Efficient Implementation of Biomedical Hardware Using Open Source Descriptions and Behavioral Synthesis

George Economakos, *Member, IEEE*

***Abstract*— Medical diagnostics are changing rapidly, aided by a new generation of portable equipment and handheld devices that can be carried to the patient's bedside. Processing solutions for such equipment must offer high performance, low power consumption and also, minimize board space and component counts. Such a multi-objective optimization can be performed with behavioral hardware synthesis, offering design quality with significantly reduced design time. In this paper, open source code of a QRS detection algorithm is implemented in hardware using an advanced behavioral synthesis framework. Experimental results show that with this approach performance improvements are introduced with a fraction of design time, reducing dramatically time-to-market for modern diagnostic devices.**

I. INTRODUCTION

Advances of *Information and Communication Technologies* (ICT) in the health sector are changing rapidly the way medical diagnostics are delivered today. New, small size, low power and improved efficiency integrated circuits can be manufactured. The communication infrastructure can connect previously isolated or abandoned areas with a wealth of information flow. As a consequence, a new generation of portable, hand-held, wearable or implantable equipment has emerged that can follow the patient in different geographic locations and through different activities.

Components designed to fit in this increasingly pervasive sensing network, offering higher processing power and the ability to transfer larger amounts of information more quickly, should offer advanced characteristics. In terms of area, they should be small enough to be carried away. In terms of processing power, they should be efficient to deliver computation speed and advanced to cover demanding applications. In terms of power consumption, they should save energy to increase battery life and thus availability time as well. So, their design should follow a multi-objective optimization path, from concept to implementation.

Such a multi-objective and much promising design technique is *Behavioral* or *High-Level Synthesis* (HLS) [4], [5]. HLS raises the level of design abstraction by translating system level algorithmic descriptions into *Register Transfer Level* (RTL) architectural descriptions. Although HLS has been a research topic for more than twenty years, it has recently gained industrial acceptance with the introduction of hardware description languages like VHDL and Verilog in design flows, and the availability of efficient synthesis methods and tools, that enable the translation of RTL designs into optimized gate level implementations. The main expectations from HLS is support for better management of the design complexity and reduction of the design cycle all together, breaking the trend to compromise evaluation of various design implementation options. Designing at higher levels of abstraction allows a better coping with the system design complexity, to verify earlier in the design process and to increase code reuse.

The design of medical diagnostic environments has employed computer analysis of vital biosignals in many cases during the last years. However, new companies are constantly emerging and applying new technologies, such as PDAs, in an effort to make smaller and cheaper systems. Each new company must implement their own analysis algorithms, duplicating much of the the efforts of every other company. Similarly, researchers who need to explore new diagnostic methods, must also implement their own versions of basic analysis functions. Thirty years of research on computer analysis of vital signals has produced a great many methods for detecting and classifying characteristic patterns, but there is still a significant effort required to go from theory to implementation.

In an effort to reduce this industry and research wide duplication of effort, open source analysis software [7], [1] has been proposed. C functions have been developed and made widely available, that implement the most basic ECG analysis operations, detection and classification of individual beats. Using this open source software new companies are able to bring reliable systems to market more quickly, and researchers are able to spend more time exploring new diagnostic techniques rather than implementing beat detectors.

In this paper, this open source software is passed through a commercial HLS tool [12], to offer the same advantages in a higher degree, for the design of modern, powerful embedded medical diagnostic devices. Specifically, the development time is greatly reduced (compared to other hardware design techniques), implementation quality is comparable to manual designs, code reuse is maximized, simulation time is reduced and there is no need to hire a specialized hardware design team (at least for prototype implementation). The resulting hardware components can be used as stand alone or co-processing elements in a *System-on-Chip* (SoC) architecture, using appropriate interface components. Moreover, this approach is not limited to ECG analysis but can be applied to any application for which an algorithmic C/C++ description is available, open source or proprietary.

Manuscript received July 5, 2008.

G. Economakos is with the School of Electrical and Computer Engineering, Division of Computer Science, National Technical University of Athens, Heroon Polytechniou 9, 15780 Zografou, Greece geconom@microlab.ntua.gr

The rest of the paper is organized as follows. Section II is a presentation of related research activities. Section III presents details about the optimizations offered through HLS for the design of diagnostic applications. Section IV gives results from the conducted experiments and finally, section V is the conclusion and the expected future extensions.

