Second Prize

Nios II Processor-Based Self-Adaptive QRS Detection System

Institution: **Indian Institute of Technology, Kharagpur**

Participants: **Sai Prashanth, Prashant Agrawal**

Instructor: **Professor Agit Pal**

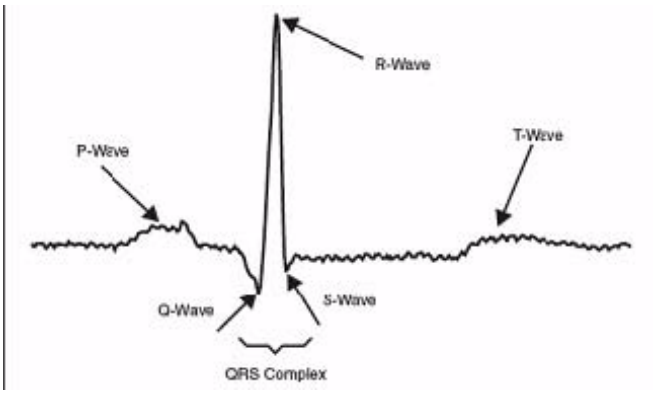
Design Introduction

For our project, we designed and implemented a cardiac arrhythmia electrocardiogram (ECG) monitoring system that can adaptively modify and change the components of its processing chain to carry out the best treatment on electrocardiogram signals. We investigated and provided a solution to a fundamental problem in the area of biomedical signal processing: accurate QRS complex detection for varying environmental and patient conditions. We implemented the ECG monitoring system in an Altera® Cyclone® II FPGA.

Background

The QRS complex is the most striking waveform within the ECG. Because it reflects the electrical activity within the heart during the ventricular contraction, the time of its occurrence and its shape provide a wealth of information about the current state of the heart. Due to its characteristic shape (see Figure 1), it is the basis for automated determination of the heart rate, an entry point for classification schemes of the cardiac cycle, and often used in ECG data compression algorithms. Therefore, QRS detection provides the fundamentals for almost all ECG analysis algorithms.

Figure 1. QRS Complex within the ECG signal

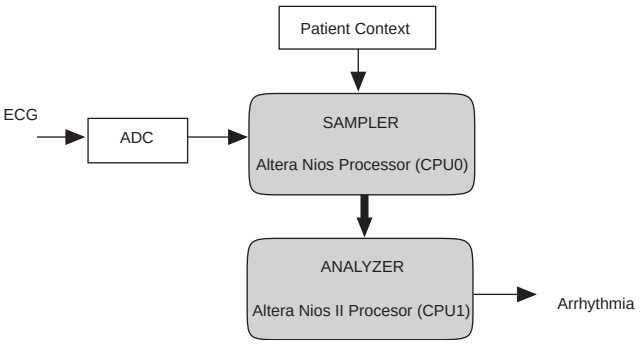


Software QRS detection has been a research topic for more than 30 years. However, experience gathered over the years shows that the proposed strategies for ECG analysis [2], and particularly QRS complex detection based on signal processing techniques, have reached an asymptotic detection performance. This situation exists is because most algorithms operate optimally in a given set of contexts (environmental and/or patient-related), and produce increasing error rates when the contexts are not matched. Therefore, choosing the QRS detection algorithm best suited to the current context is an essential step in the development of a real-time ECG analysis system. Our implementation adopts the real-time piloting system proposed by F. Portet and G. Carrault, in "Piloting Real-Time QRS Detection Algorithms in Variable Contexts" (see "References" on page 332), however, we adapted the system for optimal performance using the Altera Nios® II processor.

Project Outline

In medical monitoring, reducing false alarms and missed detections is crucial, and its importance cannot be overemphasized. Our novel adaptive, algorithm-bank-based solution reduces the number of errors by performing a periodic sampling of the input ECG signal and making a dynamic decision to find the most appropriate algorithm for QRS detection under the current context. Figure 2 shows the design overview of our ECG monitoring system. The gray area represents sub-units within the scope of this project, and is implemented using the Altera Cyclone II FPGA.

Figure 2. ECG Medical Monitoring System



Our ECG monitoring system has two distinct components: an analyzer that performs the actual QRS complex detection and medical diagnosis, and a sampler that performs acquisition, context analysis, and piloting of the analyzer. When a change occurs in the input data, the sampler should react to adapt itself to the new context. Otherwise, the analyzer transmits erroneous data and causes false alarms and low-quality diagnosis.

