered by battery and communication is wireless the USB RF dongle using PIC18F2550 MCU and XBee RF module is attached to a PC and it communicates with acquisition device equipped with the same XBee RF module [18]. The selection of communication type depends on distance between a patient and a PC if the distance is short, the USB cable will be used and if the distance is long the wireless communication will be selected. The XBee RF modules operate within the ZigBee protocol [18] and support the unique needs of low-cost, low-power wireless sensor networks.

The ECG signal is sampled with sampling rate of 960Hz. We used two bytes for transmission of each sample value; hence the transmission speed was set to 19600 Baud/s. The program of microcontroller selects the appropriate ECG channel (ECG lead) of multiplexer and performs A/D conversion or stops the program execution, depending on the code received from PC.

2.2 Graphical User Interface Design

The design of software part of the instrument is based on the traditional style of ECG paper plots with additional overlay displays and other features such as: zoom, filter, database support, etc. based on proposals of cardiologists.

The device can be used just to monitor signal waveforms (without saving the data) or it can be used to store signal at the buffer memory of PC. A user can select to preview and save the signal of any of 12 ECG leads. The application screen is divided to main and auxiliary window. When the Save button is clicked on the toolbar, the waveform of recorded signal is shown at the auxiliary window while the plotting of running signal is continued at the main window. A user can decide to overwrite already recorded signal by the signal he/she monitored at later time. After signal of all or some ECG leads are recorded at the buffers of PC (the signal of one lead is saved in one buffer), a user can save the samples from the buffers to the database with the: general data about patient, patient visit with medical anamnesis, explanation on interpretation of signals anomaly and selection of one of predefined diagnosis. All 12 leads with patient data, medical anamnesis, diagnosis and treatment can be printed in two-sided A4 paper report and given to the patient.

Software enables defining and marking the segments of ECG signal by a user (for example QT-interval), measuring the segment time duration (for example 0.412 seconds) and saving the boundaries of those segments along with signal to the database. The cardiologist can analyze the signal again, after it is recorded considering all interpretations he or somebody else put in database (for example: in the lead V1 ST depression up to 3mm, in the lead D3 small elevation, QT segment too wide etc.). By selection of diagnosis from the finite list of diagnoses for each patient visit, it becomes easier for cardiologist to search for appropriate signal waveforms, interpretations of waveforms and patients in database. Database also helps him to compare the waveforms and interpretations of ECG signal for the same patient taken in different patient visits and to make appropriate decisions on treatment and disease progress. The screenshot of application with three parts the main waveform (at the bottom), the main spectrogram/scalogram (at the central part for spectrum and on the right part for color legend), and auxiliary window (at the top-right) is presented in Figure 1.

3 Time-Frequency presentations of ECG signal in PC-ECG device

There are two modes of plotting ECG in Time-Frequency domain in the application: standard and real-time mode. The standard (off-line) mode enables plotting of spectrogram (in the case of STFT) or scalogram (in the case of wavelet analysis) after selecting the waveform of the lead already recorded in database or in memory buffer. In this case the waveform of 6 seconds duration is processed at once; the Time-Frequency coefficients are calculated and forwarded to plot function. The calculation and plotting of the coefficients is not time limited in this mode.

In the real time (on-line) mode samples of the signal appears continuously at the input port of a PC. It is not necessary to plot sample by sample in real-time, it is enough to plot group of samples one by one every 40 milliseconds (25 times in a second). The drawing of ECG

