ASIC IMPLEMENTATION OF EFFICIENT LINE SPECTRAL FREQUENCY COMPUTATION FOR SPEECH CODING APPLICATIONS

David L. Reynolds, Linda M. Head and Ravi P. Ramachandran

Department of Electrical and Computer Engineering, Rowan University, Glassboro, NJ 08028
dreynolds@ieee.org, head@rowan.edu, ravi@rowan.edu

ABSTRACT

One of the more computationally intensive portions of speech coding algorithms using linear predictive (LP) methods is the calculation of line spectral frequencies (LSFs) from the predictor coefficients. Methods for the efficient computation of LSFs have been developed. A very large scale integration (VLSI) design implementing one such method is presented. The architecture is designed to be optimized for speed and area and is suitable for integration into larger speech coding systems.

1. INTRODUCTION

Vocoders are algorithmic constructs which encode speech for digital transmission over band-limited communication channels. Vocoders based on linear-predictive (LP) analysis are a popular implementation for modern communications systems, many of which are embedded systems such as cellular telephones, digital radios and cryptographic devices. LP vocoders are model based and encode the vocal tract information representing the spectral envelope of the encoded speech frame by means of the LP coefficients, which provide a compact representation of the digitized speech. The LP residual, which contains the pitch information, is also encoded. Because LP coefficients are not conducive to low quantization distortion, they are generally converted to an equivalent parameter set known as line spectral frequencies (LSFs)[1]. Although ideal for quantization and transmission, computation of LSFs from LP coefficients requires the isolation of high order (typically 10th or 12th for practical systems) polynomial roots for each frame of digitized speech. Although generalized algorithms for performing this function are inefficient, an efficient algorithm for LSF computation based on Chebyshev polynomials was previously devised [3] and we recently proposed a VLSI architecture for the implementation of this algorithm[2]. In this paper we present a VLSI design implementing this algorithm suitable for incorporation into larger speech coding systems.

2. CHEBYSHEV POLYNOMAL TECHNIQUE

The LSFs of a given speech frame are determined by the roots of the symmetric and antisymmetric polynomials which can be computed from the LP coefficients. The algorithm described in [3] recognizes inherent properties of the symmetric and antisymmetric polynomials and leverages these properties to convert them into Chebyshev polynomials. Since the roots of the symetric and antisymmetric polynomials lie on the unit circle, when converted to Chebyshev representations, the roots are mapped onto the linear interval [-1,1] as the zero crossings of the polynomial functions. This mapping preserves the interlacing property of the polynomial functions, an important aspect of our computational approach. The core aspects of the algorithm are the evaluation of the Chebyshev polynomials and determining the location of each zero crossing. Chebyshev polynomials may be efficiently computed using Clenshaw's recurrence formula [4]. The specific steps of the algorithm may be summarized as follows[3]:

1. Given the LP coefficients, compute the symmetric and antisymmetric polynomials.
2. Deflate these polynomials by the trivial roots at -1 and +1.
3. Rearrange the deflated polynomial coefficients into the corresponding Chebyshev form.
4. Evaluate the Chebyshev polynomials over the interval -1 to +1 looking for zero crossings.
5. When a root is found, rescan the local region with finer resolution to isolate the location with more precision. Linear interpolation is performed to further pinpoint the root location.
6. Switch to the other polynomial (taking advantage of the interlacing property of the roots) and continue locating roots.
7. When all root locations are found, convert them to LSFs by computing the arccosine of each zero crossing.

These algorithmic steps lend themselves to decomposition and individual implementation. The approach taken for this design was the functional decomposition of the algorithm into specific algorithmic blocks. Design was performed exclusively in VHDL.

3. FUNCTIONAL DECOMPOSITION

The first step in the design of the VLSI architecture is functional decomposition of the algorithm into individual entities for hardware implementation. This approach also allows easy isolation of individual portions of the design for testing and validation. The design is decomposed into six major algorithmic blocks.