FPGA Design Significance

The trend in embedded system design is towards implementing the entire functional system on a single chip. The advent of high-density FPGAs with high-capacity RAM blocks and support for soft-core processors such as Altera's Nios II processor have enabled designers to implement a complete system on a chip. We use FPGAs, and in particular, the Altera Nios II soft-core processor to take advantage of the following benefits:

- Altera Cyclone II FPGA systems are portable, cost effective, and consume considerably less power compared to PCs. This fact is important when the application is designed for battery-operated devices. We can implement a complete system easily on a single chip because complex integrated circuits (ICs) with millions of gates are now available.
- SOPC Builder can trim months from a design cycle by simplifying and accelerating the design process. It integrates complex system components such as intellectual property (IP) blocks, memories, and interfaces to off-chip devices including application-specific standard products (ASSPs) and ASICs onto Altera high-density FPGAs.
- The Altera Nios II processor supports hardware/software co-design in which the time-critical blocks are written in HDL and implemented as hardware units, while the remaining application logic is written in C. The challenge is to find a good trade-off between the two. Both the processor and the custom hardware must be optimally designed such that neither is idle or under-utilized.
- FPGAs provide the best of both worlds: a microcontroller or RISC processor can efficiently perform control and decision making operations while the FPGA can perform digital signal processing (DSP) operations and other computationally intensive tasks.
- The Altera Nios II processor supports multi-core processing, which enables off loading and time-sharing critical mutually independent operations between two processor cores to offer real-time response in crucial situations. The synchronization between the processors is easily facilitated by the Avalon® bridge fabric.

Nios II-Based Design

We decided to use the Nios II processor after analyzing the various requirements for a real-time ECG medical monitoring system. A handheld, battery-operated medical monitoring system requires that the design be optimized for performance and energy efficiency. Altera offers easy customization of both these features. A basic system requires application programs, running on a customizable processor that can implement custom digital hardware for computationally intensive operations such as fast discrete cosine transform (DCT) functions, matrix inverse calculations, etc. Using a soft-core processor, we can implement and customize various interfaces, including serial and parallel. The Altera development board, user-friendly Quartus® II software, SOPC Builder, Nios II Integrated Development Environment (IDE), and associated documentation enable even a beginner to feel at ease with developing an SOPC design. We can perform hardware design and simulation using the Quartus II software and use SOPC Builder to create the system from readily available, easy-to-use components. With the Nios II IDE, we easily created application software for the Nios II processor with the intuitive click-to-run IDE interface. The development board's rich features and customization, SOPC Builder's built-in support for interfaces (such as serial, parallel, and USB), and the easy programming interface provided by the Nios II hardware application layer (HAL) make the Nios II processor and an FPGA the ideal platform for implementing our ECG medical monitoring system.