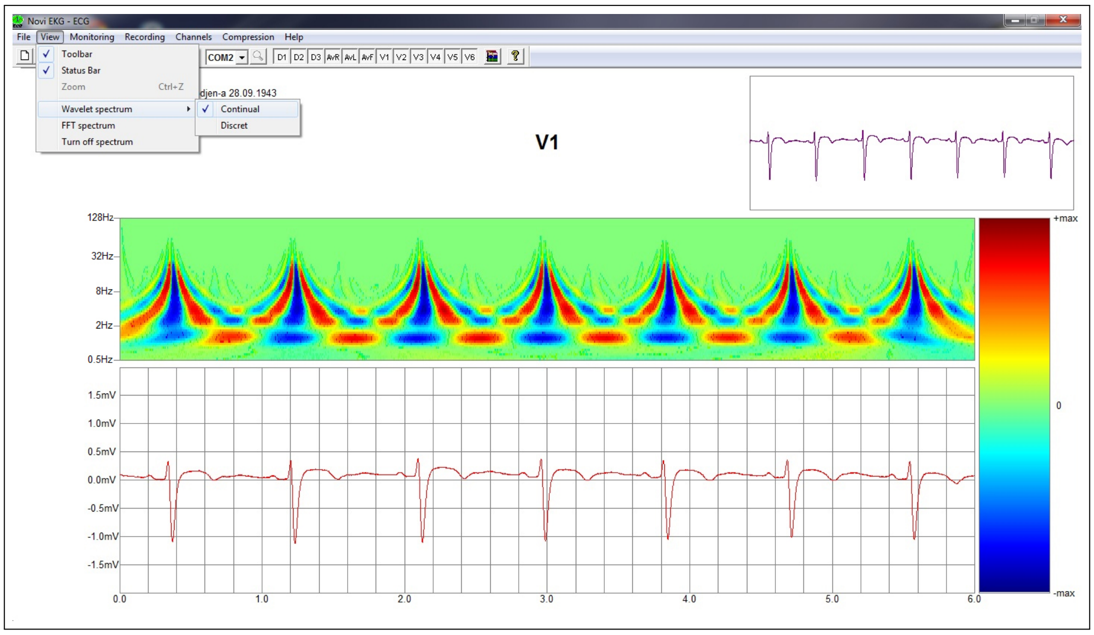
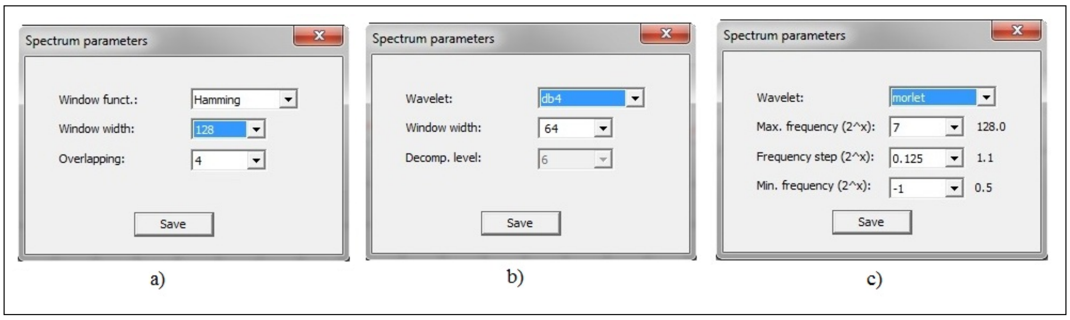


Figure 1: The Screenshot of PC-ECG Application

waveform, calculation of coefficients and drawing the Time-Frequency presentation of samples in 40 milliseconds time-frame must be performed between two calls of *OnTimer* function.

As the Time-Frequency plot consist of three axes: time, frequency and component value, we have decided to present the third axis by color. The color palette has 240 colors: from nuances of dark blue toward nuances of light blue, green, yellow, light red and dark red (see Figure 1). Before plotting of spectrogram (scalogram) the range of all calculated values of signal's coefficients in the standard mode is normalized by the maximum absolute value and proportionally divided to integer values in the range 0-239. In the real-time mode calculation of coefficients is performed on portions of the signal. The maximum of coefficients' values differs from portion to portion. This way of normalization might produce visually same spectrums for portions of signals with different amplitudes. Hence, the values of calculated coefficients are normalized by predefined maximum value (the maximum of maximum values found in standard mode spectrograms).

The STFT spectrum is calculated by using FFT function on overlapped windows. The window length (in samples), length of shifting and windowing function can be selected on appropriate application dialog (see Figure 2a).


