SPEECH RECOGNITION SYSTEMS: FROM A CONVENTIONAL SW IMPLEMENTATION TO A RELIABLE NOISE-IMMUNITY HW-SW VERSION

F. Vargas, R. D. Fagundes, D. Barros Jr.

Electrical Engineering Dept.
Catholic University – PUCRS
Av. Ipiranga, 6681. 90619-900 Porto Alegre – Brazil
vargas@computer.org

Abstract

*Hereafter, we propose a novel approach that couples HW/SW codesign with redundancy techniques to implement speech recognition systems (SRS). Due to the specific characteristics of digital signal processing (DSP) algorithms, these systems deserve special attention when partitioning the HW and SW parts. Also, in many applications like voice-oriented bank transactions or security systems, the need for reliability is also mandatory. Therefore, the methodology we present herein partitions the HW and SW parts in such a way to boost system performance while area overhead is maintained as low as possible. At the same time, reliability in terms of redundancy (Consistency Check and Transparent BIST) is included into the SRS. Additionally, a new approach to reconstruct noisy speech signals is also presented. This approach is analogous to the hardware technique known as hot standby sparing, where the main program and its copies are all running simultaneously. In this scheme, the speech signal reconstruction is based on real time reconfiguration. Preliminary simulation results have also been performed and are presented in order to illustrate the proposal. In the next sections of the paper, we will refer to the proposed approach as "**s**peech re**co**gnition-o**r**iented HW/SW **p**art**i**tioning and fault-t**o**lerant desig**n**" approach (or simply SCORPION approach)*[1].

Keywords: HW-SW Codesign; Digital Signal Processing – DSP; Speech-Recognition Systems; Fault-Tolerance Techniques; Transparent BIST; Area overhead; Performance Degradation.

1. Introduction

It is of common agreement the large increase of the number of applications requiring digital signal processing (DSP) components (implemented either in HW or SW parts). This situation is particularly true for speech recognition-oriented applications. On the other hand, the present design methodologies do not consider the specific characteristics of the most commonly used DSP algorithms. Even representative HW-SW codesign methodologies (and tools) found in the literature do not take these particularities into account. In other words, these methodologies do not consider in the design flow the specificity of the target application, for instance: *control-dominant, data processing-oriented* (DSP applications), and *real-time constraints*. In the specific case of speech recognition systems (SRS), there are parts of the algorithm that present a *strong parallelism* profile, while others are *fully sequential*. In both cases, a high-volume (and reliable) computation is required.

If we consider the case of DSP applications like real-time robot decision-making, Internet interactive multimedia, portable telephones or aircraft on-board main computers, we can have a *large spectrum of dedicated functions* such as: voice processing and recognition algorithms; image acquisition, processing and pattern recognition; real-time data and image transmission protocols. Thus a high-throughput system architecture is mandatory since these systems are expected to be used in real-time critical applications.

Therefore, in order to handle such specific characteristics of DSP algorithms, in particular those used by SRS, we are proposing hereafter a specific design flow to optimize the *performance* and *reliability* of such systems, the *SCORPION* approach. To do so, this approach performs the HW-SW partitioning and the inclusion of fault-tolerant (FT) functions into the HW part. These functions are described in VHDL and whose parameters can be modified (adjusted) by the designer in order to satisfy application requirements [1-3].

In a second step, this approach also estimates the final reliability for the system on the design. These procedures are partially automated by a CAD tool (*FT-PRO* Methodology) [1-3] and are performed at the initial steps of the design flow [17,26]. By doing so, the main goal of this approach is to minimize time and design cost (including prototyping, testing and reliability estimation) [3-7]. Fig. 1 summarizes the main steps of the design flow carried out by the proposed approach.

The reminder of the paper is divided as follows: *Section 2* presents the basic concepts involving speech recognition systems. The control and data flows, by means of general block diagrams, are briefly introduced to readers not familiar with such type of DSP systems. *Section 3* is divided into two sub-sections in order to describe the proposed methodology: while the first part is dedicated to the HW-SW partitioning issue, the second part is involved with the incorporation of FT functions in the HW part of the SRS. *Section 4* is devoted to experimental results. A computation example is presented in this section in order to illustrate the proposed HW-SW partitioning and FT functions added to improve system performance and reliability. To conclude, *Section 5* presents the final considerations and future work.