3.1. Symmetric and Antisymmetric Polynomial Computation

The input to the system which results in the computation of LSFs is the vector of LP coefficients in 32-bit IEEE 754 floating point format with an 8-bit exponent, 23-bit mantissa and sign bit. Computation of the symmetric and antisymmetric polynomial coefficients given the LP coefficients is fairly straightforward. If $A(z)$ represents the all-pole LP filter, the symmetric and antisymmetric polynomials $P(z)$ and $Q(z)$ may be computed as follows:

$$P(z) = A(z) + z^{-(P+1)}A(z^{-1}) \quad (1)$$

$$Q(z) = A(z) - z^{-(P+1)}A(z^{-1}) \quad (2)$$

Algorithmically, this is a simple matter of addition and multiplication of the coefficients. To achieve this, a state machine is implemented in the VHDL entity *atopq*. Because virtually all of the algorithmic constructs iterate through coefficients, they share similar state machine structures. The coefficients are iterated through and are presented to ports which are connected to external floating point arithmetic entities shared by all algorithmic blocks. Because this design is intended as a core that will be embedded into larger systems, simple floating point constructs have been employed. It is assumed that a complete vocoder implementation would have substantial resources available for floating-point arithmetic that could be shared among the cores. The floating point elements used are combinatorial and may limit overall throughput of the design. In addition to the input vectors, *atopq* is provided with a clock, as well as signals for start and to flag completion. The start and completion signals are used to integrate the *atopq* entity into the overall design. An additional selector signal is provided to indicate whether the $P(z)$ or $Q(z)$ polynomial has been presented to the entity.

3.2. Polynomial Deflation

The entity *polydef* is used to remove the trivial roots at $z = +1$ and $z = -1$ from the symmetric and antisymmetric polynomials. This can be accomplished by the simple addition and subtraction of coefficients. Again, because we are iterating through the coefficients in question and performing floating point operations on them, a state machine transitions through each step of the process. Floating point operands and results are presented on input and output ports of the entity. Like *atopq*, a clock, start and completion flag are used. A selector signal is used here as well to select which trivial root is to be removed.

3.3. Conversion to Chebyshev Polynomials

Given the symmetric and antisymmetric polynomials, it is necessary to compute the Chebyshev polynomial coefficients. There is an inherent symmetry in the coefficients of the original polynomials, therefore only half of them are necessary to compute the resulting Chebyshev polynomials. The *chebform* entity performs this computation. The inputs are the deflated $P(z)$ and $Q(z)$ polynomial coefficients in their 32-bit IEEE 754 format. Again, a state machine iterates through the coefficients. The Chebyshev polynomial coefficients are computed by multiplying the input polynomial coefficients by two and arranging them in the appropriate order. Other interface signals include clock, start and completion flags.

3.4. Evaluation of Chebyshev Polynomials and Root Isolation

The most important operation in the algorithm is the evaluation of the Chebshev polynomials over the linear region [-1,1] and isolating the zero crossings. Two entities fulfill this function, *rootfinder* and *clenshaw*. The *clenshaw* entity is used to efficiently evaluate the Chebyshev polynomials at a given location on the linear region. The input to the *clenshaw* entity are the Chebyshev polynomial coefficients and the x location on the linear region. A small state machine iterates through the evaluation steps of the Clenshaw recurrence formula. The result, y, is returned.

The algorithm dictates that the two Chebyshev polynomials be evaluated over the linear region [-1,1]. A coarse increment of x is used to obtain an approximate location of each root, followed by a more precise scan of the local region using a finer increment. Linear interpolation further isolates the precise root location. The interlacing property of the roots is preserved in the Chebyshev domain, so once a root of one Chebyshev polynomial is found, it is known that the next root will be a root of the other. This increases the efficency of evaluation.

The *rootfinder* entity contains the most complex state machine of all entities. Each value of x is determined. The Chebyshev polynomial is evaluated by presenting x and the polynomial coefficients to the *clenshaw* entity.