Spectrum parameters

Window funct.: Hamming
Window width: 128
Overlapping: 4

Save

a)

Spectrum parameters

Wavelet: db4
Window width: 64
Decomp. level: 6

Save

b)

Spectrum parameters

Wavelet: morlet
Max. frequency (2^x): 7 128.0
Frequency step (2^x): 0.125 1.1
Min. frequency (2^x): -1 0.5

Save

c)

Figure 2: The Application dialogs for selecting parameters for calculating spectrograms

The samples of the window are multiplied with coefficients of windowing function, the FFT is then applied on the resulting set and spectral amplitudes are obtained. The window is then shifted by chosen number of samples and the process is repeated. If N is the total number of samples in recorded signal, W is the number of samples in window and O is number of samples for which the windows are shifted, then the following is valid: $N = W + k \times O$, where $k+1$ is the total number of windows to be calculated. From W samples of one window $W/2+1$ amplitudes of frequency components are calculated and presented over the y axis, so the total number of spectral coefficients of the signal is $(W/2+1) \times (k+1)$. In the standard mode all coefficients are calculated first, then normalized and plotted. In the real-time mode it is necessary to calculate, normalize and plot coefficients of n/O overlapping windows in less than 40 milliseconds, where n is the number of samples captured in the 40 ms time frame. The application function that calculates STFT spectrum in real time takes the following input parameters: 1) the window length W, 2) the shifting size O, 3) sample set size n, 4) pointer on last n samples ($*ns$) and 5) pointer on W samples captured before last n samples ($*ws$) on the input port of a PC. The function starts with making the window from $W-O$ last samples of $*ws$ and O starting samples of $*ns$ and calls the FFT function by forwarding samples of the created window. Then the new window is created by $W - 2 \times O$ last samples from $*ws$ and $2 \times O$ starting samples from $*ns$ and forwarded to FFT function. The process is repeated until the last shifting window of length W is created and its spectral coefficients calculated. The coefficients are then normalized and plotted at appropriate portion of spectrum part of the application screen.

The DWT spectrum is calculated by using convolution function. The wavelet function and the window length (W in samples) can be selected on appropriate application dialog (see Figure 2b). After selecting the wavelet function name, the wavelet and scaling function coefficients (of length h) are retrieved from the table of wavelets. In the standard mode signal (of length N) is extended on both sides because of border distortion by symmetric boundary value replications of length $h-1$ each and filtered with appropriate wavelet (high pass filter) and scaling (low pass filter) function. The filtering is performed by convolution function. As the number of samples in the recorded signal is large, the FFT convolution [20] is applied instead of classical convolution because of speed improvements. Two filtered signals are now obtained - approximation A_1 (low pass) and details D_1 (high pass). In each signal, N central coefficients are selected and from them odd coefficients are kept and even discarded thus resulting in two groups of $N/2$ coefficients. Detail coefficients D_1 are saved at buffer and the process of obtaining the new approximation A_2 and detail D_2 coefficients from approximation coefficients A_1 is repeated again by using the same procedure, but now beginning with $N/2$ coefficients of A_1 instead of N samples of signal. The process is repeated $L = \log 2W$ times and the number of obtained coefficients is halved in each iteration (level). The DWT spectrum consists of coefficients of details D_1, D_2, …, D_L where levels represent frequency ranges.

In the real-time (on-line) mode the coefficients are calculated by processing DWT window by window (of length W) from FIFO buffer where ECG samples are captured. The border distortion is eliminating by keeping samples and coefficients of A_i in L buffers of length $h-1$ from previous windows. The coefficients of A_1 and D_1 are calculated from W samples of current window and $h-1$ samples of previous window, coefficients of A_2 and D_2 are calculated from $W/2$ coefficients of just calculated coefficients of A_1 and last $h-1$ coefficients of A_1 from previous window etc. The number of samples in window W is small and the standard convolution is used. The coefficients of details that are calculated in time of 40 ms are normalized and plotted at appropriate portion of spectrum part of the application screen.

The calculation of CWT is for now possible only in standard off line mode. From the samples of signal, CWT is obtained by filtering signal with wavelet functions of different scales, where scale represents measure of stretching of wavelet function in time.

