HIGH QUALITY, LOW BIT-RATE SPEECH CODING FOR LOW-POWER VLSI IMPLEMENTATION

by

Paul Landman

Memorandum No. UCB/ERL M91/41

20 May 1991

COVER PAGE

HIGH QUALITY, LOW BIT-RATE SPEECH CODING FOR LOW-POWER VLSI IMPLEMENTATION

by

Paul Landman

Memorandum No. UCB/ERL M91/41

20 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Acknowledgements

This report was made possible by the support and guidance of several individuals. Foremost, I would like to recognize my research advisor, Professor Jan Rabaey, for his invaluable contributions during all phases of this project. Furthermore, I thank all of the members of his research group for their assistance and patience in answering my questions. Likewise, I am grateful to Professor Robert Brodersen and his students for their equally kind support and assistance.

In addition, I would like to acknowledge my parents, Art and Fran Landman, who were an endless source of comfort throughout my academic career.

Finally, I would like to dedicate this report to Karen Lugenbehl, the guardian of my sanity for the last five years.

This research was sponsored by a fellowship from the National Science Foundation as well as DARPA grants N00039-88-C-0292 and J-FBI 90-073.

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Introduction

Over the past several decades, the topic of speech analysis and processing has been the focus of much research and study. Considering the importance of the spoken word in communications, this is not surprising. Indeed, the progress made in the development of the digital signal processing field has largely been driven by its applications to the speech signal. These applications are wide ranging and include speech recognition, synthesis, encryption, and compression among others. Of course, speech and audio signals are but one form of communication – video is an equally important medium. Recently, the possibility of combining these two capabilities into a single portable, personal communications system (PCS) has stirred a great deal of interest.

The portability requirement, however, raises several important issues. First, this form of personal, wireless communications suggests a cellular environment. In such an environment, communication bandwidth becomes a severe constraint. This has been the case for existing cellular telephone networks, and the situation will only be exacerbated by the addition of full-motion video. Thus, compression of both audio and video channel information will be a key issue.

Furthermore, as with all portable, battery-operated systems, power dissipation will be a critical concern. By minimizing the power consumed by each component of the system, the operating time between recharging sessions can be maximized. One could argue that this could also be accomplished by increasing the storage capacity of the batteries; however, batteries add both mass and volume to a portable system. The weight of the batteries

required to power a 25 Watt system for 8 hours is approximately 40 pounds, assuming modern battery technologies. Therefore, it is clear that power conservation is not a trivial concern.

In summary, data compression and low-power operation are two important issues in the development of a portable multi-media terminal. This report will address these issues through a case study of low-power speech coding. Of course, the ideas presented here are not limited to a single application. The speech coding techniques could, for example, be equally well applied to mobile telephony, secure communications, or voice mail. Moreover, the low-power techniques developed in this paper are applicable not only to speech processing, but to virtually any digital signal processing (DSP) application constrained to low-power operation.

1.2 Organization

The contents of this report can be decomposed into several sections. Chapter 2 describes the mechanism of power consumption in CMOS circuits and proposes several strategies for low-power VLSI. Chapter 3 then provides the reader with a brief background on speech processing – coding, in particular. Moreover, it describes the concepts of linear predictive coding (LPC) and code-excited linear prediction (CELP). Chapter 4 relies heavily on the backgrounds built up in the previous chapters. Basically, this chapter combines the notions of low-power VLSI and of speech coding to develop architectural and algorithmic techniques for low-power speech coders. Finally, chapter 5 discusses several possible directions for future work, and chapter 6 summarizes the results of the research.

Chapter 2

Low-Power VLSI Systems

2.1 Power Consumption in CMOS Circuits

One of the objectives of this report is to develop a methodology for achieving low-power VLSI implementations of digital signal processing algorithms. Therefore, it would be judicious at this point to discuss the mechanics of power consumption in CMOS circuits. Consider the CMOS inverter of figure 2.1. The power consumed when this inverter is in use can be decomposed into three basic components: static power, dynamic power, and short-circuit ($V_{dd} - V_{ss}$) power. Each of these components will now be analyzed individually.

Ideally, CMOS circuits dissipate no static (DC) power since in the steady state there is no direct path from V_{dd} to ground. Of course, this scenario can never be realized in practice since in reality the MOS transistor is not a perfect switch. Thus, there will

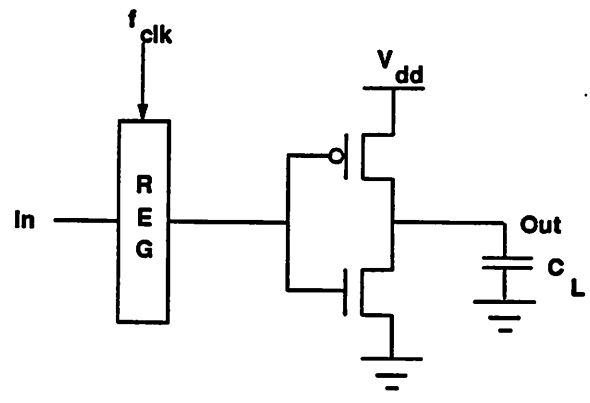


Figure 2.1: CMOS Inverter for Power Analysis

always be leakage currents and substrate injection currents, which will give rise to a static component of CMOS power dissipation. For a submicron NMOS device with an effective $\frac{W}{L} = \frac{10\mu m}{0.5\mu m}$, the substrate injection current is on the order of 1-100 μA for a V_{dd} of 5 V [1]. Since the substrate current reaches its maximum for gate voltages near $0.4V_{dd}$ and since gate voltages are only transiently in this range as devices switch, the actual power contribution of the substrate injection current is several orders of magnitude below other contributors. Likewise, reverse-bias junction leakage currents are on the order of nanoamps and will have little effect on overall power consumption.

Another component of power dissipation arises from the transient switching behavior of the CMOS device. At some point during the switching transient, both the NMOS and PMOS devices will be turned on. This occurs for gates voltages between V_{tn} and $V_{dd} - |V_{tp}|$. During this time, a short-circuit exists between V_{dd} and ground and currents are allowed to flow. A detailed analysis of this phenomenon by Veendrick reveals that for traditional CMOS circuits this component is approximately 10% of the dynamic power dissipation of the circuit [2]. Thus, although it can not be neglected, it is certainly not the dominant component of power dissipation in CMOS circuits.

Instead, dynamic power dissipation consumes most of the power used by CMOS circuits. Dynamic power dissipation is the result of charging and discharging parasitic capacitances in the circuit. The situation is modeled in figure 2.1 where the parasitic capacitances are lumped at the output in the load capacitor, C_L. Assume the inverter has as input and output the waveforms of figure 2.2. The frequency, f, of the input signal corresponds to a clock frequency in figure 2.1 of $f_{clk} = 2f$. The total power dissipation of the circuit, then, is given by:

$$P_{dyn} = \frac{1}{T}\int_0^{\frac{T}{2}} I_{dn}(t)V_o(t)dt + \frac{1}{T}\int_{\frac{T}{2}}^{T} I_{dp}(t)(V_{dd} - V_o(t))dt \tag{2.1}$$

Using the fact that $I_{dn}(t) = C_L\frac{dV_o}{dt}$ and $I_{dp}(t) = C_L\frac{d(V_{dd}-V_o(t))}{dt}$ we can solve for the power, $P_{dyn} = C_L V_{dd}^2 f = \frac{1}{2}C_L V_{dd}^2 f_{clk}$. This is the classical result for dynamic CMOS power consumption. It illustrates that, assuming continual switching, the dynamic power is proportional to the frequency, the capacitive loading, and the square of the supply voltage. In CMOS circuits, this component of power dissipation is by far the most important (accounting for over 90% of the total power dissipation), as illustrated by the previous discussion.

Having derived the expressions for CMOS power consumption, it is interesting to analyze how power consumption is distributed on a VLSI chip. It is important to realize

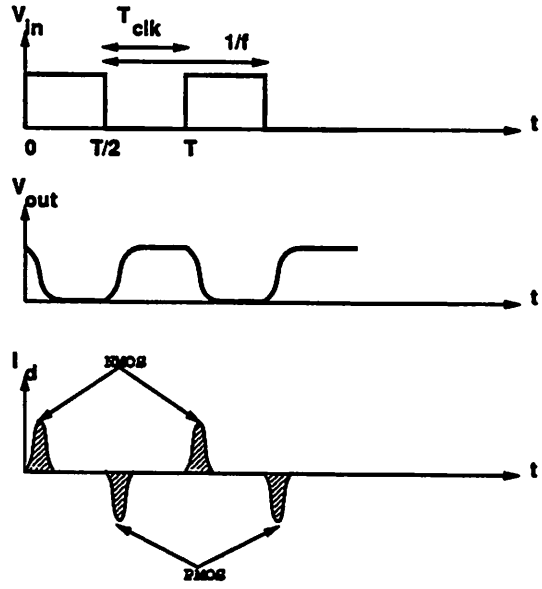


Figure 2.2: Waveforms for Power Analysis (after Weste)

that not only does the core circuitry consume power, but also the I/O drivers consume a significant amount of power. This is not surprising since I/O capacitances are on the order of tens of picofarads while on-chip capacitances are in the tens of femtofarads. More specifically, for conventional packaging technologies, Bakoglu suggests that pins contribute approximately 13-14 pF of capacitance each (10 pF for the pad and 3-4 pF for the printed circuit board traces) [3]. Since dynamic power is proportional to capacitance, I/O power is clearly a significant portion of overall chip power consumption.

2.2 Strategies for Low-Power VLSI

2.2.1 Packaging

The preceding analysis of power consumption in CMOS circuits can be exploited to yield several techniques for reducing power dissipation. The realization that I/O capacitance at the chip-level can account for as much as 1/4 to 1/2 of the overall system power dissipation suggests that reduction of I/O power is a necessity in multi-chip systems such as the portable communications terminal. For I/O power, the issue, as stated before, is mainly one of packaging technology. If the large capacitances associated with inter-chip I/O were drastically reduced, the I/O component of system power consumption

would be reduced proportionally. One currently evolving method of achieving this goal is the multi-chip module or MCM.

In an MCM, all of the chips comprising a given system are mounted on a single substrate, and the entire module is placed in a single package. Utilizing this technology, inter-chip I/O capacitances are reduced to the same order as on-chip capacitances. This is due not only to the elimination of the highly capacitive PCB trace interconnect, but also to the reduction of on-chip pad driver capacitances due to the minimized off-chip load driving requirements. Thus, utilizing MCM technology, the I/O component of system power consumption can be kept at a minimum, shifting the focus of power optimization from I/O considerations to chip core considerations.

Actually, low-power operation is but one of the advantages of MCM technology. In addition, MCM's with their reduced chip level interconnect lengths and capacitances can significantly reduce system delays resulting in possibly higher clock speeds and correspondingly higher performance levels. Furthermore, this packaging technique raises the effective system integration level several orders of magnitude over existing packaging technologies. For projected submicron technologies, an $8'' \times 10''$ MCM can be expected to house close to a billion transistors [4]. This will relax current silicon area constraints and allow much needed flexibility in designing low-power architectures such as those discussed in this report.

2.2.2 Design Styles

Logic design styles can also influence power consumption in CMOS circuits. At the highest level, the choice of logic styles is between static and dynamic. Historically, dynamic design styles have been touted for their inherent low-power properties. For example, dynamic design styles often have significantly reduced device counts. Figure 2.3 shows an implementation of a complex boolean expression in both static and dynamic design styles. For the dynamic design, the logic evaluation function is fulfilled by the NMOS tree alone, while in the static design both an NMOS and PMOS tree are required for this operation. Since dynamic device counts are drastically reduced, corresponding capacitive loading is also reduced; this, in turn, can lead to power savings.

In addition, dynamic gates don't experience short-circuit current power dissipation. In contrast, whenever static circuits switch, a brief pulse of transient current flows from V_{dd} to ground, consuming power. Furthermore, dynamic logic nodes are guaranteed to

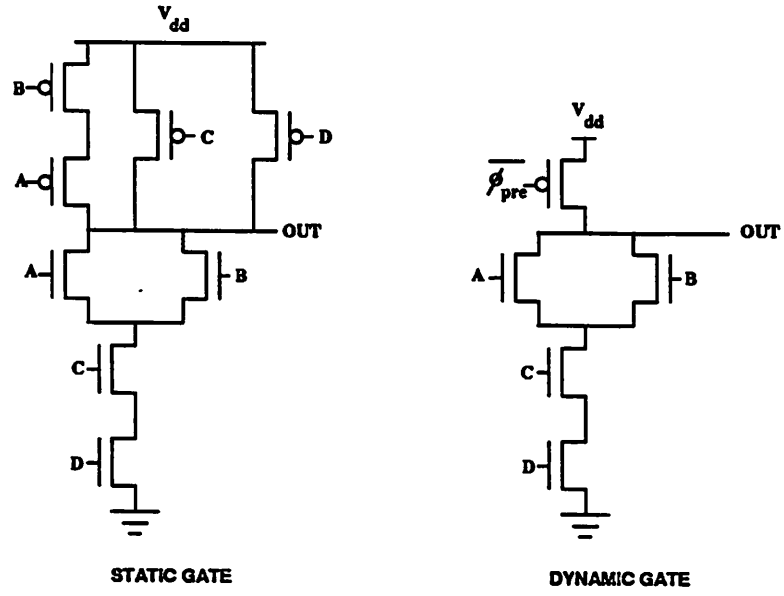


Figure 2.3: Static and Dynamic Implementations of a Complex Logic Function

have a maximum of one transition per clock cycle. Static gates, unfortunately, do not follow this pattern and can experience a glitching phenomenon whereby output nodes undergo spurious transitions before settling at their final value. This causes excess power dissipation in static gates.

In practice, however, situations can arise that result in significantly increased power dissipation in dynamic circuits. Consider a dynamic gate whose output is low over a large number of clock cycles. During the precharge phase, the output node is charged to the V_{dd} supply rail. When the gate is evaluated, however, the node is pulled down to ground. This pattern repeats over many clock cycles as long as the output of the gate remains low. Thus, power is consumed during each clock cycle. If this were a static implementation, however, only power due to leakage currents would be consumed since no switching would take place. For certain data patterns, then, static logic can actually result in lower power consumption overall.

Dynamic logic styles have additional drawbacks. For example, although reduced device counts tend to lower circuit capacitances, dynamic gates often require large precharging transistors to meet timing constraints. This, coupled with the fact that every dynamic gate must have at least one precharge transistor, can lead to huge clock line capacitances, which, aside from making reliable clocking difficult, also consume a great deal of power.

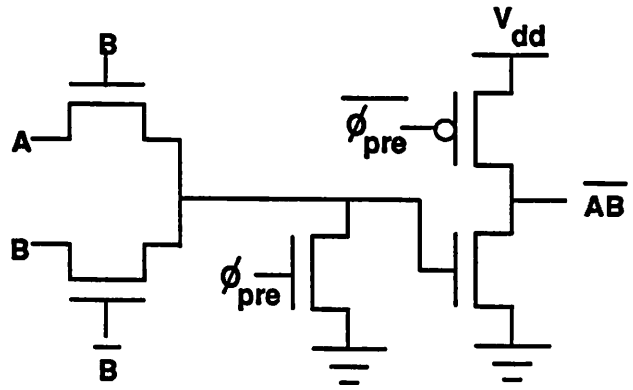


Figure 2.4: Dynamic CPL NAND Gate

Also, there is the related issue of clock distribution and skew. As clock rates and die sizes increase, these issues become increasingly important and can significantly limit the utility of dynamic circuits. Of course, similar clocking issues arise in static logic; however, the problem is not nearly as severe as in the dynamic case.

Furthermore, dynamic circuits are more difficult to design. For example, avoiding charge sharing and its associated difficulties requires careful consideration by the designer. As issues such as this arise, design times for dynamic circuits increase and dynamic logic styles become less attractive.

Clearly, dynamic design styles do, however, merit serious consideration in the development of a low-power circuit library. Indeed, adder simulations by Chandrakasan, et. al.,[4] demonstrate the low-power possibilities of dynamic circuits. In particular, their findings suggest dynamic complementary pass-gate logic (CPL) [5] as a possible candidate for a low-power design style. A dynamic CPL NAND gate is depicted in figure 2.4. The NMOS pass-transistor network performs the logic evaluation while the succeeding buffer stage drives the fan-out of the gate. An additional clocked NMOS transistor is provided to ensure that the final, inverting NMOS is off during the precharge phase. Note that this "N" stage would be preceded and followed by "P" stages based on PMOS transistors. Thus, the A and B inputs would be precharged low, disabling the pass transistors and allowing evaluation to proceed in a "domino"-like fashion. Clearly, for complex pass-transistor networks, the buffer and precharge transistor overhead becomes less significant; moreover, even if it were not required, a buffer would probably be advisable to ensure adequate load driving capabilities.

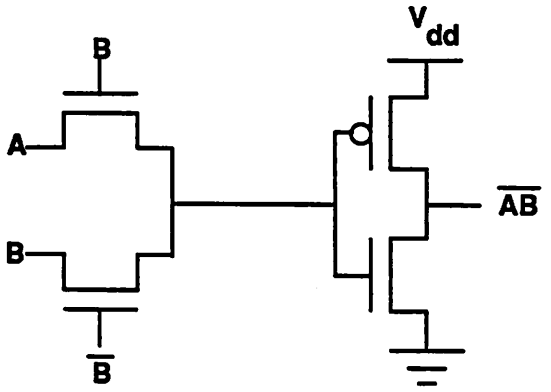


Figure 2.5: Static CPL NAND Gate

It is interesting to note, however, that the advantages of dynamic CPL are more a function of the logic style rather than the dynamic circuit style. Figure 2.5 illustrates that the static and dynamic versions differ only in the connection of the PMOS in the buffering inverter and the presence or absence of the clocked NMOS transistor. Therefore, the capacitance of the static circuit is actually less than the dynamic configuration. In fact, since clock routing and loading are now reduced, additional power savings are possible. Of course, this consideration must be balanced with the possibility of glitching and short-circuit power dissipation. Indeed, since the NMOS pass-transistors exhibit a V_t drop in the transmission of high logic levels, the PMOS buffer devices could remain slightly conducting even for a "high" logic level input. This situation can be remedied with a zero-V_t NMOS pass-transistor or a PMOS level-restorer as illustrated in figure 2.6. Overall, CPL has proven to be a fast, low-power circuit style and deserves consideration in the construction of a low-power CMOS cell library.

2.2.3 Circuit Techniques

In addition to judicious selection of design style, circuit-level techniques can also allow designers to reduce power consumption. One example where custom circuits can provide large gains in power reduction is in clock distribution. Switching power consumed in charging clock lines and transistor gates often dominates chip power consumption. This is especially true for dynamic logic in which each logic gate has at least one clocked transistor, requiring extensive clock distribution. In order to reduce clock line charging power (which is proportional to V^2), one can attempt to reduce the voltage swing on large interblock

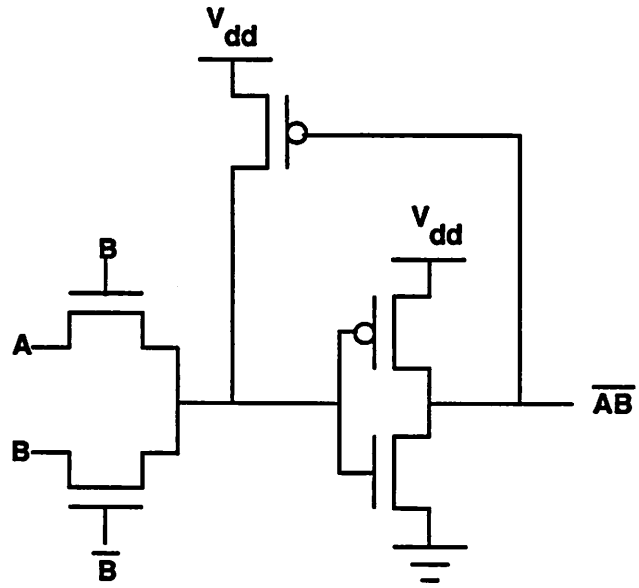


Figure 2.6: CPL NAND Gate with PMOS Level Restorer

portions of the clock line. This would, of course, require sense amps of some kind at the boundaries of each block to restore full logic levels; however, significant power gains might be possible with such a technique. Similarly, in a multi-chip, low-power system these reduced logic swing techniques could be used at chip boundaries to reduce inter-chip I/O power.

2.2.4 Technology Scaling

Scaling of physical dimensions is another consideration for low-power circuits. Analysis of device scaling is commonly coupled with a treatment of supply voltage scaling. In this report, however, they are treated separately to emphasize the effect of each technique independently on the parameters of interest. Moreover, in attempting full scaling (dimensions and supply), subthreshold conduction limitations prevent supply scaling from proceeding to the same extent as dimensional scaling. The independent scaling analyses account for this situation.

Basically, scaling involves reducing all vertical and horizontal dimensions by a factor, S, greater than one. Thus, transistor widths and lengths are reduced, oxide thicknesses are reduced, depletion region widths are reduced, interconnect widths and thicknesses are reduced, etc. From classical analysis [6], the first order effects of such a scaling on important parameters can be derived. Device loading capacitances, for example, are of the

form, $C = C_{ox}WL$. But $C_{ox} \propto \frac{1}{t_{ox}} \propto S$ and $W \propto \frac{1}{S}$ as is L. The result is that gate capacitances scale down by the factor S. This bodes well for reduced power consumption since $P \propto CV_{dd}^2 f$. So if f and V_{dd} remain constant, power also sees a reduction of S. The effect of scaling on delays is equally promising. First, notice that current drive increases: $I \propto C_{ox}\frac{W}{L} \propto S$. Thus, propagation delays, which are proportional to $\frac{CV_{dd}}{I}$, scale down by a factor of S^2. The power-delay product, then, shrinks by a factor of S^3.

This discussion, however, ignores many important second order effects. For example, as dimensions scale down, more functionality tends to be placed on a single chip leaving die sizes relatively constant. As a result interconnect lengths don't scale down by the full scaling factor S. In fact, they tend to remain constant. As a result, interconnect capacitance now remains constant rather than scaling down by S. Thus, it begins to dominate gate capacitance terms, which scale down by S. As this occurs, power consumption is no longer reduced by S; instead, it too remains constant. Furthermore, delays are now reduced only linearly, rather than quadratically, with S. Finally, combining these results, the power-delay product sees only a factor of S improvement overall.

Another second order effect is that of velocity saturation. As dimensional scaling continues to submicron dimensions without commensurate scaling of supply voltage, high electric fields develop in device channels. Eventually, carrier mobility is heavily degraded due to increased carrier collisions with the crystal lattice; this, in turn, brings on velocity saturation. The net effect is the reduction of the current drive voltage dependence from a square law to a linear relation. As would be expected, this significant reduction of current drive has an adverse effect on circuit delays. Nevertheless, technology scaling does offer significant advantages for low-power CMOS circuits.

Unfortunately, scaling is not always a viable option. Aside from the drawbacks of interconnect non-scalability and submicron effects, chip designers often don't have complete freedom to arbitrarily scale their fabrication technology. Instead, the capabilities of their fabrication facilities impose limits on minimum lithographic dimensions. For this reason, in order to achieve widespread acceptance, an ideal low-power methodology should not rely solely on technology scaling or specialized processing techniques. Indeed, the methodology should be applicable not only to different technologies, but also to different circuit and logic styles. Of course, whenever possible scaling and circuit techniques should be combined with the high-level methodology to further reduce power consumption; however, the general low-power strategy should not *require* these tricks. Again, the advantages of scaling and

low-level techniques cannot be overemphasized, but they should not be the sole arena from which the designer can extract power gains.

2.2.5 Supply Voltage Scaling

Having treated dimensional technology scaling, the effects of voltage scaling will now be considered independently. Specifically, consider reducing the supply voltage by a factor, S_v, greater than one. Since dynamic power consumption is proportional to the square of the supply voltage, power dissipation is reduced in a quadratic fashion with S_v. This square-law reduction offers a direct and dramatic means of minimizing energy consumption. Without requiring any special circuits or technologies, a factor of two reduction in supply voltage yields a factor of four decrease in power. Furthermore, this power reduction is experienced not only in the chip core, but also for chip I/O (assuming the chip is communicating with other low-voltage components).