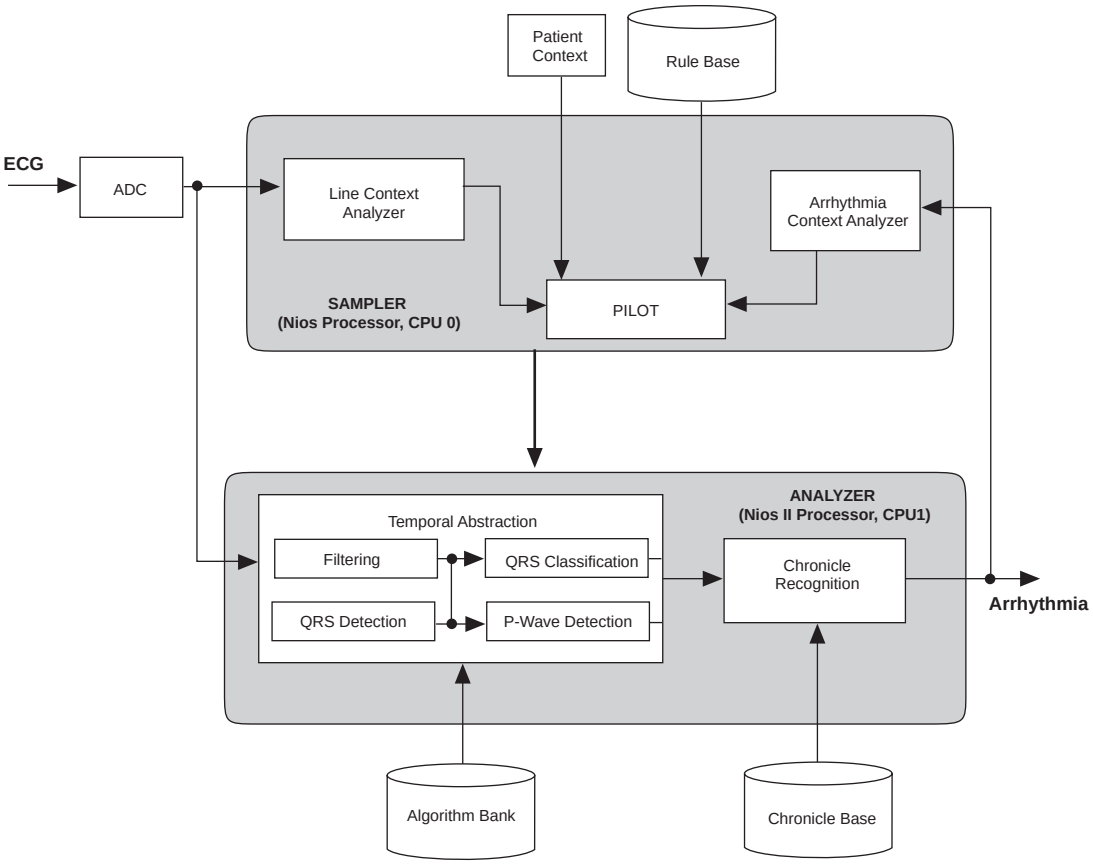
Application Scope and Target Users

Our design is customized for optimal real-time response, which is critical in a medical setting such as an electrocardiogram monitoring system. The design is implemented on a Cyclone II FPGA, and is very power efficient, which makes it suitable for handheld, battery-operated devices like the Holter ECG monitoring systems. It can also be used as a stand-alone clinical system for accurate patient heartbeat monitoring in hospitals and ambulances.

Functional Description

Our ECG monitoring system is devoted to cardiac arrhythmia recognition. Arrhythmia can be diagnosed from the morphology of the P and QRS waves and their temporal relationships. Our system computes a diagnosis from an abstracted representation. Figure 2 shows a high-level overview of the design. The analog electrocardiogram signals are first digitized using an external analog-to-digital converter (ADC) and are fed into to the system through the serial port. The system is composed of two on-line main modules: a sampler and an analyzer, each of which is implemented using a separate Nios II processor. Figure 3 shows a detailed overview of the module.

Figure 3. ECG Medical Monitoring System



At a high level, the sampler module continuously samples the input ECG signals to analyze the line context, and combines this information with the arrhythmia context of the higher-level patient context information. The sampler is governed by a pilot, which uses a set of statistically obtained piloting rules to determine the context. When a change of context is detected, it triggers the analyzer module to switch the algorithm used for the temporal abstraction, i.e., the QRS detection algorithm. The analyzer module consists of the signal processing algorithms that detect and classify the ECG events from the ECG signal. The chronic recognition module analyzes the vents flow and computes the diagnosis. The ECG monitoring system is piloted in three ways: the pilot activates and deactivates the temporal abstraction tasks, chooses and tunes the signal processing algorithms, and selects the level of detail that the arrhythmia recognition needs.

Arrhythmia Recognition Piloting

An arrhythmia can be diagnosed according to several ECG features. In our system, all features are constantly extracted and sent to the arrhythmia recognition, but in some contexts, a reduced number of features can be sufficient to recognize an arrhythmia. Thus, the arrhythmia recognition piloting involves choosing the chronicle abstraction level to recognize by selecting corresponding chronicle models, according to the current diagnosis hypotheses.

Temporal Abstraction Piloting

Temporal abstraction is composed of four linked tasks that extract four main features:

- Filtering separates the actual ECG signal from the noisy part of the signal
- QRS detection identifies QRS occurrence dates
- QRS classification labels QRS morphologies
- P wave detection identifies P wave occurrence dates

Depending on the context chosen by the arrhythmia recognition piloting system, a subset of the temporal abstraction piloting unit is activated. To be more efficient and to base the recognition on reliable information, the architecture enables the activation and deactivation of the temporal abstraction tasks according to the needs and to specific contexts.

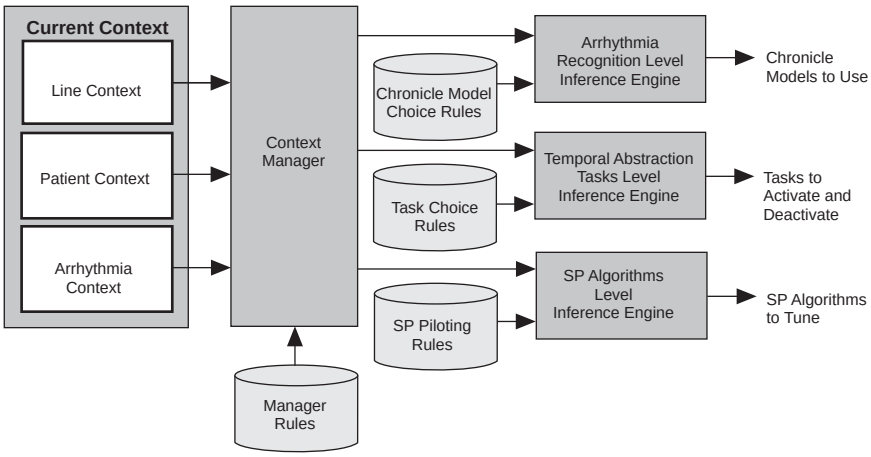
SP Algorithm Piloting

The temporal abstraction tasks are performed by shortest path (SP) algorithms. In our system, a unique SP algorithm is devoted to a particular task. However, related literature describes several possible algorithms, whose performance vary according to the context, to achieve these tasks. The preliminary study, described in [2], showed that the performance of various QRS detection algorithms change with the current context (line noise and QRS morphology). The new extended algorithm base contains several SP algorithms for each task. Therefore, the pilot must choose the algorithm best suited for the current context and then tune its parameters.