The pseudo frequency components are obtained from scales by formula:

$$f_q = \frac{f_c}{sc \times \Delta} \tag{1}$$

where f_c is central frequency of chosen wavelet (for example f_c=0.8125Hz for Morlet wavelet), sc represents scale and Δ is sampling period of the signal [21]. Usually the CWT of ECG signal is presented in so called semi-log axes - the frequency axis is logarithmic and time axis is linear (see Figure 1). If the spectrogram (scalogram) is to be presented in logarithmic axis from frequency 2^{-1} to 2^7 Hz (scales from 1560 to 6.1 with use of $\Delta = 1/960$ s) in s (for example 65) scales, s filters has to be derived from chosen wavelet function. The orders of these filters for Morlet wavelet (of order 16) are from 98 (for frequency 2^7) to 24961 (for frequency 2^{-1}) and hence the complexity of calculating CWT is large. The number of filters determines y axis resolution - the larger number the better resolution. Final number of calculated coefficients is $s \times N$ where N is the total number of samples of recorded signal. The application dialog for selecting frequency range and resolution is presented in Figure 2c.

4 Multi-threading

The multi-threading in PC-based ECG device is performed by using three threads: the main window standard thread, the worker thread and the GDI - graphic thread.

The main window standard thread starts with starting of application execution. This thread can work stand alone. In the thread a user can open the database, search the particular patient by name, date and diagnosis, open the ECG waveforms of patient found, measure the time width of particular segments, and analyze the waveform already stored in both: time domain and time-frequency domain using FFT, DWT and CWT in standard mode. A cardiologist can analyze recorded standard ECG waveforms and change diagnosis and add some comments. The printing function is also controlled by this thread.

When a user wants to monitor ECG signal of a patient in real-time, by pressing appropriate button in application, the working thread and timer are started (by function *OnTimerStart*) and the main thread is then directed to execute function *OnTimer* periodically.

The communication between the acquisition module and a PC is available only by using the working thread. This thread represents listener because it listens for data appearance on appropriate serial communication port of PC that are sent by the acquisition module. When data appear on the port, the working thread copy them to the transmission buffer which task is to copy data from one thread to the other. The transmission buffer holds the data that can be accessed only one thread a time. As the size of the buffer is initialized at the program start, the varying size of currently transmitted data between threads is controlled by unsigned integer variable that shows how many bytes are copied to the buffer.

The main thread periodically (on timer ticks) checks for the new contents of the transmission buffer, copies data to the local buffer, resets the variable that shows how many bytes are copied from the buffer to zero and forwards data from local buffer to the drawing function. The drawing function then draws the portion of the signal based on just received data on the main screen using appropriate sizing of signal values. At this point, it is important to say that not the whole screen is re-drawn after reception of new data, but only the portion of it that corresponds to the new data size. As the time axis of the main screen is predefined to be 6 seconds, and the sampling frequency of input data is known, it is easy to calculate the window region that should be refreshed (MFC function *invalidaterect()*) upon reception of the new data. The timer frequency is set to be 25 ticks per second that results in about 40 samples per tick. If the ECG signal in those samples is "polluted" by public power network frequency of 50Hz, the band pass IIR filter will eliminate this noise and then the samples of "clean" signal are forwarded to the drawing function. The speed of modern computers allows this filter to be implemented in real-time without jeopardizing continuity of signal drawing on the main screen.

Unfortunately, the time-frequency transform and drawing spectrograms (scalograms) cannot be always executed in real time by processing samples of signal in the main thread after drawing the signal in time domain. By introducing GDI thread that works independently of the main thread, it is possible to copy new samples from the main to the GDI thread and to let the GDI thread calculate spectrum portion of the signal and present the values of its coefficients to the spectrum screen simultaneously with the main screen drawing of the signal waveform performed by main thread. The diagram of running multi threads in our PC-based ECG device is presented in Figures 3 and 4.

4.1 Threads synchronization

To synchronize work of three threads in parallel, we used six "Event" variables. This type of variable can have two states - "Set" and "Reset". Appropriate built-in functions: *WaitForSingleObject* and *WaitForMultipleObjects* that take parameters of "Event" type, enable checking of event states. The both functions can control the workflow of execution of the program lines within the thread based on the state of event variable they control. The input parameters of the functions are: the name(s) of event(s) they control and the time the function has to wait for event to change state from Reset to Set. If the input event variable changes the state to "Set" within the waiting time interval, the program executes one block of program lines - at the "detected Set event" output of the function, and if state of event variable remained "Reset" within the time interval, the program executes another block of program lines - at the "time out" output.