This savings does not, however, come without a cost. For as supply voltage is lowered, circuit delays increase. To the first order, $I_{dd} \propto V_{dd}^2 \propto \frac{1}{S_v^2}$. Thus, propagation delays, which go as $\frac{CV_{dd}}{I}$, will increase linearly with S_v. In order for the circuit to remain functional, these delay increases cannot go unchecked. Some techniques must be applied, either technological or architectural, in order to compensate for this effect. This is the topic of the next section. First, however, a short discussion of second order effects in supply scaling is in order.

Several important refinements can be added to the classical voltage scaling model. First, the quadratic dependence of current drive on supply voltage is an approximation valid only for supply voltages much larger than the device threshold voltage, V_t. In reality, $I \propto (V_{dd} - V_t)^2$ and, unfortunately, V_t cannot be scaled down indefinitely with supply voltage. Instead, it is limited to approximately 0.6 V, by subthreshold conduction and noise margin considerations. As the supply voltage decreases, the significance of this factor is manifested as an undesirable loss of current drive.

Another, previously mentioned, second order effect is velocity saturation. Since voltage scaling cannot track dimensional scaling beyond V_t^{min}, it will not prevent high channel fields from eventually developing, causing velocity saturation. Consequently, even under voltage scaling conditions, saturated device currents will begin to drop and propagation delays will increase. This further limits the amount that the voltage supply can be reduced

without incurring unacceptable delay penalties.

2.2.6 Architectures

Parallelism

In addition to the several low-level strategies presented, certain architectural issues deserve consideration. As stated in the previous section, supply voltage scaling causes linear increases in propagation delays which must be dealt with in order to maintain constant system throughput. Parallel computing techniques are ideal for this objective [4]. In this context, architectural parallelism refers not only to systems with physical hardware duplication, but also to pipelined systems. In general, any technique that increases computational concurrency can be used to compensate for the negative effects of voltage scaling.

As an example, consider a functional block that performs some complex operation, A, as illustrated in figure 2.7. The registers supplying operands and storing results for A are clocked at a frequency, f. Further assume that algorithmic and data dependency constraints do not prevent concurrency in the calculations performed by A. When the computation of A is parallelized, figure 2.8 results. Basically, the hardware comprising block A has been duplicated N times resulting in N identical processors. Since there are now N processors, a throughput equal to that of sequential processor, A, can be maintained with a clocking frequency N times lower than that of A. That is, although each block will produce a result only $1/Nth$ as frequently as processor A, there are N such processors producing results. Consequently, identical throughput is maintained.

The key to this architecture's utility as a power saving configuration lies in this factor of N reduction in clocking frequency. In particular, with a clocking frequency of f/N, each individual processor can run N times slower. Since delays vary approximately linearly with voltage supply, this corresponds to a possible factor of N reduction in supply voltage. Examining the dynamic power consumption relative to the single processor configuration, we see that capacitances have increased by a factor of N (due to hardware duplication), while frequency and supply voltage have been reduced by the same factor. Thus, since $P_{dyn} \propto CV_{dd}^2 f$, dynamic power consumption is reduced by the square of the concurrency factor, N (i.e. $P_{dyn} \propto \frac{1}{N^2}$).

Parallelism is not, however, the only form of concurrent computation that can be exploited for power reduction – pipelining can be equally interesting. The pipelined case

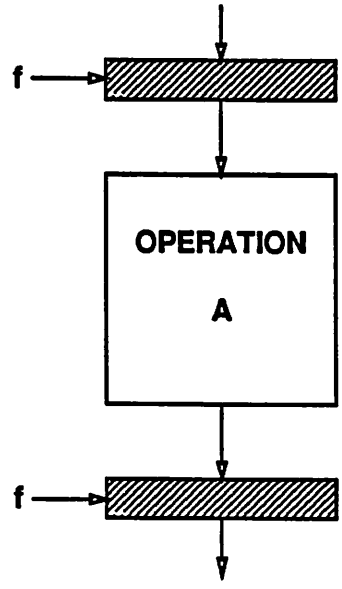


Figure 2.7: Sequential Processor for Operation A

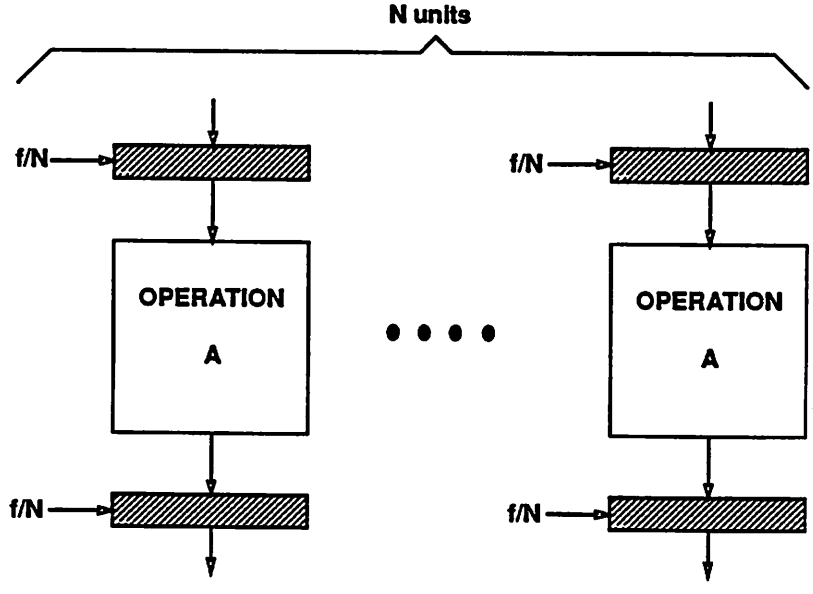


Figure 2.8: N-way Parallel Processor for Operation A

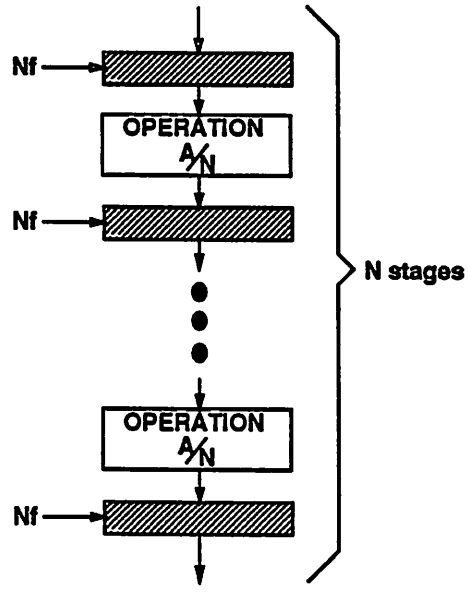


Figure 2.9: N-stage Pipelined Processor for Operation A

is shown in figure 2.9. In this situation, rather than duplicating hardware, concurrency is achieved by inserting pipeline registers, arriving at an N-stage pipelined version of processor A (assuming processor A can be pipelined to this extent). In this implementation, maintaining throughput requires that we maintain clocking frequency, f; however, ignoring the overhead of the pipeline registers, the capacitance, C, also remains constant. The advantage of this configuration is derived from the greatly reduced computational requirements between pipeline registers. Rather than performing the entire computation, A, within one clock cycle, only $1/Nth$ of A need be calculated per clock cycle. Again, this allows a factor N reduction in supply voltage and, considering the constant C and f terms, the dynamic power consumption is reduced by N^2. Thus, both concurrency techniques – pipelining and parallelism – result in a first-order quadratic reduction in power consumption.

Bit-serial processing was another architectural style given consideration in this study. Bit-serial processors do, in some sense, exhibit concurrency. Rather than time-domain concurrency, however, they display what can be called spatial concurrency. There is an important distinction to be made between time-domain concurrency and spatial-domain concurrency. Pipelining and parallelism are examples of time-concurrent techniques. In other words, at a given time, computations are occurring concurrently across a large number of spatially distinct processing units. In the case of pipelining, these units are the N stages

of the pipeline, while in the parallel scenario they are the N identical processors. In contrast, spatial concurrency refers to architectures that perform computations using the *same* piece of hardware (i.e. *spatial concurrency*) over a large number of clock cycles. This is commonly referred to as time-shared hardware, and bit-serial processing is a prime example.

Unfortunately, this form of processing has no advantages for low-power circuits. Consider, for example, a function, B, that operates on a word of N bits. In bit-serial form, this function would be implemented by a single-bit processor, which would clearly possess $1/Nth$ the capacitance of the bit-parallel case. Unfortunately, in order to maintain throughput, the bit-serial processor must be clocked at a frequency N times higher than the bit-parallel case. As in the pipelined and parallel scenarios, then, the effects of the capacitive and frequency components in the dynamic power equation cancel out. Unlike, those cases, however, the bit-serial processor cannot operate at a reduced voltage supply. That is, due to the increased clocking frequency, the time allotted for the calculation of a single result bit has not increased over the bit-parallel case; therefore, no V_{dd} reduction and, subsequently, no power reduction are possible. Clearly then, it is time-*concurrent* rather than time-*sharing* architectures that are attractive for low-power systems.

The preceding analysis has considered only the first order effects of parallelism on power consumption. There are second order effects that detract from the power savings achieved by using concurrent hardware. In a pipelined processor, for example, the pipeline registers represent an overhead in both power and area that cannot be ignored. As the pipeline depth increases, the hardware (capacitance) associated with the pipeline registers approaches that of the actual processor stages. At that point, further pipelining becomes unattractive.

Hardware parallelism also has its disadvantages. For instance, complete hardware duplication entails a severe area penalty. In addition, associated with these distributed processors is hardware and interconnect overhead related to signal distribution at the processor inputs and signal merging at the outputs. As in the pipelined case, these contribute to increased power consumption and tend to limit the utility of excessive parallelism. Even before considering this overhead, the area requirements of full parallelism can be a limiting factor; however, the advent of MCM's should help to minimize this concern.

Arithmetic Format

Another architectural consideration is arithmetic format. High quality, low bit-rate speech coding places severe computational requirements on the datapath of its hardware platform. In particular, squaring and multiply-accumulate (MAC) instructions are quite common (as in many DSP applications). Thus, dynamic range in the datapath becomes an important issue. Among the several arithmetic formats available are fixed-point, floating-point, and block floating-point. Each of these has different implications for low-power performance.

Fixed-point offers the minimum hardware requirements and power consumption, among the three options. Unfortunately, it also suffers the most from dynamic range difficulties. Algorithmic scaling offers some relief from this problem, however, it must be incorporated into the processor μ-code and thus has some runtime overhead. Floating-point, in contrast, alleviates the dynamic range issue at the expense of extensive hardware additions. This increased hardware leads to correspondingly higher capacitances and, as a result, higher power consumption. Block floating-point may offer the best compromise between the two techniques. At the expense of a small amount of additional hardware (and power), it accomplishes the scaling process (for the most part) automatically. Thus, it achieves some compromise between dynamic range and hardware overhead.

2.2.7 Algorithms

These architectural considerations suggest several algorithmic criteria for low-power. Although further developed in section 3.3, the key issues are presented now in an introductory fashion.

The technique of scaling supply voltage is bounded not only by threshold voltage considerations, but also by real-time requirements. Specifically, as voltage levels are scaled down, propagation delays increase proportionally. At some point, the requirement for real-time operation sets a limit on supply scaling. In other words, the maximum number of sequential operations that can be executed in one coding cycle relates directly to the operating voltage.

As this relationship suggests, reducing the required number of sequential operations per coding cycle translates directly to a lower supply voltage and, ultimately, to reduced power consumption. Significantly, the number of "sequential" operations and not

the total number of operations per cycle limits power supply reduction. Independent operations may be performed concurrently and, thus, do not serve to increase the time required to complete a given coding cycle. Of course, this assumes that the hardware to efficiently handle this concurrency is in place and, indeed, that was the focus of the previous section.

Parallelizability is but one algorithmic consideration. As suggested previously, suitability for fixed-point or block floating-point implementation can also be important. Again, this is intimately related to the dynamic range issue that can be so troublesome in DSP applications such as speech coding. The issue is not purely one of hardware. Certain algorithms and filter structures are more suitable for fixed-point implementations than others. For example, the auto-regressive lattice synthesis filters associated with many speech coding techniques are highly advantageous for fixed-point implementations. This stems from the property that the tap coefficients (reflection coefficients) for these stable filters are less than one in magnitude. This allows greatly simplified quantization schemes and minimizes fixed-point implementation difficulties. Although not always the structure of choice for various other reasons, the lattice example illustrates that algorithmic considerations can affect power-related issues.

2.2.8 Power Management

At a slightly higher level, well-known power-management techniques can be applied to reduce energy consumption. Specifically, processing blocks need only be clocked and powered when they are performing some computation. During idle periods these processors may be powered down (assuming state information is properly preserved). In this way, large power savings are possible (depending on the utilization level of the particular processing unit).

For speech coding applications, in particular, power-down techniques can provide significant power savings. In most speech coding environments, long periods of relative silence are observed. A speech coder that monitors input energy levels can take advantage of this fact by shutting down during periods of inactivity. Moreover, for a full-duplex (transmitting and receiving) coder, only one of the processes, either the coder or the decoder will be active at a time. This can provide an additional measure of power reduction.

2.3 Summary of Low-Power Methodology

To summarize, the proposed low-power methodology involves techniques ranging from the circuit to the architectural level of design. At the highest level, the supply voltage is scaled down from 5 V to some optimum technology-dependent voltage (roughly 1.5 V for the 2 μm MOSIS CMOS process)[4]. In order to compensate for linearly increasing circuit delays, some form of concurrent computational technique, such as parallelism or pipelining is employed. The result is a power reduction, which to the first order goes as the square of the supply reduction.

Beyond this, circuit techniques such as reduced clock line and I/O pad voltage swings, with accompanying sense amp circuitry, can be applied to further reduce power consumption. In addition, judicious choice of logic style can allow additional power reduction, as can technology scaling and advanced packaging techniques. Finally, proper selection and optimization of candidate algorithms to achieve maximum concurrency offers equally significant gains in power conservation. Combinations of these schemes, coordinated through an intelligent power management methodology, can help to achieve maximum overall power savings.

Of course, if some of these techniques are unavailable or undesirable for a particular design effort, the voltage scaling alone will provide large power savings. As stated previously, this is one of the advantages of the parallelism technique: no additional circuit or technology tricks are required. They are, however, extremely desirable and should be employed when possible.

For the most part, this methodology is not specific to speech coding. This was one of the key objectives of this research. As a result, the strategies discussed in this report can be applied equally well to a wide range of DSP applications and algorithms.

Chapter 3

Speech Coding

The previous chapter presented techniques for achieving low-power VLSI implementations of DSP algorithms. Furthermore, it contained important background material on topics that are of critical concern to low-power circuits. The current chapter provides additional background information, this time on speech processing, that will be central to understanding the detailed analyses of the coding algorithms to be presented in chapter 4.

3.1 Speech Processing Background

3.1.1 Human Speech Generation

In order to facilitate an understanding of modern speech coding methods, an understanding of the speech signal and the mechanism by which it is generated is essential. Figure 3.1 shows a model of the human speech generation system. During speech production, the diaphragm forces air out of the lungs and into the trachea. This flow of air passes by the vocal cords resulting in one of two possible scenarios. In the first, the vocal cords are tense, and the rush of air from the lungs causes them to vibrate at some pitch frequency, thereby, modulating the flow of air into a series of discrete bursts. This corresponds to a *voiced* sound such as a vowel. In the second scenario, the vocal cords are relaxed and the air passes by relatively unaffected. This *unvoiced* mode of speech corresponds to consonants such as *s*, *f*, and *p*.

Continuing through figure 3.1, the stream of air proceeds past the larynx and passes through a series of resonant cavities – the pharynx cavity, the mouth cavity, and the

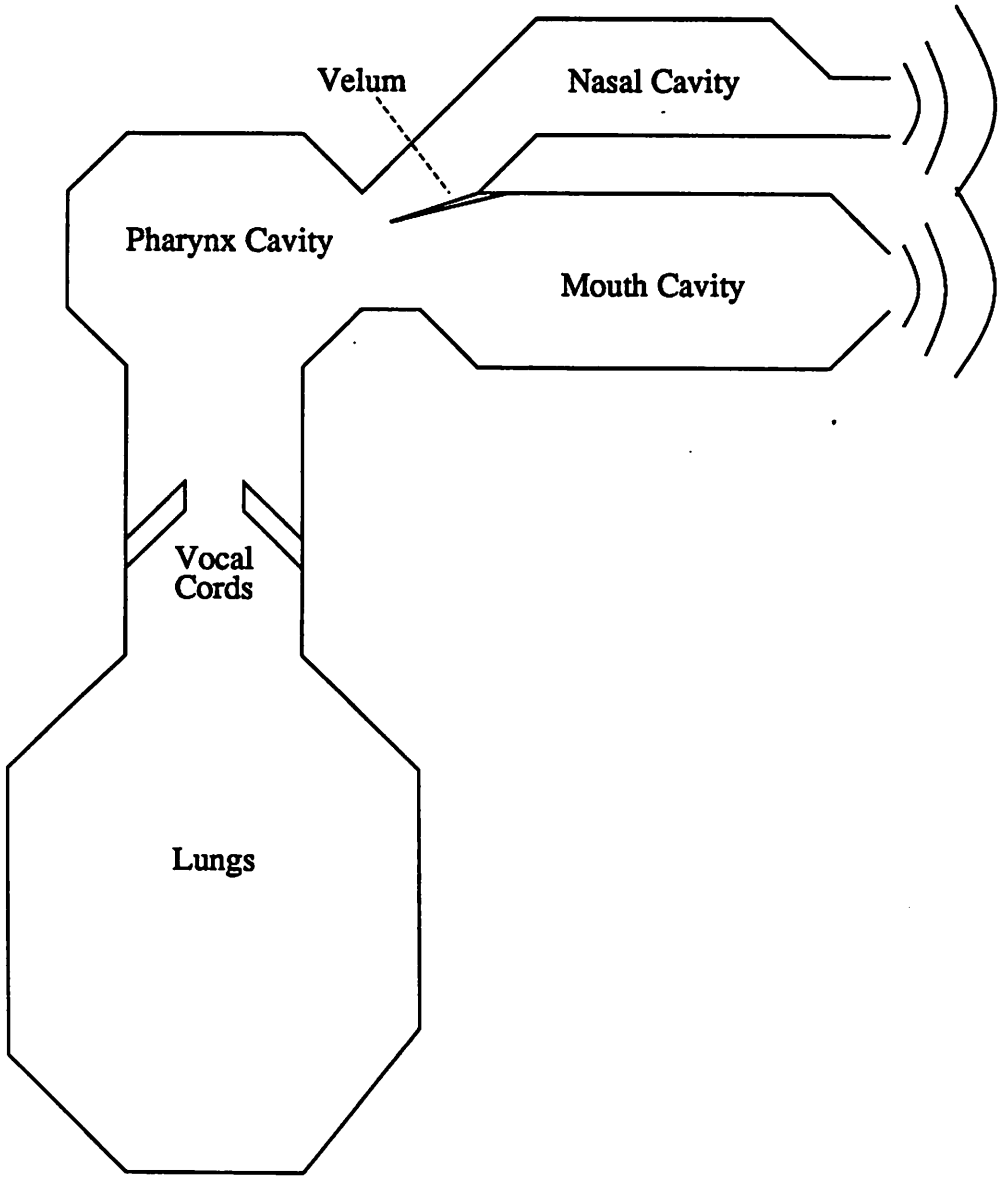


Figure 3.1: Human Speech Generation (after Flanagan)

nasal cavity, in particular. The flow of air, either modulated or unimpeded, excites these cavities and causes them to resonate. The frequencies of these resonations is determined by the size and shape of the cavities. This, in turn, depends upon the positions of various vocal tract components including the jaw, the velum, the tongue, and the lips. As an example, consider the production of the so-called *nasal* phonemes such as *n*. During a nasal utterance, the velum allows the unimpeded flow of air into the nasal cavity as well as the mouth. Thus, both cavities respond to the air flow and sound is radiated not only from the lips, but also from the nostrils.

As a further example illustrating the importance of the vocal articulators, consider the generation of the unvoiced consonants *s* and *p*. In the production of an *s*, the velum is closed and the steady flow of air from the lungs directly enters the mouth. Were the lips and tongue positioned so as to open the mouth cavity, the air would pass through and radiate from the lips forming a sound similar to that of an *h*; however, the formation of an *s* requires that the tongue be pressed to the roof of the mouth. Thus, the flow of air is impeded and it passes by the tongue and lips as a turbulent stream. This is recognized as the "hissing" quality of the *s* phoneme. When forming the phoneme *p*, the lips rather than the tongue impede the air flow. Specifically, the lips cause a brief, but total, closure of the vocal tract allowing a buildup of pressure, which is released as a transient burst as the lips are opened. The result if the "popping" sound characteristic of this phoneme.

The shape of the vocal cavities determines the resonances of the vocal tract filter. These resonant frequencies are called formants and are largely responsible for the character of the speech generated. In other words, for voiced speech, altering the formant frequencies can change the generated phoneme from, say, $\bar{a}$ to some other voiced phoneme such as $\bar{e}$. Thus, the information contained in the formant frequencies is closely related to the particular phoneme uttered. Indeed, extraction of these formants can be (and has been) used for rudimentary speech recognition.

3.1.2 The Speech Waveform

The previous discussion suggests several attributes of the speech waveform. For example, realizing that (voiced) speech generation is excited by the modulated puffs of air passing the vocal cords, a speech signal containing a periodic component at the frequency of the vocal cord vibrations would be expected. In the time-domain, this signal component

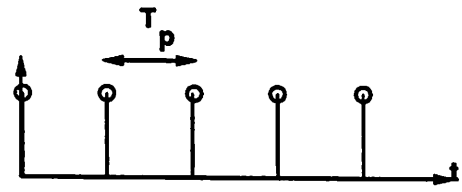


Figure 3.2: Pitch Signal in Time-Domain

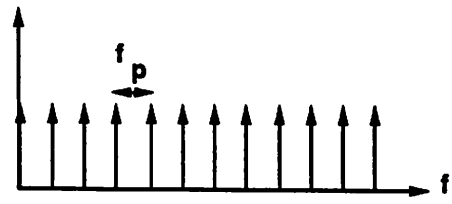


Figure 3.3: Pitch Signal in Frequency-Domain

could be thought of as a discrete, periodic series of pulses recurring at an interval referred to as the pitch period, T_p (see figure 3.2). In the frequency-domain, this corresponds to a power spectrum with components at the pitch frequency, f_p, and all of its harmonics (see figure 3.3). This pitch-periodic signal is, however, but one component of the overall speech signal.

In addition to the pitch excitation, the response of the vocal tract filter must be considered. This component lends itself most directly to analysis in the frequency-domain. Remember, the effect of the resonant vocal cavities is to introduce formant frequencies into the speech signal. These resonances appear as peaks in the speech power spectrum. In general, three to five formant frequencies can adequately describe the vocal tract frequency response. This number corresponds roughly to the number of resonant cavities present in the human vocal tract. Thus, the cavities of the vocal tract filter contribute a frequency-domain envelope of the form shown in figure 3.4.

Combining excitation and envelope, an actual frequency-domain power spectrum of the voiced phoneme $\bar{a}$ is shown in figure 3.5. Notice the overall spectral envelope with roughly three formant frequencies representing the response of the vocal tract filter. Furthermore, the periodic fine-structure at the 125 Hz fundamental and its harmonics is the direct contribution of the pitch periodicity of the voiced vowel. In the time-domain (figure 3.6), the periodic pitch excitation is again noticeable at approximately 8 ms (=1/125 Hz) intervals. Each impulse-like pitch excitation is followed by an oscillating response corresponding

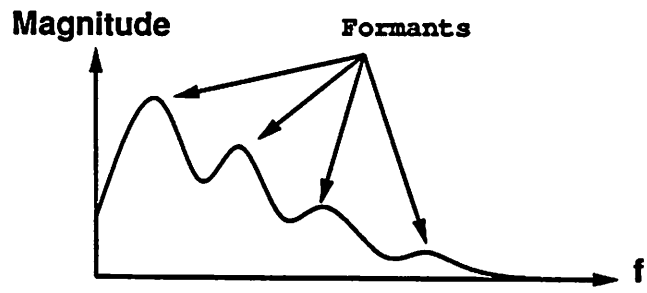


Figure 3.4: Spectral Envelope of Speech Signal Due to Vocal Tract Filter

directly to the impulse-response of the vocal tract filter. As this response begins to die out, it is excited anew by the arrival of the next pitch impulse.