Pilot

Figure 4 shows the pilot architecture. It has three inference engines that deduce the actions to perform on the system for the three piloting levels and a context manager that deduces the information needed by the engines from the current context. The context manager instantiates and updates useful variables from the raw information transmitted by the context analyzers. Its knowledge is represented by expert rules stored as rules of thumb in the manager rule base. The system is piloted at three levels: the arrhythmia recognition level, the temporal abstraction tasks level, and the SP algorithms level. From the information transmitted by the context manager, the engines infer the actions to perform on the system. Their piloting rules are mainly defined by an expert and are grouped: chronicle model choice rules, task choice rules, and SP algorithm choice rules. The chronicle recognition adapts the abstraction level to the context. The temporal abstraction tasks are activated according to the needs and to technical constraints. The SP algorithm choice rules determine the algorithm best-suited to the task according to the temporal abstraction tasks and tune it.

Figure 4. Pilot Architecture



We used four real-time QRS detectors:

- *pan*—The Pan and Tompkins [3]
- *gritzali*—The Gritzali's detector [4]
- *df2*—The Okada's detector modified by [5]
- *af2*—A derivative QRS detector modified by [5]

We obtained the QRS detection piloting rules by performing a statistical analysis. We inferred the following rules:

IF <*L* and *bw* and *SNR* >= -5 dB>
THEN <*choose Gritzali's QRS detector*>
IF <(*L* or *F*) and *no noise*>
THEN <*choose Gritzali's QRS detector*>
IF <(*F* or *P*) and *bw* and *SNR* >= 0 dB>
THEN <*choose Gritzali's QRS detector*>
IF <*em* and ((*N* or *A* or *P* or *R*) and *SNR* = -15 dB)>
THEN <*choose df2 QRS detector*>
IF <*em* and (*SNR* = -5 dB and *P*)>
THEN <*choose df2 QRS detector*>
IF <*default*>
THEN <*choose PAN's QRS detector*>

The first rule means that if the line context has the value bw noise at -5 db and the arrhythmia context informs that it has mainly QRS of form L, then the Gritzali's detector is chosen.

Performance Parameters

For this system, accuracy and identification speed are the most important performance parameters; therefore, we focused on these areas. Using the Altera Quartus II design platform, we were able to speed up the design without lowering the design complexity. A single identification, including complex preprocessing, context checking, accelerated C-to-hardware acceleration (C2H) preprocessing, and hardware QRS complex detection, should be performed in real time to operate on the data streaming in. According to the MIT-BIH database (from the Harvard-MIT Division of Health Sciences and

Technology) benchmark experiment data, the threshold gives a very low 10.6percent error rate, which is considerably lower than the 14.3 percent error rate obtained when no sampler is used.

SOPC Builder allows the user to configure additional aspects of the microprocessor to improve computation speed, at the expense of using more system memory and logic elements (LEs). Specifically, the user can control the core type (Nios II/s, Nios II/f, etc.), pipelining, hardware multiply and divide, and cache allocation. Pipelining allows multiple instructions to be fed into each stage of the microprocessor execution cycle in parallel, enabling maximum execution performance of the navigation system software. Larger caches provide more memory data storage, which makes code execute faster. A large cache is particularly useful for the monitoring system software, which uses an incremental iterative process (i.e., values in a discrete wavelet transform (DWT) matrix are updated in a scanned incremental manner) to determine the DWT. However, larger caches also use more FPGA LEs and memory, and the designer can inadvertently create a system that does not fit into the target Cyclone device. Ultimately, we selected the cache size and pipelining based on trial and error, with the goal of maximizing the cache size while still fitting the design into the Cyclone FPGA.

Performance is assessed by the number of errors (Ne), which reflects both false alarms and missed beats. For each test, FN (the number of false negatives or missed QRSs) and FP (false positives or false alarms) are computed to obtain Ne = FP + FN. The error rate is Er = Ne / NQRS, where NQRS is the total number of actual QRSs. The study leads to 16,000 Ne values, and for this amount of data, we performed a principal components analysis (PCA) to analyze the detector results graphically. To test the piloting rules, five ECGs were generated from the MIT-BIH database. Each ECG lasted from 20 to 30 minutes for a total of about 2 hours. Three to four different contexts are introduced in each test ECG to assess the system performances in the specific contexts as well as around the context transitions. Parts of the original ECGs were corrupted with the three real clinical noise types defined previously (bw, ma, and em). In each context, the pilot chooses the best algorithm with the aid of the piloting rules. In this study, the algorithm thresholds are optimal in the sense that Ne is minimum. See Table 1.

