Toward the Software Realization of a GSM Base Station

Thierry Turletti, *Member, IEEE*, Hans J. Bentzen, and David Tennenhouse, *Member, IEEE*

***Abstract*—Recent advances in processor and analog-to-digital conversion technology have made the software approach an increasingly attractive alternative for implementing radio-based systems. For mobile telephony base stations, the advantages with the new architecture are obvious: great cost savings by using one transceiver per base transceiver station (BTS) instead of one per channel, tremendous flexibility by moving system-specific parameters to the digital part, and allowing the support of a wide range of modulation and coding schemes.**

This paper considers the software implementation of a GSM BTS, and analyzes the performance of each of its radio interface modules. The performance of each software module is evaluated using both a % CPU metric and a processor-independent metric based on SPEC benchmarks. The results can be used to dimension systems, e.g., to estimate the number of software-based GSM channels that can be supported by a given processor configuration, and to predict the impact of future processor enhancements on BTS capacity. Two novel aspects of this work are the portability of the software modules and the platform-independent evaluation of their computational requirements.

***Index Terms*— Base station, digital signal processing, GSM, software complexity, software radio, SPEC.**

I. Introduction

HISTORICALLY, the use of software within radio systems has been restricted to low-bandwidth *out-of-band* tasks, such as signaling, operations, and management. Implementation of the *in-band* radio interface functionality has leveraged analog front ends that partition the band into individual FDMA channels, and digital components that perform the time-division and speech-processing functions. The design of the digital portion is typically based on a compromise between a full application-specific integrated circuit (ASIC) implementation and programmable digital signal processors (DSP's). In particular, past efforts to develop *programmable radios* [9], [11], [21] use some general-purpose processor for embedded control and internetworking functions, but leave the computationally intensive radio functions part to ASIC, field programmable gate array (FPGA), or DSP devices.

In contrast, the *virtual radio* approach [18], [20] involves the reexamination of the overall system from a software designer's perspective, with specific emphasis on the *virtualization* of RF sources. Advances in processor [16] and analog-to-digital conversion technology [7], combined with recent improvements in memory and I/O bandwidth, have enabled a complete software solution. In particular, wide-band analog-to-digital converters (ADC's) can now be used to sample an entire RF band, and the resultant sample stream can be directly deposited into memory for software-based analysis. This eliminates the need for dedicated per-channel frequency partitioning hardware. Furthermore, the temporal decoupling of the sample stream, using large memory buffers, allows conventional operating systems and networks to bridge the gap between the ADC and the software processing modules. Accordingly, both the frequency partitioning and the digital processing (previously performed by ASIC's and DSP's) can be implemented in portable software modules that execute on general-purpose processing platforms. This approach is being pioneered by the SpectrumWare[1] project, whose aim is to extend the reach of application software as close as possible to the analog-to-digital conversion boundary [17], [18].

This paper focuses on the design and performance of a library of software modules that can be used to implement the *in-band* portions of a GSM BTS. GSM, the global system for mobile communications, is a digital cellular telephony system that has gained worldwide acceptance.[2] Besides its widespread adoption, GSM is an ideal experimental vehicle because its complexity is high enough to generate a variety of interesting problems; it is a real system whose implementations are worthy opponents, i.e., they use state-of-the-art analog and DSP technology; and its design is well documented.

The structure of the paper is as follows. Section II briefly describes the hardware and software environment in which we are developing the software BTS. Sections III and IV describe the implementation of the sending side and the receiving side of the BTS, respectively, and evaluate the computational requirements of each of the individual modules. In Section V, we discuss the overall computational requirements of a complete BTS and a potential system configuration for its realization.

Except where specifically noted in the text, little time has been spent optimizing the code of the individual modules, so it is likely that the reported performance can be improved. On this basis, within three years, it will be possible to implement a *minicell* that supports 7–15 speech channels, i.e., one–two FDMA channels, using a commercially available multiprocessor server or a small cluster of workstations.

Manuscript received November 13, 1977; revised April 15, 1998 and August 23, 1998. This work was supported by DARPA under Contract F30602-92-C-0019 (monitored by AFSC, Rome Laboratory) and Contract DABT-6395-C-0060 (monitored by U.S. Army, Fort Huachuca).

T. Turletti is with the High Speed Networking RODEO Group, INRIA, BP 93, 06902 Sophia Antipolis, France.

H. J. Bentzen is with McKinsey and Co., Zurich, Switzerland.

D. Tennenhouse is with the Software Devices and Systems (SDS) Group, LCS, Massachusetts Institute of Technology, Cambridge, MA 02139 USA.

Publisher Item Identifier S 0733-8716(99)02975-3.

[1] See URL http://www.sds.lcs.mit.edu/SpectrumWare/.

[2] By the end of 1997, GSM networks accounted for more than 66 million customers of the world (110 countries/areas)—equivalent to 31% of the world's wireless market; see URL www.gsmworld.com/.

Channel partitioning, equalization, and demodulation modules are those most in need of further development and optimization.

Two novel aspects of this work are the portability of the software modules and the platform-independent evaluation of their computational requirements. For each module, one must first determine the fraction of CPU it requires when executing[3] on a range of workstation platforms. One must then normalize these individual results with respect to the known performance of the workstations, using the industry-standard SPEC[4] benchmarks. This gives an estimate of the module's computational requirements in SPECmarks. Although this is far from a precise characterization, it has the benefit of being platform independent. Furthermore, it allows one to predict module performance on future platforms, based on anticipated improvements in the SPECmark ratings of next-generation workstations and servers.

In summary, the software realization of a GSM BTS is becoming a viable alternative to present designs. A software-only approach has the potential of offering many benefits: a software BTS can support several modulation schemes and otherwise incompatible standards (e.g., AMPS, NA-TDMA, CDMA, GSM). Furthermore, the elimination of dedicated hardware reduces the complexity of the BTS, and introduces tremendous flexibility and platform independence into the overall system.

II. System Overview

This section gives a brief description of the minimal hardware required to support a software BTS, and describes the software environment for development of the GSM modules.

A. Simple Wide-Band RF Transceivers

Wide-band ADC technology allows the construction of simple RF transceivers, consisting of only an antenna, a preamplifier, a bandpass filter, and a wide-band ADC in the receive path; and a digital-to-analog converter (DAC) and an amplifier in the transmit path. Today, state-of-the-art ADC's and DAC's are about to comply with GSM performance requirements [6]—the critical issue is the high dynamic range in both receiver and transmitter, and especially the multicarrier power amplifier.