Of particular interest to many communications applications is the bandwidth required to adequately represent the speech waveform. As seen in figure 3.5, the power spectrum of the speech signal is largely band-limited to frequencies below 4 Khz. Indeed, the sampling rate for contemporary digital telephone communications networks is 8 Khz, resulting in a maximum representable frequency component of 4 Khz. Telephone network anti-aliasing sending filters, however, further reduce this maximum to approximately 3.6 Khz (-3 dB point). As a matter of common experience, the resulting speech quality is quite satisfactory for most verbal communication purposes. Indeed, the input to the majority of existing speech coders is 8 Khz "toll-quality" speech and, thus, for these coders it represents the ultimate achievable quality for the compressed and decoded speech signal.

3.1.3 A Model for Speech Generation

The characteristics of the speech signal and its production suggest a straightforward model for human speech generation. The model consists of two components: an excitation source and a vocal tract filter as illustrated in figure 3.7[8]. The excitation source is selected from two possible generators. The first is simply a periodic impulse train which models the voiced speech excitation signal. The unvoiced speech excitation, in contrast, is modeled by a "white"-noise source, which corresponds physically to the turbulent air stream that is characteristic of unvoiced speech.

As suggested in section 3.1.2, the effect of the vocal tract can be simulated by a filter capable of modeling three to five resonant frequencies. Since the shape and size of the vocal cavities vary with time in the production of continuous speech, the vocal tract

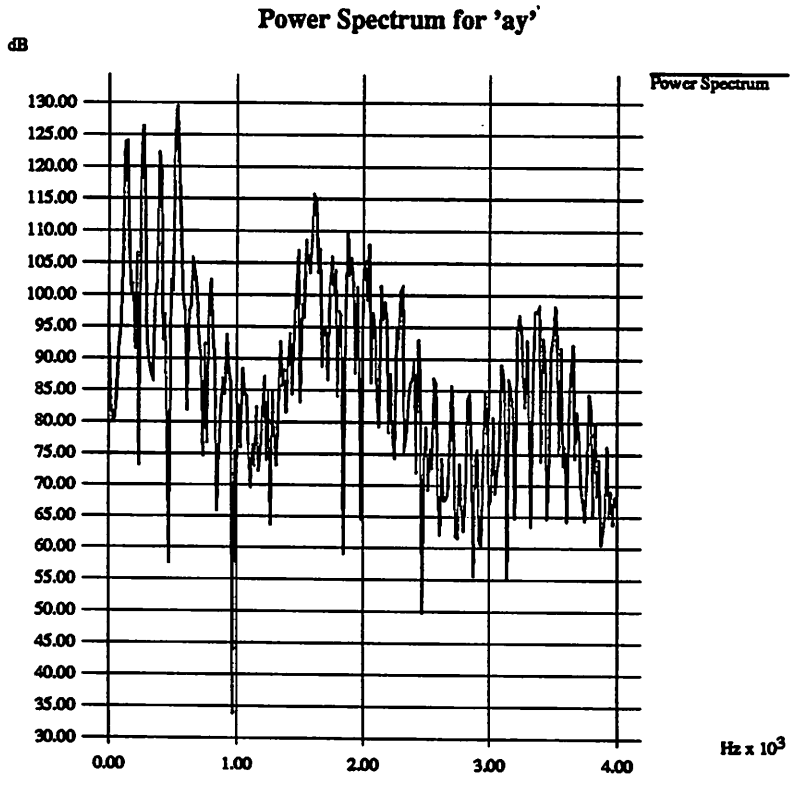


Figure 3.5: Power Spectrum of Phoneme $\bar{a}$

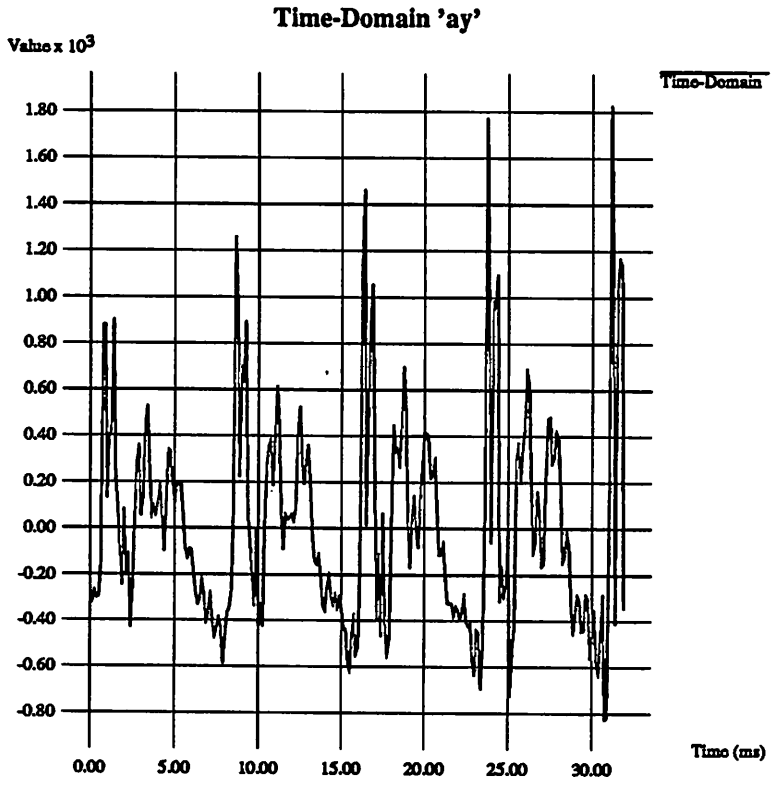


Figure 3.6: Time-Domain $\bar{a}$

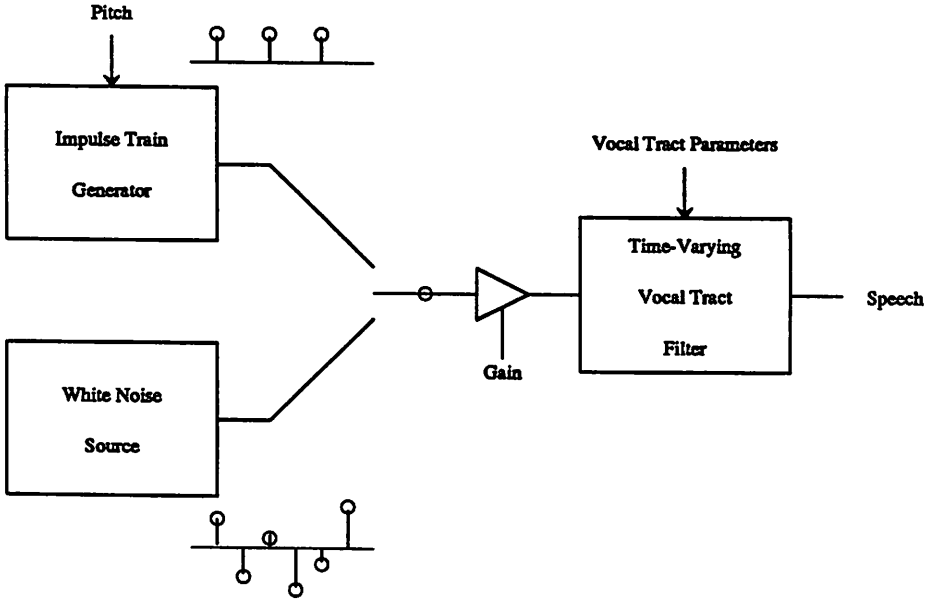


Figure 3.7: Simplified Model for Human Speech Generation (after Rabiner)

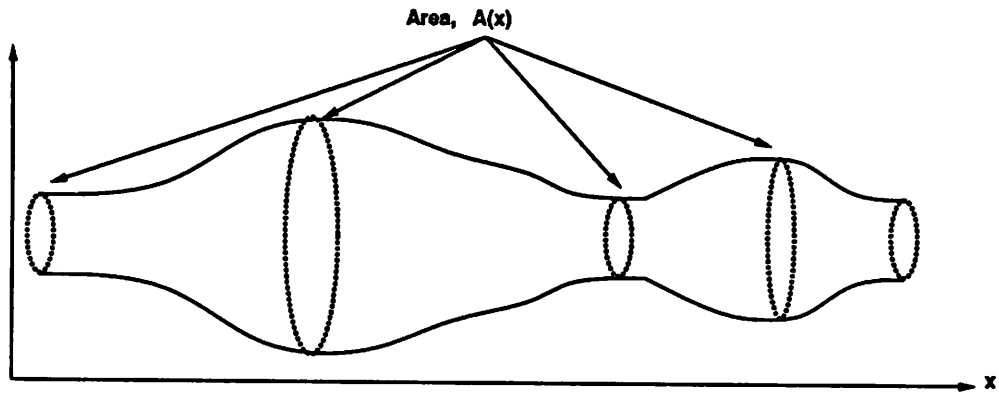


Figure 3.8: Acoustical Tube Model of Vocal Tract Filter

filter model must, of necessity, be time-varying. The detailed nature of this filter can be derived by comparing the vocal tract to a cylindrical acoustical tube of non-uniform cross-sectional area (see figure 3.8). The behavior of such a system has been examined extensively [7, 8]. Using the mass conservation and momentum equations we can derive the theoretical response of the system:

$$-\frac{\partial P}{\partial T} = \frac{\rho v^2}{A(x)}\frac{\partial U}{\partial x} \tag{3.1}$$

$$-\frac{\partial U}{\partial T} = \frac{A(x)}{\rho}\frac{\partial P}{\partial x} \tag{3.2}$$

In these equations, x is the distance along the vocal tract, U is the volume-velocity, P is the sound pressure, $A(x)$ is the cross-sectional area at x, ρ is the density of air, and v is the velocity of sound. These equations can be solved for the approximate form of the vocal tract transfer function, $H(z)$ [8]:

$$|H(z)| \approx \frac{G}{1 - \sum_{k=1}^{N} a_k z^{-i}} \tag{3.3}$$

where the relationship becomes exact as N goes to infinity.

This transfer function has the form of an all-pole filter. Thus, the vocal tract filter can be modeled as a time-varying all-pole filter of order N. The determination of the order, N, relates to the number of formant frequencies to be modeled. Two complex-conjugate poles are required to model a single formant frequency. Therefore, it is not surprising that a ten-pole filter quite effectively models the response of the vocal tract (which is primarily influenced by its first five formants). A pole-zero plot of a vocal tract filter model is shown

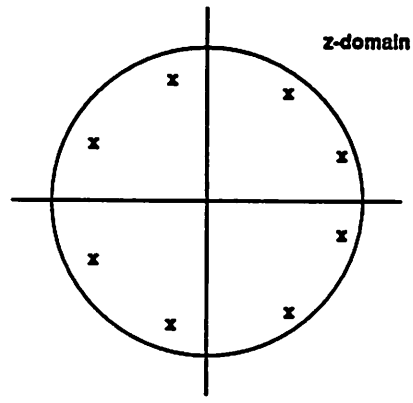


Figure 3.9: Pole-Zero Plot for Vocal Tract Filter

in figure 3.9. The spectral envelope produced by this model would be quite similar to that of figure 3.4.

The applicability of the model is, of course, not universal. For example, the model assumes the separability and independence of the excitation source and vocal tract filter. This assumption does not strictly hold, especially in the case of the plosives (p, for example). For most phonemes, however, the condition is sufficiently satisfied and it need not be a point of concern. Another restriction stems from the all-pole nature of the vocal tract model. The all-pole filter models vocal tract resonances well, however, the anti-resonances introduced by the nasal cavity are best modeled by zeros. Very high quality can still be achieved, however, with the classical all-pole vocal tract filter. Furthermore, its relative simplicity with respect to a full pole-zero representation makes the all-pole filter far more useful than the pole-zero model.

3.2 Speech Coding Methods

Previous sections have dealt at length with the background information required to understand the speech signal and how it can be modeled. This section now delves further into the issue of speech processing and, specifically, into methods and algorithms for speech coding. Speech coding refers to techniques that, taking the speech signal as input, reduce the amount of information required to describe the waveform.

Consider, for example, a "toll-quality" digital speech waveform. As discussed in section 3.1.2, this corresponds to an analog speech waveform sampled at 8 Khz. Further

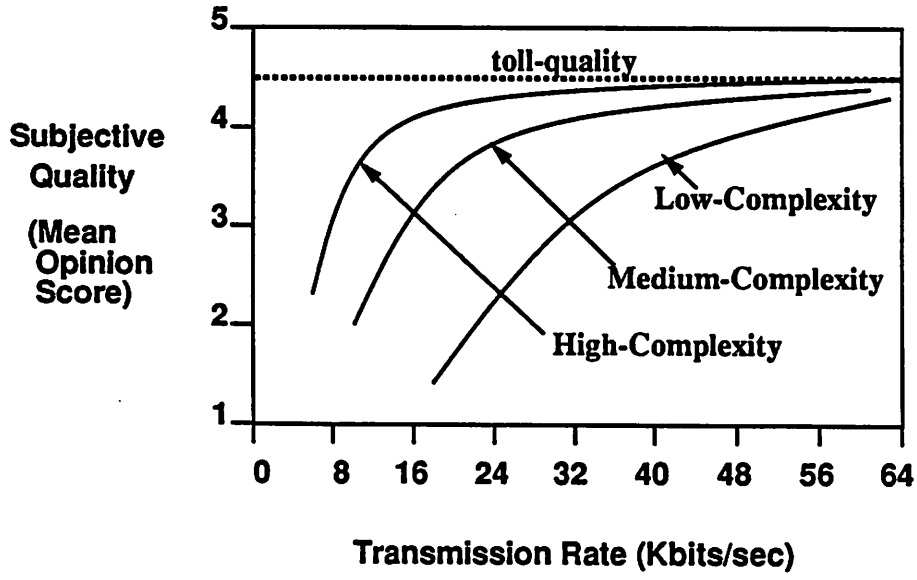


Figure 3.10: Coder Complexity versus Subjective Quality (after Noll)

assume that the speech samples require 8-bits of storage each (corresponding to the American telephony standard 8-bit μ-law). For this case, the data bandwidth required to store or transmit the speech signal is 64 Kbits/sec. The purpose of a speech codec (coder/decoder) is to reduce this bandwidth requirement. Taking advantage of redundant information in the speech signal, the coder produces a reduced bandwidth data stream that represents the input speech signal. This channel information is then stored for later decoding or is transmitted directly to the decoder, which reverses the process to retrieve the original speech waveform (or an approximation).

The compression ratios available from contemporary coding techniques vary widely as does the quality of the decoded speech signal. Figure 3.10 shows a plot of subjective speech quality versus transmission rate for coders of varying complexity [9]. In this graph, complexity is defined by a criterion such as the number of multiply-accumulate operations required per input sample. As a point of reference, toll-quality telephony achieves a Mean Opinion Score (MOS) of 4.5 at 64 Kbits/sec.

Basically, coders can be subdivided into two categories: waveform coders and parametric coders. Waveform coders perform time-domain sample-by-sample coding of the speech signal. That is, these coders do not use prior knowledge of the nature of speech in order to reduce the information bandwidth requirements. Pulse Code Modulation (PCM)

is the most common example of this type of coding. PCM is among the simplest coding methods and, as such, it is also among the most prevalent. Quite simply, PCM coding corresponds to sampling the analog waveform and quantizing sample amplitudes. Digital telephony networks employ PCM with logarithmic (μ-law) quantizing schemes to achieve their 64 Kbit/sec transmission rates.

A slightly more complicated scheme known as differential pulse code modulation (DPCM) can be employed to reduce the transmission rate for toll-quality speech. In DPCM, differences between successive sample amplitudes, rather than their absolute values, are encoded. Another member of this family, adaptive differential pulse code modulation (ADPCM) performs the identical function with the addition of adaptive quantization. These coders achieve subjective qualities approaching 64 Kbit/sec PCM at bit-rates of 32 Kbits/sec.

Waveform coders are, however, only one class of coders; the second class, parametric coders, are equally important. Parametric speech coders utilize a priori information about the properties of the speech signal to reduce transmission bandwidth. Most parametric speech coders rely on linear predictive coding (LPC) in one form or another to achieve this data reduction. Basically, these coding methods take into consideration the fact that speech is a highly redundant signal. That is to say, speech samples in close proximity are highly correlated. By modeling this correlation, LPC coders can extract a small number of parameters that describe the speech waveform. These then, rather than the quantized signal itself, are transmitted and used to synthesize decoded speech at the receiver. Included in this important class of coders are residual-excited linear predictive coders (RELP), self-excited linear predictive coders (SELP), and code-excited linear predictive coders (CELP). The latter has proven extremely useful for low bit-rate coding and will be discussed later in much more detail.

3.3 Criteria for Candidate Algorithms

Clearly then, there is a wide variety of algorithms available for the implementation of speech codecs. Selection of a particular coding style depends on the unique criteria of a particular application. For example, several factors were considered in selecting candidate algorithms for the PCS speech coder. Most importantly, the search was constrained to high subjective-quality coders (approaching toll-quality). Subjective quality, unfortunately,

is difficult to define and to measure. Several different measures exist and they include the Mean Opinion Score (MOS), the Diagnostic Rhyme Test (DRT), and the Diagnostic Acceptability Measure (DAM).

A second requirement – antagonistic to the first – was a compression ratio of 4:1 or more. Beginning with a 64 Kbit/sec toll-quality input stream, this requirement corresponds to a channel stream of 16 Kbits/sec or less. Furthermore, channel streams of 9.6 Kbits/sec or less were even more desirable if high quality could be maintained. Unfortunately, as figure 3.10 demonstrates, the combination of high quality and low bit-rates dictates the use of high-complexity coders. Though unfortunate from an implementation standpoint, the high-complexity of the candidate coding algorithms does provide a forum for thoroughly testing the proposed low-power strategies.

Related to coder complexity is the concept of encoding delay. Encoding delay describes the length of the waveform segment that must be analyzed in order to utilize waveform redundancies to achieve compression. Thus, encoding delay is directly related to memory storage requirements, which is one measure of implementation complexity.

Another issue related to encoding delay is that of real-time operation. Long coder-decoder delays can be quite a hindrance to real-time communication. Assuming appropriate echo control, however, total end-to-end delays on the order of 100-200 ms are acceptable. Implied in this discussion is the requirement that the codec hardware operate in real-time. For some systems, such as voice-mail, where speech is stored rather than decoded immediately, a real-time codec is not a necessity; however, the main application for this speech coder is a real-time communication system. This places some restrictions on coder complexity since there is a limit to the number of operations per sample that can be implemented in real-time.

An additional criterion for the candidate algorithms was suitability for low-power implementation. As shown in the preceding sections, this corresponds, in some sense, to suitability for parallel implementation. The highly parallel nature of the proposed architecture is most suited to an algorithm with a good deal of inherent concurrency. Remember, for a parallel architecture, it is not the total number of operations per sample that dictates the real-time requirements, but rather the total number of sequential operations per sample.

Another condition related to low-power operation is the suitability of the algorithm for fixed-point or block floating-point implementation. As stated in section 2.2.6, full-scale floating-point arithmetic is extremely costly in terms of power. It is, therefore, desirable

to implement the codec on a fixed-point or block floating-point processor, if possible. Unfortunately, fixed-point quantization and dynamic range issues can be extremely important in determining overall coding quality. Moreover, algorithm designers rarely delve into such analyses when defining their algorithm; instead, they assume floating point platforms such as commercially available DSP chips (i.e. AT&T DSP32c).

A final desirable property for the candidate algorithms is compatibility with existing coding standards. Of course, this is by far the least critical concern; however, in order to promote maximum utility of the speech coder, compatibility with some well-accepted standard is a desirable situation.

3.4 Selection of Candidate Coding Class

The prerequisites for the candidate coding algorithms are numerous. In general, the CELP class of coders, however, meets these requirements admirably. Actually, the high quality at low bit-rates criterion in itself narrows the field quite significantly. At bit-rates at and below 16 Kbits/sec, waveform coders are severely disadvantaged and yield very low quality speech. In this realm, parametric LPC-based coders begin to demonstrate their coding capabilities. Straight LPC coding, however, although intelligible at low bit-rates, can not be considered "high quality." The decoded speech has an unmistakable "synthesized" character that we are trying to avoid. Unlike straight LPC, however, CELP performs very well at bit-rates down to 4.8 Kbits/sec. Thus, the high quality, low bit-rate requirements alone were enough to suggest a CELP coding scheme.

The third criterion, minimization of the encoding delay, is a difficult problem for high quality, low bit-rate speech. Inherently, achieving high quality at low bit-rates requires that the coder make use of speech redundancies; however, this requires the coder to build a history of the input speech waveform, which can then be analyzed for correlations. This implies a coding delay of at least a frame (approximately 20 ms). Furthermore, channel delays can themselves be on the order of 100 ms. Fortunately, however, with adequate echo control, CELP encoding delays are not a hindrance to real-time communication.

Real-time considerations have long been another drawback for CELP coders. CELP coders are based upon an analysis-by-synthesis methodology. In other words, several candidate speech waveforms are synthesized, and the parameters corresponding to least distorted synthesized signal are transmitted to the decoder. Such a methodology is, clearly, computa-

tionally intensive. To some extent, however, this is the price that must be paid for very high quality speech coding. Furthermore, the heavy computational requirements will provide an extreme test of the low-power techniques espoused in this report.

Suitability for low-power implementation was an additional prerequisite. For low-power, parallel architectures, this criterion is equivalent to requiring a highly concurrent algorithm. CELP coders meet this requirement admirably. Their analysis-by-synthesis approach requires the synthesis of numerous independent speech waveforms. For the classic CELP coder, these synthesis operations can be performed concurrently. Since this "code-book search" traditionally embodies the majority of the algorithmic complexity, CELP is well-suited for low-power implementation.

Aside from concurrency, similarity to more general DSP algorithms is another desirable quality. Again, the main component of the CELP algorithm is the synthesis of several candidate speech waveforms. This synthesis is performed by a linear, time-varying all-pole filter. This multiply-accumulate intensive operation is present in many, if not all, DSP algorithms. Thus, optimizing a low-power architecture based upon this operation will go a long way towards defining a more general-purpose low-power DSP architecture.

All of these considerations led us to restrict candidate algorithms to the CELP family of coders. From this class, three algorithms were selected as candidates; they were compared on the basis of all the previous criteria. In addition, suitability for fixed-point or block floating-point implementation was evaluated, as was the issue of compatibility with existing systems and standards. The three algorithms are Motorola's 8.0 Kbit/sec VSELP algorithm [10], the Department of Defense's (DoD) 4.8/9.6 Kbit/sec CELP algorithm [11], and AT&T's 16 Kbit/sec LD-CELP algorithm [12]. The three coders are very high quality, all achieving Mean Opinion Scores greater than 4.0 (4.5 corresponds to toll-quality). Since they all meet the high quality coding criterion, it was not considered in final algorithm selection.

3.5 Overview of LPC and CELP

3.5.1 Linear Predictive Coding

Before delving into an analysis of these three algorithms, an explanation of the concept of linear prediction and, specifically, code-excited linear prediction is desirable.

The use of linear prediction in speech coding dates back to the late 1960's [13] and is based on an extremely simple concept. In traditional LPC, bandwidth compression is achieved by utilizing the inherent correlation present in the speech signal. Basically, a small set of parameters containing information such as the pitch of the input speech and the coefficients of the vocal tract filter are extracted from a frame of input speech. These parameters are then transmitted to the decoder and used to resynthesize a time-domain approximation to the original speech.