Table 1. QRS Detection Results for Different Detectors and Pilot

ECG Score	Ne 1	Ne 2	Ne 3	Ne 4	Ne 5	Total	
						Ne	Er (%)
Pan	*20	*91	*240	*312	*367	1,030	14,3
Gritzali	20	*160	388	360	*295	1,223	17
df2	307	278	*174	*160	*302	1,221	17
Pilot	20	88	185	167	304	764	10,6

C2H Compiler

The Nios II C2H Compiler can automatically integrate high-performance C programs into the hardware accelerator, which is then integrated into the FPGA-based Nios II subsystem. The C2H Compiler supports standard ANSI C code, accelerates multiple application programs, and improves operational efficiency, including access to local and external memory and peripherals. We used SOPC Builder to generate a broadband Avalon interconnected architecture, which processes the external memory and peripherals, such as pointer dispersal and array access. The Nios II C2H Compiler accelerates implementation of memory interfaces, and generates hardware accelerator logic and the correct Avalon host and slave interface to match the memory delay. It shares the data computing and memory access functions with the Nios II processor, and lets the processor perform other tasks. Because the Avalon architecture does not limit the number of hosts and slaves in a system, the Nios II C2H Compiler can generate multiple hardware accelerators according to the target code's transfer requirements. The C2H Compiler helps embedded system developers improve design efficiency. In our system, the signal preprocessing function is implemented in software. Because we have high-speed identification requirements and the C software code takes a long time to perform the task, we optimized the ECG signal preprocessing module with the Nios II C2H Compiler to accelerate processing. We tested the implementation speed. With this optimization, the design uses extra logic resources: 65% instead of 20% without optimization.

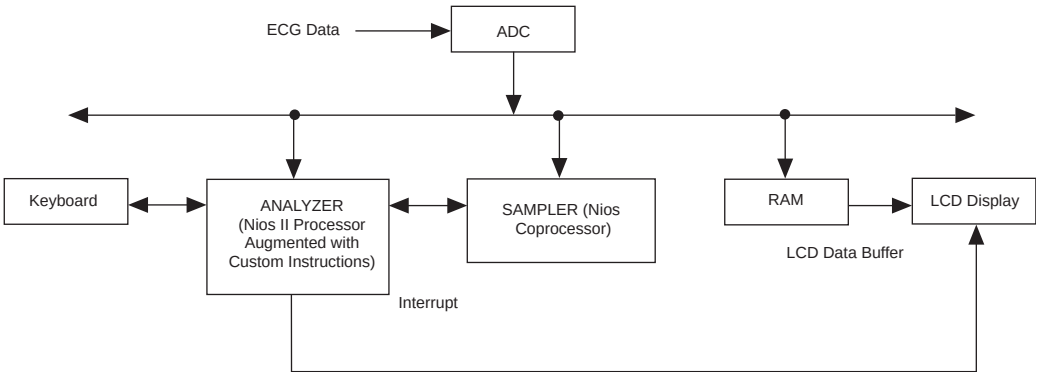
Nios II Processor

The Nios II processor's excellent performance facilitated our design. We chose the Nios II/f CPU because of our high-speed processing requirements. We combined the processor, peripherals, memory, and I/O interface with the Nios II processor and FPGA design. Because the Nios II processor is configurable, we could modify the system performance requirements at any time. Furthermore, we were able to improve the module performance with Nios II custom instructions.

Design Architecture

The Figure 5 shows an abstract hardware design block diagram of the ECG monitoring system.

Figure 5. ECG Hardware Block Diagram



The hardware system consists of two Nios II processors, which implement the sampler and analyzer modules, a SRAM, an external keyboard for user input, and an LCD display for graphically displaying the ECG signal. The input analog ECG signals are converted to digital format using an external ADC. The Avalon tristate bridge provides seamless communication between the various components of the system. Figure 6 shows the general software flow chart of the QRS detection algorithms. All QRS detection algorithms used in our context follow this methodology for identifying the QRS complex.

Figure 7 shows the user interaction state transition diagram and is important to understand the system operation. The user can interact with the ECG monitoring system via the keyboard and select the required operation mode. Some important operation modes are acquire signal from patient, analyze data, retrieve data, and transmit data.