The sampled signal data are directly transferred into the memory of the workstation via a direct memory access (DMA) interface, such as the general-purpose PCI interface (GuPPI) developed at M.I.T. This PCI expansion board provides an efficient means for the continuous streaming of sample data to/from the host memory [2].

B. The Software Environment

The prototype software is based on the VuSystem, a programming system for the dynamic manipulation of temporally sensitive data [10]. This programming system runs on a standard Unix platform not specifically designed for the manipulation of digital media, such as audio and video. The VuSystem implements a two-level strategy, in which programs are partitioned along an *in-band* axis, that supports the flow of temporally sensitive information, and an *out-of-band* axis that supports the event-driven program components, including the user interface and the configuration and control of the in-band processing pipeline (see Fig. 1). In-band processing and out-of-band processing are best handled in separate partitions instead of together because choices of language and architecture can then be made for each partition separately (e.g., C++ for in-band and Tcl scripting language [15] for out-of-band processing). The in-band processing partition is arranged into processing *modules* which logically pass dynamically typed data *payloads* through input and output ports. The payload abstraction is an efficient way to hide the implementation details such as shared-memory data regions, so that the designer of in-band processing modules needs only to know a few simple rules about payload handling.

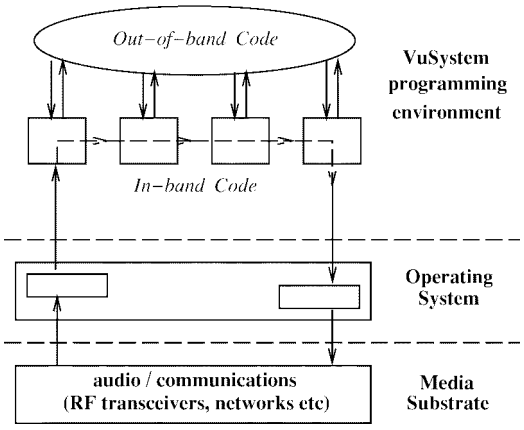

Fig. 1. VuSystem programming environment.

The following sections describe the software implementation of the in-band[5] portion of a GSM BTS; for further detail concerning GSM, see [12]. The in-band modules may be classified into two classes: the downlink modules that correspond to the sending side of the BTS, and the uplink modules corresponding to functions performed in the reverse path. For each module, the section briefly reviews the functionality provided, the algorithm chosen, and the performance obtained.

C. Estimation of Computational Requirements

The functionality and performance of the downlink and uplink modules will be discussed in detail in Sections III and IV, respectively. The following describes the methodology used to measure the performance of the individual modules.

To measure performance, each module is isolated and runs separately in a loop, processing $T = 20$ ms of audio chunk each time through the loop.[6] Consider the speech coder module in detail. To estimate the module's execution time, measure

[3] See details in Section II-C.

[4] SPEC stands for system performance evaluation cooperative; see URL www.specbench.org/spec/.

[5] Currently, we have not implemented any signaling functions.

[6] In GSM, speech is transmitted using groups of 260 bits every 20 ms; see Section III-A.

Fig. 2. Block diagram of a GSM sender BTS (downlink).

the time needed to process N chunks of audio data on each platform, allowing the module to sequentially process these N chunks of data as fast as possible (i.e., without pausing between processing the individual chunks of data). Use the *c*-shell *time* function to measure the overall amount of time needed.[7] Dividing this execution time by NT (N times the required periodicity of the process) gives the fraction of the CPU that the module consumes when performing the processing associated with a single GSM channel. In order to get an accurate measure of time, the N value is chosen high enough[8] to get at least 10 s of processing time on the fastest platform.

To measure performance of the channel coder module, run the speech coder module once to get the 256 input bits, and then run the channel coder module N times over these data. The execution time of the speech coder module is subtracted from the overall execution time measured to give the overall channel coder execution time. Then, the ratio of N divided by T yields the corresponding fraction of CPU used by this module, etc.

The resultant "% CPU/GSM channel" is a good metric to report the performance of each module on a given platform. However, these results will rapidly become outdated as new generations of processors appear. Quantification of computational requirements in a *processor-independent* way is critical for a vision of plug-and-play telecommunications. Accordingly, the SPEC benchmark is adopted as a vehicle for the normalization of our results. SPEC benchmarks have been designed to provide a comparable measure of performance of a system executing a known compute-intensive workload. The benchmarks SPEC92 are included because they still have high popularity in the computer community. Since 1995, SPEC is replacing SPEC92 with an improved SPEC95, with the subcomponents CINT95 (focusing on integer/nonfloating-point compute-intensive activity) and CFP95 (focusing on floating-point compute-intensive activity). The benchmark of the software modules normalize the % CPU results with respect to the SPEC92 and SPEC95 ratings of each platform, as presented in Tables XIV and XV, respectively. This gives an estimate of the module's computational requirements in SPECmarks. Although the mapping to a module's SPECmark requirement is not entirely platform independent, the SPECmark range can provide a basis for performance predictions. This generic metric is not precise, but it is nonetheless an improvement on platform-dependent measures of computational requirements.

III. From Speech to Radio (Downlink)

The sequence of downlink modules is shown in Fig. 2. Basically, the GSM speech coder compresses speech into digital blocks. Channel coding adds redundancy to the blocks, which are then interleaved and spread into pieces called bursts. Finally, after the ciphering operation, the bursts are used to modulate the phase of a carrier.

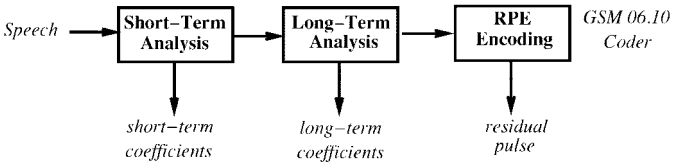

Fig. 3. Block diagram of the GSM speech coder.