[1] *This work is partially supported by CNPq and FAPERGS.*

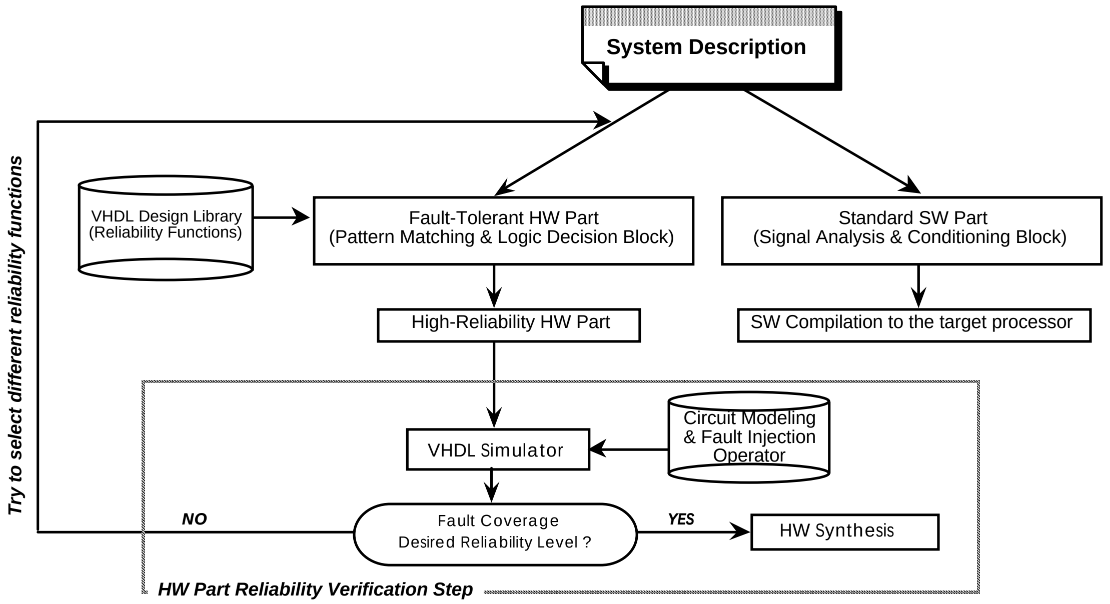

System Description
Try to select different reliability functions
VHDL Design Library (Reliability Functions)
Fault-Tolerant HW Part (Pattern Matching & Logic Decision Block)
Standard SW Part (Signal Analysis & Conditioning Block)
High-Reliability HW Part
SW Compilation to the target processor
VHDL Simulator
Circuit Modeling & Fault Injection Operator
NO
Fault Coverage Desired Reliability Level ?
YES
HW Synthesis
HW Part Reliability Verification Step

Fig. 1. Design flow of the proposed *"**s**peech re**co**gnition-o**r**iented HW/SW **p**art**i**tioning and fault-t**o**lerant desig**n**" SCORPION approach.*

2. Preliminary Considerations on the General Structure of Speech Recognition Systems

A speech recognition system (SRS) is basically a pattern recognition system dedicated to detect speech. In other words, to identify language words into a sound signal achieved as input from the environment. Fig. 2 shows the main steps performed by a front-end speech recognition system [8,9].

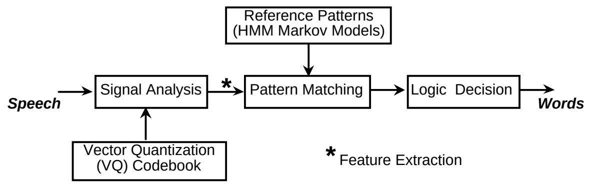


Fig. 2. General block diagram of speech recognition systems [8,9].

In the ***signal analysis*** step, a speech sampling will be made with an A/D converter. Those samples are processed in order to extract some relevant features from speech signal input. This step is responsible for signal handling, by converting the analog signal sampling, into a digital representation. The last task performed in this step is the vector quantization, when the speech signal is then replaced by a proper sequence of label-codes (this is the input for the next step of the SRS system, which is responsible for pattern matching).

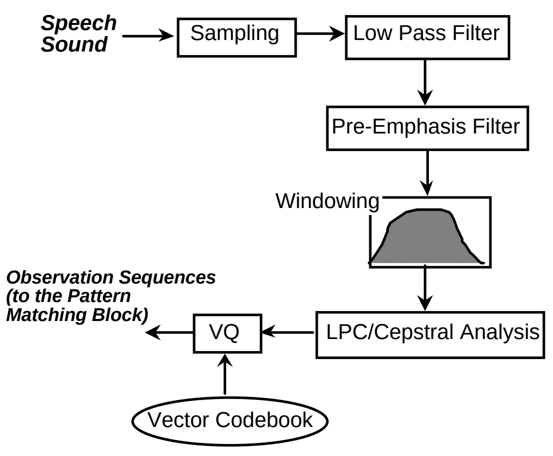
The main tasks performed in the signal analysis step are depicted in fig. 3a, and are described as follows:

- Sampling: the SRS converts speech sound from the outside world into digital representation. Essentially, this task will include a sample and hold device, and an analog-digital (A/D) converter.
- Low pass filter: cuts those high frequencies found on the signal due to sampling. Usually this filter is adjusted by sampling rate [10].
- Pre-emphasis filter: adjusts the high variations on spectrum frequencies due to glottal pulse and lips radiation found in the speech signal behavior [11,12].
- Windowing: cuts the speech signal into blocks of 10 ms signal frame each. A hamming window adjusts those frame samples [13].
- LPC/Cepstral analysis: algorithms process each frame in order to complete the cepstral coefficients from linear predictive coefficients [11,14].
- VQ – Vector quantization[1]: each vector of cepstral coefficients is evaluated by distance measure. Using, as a map, a codebook with reference vectors in the acoustic space. The final output is a sequence of label codes[2] (usually called observation sequence) that will be evaluated by the pattern matching process [10,12,15].

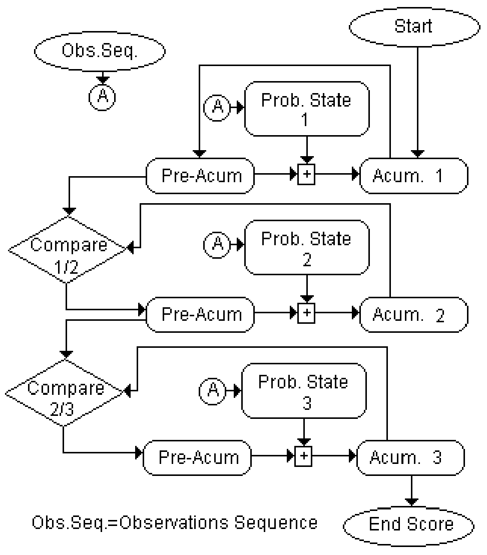
[1] *LPC: "Linear Predictive Coding ". Cepstral is a homomorphic analysis usually employed in speech processing [8,11,14].*

[2] *These labels are numbers relating the codebook reference vectors, usually called centroids, in VQ. Each label-code is just a label to those centroids. After VQ, each label-code in the sequence output is called observation.*

The *pattern matching* and the *decision logic* steps are the "identification" steps, where the words spelled in the speech signal are recognized by generating a sequence of text words. The observation sequence is evaluated using *Hidden Markov Models* (HMM), which, as the acoustic reference pattern, plays the main role in the recognition process [16-18].



(a)



(b)

Fig. 3. General block diagrams: ***(a)*** *Signal Analysis* Block; ***(b)****Pattern Matching & Logic Decision* Block [8,9].