The method is based on the concept of linear prediction. This notion suggests that speech can be modeled as an auto-regressive (AR) process. In other words, it assumes that a sample of speech can be represented as a linear combination of N_p previous speech samples, where N_p is the prediction order. Expressed mathematically,

$$s_p(n) = a_1 s(n-1) + a_2 s(n-2) + \cdots + a_{N_p} s(n-N_p) \tag{3.4}$$

where $s_p(n)$ is the predicted value of the sample $s(n)$. The next step is to define an error signal,

$$e(n) = s(n) - s_p(n) \tag{3.5}$$

By appropriate selection of the LPC coefficients, a_i, it is, of course, possible to make the error signal identically zero for any particular sample, $s(n)$. If this were done, however, no information reduction would be possible since new LPC coefficients would have to be transmitted for every sample. The goal, then, is to optimize the LPC coefficients over a wide range of, say, N_A input samples, where N_A is the analysis interval. This optimization is accomplished by minimizing the energy contained in the error signal over the analysis interval. Defining the energy,

$$E = \sum_{n=0}^{N_A-1} e^2(n) \tag{3.6}$$

To minimize energy with respect to the LPC coefficients we set its partial derivatives to zero:

$$\frac{\partial E}{\partial a_i} = 0 \qquad \text{for } i = 1, \ldots, N_p \tag{3.7}$$

This results in a system of N_p equations with N_p unknowns. The equations are known as the autocorrelation normal equations and in matrix form are given by

$$[R(i-k)][a_i] = [R(k)] \qquad \text{for } i, k = 1, \ldots, N_p \tag{3.8}$$

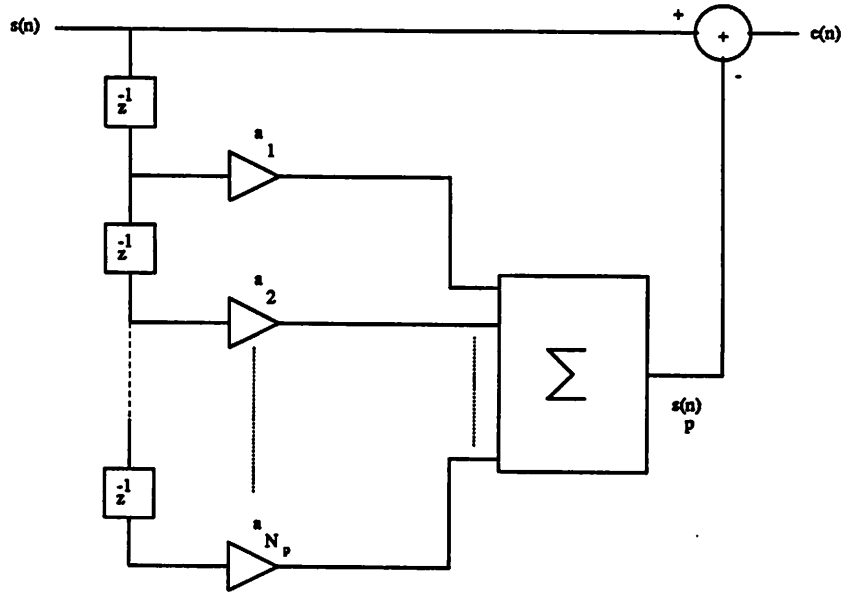


Figure 3.11: LPC Analysis Filter

where $R(k)$ is the short-term autocorrelation function,

$$R(k) = \sum_{n=0}^{N_A-1} s(n)s(n-k) \tag{3.9}$$

In general, solving such a system of equations would be a difficult task; however, the autocorrelation matrix, $[R(i-k)]$, has some useful properties. First, it's symmetric; this reduces not only computational requirements, but also storage requirements since the matrix is uniquely determined by its upper triangular component. Furthermore, the matrix is Toeplitz (the elements on each diagonal are equal). For Toeplitz matrices, a particularly nice method known as the Levinson-Durbin recursion exists for solving system 3.8.

Notice that the previous derivation inherently depends on the stationarity of the input speech signal. Although the speech signal is not stationary in general, short segments of speech can be approximated as stationary. This sets an upper bound on the size of the analysis interval and requires that the LPC coefficients be updated regularly.

Using the coefficients defined by the normal equations, we can consider two interesting filter configurations. The first is the LPC analysis filter depicted in figure 3.11. This filter is simply a realization of equations 3.4 and 3.5 and is given in the z-domain by

$$A(z) = 1 - \sum_{i=1}^{N_p} a_i z^{-i} \tag{3.10}$$

Since the purpose of LPC analysis is to fully describe the correlation in the input signal, the optimal output of the analysis filter would be white noise (i.e. a signal with no correlation between its samples). For such a signal, each new sample gives new information about the stochastic process. For that reason, it is called an innovations process [14]. The innovations process wastes no energy describing redundant (correlated) information. For this ideal scenario, the power spectrum of the input speech waveform is completely described by the LPC coefficients, and these coefficients would be all that was required at the receiver to synthesize the decoded speech.

Actually, Wiener filter theory demonstrates that for large enough prediction orders, N_p, the residual signal, $e(n)$, will, indeed, be white. Computational as well as stationarity considerations, however, limit the the prediction order. In practice, a prediction order of ten is quite common. Assuming this value for the prediction order, we can study the actual form of the residual signal, $e(n)$. While a tenth order predictor can remove the short-term correlations due to the vocal tract response following a pitch excitation, it cannot predict the long-term pitch correlations present in the input speech. Therefore, for voiced speech, we expect $e(n)$ to closely resemble an impulse train, with period equal to the pitch period, emersed in low-energy white noise. For unvoiced speech, there is no long-term pitch correlation, and we expect $e(n)$ to be nearly white.

Considering the inverse of the analysis filter (the synthesis filter) will give us further insight into the situation. The synthesis filter depicted in figure 3.12 can be derived from equations 3.4 and 3.5 in the z-domain is given by

$$\frac{1}{A(z)} = \frac{S(z)}{E(z)} = \frac{1}{1 - \sum_{i=1}^{N_p} a_i z^{-i}} \tag{3.11}$$

Notice that the form of $\frac{1}{A(z)}$ is that of an all-pole filter. In the figure, the residual, $e(n)$, is used to excite the all-pole synthesis filter, $\frac{1}{A(z)}$, to produce the original input speech. Ideally, if $e(n)$ as well as the LPC coefficients were available at the decoder, the input speech could be reproduced exactly; however, in order to reduce transmission requirements we can take advantage of a priori knowledge of the form of the residual, $e(n)$. In the voiced case, we can approximate $e(n)$ at the decoder by an impulse train and, thus, only the pitch period need be transmitted to the decoder. For the unvoiced case, $e(n)$ is approximately white, and a white noise generator at the decoder can be used to excite the synthesis filter.

It is of interest that the preceding discussion has provided a justification for the original speech generation model of section 3.1.3. Using that model as a guide, we see that

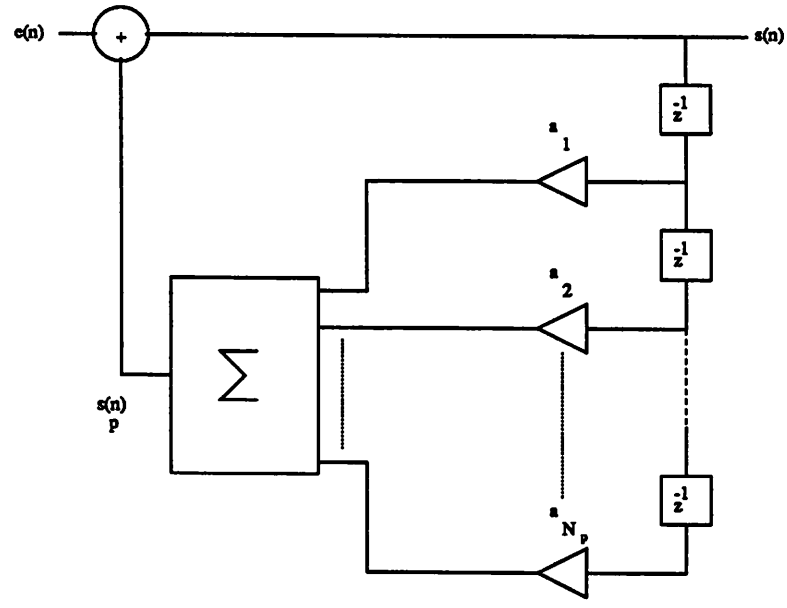


Figure 3.12: LPC Synthesis Filter

the residual signal, $e(n)$, can be thought of as the excitation source in the model. Moreover, the time-varying all-pole vocal tract filter is implemented by $\frac{1}{A(z)}$. This realization also justifies the choice of ten as the prediction order since this value corresponds to two complex-conjugate poles for each of five formants.

3.5.2 Code-Excited Linear Prediction

Ideally, we would like to excite the synthesis filter in the decoder with the actual residual signal, $e(n)$. Code-Excited Linear Prediction (CELP) is a variation of LPC that attempts to achieve this goal. The CELP concept is similar to LPC with the addition that a vector quantized version of the excitation, $e(n)$, is transmitted to the decoder and is then used to excite the synthesis filter there (see figure 3.13) [15]. The vector quantization process is achieved through an analysis-by-synthesis approach. Specifically, a codebook of vector quantized excitation signals is kept in both the coder and decoder. In the coder, each possible excitation in the codebook is passed through an LPC synthesis filter and the synthesized signal is compared to the input speech. The codevector that produces a synthesized speech segment that is perceptually closest to the input speech is selected, and an index to this vector quantized version of $e(n)$ is transmitted along with the LPC coefficients to the decoder. In the decoder, this excitation is selected out of an identical

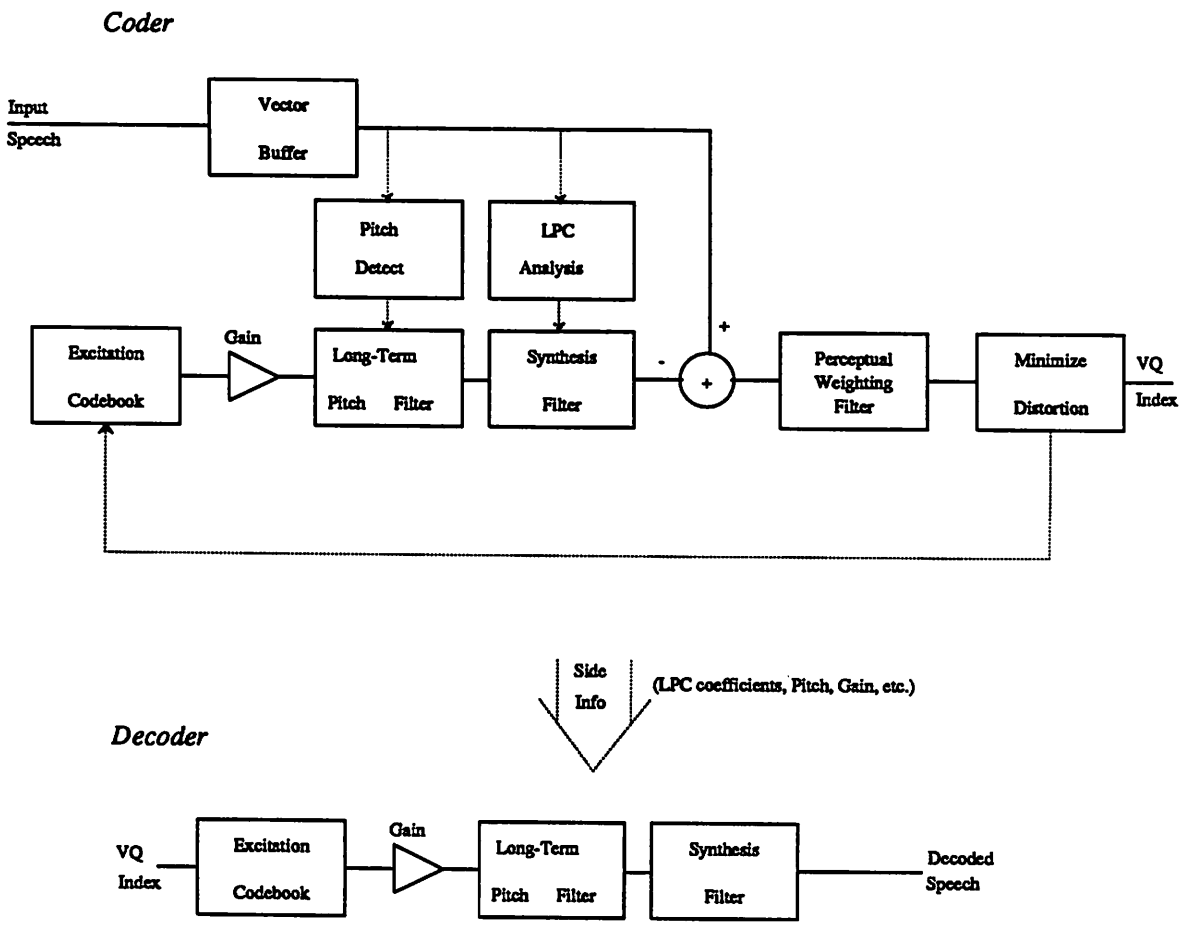


Figure 3.13: Simplified CELP Speech Coder/Decoder

codebook and is used to synthesize the decoded speech.

The vector quantization process will be most efficient for a white residual signal. Therefore, in almost all CELP coders a long-term pitch filter is included in the synthesis path in order to introduce the pitch correlations into the excitation from the stochastic codebook. The form of this pitch filter is that of a single tap all-pole comb filter with a tap delay equal to the pitch period. The filter and its frequency-domain response are illustrated in figure 3.14. From the figure, it is clear that the filter introduces the pitch periodic fine-structure into the frequency-domain speech signal as discussed in section 3.1.2. An identical long-term pitch filter must be included in the decoder, and the pitch lag, L, must be transmitted along with the codebook index and LPC coefficients.

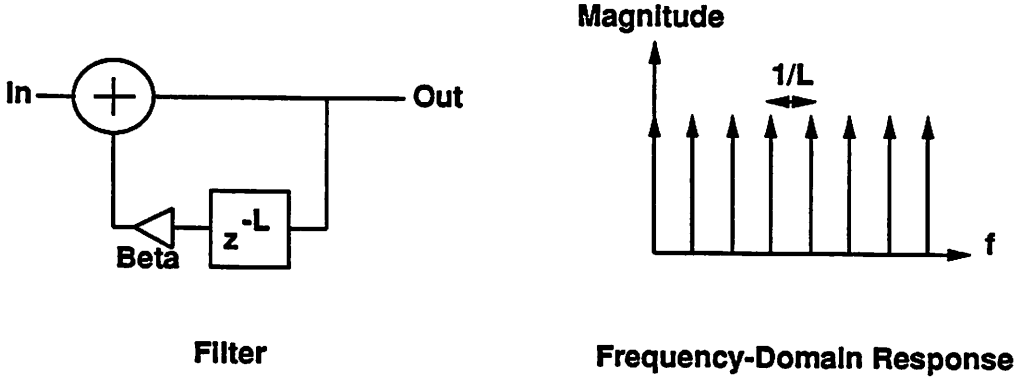


Figure 3.14: Long-Term Pitch Filter Schematic and Frequency-Domain Response

Chapter 4

Low-Power Speech Coding

4.1 Proposed Architectural and Algorithmic Approach

4.1.1 Proposed Architecture

With the knowledge of both low-power VLSI and CELP speech coding accumulated thus far, it is possible to formulate a proposal for a candidate low-power speech coding architecture. As section 2.2.6 suggests, the architecture should provide for highly concurrent execution. Parallel, pipelined, or hybrid architectures are all possibilities. A fully parallel architecture has the inherent disadvantage of extremely large area consumption. Thus, the parallel datapath hardware would dominate chip area. Unfortunately, CELP coders require large amounts of memory – both ROM and RAM. Among other things, this memory is used to store the necessary excitation and pitch codebooks as well as the buffered input samples used to adapt the LPC filter coefficients. Since minimizing I/O provides much needed power savings, this memory should, if at all possible, be stored on-chip. The fully parallel architecture, however, all but eliminates this possibility. Indeed, fitting several parallel datapaths each containing a full array multiplier on a single chip is, in itself, a questionable endeavor.

A pipelined architecture offers a possible solution to this conflict. The only area overhead associated with a pipelined implementation results from the addition of pipeline registers. Thus, the overall datapath area consumption is drastically reduced relative to the parallel implementation. The main disadvantage of this approach lies in the difficulty of efficiently programming a deeply pipelined datapath. Historically, pipelined solutions suffer

from *hazards*. These hazards stem from the fact that in a deeply pipelined processor, the prefetching and execution of an instruction begins before the previous instruction(s) has completed.

The two main types of hazards are branching hazards and data hazards. The former occurs when, while prefetching the next instruction, the wrong assumption is made about whether a branch will be taken. If this occurs, the prefetched instruction(s) must be discarded, and prefetching must begin again at the correct instruction. In contrast, a data hazard refers to an attempt to utilize the result of a previous computation before that calculation has completed. Both of these types of hazards can seriously hamper attempts to efficiently program a pipelined processor.

A possible solution to this difficulty involves interleaving the execution of multiple programs in a single pipeline [16]. As Lee demonstrates, under certain conditions, this pipelined-interleaved (PI or Π) processor behaves as N parallel processors sharing a single memory without contention (where N is the pipeline depth). Therefore, the Π processor avoids the programming difficulties associated with pipelined processors while occupying only slightly more area (additional area is required to store the states of the virtual processors). Figure 4.1 shows a high-level diagram of Lee's Π processor architecture. The architecture of this figure contains one refinement over Lee's implementation. That refinement is the ability to switch off the interleaving mechanism when desired, turning the processor into a classic deeply pipelined processor. This capability is manifested in the architecture by multiplexers, which allow the user to bypass the interleaving state registers associated with the accumulator, the indexing registers, and the program counter.

Based on the particular tasks associated with CELP speech coding, this basic architecture can be further refined. The CELP coding algorithm decomposes into, basically, two types of subtasks. One of these subtasks is characterized by highly sequential, branch-intensive, control-driven code. This type of code avoids parallel implementation and is characteristic of the tightly coupled adaptive feedback loops associated with LPC coefficient filter updates. The second type of task is the datapath-intensive computation associated with the analysis-by-synthesis codebook search procedures.

Unfortunately, while the inherently parallel codebook search algorithms conform well to the independence requirements of the Π processor, the branch-intensive, sequential components of CELP would severely undermine the efficiency of the processor. For when executing a single, sequential piece of code, interleaving is of no use and the processor

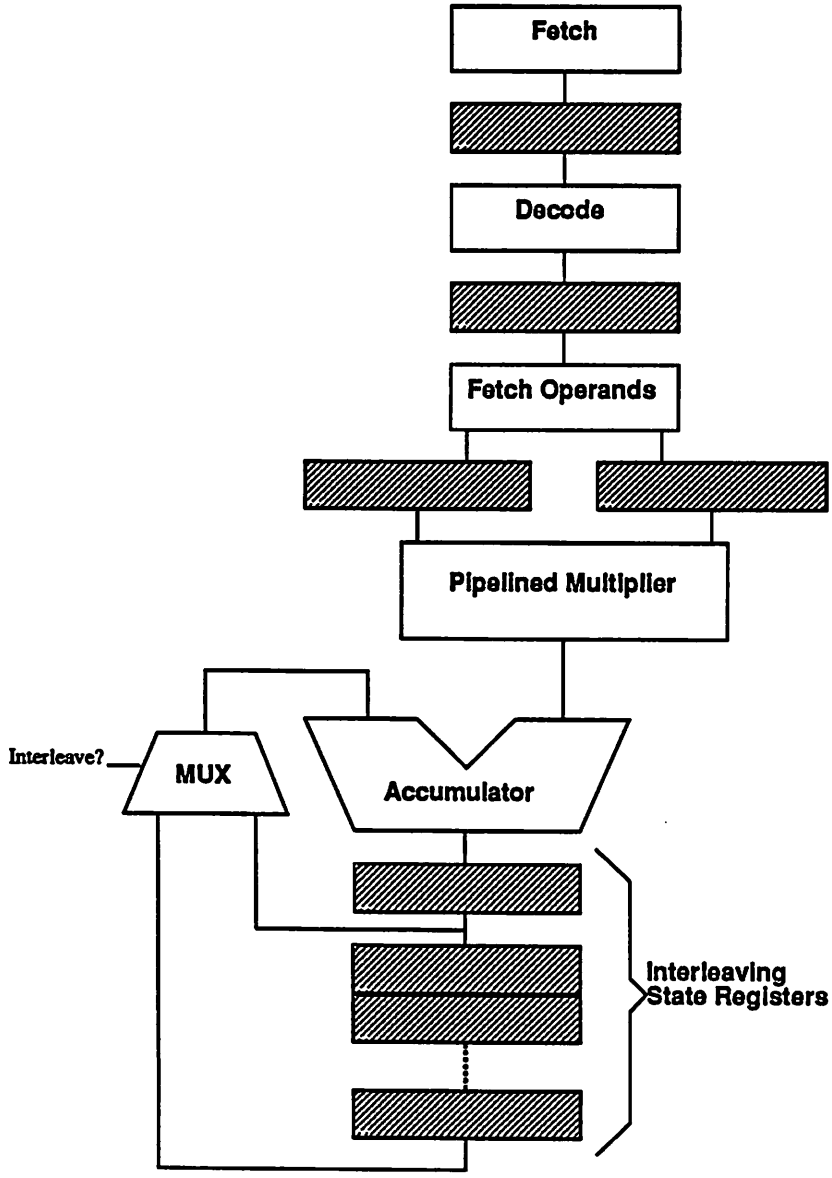


Figure 4.1: High-Level Diagram of Pipelined-Interleaved (Π) Architecture (after Lee [16])

behaves as a conventional deeply pipelined processor. Such an architecture is notoriously unsuited for branch-intensive, sequential code. In this situation, a less pipelined Von Neumann architecture gives much better performance.

This suggests modification of the proposed low-power architecture to a co-processor organization. Specifically, a promising configuration consists of a control-oriented processor aimed at efficiently executing the definitively non-parallel sections of the CELP algorithm, coupled with a deeply pipelined Π co-processor used to perform the computationally intensive pitch and codebook searches at the heart of the CELP algorithm. This Π processor component would have minimal control overhead, allowing only the most basic forms of branching and looping. This optimization minimizes power consumption associated with control overhead, while still enabling efficient handling of the easily parallelized, regular code that would be mapped to the Π co-processor.

4.1.2 Analysis of Architectural and Algorithmic Issues

At this point, a comparison of possible power savings from various architectural techniques will be presented. In addition, the effects of the algorithm parallelization strategy described in section 2.2.7 will be discussed. In particular, with a purely sequential CELP processor as a basis for comparison, the following architectural alternatives will be analyzed: a fully parallel (or pipelined-interleaved) processor, a fully parallel processor executing a parallelized algorithm, a co-processor configuration, and a co-processor executing the parallelized algorithm.

Sequential Processor

A sequential processor executing the CELP algorithm will be used as the metric against which all other architectural and algorithmic variations will be measured (see figure 4.2). In this analysis, the following definitions are of interest:

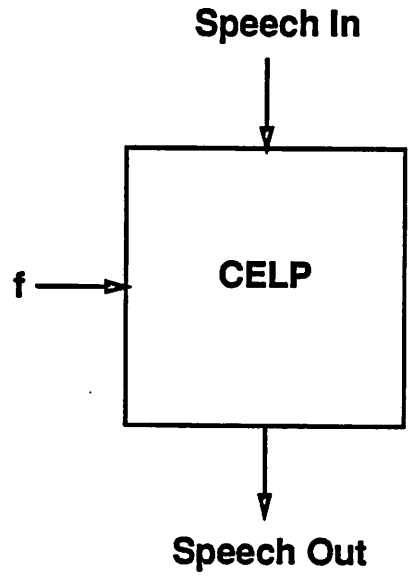


Figure 4.2: Sequential CELP Processor

σ = Number of *indisputably* sequential operations per frame
π = Number of *possibly* parallel operations per frame
$O = \sigma + \pi$ = Total number of operations per frame
τ = Time allotted to process a frame
M = Average number of machine cycles per operation
$f = \frac{1}{T}$ = Clock frequency
C = Capacitance associated with processor hardware
V = Supply voltage from which processor is operated
P = Dynamic power consumed by processor

Using these definitions, a straightforward analysis reveals the following relationships:

$$f = \frac{1}{T} = \frac{(\sigma + \pi)M}{\tau} \tag{4.1}$$

$$P \propto CV^2 f \tag{4.2}$$

Thus, for a sequential processor, the frequency at which the clock must run is determined by the total number of operations to be performed, the average number of clock cycles per operation, and the time allotted to perform these operations. Dynamic power is as derived in section 2.1.

Fully Parallel Processor

With these results as a base, the relevant parameters for a fully parallel processor can be derived. These results are easily extended to the pipelined-interleaved case, which in the algorithmic sense is equivalent to the parallel approach. Since an N-stage Π processor behaves like an N-way parallel processor configuration, the analysis is carried out for the conceptually simpler parallel case. A depiction of the configuration is shown in figure 4.3. Note that all sequential portions of CELP code would be executed on processor CELP_1 with the other processors idle and powered down, while the parallel code segments would execute on all N processors concurrently. The underlying assumption is that all "parallel" code segments are N-way parallel (i.e. may be mapped efficiently onto N independent processors). For this derivation the following additional definitions are convenient:

$\sigma_{\%} = \frac{\sigma}{O}$ = Fraction of operations that must be executed sequentially
$\pi_{\%} = \frac{\pi}{O}$ = Fraction of operations that may be executed concurrently on N processors

Furthermore, the subscript p in the following equations refers to the fact that these parameters relate to the parallel implementation.