State transitions are made based on each result. A zero crossing is located by examination of the sign bit in the IEEE 754 floating point representations of the result obtained from the *clenshaw* entity. This avoids a full 32-bit floating point magnitude comparison, which is the practice generally employed in software implementations of the algorithm. Once an approximate root location is found, the increment and location are set to perform a fine resolution rescan of the area. A different portion of the state machine evaluates the linear interpolation.

As with the other major algorithmic blocks, the *rootfinder* entity is provided with a clock and start signal, and asserts a completion signal when it completes the scan of the [-1,1] region. Outputs are the x location of each zero crossing.

3.5. Computation of Arccosine

A Taylor series expansion of four terms is used to compute the LSFs from x locations. Again, a state machine is used. The x locations are presented in IEEE 754 floating point format to the entity. The steps of the Taylor series expansion are decomposed into the various states. Multiplications and additions are presented to the external floating point adder and multiplier as required. While the design described limits evaluation to four terms, the Taylor series expansion may be easily expanded should additional precision be required.

4. TOP LEVEL DESIGN

The entities described are assembled into a top level architecture by cascading them in the appropriate order. The floating point adder, multiplier and divider are treated as global resources and connected to the individual algorithmic blocks on a simple bus. Figure [1] shows the top level architecture of the design. The sequential nature of the design can clearly be seen. The input vectors of LP coefficients is presented to the input of the system. Each entity performs the required operations on the data and asserts the *DONE* flag to begin processing of the next stage. Each stage corresponds to a step in the algorithm.

5. SIMULATION RESULTS

Simulation of the source VHDL for each entity has been performed. Test benches have been designed to exercise the function of each entity using LP coefficient data derived from the TIMIT database. The LSFs resulting from given input LP coefficients have been validated by a software implementation of the algorithm. Figure [2] shows the computation of Chebyshev coefficients from an input vector of symmetric polynomial coefficients. This operation is accomplished in 11 clock cycles, or approximately 32ps from the assertion of the *START* signal until completion.

Of particular interest is the performance of the circuit including post-synthesis performance and parasitics. Synthesis of individual entities have been performed using Leonardo Spectrum and the MOSIS AMI05 synthesis library. VHDL models of the resulting circuits have been obtained and simulation performed at the gate level. These results reflect the timing performance of the primitives extracted from the AMI05 library during synthesis.

IC layout is currently in progress. Following layout, parasitic capacitances may be extracted and additional simulation performed. Based on results to date, it is expected that throughput of the design will far exceed the maximum input data rate for typical speech coding applications. Recall, however, that this design computes LSFs only and does not encompass LP analysis which must also be performed on each frame of input speech data.

6. SUITABILITY FOR TARGET APPLICATIONS

As mentioned previously, the design is intended for use in larger speech coding systems for integration in embedded systems. In such environments, power, space and memory are at a premium. The implementation of the algorithm addresses these concerns in several ways.

6.1. Resource Conservation

The algorithm of Kabal and Ramachandran is unique in that it does not require prior computation or storage of trigonometric functions[3]. Therefore, significant amounts of chip space are spared. The implementation of LP speech coding systems may be based on larger, general purpose processor cores or systems. In contrast to systems such as these, only small amounts of register storage are required by the VLSI implementation described. One disadvantage of the approach described is that it is necessary to determine the system order during the design phase. A software-centric approach would allow easy adaptation to lower or higher order systems as required.

6.2. Power Conservation

Efficient use of power is of increasing importance in modern embedded systems. Common items such as cellular telephone have made significant advances in terms of battery life. Continual pressure to increase battery life on such platforms while striving for decreased size makes design for power conservation critical. Thermal considerations are a related, and equally important factor in embedded systems such as these[7].

There are various techniques used for reducing the power requirements of digital circuitry. One common example is supply voltage scaling, which recognizes power dissipation due to switching effects decreases strongly with supply voltage[7][8]. Because simply decreasing the supply voltage will increase switching times[7], care must be taken that the maximum operational speed of the circuit is not reduced in excess of what is appropriate for the application. There are, however, other approaches to the reduction of switching losses which may be investigated. A simple technique is to ensure that no portions of the circuit are unnecessarily clocked. In our design, the sequential nature of the algorithm allows us to ensure that entities not involved in the current algorithmic step are not clocked. Also, examination of the sign bit in the IEEE 754 floating point representation for detecting a zero crossing causes significantly less switching activity than a full 32-bit floating-point magnitude comparison. This is important considering the large number of comparisons required for each frame of speech.

