VLSI ARCHITECTURE FOR THE EFFICIENT COMPUTATION OF LINE SPECTRAL FREQUENCIES

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

In applications where cost, power and size are of critical importance, the use of application specific very large scale integration (VLSI) speech coding systems have certain advantages over the use of general purpose digital signal processors. The optimization of such VLSI systems can further reduce the complexity and power requirements while enhancing performance. The use of linear predictive (LP) methods for speech coding is common practice in such systems. However, there are several parts of LP based speech coding algorithms which exhibit fairly high levels of computational complexity.

One such example is the computation of line spectral frequencies (LSFs) given the linear predictive coefficients. This involves the isolation of polynomial roots which is a complex and resource consuming undertaking. A method to isolate the line spectral frequencies efficiently has been proposed in [1] and is the algorithm implemented in this VLSI design. This method starts with the P coefficient LP filter given by

$$A(z) = 1 - \sum_{k=1}^{P} a(k) z^{-k} \tag{1}$$

where $a(k)$ are the LP coefficients. The first algorithmic step is to compute the symmetric polynomial $F_1(z)$ and the antisymmetric polynomial $F_2(z)$ from $A(z)$. The corresponding equations are

$$F_1(z) = A(z) + z^{-(P+1)} A(z^{-1}) \tag{2}$$

$$F_2(z) = A(z) - z^{-(P+1)} A(z^{-1}) \tag{3}$$

The roots of $F_1(z)$ and $F_2(z)$ are on the unit circle, are simple and interlace. The LSFs are the angles of the roots whose imaginary part is positive. For practical applications, the order P is typically 10 or 12, making the isolation of the polynomial roots an arduous task given the high resource cost in most VLSI systems.

Two trivial roots at $z = \pm 1$ are first removed using a simple difference equation [1] that essentially accomplishes polynomial deflation. The remaining roots must be found explicitly. When P is even, we define the deflated polynomials as $G_1(z) = F_1(z)/(1+z^{-1})$ and $G_2(z) = F_2(z)/(1-z^{-1})$. When P is odd, we define the deflated polynomials as $G_1(z) = F_1(z)$ and $G_2(z) = F_2(z)/(1-z^{-2})$. Suppose the orders of $G_1(z)$ and $G_2(z)$ are $2M_1$ and $2M_2$ respectively. When P is even, $M_1 = M_2 = P/2$. When P is odd, $M_1 = (P+1)/2$ and $M_2 = (P-1)/2$. Note that $G_1(z)$ and $G_2(z)$ have an inherent coefficient symmetry [1]:

$$\begin{aligned} G_1(z) &= 1 + g_1(1)z^{-1} + \ldots + g_1(M_1)z^{-M_1} + \ldots \\ &+ g_1(1)z^{-(2M_1-1)} + z^{-2M_1} \end{aligned} \tag{4}$$

$$\begin{aligned} G_2(z) &= 1 + g_2(1)z^{-1} + \ldots + g_2(M_2)z^{-M_2} + \ldots \\ &+ g_2(1)z^{-(2M_2-1)} + z^{-2M_2} \end{aligned} \tag{5}$$

The second algorithmic step is to deflate the polynomials $F_1(z)$ and $F_2(z)$ to get $G_1(z)$ and $G_2(z)$ respectively.

Since only the roots with positive imaginary part are of interest, the total number of LSFs is $M_1 + M_2 = P$ whether P is odd or even. The LSF vector consists of an ordered set of angles between 0 and π. Using coefficient symmetry, substituting $z = e^{j\omega}$ in the expressions for $G_1(z)$ and $G_2(z)$ and removing the linear phase term results in the following cosine series expansions [1]:

$$\begin{aligned} G_1(\omega) &= 2cosM_1\omega + 2g_1(1)cos(M_1-1)\omega + \ldots \\ &+ 2g_1(M_1-1)cos\omega + g_1(M_1) \end{aligned} \tag{6}$$

$$\begin{aligned} G_2(\omega) &= 2cosM_2\omega + 2g_2(1)cos(M_2-1)\omega + \ldots \\ &+ 2g_2(M_2-1)cos\omega + g_2(M_2) \end{aligned} \tag{7}$$

