ARCHITECTURAL EXPLORATION IN BIOMEDICAL HARDWARE DESIGN USING A NOVEL BEHAVIORAL SYNTHESIS METHODOLOGY

George Economakos

Microprocessors and Digital Systems Laboratory
School of Electrical and Computer Engineering
National Technical University of Athens
Heroon Polytechniou 9, GR 15780 Zografou, Greece
phone: + (30) 210 7723341, fax: + (30) 210 7722428
email: geconom@microlab.ntua.gr
web: www.microlab.ntua.gr

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, an architectural exploration of the hardware implementation of a known QRS detection algorithm is performed, following a rapid prototyping approach offered by 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.

1. 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) [5, 6]. 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. 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, 2] 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 tool that offers HLS advantages, the Bluespec synthesis tool [1]. This proposed methodology combines advantages of open source software and abstract hardware specification and synthesis for rapid prototyping 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.

The rest of the paper is organized as follows. Section 2 is a presentation of related research activities and section 3 presents Bluespec synthesis. Section 4 gives results from the conducted experiments and finally, section 5 is the conclusion and the expected future extensions.

2. RELATED RESEARCH

Electrocardiography (ECG or EKG) is the recording of the electrical activity of the heart over time, via skin electrodes. An ECG displays the voltage between pairs of these electrodes, and the muscle activity that they measure, from different directions. This display indicates the overall rhythm of the heart and weaknesses in different parts of the heart muscle. A typical ECG tracing of a normal heartbeat (or cardiac cycle) consists of a P wave, a QRS complex and a T wave. The duration, amplitude, and morphology of the QRS complex is useful in diagnosing cardiac arrhythmias, conduction abnormalities, ventricular hypertrophy, myocardial infarction, electrolyte derangements, and other disease states. Diagnosis can been supported by computer analysis of the ECG signal [8, 13]. Recently, special purpose embedded hardware devices have been proposed [3, 4, 9, 10, 12, 15], following 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 [15] mathematical morphological filtering is put to use while in [3, 4] 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 [3] a working frequency of 82MHz is reported while in [10] the frequency reported is 73MHz. In [4] the device presented in [3] 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 [12] 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 [12] 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, 3, 4], 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.

3. BLUESPEC SYNTHESIS

Bluespec synthesis is a powerful industrial level environment supporting iterative design refinements based on abstraction and parameterization mechanisms. These are available in Bluespec System Verilog (BSV), the proprietary language used for input specification. While BSV is proprietary, it is not completely strange to hardware designers since it is based on System Verilog, which is an extension to Verilog. Since Verilog's syntax resembles that of ANSI C, BSV looks familiar to software designers also. What BSV adds to System Verilog is the idea of atomic transactions for hardware modeling. The behavior of a system is described as parallel blocks of statements, called rules, that are executed as atomic units (either all statements of a rule are executed or none is executed). Each rule is executed under a specific condition that when holds, the rule is said to fire.

BSV offers a number of modeling and design advantages. First, BSV allows one to abstract out the concept of a 'functional component' as a reusable building block. Then, separately, one can express how to compose these functional components into microarchitectures, such as combinational, pipelined, iterative, or concurrent structures. For example, a function of 'ActionValue' type in BSV expresses piece of sequential behavior. A function of type 'Rule' expresses a complete piece of reactive behavior, in fact a complete reactive atomic transaction. All these components are "first class" data types, so one can build "collections" such as lists and vectors of ActionValues, Rules, and so on. Second, BSV has some powerful 'generate' mechanisms that allow the composition of microarchitectures flexibly and succinctly. For example, the microarchitectural structure can be expressed using conditionals, loops, and even recursion. These can manipulate lists of rules, interfaces, modules, ActionValues, and so on. Third, BSV has very powerful parameterization. One can write a single piece of parameterized code that, based on the choice of parameters, results in different microarchitectures (such as pipelined vs. concurrent vs. iterative, or varying a pipeline pitch, or using an alternative modules, and so on.).

Most importantly, the feature of BSV that makes all this flexibility feasible is that BSV is based on synthesis from atomic transactions. Each change in microarchitecture using above capabilities needs a corresponding change in the control logic. For example, if two functional components are composed in a pipelined or concurrent fashion, then they may conflict on access to some shared resource, whereas when composed iteratively, they may not. Each one of these cases require different control logics. When designing with RTL, it is simply too tedious and error-prone to even contemplate such changes and to redesign all this control logic from scratch. Because of BSV's synthesis from atomic semantics, this control logic is resynthesized automaticallythe designer does not have to think about it.