Independent of other design decisions, reduced switching losses are achieved simply through the increased efficiency of the algorithm. Should a generalized root-finding algorithm be implemented, the number of operations, and thus the amount of circuit switching, is significantly higher. Likewise, the complexity of general purpose processor or DSP systems would carry more overhead in terms of power requirements.

7. SUMMARY AND FUTURE WORK

Currently, design of an 8-bit bus interface wrapper is underway to allow packaging of the design as a stand alone chip. Following completion, layout of the design will be completed and the design submitted to MOSIS for fabrication. Post synthesis results indicate that the design is viable and well suited for integration in larger LP speech coding systems. Numerical performance of the design is driven primarily by the precision of the floating point entities used, the length of the Taylor series expansion for arccosine computation, and the choice of fine resolution increments in the isolation of zero crossings in the Chebyshev domain. Following fabrication, it is planned to develop a test platform to allow the design to be exercised with large amounts of LP coefficients from the TIMIT database. The results will be compared with the corresponding LSFs and the outputs correlated with pre- and post-synthesis results.

8. REFERENCES

1. R.P. Ramachandran, M.M. Sondhi, N. Senshadri and B.S. Atal, "A Two Codebook Format for Robust Quantization of Line Spectral Frequencies", *IEEE Transactions on Speech and Audio Processing*, Vol. 3, No. 3, pp. 157-168, May 1995.
2. D.L. Reynolds, L.M. Head and R.P. Ramachandran, "VLSI Architecture for the Efficient computation of Line Spectral Frequencies", *IEEE International Conference on Circuits and Systems*, Vol. 3, pp. 718-721, May 2003.
3. P. Kabal and R. P. Ramachandran, "The Computation of Line Spectral Frequencies Using Chebyshev Polynomials", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-34, No. 6, pp. 1419–1425, December 1986.
4. W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.
5. K. J. Byun, M. Hahn and K. S. Kim, "Implementation of 13kbps QCELP Vocoder ASIC", *The First IEEE Asia Pacific Conference On ASICs*, pp. 258–261, 1999.
6. K. C. Chang, *Digital Systems Design with VHDL and Synthesis*, IEEE Computer Society Press, 1999.
7. W. Athas, *Low-Power VLSI Techniques for Applications in Embedded Computing*, ÎEEE Allesandro Volta Memorial Workshop on Low-Power Design, pp. 14-22, 1999.
8. T. Kuroda, "Low-Power, High-Speed CMOS VLSI Design"', *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 310-315, 2002.