TABLE I
GSM Speech Coding Performance

Speech Coder (RPE-LTP 06.10)	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	8.1	20	0.45	28	0.73
DEC 3000/800	10	14	–	19	–
Pentium 166	10	20	0.48	–	0.34
Pentium Pro 200	5.9	19	0.48	17	0.40

A. GSM Speech Coding

The GSM full-rate speech coding algorithm is called RPE–LTP, which stands for regular pulse excitation with long-term prediction [5]. The analog voice is first digitized in an ADC to 8000 samples/s, uniformly coded at 13 bits each. Then, the encoder divides the speech into a short-term predictable part, a long-term predictable part, and the remaining residual pulse (see Fig. 3). Finally, it encodes the residual pulse and parameters describing the two predictors. The speech coding algorithm produces a speech block of 260 bits every 20 ms. These 260 bits are classified into three classes (Ia, Ib, and II) according to their importance.

The software implementation of this vocoder was implemented by Degener and Bormann from the Technical University of Berlin.[9] The performance[10] of this vocoder on a number of platforms is shown in Table I.

B. Channel Coding

Class Ia bits are first protected by three CRC bits for error detection. The Class Ib bits are then added to this result. A convolutional code with rate $r = 1/2$ and constraint length $K = 5$ is then applied to this complete class I sequence. The resulting 378 bits are used in conjunction with the 78 unprotected Class II bits, to form a complete coded speech frame of 456 bits (see Fig. 4).

1) Error-Detecting Codes: As indicated in Fig. 4, the most sensitive portion of the speech frame (i.e., the 50 bit category

[7] The *c*-shell *time* function ensures that the application uses the whole CPU.

[8] For example, we chose $N = 10^4$ for the speech coder module measurement.

[9] The implementation is available in the URL www.cs.tu-berlin.de/~jutta/toast.html.

[10] All C and C++ modules are compiled with gcc v2.7.2 using the $-O2$ optimization flag.

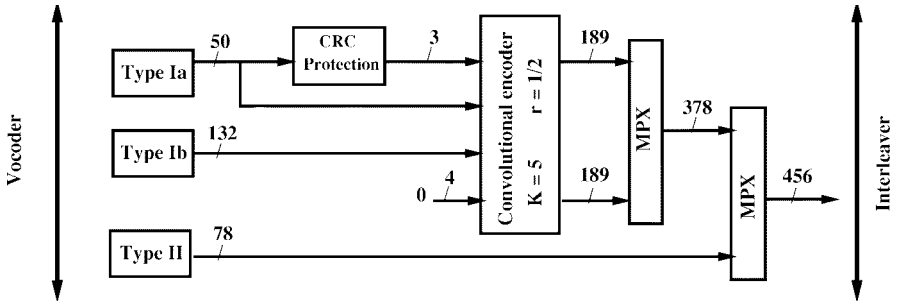

Fig. 4. TCH/FS transmission mode.

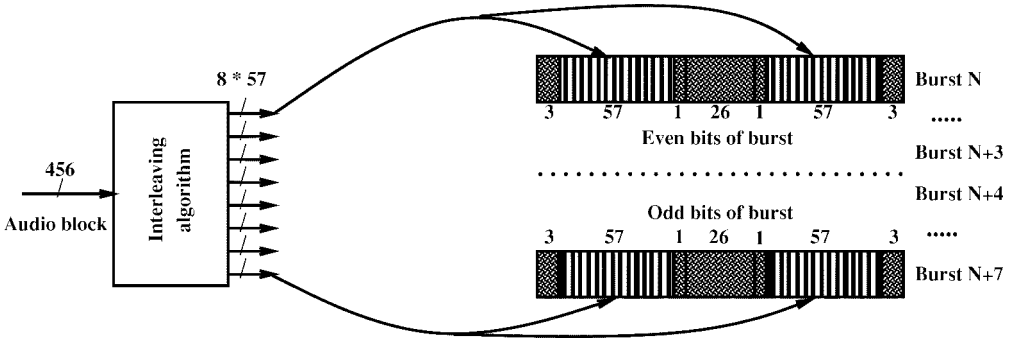

Fig. 5. Interleaving.

TABLE II
CHANNEL CODING PERFORMANCE

Channel Coder	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	0.20	0.50	0.011	0.70	0.018
DEC 3000/800	0.45	0.73	–	0.85	–
Pentium 166	0.38	0.76	0.018	–	0.013
Pentium Pro 200	0.24	0.76	0.019	0.68	0.016

TABLE III
INTERLEAVING PERFORMANCE

Interleaver	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	0.04	0.11	0.002	0.15	0.004
DEC 3000/800	0.12	0.17	–	0.23	–
Pentium 166	0.15	0.30	0.007	–	0.005
Pentium Pro 200	0.045	0.14	0.004	0.13	0.003

Ia) is protected by three CRC bits used for error detection. These bits are computed using the polynomial generator[11]: $G(X) = X^3 \oplus X \oplus 1$. The 3-bit remainder of this division is transmitting along with input data.

2) Convolutional Coding: The convolutional step involves the padding out of the class I sequence with four extra bits and a convolution using the two following polynomials:

$$P_1(X) = X^4 \oplus X^3 \oplus 1 \quad (1)$$

$$P_2(X) = X^4 \oplus X^3 \oplus X \oplus 1. \quad (2)$$

The result of this step is two 189 bit sequences, which are multiplexed with the (unprotected) type II bits to yield a 456 bit coding block, see Fig. 4. The performance of this channel coding algorithm is summarized in Table II.

C. Interleaving

The aim of interleaving is to decorrelate the relative positions of the coded bits within the code words and in the modulated radio bursts. GSM coding blocks are interleaved on eight bursts: the 456 bits of one block are split into eight groups of 57 bits. Each group of 57 bits is then carried in a different burst (see Fig. 5).

The performance of the software implementation is shown in Table III.

D. Ciphering

Ciphering is achieved by performing an XOR (exclusive OR) operation between a pseudorandom bit sequence and the 114 bits of each burst. The deciphering operation is identical to ciphering. The pseudorandom sequence is derived[12] from the burst number and a session key that itself is determined through signaling when a call is established. The performance of the algorithm is shown in Table IV.

E. Modulation

GSM uses Gaussian modulation shift keying (GMSK) [13] with modulation index $h = 0.5$, BT (filter bandwidth times bit period) equal to 0.3, and a modulation rate of 271 kbaud. There are many ways to implement a GMSK modulator. A block diagram of such a modulator is shown in Fig. 6. A

[11] Note that the "$\oplus$" operation stands for the XOR logical operation.

[12] Note that the algorithm used to generate the pseudorandom sequence is not fully included in the public GSM specification.