We now find that,

$$f_p = \frac{1}{T_p} = \frac{(\sigma + \frac{\pi}{N})M}{\tau} \tag{4.3}$$

Thus, the required clock frequency is reduced (relative to the sequential case) since N of the parallel instructions can be executed simultaneously during each clock cycle. As discussed in section 2.2.6, this reduced clock frequency allows us to reduce the supply voltage in the same proportion. If the supply voltage, V_p, is scaled such that $V_p = \alpha_p V$ then,

$$\alpha_p = \frac{f_p}{f} = \sigma_{\%} + \frac{\pi_{\%}}{N} \tag{4.4}$$

Therefore, for fully parallel code, the voltage supply can be reduced by a factor of N. Furthermore, a simple derivation reveals that with this voltage scaling,

$$P_p = P_p^{sequential} + P_p^{parallel} = \alpha_p^2 P \tag{4.5}$$

This result clearly shows the importance of minimizing the scaling factor α_p. If the number of parallel processors is set large enough so that execution of parallel code is not the computational bottleneck, then the lower bound on α_p is simply $\sigma_{\%}$, the fraction of sequential code in the algorithm. That is,

$$\text{if } N \gg \frac{\pi_{\%}}{\sigma_{\%}} \text{ then:} \qquad \frac{P_p}{P} \approx \alpha_{p,min}^2 \approx \sigma_{\%}^2 \tag{4.6}$$

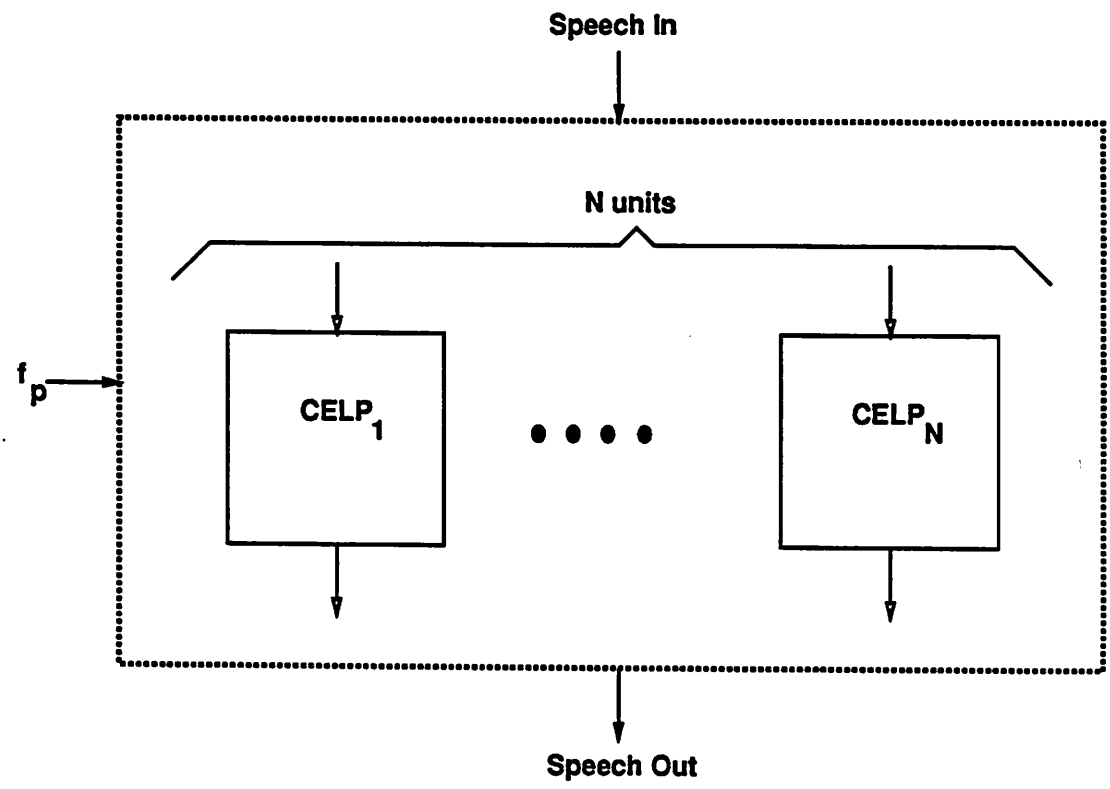


Figure 4.3: Parallel CELP Processor

An $N \simeq (10 \ldots 20)\frac{\pi_{\%}}{\sigma_{\%}}$ should be sufficient. The key objective, then, is to minimize the number of sequential instructions while providing enough processors to effectively handle the parallel computational load.

Consider the following example based loosely upon the Motorola VSELP algorithm. Let $M = 1$ cycle/op, $V = 5$ V, $\tau = 20$ ms, $\sigma + \pi = 350000$, $\sigma_{\%} = 0.5$, and $\pi_{\%} = 0.5$. On a sequential processor, this algorithm would require a clock frequency of approximately 17.5 MHz. Figures 4.4 and 4.5 show the behavior of parallel power and frequency as N is increased. An $N \simeq 15$ gives a power reduction close to the theoretical limit of four. Using this value of N, the supply voltage and operating frequency become 2.67 V and 8.75 MHz, respectively. In order to further reduce power consumption, the number of sequential operations in the algorithm must be reduced.

Fully Parallel Architecture Executing Parallelized Algorithm

In certain cases, sequential code can be transformed into parallel code; however, it is usually not a one-to-one translation. That is, each sequential instruction corresponds to several, say k, parallel instructions. Using prime ($'$) notation to refer to parallelized parameters, define

$$\begin{aligned}
\sigma' &= \sigma - \Delta\sigma = \text{Number of sequential operations in parallelized algorithm} \\
\pi' &= \pi + \Delta\pi = \pi + k\Delta\sigma = \text{Number of parallel operations in parallelized algorithm} \\
\Delta\sigma_{\%} &= \frac{\Delta\sigma}{\sigma+\pi} = \text{Fraction of original algorithm parallelized} \\
\Delta\pi_{\%} &= \frac{\Delta\pi}{\sigma+\pi} = \text{Ratio of additional parallel instructions to number of original instructions} \\
s &= \frac{\sigma'}{\sigma} = \text{Fraction of sequential operations remaining sequential after parallelization}
\end{aligned}$$

For convenience, the case under consideration will, in future, be referred to as the parallelized-parallel case. The processor configuration is identical to the parallel scenario with the exception that the number of processing units is now N'; however, for comparative purposes we may take $N = N'$ in the following. A simple analysis leads to,

$$\alpha_p' = (\sigma_{\%} + \frac{\pi_{\%}}{N'}) - \Delta\sigma_{\%}(1 - \frac{k}{N'}) = (\sigma_{\%} - \Delta\sigma_{\%}) + \frac{1}{N'}(\pi_{\%} + k\Delta\sigma_{\%}) \tag{4.7}$$

From these equations, it is clear that if $k \geq N'$ then $\alpha_p' \geq \alpha_p$, and no power reduction over the parallel case is achieved. Thus, the prerequisite for a useful parallelization scheme is that $k \ll N'$. In other words, the number of additional parallel instructions required to parallelize each sequential instruction should be much less than the number of processors.

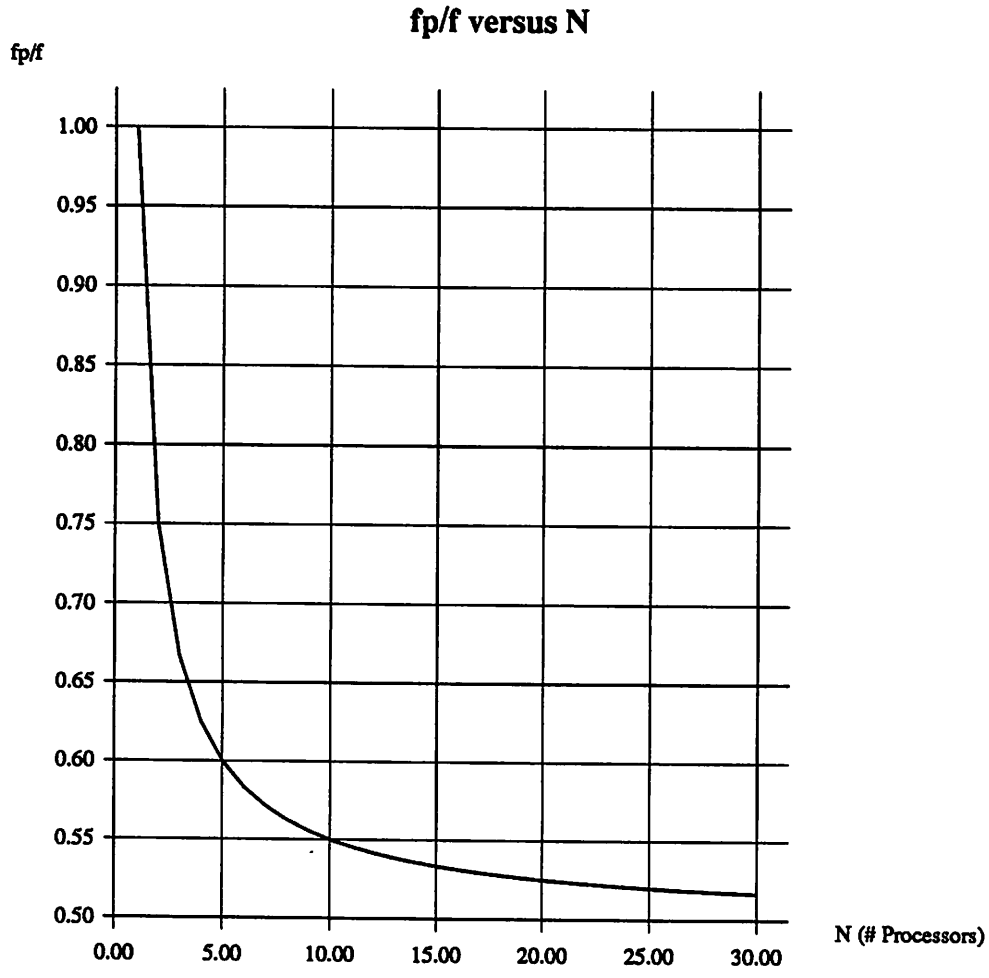


Figure 4.4: Effect of Number of Parallel Processors on Operating Frequency

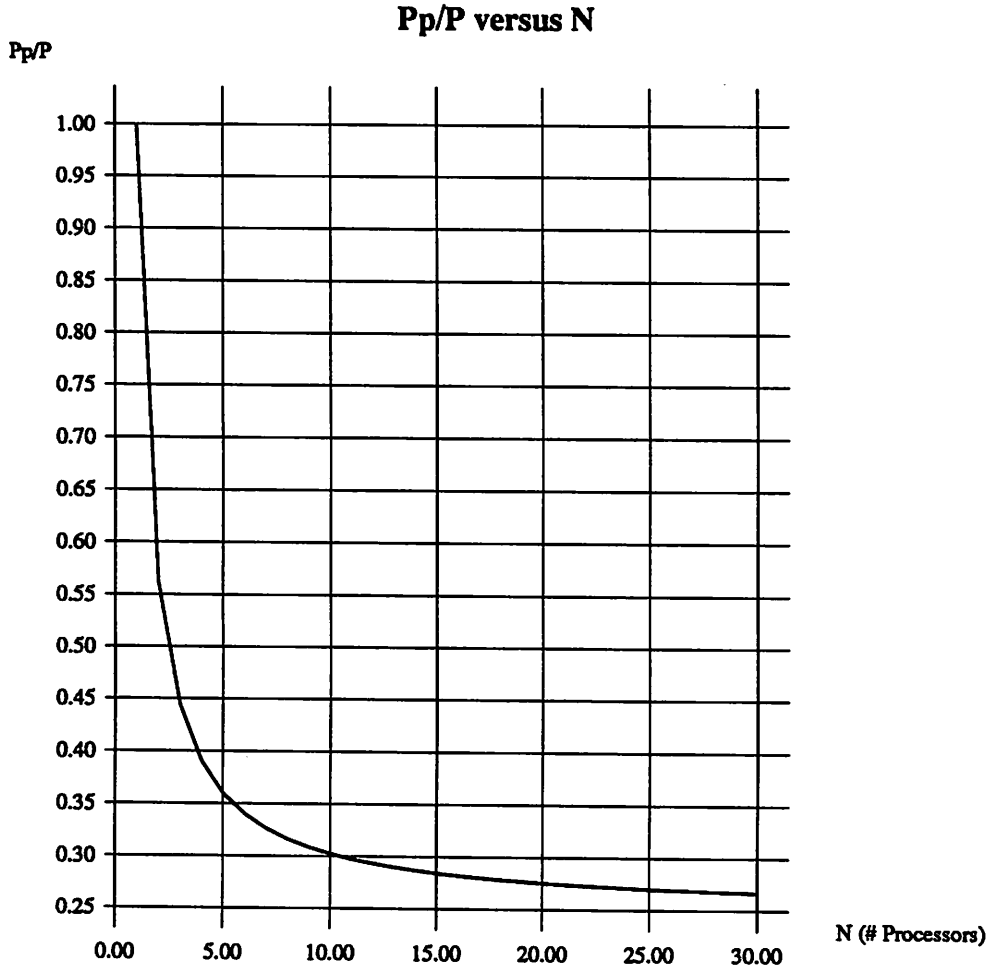


Figure 4.5: Effect of Number of Parallel Processors on Power Savings

Otherwise, it would require more cycles to execute the parallelized algorithm than the original algorithm. In mathematical terms:

$$\text{for } k \ll N': \qquad \alpha'_p \rightarrow \alpha_p - \Delta\sigma_{\%} \tag{4.8}$$

Since α_p approaches $\sigma_{\%}$, for large N, α'_p and, subsequently, the power can be minimized by allowing $\Delta\sigma_{\%}$ to approach $\sigma_{\%}$. This corresponds to parallelizing all of the sequential code in the original algorithm. In this case,

$$\text{as } \Delta\sigma_{\%} \rightarrow \sigma_{\%}: \qquad \alpha'_p \rightarrow \frac{1}{N'}(\pi_{\%} + k\sigma_{\%}) \tag{4.9}$$

In theory, as N' is increased, α'_p and the power consumption could become arbitrarily small. Recall, however, that $\frac{V'_p}{V} = \alpha'_p$ implying that for large N', the supply voltage would become arbitrarily small. Due to physical limitations this is clearly impossible. To the first order, assume that the minimum V'_p is just the threshold voltage ($V_t \approx 0.8$ V). Then, using equation 4.9 we have

$$\text{for } \Delta\sigma_{\%} \rightarrow \sigma_{\%}: \qquad N' \leq \frac{V}{V_t}(\pi_{\%} + k\sigma_{\%}) \tag{4.10}$$

Turning now to the expression for power consumption we find,

$$P'_p = \alpha'^2_p \frac{\sigma' + \pi'}{\sigma + \pi} P = \alpha'^2_p P' \tag{4.11}$$

where P' is the power that would be consumed were the newly parallelized algorithm executed on the original sequential processor. Further computations show that

$$\frac{\sigma' + \pi'}{\sigma + \pi} = 1 + (k-1)\Delta\sigma_{\%} \tag{4.12}$$

This term, which will be larger than one, represents the degradation in power savings due to the instruction count overhead of the parallelization process.

Equation 4.11 relates the parallelized power consumption to the sequential processor power; however, a comparison to the parallel implementation would be more meaningful. This comparison leads to the following equation,

$$\frac{P'_p}{P_p} = \frac{\alpha'^2_p}{\alpha^2_p}\frac{O'}{O} \tag{4.13}$$

This relationship exhibits an interesting behavior when only a small fraction of the sequential operations are parallelized ($\Delta\sigma_{\%} \ll \sigma_{\%}$). That is to say, when very little parallelization

has been done. Specifically, for small amounts of additional parallelization, the power consumption actually increases over the parallel implementation of the original algorithm. The reason for this lies in the fact that for small $\frac{\Delta\sigma_{\%}}{\sigma_{\%}}$, the overhead factor, $\frac{O'}{O}$, due to the increased overall instruction count, rises more rapidly than the scaling ratio, $\frac{\alpha'^2_p}{\alpha^2_p}$, falls. If $N' \gg k$, however, this trend quickly reverses and increased parallelization begins to offer significant power savings. In the limit, as all existing sequential code is parallelized,

$$\text{as } \Delta\sigma_{\%} \rightarrow \sigma_{\%}: \qquad \frac{P'_p}{P_p} \rightarrow \frac{[\frac{\pi_{\%}+k\sigma_{\%}}{N'}]^2[1+(k-1)\sigma_{\%}]}{[\sigma_{\%}+\frac{\pi_{\%}}{N}]^2} \tag{4.14}$$

An example will help to clarify this concept and others. This example is actually an extension of the previous VSELP case study. In addition to the parameters defined there, we specify that each sequential instruction translates on average to four parallel instructions ($k = 4$). We further specify that half of the originally sequential instructions are to be parallelized ($s = 0.5$, $\Delta\sigma_{\%} = 0.25$). Using these parameters in equation 4.10 (and assuming $V_t = 0.8$ V) gives $N' \leq 15.6$. We choose $N' = 15$ and note that this satisfies the desired $N' \gg k$ relationship.

Assuming $N = N'$ and using equations 4.4 and 4.7, we see that $\alpha_p = 0.5\overline{33}$ and $\alpha'_p = 0.35$. Accounting now for the increased total instruction count overhead, we find $\frac{O'}{O} = 1.75$. Applying equation 4.13 yields the result that $P'_p = 0.76P_p$. Furthermore, the operating voltage, V'_p, becomes 1.75 V while the operating frequency goes to 6.13 MHz.

As stated above, however, algorithmic parallelization does not always lead to reduced power consumption. Figure 4.6 shows a normalized graph of $\frac{P'_p}{P_p}$ verses $\frac{\Delta\sigma_{\%}}{\sigma_{\%}}$. Clearly, in some cases increased parallelization actually increases power consumption. This is mainly true for relatively small values of N'; however, even for values of N' near $N_{max} = 15$ there is still a brief region, for $\frac{\Delta\sigma_{\%}}{\sigma_{\%}}$ small, where increased parallelization is of no benefit.

One might wish to calculate the bound on power reduction when all sequential instructions in the original algorithm are parallelized, $\Delta\sigma_{\%} \rightarrow \sigma_{\%}$. Using equation 4.14 for this case, $P'_p \rightarrow 0.24P_p$. For this bounding case, the corresponding supply voltage and clock frequency are given as 0.83 V and 2.92 MHz. This demonstrates the failings of this first order analysis. For as $V_{dd} \rightarrow V_t$, the linear scaling of delays with V_{dd} breaks down and the results become meaningless.

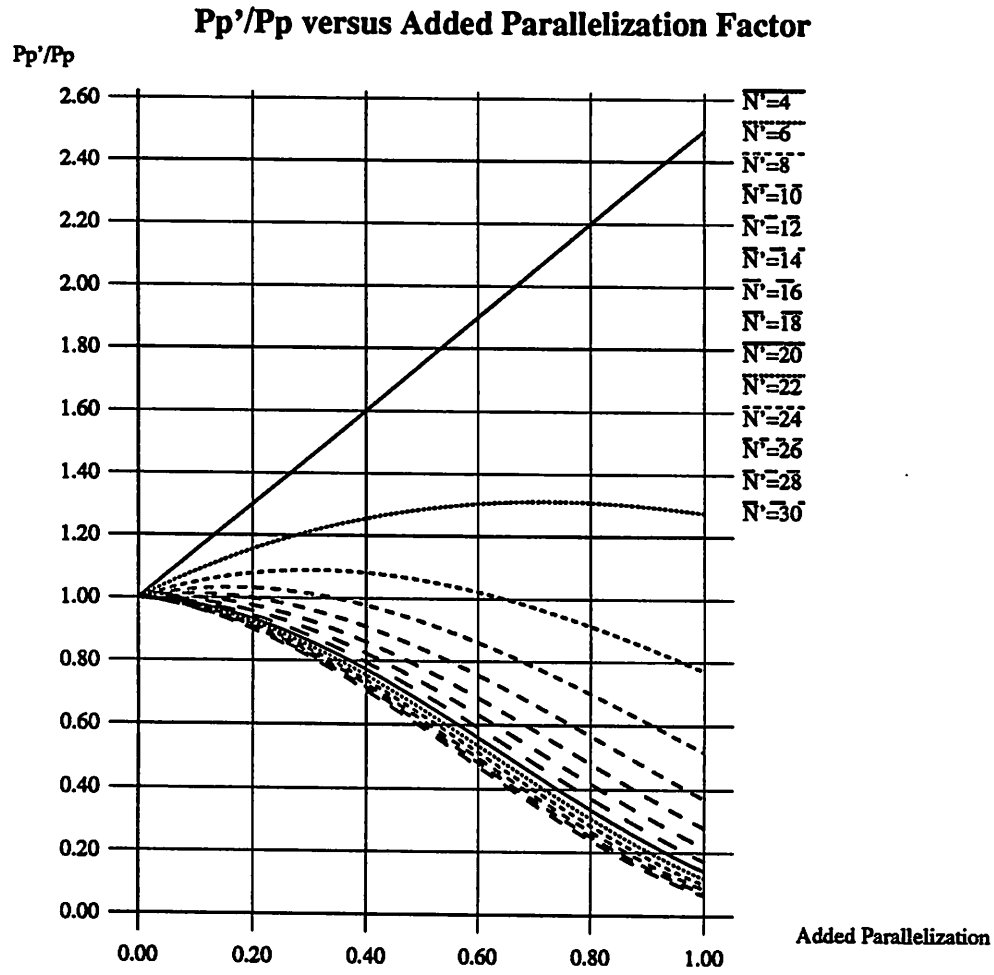


Figure 4.6: Effect of Parallelization on Power Savings Relative to Original Parallel Implementation (k=4)

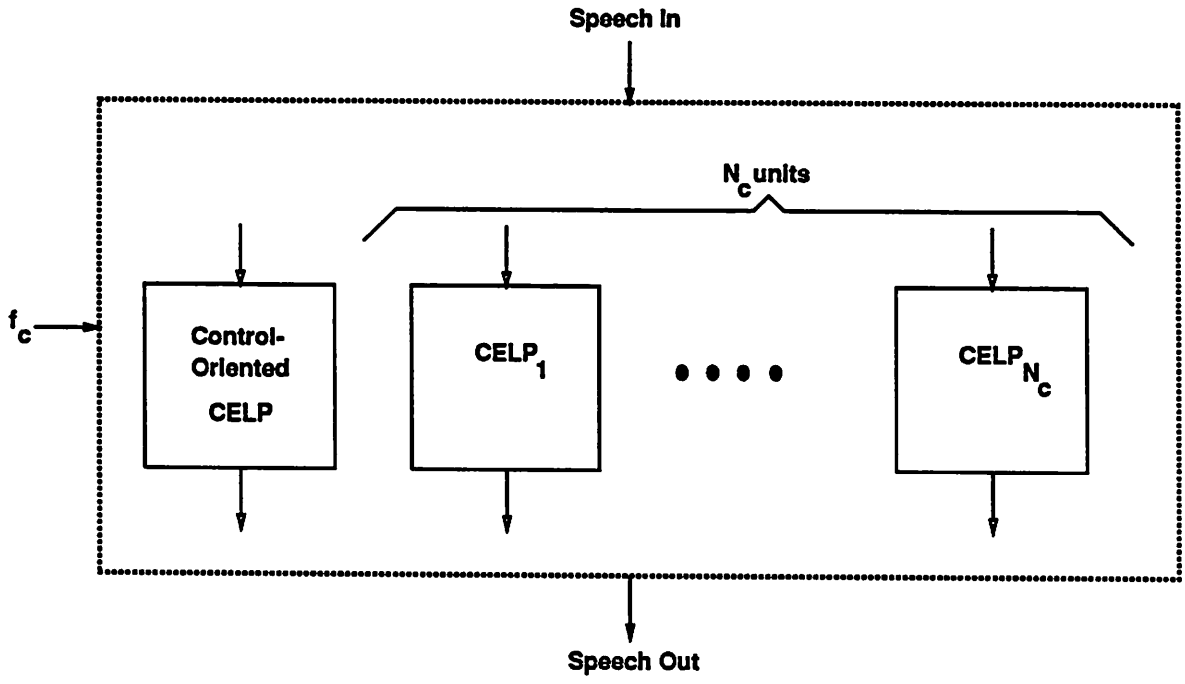


Figure 4.7: CELP Implementation Based on Co-Processor Architecture

Co-Processor Configuration

The analysis of the co-processor coder configuration follows closely that of the parallel processor architecture. This is understandable since the co-processor architecture is a simple extension of the parallel case. For the parallel configuration there were N identical processors acting in parallel to execute concurrent code. When executing sequential code, however, all but one of the processors were shut down, leaving a single sequential processor. In the co-processor scenario of figure 4.7, on the other hand, instead of using one of the parallel datapaths, a custom control-oriented sequential code processor is provided to process non-parallel code.