II. RELATED RESEARCH

Computer analysis of biosignal and especially the ECG signal is not new [8], [14]. Recently, special purpose hardware devices are proposed [2], [3], [9], [10], [13], [16], based on the wide adoption of hardware description languages and *Field Programmable Gate Arrays* (FPGAs).

More than half of the above referenced hardware implementations deal with new algorithms for ECG QRS detection. In [10] a wavelet based approach is presented, in [16] mathematical morphological filtering is put to use while in [2], [3] geometrical properties of a phase-space portrait are exploited. All approaches present more than 99.50% sensitivity of QRS detection on the MIT-BIH arrhythmia database [11]. However, not many details about the methodologies and the design decisions taken during hardware design are given. In [2] a working frequency of 82MHz is reported while in [10] the frequency reported is 73MHz. In [3] the device presented in [2] is used as a coprocessor in an embedded system with a general purpose microprocessor.

In the remaining two referenced implementations, a circuit-aware presentation is given. In [13] comparisons are given between implementations of different QRS detection algorithms reaching an operating frequency of 34MHz. In [9] a low power implementation, used for implantable devices is presented. This final implementation is considered for fabrication as an *Application Specific Integrated Circuit* (ASIC), compared to the FPGA prototypes of all other cases. Also, detailed power measurements are given instead of operating frequencies.

Our approach is similar to the one in [13] but uses a higher level of design abstraction offering reduced design time and increased code reuse and overall productivity. The results obtained are better in terms of operating frequency and comparable to those given in [10], [2], [3], for the same hardware implementation FPGA platform. The advantage of our work is that more design alternatives are considered, resulting in better design space exploration.

III. BEHAVIORAL OPTIMIZATIONS

Our design exploration methodology is based on a commercial behavioral optimization tool, CatapultC synthesis from Mentor Graphics [12], used also in the past for industrial telecommunication applications [6]. As is well-known (e.g. [4], [5]), HLS is the process of transforming a system level behavioral specification of a digital system into an RTL structural description implementing that behavior. HLS acts upon the dataflow graph of an application with the following basic behavioral transformations:

- **Allocation:** select the appropriate number of functional units, storage units and interconnect units from available component libraries.
- **Scheduling:** determine the sequence in which every operation is executed.
- **Binding:** assign operations to functional units, values to storage units and connect these components to cover the entire datapath.

These transformations are related to each other and no problem can be solved without considering the others. Moreover, each problem is known to be NP-hard making exact solutions impractical and thus, different practical heuristics have been proposed and applied over the past twenty years.

Recently, there has been a universal acceptance of hardware description languages like VHDL and Verilog in design flows mainly due to the availability of efficient synthesis methods and tools, that enable the translation of RTL designs into optimized gate-level implementations. Many expect that the same approach could be effectively adapted at higher levels of abstraction. In the emerging System-on-Chip (SoC) context, the traditional IC design methodology relying on EDA tools used in a two stages design flow - a VHDL/Verilog RTL specification, followed by logical and physical synthesis - is indeed no more suitable. Thus, actual complex SoCs need new system level tools in order to raise the specification abstraction level up to the algorithmic / behavioral one. Languages like C/C++/SystemC offer high abstraction levels. However, in order to provide the designers with an efficient automated path to implementation, new high level synthesis tools are required.

The main expectations from the system design teams concern both methods and tools supporting better management of the design complexity and reduction of the design cycle all together, breaking the trend to compromise evaluation of various design implementation options. Designing at higher levels of abstraction is an obvious way as it allows a better coping with the system design complexity, to verify earlier in the design process and to increase code reuse.

CatapultC takes as input a functional description of an algorithm in C or System C and produces a VHDL RTL description of a logic circuit, which implements this algorithm. In particular, CatapultC can be used for obtaining optimized designs, because it considers several alternative implementations of the algorithm and selects one or more among them according to user-specified criteria. These criteria may have to do with performance measures, resource usage constraints, or other characteristics of the resulting circuit. The program also contains a number of visual tools for graphical representation, analysis, comparison and evaluation of the results.