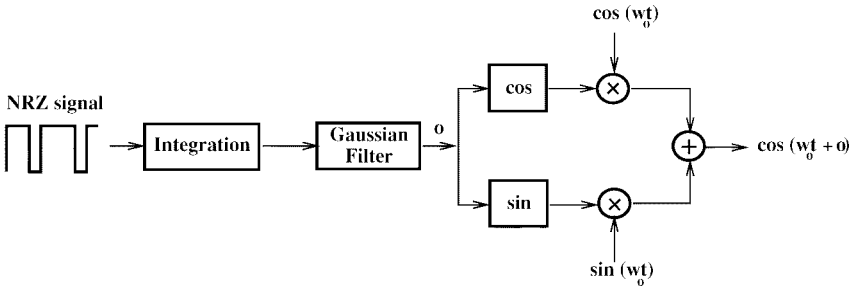

Fig. 6. GMSK modulation block diagram.

Fig. 7. Block diagram of a GSM receiver BTS (uplink).

TABLE IV
CIPHERING AND DECIPHERING PERFORMANCE

De/Cipher	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	0.08	0.19	0.004	0.26	0.007
DEC 3000/800	0.16	0.22	–	0.30	–
Pentium 166	0.09	0.17	0.004	–	0.003
Pentium Pro 200	0.05	0.16	0.004	0.14	0.003

TABLE V
GMSK MODULATOR PERFORMANCE

GMSK Modulator	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	2.1	5.3	0.12	7.4	0.19
DEC 3000/800	4.5	6.2	–	8.4	–
Pentium 166	3.7	7.3	0.18	–	0.12
Pentium Pro 200	1.7	5.4	0.14	4.8	0.11

previous modulator reported in [20] used an IIR algorithm to process the Gaussian filtering. In the current algorithm, the Gaussian filter is achieved by directly using the phase-shaping response of the Gaussian filter $\Phi(t)$. Basically, $\Phi(t)$ consists of a $\pi/2$ step function, smoothed in order to have a more narrow spectrum than if the step were steeper. The new algorithm precomputes the function $\Phi(t)$, and then obtains the sum over all input bits k_i using (3)

$$\phi(t) = \phi_0 + \sum_{i=0}^{147} k_i \, \Phi(t - iT) \ (\phi_0 \text{ may take any value}). \quad (3)$$

Theoretically, a bit k_i influences the output infinitely long. However, in practice [3], this influence becomes negligible outside a $3T$ period. So, the algorithm can precompute pieces of outputs of bit sequences of length 3, and find the whole output by assembling pieces together. The speedup compared to the IIR-based algorithm is about 9 [3]. Table V reports the performance of the new modulator.

IV. FROM RADIO TO SPEECH (UPLINK)

The sequence of uplink modules is shown in Fig. 7. First, the demodulator reconstitutes the GSM bursts. Then, after the deciphering and the deinterleaving operations, the channel decoder recovers the transmitted blocks of data. Finally, a GSM speech decoder converts them into audio samples.

TABLE VI
POLYPHASE TRANSFORM PERFORMANCE
(200 kHz SPACING BETWEEN CHANNELS)

Polyphase Transform	Perf. (%cpu) # frequency channels			
	1	2	4	8
SUN Ultra M170	20	60	180	620
DEC 3000/800	25	80	220	670
Pentium 166	40	110	340	1200
Pentium Pro 200	25	70	210	720

A. Channel Partitioning

The number of frequency channels used by a GSM BTS depends mainly on the density of the network and the type of cells used (micro cells, urban, rural or road cells). It can be one in a low-density area, and two–four in high-density areas. Typically, adjacent channels are not used within the same cell to avoid spectral recovering, and a GSM BTS uses every second channel (i.e., 200 kHz spacing) to avoid high level interferences. In the GSM terminology, a logical user channel corresponds to a speech channel (defined by its frequency and its time slot number), whereas a physical channel stands for an FDMA channel (which contains eight logical channels).

Multichannel receivers for digitized data can be synthesized using a polyphase transform algorithm [7], [22]. Basically, this algorithm performs the following tasks: 1) frequency translation (each center frequency to baseband), 2) bandwidth reduction of the translated spectrum to match the signal bandwidth, and 3) resampling of the output to match the reduced channel bandwidth.

Table VI reports the percentage of CPU per platform required to partition N GSM frequency channels ($N \in [1, 8]$) at a sampling rate equals to 2.5 times the maximal frequency and with a 200 kHz spacing scheme. The filter used in the implementation has 255 coefficients. The corresponding SPEC performances are shown[13] in Table XII, Section V-A.

[13]Note that the SPECint numbers are not very significant since the partitioning algorithm mainly executes floating-point multiplications.

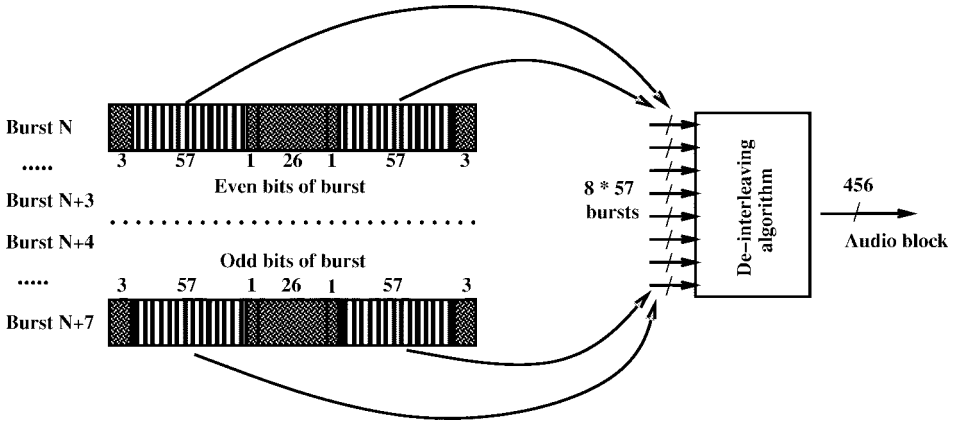

Fig. 8. Deinterleaving.