With this processor available, the control-related hardware in the parallel processors can be minimized, reducing the capacitance associated with this hardware. This, in turn, offers the possibility of power reduction. Unfortunately, the elimination of advanced control features from the parallel processors will most likely lead to an increase in the parallel operation count. In the subsequent detailed analysis, the following definitions are

useful:

$S_{C_\pi} = \frac{C_\pi}{C}$ = Factor by which total capacitance of a parallel processor scales ($S_{C_\pi} < 1$)
$\Delta\pi_c$ = Additional parallel instructions required to compensate for reduced control
$\pi_c = \pi + \Delta\pi_c$ = Total number of parallel operations after including $\Delta\pi_c$
$S_\pi = \frac{\pi_c}{\pi}$ = Factor by which number of parallel operations is increased ($S_\pi > 1$)

Note that S_{C_π} encompasses the reduced capacitance afforded by minimized control hardware requirements. Also, note that in this section, the subscript c refers to co-processor parameters.

For this configuration, the new scaling factor, α_c, is given by

$$\alpha_c = \sigma_\% + \frac{S_\pi \pi_\%}{N_c} \tag{4.15}$$

In the case of large N_c where the sequential instructions represent the execution bottleneck we have,

$$\text{for } N_c \gg \frac{S_\pi \pi_\%}{\sigma_\%}: \qquad \alpha_c \rightarrow \sigma_\% \simeq \alpha_p \tag{4.16}$$

Furthermore, as always $\frac{V_c}{V} = \frac{f_c}{f} = \frac{T}{T_c} = \alpha_c$.

Of particular interest are the power consumption results. Relative to the original sequential case we have,

$$P_c = (\sigma_\% + S_{C_\pi} S_\pi \pi_\%)\alpha_c^2 P \tag{4.17}$$

Now extending the comparison to the parallel processor architecture and looking at the sequentially bounded case (large N_c), we arrive at

$$\text{for } N_c \gg \frac{S_\pi \pi_\%}{\sigma_\%}: \qquad \frac{P_c}{P_p} = \sigma_\% + S_{C_\pi} S_\pi \pi_\% \tag{4.18}$$

The co-processor architecture is only useful when $\frac{P_c}{P_p} \ll 1$, resulting in the following condition:

$$\text{for } N_c \gg \frac{S_\pi \pi_\%}{\sigma_\%}: \qquad P_c \ll P_p \text{ when } S_\pi \ll \frac{1}{S_{C_\pi}} \tag{4.19}$$

Thus, in order to benefit from the co-processor architecture, the fractional increase in the number of parallel instructions should be much less than the inverse of the capacitive scaling factor.

Consider, once again, the VSELP example. Assuming control hardware overhead accounts for approximately 25% of the capacitance associated with the parallel processors, we expect a minimum $S_{C_\pi} \simeq 0.8$. Using restriction 4.19, we see that S_π must be less

than 1.25. This is quite a tight restriction, but let's assume that we meet it with $S_\pi = 1.10$ (a 10% instruction increase). Even with this low instruction overhead, we achieve only a 6% power reduction relative to the parallel processor case (i.e. $\frac{P_c}{P_p} = 0.94$). Increasing parallelization will improve the situation; however, even for a completely parallel program we can, at best, achieve $\frac{P_c}{P_p} = S_{C\pi} S_\pi = 0.88$.

This result does not seem particularly promising for the co-processor configuration; however, we have ignored one important fact. We have assumed that sequential code executes as efficiently on a single slice of the parallel processor as it does on the sequential co-processor. In reality, the deeply pipelined character of the parallel processor slice greatly degrades its performance on sequential code. Thus, the relative power savings from the co-processor approach will be much larger than the predicted 12%.

4.1.3 Summary of Architectural and Algorithmic Techniques

From the preceding discussion we can draw several conclusions. First, of the cases studied, maximal power savings can be achieved from a co-processor configuration. In this architecture, one processor is a highly concurrent computational unit, while the other is optimized for executing sequential, control-intensive code. In order to avoid the large area penalties associated with fully parallel implementations, concurrency could be achieved using a pipelined-interleaved processor as suggested by Lee. Such a processor may be programmed as if it were N independent, parallel processors and, thus, we avoid the programming difficulties related to deeply pipelined processors.

Furthermore, as shown by equation 4.6, power will be minimized by choosing $N \gg \frac{\pi\%}{\sigma}$. In the example, the power reduction attributable to a 15-way parallel processing system was shown to be $\frac{P_p}{P} = 0.28$. By further increasing concurrency in the algorithm we were able to compound power savings; however, increased concurrency came at the cost of some operational overhead. In the example, we assumed that for each sequential operation eliminated, four parallel operations were required. In order for the parallelization process to be worthwhile, this instruction overhead must be less than the number of parallel processors.

In addition, we found that the parallelization process could not proceed without bound; the inability to scale V_{dd} past V_t made parallelization ineffective beyond some point. For the above example, the maximum effective N was shown to be fifteen. Using this result, by parallelizing 50% of the remaining sequential code we saved additional power over the

parallel case as given by $\frac{P_p'}{P_p} = 0.76$.

The final optimization, results from utilizing a co-processor approach rather than a strictly parallel processor approach. By combining a processor optimized for sequential, control-driven code with a deeply pipelined-interleaved processor for executing parallel code, an additional power reduction on the order of 10-30% was shown possible. Combining these results, an overall power reduction of a factor of six or more over a strictly sequential implementation should be quite reasonable.

The accumulated strategy for low-power algorithmic optimization, then, is as follows. Algorithm selection should be based on the extent of existing concurrency as well as overall instruction count. Furthermore, the selected algorithm should be amenable to additional parallelization with minimal overhead. Finally, the algorithm should be implemented on a co-processor architecture consisting of a control-intensive Von Neumann style processor coupled to a deeply pipelined ($\approx$ 15 stages) interleaved co-processor for execution of concurrent code. This methodology is general enough to apply not only to speech coding algorithms, but also to a wide variety of DSP applications. Recall, however, that the validity of these results depends intimately on several assumptions. For example, the final algorithm must have enough inherent concurrency to efficiently utilize the parallelism provided by the suggested architecture. Moreover, the overhead incurred in delay and power by the required pipelining hardware must not become significant relative to that associated with the processing elements.

4.2 VSELP Case Study

As an application of the proposed low-power methodology, three high quality, low bit-rate CELP coding algorithms will now be analyzed for low-power suitability. The first algorithm, Motorola's 8.0 Kbit/sec VSELP coder [10], will be analyzed in detail. Like the other two algorithms under consideration, VSELP takes as input a 64 Kbit/sec input stream of 8 KHz, 8-bit μ-Law PCM samples, which it immediately converts to 16-bit uniform PCM for further processing. Similarly, the output, after decoding, is another 64 Kbit/sec, 8-bit μ-Law PCM stream. Furthermore, since all three coders achieve very high quality output speech with MOS scores above 4.0, coding quality was not emphasized as a basis for comparison of the algorithms.

The analysis in this section will begin with a description of the computational

complexity of the VSELP algorithm. In this context, complexity refers mainly to the number of arithmetic and comparison operations that must be performed by the algorithm during each coding frame. The next section will discuss the suitability of the VSELP algorithm for parallel implementation. Following that discussion, the analysis will proceed with low-power optimizations for key subsections of the algorithm. These optimizations focus on increasing the available concurrency in the algorithm. Following this analysis, focus will be shifted to the two other CELP algorithms under consideration: the AT&T LD-CELP algorithm and the DoD CELP algorithm. Having covered the main low-power issues in the detailed VSELP presentation, discussion of the other two algorithms will be mainly by way of comparison and contrast to the VSELP coder.

4.2.1 Overview of Algorithm

Before proceeding with the VSELP case study, a brief overview of the distinctive features of the algorithm is required. The VSELP frame size is 20 ms and refers to the period with which the LPC coefficients of the synthesis and perceptual weighting filters are recalculated. Each frame is divided into four subframes of 40 samples and 5 ms each. For each subframe, the codebook search procedures determine the appropriate codebook indices and gains. The results of this process are transmitted subframe-by-subframe to the decoder.

A high level block diagram of the VSELP coder is shown in figure 4.8. As with all CELP algorithms, coding is based on an analysis-by-synthesis methodology. Conceptually, linear combinations of the vectors in the three codebooks are taken as excitations to a synthesis filter embedded in the coder. The resulting synthesized speech is perceptually compared to the input speech and a mean-square distortion measure is calculated. Using this synthesis approach, an attempt is made to find the codebook indices and gains that result in the least distorted synthesized speech.

These gains and indices are then transmitted to the decoder along with the appropriate LPC coefficients. As illustrated by the figure, the decoder is basically a replica of the codebook search portion of the coder. It consists of the three excitation codebooks, a synthesis filter, and an optional spectral postfilter. As in the coder, the output speech is created by exciting the synthesis filter with a linear combination of the indexed codevectors from the three codebooks. A more detailed description of the entire process follows:

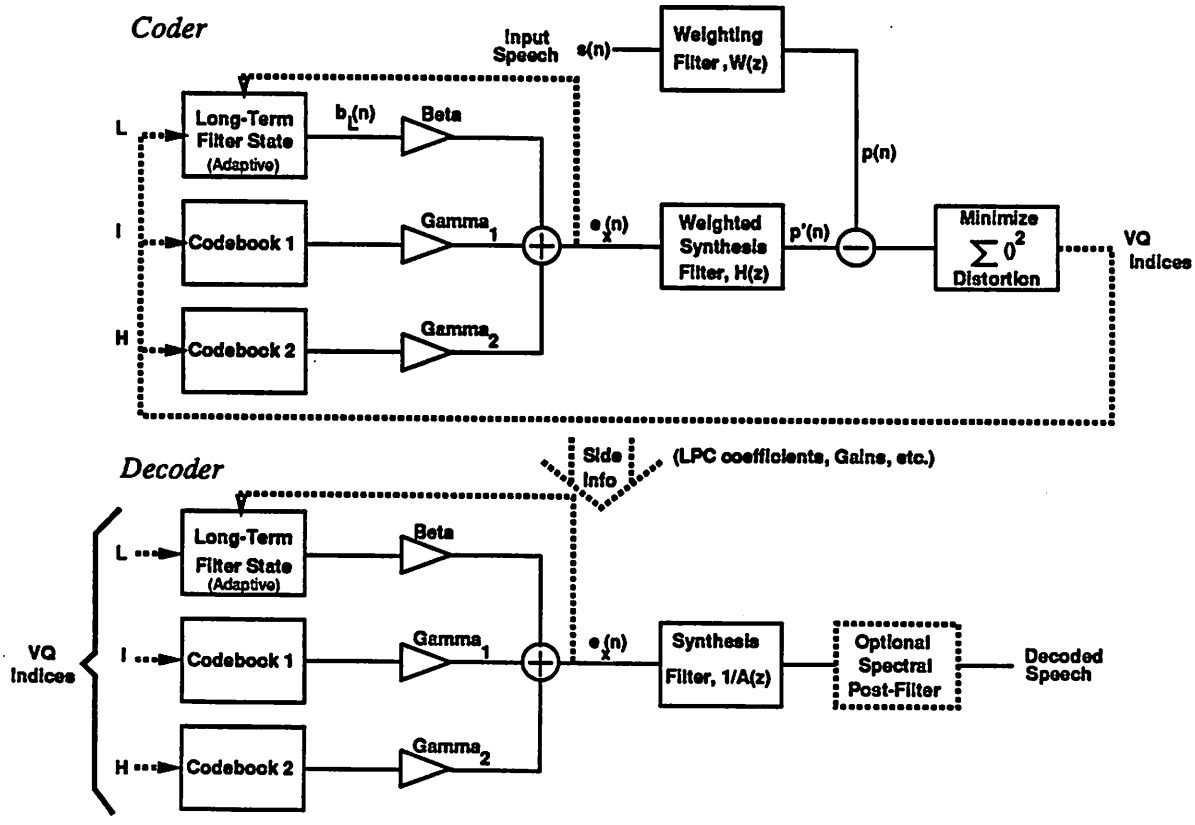


Figure 4.8: High-Level Block Diagram of VSELP Coder/Decoder

Coder

Preparations for Codebook Search Once in every frame, an LPC analysis, as described in section 3.5.1, is used to calculate the coefficients of the synthesis and perceptual weighting filters. This analysis is centered in the fourth subframe of each frame and utilizes an analysis window of 170 samples (21.25 ms). The analysis procedure is a fixed-point covariance lattice algorithm called FLAT and is similar to the Burg algorithm for reflection coefficient calculation [14]. Since the LPC analysis is performed only once per frame, the direct form LPC coefficients for each subframe are interpolated based on the values from the current and previous frames. A by-product of the FLAT analysis is an estimate of the energy associated with the current frame. As with the LPC coefficients, subframe energies are estimated by interpolation of frame energies. In this case, however, interpolation is accomplished by a geometric, rather than a linear, interpolation process.

In addition, a perceptual noise-weighting filter is employed to account for the acoustical noise masking properties of the human ear. People are less able to distinguish noise (error) energy concentrated near speech formants than they are noise located in the troughs of the spectral speech envelope. The coding process accounts for this by means of a perceptual weighting filter, $W(z)$. Figure 4.9 conceptually illustrates the generation of the weighted error signal during a codebook search. This configuration is completely equivalent to that depicted in figure 4.8. The initial error signal is given by the difference between the input speech and the synthesized speech signals. This error signal is then passed through the weighting filter with transfer function, $W(z) = \frac{A(z)}{A(\frac{z}{\lambda})}$, where $A(z)$ is the LPC analysis filter and $\lambda = 0.8$. In the frequency-domain, this filter has troughs at each of the formant frequencies. As a result, noise energy concentrated near these frequencies will receive less weight than noise occurring in other spectral regions. Thus, noise is allowed to accumulate near formant frequencies, while noise occurring near anti-resonances is minimized. In the actual coder implementation, the weighting process takes place at the inputs of the speech signal differencing node rather than the output. In other words, the input and synthesized speech signals are filtered by $W(z)$ prior to generating the error signal. In the synthesis path, then, $H(z) = \frac{W(z)}{A(z)} = \frac{1}{A(\frac{z}{\lambda})}$ is referred to as the weighted synthesis filter.

The next step in the coding process is to subtract off the zero-input response of weighted synthesis filter, $H(z)$, from the weighted input speech. This is an attempt to minimize the complexity of the upcoming codebook search procedure. For if the state com-

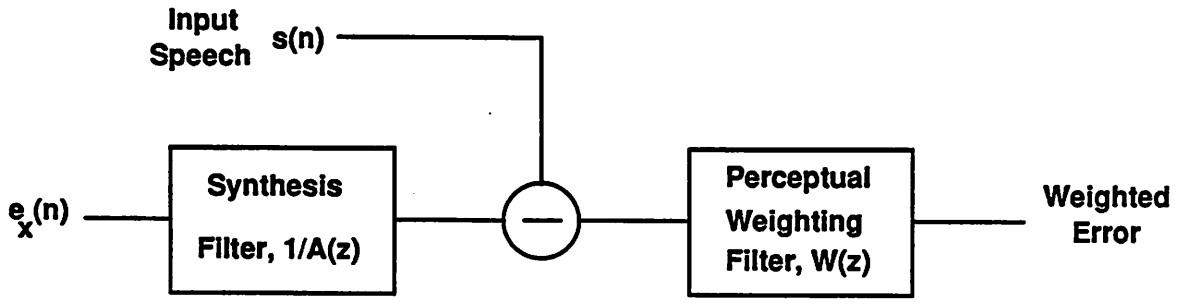


Figure 4.9: Concept of Perceptual Noise-Weighting

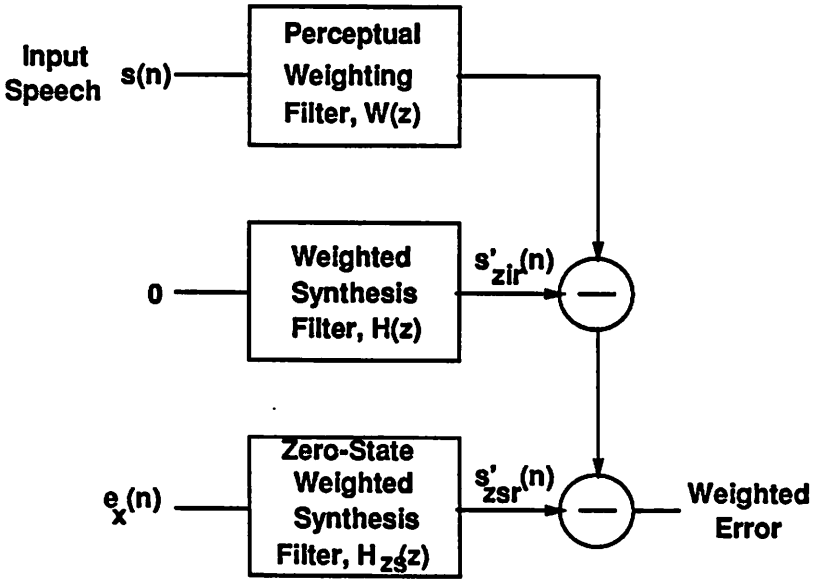


Figure 4.10: Removal of Synthesis Filter Memory Component

ponent of $H(z)$ can be eliminated, the number of multiplications required in the synthesis filtering process can be drastically reduced. To this end, the response of $H(z)$ to an excitation is decomposed into two parts: a zero-state response and a zero-input response. The zero-input response contains the component of synthesized speech due to the state of the filter, $H(z)$. It is this component that is subtracted from the weighted input speech before the codebook searches begins. The only component that must be calculated during the search, then, is the zero-state response. As a result, significantly less multiplications are required. Figure 4.10 illustrates this procedure.

Pitch Codebook Search The pitch filter component of the classical CELP algorithm (as described in section 3.5.2) is implemented in the VSELP coder as an adaptive codebook. This concept requires some explanation. The purpose of the long-term pitch filter in traditional CELP coders is to introduce pitch periodicity into the excitation waveform. This allows the excitation codebooks to be approximately white and increases the efficiency of the vector quantization procedure. A simple filter that achieves this end is of the form $B(z) = \frac{1}{1-\beta z^{-L}}$ (see figure 3.14). In the time-domain, for the simple case of $\beta = 1$, the impulse response is an impulse train of unity amplitude and period L, where L corresponds to the desired pitch period. Similarly, in the frequency-domain the transfer function has the form of a comb filter with discrete impulse components at the pitch frequency and all of its harmonics. Thus, as desired, the filter introduces the pitch component into the time and frequency-domain representations of the excitation signal.

The VSELP coder implements the L-stage delay line associated with this filter as an adaptive codebook whose contents are simply past values of the excitation signal. By performing a codebook search, the appropriate pitch lag, L, and pitch gain, β, are found. The codebook is adaptive in the sense that after each subframe excitation has been calculated, the pitch codebook contents must be adapted to reflect this recent history of the excitation waveform. This methodology is completely equivalent to the physical $B(z)$ filter implementation.

The actual codebook search methodology is based on an end-point recursion technique. Conceptually, for each codevector corresponding to a given lag, L, there exists an optimal amplification, β_{opt}. For every possible lag, L, the indicated codevector is amplified by its optimal β_{opt} and then used to synthesize a segment of speech. The value of L resulting in the least distortion is selected as the pitch lag. Since codevectors for adjacent values of L differ only in their two endpoints, an endpoint recursion technique is used to reduce the computational requirements of the $H(z)$ filtering process. This recursive methodology is a key stumbling block in the low-power implementation of this algorithm and the issue will be revisited in a later section.

Stochastic Codebook Searches After determining the optimum pitch lag component of the excitation, it remains to determine the optimum components from codebooks one and two. Selection of the codevector from the first codebook proceeds under a joint optimization assumption of gains β and γ_1 (where γ_1 is the gain of the codevector from codebook one).

Selection of the optimal codevector from codebook two is undertaken with the assumption of jointly optimized β, γ_1, and γ_2 (γ_2 is the gain of the second codevector). In order to simplify this joint optimization process, the $H(z)$-filtered codevectors from codebook one are first orthogonalized to the previously selected filtered pitch excitation component prior to the first codebook search. Likewise, before the second codebook search, the filtered codevectors from codebook two are orthogonalized to the filtered pitch codevector as well as the filtered codebook one vector. This orthogonalization process decouples the selection of successive codevectors from previously determined excitation components.

Under normal circumstances, this orthogonalization procedure would be prohibitively complex since it would have to be performed on every codevector from each codebook (and twice on those from the second codebook) – quite an expensive computation for codebooks of 128 vectors each. The unique structure of the VSELP codebooks, however, circumvents this difficulty. Each codebook is described completely by linear combinations of seven basis vectors. Furthermore, the weights associated with these basis vectors are restricted to ± 1. Thus, the aforementioned orthogonalization process need only be carried out on the seven filtered basis vectors from each codebook.

Once again, a recursive search procedure is employed, although this time it is not an end-point recursion. Through a gray-coding procedure, the weights of successive codevectors are constrained to differ in only one basis vector. Using this information, a recursion can be developed to implement the codebook search procedures. This recursion applies equally well to the first and second codebook searches. Moreover, since it is a linear operation, the synthesis process represented by the zero-state $H(z)$ filter need only be performed on the fourteen basis vectors.

Joint Optimization of Excitation Gains At this point, the optimal indices into the adaptive pitch codebook and the two stochastic codebooks have been selected. It remains to jointly optimize the three gains β, γ_1, and γ_2 to achieve minimum distortion. This process is implemented as a classical three-dimensional vector quantization procedure. Instead, of quantizing those three gains, however, three equivalent gains – GS, P_0, and P_1 – are quantized. It turns out that these gains can be efficiently vector quantized due to convenient restrictions on their range space. They are, however, completely equivalent to the three original gains. The vector quantization procedure is carried out over a three-dimensional codebook containing 256 codevectors.

Channel Transmission and Filter Updates The encoding algorithm concludes by transmitting the relevant LPC, codebook index, and gain parameters to the decoder. In addition, the states of the adaptive pitch codebook and the weighted synthesis filter are updated based on the selected (optimal) excitation. Finally, processing of the next subframe begins and the entire procedure repeats.

Decoder

The VSELP decoder is a subset of the coder. As shown in figure 4.8, it consists of the three codebooks, the excitation gain amplifiers, the synthesis filter (unweighted), and an optional spectral postfilter. Using the gains and codebook indices received from the coder, the excitation signal is reconstructed from a linear combination of the gain-scaled codevectors. This excitation is applied to the synthesis filter to produce the output speech (assuming no post-filter). As in the coder, the state of the adaptive pitch codebook and the synthesis filter are then updated. Finally, the whole process repeats.

4.2.2 Computational Complexity

The computational complexity of the VSELP algorithm can be approximately measured by an operation count since the number of operations that must be executed in a given time determines the operational requirements of the host processor. In general, the required processor operations can be broken into two categories: arithmetic instructions and control instructions. The arithmetic operations include multiply-accumulates (MACs), multiplications, additions, and subtractions. The main control instruction is the branch operation, which alters the instruction flow (usually) based on the result of some comparison.

The VSELP algorithm actually requires three other important arithmetic instructions: divisions, common logarithms, square roots, and exponentials. These instructions are not directly supported on most architectures and, thus, some other method (such as the cordic algorithm) based on the existing operations must be applied to implement the instructions. Thus, an accurate instruction count must break these special operations down into their component suboperations; however, for the VSELP algorithm the number of these special instructions relative to the total operation count is so small that this process is unnecessary. Therefore, even these complex instructions are counted as single operations.