CatapultC synthesis is an industrial class HLS approach that raises the level of abstraction by clearly separating algorithmic function from the actual architecture to implement it in hardware (interface cycle timing, etc.). The inputs to the CatapultC are (a) the algorithmic specification expressed in sequential, ANSI-standard C/C++ or SystemC and (b) a set of directives which define the hardware architecture. The clear separation of function and architecture allows the input source to remain independent of interface and performance requirements and independent of the FPGA target technology. This separation provides important benefits.

- The source is concise, the easiest to write, maintain, and debug. Because of its high-level of abstraction, its behavior can be simulated at much higher speeds (X10000 faster) than RTL, cycle accurate, or traditional behavioral-level specifications.
- The source can be leveraged as algorithmic intellectual property (IP) that may be targeted for various applications and FPGA technologies.
- Obtaining a new architecture is a matter of changing architectural constraints during synthesis. This reduces the risk of prolonged manual recoding of the RTL to address last-minute changes in requirements or to address timing closure or to satisfy power and area budgets.
- By avoiding manual coding of the architecture in the source, functional bugs that are common when coding RTL are also avoided. It is estimated that 60% of all bugs are introduced when writing RTL. The importance of avoiding such bugs cannot be overstated.

Optimization in CatapultC is performed with a number of algorithmic transformations in cooperation with the basic HLS transformations of allocation, scheduling and binding. Some of them are applied in all cases by the tool, like dead code elimination or common subexpression elimination, and others are applied under user selection. A selection of the most widely used optimizations are the following:

- **Speculative Execution:** This optimization analyzes conditional logic (such as if-else) in the design in order to reduce latency. It analyzes all conditional branches in order to find operations that are not dependent on the condition, and therefore can be scheduled in a different clock cycle.
- **Share Mutually-Exclusive Components:** This optimization allows components to be shared by more than one operation if their use is mutually exclusive.
- **Automatically Re-Allocate Components:** Re-allocates a component if it causes a smaller area. This optimization is performed if a smaller component is available that meets timing, or if re-allocating the component results in more sharing. Multi-function components, such as add-subs, will also be targeted by this optimization.
- **Assignment Overhead:** Specify the percent of the clock period that will not be used by sharing and re-allocation. This constraint can be used to reserve some of the clock period for routing delay in the back-end flow. The value can be a negative number.
- **Safe FSM:** This optimization enables support for safe FSM flows. When enabled, CatapultC will allow for a default state that is not reachable from other states during RTL simulation. The default state will unconditionally set the state to the first state in the FSM or the reset state, if such a state exists.
- **Use Old Scheduling and Allocation Algorithms:** This optimization option is used to switch between a new combined allocation and scheduling algorithm and an older separate scheduler. The new scheduler can reduce area on many designs because it is able to change the type and number of allocated components while the design is being scheduled.

A major part of the optimization performed by CatapultC is done in loops, nested or not. This is due to the fact that loops are the areas of an algorithm that require more processing time and thus, any optimization applied to them can have a drastic effect in the final implementation. Such loop transformations are:

- **Loop iterations:** Loop iterations refers to the number of times the loop runs before it exits. The user can constrain the number of loop iterations if the number estimated by CatapultC is not satisfactory.
- **Loop unrolling:** Loop unrolling refers to the number of times to copy the loop body. After the loop iterations have been determined, the loop is unrolled based on the unrolling constraints.
- **Loop merging:** Loop merging is a technique used to reduce latency and area consumption in a design, by allowing parallel execution, where possible, of loops that would normally execute in series.
- **Loop pipelining:** Loop pipelining is how often to start the next iteration of the loop. After loop unrolling and several other transformations are complete, the scheduler uses the pipelining constraints to build a pipelined loop.

Another powerful optimization option in CatapultC is the way to handle hierarchical designs, that is, designs that have more than one C function in their specification. For designs that have such a structure, the user must designate which functions are the entry points of each hierarchical block. Every function will have one of the following settings: Top, Block, or Inline. One function must be designated Top, meaning the toplevel block for the entire design. The entry function for any sub-block is designated a Block. All other functions are designated Inline. CatapultC generates hardware for the top level block and any function called by it recursively, either as sub-blocks or as an inline code substitutions. The default behavior is to allow all functions to be inlined. This allows the maximum level of optimization, but it will also increase runtime. This happens because, each function designated as a sub-block, only maps to one block of hierarchy, regardless of the number of times the function is called. This may lead to designs that do not function at the required performance. On the contrary, each function designated as inline is substituted in all places that is called and is optimized in all these places along with all other neighboring code, leading to more efficient but time consuming overall optimization.