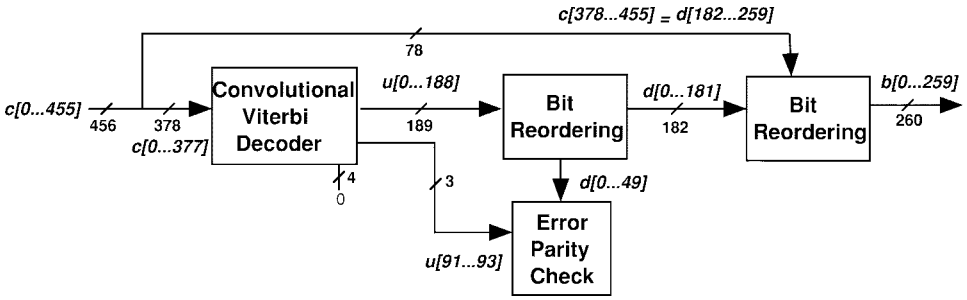

Fig. 9. GSM channel decoding.

The high CPU requirement[14] is due to the high input data rate and the need to compute imaginary and real components of output samples for each frequency channel (requiring multiple floating-point operations).

B. Demodulation and Equalization

The GSM specifications do not impose a particular demodulation algorithm. However, they impose minimal performance criteria, and the algorithm used is expected to cope with two multipaths of equal power received at intervals of up to 16 μs (i.e., more than four symbols). With such a level of intersymbol interference, simple demodulation techniques are ineffective, and an equalizer is required. We are currently implementing an equalizer based on the Viterbi algorithm [4], [14], and this module is expected to be processor intensive[15] [3].

C. Deciphering

Deciphering performs the same operation as the ciphering module described in Section III-D, and hence its performance is summarized by the results in Table IV.

D. Deinterleaving

Deinterleaving performs the inverse operation to the interleaving described in Section III-C. Every 40 ms, eight 114 bit bursts are merged into a 456 bit buffer as shown in Fig. 8.

The performance of this module is shown in Table VII.

TABLE VII
DEINTERLEAVING PERFORMANCE

De-interleaver	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	0.04	0.11	0.003	0.16	0.004
DEC 3000/800	0.17	0.24	–	0.32	–
Pentium 166	0.16	0.32	0.008	–	0.005
Pentium Pro 200	0.04	0.13	0.003	0.12	0.003

E. Channel Decoding

Channel decoding involves the retrieval of the original compressed speech data from the (possibly corrupted) received flow.

Fig. 9 shows the different steps of the channel decoding algorithm. The deinterleaving algorithm generates a 456 bit data buffer from eight GSM bursts. Some of these bits (the 378 bits corresponding to the Type I data bits) are fed into the convolutional decoder, which tries to reconstitute the 189 bits corresponding to the original sequence. After a bit-reordering step, the 50 bits that have the highest priority (Type Ia) are checked using an error control algorithm. If there is no error, a final bit reordering is performed. These steps are further described in the following paragraphs.

1) Convolutional Decoding: Convolutional decoding can be performed using the Viterbi algorithm. The encoder memory is limited to K bits; a Viterbi decoder in steady-state operation takes only 2^{K-1} paths. Its complexity increases exponentially with the constraint length K, which is equal to 5 for the GSM convolutional code; see Section III-B.

[14] The software implementation of this algorithm has been optimized in C.

[15] It has previously been shown that a GSM BTS equalizer can be implemented using a full TMS 320/C40 DSP, i.e., consuming 50 MIPS.

TABLE VIII
CHANNEL DECODING PERFORMANCE

Channel Decoder	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	1.7	4.2	0.09	5.9	0.15
DEC 3000/800	2.2	3.1	–	4.2	–
Pentium 166	2.1	4.2	0.10	–	0.07
Pentium Pro 200	1.5	4.8	0.12	4.2	0.10

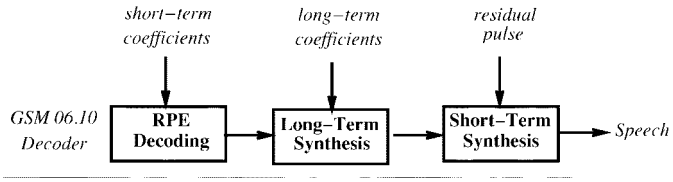

Fig. 10. Block diagram of the GSM speech decoder.

The present implementation is a modified version of the Viterbi decoder[16] developed by Karn for the NASA standard code ($K = 7, r = 1/2$) [8].

2) Reordering Algorithm: The 189 bits generated by the Viterbi decoder ($u(0), u(1), \cdots, u(187), u(188)$) consist of 182 information bits of class 1 ($d(0), d(1), \cdots, d(180)$), three CRC bits, and four tail bits. The relation between u bits and d bits is defined by (4):

$$u(k) = d(2k) \text{ and } u(184-k) = d(2k+1), \quad k \in [0, 90]. \tag{4}$$

3) Error-Detecting Codes: The polynomial operation described in Section III-B is applied to the CRC-protected data. If the remainder differs with the received CRC bits, an error is detected, and the audio frame is ignored and eventually discarded.

4) Reordering Algorithm: The last step of the channel decoding regroups the 260 bits of the speech block from the position that corresponds to the order of decreasing importance ($d(0), d(1), \cdots, d(259)$) to the position that matches their classification ($b(0), b(1), \cdots, b(258), b(259)$) as specified in [5] and [12]. This operation (which requires no effort when implemented in hardware) can be efficiently implemented in software using lookup tables.

The performance of the whole channel decoding algorithm is summarized in Table VIII.

F. GSM Speech Decoding

The decoder reconstructs the speech by passing the residual pulse first through the long-term prediction filter, and then through the short-term predictor (see Fig. 10). The performance of this module is shown in Table IX.

V. PUTTING IT ALL TOGETHER

In this section, we discuss the overall computational requirements of a generic BTS configuration and our efforts to integrate the modules into an experimental platform.

[16] See URL http://people.qualcomm.com/karn/.