To do so, the *pattern matching* and *decision logic* block input data (i.e., the observation sequences from the Signal Analysis Block) are actually an "index" to access the local cache memories associated with each HMM block, as seen in fig. 3b. As response to these accesses, the local memories output the respective "probability of changing state from one node to another", in the Markov Model). Then, these probability values are added to the previous values stored in the pre-accumulators of the *pattern matching* and *logic decision* block. (See fig. 3b.)

At the end of this process, the *logic decision* compares the final probability values from the HMM blocks against a reference (threshold) value and selects the one with higher score (i.e., higher probability) of being the searched word. Then, recognizing the word. Usually, the *pattern matching* and *decision logic* steps have been implemented by the Viterbi algorithm [8,12,17,25] to perform the evaluation of the code-label sequence against the HMM structures. Fig. 3b illustrates this situation for the pattern matching process [8,9].

3. *SCORPION*: The Proposed Methodology

3.1. Partitioning the SRS into HW and SW Parts

The basic idea behind the proposed methodology is to take advantage of the specific SRS dataflow, as seen in figures 2 and 3, to optimize the HW-SW partitioning step. In this case, the specific dataflow involves by one side *low-complexity high-volume computations* (represented by parallel additions followed by XOR bit-a-bit operations to perform *pattern matching* & *logic decision*), and by other side *high-complexity high-volume computations* (represented by a sequence of digital filtering operations to adjust frequency variations during the *signal analysis* and *conditioning* procedure). Also, note that there is almost no data dependency in the *pattern matching* & *logic decision* algorithms, while in the *signal analysis* one, data are strongly dependent one to each other. Thus, while the *pattern matching* and *logic decision* algorithms present an intrinsic parallel-execution profile and can be implemented by an SIMD architecture, the later algorithm (*signal analysis*) is characterized by a series-execution profile and thus, being implemented by MISD architectures.

Therefore, due to the *high-complexity* and the *sequential profile* of the functions performed by the Signal Analysis Block, this SRS part should be implemented in SW. On the other hand, the *huge parallelism profile* of the functions that implement the Pattern Matching & Logic Decision Block associated with the *high-volume* and *low-complexity computations* performed by these functions, this block is mapped to HW. Fig. 4 summarizes this behavioral analysis. The SRS main blocks shown in this figure have been partially prototyped on the Texas EVM-320TMS67xx platform (the SW part) [23] and on two FPGAs 10K20 and 7K128 Altera components (the HW part) [24]. *Preliminary experimental results can be found in [26].*

3.2. Implementing fault-tolerance in the parts of the SRS

After partitioning the speech recognition system into HW and SW parts as depicted in the previous section, the discussion hereafter is involved with the reliability requirements associated with this type of system. In this case, there are two points-of-views: the SW domain and the HW domain.

First, consider the case of f*ault-tolerance in SW:* in traditional SRS implementations (i.e., fully SW-based microprocessor implementations), it is of common agreement the use of techniques for filtering and conditioning analog (and digital) signals through the whole acquisition process: specially in the “sampling”, “low-pass filter”, pre-emphasis”, and “windowing” (see fig. 3a.). These techniques are very well known by the DSP designers community and present, in general, a very high degree of success [22]. Therefore, the assessment of fault-tolerance in SW is not really a dramatic issue, so that it is not considered in the present work.

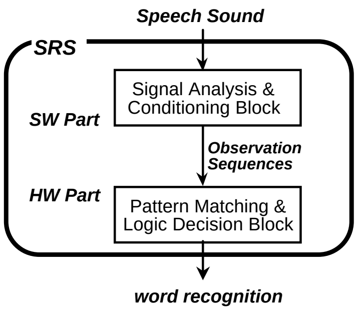

Speech Sound
SRS
Signal Analysis & Conditioning Block
SW Part
Observation Sequences
HW Part
Pattern Matching & Logic Decision Block
word recognition

Fig. 4. SRS main blocks, after partitioning into SW and HW parts.

Now, consider *fault-tolerance in the HW part:* as the main goal of this work, there are three approaches that have proposed to be incorporated into the SRS systems architecture, and whose reasons are described hereafter. Two of them are completely new proposals (“*Concurrent Consistency Check*”, and “*On-line Speech Signal Reconstruction and Checking*”), while the third one (“*Transparent BIST*”), was adapted from the literature to satisfy the specific characteristics of DSP systems, in particular those dedicated to speech recognition applications.

3.2.1. Concurrent Consistency Check (CCC)

In the following discussion, consider initially that the *pattern matching* & *logic decision* step would be implemented by traditional SW algorithms running on commercial DSP processors. In general, the datapath of these components are fit to operate with 16- or 32-bit length words. Since voice signals are commonly represented in 8- or 16-bit length words, the occurrence of overflow during the typically performed *multiply-and-accumulate* (MAC) operations is very uncommon and does not really affect substantially system operation performance.

As consequence, reliability is not really an important issue when one considers traditional SW implementations of the *pattern matching* & *logic decision* step. (Instead of this, the most important concern of designers are, in this case, time constraints, since this step must be performed in real time.)

Now, consider that the implementation of the *pattern matching* & *logic decision* step is based on a HW part. In this case, reliability becomes an important issue because in general, the datapath of this part is only 8-bit wide, in order to minimize area overhead. In other words, the occurrence of overflow during MAC operations is much more frequent when compared with its counterpart implemented in SW and running in a 32-bit commercial processor.