As an example, consider the case of a classical QRS detection algorithm [8], which applies low-pass filtering, high-pass filtering, derivation and averaging to the input signal in order to isolate QRS complexes. An open source software implementation of this algorithm [2] has been used as a case study for all experiments found in the following section. Looking at the code found in [2], detection is performed in the following fragment:

```
fdatum = lpfilt( datum, 0 ) ;
fdatum = hpfilt( fdatum, 0 ) ;
fdatum = deriv2( fdatum, 0 ) ;
fdatum = abs(fdatum) ;
fdatum = mvwint( fdatum, 0 ) ;
```

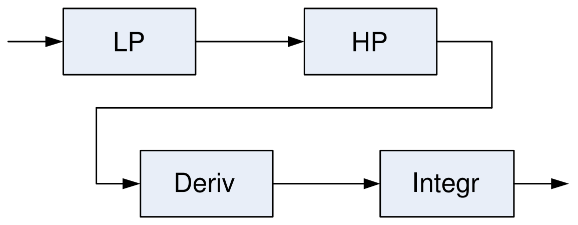

Figure 1: QRS detection filter cascade

Each code line is the application of a different filter to the new input datum. Every filter has an internal buffer that keeps enough previous values to perform the required computation, and produce a new output. This output is passed as input to the next filter, through fdatum. The way this code is written forces each input to pass through all filters serially, as shown in figure 1. This behavior can be written in BSV with the following code fragment (only the low-pass and the high-pass filters are shown for simplicity):

```
rule update_step1;
  lpFilt.read(new_data);
endrule

rule update_step2;
  hpFilt.read(lpFilt.write());
endrule

rule update_step3;
  new_result<=hpFilt.write();
endrule
```

In the above code fragment, as said earlier, each rule is a concurrent statement and all statements withing a rule are executed in parallel. Rule update_step1 gives new input to the low-pass filter, rule update_rule2 gives the output of the low-pass filter as input to the high-pass filter and rule update_rule3 takes the output of the high-pass filter. Since both filters need some time to compute their outputs, the commands in the 3 rules cannot be executed in parallel due to data dependencies. Bluespec synthesis examines the dependencies and schedules the rules to be executed serially, from top to bottom. Then, it generates the corresponding controller. The storage elements and the computations of each filter can be written as below, for the case of the high-pass filter:

```
Vector#(`HpBufferLength, Reg#(Int#(32)))
            data <- replicateM(mkRegU);
Reg#(Int#(32)) y <- mkRegU();
Wire#(Int#(32)) new_data <- mkWire();
Wire#(Int#(32)) new_result <- mkWire();

rule update;
  Int#(32) new_output;
  new_output=y + new_data -
            data[`HpBufferLength - 1];
  y <= new_output;
  new_result <= data[`HpBufferLength/2]
                        - new_output;
  writeVReg(data, shiftInAt0(
            readVReg(data), new_data));
endrule
```

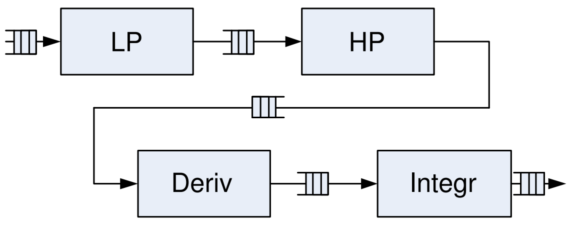

Figure 2: FIFO connected QRS detection filter cascade

However, for improved performance, FIFO queues can be inserted between filters, instead of single wires, forming a pipeline as shown in figure 2. While in the original C source code this information can not be described, BSV can describe it and Bluespec synthesis will generate a different controller.

The following code fragment shows the changes that have to be made to the high-pass filter description in order to insert FIFO queues in its input and output lines:

```
Vector#(`HpBufferLength, Reg#(Int#(32)))
            data <- replicateM(mkRegU);
Reg#(Int#(32)) y <- mkRegU();
FIFO#(Int#(32)) new_data <- mkFIFO();
FIFO#(Int#(32)) new_result <- mkFIFO();

rule update;
  Int#(32) new_output;
  new_output=y + new_data.first() -
           data[`HpBufferLength - 1];
  y <= new_output;
  new_result.enq(data[`HpBufferLength/2]
                       - new_output);
  writeVReg(data, shiftInAt0(
    readVReg(data), new_data.first()));
  new_data.deq();
endrule
```

This code changes only the type of the input and output variables (new_data and new_result) and the way data are passed into them (through the enq, deq and first method functions). All computations are kept unchanged. However, using the FIFOs enables the detection algorithm to execute all filters in parallel, leaving FIFO synchronization to take care of data dependencies, like in the code fragment below:

```
rule update;
  Int#(32) temp1,temp2;
  lpFilt.read(new_data);
  temp1 <- lpFilt.write();
  hpFilt.read(temp1);
  temp2 <- hpFilt.write();
  new_result<=temp2;
endrule
```

This last coding style offers a 10x performance gain, when passed through an RTL synthesis environment. So, BSV can be used to describe the algorithm in an abstract way but also help in architectural explorations that offer performance improvements that ANSI C can not cover. Note that BSV specific syntactic constructs like the $<-$ ActionValue assignment operator have very simple semantics but the explanation of all of them is out of the scope of this work (for details, see Bluespec reference guide [1]).