TABLE IX
GSM 06.10 DECODING PERFORMANCE

Speech Decoder (RPE-LTP 06.10)	Perf. (metric/user-channel)				
	% cpu	SPECint		SPECfp	
		92	95	92	95
SUN Ultra M170	4.4	11	0.24	15	0.40
DEC 3000/800	5.2	7.2	–	9.8	–
Pentium 166	3.9	7.7	0.19	–	0.13
Pentium Pro 200	2.1	6.7	0.17	5.9	0.14

TABLE X
AGGREGATE GSM MODULE PERFORMANCE

GSM Modules (for one GSM logical channel)	Performance (metric/user-channel)							
	SPECint92		SPECfp92		SPECint95		SPECfp95	
	min	max	min	max	min	max	min	max
Speech coder	14	20	17	28	0.45	0.48	0.34	0.73
Channel coder	0.50	0.76	0.68	0.85	0.011	0.019	0.013	0.018
Interleaver	0.11	0.30	0.13	0.23	0.002	0.007	0.003	0.005
Cipher	0.16	0.22	0.14	0.30	0.004	0.004	0.003	0.007
Modulator	5.3	7.3	4.8	8.4	0.12	0.18	0.11	0.19
Decipher	0.16	0.22	0.14	0.30	0.004	0.004	0.003	0.007
Deinterleaver	0.11	0.32	0.12	0.32	0.003	0.008	0.003	0.005
Channel decoder	3.1	4.8	4.2	5.9	0.09	0.12	0.07	0.15
Speech decoder	6.7	11	5.9	15	0.17	0.24	0.13	0.40
Total	30	45	33	59	0.85	1.1	0.68	1.51

A. Estimating a BTS's Computational Requirements

Table X summarizes the performance of the modules required to encode and decode one GSM logical (TDM) channel, exclusive of the demodulator (including equalizer) and partitioner modules. We give the minimum and maximum SPEC numbers we have obtained with the four different platforms.

Table XI (based on Table X) shows the computational requirements for the encoding and decoding of N GSM physical FDMA channels ($N \in [1, 8]$), where each physical channel supports eight logical user channels.

Here, again, the entries are exclusive of the partitioner and demodulator/equalizer modules. Furthermore, it is assumed that, for each group of up to four physical channels, one of the derived logical channels is dedicated to signaling, and the remaining logical channels require speech processing. This table suggests that a single Sun UltraSparc M170 or Pentium Pro 200 workstation can just about support the encoding/decoding requirements of one physical GSM channel. Fig. 11 shows the corresponding percentage of CPU required per GSM module on the Pentium Pro 200 platform; a total of 93% of CPU is used.

Table XII reports the SPEC-normalized performance of the partitioning algorithm discussed in Section IV-A. The results suggest that our current Sun UltraSparc M170 platform would be sufficiently powerful to perform the partitioning of two physical FDMA channels (i.e., 16 logical channels).

Table XIII provides a summary of computational demands, including the partitioning algorithm and exclusive of the demodulator/equalizer.

Taken together, the above tables suggest that a *minicell* base station supporting two physical channels, i.e., up to 15 logical channels, could be assembled using one processor to perform the partitioning of incoming samples; one processor to multiplex the outgoing sample streams, two processors to per-

TABLE XI
GSM BTS PERFORMANCE (WITHOUT PARTITIONING AND EQUALIZATION)

BTS # frequency channels	# log. ch data/total	SPECint92 min	SPECint92 max	SPECfp92 min	SPECfp92 max	SPECint95 min	SPECint95 max	SPECfp95 min	SPECfp95 max
1	7/8	240	360	260	470	6.8	8.8	5.5	12
2	15/16	480	720	530	940	14	18	11	24
4	31/32	960	1440	1060	1890	27	35	22	48
8	62/64	1920	2880	2110	3780	54	70	44	96

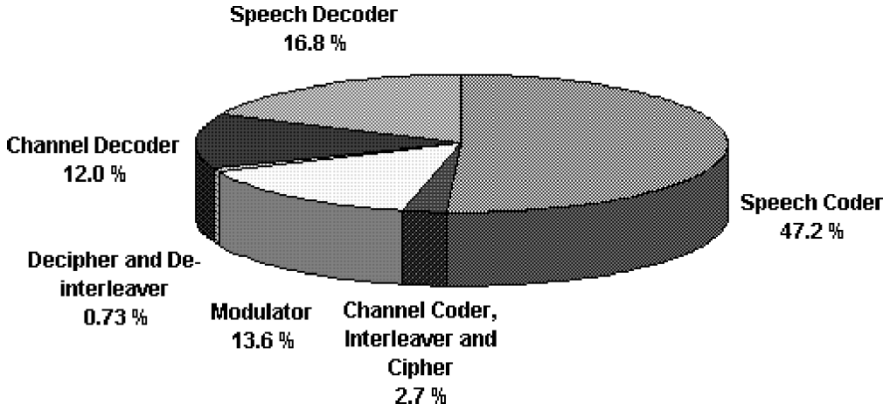

Fig. 11. CPU requirement for eight logical channels on a Pentium Pro 200.

TABLE XII
PARTITIONING PERFORMANCE

BTS # frequency channels	SPECint92 min	SPECint92 max	SPECfp92 min	SPECfp92 max	SPECint95 min	SPECint95 max	SPECfp95 min	SPECfp95 max
1	35	80	47	70	1.1	2.0	1.4	2.0
2	110	220	150	210	3.3	5.6	3.7	5.6
4	300	670	410	630	10	17	11	17
8	930	2400	1300	2300	34	57	40	58

form the encoding/decoding, and some number of additional processors to realize the equalization/demodulation stages. Even after allowing a considerable margin for management and overhead,[17] it will be possible to realize a complete two channel system using a cluster of 8–15 workstations, or possibly, a single multiprocessor server. Although this seems rather expensive today, improvements in processor technology suggest that this level of functionality will be available in a standard server within three years.

B. Module Integration

With the exception of the equalizer, which is currently being implemented, the modules described in this paper were implemented in C and C++. These modules have been incorporated into the VuSystem programming environment, which supports their integration into experimental applications. Fig. 12 shows the flow graph of the VuSystem-based BTS application. The next steps will be to tune the performance of the individual modules, and to experiment with the system as a whole. The project is moving toward *real-world* experiments, initially using an RF channel simulator, such as the HP 11759c. The goal is, eventually, to demonstrate interoperation of the uplink modules with a production handset. Two key questions to be addressed involve: the multitasking and communication overhead associated with the concurrent operation of the entire suite of modules, and the latency characteristics of the software-based system. The latter may be less of an issue here than in other domains, given the built-in latency of the GSM speech coder.

[17] The present analysis is somewhat optimistic concerning the level of CPU utilization that can be attained in a practical system. On the other hand, little effort has been spent tuning the code of the module implementations, which are processor intensive and should be amenable to optimization.

VI. RELATED WORK

The Speakeasy multiband multimode radio is the most aggressive implementation of a programmable radio [9]. That project's objective was to emulate more than 15 existing military radios, operating in several frequency bands (HF, VHF, and UHF). The Speakeasy system was designed to facilitate the addition of new coding and modulation standards. However, the effort put into the DSP processor design, coupled with the low-level programming environment, limit the portability of the system to other processing platforms.