To have a better understanding of this problem, consider again fig. 3b. This figure shows basically the block diagram of a Hidden Markov Model (HMM) we have developed to execute the Viterbi algorithm in HW [9]. The goal of this HMM structure is to evaluate the *observation sequence* (which comes from the *signal analysis* & *conditioning* step, i. e. from the SRS SW part) and associates a given score to it. This is done by performing a sequence of MAC operations by using the adders, pre-accumulators and accumulators shown in fig. 3b. Note that this is a 3-state HMM model, since the input observation sequence is fed in parallel into 3 adders simultaneously. (For more details about the operation of such a HW implementation of the Viterbi algorithm, readers should address references [8,9].) The HMM shown in fig. 3b. is used to compute the score for only one word. Therefore, if the SRS is designed to recognize **n** words, a number of **n** HMM blocks is required. However, there is a lack of reliability in this structure due to the sequence of MAC operations performed in the narrow, low-cost datapath of 8-bit wide. *The consequence is the frequent occurrence of overflows during the MAC operations that considerably impacts the confidence of the scores generated at the end of this step. Thus, impacting the confidence of the words recognized by the SRS.*

To overcome this problem, we have proposed in this work a *concurrent consistency check* (CCC) for every MAC operation performed in HW. *Therefore, the CCC approach performs an 1-bit shift right in the contents of the HMM accumulators* (accum. 1, accum. 2, accum. 3, in fig. 3b.) *after every MAC operation*. By dividing by a factor of 2, we maintain constant the distance between the partial scores stored in the pre-accumulators in each of the states of the HMM model (pre-accum. 1, pre-accum. 2, pre-accum. 3, in fig. 3b.), at the same time that we avoid the occurrence of overflow during a large sequence of MAC operations.

3.2.2. Transparent BIST

Another reliability problem we have addressed during the HW implementation of the *pattern matching* & *logic decision* block is the confidence of the large amount of reference data stored in the memories (codebooks) associated with the HMM structure. More precisely, every time an observation sequence enters in the HMM structure, it addresses a memory array in order to select a *reference code*, which is just the Markov probability for it to change from that state to the next. Typically, one codebook is associated to each input of the HMM structure, connected at the inputs of the adders in the HMM model (“Cache Memory” blocks, in fig. 5). Typically, a codebook is an array of 128, 256 or more addresses, each address storing one byte of information. Thus, for the simple SRS shown in fig. 5 a memory system to store 1152 bytes is required (128 bytes x 3 states x 3 words). For more realistic SRSs, with 6-state HMMs and 256-address codebooks, which are also able to recognize some hundreds or thousands of words, typical memory capacities rang from 256KBytes to 25.6Mbytes. In order to protect this large space of critical memory data, we implemented the Transparent Built-In Self Test (BIST) approach [19-21] in the *pattern matching* & *logic decision* block. This choice can be explained as follows:

a) need of minimizing area overhead (this approach is one of the best choices found in the literature in terms of area overhead and types of faults detected in memory structures. Just a an example, in [20] the authors claim an area overhead of 1.2% due to the inclusion of Transparent BIST in the case of a 128Kbytes X 8Bytes memory (and this value decreases as the RAM size increases).

b) associated with low area overhead, the Transparent BIST approach presents a high capability of fault detection by indicating the occurrence of stuck-at faults, transition faults, coupling faults, decoder faults and read/write logic faults [20].

c) since SRSs are used in real time applications, there is a need for very short "down periods", that are periodically required to check the functionality of the SRS mass memory system. At this point, the Transparent BIST approach also presents the incomparable advantage of preserving the contents of the RAM memory after testing. Thus, this approach is very suitable for periodic testing since we do not need to save the RAM contents before the test session and to restore them at the end of this session.

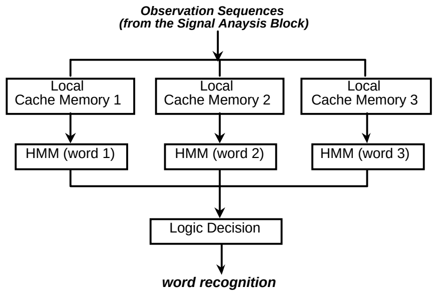


Fig. 5. Pattern Matching & Logic Decision Block implemented to recognize 3 words [9].

Table 1 presents the algorithm for Transparent BIST. In this table **Rai** means that we perform a read operation on word **i** and the read value is equal to **ai** (the initial value of word **i**). Similarly, **Rai(not)** means that we perform a read operation on word **i** and the expected value to be read is equal to **ai(not)**. **Wai** and **Wai(not)** mean that we write **ai** or **ai(not)** on cell **i**. We remark that the last values written in the RAM are equal to the initial RAM contents. Thus, the algorithm is transparent.

The sequences S1 and S2 address the RAM in some order while the sequences S3 and S4 address the memory words in the reverse order. Note also that the test output responses must be checked. This is done by adding an algorithm which allows predicting the signature of the RAM output responses. This algorithm is called "Signature Prediction Algorithm", and is given in table 2. Therefore, the data read during the execution of the sequence shown table 2 are injected into an output response compactor (MISR – Multiple Input Shift Register) [20,22] in order to generate the *predicted signature* of the memory test. After this step, the same procedure is performed during the execution of the transparent BIST itself, as shown in table 1, in such a way that all the data read from the RAM are also injected into the same MISR, so that this time the *test signature* is generated. The last step of the transparent BIST approach is the comparison of both of the signatures: the *predicted* and the one generated during the *test* itself. For a fault-free RAM, one can expects that the signatures are equal.

Note that the data read during the execution of sequences S1' through S4' of the signature prediction algorithm (table 2) are sometimes inverted in order to match the data read during the execution of sequences S1 through S4 of the transparent BIST test (table 1). For more detailed description and operation of the Transparent BIST technique, readers can address references [19-21].

S1	S2	S3	S4
Ra1 Wa1(not) Wa1 Wa1(not)	Ra1(not) Wa1 Ra1 Wa1(not)	Ra1(not) Wa1 Wa1(not) Wa1	Ra1 Wa1(not) Ra1(not) Wa1
Ra2 Wa2(not) Wa2 Wa2(not)	Ra2(not) Wa2 Ra2 Wa2(not)	Ra2(not) Wa2 Wa2(not) Wa2	Ra2 Wa2(not) Ra2(not) Wa2
...	...	...	...
...	...	...	...
Ran Wan(not) Wan Wan(not)	Ran(not) Wan Ran Wan(not)	Ran(not) Wan Wan(not) Wan	Ran Wan(not) Ran(not) Wan

→ execution time

Table 1. Algorithm for Transparent BIST.

S1	S2	S3	S4
R1	R1 R1	R1	R1 R1
R2	R2 R2	R2	R2 R2
...	...	...	...
...	...	...	...
Rn	Rn Rn	Rn	Rn Rn

→ execution time

Table 2. Signature Prediction algorithm.