The analysis-by-synthesis nature of CELP algorithms makes them inherently com-

Operation	Operations Per 20 ms Frame	Mega-Operations Per Second (MOPS)
Comparison	147707	7.3854
MAC	109664	5.4832
$\times$	39889	1.9945
$\pm$	35823	1.7912
$/$	179	0.0090
$\sqrt{\ }$	28	0.0014
log	9	0.0005
$\uparrow$	1	0.0001
Total	333300	16.7 MOPS

Table 4.1: Complexity of VSELP Coder

plex. For each subframe, all codevectors in the codebook must be synthesized and compared to the input speech. This requirement has led to real-time implementation difficulties. End-point recursions and structured codebooks are often used to reduce instruction counts and achieve real-time implementation of CELP coders; however, the complexity of CELP coders remains among the highest in the speech coding domain.

Table 4.1 presents the instruction counts for the coder portion of the VSELP algorithm. This data is based upon a *C* language implementation of the VSELP algorithm coded by Robert Kavaler of Teknekron. Clearly, the multiply-accumulate and comparison operations dominate the instruction count. Often, comparisons used for looping constructs can be overlapped with instructions from within the loop and their contribution to algorithmic complexity is effectively eliminated; however, these comparisons are explicitly included here in an attempt to account for inevitable execution overhead.

In order to determine which sections of the algorithm dominate the coding process, a breakdown of instruction counts by subblock is useful. Using the *gprof* profiling program, table 4.2 highlighting the computationally intensive subblocks of the VSELP algorithm can be derived. These results illustrate that the majority of the computational complexity of the VSELP coder can be attributed to the pitch search algorithm and the two codebook searches. Specifically, it is the section of the codebook searches that performs the $H(z)$ filtering that dominates the codebook searches.

Since *gprof* results are somewhat dependent on processor and compiler efficiency, a manual count of operations in these routines was performed to verify the results. The data

Task	Subtask	% of Total Runtime	
Pitch Search		37%	
Codebook Searches	$H(z)$ Filtering	25%	41%
	Search Recursions	11%	
	Orthogonalization	5%	
Total		78%	

Table 4.2: *gprof* Breakdown of Dominant VSELP Coder Tasks

Task	Subtask	% of Total Operations	
Pitch Search		35%	
Codebook Searches	$H(z)$ Filtering	19%	36%
	Search Recursions	11%	
	Orthogonalization	6%	
Total		71%	

Table 4.3: *Manual* Breakdown of Dominant VSELP Coder Tasks

from this analysis is summarized in table 4.3. As shown by this table, the computational load is, indeed, centered on the pitch search and codevector filtering operations.

Although the decoder requires relatively few computational resources, table 4.4 is included for completeness. Clearly, since the decoder represents only 4.8% of the overall instruction count, and since it contains no codebook search procedures, it should not be the focus of much optimization. Consequently, it is upon the codebook search and filtering routines of the coder that parallelization and optimization efforts should be focused.

4.2.3 Suitability for Low-Power Implementation

Algorithmic Concurrency

Clearly, the criteria for an algorithm that will efficiently execute on a low-power, parallel architecture are quite different from those for an algorithm intended to execute on a classical Von Neumann architecture. In particular, it is not the overall instruction count that is critical; instead, it is the number of operations that must be performed sequentially that limits performance and power savings.

In VSELP, as with most CELP algorithms, the codebook search procedures domi-

Operation	Operations Per 20 ms Frame	Mega-Operations Per Second (MOPS)
Comparison	7410	0.3705
MAC	4304	0.2152
$\pm$	2921	0.1461
$\times$	1741	0.0871
$/$	383	0.0192
$\sqrt{\ }$	16	0.0008
$\uparrow$	1	0.0001
Total	16776	0.839 MOPS

Table 4.4: Complexity of VSELP Decoder

nate the computational requirements. For a parallel, low-power architecture this is a fortuitous circumstance since for *most* CELP codebooks individual codevectors are independent. As a result, the synthesis filtering and distortion calculations for each of these codevectors can be performed concurrently. Thus, CELP codebook searches are inherently parallel and map well to the pipelined-interleaved processor described above.

Unfortunately, most algorithm designers do not have parallel architectures in mind when they specify a particular CELP algorithm such as VSELP. Usually, the algorithms are targeted at existing off-the-shelf DSP processors such as AT&T's DSP32c or Motorola's 56001 DSP. These can be considered sequential machines relative to the massively pipelined architectures proposed in previous sections for low-power implementations. Therefore, the designers are not concerned with maximizing parallelism in their algorithms. Instead, they attempt to minimize total instruction counts – multiply-accumulates in particular. To this end, the extensive use of recursions in codebook search procedures is common. This methodology was adopted by the VSELP designers in both the pitch and stochastic codebook searches. These highly non-parallel recursions, are not ideal for implementation on a concurrent architecture. Therefore, it can be advantageous to parallelize certain sections of the algorithm at the expense of a (hopefully) slight increase in the total operation count. Once again, these optimization will be concentrated in the pitch and codebook search procedures.

Suitability for Fixed-Point Arithmetic

Another issue of suitability for low-power implementation is that of fixed-point or block floating-point feasibility. Recall that full floating-point implementations require excessive hardware overhead that leads to increased power dissipation. Unfortunately, algorithmic designers who are targeting floating-point DSP architectures often don't consider fixed-point issues. This can be troublesome for applications like speech coding that, historically, suffer from dynamic range issues arising from the widespread squaring and multiplication operations present in the algorithm.

Fortunately, the VSELP designers were aware of this issue. They put some effort into arriving at an algorithm that lends itself well to block floating-point implementation. The fact that Motorola's 56001 DSP chip is block floating-point based might have had some influence on this decision. Since the classic Levinson-Durbin LPC analysis can often frustrate attempts at fixed-point implementation, VSELP relies on the FLAT algorithm, which is specifically intended for a fixed-point processor. This is a point in favor of VSELP as a candidate low-power coding algorithm.

4.2.4 Parallelization of Algorithm

Aside from fixed-point considerations, parallelization of the codebook search and codevector filtering operations are the primary focus of low-power optimizations. These optimizations are presented in two parts, the first focusing on the pitch search procedure and the second on the two subsequent codebook searches (in particular, the codevector filtering portion).

Pitch Codebook Search

The complexity of the pitch search procedure can mainly be attributed to the requirement that all codevectors in the adaptive codebook be passed through the weighted synthesis filter, $H(z)$. In order to reduce the overall instruction count, the VSELP algorithm takes advantage of the overlapping nature of the pitch codebook. Specifically, for successive pitch lag indices, the corresponding codevectors differ only in their two endpoints. Thus, an end-point recursion can be used to generate the required $H(z)$-filtered signal, $b'_L(n)$. This

recursion is described by the following equations:

$$b'_L(n) = \begin{cases} z_L(n) & L > n \\ z_L(n) + z_L(n-L) & L \leq n \end{cases} \tag{4.20}$$

and

$$z_L(n) = \begin{cases} r(-L)h(0) & n = 0 \\ z_{L-1}(n-1) + r(-L)h(n) & 1 \leq n \leq N_T \\ z_{L-1}(n-1) & N_T \leq n \leq N-1 \end{cases} \tag{4.21}$$

where $r(\cdot)$ is the memory of the adaptive pitch filter codebook, $h(\cdot)$ is the impulse response of $H(z)$, L is current value of the pitch lag index, N is the subframe length, and N_T is the duration of impulse response $h(\cdot)$.

The pitch search procedure begins by calculating an initial value for $z_{L_{min}}(n)$ using a straightforward convolution. Then for all successive values of L the above recursion is applied.

The recursive nature of this pitch search is inherently non-parallel. This has been shown to be an obstacle to achieving a low-power implementation. One method of circumventing this situation is to split the codebook into several independent sections. These sections would all execute concurrently. Each would begin with a convolution to calculate their respective $z_{L_{i,min}}(n)$. At that point, however, calculation of the filtered codevectors, $b'_{L_i}(n)$, could proceed in parallel using the above recursion. For although the independent $b'_{L_i}(n)$ calculations are locally recursive, their calculation for mutually exclusive ranges of L can be interleaved into a processor pipeline to achieve concurrency.

As stated previously, this concurrency comes at the price of some increased instruction overhead. In this case, the overhead comes from the extra initial convolutions that must be carried out. These convolution amount to manually filtering given codevectors through the $H(z)$ filter. This filtering process, however, can itself be parallelized, and this is the topic of the next section.

Stochastic Codebook Searches

The majority of the computational load attributable to the stochastic codebook searches is due to the filtering of the fourteen basis vectors through the $H(z)$ weighted synthesis filter. Furthermore, as was seen in the last section, the efficiency of this filter determines the extent of the overhead caused by the pitch search parallelization procedure.

Fortunately, a simple transformation can be used to parallelize the synthesis filter. Section 3.5.1 described the form of the all-pole synthesis filter, $\frac{1}{A(z)}$. The weighted synthesis filter has a similar form and is given by $H(z) = \frac{1}{A(\frac{z}{\lambda})}$. The direct-form implementation of this filter is depicted in figure 4.11. Using the principle of retiming, we migrate delays across the coefficient multipliers resulting in figure 4.12. A final application of this retiming procedure achieves the filter of figure 4.13. This filter has some nice properties for a pipelined implementation. The presence of the delays in the feedback accumulation path effectively decouples the calculation of successive output values. This effect is best illustrated by a resource allocation diagram. For simplicity, assume the number of pipeline stages in the processor is equal to the order of the synthesis filter (ten in this case). Furthermore, the length of the input vector to be filtered is 40 samples. Figure 4.14 shows the schedule for the original direct-form synthesis filter of figure 4.11. Notice that the calculation of each successive sample depends upon the results for the previous sample. This introduces "bubbles" into the pipeline and, therefore, the utilization factor is quite poor at only 52.6%. For the restructured filter, however, figure 4.15 shows that, aside from the time it takes to fill the pipeline at the beginning of the procedure and the time it takes to empty it at the end, the pipeline remains fully utilized. The pipeline utilization factor achieved for this scenario is 97.8%.

For the stochastic codebook searches this utilization can be further improved since not one but fourteen codevectors need to be filtered. These filtering operations can be overlapped, thus, occupying those triangular pipeline emptying and filling regions shown in the figure. The new utilization is 99.8%.

The $H(z)$ filtering process is the major contributor to the complexity of the stochastic codebook search procedures; however, the inner search loop, itself, also makes a significant contribution. Fortunately, this loop is based on a recursion similar in style to the pitch search recursion. Therefore, the same codebook subdivision process can be applied here in order to parallelize the search.

Clearly then, efficient parallel implementations can be developed for the pitch search, codevector filtering, and stochastic codebook search procedures. Furthermore, the required transformations are easy to derive and to implement.

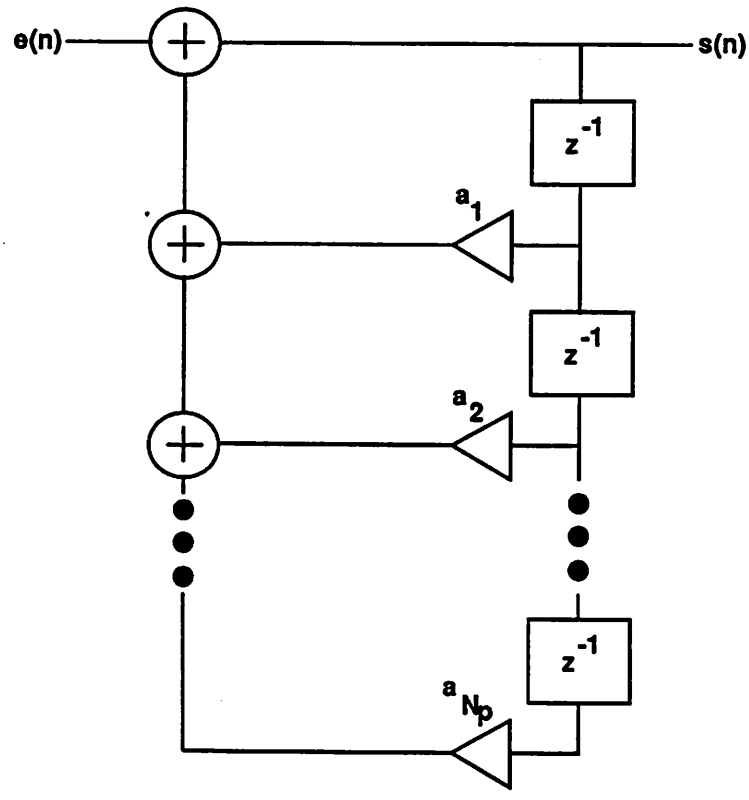


Figure 4.11: Direct-Form Synthesis Filter

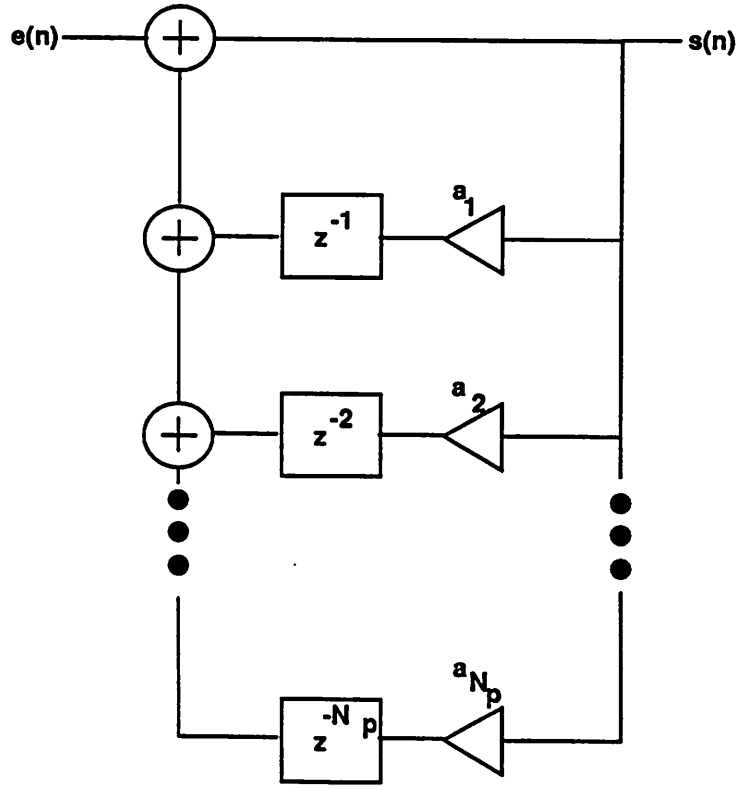


Figure 4.12: Synthesis Filter After First Retiming Step

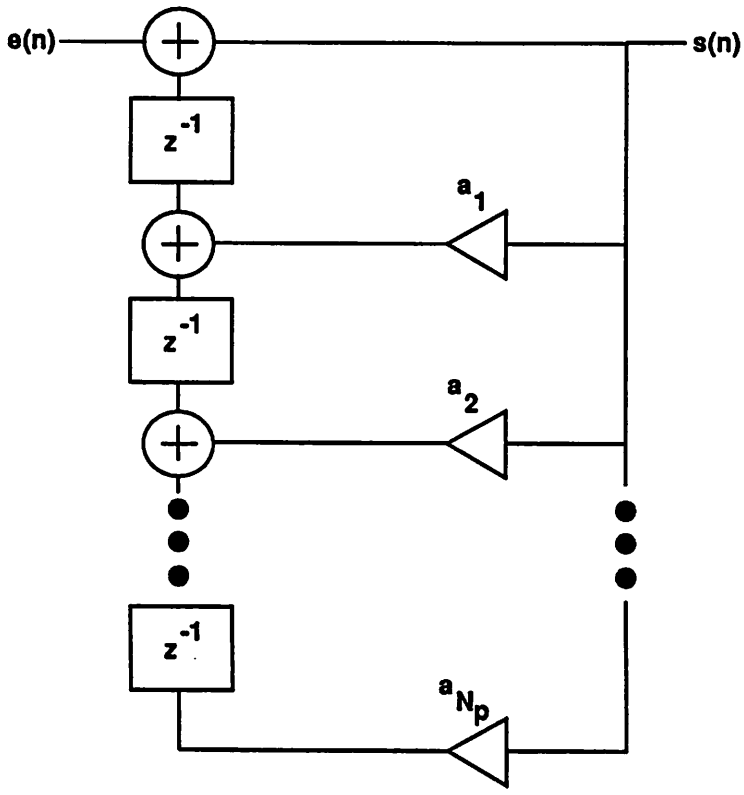


Figure 4.13: Synthesis Filter After Second Retiming Step

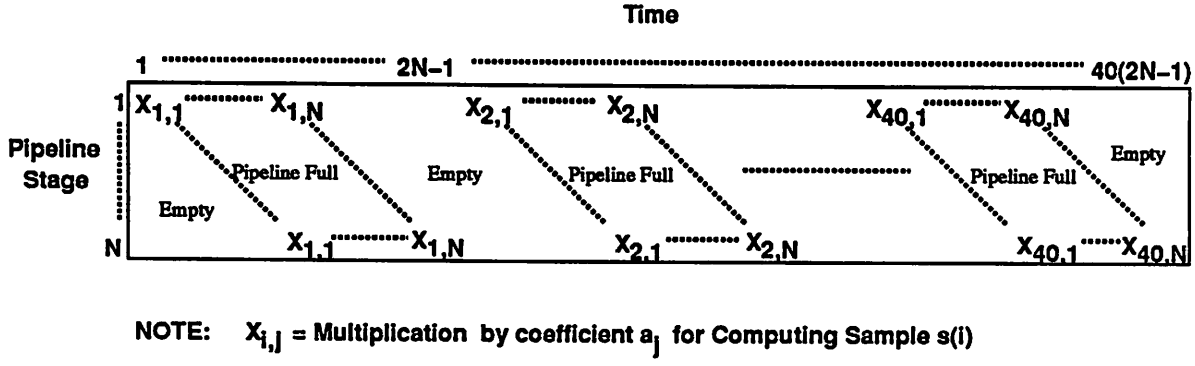


Figure 4.14: Resource Allocation Schedule for Direct-Form Pipelined Synthesis Filter

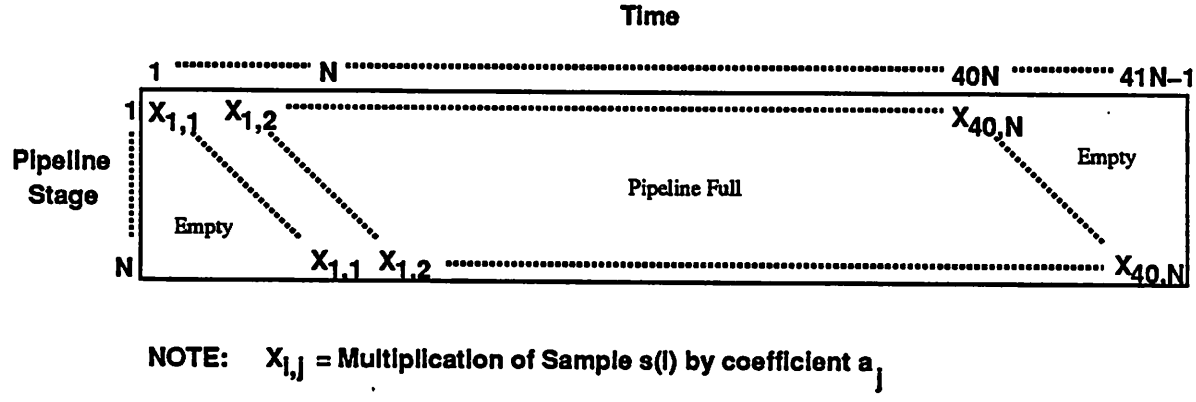


Figure 4.15: Resource Allocation Schedule for Restructured Pipelined Synthesis Filter

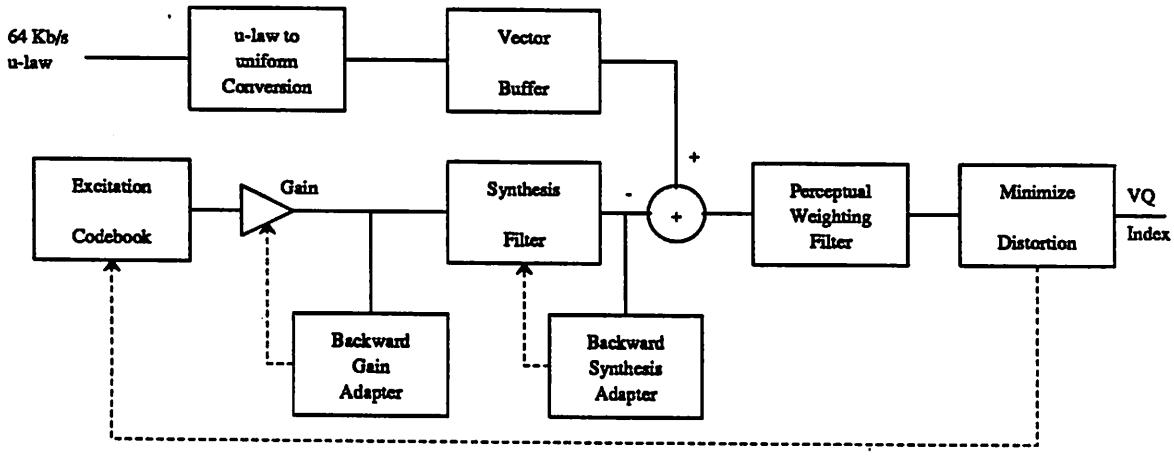
4.3 Comparative Analysis of LD-CELP Algorithm

The preceding detailed analysis of the Motorola VSELP algorithm covered the key issues involved in optimizing an algorithm for low-power implementation. Indeed, in many ways the VSELP algorithm is typical of DSP algorithms, and the optimizations applied to it can also be applied to other DSP applications. The analysis, then, of the AT&T LD-CELP [12] candidate algorithm will be undertaken in a comparative, rather than exhaustive, fashion. The overview of the algorithm will highlight key points unique to LD-CELP and will illustrate where it differs from traditional CELP as well as the VSELP algorithm. Furthermore, the subsequent analyses of computational complexity and suitability for fixed-point and parallel implementation will rely heavily on the preceding VSELP discussion.

4.3.1 Overview of Algorithm

Figure 4.16 shows a high-level representation of the LD-CELP algorithm. Whereas the VSELP algorithm achieves a channel stream of 8.0 Kbits/sec, the LD-CELP algorithm aims at only a 4:1 compression ratio with a 16 Kbit/sec channel stream. In lieu of the high compression, however, LD-CELP achieves the extremely low encoding delay of less than 2 ms. Indeed, LD-CELP is an acronym for low-delay CELP. In certain communications systems requiring echo cancellation, this low encoding delay is extremely desirable. For the purposes of this research, however, it is not a necessity and the LD-CELP algorithm will be analyzed purely on the basis of its other merits.

Coder



Decoder

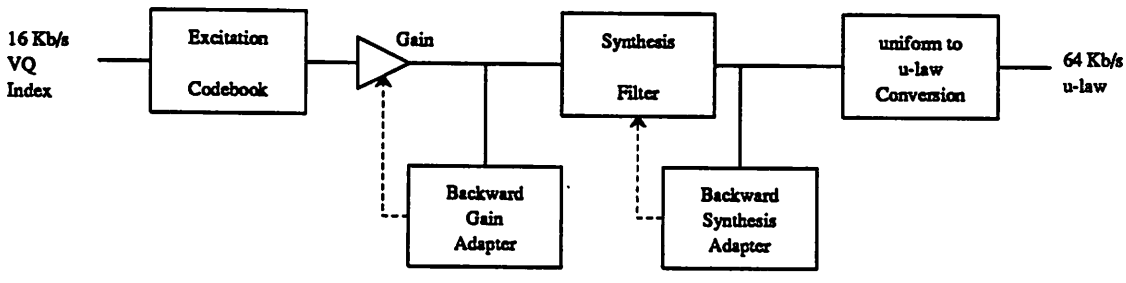


Figure 4.16: LD-CELP Speech Coder/Decoder

Coder

LPC Analyses A major difference between LD-CELP and conventional CELP coders (including VSELP) is that the AT&T algorithm relies on backward adaptation of LPC filter parameters. Traditional CELP coders analyze the input speech signal to determine appropriate LPC coefficients, which describe the various synthesis and perceptual weighting filters in the coder. These parameters are then transmitted to the decoder as side information in order to ensure that the coder and decoder filter configurations track one another. LD-CELP avoids this side-channel by using a backwards adaptation method to update the filter coefficients. Instead of analyzing the input speech to determine the appropriate predictor parameters, the algorithm analyzes the synthesized (decoded) speech waveform. Since this signal is available in identical form in both the coder and the decoder, filter updates can track on both ends. Excitation gains in the synthesis path are updated in a similar manner.