The behavior of these functions is symbolically presented by yellow decision modules in Figures 3 and 4, where ”Yes” output directs the program flow to the first block of lines and ”No” output directs to the second.

Two event variables - *MainWEvent* and*RdrEvent* are used to synchronize copying data between the working and the main thread. *MainWEvent* is controlled by the main thread (only main thread can change its state) and *RdrEvent* is controlled by the working thread. The working thread is started by command *OnTimerStart* in the main thread and *MainWEvent* is initiated to the "Set" state. It enables the working thread to copy data from serial port to transmission buffer_1 before the main thread starts checking the content of transmission buffer_1 (then the *MainWEvent* is put in "Reset" state). When data are received at the serial port in working thread, this thread resets the *RdrEvent* (by this action it forbids the main thread to access transmission buffer_1) and checks the state of *MainWEvent.* If the state of this event is "Set", data from local buffer (of this thread) are copied to transmission buffer_1. As the state of *RdrEvent* is "Reset" during copying, the main thread in Timer function is stopped for time period of 1ms. If the copying is finished before the waiting time expires, the reader thread change the state of *RdrEvent* to "Set", and enables main thread to copy data from transmission buffer_1 to its local buffer (see Figures 3 and 4).

It may happen that both variables - *MainWEvent* and *RdrEvent* are simultaneously in "Reset" state and both wait for each other to change the state. The working thread is set to wait for the state of *MainWEvent* infinitely, and the execution of this thread literally stops when *MainWEvent* is in "Reset" state, but the main thread is set to wait for the change of *RdrEvent* state just 1 ms. When 1 ms passes, the main thread resolves the conflict by changing the state of *MainWEvent* and thus enabling the working thread to continue.

The situation when both variables are simultaneously in "Set" state is impossible, because before checking the state of variable that is controlled by another thread, each thread resets the