3.2.3. On-line Speech Signal Reconstruction

Although convolutional codes, first introduced by Elias [25], have been applied over the past decades to increase the efficiency of numerous communication systems, where they invariably improve the quality of the received information, there remains to date a lack of reliability when such an information is used to represent speech. In other words, voice-oriented systems used in bank-transaction or security applications are frequently struggled by noise like electromagnetic interference, or just background noise. As consequence, the information (i.e., the speech) is partially (or even totally) buried by noise, thus reducing the reliability of the incoming signal.

To overcome this problem, we are proposing a new approach, namely *On-line Speech Signal Reconstruction*, whose goal is to minimize (or even eliminate) the noise associated with the voice signal, and thus allowing a more reliable speech recognition process by the system. First, this is performed by reconstructing the incoming signal when the scores of the words expected to be recognized are lower than predefined values. Second, by checking the consistency of the "reconstructed" words by recognizing them a second time (in this case, high recognition scores are expected). In some sense, this approach is analogous to the hardware technique known as *hot standby sparing*, where the main program and its copies are all running simultaneously [22]. In this scheme, the speech signal reconstruction is based on *real time reconfiguration*. Fig. 6 details the main blocks of the proposed technique.

Basically, the approach works as follows: the incoming signal is analyzed by the Viterbi Algorithm which was implemented by means of the Hidden Markov Models (HMM), exactly as described in Section 2 (fig. 3b). Note that the incoming signal (the SRS input indicated by "*" in fig. 6) is the observation sequences, i.e., digital code labels representing speech. Therefore, in the occurrence of an incoming message, the *Subsystem-I* begins the process of recognizing the speech from the arriving observation codes. Note that the SRS must be previously trained to do this work, and the number of words that can be recognized is equal to the *HMM Blocks* inside *Subsystems I* or *II*. Each one of these *HMM Blocks* is used to model a single word.

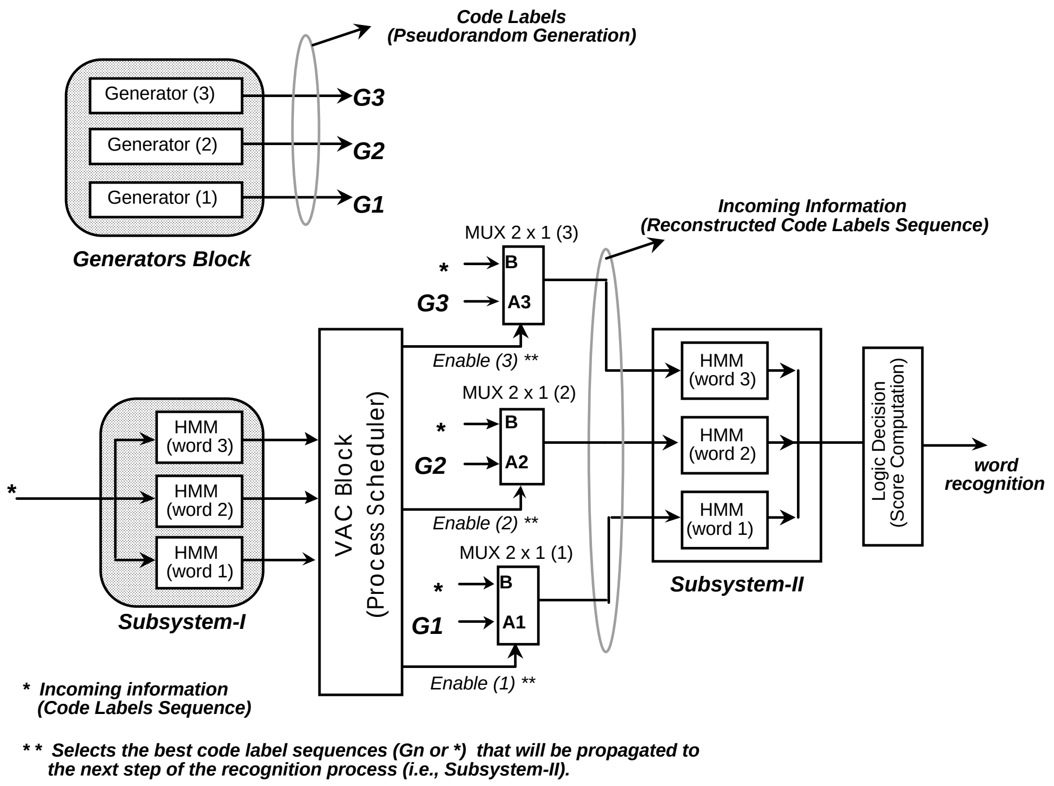

Code Labels (Pseudorandom Generation)
Generator (3)
Generator (2)
Generator (1)
G3
G2
G1
Generators Block
Incoming Information (Reconstructed Code Labels Sequence)
MUX 2 x 1 (3)
MUX 2 x 1 (2)
MUX 2 x 1 (1)
B
A3
A2
A1
Enable (3) **
Enable (2) **
Enable (1) **
VAC Block (Process Scheduler)
HMM (word 3)
HMM (word 2)
HMM (word 1)
Subsystem-I
Subsystem-II
Logic Decision (Score Computation)
word recognition
* Incoming information (Code Labels Sequence)
** Selects the best code label sequences (Gn or *) that will be propagated to the next step of the recognition process (i.e., Subsystem-II).

Fig. 6. General Scheme of the On-line Speech Signal Reconstruction Approach.

The whole approach is controlled by the *Viterbi Algorithm Controller* ("*VAC Block*", in fig. 6) which supervises the recognition process running on the *HMM Blocks* in *Subsystem-I*. While the goal of these *HMM Blocks* is to compute the HMM probability scores for each of the incoming observations, the goal of the *VAC Block* is to select the best code label sequences[1] (G_n or *****) that will be propagated to the next step of the recognition process (i.e., to *Subsystem-II*). As can be seen in fig. 6, this selection is done through MUX 2x1 control signals "*Enable (n)*".