In the commercial sector, one of the first products to implement a programmable radio has been developed by Securor in the U.K. Based on a 50 MHz DSP engine, the radio allows intelligent programmable channeling between 5–25 kHz and adaptive data rate switching using dynamic measurement of bit-error rates [21].

VII. CONCLUSIONS AND FUTURE WORK

This paper has presented the design and implementation of the in-band components of a software-only realization of a GSM base station. The computational requirements of each of the components has been evaluated. Modulation, equalization, and demodulation stages have been identified as the key targets for optimization and tuning.

This paper has introduced the SPEC benchmark metric as a platform-independent means to estimate the requirements of the individual modules. Although this method is not perfect, it provides a very useful way to benchmark the algorithms

TABLE XIII
GSM BTS PERFORMANCE (INCLUDING PARTITIONING AND WITHOUT EQUALIZATION)

BTS # frequency channels	# log. ch data/total	SPECint92 min	SPECint92 max	SPECfp92 min	SPECfp92 max	SPECint95 min	SPECint95 max	SPECfp95 min	SPECfp95 max
1	7/8	275	440	307	540	7.9	10.8	6.9	14
2	15/16	590	940	680	1150	17.3	23.6	14.7	29.6
4	31/32	1260	2110	1470	2520	37	52	33	65
8	62/64	2850	5280	3410	6080	88	127	84	154

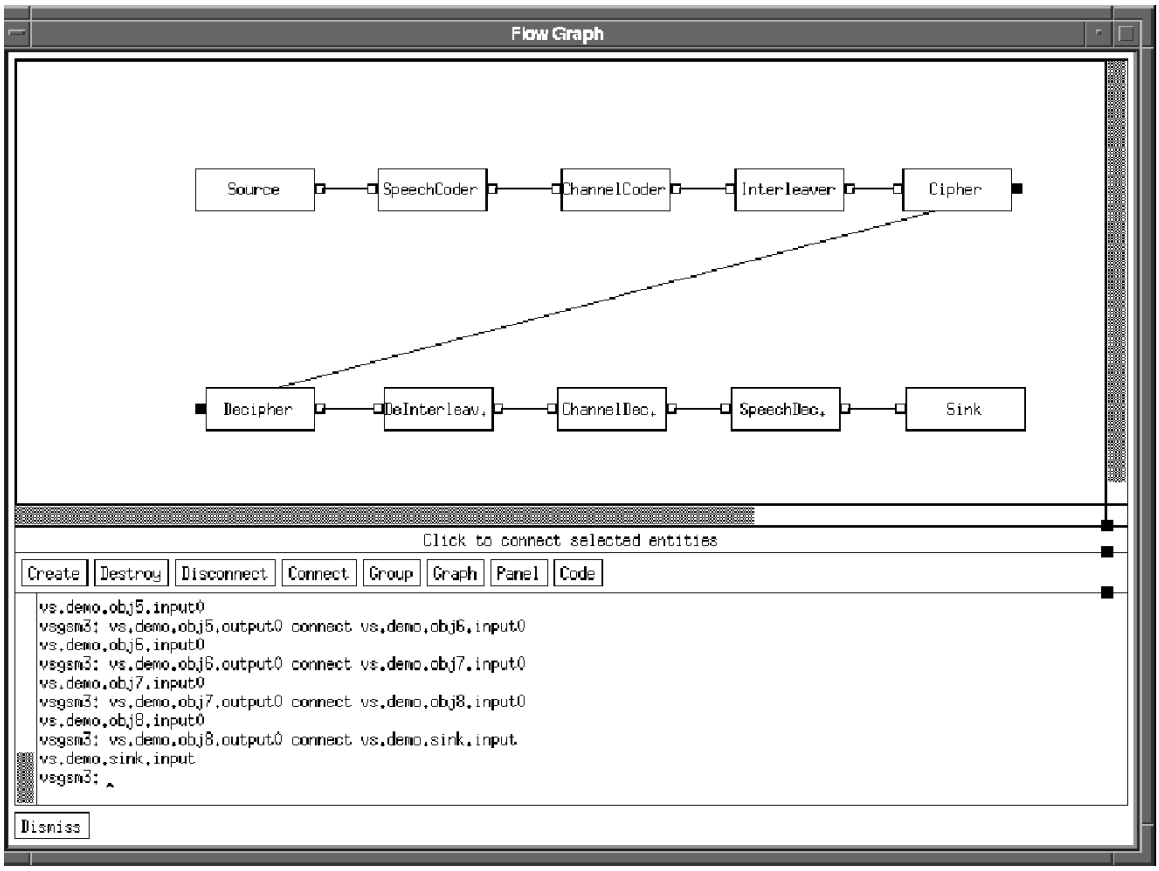

Fig. 12. GSM BTS flow graph.

and to forecast module performance on future platforms. This appears to be the first use of the SPEC metrics as a means of benchmarking communications software to project its performance on future programmable radio platforms.

This approach allows one to project the impact of software-based processing on wireless communications. Within three years, it may be possible to implement a complete BTS on an off-the-shelf server. This suggests that organizations embarking on new design efforts should be cognizant of the software-only approach. As processor and wide-band ADC technology continue to improve, it will be possible to implement increasingly sophisticated and flexible processing on ever wider bands of sampled RF spectrum.

The software approach provides tremendous flexibility for mobile communications systems. For example, changing modulation schemes or frequency assignments only involves the loading and/or configuration of software modules. Similarly, the BTS software can be upgraded to support new modulation schemes without having to upgrade hardware, especially the RF components. At the same time, the scheme preserves the vendor's software investment, as modules can be ported to higher performance platforms, e.g., to support increased cell capacity. Finally, note that the software-only strategy can also be applied to the design of the mobiles; units moving between service providers, or continents, can be dynamically configured to interoperate with local facilities.

The Vusystem programming environment was designed for multimedia applications, and not specifically for virtual radios [10]. A more flexible programming environment supporting adaptive signal processing is currently being designed at M.I.T: the SPECTRA programming environment [2]. SPECTRA will adapt to the environment (e.g., adaptive channel coding), the user requirements (e.g., AM/FM switch), and the available resources (e.g., some robustness or accuracy may be sacrificed if the CPU resources become scarce).