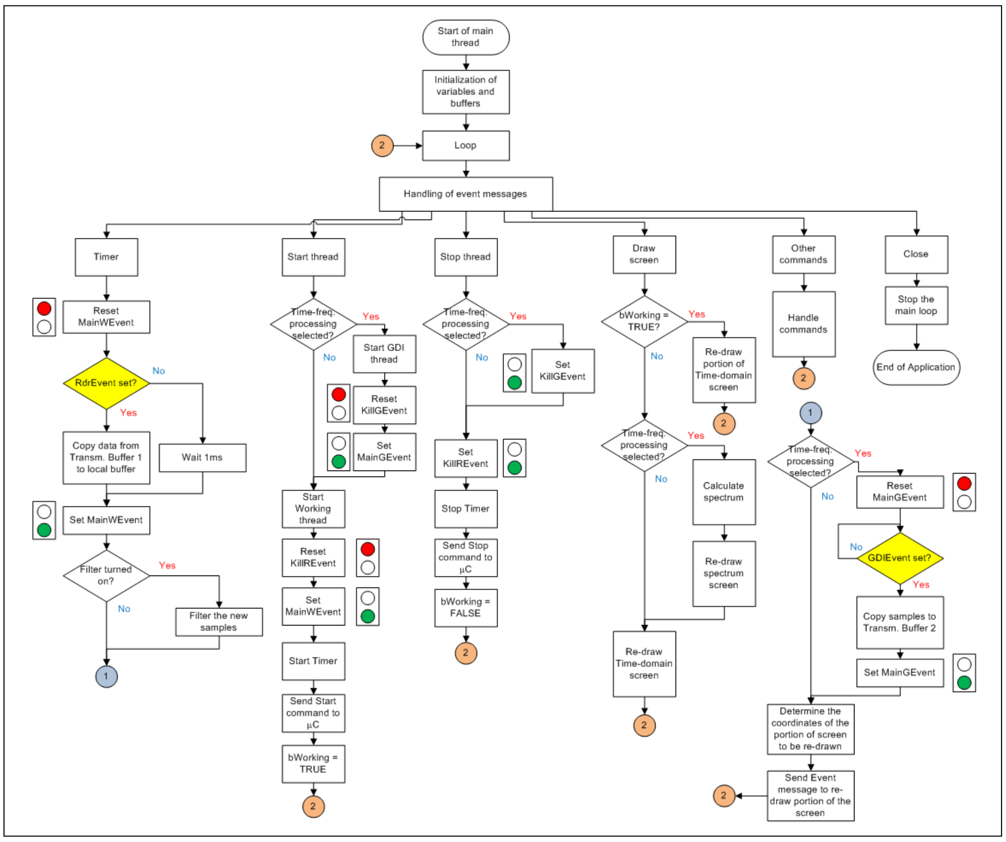


Figure 3: The work-flow diagram of the main thread

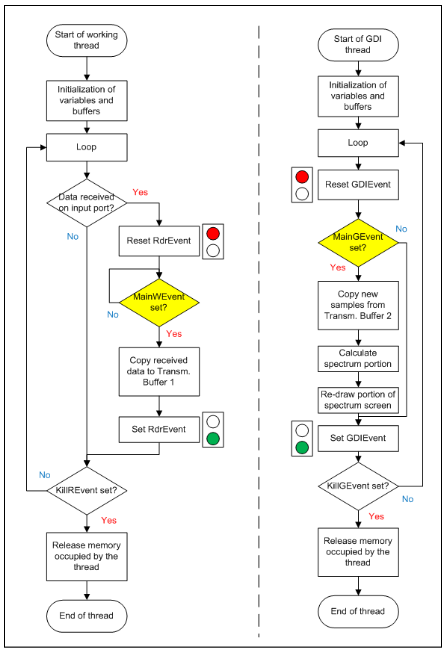

Start of working thread
Initialization of variables and buffers
Loop
Data received on input port?
Reset RdrEvent
MainWEvent set?
Copy received data to Transm. Buffer 1
Set RdrEvent
KillREvent set?
Release memory occupied by the thread
End of thread
Start of GDI thread
Reset GDIEvent
MainGEvent set?
Copy new samples from Transm. Buffer 2
Calculate spectrum portion
Re-draw portion of spectrum screen
Set GDIEvent
KillGEvent set?
Yes
No

Figure 4: The work-flow diagram of the working (left) and graphical (right) threads

own event variable. Setting and resetting of event variables are presented in Figures 3 and 4 by small semaphores (traffic lights) with red and green lights.

The synchronization between the main and the graphical thread is organized in the same way as the synchronization between the main and the working threads. Here we have another two event variables - *GDIEvent* (controlled by the GDI thread) and *MainGEvent* (controlled by the Main thread). The transmission of data between these threads is performed through transmission buffer_2.

As can be seen from the Figures 3 and 4, the task of the working thread is to receive data from serial port and to copy it to the main thread. The main thread can draw the waveform portion of input samples to "Time-domain" screen in real-time and, if time-frequency analysis is selected, forward the samples to the graphical thread to perform calculation of signal spectrum and present spectrogram (scalogram) on separate screen window.

The role of the event variable *KillREvent* is to control running of program loop in working thread. The state of this variable is controlled by the main thread. When the working thread starts, the initial value of *KillREvent* is "Reset" and it enables the program lines in this thread to be executed in the loop. But when the state of the event has been changed, the loop in the working thread is finished and this thread is closed. The situation of closing the graphical thread is the same as for closing the working thread, but the variable *KillGEvent* is used.

5 Experimental Results and Discussion

In order to test efficiency of multi-threading, we recorded the waveform samples of ECG signal of 72 seconds duration (equal to 69120 samples). Then we tested the response of the software by sending recorded samples from another computer in four different baud-rates: 19200, 38400, 57600 and 115200 bits/second. The software is modified in a way that user can choose to see scalogram that will be 1) drawn in the same thread where the ECG waveform is drawn (we call this method "single") or 2) drawn in another (GDI) thread (we call this method "multi").

We measured time needed to main thread executes *OnTimer* function where:

- data are copied from working thread,
- filtering of the received samples is performed (using IIR filter) [17,18],
- connection to the database of patients,
- the coordinates of ECG waveform are calculated and drawn,
- samples are sent to GDI thread (in the case of multi-threads) or wavelet transform is calculated and coefficients are drawn in separate portion of the screen (in the case of single thread).