The *Generators Subsystem* operates synchronously with the *HMM Blocks*, under a single clock signal generated by the *VAC Block*. The goal of the *Generators Subsystem* is to generate those segments of the incoming information that are buried in noise so that the information cannot be recognized by the *HMM Blocks* as "speech". To do so, the *Generators Subsystem* simulates a "random hidden Markov process[2]", and generates an observation sequence of code labels with the same probabilistic properties of the incoming signal. Thus, those new observations replace the noisy segments whenever the decreasing probabilistic scores are detected.

Once the incoming information (i.e., the reconstructed code label sequences) reaches *Subsystem-II*, the goal of the *HMM Blocks* therein is to compute, as example of the *HMM Blocks* inside *Subsystem-I*, the probability scores of changing from one state to another in the Markov chain. However, in *Subsystem-II* this probability is computed for the reconstructed signal (in *Subsystem-I*, these probabilities were computed for the original incoming signal, before the reconstruction process), which is composed partly by the original code labels (indicated by "*" in fig. 6), and partly by the pseudorandom-generated code labels (G3, G2, or G1).

Finally, the *Logic Decision Block* chooses between the three words the one with the higher score, i.e., the one with the higher probability of occurrence.

4. Experimental Results

This section presents a computation example that we have developed to illustrate the proposed approach. With this purpose, we implemented and trained[3] an SRS to recognize 2 words. This system was prototyped on a HW-SW development environment based on the *TMS-320C67 Texas DSP microprocessor* [23] and on the *FLEX10K20 FPGA Altera Compoment* [24]. The first goal was to show the area overhead resulted due to the inclusion of the transparent BIST by one side, and the shifter logic necessary to perform the *concurrent consistency check* (CCC) for every MAC operation executed in HW, by the other side. The second goal was to show the performance boost resulted from migrating into HW, part of the SW code that executes the pattern recognition and logic decision tasks in traditional SRSs. Therefore, Table 3 summarizes these numbers.

Note that we restricted the SRS implementation to 2 words due to the space limitation imposed by the FPGA component (*EPF10K20RC240-4*). This particular implementation yielded the *worst-case implementation* for the transparent BIST approach in terms of area overhead (as can be seen in table 3c: 20.99%). However, note that the area required to implement the transparent BIST is approximately constant (roughly 190 CLBs) while the increase of the number of words to be recognized by the SRS means that more cache memory must be proportionally added to the *pattern recognition & decision logic* block. Therefore, for increased-vocabulary SRSs, for instance 4, 8, 16, 32, or 64 words, one can expect the transparent BIST area overhead be reduced to some order of 11%, 5.5%, 3%, 1.5%, and 0.8%, respectively, and so on.

Note also that the increase of memory space to accommodate the additional information required by the SRS to handle larger vocabularies does not mean necessarily that the time required to test this memory will increase (in this case, approximately 0.06132 s). The reason is that the increase of memory is done in the form of adding new local cache memories to the new HMM blocks that are required to model the additional words to be recognized by the system. In this case, the transparent BIST performs a periodical test of all the local cache memories in parallel, which does not actually increase the overall time required to test the SRS memory.

Note also that even with the HW implementation including the *transparent BIST*, the runtime of the proposed HW-SW partitioning approach (0.06132 s) is roughly 10 times faster than the one of traditional SW-based microprocessor implementations (0.577 s). This condition drives us to the comfortable position to execute periodical tests of the SRS with negligible performance penalty, and completely transparent to the user.

Finally, the SRS word recognition effectiveness resulted by the use of the *concurrent consistency check (CCC)* technique is illustrated in table 3d. By comparing both of the HW implementations: the original one (without redundancy) and the one with *CCC*, the confidence was improved approximately by a factor of two. When the confidence of the HW implementation based on the CCC technique is compared with the one yielded by the traditional SW-based microprocessor implementation, the same degree of success is verified. Note that in both cases, a degree of 100% could not be reached due probably to *undesired environment noise interference* during the generation of the reference data to be stored in the *vector codebook* and in the *local cache memories* associated with

[1] *Note that it may happen the incoming signal can be partial or totally corrupted by noise. In this case, the VAC Block switches from the original incoming signal (represented by "*" in fig. 6) to the pseudorandom generated code labels during a predefined period of time, and then, bouncing back to the original incoming signal. This process is dynamically performed every time the original incoming signal is corrupted by noise (i.e., the computed probability by the HMM Blocks in Subsystem-I is lower than a predefined threshold value).*

[2] *Note that actually, the observation sequences are not completely randomly generated because the universe of label codes explored by the Generators Block to generate the observation sequence is bounded by the specific HMM probability model that was selected by the VAC Block.*

[3] *System trained using the* ***HTK software tool*** (Hidden Markov Model Tool Kit).

the HMM blocks. While the *vector* codebook stores the *observation sequence values that represent reference vectors in the acoustic space*, the cache memories store the *probability of each of these sequence values to change state from one node to another, in the Markov Model*. These data are critical for the correct operation of the SRS, and any change in these values (induced by noise, for instance) may lead to system malfunction.

	System Performance (s) **[time required to recognize a word]**	**Performance Improvement**
Traditional SW-based microprocessor implementation	0.577	---
HW-SW based partitioning approach (original HW implementation without redundancy)	0.00132	437 times

(a) SRS performance improvement due to the system partitioning according to the proposed HW-SW codesign technique.

	System Performance (s) **[time required to recognize a word]**	**Performance Degradation**
Original HW implementation (without redundancy)	0.00132	---
HW implementation including the *transparent BIST* *****	0.06132	46 times

***** *Standalone runtime for the transparent BIST through the local cache memories of the HMM blocks: 60ms.*

(b) SRS performance degradation due to the inclusion of the transparent BIST into the HW part. (The *concurrent consistency check* (CCC) is performed in parallel with the application, thus not resulting in performance penalty.)

	Area (configurable logic blocks – CLBs)	**Area Overhead (%)**
Original HW implementation (without redundancy)	905	---
HW implementation including the *concurrent consistency check (CCC)*	907	0.22
HW implementation including the *transparent BIST*	1095	20.99
HW implementation including both *CCC* and *transparent BIST*	1097	21.21

(c) Area overhead required by the different implementation forms of the fault tolerant *Pattern Matching & Logic Diagram* Block.