Figure 6. Software QRS Detection Algorithm Flow Chart

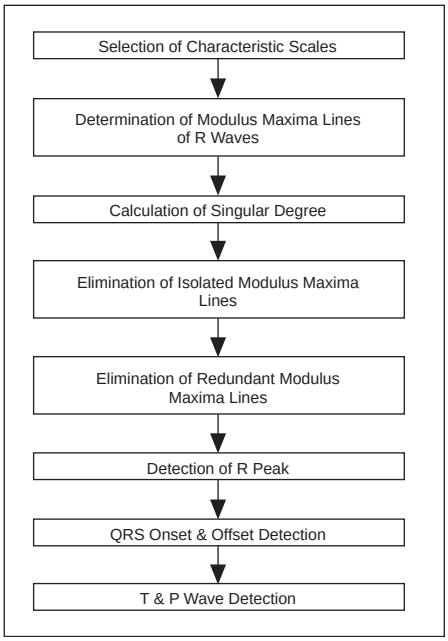
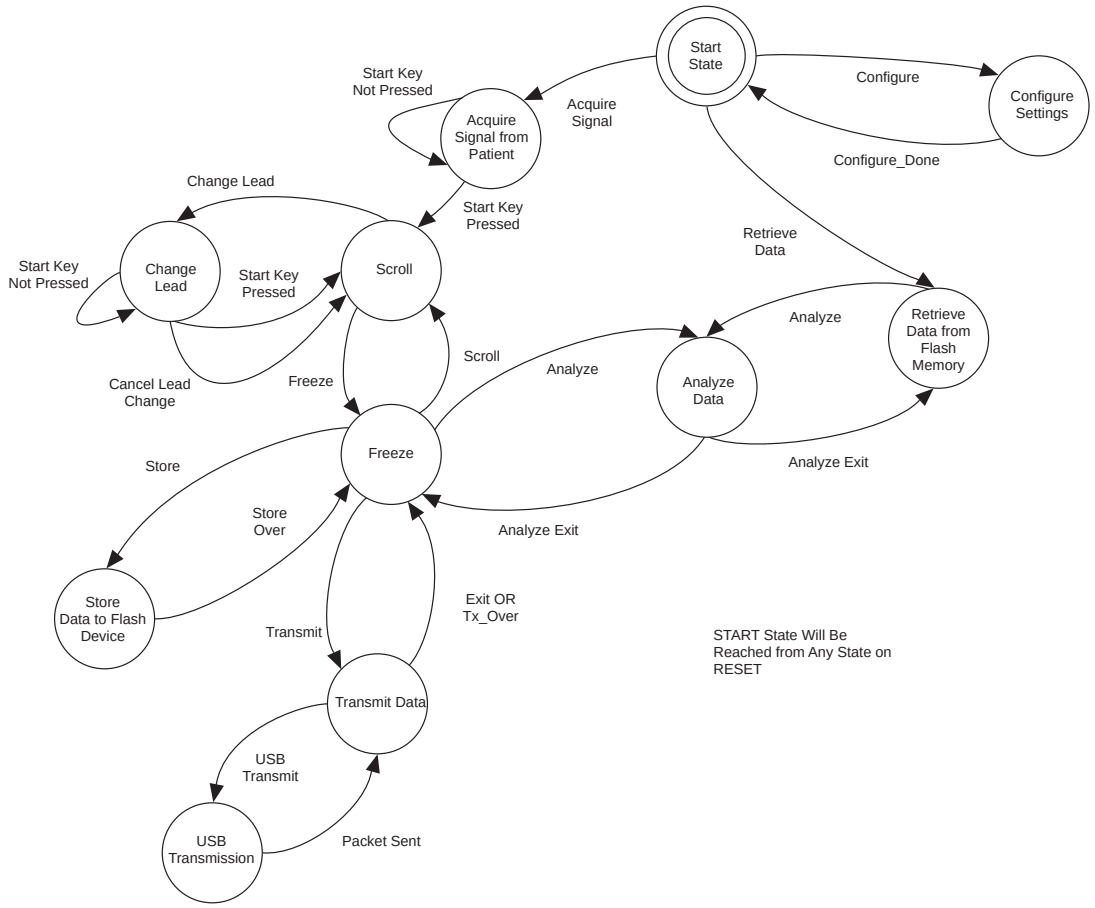


Figure 7. User Interaction State Transition Diagram



Design Description

The design's implementation steps are as follows:

1. Research and determine a set of complementary QRS detection algorithms that work effectively in a mutually exclusive set of contexts, and develop software algorithms to support them. Research Altera FPGAs using the Quartus II software, SOPC Builder, and Nios II IDE. Use the web editions of the software and documentation available from the Altera web site.

2. Create a Quartus II project, selecting appropriate Nios II dual-core processors in SOPC Builder, as shown in Figure 3 on page 322. Compile and debug the Quartus II design and review the compilation report. Test the project, including testing the processor response, error rate, and miniboard hardware (UART communications, etc.), with the simple hello world Nios II program.

3. Create an algorithm bank consisting of four different QRS complex detection algorithms implemented in the Nios II processor. Test the performance in the Cyclone device with the appropriate test input.

4. Update the SOPC Builder processor configuration to determine the fastest possible configuration that fits in the device's memory and available LEs. Optimize the processor until the desired performance requirement is met.

5. Create interrupt-based interfaces using the Nios II IDE to control the appropriate input and output to integrate the ECG medical monitoring system with other systems. Test these I/O interfaces.

6. Test the ECG medical monitoring system performance using standard available electrocardiogram signals from the MIT-BIH database and determine the error rates of the QRS complex detection.

Design Environment

We used the Altera Cyclone II development board for the initial code debugging and then used the development and education (DE2) development platform for final implementation. The DE2 board has a variety of integrated peripheral interfaces that were convenient to use in our design.

Software and Hardware Design

Figure 8 shows the SOPC Builder configuration.