A future SPECTRA environment may include new functionalities to help programmers implement the control-intensive parts of the application. Most applications have both data-intensive and control-intensive handling activities that require different programming techniques. By linking a formal

TABLE XIV
SPEC92 BENCHMARKS

Platform	Clk (MHz) ext/in	Cache ext+I/D	SPECint 92	SPECfp 92	Info date	Source obtained
SUN Ultra M170	83/167	512+16/16	252	351	Nov95	SunIntro
DEC 3000/800	40/200	2M+8/8	138.4	187.6	May94	c.sun.hw
Intel XXpress Pentium	66/166	1M+8/8	197.5	–	Jan96	www.intel
Intel Alder PentiumPro	200	256+8/8	318.4	283.2	Jan96	www.intel

TABLE XV
SPEC95 BENCHMARKS

Platform	Clk (MHz) ext/in	Cache ext+I/D	SPECint 95	SPECfp 95	Info date	Source obtained
SUN Ultra M170	83/167	512+16/16	5.56	9.06	Mar96	c.bmarks
DEC 3000/800	40/200	2M+8/8	–	–	–	–
Intel XXpress Pentium	66/166	1M+8/8	4.76	3.37	Jan96	www.intel
Intel Alder PentiumPro	200	256+8/8	8.09	6.75	Jan96	www.intel

synchronous language such as Esterel [1] to SPECTRA, a large set of GSM signaling functions could be directly implemented from automata described in the GSM specifications.

Finally, one must be prepared to use FPGA's if these devices become available on future general-purpose workstations. They could significantly help with the most CPU-greedy application functions. For example, speedups of close to 50 over strict software implementations have already been achieved for computing the DCT [23].

VIII. SPECmarks FOR SEVERAL MACHINES

Tables XIV and XV report, respectively, SPEC92 and SPEC95 benchmarks for the four platforms[18] used in our experiments.

ACKNOWLEDGMENT

The authors would like to thank J. Kurose and the Editors for several comments and suggestions that have improved the quality of this paper.

REFERENCES

[1] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Sci. Comput. Programming*, vol. 19, no. 2, pp. 87–152, 1992.
[2] V. Bose, M. Ismert, M. Welborn, and J. Guttag, "Virtual radios," this issue, pp. 591–602.
[3] H. Bentzen, "A software GSM base station," Master's thesis, ETH Zurich, M.I.T., Cambridge, Mar. 1997.
[4] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.
[5] "GSM 06.10—European digital cellular telecommunications system (phase 2); Full rate speech transcoding," *ETS 300 580-2, European Telecommunication Standard*, Sept. 1994.
[6] B. Hedberg, "Technical challenges in introducing software radio for mobile telephony base stations," in *ACTS Software Radio Workshop'97*, Brussels, Belgium, May 1997.
[7] F. Harris and D. Steinbrecher, "Wireless interface design for digital processors," in *ICUPC'93 Tutorial 5*, Oct. 1993.
[8] P. Karn, "Convolutional decoders for amateur packet radio," in *Proc. ARRL'1995 Digital Commun. Conf.*, 1995.
[9] L. Lackey and U. Upmal, "Speakeasy: The military software radio," *IEEE Commun. Mag.*, vol. 33, pp. 56–61, May 1995.
[10] C. J. Lindblad, D. Wetherall, and D. L. Tennenhouse, "The VuSystem: A programming system for visual processing of digital video," in *Proc. ACM Multimedia 94*, San Francisco, CA, Oct. 1994.
[11] J. Mitola, "Software radios survey, critical evaluation and future directions," in *Proc. IEEE Nat. Telesys. Conf.*, Washington, DC, May 1992.
[12] M. Mouly and M. B. Pautet, *The GSM System for Mobile*, ISBN 2-9507190-0-7, 1992.
[13] K. Murota and K. Hirade, "GMSK modulation for digital radio telephony," *IEEE Trans. Commun.*, vol. COM-29, pp. 1044–1050, July 1981.
[14] S. Ono, H. Hayashi, T. Tanak, and N. Kondoh, "A MLSE receiver for the GSM digital cellular system," in *Proc. 44th IEEE Veh. Technol. Conf.*, June 1994, pp. 230–233.
[15] J. K. Ousterhout, *Tcl and the Tk Toolki* (Addison-Wesley Professional Computing Series). Reading, MA: Addison-Wesley, 1994.
[16] L. C. Steward, A. C. Payne, and T. M. Levergood, "Are DSP chips obsolete?," TR 92/10 DEC Cambridge Res. Lab., Cambridge, MA, Nov. 1992.
[17] D. L. Tennenhouse and V. G. Bose, "SpectrumWare—A software-oriented approach to wireless signal processing," in *Proc. ACM Mobile Computing and Networking 95*, Berkeley, CA, Nov. 1995.
[18] D. L. Tennenhouse, T. Turletti, and V. G. Bose, "The spectrumware tesbed for ATM-based software radios," in *Proc. ICUCP'96*, Boston, MA, Sept. 1996.
[19] T. Turletti, "A brief overview of the GSM radio interface," Tech. Memo., TM 547, MIT, Mar. 1996.
[20] T. Turletti and D. L. Tennenhouse, "Estimating the computational requirements of a software GSM base station," in *Proc. ICC'97*, Montreal, Canada, June 1997, pp. 169–175.
[21] W. Tuttlebee, "The impact of software radio," in *ACTS Software Radio Workshop '97*, Brussels, Belgium, May 1997.
[22] P. P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and applications: A tutorial," *Proc. IEEE*, vol. 78, Jan. 1990.
[23] R. D. Wittig and P. Chow, "OneChip: An FPGA processor with reconfigurable logic," in *Proc. IEEE Symp. FPGA's for Custom Compute Machines (FCCM'96)*, 1996.

[18] Some SPECmarks are missing in the list of benchmarks available in the public domain; see URL www.specbench.org/osg/cpu95/results/.

Thierry Turletti (M'97), for a photograph and biography, see this issue, p. 513.

Hans J. Bentzen received the M.Sc. degree in electrical engineering from the Swiss Federal Institute of Technology, where he wrote his thesis on the Software Devices and Systems Group. The thesis, "A software based GSM basestation," was part of the SpectrumWare project.

He is an Associate at McKinsey & Company. He joined the Zurich Office of McKinsey & Company in early 1997. His interests include virtual radios and cryptography.

David Tennenhouse (M'87), for a biography, see this issue, p. 513.