IV. EXPERIMENTAL RESULTS

In order to evaluate HLS design methodologies and optimizations for the development of embedded diagnostic devices, we used an open source software implementation [1] of a classical QRS detection algorithm. The implementation follows the digital filtering approach presented in [8], which applies low-pass filtering, high-pass filtering, derivation and averaging to the input signal in order to isolate QRS complexes. The selection of the specific application is not determined by our approach, which can easily be applied to other algorithms implemented in behavioral C/C++ code. However, its availability as open source code make it available to every researcher or company involved in the field, which gives a clear design time acceleration for the specific flow.

In [1], different implementations of the same basic algorithm can be found. The first, called QRSDET1 in the experimental results presented in this subsection, uses a median filter to find the average of the ECG signal over a period of 80 ms. This solution presents QRS detection sensitivities near 99.7% and QRS detection predictivities near 99.8%. The second, called QRSDET2, uses a mean filter which is much for efficient in both software and hardware implementations. Moreover, it improves QRS detection sensitivities to 99.8% while QRS detection predictivities are unaffected. The third, called QRSDET3, is generated from QRSDET2 after eliminating a search back technique presented in [7], which reconsiders previous samples if a QRS complex has not been found within a 1.5 R-to-R time interval. In this case, QRS detection sensitivities and positive predictivities drop to near 99.7%, which is quite acceptable for diagnostic reasons, related to the performance improvements offered. The final implementation, QRSDET4, ignores all peaks for 200ms following a QRS detection that may lead to large P waves to be detected as QRS complexes and the following QRS complex (within 200ms) to be ignored. The algorithm of QRSDET4 is simpler and faster but the QRS detection sensitivities drop to 99.2% and the predictivities to 99.5%.

For the hardware implementation of the proposed algorithms we have used two types of behavioral transformations: loop unrolling and pipelining. Loop unrolling exposes more algorithmic constructs to the HLS tool (more sentences), resulting in more optimization. Pipelining forces the resulting implementation to run at the specified throughput frequency. In other words, the generated circuit is forced to produce new outputs with the specified frequency, regardless of the overall time needed for all computations. When required, appropriate registers are inserted to hold internal values of the pipeline architecture. In CatapultC, both transformations are selected through the GUI by pressing buttons and inserting appropriate values (i.e. pipeline interval), or through environment variables.

The summary of all conducted experiments is given in table I. The first column of table I shows the implemented algorithm. The second shows the optimizations applied in each case. These are either a full unroll of all loop iterations for loops with known bonds, or a partial unroll of 2 loop iterations in the case of the median filter that has unknown bounds, or the pipeline interval forced on the output of the generated architecture. The third column shows the clock frequency of the device used in each experiment. For