Figure 8. SOPC Builder Configuration

The two important modules in the ECG monitoring system are the sampler module and the analyzer module. We implemented them using separate Nios II processors to facilitate real-time response to the streaming in electrocardiogram signal. The sampler module is implemented using **cpu0**, and it incorporates the line context analyzer and the pilot. The line context analyzer analyzes the quality of the incoming ECG signal and determines the decibel noise level. It also has an arrhythmia context analyzer, which contains information about the QRS morphologies that occurred in the past. Using this information, in addition to the high-level patient related context information, the pilot then uses statistically determined piloting rules to decide which context is most suitable for the analyzer to use. It makes a decision dynamically and interrupts the analyzer module to change its processing cycle.

The analyzer module consists of the temporal abstraction unit, which is composed of the signal processing algorithms and the chronicle recognition unit. Depending on the interrupt received from the sampler module, the analyzer uses the appropriate QRS detection algorithm for processing the ECG signal, as outlined in “Functional Description” on page 322. Upon appropriate processing, the morphologies are then passed to the chronicle recognition unit to determine any arrhythmia, which can then be subject to medical diagnosis. Figure 9 shows the block diagram for system implementation using the Quartus II software.

Applying SOPC Concepts

Altera introduced system-on-a-programmable-chip (SOPC) technology and its related development platform, the Quartus II software. SOPC is the FPGA version of system-on-chip (SOC). Compared to ASIC SOC, SOPC has many unique features. Our design uses SOPC concepts in the following ways:

- *Modular system design*—At the beginning of the system design, we partitioned the system into a line context analyzer (preprocessing) and processing. The system is divided and simplified, which makes it easier to implement. According to the module interfaces, we can accurately evaluate the design's application scope and future development at the initial design stage. We can perform market exploration and product research and development simultaneously in the practical product design, which shortens the time-to-market and accelerates enterprise development.

- *System integration*—An embedded system shows its features with its size, power consumption, and integrity. Except for the expanded 1-Mbyte SRAM front-end collection module, we could implement all system functions on the development board. It is very difficult to implement such a highly integrated design without lowering the design target or using a different FPGA.

- *Various modes*—We can diversify the implementation. For example, the front-end module uses an IP core, preprocessing is implemented with software, and key steps are fulfilled using the C2H Compiler to transfer operations to hardware. The trademark checking module uses hardware, software, or hardware/software with custom peripherals and instructions. Using SOPC concepts and excellent design tools enabled us to use these various modes.

- *Final system can be upgraded*—The design can be flexibly configured and updated during the design process.

Figure 9 shows the Quartus II system implementation block diagram.

Figure 9. Quartus II System Implementation Block Diagram

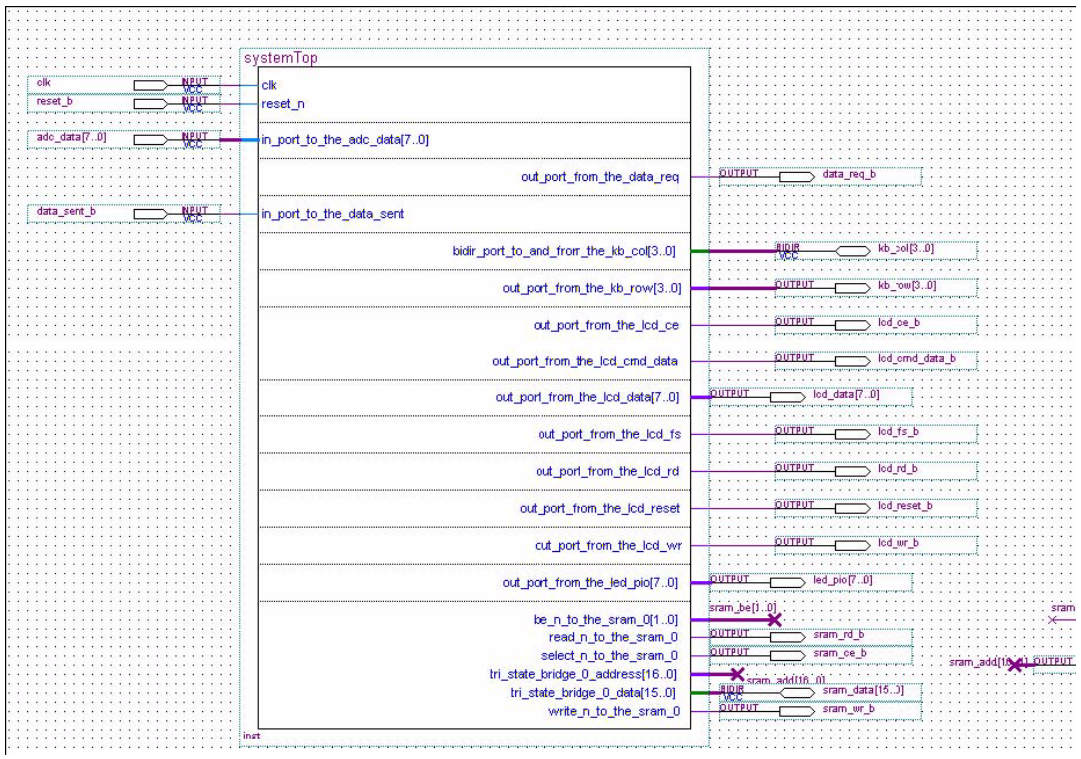


Table 2 shows the percentage of execution time spent in each of the sub-functions that constitute the QRS detection algorithm. The `dwt_ecg` function, which performs the DWT computation, consumes the most time. The computationally intensive portions of the QRS complex detection algorithm are implemented using custom instructions. The most time-consuming function computes the DWT, which