Orthogonal to the above microarchitectural considerations, BSV has other features that help the implementation of mathematical algorithms. BSV's very strong type system permits definition of abstract mathematical data types, such as fixed-point data. Widths of data fields can be specified precisely, with detailed static checking of constraints between widths of related data (for example, the width of the output of a multiplier based on the widths of the operands).

4. EXPERIMENTAL RESULTS

In order to evaluate architectural exploration methodologies and optimizations for the development of embedded diagnostic devices, we used (as briefly stated in the previous section) an open source software implementation [2] 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 [2], 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 more 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: inlining and pipelining. Inlining treats modules used by other higher level modules as inline functions, and exposes more algorithmic constructs to the synthesis 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 each input to pass through all computations (latency). When required, appropriate FIFO queues are inserted to hold internal values of the pipeline architecture. In BSV, each transformation requires different coding style.

Algorithm	Optim.	Clock Freq.	Through. Freq.	Util.
QRSDET1	None	100MHz	0.1MHz	27.45%
QRSDET1	Inline	100MHz	0.11MHz	56.60%
QRSDET1	None	200MHz	0.09MHz	28.80%
QRSDET1	Inline	200MHz	0.099MHz	29.70%
QRSDET1	None	400MHz	0.11MHz	29.44%
QRSDET1	Inline	400MHz	0.18MHz	60.39%
QRSDET2	None	100MHz	0.47MHz	34.43%
QRSDET2	Inline	100MHz	1.01MHz	50.32%
QRSDET2	Pipeline	100MHz	20MHz	85.93%
QRSDET2	None	200MHz	0.50MHz	27.15%
QRSDET2	Inline	200MHz	1.47MHz	44.10%
QRSDET2	Pipeline	200MHz	20MHz	69.89%
QRSDET2	None	400MHz	0.58MHz	26.65%
QRSDET2	Inline	400MHz	1.15MHz	46.04%
QRSDET2	Pipeline	400MHz	13.33MHz	79.17%
QRSDET3	None	100MHz	0.46MHz	22.04%
QRSDET3	Inline	100MHz	1.16MHz	46.56%
QRSDET3	Pipeline	100MHz	33.33MHz	99.46%
QRSDET3	None	200MHz	0.61MHz	23.93%
QRSDET3	Inline	200MHz	1.57MHz	39.27%
QRSDET3	Pipeline	200MHz	50MHz	90.67%
QRSDET3	None	400MHz	0.71MHz	24.06%
QRSDET3	Inline	400MHz	1.24MHz	41.49%
QRSDET3	Pipeline	400MHz	80MHz	97.32%
QRSDET4	None	100MHz	0.46MHz	21.22%
QRSDET4	Inline	100MHz	1.14MHz	36.19%
QRSDET4	Pipeline	100MHz	33.33MHz	98.73%
QRSDET4	None	200MHz	0.61MHz	23.07%
QRSDET4	Inline	200MHz	1.6MHz	38.04%
QRSDET4	Pipeline	200MHz	66.66MHz	89.71%
QRSDET4	None	400MHz	0.72MHz	22.22%
QRSDET4	Inline	400MHz	1.6MHz	37.19%
QRSDET4	Pipeline	400MHz	100MHz	95.41%

Table 1: QRS detection implementations through HLS

The summary of all conducted experiments is given in table 1. The first column of table 1 shows the implemented algorithm. The second shows the optimizations applied in each case. The third column shows the clock frequency of the device used in each experiment. For comparison reasons, this device is an FPGA of the Virtex-II Pro device family [14] 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 optimization. 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 1, 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.

The implementations for the QRSDET1 detection algorithm can not be pipelined because of the median filter which has an unknown iteration bound (at least in its implementation in [2]) and includes inter-iteration dependencies that cause problems to pipelining. From the other three detection algorithms, pipelining offers great speed improvements. Implementations with lower clock frequencies offer deeper pipelining opportunities because they utilize slower components that can be forced to operate in a deep pipeline. Implementations with higher clocks frequencies use faster components, perhaps already internally pipelined, which can not be forced to work in a deep pipeline. However, the combination of the increased clock frequency and the not-so-deep pipeline can offer drastically reduced final throughput frequencies.

For almost two times the experiments presented in table 1 (the ones that gave correct results and the ones that did not), total 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 1, operating at 400MHz and producing new outputs at 100MHz, using almost the entire FPGA device. Other interesting implementations are shown in rows 9, 12, 18, 27 and 30. 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.

5. CONCLUSIONS AND FUTURE WORK

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. 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.

Eventhough extensive experimentation has been performed in the current work to support the combination of HLS and computer assisted biosignal manipulation applications, there 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] Bluespec. `http://www.bluespec.com`.

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

[3] 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.

[4] 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.

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

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

[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] 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.

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

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

[15] 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.