Algorithm	Optimizations	Clock Frequency	Throughput Frequency	Utilization
QRSDET1	None	100MHz	0.1MHz	27.45%
QRSDET1	Full unroll	100MHz	0.11MHz	56.60%
QRSDET1	Full unroll 2xPart. unroll	100MHz	0.19MHz	62.36%
QRSDET1	None	200MHz	0.09MHz	28.80%
QRSDET1	Full unroll	200MHz	0.099MHz	29.70%
QRSDET1	Full unroll 2xPart. unroll	200MHz	0.18MHz	30.13%
QRSDET1	None	400MHz	0.11MHz	29.44%
QRSDET1	Full unroll	400MHz	0.18MHz	60.39%
QRSDET1	Full unroll 2xPart. unroll	400MHz	0.25MHz	67.26%
QRSDET2	None	100MHz	0.47MHz	34.43%
QRSDET2	Full unroll	100MHz	1.01MHz	50.32%
QRSDET2	Full unroll Pipeline 20	100MHz	5MHz	58.94%
QRSDET2	Full unroll Pipeline 5	100MHz	20MHz	85.93%
QRSDET2	None	200MHz	0.50MHz	27.15%
QRSDET2	Full unroll	200MHz	1.47MHz	44.10%
QRSDET2	Full unroll Pipeline 20	200MHz	10MHz	61.12%
QRSDET2	Full unroll Pipeline 10	200MHz	20MHz	69.89%
QRSDET2	None	400MHz	0.58MHz	26.65%
QRSDET2	Full unroll	400MHz	1.15MHz	46.04%
QRSDET2	Full unroll Pipeline 30	400MHz	13.33MHz	79.17%
QRSDET3	None	100MHz	0.46MHz	22.04%
QRSDET3	Full unroll	100MHz	1.16MHz	46.56%
QRSDET3	Full unroll Pipeline 3	100MHz	33.33MHz	99.46%
QRSDET3	None	200MHz	0.61MHz	23.93%
QRSDET3	Full unroll	200MHz	1.57MHz	39.27%
QRSDET3	Full unroll Pipeline 7	200MHz	28.57MHz	88.09%
QRSDET3	Full unroll Pipeline 4	200MHz	50MHz	90.67%
QRSDET3	None	400MHz	0.71MHz	24.06%
QRSDET3	Full unroll	400MHz	1.24MHz	41.49%
QRSDET3	Full unroll Pipeline 20	400MHz	20MHz	65.89%
QRSDET3	Full unroll Pipeline 17	400MHz	23.52MHz	69.67%
QRSDET3	Full unroll Pipeline 5	400MHz	80MHz	97.32%
QRSDET4	None	100MHz	0.46MHz	21.22%
QRSDET4	Full unroll	100MHz	1.14MHz	36.19%
QRSDET4	Full unroll Pipeline 5	100MHz	20MHz	79.56%
QRSDET4	Full unroll Pipeline 3	100MHz	33.33MHz	98.73%
QRSDET4	None	200MHz	0.61MHz	23.07%
QRSDET4	Full unroll	200MHz	1.6MHz	38.04%
QRSDET4	Full unroll Pipeline 7	200MHz	28.57MHz	69.59%
QRSDET4	Full unroll Pipeline 3	200MHz	66.66MHz	89.71%
QRSDET4	None	400MHz	0.72MHz	22.22%
QRSDET4	Full unroll	400MHz	1.6MHz	37.19%
QRSDET4	Full unroll Pipeline 15	400MHz	26.66MHz	64.59%
QRSDET4	Full unroll Pipeline 13	400MHz	30.76MHz	68.41%
QRSDET4	Full unroll Pipeline 4	400MHz	100MHz	95.41%

TABLE I
QRS DETECTION IMPLEMENTATIONS THROUGH HLS

comparison reasons, this device is an FPGA of the Virtex-II Pro device family [15] from Xilinx (XC2VP30FF896-7) in all cases. The fourth column shows the throughput frequency of the generated hardware device and is related to the selected pipeline interval. Finally, the fifth column shows the percentage of the overall device resources (logic cells, flip-flops, DSP blocks, memory, IO) dedicated for each implementation.

As it is shown in table I, the QRSDET4 algorithm gives better results both in terms of speed and area. This is expected as QRSDET4 involves less comparisons to select a QRS peak. Moreover, QRSDET4, QRSDET3 and QRSDET2 use the median filter to find the moving average, which is simpler in implementation than the mean filter used in QRSDET1, as it has already been said. If quality of results is also considered, QRSDET3 can be considered the most efficient implementation. Another general observation from table I is that more advanced implementations require more resources, as expected. Also, the exact same optimization sets cannot be applied as the clock frequency is raised. This happens because CatapultC is forced to used different components with faster response time, which may not be able to be part of a very dense pipeline architecture. The optimization combinations presented in table I are only the ones that gave correct output and not all that were tested.

The implementations for the QRSDET1 detection algorithm cannot be pipelined because of the median filter which has an unknown iteration bound (at least in its implementation in [1]) and includes inter-iteration dependencies that cause problems to pipelining. From the other three detection algorithms, implementations with lower clock frequencies can be pipelined more easily because there are less opportunities for implementation resources when using high clock building blocks. However, the increased clock frequency compensates for that and the final throughput frequency can be drastically reduced in these cases also (not as much as the pipelined implementations though). Another part that creates problems in pipelining is the coding style used to initialize all internal filter registers. Loops with inter-iteration dependencies are also used there, even though the iteration bounds are known. However, in order not to change the characteristics of each algorithm no modifications to the code found in [1] were introduced. Modifications improving HLS performance may be introduced in a future version of this work.