is implemented using custom instructions. Prior to this, loop unrolling was performed on the initial code to reduce the number of iterations required to perform a single DWT calculation.

Table 2. Execution Time for Various Functions

Function	% of Total Time
dwt_ecg	69.2
detect_mm_R	9.99
detect_r	0.13
detect_qrs	0.16
detect_t	13.7
detect_p	6.7

Design Features

Our ECG medical monitoring system has the following features:

- It is a standalone system for detecting QRS complexes in an electrocardiogram for further medical diagnosis without using a PC for recognition.
- Performs accurate QRS complex detection under varying conditions, where any single algorithm would fail to function effectively. These conditions include environmental disturbances (such as noise due to electrical interference, muscular activity, or loss of contact) and patient characteristics (i.e., varying heart beat classifications).
- Custom instructions are optimized for area and energy using Nios II architectural features, such as an extended custom instruction architecture (for resource sharing) and internal registers (to reduce memory access latency). These features reduce the device cost.
- Run-time electrocardiogram signal acquisition, processing, and morphology recognition makes the system suitable for practical use in Holter ECG systems, clinical use, etc. Implementation is optimized for minimal latency by exporting computationally intensive parts of QRS complex detection algorithms to custom instructions, and using the periodic sampling subunit as a co-processor. This technique enables the Nios II processor to exhibit real-time performance, which is critical in biomedical signal processing applications such as heart beat monitoring.
- The algorithm software is also optimized (using techniques such as loop unrolling) with the C2H Compiler.
- Overall energy-efficient system design enables use of the design in hand-held, battery-operated devices, such as Holter systems.
- SOPC design plays a central role in all design features, and enables easy optimization for minimal latency, area, and energy consumption. Several Nios II architectural and support features ease the process of system design and development.
- Displaying the ECG signal on a liquid crystal display (LCD) aids a specialist in deducing graphical conclusions from the morphologies.
- The design uses a variety of features and components available for Nios II-based development, such as PIO, UART, and RS- 232 communication. In the future, we would like to implement USB communication as well so that we can provide a standalone ECG monitoring system that can automatically log data in an auxiliary storage device for archiving.

- The system has low cost and high performance. The single chip ECG monitoring system is small, easy-to-carry, and cost effective, which satisfies the needs of most engineering technicians. Compared to the expensive medical monitoring systems currently on the market, this system provides excellent performance at a lower cost.

- The system is portable: the single FPGA and Nios II processor can implement ECG signal collection, analysis, storage, control, and transfer, which allows the ECG monitoring system to migrate from large a desktop to small handsets. Additionally, it provides portable terminals suitable for working outdoors.

Conclusion

The Altera Nios II design contest enabled us to develop a better understanding of the Nios II processor. Using it, we were able to design our system easily, including dual-core embedded processors, on-chip and off-chip memory, and high-speed I/O ports. Altera development tools let us develop our own multi-functional custom instructions quickly. Additionally, we could modify the CPU hardware at any time for multi-purpose development using SOPC Builder. We hope to use the Nios II IDE debug function in future to shorten the software development time significantly. Altera's ability to develop and update the Nios II processor and functions was extremely important. For example, using custom instructions we could accelerate the hardware computation speed, which improved our system's efficiency. We thank Altera for having the contest and acknowledge their support when we had design problems. On the whole, using SOPC concepts allowed us to create a more flexible, dynamically reconfigurable, and computationally intensive implementation.

References

[1] F. Portet and G. Carrault, "Piloting Real-Time QRS Detection Algorithms in Variable Contexts," *IFMBE Proceedings*, Volume 11, Prague/Czech Republic 2005.

[2] B.U. Kohler, C. Hennig and R. Orglmeister, "The Principles of Software QRS Detection," *Engineering in Medicine and Biology Magazine, IEEE*, Volume 21, Issue 1, pp 42-57, Jan./Feb. 2002.

[3] Pan, J. and Tompkins, W.J. "A real-time QRS detection algorithm"

[4] Gritzali, F. "Towards a generalized scheme for QRS detection in ECG waveforms"

[5] Friesen, G.M., Jannett, T.C., Jadallah, M.A., Yates, S.L., Quint, S.R., and Nagle, H.T. "A comparison of the noise sensitivity of nine QRS detection algorithms"