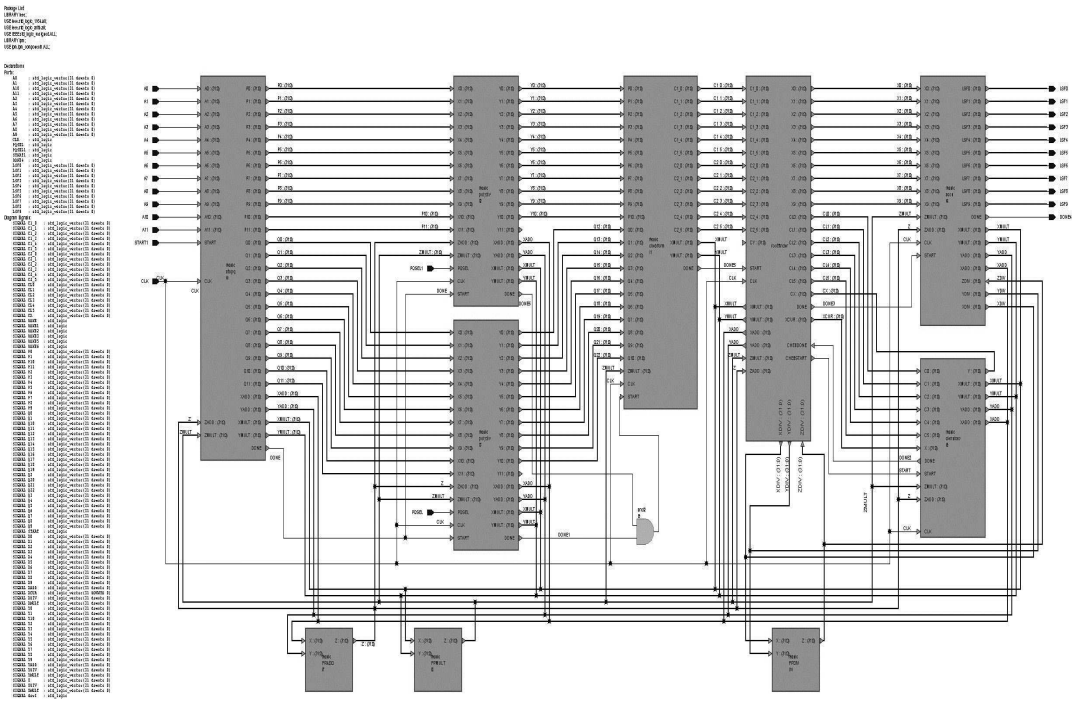

Figure 1: Top Level Block Architecture

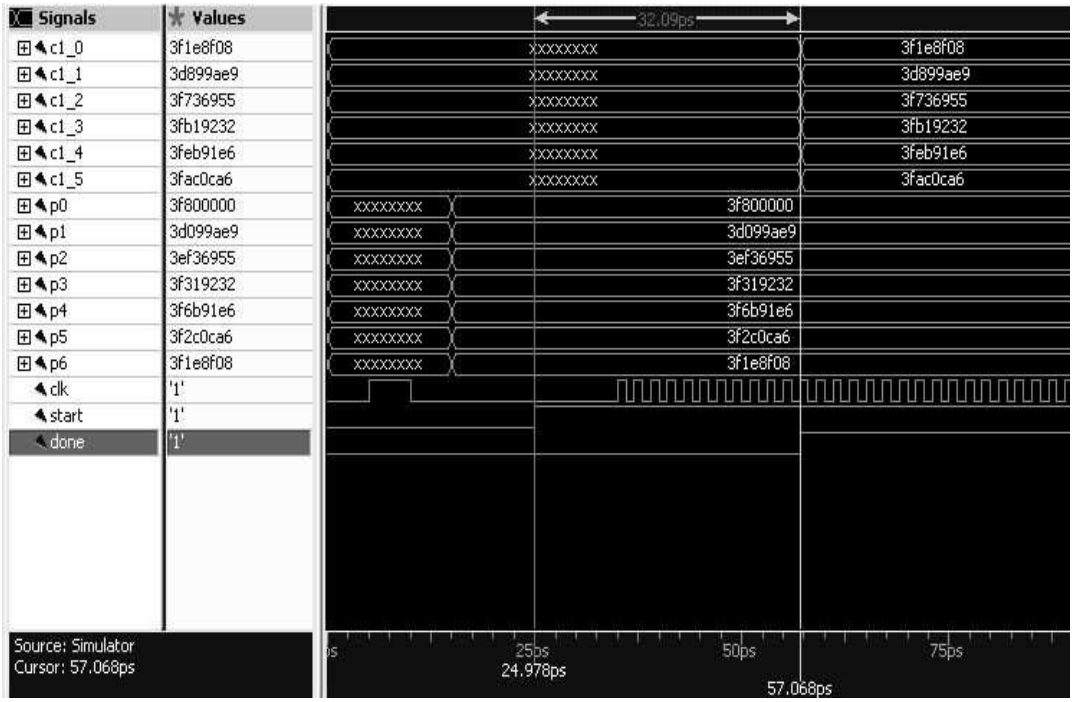

Figure 2: Simulation of *chebform*