We used the following C++ instructions for calculation of time of execution of *OnTimer* function:

```
void CECGView::OnTimer(UINT nIDEvent){
  LARGE_INTEGER ntime1,ntime2;
  LARGE_INTEGER freq;
  QueryPerformanceFrequency(&freq);
  QueryPerformanceCounter(&ntime1);
  . . .
  Set of commands in OnTimer function
  . . .
  QueryPerformanceCounter(&ntime2);
  ntime = (ntime2.QuadPart-ntime1.QuadPart)/(freq.QuadPart/1000);
}
```

In this way we were able to measure time between beginning and ending of function *OnTimer* with accuracy of 1ms. The wavelet we used in our experiments was Daubechies 4 with 8 coefficients and the window width was set to 32 samples.

The experiments were performed on 8 different computers with different processors, different graphical cards and different services running in background. The purpose of making this experiment was not to compare the speed of these computers, but just to compare the speed of running program in single and multiple threads in different baud rates. For each computer 8 different experiments were performed by changing baud-rate and number of threads. The number of samples received during one call of *OnTimer* function, duration of execution of instructions within the function and time step between two successive calls of the function for baud rate of 19200 bauds/second in the single thread is given in the Table 1.

Table 1: The measurements of number of samples received, duration of *OnTimer* function and time step between two succesive calls of *OnTimer* function in single thread

Number of samples received	Duration of OnTimer function	Time step (ms)
48	0	47
48	1	47
48	0	47
32	0	47
48	1	47
48	0	47
...	...	...

By averaging number of samples received in one call of *OnTimer* function we obtained the following values: 44.5, 88.5, 134 and 267 samples for baud-rates of 19200, 38400, 57600 and 115200 bits/second respectively. The average time between two successive calls of the function was 46.8ms for modern PC and 40ms for old computer with Intel Celeron working on frequency of 800 MHz (the timer clock was set to trigger function in 40ms for all computers). The average times of execution of *OnTimer* function in milliseconds are given in the Table 2.

Table 2: Average execution time in [ms] of *OnTimer* function in different computers

Computer	**Threads**	**19200**	**38400**	**57600**	**115200**
Intel I7 2600K	Multi	0.000	0.000	0.000	0.000
	Single	0.000	0.000	0.022	1.012
Compaq 8710p	Multi	0.007	0.008	0.008	0.009
	Single	1.172	2.640	3.932	7.210
Vostro 1510	Multi	0.007	0.007	0.013	0.012
	Single	1.340	3.109	2.314	4.837
Celeron 800MHz	Multi	0.000	0.000	0.000	0.016
	Single	1.422	2.357	3.749	8.112
ProBook 6460b	Multi	0.001	0.001	0.003	0.035
	Single	0.503	1.707	6.172	11.102
AMD AI 3000+	Multi	0.000	0.000	0.000	0.062
	Single	0.391	1.076	2.094	4.517
AMD AII 5000+	Multi	0.005	0.006	0.115	0.115
	Single	0.077	1.515	2.430	4.617
Intel Atom N450	Multi	0.109	0.124	0.147	0.168
	Single	4.148	9.048	13.674	29.955

When the software was executed on Intel Atom netbook computer, with single thread and baud-rate of 115200 bauds/second, after several seconds of running, the execution time of *OnTimer* function rose to 94ms and running of the software was blocked.

It is obvious from the Table 2 that calculating coordinates for drawing waveform, performing filtering of ECG samples and passing samples to GDI thread takes less time than with addition of calculating spectral coefficients and drawing them in the same thread. In the case of slower computers the next call of *OnTimer* function can happen before the previous running of the same function is finished. In this case both instances of the function will need access to the same buffer. Then running of the software is blocked.

6 Conclusion

The PC based ECG device is a solution for real time complex signal processing in analysis of ECG signal. It provides presentation of ECG signal in Time and Time-Frequency domain including filtering of signal and database connection.

We have designed source and processing modules to enable parallel processing of incoming data by using multi-threading technology on a PC. The capturing of incoming data, displaying data in Time domain and calculation of spectral coefficients and its presentations in Time-Frequency domain through spectrograms (scalograms) is realized in three threads respectively: working, main and graphical. The calculation of spectral coefficients is performed using three Time-Frequency transformations: Short Time Fourier Transform, Discrete Wavelet Transform and Continuous Wavelet Transform.