For almost two times the experiments presented in table I (the ones that gave correct output and the ones that did not), design time was approximately one week using a Pentium 4 3GHz Linux workstation. With this effort, the best result obtained was the one in the last row of table I, operating at 400MHz and producing new outputs at 100MHz, using almost the entire FPGA device. Other interesting implementations are shown in rows 13, 17, 23, 26, 27, 30, 31, 32, 35, 36, 39, 40, 43 and 44. All other implementations are by no means useless however. Even an implementation with 100KHz throughput frequency is more than enough for the ECG signal which may reach at most 300Hz and its sampling rate most of the times may be a little more higher than 1KHz. However, in an embedded SoC environment burst mode computations may be required at specific time intervals. Then, more advanced implementations may be selected. Another advantage of low speed implementations is that they use a small portion of the FPGA device, leaving room for other functionality to be implemented on the same fabric.

For another view of the experimental results, figure 1 shows the throughput frequency (Y axis) against FPGA utilization percentage (X axis) for all implemented designs. Note that all data is sorted and there is no link to rows of table I.

Figure 1 shows that QRSDET4 gives better results but what is more interesting is the high peaks in the QRSDET4 and QRSDET3 lines. The peaks show that there exist local maximum values in throughput frequency that are not maximum in resource utilization. This is the advantage of using HLS as a synthesis technique. Such cases in the design space can be easily spotted and best implementation (with respect to user constraints) can be selected. In the previous case, if we choose the solution that occupies most of the FPGA area, expecting it to be more advanced, we lose in performance. Diagrams of this type (in fact many different styles) are presented during the design process through the GUI of CatapultC to help the designer choose the best solution.

V. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

HLS has been a design methodology and a hot research topic for the past twenty years. The same has happened to computer assisted biosignal detection and classification. Recently, there has been a universal acceptance of hardware description languages like VHDL and Verilog in design flows mainly due to the availability of efficient synthesis methods and tools that enable the translation of RTL designs into optimized gate-level implementations. On the other hand, the number of medical diagnostic applications and devices is continuously increasing with open source software implementations offered as a good starting point to further accelerate research and development. These two fields can be combined and advances in design through HLS can be put to use for the design of efficient devices, offering higher processing power and the ability to transfer larger amounts of information more quickly. The main expectations from this combination are support for better management of the design complexity and reduction of the design cycle all together, breaking the trend to compromise evaluation of various design implementation options. Designing at higher levels of abstraction allows a better coping with the system design complexity, to verify earlier in the design process and to increase code reuse.

B. Future Works

Eventhough extensive experimentation has been performed in the current work to support the combination of HLS and computer assisted biosignal manipulation applications, there