	System Confidence **[frequency of which words are recognized correctly]**
Traditional SW-based microprocessor implementation	approx. 90%
HW-SW based partitioning approach (original HW implementation without redundancy)	approx. 40%
HW implementation including the *concurrent consistency check (CCC)*	approx. 90%

(d) SRS reliability (word recognition confidence) due to the inclusion of the *concurrent consistency check (CCC)* technique after each MAC operation in the *pattern recognition & logic decision* block.

Table 3. Summary of the SRS implemented to recognize 2 words.

Similarly to Table 3,where the SRS implemented partly in HW (FPGA) and partly in SW (DSP processor) was trained by using the *HTK software tool*, Table 4 shows simulation results for another SRS also trained with the same tool, but this time only simulated in Matlab environment. With this purpose, we have considered 3 words to be recognized by the system, each of them represented by a group of 30 10ms-duration code labels. In this case, the incoming information indicated by "*****" in fig. 6 is represented by a sequence of 90 code labels (3 words x 30 code labels x 10ms = 900ms overall speech time duration). The basic idea behind Table 4 is to verify the improvement of the SRS confidence resulted from using the on-line speech signal reconstruction technique described in Section 3.

After analyzing this table, we can take the following conclusions:

1) The proposed technique is very effective for noise corruption levels hanging from 50 to 80 percent of the total incoming information.

2) Values above 80% must be discarded, since due to the high values of noise the SRS (with or without the proposed approach) is expected to "randomly" recognize the correct word.

3) For values below 50%, the SRS degree of success is quite similar to the one obtained without the proposed technique. This is explained due to the fact that the *VAC Block* rarely selects the pseudorandom-generated code labels from the *Generators Subsystem* (G_n , in fig. 6), since the original segments of the incoming information that are buried in noise can still be correctly recognized by the *HMM Blocks* in Subsystem-I as "speech". As consequence, the original information indicated by "*****" in fig. 6 is selected during the most part of the time by the *VAC Block* (through MUX 2x1 components) and propagated to the next stage (*Subsystem-II*). This results that the incoming information *before* reconstruction is quite similar to the one *after* reconstruction. In this case, the SRS is expected to present approximately the same degree of success in terms of recognizing the correct word.

Percentage of the incoming information corrupted by noise * (%)	**System confidence (%) [frequency of which words are recognized correctly]**	
	Before	**After**
90	33.33	23.33
80	50.00	63.33
70	73.33	76.67
60	90.00	92.00
From 0 to 50	98.00	98.00

******* *Sequence size: 30 code labels per word. Each code label represents 10ms of continuous speech sound. Three words were considered in this simulation. (This sequence of code labels is represented by the incoming information indicated by "*" in* ***fig. 6****.)*

Table 4. Preliminary results showing the SRS performance in terms of recognizing the correct word **before** and **after** using the on-line speech signal reconstruction technique. (Simulation results obtained for a software implementation of the on-line speech signal reconstruction technique running in a Matlab environment.)

5. Final Discussions & Future Work

In the previous sections, we have proposed a novel approach (namely, "**s**peech re**co**gnition-o**r**iented HW/SW **p**art**i**tioning and fault-t**o**lerant desig**n**" - *SCORPION*) that couples HW/SW codesign with redundancy techniques to implement speech recognition systems (SRS).

Due to the specific characteristics of digital signal processing (DSP) algorithms, these systems deserve special attention when partitioning the HW and SW parts. Also, in many applications like voice-oriented bank transactions or security systems, the need for reliability is also mandatory. Therefore, the methodology we presented drives the partitioning of the system in HW and SW parts in such a way to boost system performance while maintaining area overhead as low as possible. At the same time, reliability in terms of redundancy (*Consistency Check* and *Transparent BIST*) was also considered during the implementation of SRSs. Additionally, a new approach to reconstruct noisy speech signals was also presented. This approach is analogous to the hardware technique known as *hot standby sparing*, where the main program and its copies are all running simultaneously. In this scheme, the speech signal reconstruction is based on real time reconfiguration.

With the purpose of estimating *area overhead*, *reliability improvement*, and *speed degradation* due to: ***(a)*** the incorporation of fault-tolerant techniques and ***(b)*** the use of specific system partitioning, we conducted a first laboratory experiment, where we specified an SRS to recognize 2 words. It was partitioned according to the proposed methodology. The SW part was compiled to the target processor (*Texas DSP TMS320C67*), while the HW part was implemented in an Altera FPGA (*EPF10K20RC240-4*).

The obtained preliminary results indicate that the area overhead due to the inclusion of the *transparent BIST* and the *concurrent consistency check (CCC)* is negligible for systems that recognize a vocabulary of 32 words or more. In addition, performance degradation due to the periodical execution of the transparent BIST approach was estimated to be on the order of 46 times more than the time required to recognize a single word. However, this condition does not affect the real-time speech recognition characteristic of the DSP system because it is roughly 10 times faster than traditional SW-based microprocessor implementations. This condition drives us to the comfortable position to execute periodical tests of the SRS with negligible performance penalty, and completely transparent to the user. Additionally, the *CCC* technique has also shown to be extremely affective in avoiding overflow conditions during MAC operations based on 8-bit length hardware.

In a second laboratory experiment, we have simulated in Matlab environment an SRS to recognize 3 words. This simulation aimed at verifying the effectiveness of the on-line speech signal reconstruction technique we have proposed. For this 3-word SRS, the obtained results have shown that the proposed technique is very effective to reconstruct speech signals when noise affects between 50 and 80 percent of the overall signal.