The experiments performed prove that using of multi-threads obviously enable larger number of calculations in shorter time compared to the same number of calculations using single thread.

The proposed multithread concept enables the real time display of signals spectrum and waveform, encourages and provides ample evidence to pursue multi-thread architecture as a serious candidate for real-time signal processing.

Acknowledgement

This work has been done within the project - 'Optimal Software Quality Management Framework', supported in part by the Ministry of Science and Technological Development of the Republic of Serbia under Project No.TR-35026.

Bibliography

[1] Ferrero, A. ; Software for Personal Instruments, *IEEE Trans. On Instrum. And Measurement*, 39(6): 860-863, 1996.

[2] National Instruments, Instrumentation: reference and catalogue, Austin, Texas, USA, 1996.

[3] House, R. ; Choosing the Right Software for Data Acquisition, *IEEE Spectrum*, March, 23-40, 1995.

[4] Kunzmann, U. et al ; Parameter extraction of ECG signals in real time, *Biomedizinische Technik*, Vol. 47, 2002.

[5] Marchesi, C. et al ; ECG processing algorithms for portable monitoring units, *The Internet Journal of Medical Technology*, 1(2), 2004.

[6] Tarvainen, M. et al ; Time varying analysis of heart rate variability signals with Kalman smoother algorithm, *Physiological Measurement*, 27(3): 225-239, 2006.

[7] Ilić, S. ; Comparison of Compression Ratios for ECG Signals by Using Three Time-Frequency Transformations, *Facta Univ. Ser.: Elec. Energ.*, 20(2): 223-232, 2007.

[8] Ilić, S. ; Detection of the Left Bundle Branch Block in Continuous Wavelet Transform of ECG Signal, *Electronics And Electrical Engineering*, 2(74): 33-36, 2007.

[9] Ilić, S. et al ; Improvement in decision-making in emergency medicine by using PC-based system for electrocardiography, *Tehnika*, 3, 2011.

[10] Addison, P. et al; Evaluating arrhythmias in ECG signals using wavelet transforms, *IEEE Engineering in Medicine and Biology*, Vol. September/October, 104-109, 2000.

[11] Ustun, T. et al; Real-time processing for Fourier domain optical coherence tomography using a field programmable gate array, *Review of Scientific Instruments*, 79, 2008.

[12] Gross, S. et al; RealTimeFrame A Real Time Processing Framework for Medical Video Sequences, *Acta Polytechnica*, 48(3): 15-19, 2008.

[13] Eom, T. et al; Real time display Fourier-domain OCT using multi-thread parallel computing with data vectorization, *Optical Coherence Tomography and Coherence Domain Optical Methods in Biomedicine*, http://dx.doi.org/10.1117/12.874476 , 2011.

[14] Pan, Y. et al; Continuous Wavelet Transform On Reconfigurable Meshes, *Proc. of the 15th International Parallel and Distributed Processing Symposium (IPDPS 01)*, 1158-1163, 2001.

[15] Thulasiraman, P. et al; A fine-grain load-adaptive algorithm of the 2D discrete wavelet transform for multithreaded architectures *Journal of Parallel and Distributed Computing*, 64: 68-78, 2004.

[16] Diduch, L. et al; A Framework for Modular Signal Processing Systems with High-Performance Requirements *IEEE Int. Conf. on Multimedia & Expo (ICME)*, Beijing, China, 1159-1162, 2007.

[17] Žorić, A. et al; PC-based system for electrocardiography and data acquisition, *Scientific Research and Essays*, 7(4): 468-476, 2005.

[18] Žorić, A. et al; Wireless electrocardiography system, *Proc. of 7th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services TELSIKS*, Niš, Serbia and Montenegro, 7(4): 468-476, 2012.

[19] Dunbar, M.; Plug-and-Play Sensors in Wireless Networks *IEEE Instrumentation and Measurement Magazine*, 4(1): 19-23, 2001.

[20] Smith, W.; *The Scientist and Engineers Guide to Digital Signal Processing*, California Technical Publishing, 1997.

[21] Misiti, M. et al; Wavelet Toolbox For Use with MATLAB, http://www.mathworks.com/help/pdf_doc/wavelet/wavelet_ug.pdf