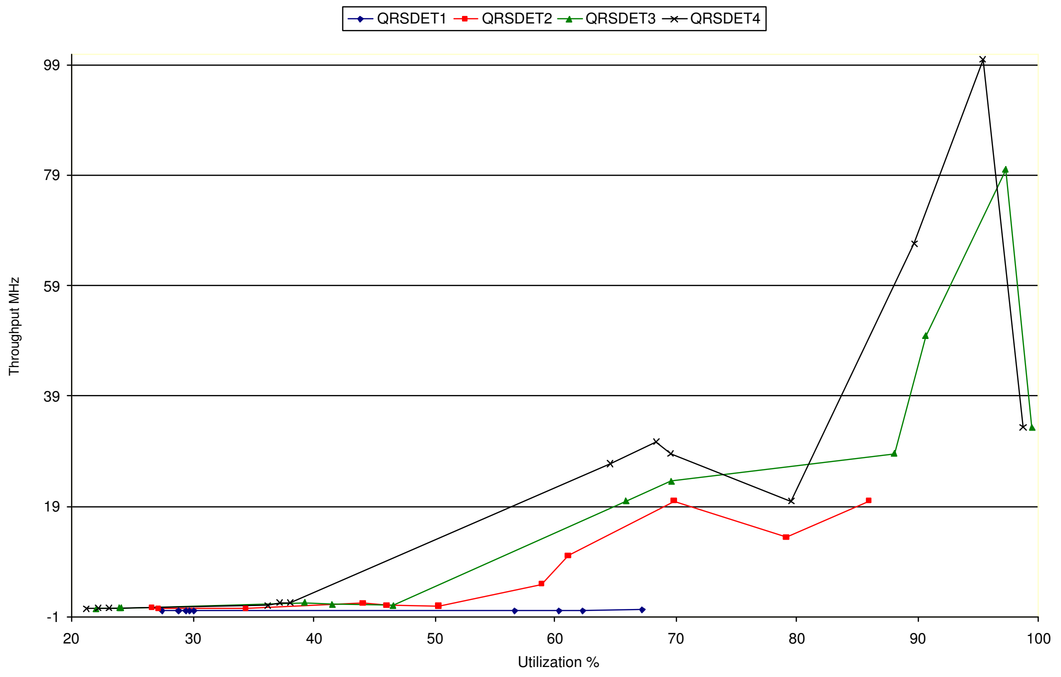

Fig. 1. QRS detection hardware designs

are still a lot that can be done. First of all, the same approach can be applied to other biosignals except the ECG or other applications for ECG, like classification, or other algorithms for QRS detection. Another step would be to install the generated hardware devices as coprocessors in an embedded SoC platform, to build more advanced systems offering advanced functionality. Finally, it would be interesting to introduce power optimization to the whole flow aiming at implantable and portable devices.

REFERENCES

[1] Open source arrhythmia detection software. `http://www.eplimited.com/software.htm`.

[2] M. Cvikl, F. Jager, and A. Zemva. Hardware implementation of a modified delay-coordinate mapping-based qrs complex detection algorithm. *EURASIP Journal on Advances in Signal Processing*, 2007(1), 2007.

[3] M. Cvikl and A. Zemva. Fpga-based system for ecg beat detection and classification. In *11th Mediterranean Conference on Medical and Biomedical Engineering and Computing*, pages 66–69. IFMBE, 2007.

[4] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.

[5] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis*. Kluwer Academic Publishers, 1992.

[6] Y. Guo, D. McCain, J. R. Cavallaro, and A. Takach. Rapid industrial prototyping and soc design of 3g/4g wireless systems using an hls methodology. *EURASIP Journal on Embedded Systems*, 2006(1):18–42, 2006.

[7] P. Hamilton. Open source ecg analysis. In *Computers in Cardiology*, pages 101–104. IEEE, 2002.

[8] P. S. Hamilton and W. J. Tompkins. Quantitative investigation of qrs detection rules using the mit/bih arrhythmia database. *IEEE Transactions on Biomedical Engineering*, 33(12):1157–1165, 1986.

[9] T.-T. Hoang, J.-P. Son, Y.-R. Kang, C.-R. Kim, H.-Y. Chung, and S.-W. Kim. A low complexity, low power, programmable qrs detector based on wavelet transform for implantable pacemaker ic. In *International SOC Conference*, pages 160–163. IEEE, 2006.

[10] K. Kuzume, K. Niijima, and S. Takano. Fpga-based lifting wavelet processor for real-time signal detection. *Signal Processing*, 84(10):1931–1940, 2004.

[11] R. G. Mark, P. S. Schluter, G. B. Moody, P. Devlin, and D. Chernoff. An annotated ecg database for evaluating arrhythmia detectors. In *4th Engineering in Medicine and Biology Society Conference*, pages 205–210. IEEE, 1982.

[12] Mentor Graphics. *Catapult Synthesis Users and Reference Manual*, 2007.

[13] M. M. Peiro, F. Ballester, G. Paya, J. Belenguer, R. Colom, and R. Gadea. Fpga custom dsp for ecg signal analysis and compression. In *International Conference on Field Programmable Logic and Applications*, pages 954–958. IEEE, 2004.

[14] W. J. Tompkins. *Biomedical Digital Signal Processing: C-Language Examples and Laboratory Experiments for the IBM PC*. Prentice Hall, 1995.

[15] Xilinx. *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*, 2007.

[16] F. Zhang, J. Tan, and Y. Lian. An effective qrs detection algorithm for wearable ecg in body area network. In *Biomedical Circuits and Systems Conference*, pages 195–198. IEEE, 2007.