Another unique attribute of LD-CELP is its high-order LPC predictor. In contrast to VSELP and most CELP coders, the AT&T coder makes use of a 50th (rather than 10th) order LPC synthesis filter. This can be attributed to the fact that LD-CELP has no pitch filter in the synthesis path. Instead, the functions of the normally 10th order vocal tract filter and the long-term pitch filter are combined into this single high-order filter. This presents problems in both numerical precision and computational complexity that will be addressed shortly.

As mentioned briefly, the excitation gain is updated by a 10th order adaptive linear predictor. This predictor operates in the logarithmic domain and the LPC analysis can be accomplished by the standard Levinson-Durbin recursion operating on previous values of excitation gains.

Codebook Search The LD-CELP codebook search is based on a single codebook in the classic CELP tradition. Rather than being formed by linear combinations of basis vectors, the LD-CELP codevectors are individually stored in the excitation codebook. In order to achieve the low 2 ms encoding delay, the vector length is reduced from VSELP's 40 samples to a mere five samples. The codebook contains 1024 entries, resulting in a codebook index of 10 bits that must be transmitted to the decoder for each five samples of input speech. As with VSELP, the algorithm takes advantage of a zero-state synthesis filter, however, this

Operation	Operations Per 2.5 ms Frame	Mega-Operations Per Second (MOPS)
Comparison	17984	7.1936
MAC	13977	5.5908
$\pm$	5448	2.1792
$\times$	5184	2.0736
/	78	0.0312
log	4	0.0016
$\uparrow$	4	0.0016
Total	42679	17.1 MOPS

Table 4.5: Complexity of LD-CELP Coder

filter is now of 50th rather than 10th order. Again, as in VSELP, perceptual weighting is performed and a mean-square distortion measure is applied to the search procedure. Finally, the codebook vectors do not overlap, but are, instead, completely independent.

Decoder

Like the VSELP decoder, the LD-CELP decoder is a subset of the coder. Taking the received codebook index, it selects the appropriate excitation vector, gain-scales it, and passes it through the 50th order synthesis filter. Filter coefficients and excitation gains are updated in a backwards adaptive fashion, as in the coder.

4.3.2 Computational Complexity

Table 4.5 shows the computational breakdown of operations for the LD-CELP coder. The total number of operations required per second exceeds that of the VSELP coder. Considering their compression ratios, this is quite a surprising result. The VSELP coder achieves a factor of two greater compression than LD-CELP, yet it requires less operations per second to reach that end.

Decomposing the coder complexity into its dominant subcomponents reveals that the computational load is concentrated in two subblocks: the codebook search and the 50th order LPC analysis. The data for this result is shown in table 4.6. The significance of the computational requirements of the codebook search algorithm are, perhaps, not surprising. This situation arises in basically all CELP coders due to the analysis-by-synthesis method-

Task	% of Total Operations
50th Order LPC analysis	44%
Codebook Search	41%
Total	85%

Table 4.6: Breakdown of Dominant LD-CELP Coder Tasks

ology. The only point of interest lies in the fact that the codebook has no special structure like the VSELP codebook that can be exploited to reduce overall instruction counts. Thus, the LD-CELP algorithm opts for the more traditional, exhaustive approach to the codebook search.

The fact that the LPC analysis is a dominant component of computational load is probably a more interesting result. It can be directly attributed to the union of the pitch filter and the traditional synthesis filter into a single 50th order filter. As table 4.6 shows, performing the Levinson-Durbin recursive analysis required by this filter is not a trivial task. Moreover, the filtering operation itself requires significant computational resources since the synthesis filter contains 50 taps.

The decoder, although computationally simpler than the coder, requires some non-trivial resources (see table 4.7). Since coefficients are updated backwards adaptively in both coders, the LD-CELP decoder must also perform a 50th order LPC analysis, as well as a 10th order excitation gain LPC analysis. Thus, although no codebook search is required, a significant number of operations must still be performed. Of course, these operations are in addition to the traditional decoder functions of synthesizing the output speech based on the selected codebook excitation vector.

An overall comparison of the VSELP and LD-CELP algorithms reveals that even with a compression ratio two times less than the VSELP algorithm, LD-CELP requires 47% more operations per second. Thus, although the AT&T algorithm achieves high quality with low encoding delay, it does this at a significant price.

Operation	Operations Per 2.5 ms Frame	Mega-Operations Per Second (MOPS)
Comparison	9199	3.6796
MAC	7181	2.8724
$\pm$	4100	1.6400
$\times$	1284	0.5136
/	68	0.0272
log	4	0.0016
$\uparrow$	4	0.0016
Total	21840	8.74 MOPS

Table 4.7: Complexity of LD-CELP Decoder

4.3.3 Suitability for Low-Power Implementation

Algorithmic Concurrency

As we have seen, the operational requirements of the LD-CELP algorithm are significantly higher than those of VSELP. Throughout this paper, however, we have emphasized that for low-power implementations it is not necessarily the total instruction count that can be detrimental but, instead, the total *sequential* instruction count. If an algorithm can be sufficiently parallelized, the drawbacks of a high overall instruction count can be lessened.

In some respects, LD-CELP is ideally suited for parallel implementation. The vector quantized codebook consists of completely independent codevectors and, thus, the synthesis of these codevectors can occur simultaneously. This is the advantage of a non-structured codebook. For although the total instruction count is increased by the lack of an efficient recursive search procedure, the search can be easily implemented on a parallel architecture. Indeed, the codebook search algorithm requires no further parallelization – the concurrency already exists.

Not all of the algorithm, however, lends itself so readily to parallel implementation. The 50th order LPC analysis, for example, stubbornly resists attempts to map it to a parallel architecture. For the VSELP algorithm, too, the LPC analysis routine does not conform easily to a concurrent implementation; however, since the 10th order VSELP analysis occupies so little of the computing power of the processor this fact is of little import. The AT&T LPC analysis, however, is one of the two resource dominating calculations required

by the algorithm. Therefore, its inherently sequential nature is not a trivial matter.

Suitability for Fixed-Point Arithmetic

When considering suitability for low-power, fixed-point implementation, the LD-CELP LPC analysis is again a stumbling block. A 10th order Levinson-Durbin recursion presents precision difficulties in itself, but a 50th order version (almost) *requires* floating-point capabilities. Moreover, the *extreme* sensitivity of the 50th order synthesis filter pole positions to imprecisions in the direct-form LPC coefficients adds to these difficulties. Consequently, any implementation of the LD-CELP algorithm has little hope of utilizing a fixed-point or even block floating-point architecture. Clearly, the required floating-point hardware overhead would have quite a negative effect on power consumption.

4.3.4 Parallelization of Algorithm

As suggested above, parallelization of the 50th order LPC analysis procedure is a key stumbling block to a low-power implementation. This is due to the Levinson-Durbin recursion's strong resistance to parallel implementation. The basis for this resistance is two-fold. First, the algorithm is, as stated, a recursion with each successive iteration depending on the results of the previous stage. This makes pipelining or interleaving difficult. Furthermore, the individual operations comprising the recursion are mostly inner products, which, taken individually, are difficult to parallelize.

The Schur algorithm offers a solution to these difficulties. Like the Levinson-Durbin recursion, it calculates the reflection coefficients for the LPC synthesis filter; however, it requires no inner products and is easily implemented on a pipelined processor. The interested reader is referred to Haykin [14] for a pipelined implementation of the Schur algorithm.

Thus, by proper selection of the LPC analysis algorithms we have effectively parallelized its computation. This further illustrates the important influence algorithm selection can have on the ability to achieve a low-power implementation.

Unfortunately, the numerical precision issue is not abated by this technique. As in the Levinson-Durbin recursion, the high order of the required analysis virtually precludes fixed-point implementation. Thus, while the parallelization issue has a straightforward solution, the issue of floating-point hardware overhead remains unresolved.

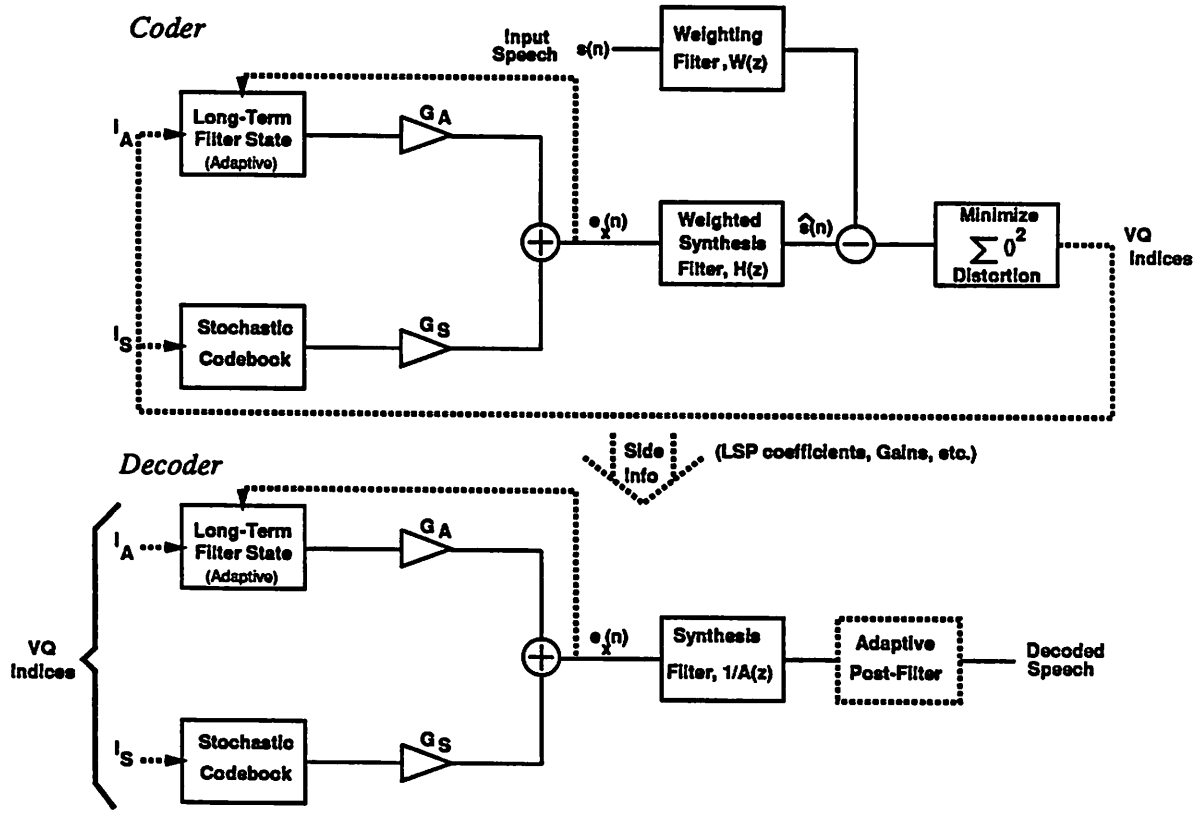


Figure 4.17: DoD CELP Speech Coder/Decoder

4.4 Comparative Analysis of DoD CELP Algorithm

The final candidate algorithm for the low-power, high quality speech coder is the Department of Defense's (DoD) CELP 3.1 coder [11]. As before, the analysis of this algorithm is performed on a comparative basis with the previous two algorithms.

4.4.1 Overview of Algorithm

Figure 4.17 shows a schematic view of the DoD algorithm. This particular version of the DoD CELP coder compresses the 64 Kbit/sec input speech waveform into a channel of 9.6 Kbits/sec. The frame length of 15 ms determines the LPC analysis rate and is divided into four subframes during which separate codebook searches are performed.

Codebook Search

DoD CELP differs from the VSELP algorithm mainly in the structure of the excitation codebook. DoD CELP relies on two rather than three codebooks. The first is an adaptive codebook identical in spirit to the one used by VSELP to introduce the pitch component of the excitation. Like the VSELP algorithm, the DoD coder employs an end-point recursion technique to search this adaptive codebook. The DoD algorithm slightly extends the utility of this pitch codebook, however, by allowing non-integer pitch lags. These are achieved by interpolating the contents of the pitch codebook to the desired non-integer lag value.

The codebook search procedure also differs for the stochastic codebook. For DoD CELP there is one stochastic codebook rather than two. The codebook contains 512 codevectors and has a novel structure. This structure allows computational reductions similar to those achieved by VSELP. The codebook structures, however, are drastically different. The sample values in the DoD codebook are ternary-quantized, Gaussian, unit-variance white-noise. The fact that all samples are either -1, 0, or +1 is used to reduce computational complexity; however, this is not where the major savings occurs. In addition to being a ternary codebook, successive codevectors in the codebook overlap completely save for two samples at each end. Since the codevectors are white, this causes no loss in vector quantization efficiency; however, this structure can now be exploited using an end-point recursion technique to reduce the total instruction count.

LPC Analysis

The DoD algorithm also differs in the form of the LPC coefficients used. Whereas, both VSELP and LD-CELP rely on the reflection and direct-form coefficients, the DoD CELP algorithm makes use of line spectrum pair (LSP) coefficients. The choice of these coefficients revolves around their behavior under linear interpolation. Since, as in the previous algorithms, LPC analysis takes place only once per frame, interpolation must be used to arrive at the coefficient values to be used for each subframe. Reflection and direct-form coefficients have relatively poor linear interpolation characteristics [17]. LSP coefficients, on the other hand, introduce little distortion due to linear interpolation. Unfortunately, calculation of LSP coefficients is more complex than that of direct-form coefficients and involves non-primitive operations including sinusoids and cosinusoids. Fortunately, the analysis is

Operation	Operations Per 15 ms Frame	Mega-Operations Per Second (MOPS)
Comparison	147479	9.8319
MAC	128969	8.5979
×	46483	3.0989
±	41507	2.7671
cos	4520	0.3013
/	742	0.0495
↑	187	0.0125
sin	145	0.0097
Total	370032	24.7 MOPS

Table 4.8: Complexity of DoD Coder

Task	% of Total Operations
Codebook Search	67%
Pitch Search	20%
Total	87%

Table 4.9: Breakdown of Dominant DoD Coder Tasks

10th order unlike the 50th order LD-CELP algorithm.

4.4.2 Computational Complexity

As with traditional CELP coders, the DoD coder's computational complexity is dominated by the adaptive and stochastic codebook searches. Table 4.8 shows operation counts for the coder as a whole, while table 4.9 illustrates the dominance of the codebook searches relative to the overall computational load. Therefore, as with the VSELP coder, low-power optimizations should be focused on the codebook and pitch search procedures.

Like the VSELP algorithm, the decoder requires relatively few operations. Unlike the AT&T coder, coefficients are passed to the decoder on a side-channel and no expensive backwards adaptations are required. Table 4.10 summarizes the operational requirements of the decoder.

Overall, the complexity of the DoD coder (as given by instruction counts) is similar to the AT&T coder; however, both of these coders are significantly more complex than the Motorola VSELP algorithm. Furthermore, neither algorithm achieves the compression

Operation	Operations Per 15 ms Frame	Mega-Operations Per Second (MOPS)
Comparison	5402	0.3601
MAC	4680	0.3120
$\pm$	2669	0.1779
$\times$	2430	0.1620
/	355	0.0237
$\uparrow$	88	0.0059
cos	50	0.0033
Total	15674	1.04 MOPS

Table 4.10: Complexity of DoD Decoder

available from VSELP (although the DoD algorithm comes close).

4.4.3 Suitability for Low-Power Implementation

Algorithmic Concurrency

As with the VSELP algorithm, the use of end-point recursion in the codebook search nullifies the concurrency inherent in traditional CELP coders. The discussion of this issue for the DoD algorithm is identical to that presented for VSELP and the reader should refer to section 4.2.3 for a detailed account.

Suitability for Fixed-Point Arithmetic

The issue of suitability for fixed-point implementation arises here once again in relation to the use of the Levinson-Durbin recursion. Unlike VSELP, which is designed to utilize a fixed-point algorithm called FLAT, the DoD CELP algorithm is based on the straightforward Levinson-Durbin recursion. Not only does this introduce dynamic range and numerical precision difficulties but, in addition, the resulting direct-form coefficients are converted to LSP coefficients prior to interpolation. Following interpolation they are converted back to direct-form coefficients once again. This procedure tends to foster accumulation of numerical errors and is best implemented on a floating-point processor.

4.4.4 Parallelization of Algorithm

Since the DoD algorithm relies on end-point recursion for the codevector filtering required in both of its codebook searches, we can fall back on the same techniques developed to handle this difficulty in the VSELP algorithm. Specifically, the codebook searches can be subdivided into several independent and concurrently executable searches involving subsections of the codebooks. Moreover, the codevector filtering process can be parallelized through the retiming transformations introduced in the VSELP analysis. Again, for details of these techniques the reader should refer to the appropriate VSELP discussions.

4.5 Conclusions Regarding Low-Power Speech Coding

The issues involved in the selection of a candidate speech coding algorithm for a low-power host architecture include not only the usual coding quality, complexity, and bit-rate concerns, but also issues such as algorithmic concurrency, suitability for parallelization, and feasibility of fixed-point implementation. The three algorithms that have been analyzed in this report represent a nice sampling of the domain of CELP-based coding algorithms. Since all three coders result in high quality decoded speech (MOS $\geq$ 4.0), the algorithm selection does not focus on this point. Instead suitability for low-power implementation dominates the selection criteria.

In this respect, Motorola's VSELP algorithm is the logical choice. For while not only achieving the highest compression ratio of the three coders, it also has by far the minimum computational requirements. As has been repeatedly emphasized, however, absolute instruction count is not necessarily the key issue for low-power implementation. Instead, it is the sequential instruction count after the algorithm has been parallelized that is of critical importance. While the AT&T algorithm is perhaps the most readily parallelized, we have shown that the VSELP algorithm can be parallelized with only slight modifications.

Furthermore, the VSELP designers' consideration of fixed-point numerical precision issues in their specification of the algorithm make it even more attractive from a low-power perspective. Finally, VSELP has been accepted by the Telecommunications Industry Association as the North American digital cellular standard. Thus, a chip based on this algorithm would find wide acceptance. For all of these reasons, the VSELP algorithm is the most promising for a low-power VLSI implementation.

Chapter 5

Directions for Future Work

This paper has presented several strategies for the low-power CMOS design of a speech coder. It has raised issues relating to power consumption in all levels of design from processing to algorithmic. These concepts, though plausible in theory, must be demonstrated in practice to achieve widespread acceptance. Furthermore, the original motivation of the low-power speech coder was its intended use in a portable, multi-media personal communications terminal. Therefore, the most immediate goal of future research will be to implement the coder as a low-power ASIC (application-specific integrated circuit).

As occurs in any implementation phase, issues will arise suggesting modifications to the low-power methodologies and architectures developed in this paper. Moreover, it is likely that new, additional techniques for low-power will be developed. There is never only one solution to a problem, and hopefully through continued research in the area of low-power many important new concepts will be developed. The implementation of the coder chip will propel this process.

Another future topic of related importance is the development of a CAD framework for low-power VLSI. One of the key components of this CAD system would be a series of hierarchical power estimation tools. Here, hierarchical refers to a series of power tools that follow the design process from the algorithmic to physical levels providing continually refined estimations of power consumption. At the algorithmic level, power bounds could be based simply on the type, quantity, and timing constraints of the required operations. As the design traverses to lower levels, progressively more detailed models of circuit power consumption could be applied.

Another important part of a complete CAD framework might be a low-power VLSI

synthesis system. This would require the development not only of the traditional silicon compilation software and low-power cell libraries, but also of programs to seek out, create, and exploit concurrencies in the target algorithms. This is by no means a trivial task, as parallel computing professionals can attest. It could, however, be an important addition to a designer's low-power arsenal.

Finally, a future goal of this research could be to generalize the ideas presented in this report and to develop a fully programmable low-power DSP chip with accompanying compiler. The strategies presented in this paper are, for the most part, quite general and it is likely that they could be extended to a general-purpose DSP processor. Once again this would require an extremely powerful compiler that could exploit algorithmic concurrencies in order to achieve efficient implementations on the low-power DSP processor.

Chapter 6

Conclusions

This report has presented a detailed analysis of several high quality, low bit-rate speech coding algorithms with respect to implementation on a low-power hardware platform. In this vein, several techniques for low-power VLSI were presented. Among the most important was the concept of lowering supply voltages in order to achieve a quadratic power reduction, while exploiting parallelism to compensate for circuit delay increases.

This basic philosophy led to several criteria for low-power speech coding algorithms. In particular, parallelizability and low sequential instruction count were key issues. It is these properties that would allow the algorithm to be efficiently implemented in real-time on a low-power ASIC.

Issues such as parallel implementations of recursions and high-order LPC analyses also became important. Recursions, which increase algorithmic efficiency in a sequential processor, can actually be detrimental to a parallel architecture. Fortunately, these recursions can be parallelized in a straightforward manner with little operational overhead. Methods of implementing high-order LPC analyses on parallel processors include the Schur algorithm, which maps easily to a pipelined architecture.

With the addition of technology scaling and circuit and logic level techniques, power savings can be even further increased. Techniques such as reduced voltage swings on inter-block clock lines and inter-chip I/O lines, coupled with novel use of sense-amp circuitry could dramatically improve power savings. Furthermore, an intelligent power management strategy, including idle processor shutdown and recognition of periods of input speech silence could also contribute significantly to the low-power speech coder implementation.

The amalgamation of all of these issues resulted in a high-level proposal for an

ASIC speech coding architecture. Based on a multi-processor approach, the architecture would contain a small Von Neumann processor to handle the non-parallel, control-intensive portions of the algorithm, coupled with a highly pipelined co-processor to handle the massive parallel computational requirements.

All of these notions will be applied to the implementation of the low-power CELP coding ASIC and, indeed, to the entire personal communications system. The result should be a portable, multi-media terminal capable of functioning at power consumption levels several times below those possible with conventional techniques. Moreover, the results and knowledge gained in this endeavor will enjoy almost direct applicability to other DSP applications constrained to low-power operation.

Bibliography

[1] R. K. Watts, ed., *Submicron Integrated Circuits*, John Wiley & Sons, NY, 1989.

[2] H. J. M. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits," *IEEE Journal of Solid-State Circuits*, Vol. SC-19, August 1984.

[3] H. B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, Menlo Park, CA, 1990.

[4] A. P. Chandrakasan, S. Sheng, and R. Brodersen, "Design Considerations for a Future Portable Multimedia Terminal," *Proceedings of Second Rutgers Workshop on Third Generation Wireless Information Networks*, NJ, October 1990.

[5] K. Yano, et. al., "A 3.8 ns CMOS 16 $\times$ 16 Multiplier Using Complimentary Pass Transistor Logic," *Proceedings of the IEEE*, September 1987.

[6] D. A. Hodges and H. G. Jackson, *Analysis and Design of Digital Integrated Circuits*, McGraw-Hill, NY, 1988.

[7] S. Furui, *Digital Speech Processing, Synthesis, and Recognition*, Marcel Dekker, inc., NY, 1989.

[8] L. Rabiner and R. Schafer, *Digital Processing of Speech Signals*, Prentice-Hall, NJ, 1978.

[9] N. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, NJ, 1984.

[10] I. Gerson and M. Jasiuk, "Vector Sum Excited Linear Prediction (VSELP) Speech Coding at 8 Kbps," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1990.

[11] J. Campbell, Jr., V. Welch, and T. Tremain, "An Expandable Error-Protected 4800 Bps CELP Coder," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1989.

[12] J-H Chen, "High-Quality 16 Kb/s Speech Coding with a One-Way Delay Less Than 2 ms," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1990.

[13] F. Itakura and S. Saito, "Analysis Synthesis Telephony Based on the Maximum Likelihood Method," *Proceedings of the 6th international Conference on Acoustics*, 1968.

[14] S. Haykin, *Modern Filters*, MacMillan Publishing Company, NY, 1989.

[15] M. Schroeder and B. Atal, "Code-Excited Linear Prediction (CELP): High Quality Speech at Very Low Bit Rates," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1985.

[16] E. Lee, *A Coupled Hardware and Software Architecture for Programmable Digital Signal Processors*, Ph.D. Dissertation, University of California, Berkeley, 1986.

[17] S. Saito and K. Nakata, *Fundamentals of Speech Signal Processing*, Academic Press, FL, 1985.