In the first laboratory experiment we conducted, the SRS implemented was designed to recognize 2 words, with a *vector codebook* storing 66 observation sequence values per word. This system specification is sufficient to yield good recognition responses (around 90%, Table 1d). For *future work* we will double this parameter to at least 132 observation sequence values per word to be stored in the *vector codebook* structure. By doing so, we intend to increase the SRS degree of success to a value higher than 90% in terms of recognizing the correct words in a larger vocabulary. However, for vocabularies containing much more than 2 words the storage area represented by the *vector codebook* becomes an important issue. Consequently, as example of the transparent BIST that is applied to the local cache memories associated with the *pattern matching & logic decision* block, this test approach could also be used to protect the vector codebook memory structure as well. This situation deserves a deeper analysis. The next studies must evaluate in more detail the trade-off between the system *performance degradation* versus *reliability* while preserving the *real time response requirement* of the DSP system applied for larger vocabularies.

References

[1] VARGAS, F.; BEZERRA, E.; TERROSO, A. Testability Verification of Embedded Systems Based on Weak Mutation Analysis. 3rd IEEE International Workshop on Testing Embedded Core-Based System-Chips -TECS'99. Dana Point - CA, USA, Apr. 28-29,1999. pp. 31-37.

[2] VARGAS, F.; AMORY A. Transient-Fault Tolerant VHDL Descriptions: A Case-Study for Area Overhead Analysis. 9th IEEE Asian Test Symposium - ATS'00. Taipei, Taiwan, Dec. 04-06, 2000.

[3] VARGAS, F.; AMORY A.; VELAZCO, R. Estimating Circuit Fault-Tolerance by Means of Transient-Fault Injection in VHDL. 6th IEEE International On-Line Testing Workshop. Mallorca, Spain, Jul. 3-5, 2000.

[4] ARLAT, J.; AGUERA, M.; AMAT, L.; CROUZET, Y.; FABRE, J.-C.; LAPRIE, J.-C. MARTINS, E.; POWELL, D. Fault Injection for Dependability Validation: a Methodology and Some Applications. IEEE Transactions on Software Engineering, Vol. 16, No. 2, Feb. 1990, pp. 166-182.

[5] KARISSON, J.; LIDEN, P.; DAHLGREN, P.; JOHANSSON, R.; GUNNEFIO, U. Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms. IEEE Micro, Vol. 14, No. 1, 1994, p. 8-23.

[6] KANAWATI, G. A.; KANAWATI, N. A.; ABRAHAM, J. A. FERRARI: A Flexible Software-Based Fault and Error Injection System. IEEE Transactions on Computers Vol. 44, No. 2, February 1995, pp. 148-160.

[7] SIEH, V.; TSCHÄCHE, O.; BALBACH, B. VERIFY: Evaluation of Reliability Using VHDL-Models with Embedded Fault Descriptions. 27th International Symposium on Fault-Tolerant Computing (FTCS '97).

[8] RABINER, L. R., JUANG, B. H. Fundamentals of Speech Recognition. New Jersey , Prentice Hall, 1993.

[9] FAGUNDES, R. D. R.; VARGAS, F.; BARROS Jr., D. A Viterbi Algorithm Implementation Using Hardware/Software Co-Design in Speech Recognition Systems. The International Association of Science and Technology for Development Conference – IASTED'2000. Marbella, Spain, Sep. 19-22, 2000. (www.iasted.com)

[10] DELLER, J., PROAKIS, J.G., HANSEN, J. H. L. Discrete-Time Processing of Speech Signals. New York: Macmillan, 1993.

[11] GRAY Jr., A.H.; MARKEL, J. D. Linear prediction of speech. Communication and Cybernetics, 3 ed., Berlin. Springer, 1982.

[12] RABINER, L. R.; LEVINSON, S. E. A speaker-independent, syntax-directed, connected word recognition system based on hidden markov models and level building. IEEE Transactions on Acoustics, Speech, and Signal Processing, v.33, n.3, p.561-73, June 1985.

[13] O'SHAUGHNESSY, D. Speech Communication Human and Machine. Massachusetts: Addison-Wesley, 1987.

[14] MAKHOUL, J. Linear prediction: a tutorial review. Proceedings of the IEEE. v.63, n.4, p. 561-580, Apr 1975.

[15] LINDE, Y.; BUZO, A.; GRAY, R.M. An algorithm for vector quantizer design. IEEE Transactions on Communications, v.28, n.1, p.84-95, Jan. 1980.

[16] RABINER, L. R.; WILPON, J. G.; SOONG, F. K. High performance connected digit recognition using Hidden Markov Models. IEEE Transactions on Acoustics, Speech, and Signal Processing, v.37, n.8, p.1214-1225, Aug. 1989.

[17] VARGAS, F.; FAGUNDES, R. D. R.; BARROS Jr., D. Orienting Redundancy and HW/SW Codesign Techniques Towards Speech Recognition Systems. 2nd IEEE Latin American Test Workshop - LATW2001. Cancun, Mexico, Feb. 11-14, 2001. pp. 226-233.

[18] FAGUNDES, R. D. R. Abordagem Fonético-Fonológica em Sistemas de Reconhecimento de Voz de Linguagem Contínua. São Paulo, 1998. Tese de Doutorado - Escola Politécnica, Universidade de São Paulo.

[19] MARINESCU, M. Simple and Efficient Algorithms for Functional RAM Testing. IEEE International Test Conference – ITC, 1982.

[20] KEBICHI, O.; NICOLAIDIS, M. A Tool for Automatic Generation of BISTed and Transparent BISTed RAMs. IEEE International Test Conference – ITC, 1992. pp. 570-575.

[21] NICOLAIDIS. M. Transparent BIST for RAMs. TIM3 Report, Apr. 1992.

[22] PRADHAN, D. K. Fault-Tolerant Computer System Design. Prentice-Hall, 1996. 544p.

[23] http://www.ti.com

[24] FLEX 1OK Literature (internet source), http://www.altera.com/html/literature/lf10k.html.

[25] VITERBI, A. J. Convolutional Codes and Their Performance in Communication Systems. IEEE Transactions on Communications Tech., Vol. Com-19, No. 5, Oct. 1971, pp.751-772.

[26] VARGAS, F.; FAGUNDES, R. D. R.; BARROS Jr., D. 26th IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP2001. Salt Lake City, Utah – USA, 2001. (www.icassp2001.org).