These series may be expressed in the form of Chebyshev polynomials in x by applying the frequency mapping $cosm\omega = T_m(x)$ where $T_m(x)$ is the mth order Chebyshev polynomial in x. The mapping applied to $G_1(\omega)$ and $G_2(\omega)$ leads to

$$\begin{aligned} G_1(x) &= 2T_{M_1}(x) + 2g_1(1)T_{M_1-1}(x) + \ldots \\ &+ 2g_1(M_1-1)T_1(x) + g_1(M_1) \end{aligned} \tag{8}$$

$$\begin{aligned} G_2(x) &= 2T_{M_2}(x) + 2g_2(1)T_{(M_2-1)}x + \ldots \\ &+ 2g_2(M_2-1)T_1(x) + g_2(M_2) \end{aligned} \tag{9}$$

Any Chebyshev polynomial series of this form can be evaluated efficiently through the application of Clenshaw's Recurrence Formula [2]. Thus each evaluation of N terms may be achieved with N multiplications and $2N$ additions [1]. The third step is to transform $G_1(z)$ and $G_2(z)$ into their Chebyshev polynomial series form $G_1(x)$ and $G_2(x)$ respectively.

Transforming the polynomials into the Chebyshev domain effectively maps the upper part of the unit circle onto a region from

0-7803-7761-3/03/$17.00 ©2003 IEEE

$x = -1$ to $x = 1$. The roots are isolated through the evaluation of the Chebyshev polynomial series over this region and observing the zero crossings. Figure 1 illustrates the Chebyshev polynomial series $G_1(x)$ and $G_2(x)$ for a particular speech frame. A zero crossing is detected by observing a sign change in the Chebyshev polynomial series. Then, the root location is evaluated at a higher resolution in the neighborhood of the sign change. In [1], it was determined experimentally that a coarse resolution of 0.02 and a fine resolution of 0.0015 is adequate to isolate the root to an acceptable precision. These are the values used in this design. It should also be noted that the interlacing root property on the unit circle carries over to the Chebyshev domain. Thus, the first root found by starting the search at $x = 1$ will be a root of $G_1(x)$. The second root will be a root of $G_2(x)$. This further increases the efficiency of the algorithm as one can alternate between evaluation of $G_1(x)$ and $G_2(x)$ as roots are found. The fourth step is to isolate the roots of $G_1(x)$ and $G_2(x)$ as described above. When all the roots are found, the LSFs are computed as the inverse cosine of the roots of $G_1(x)$ and $G_2(x)$. The implementation of this algorithm in a VLSI design suitable for integration into larger speech coding systems is now discussed.

2. VLSI IMPLEMENTATION

The design of an architecture implementing the described algorithm was undertaken with the following design goals. First, the design is to be realized entirely in VHDL. Second, The design is to be optimized for speed and minimal size. Third, a structure should be chosen which eases integration into larger systems requiring computation of LSFs. Fourth, the design must accommodate up to 12th order LP analysis.

Implementation of the algorithm while observing these design goals required the implementation of various algorithm and utility blocks. These are:

1. *atof1f2*: Compute $F_1(z)$ and $F_2(z)$ from $A(z)$.
2. *polydiv*: Remove the roots of $F_1(z)$ and $F_2(z)$ at $z = \pm 1$.
3. *chebform*: Compute the Chebyshev polynomial series $G_1(x)$ and $G_2(x)$.
4. *rootfinder*: Isolate the roots of $G_1(x)$ and $G_2(x)$ in the interval from -1 to 1.
5. *accos*: Compute the arccos of the roots to obtain the LSFs using a Taylor series expansion.
6. *clenshaw*: Compute the result of a Chebyshev polynomial series efficiently [2]. This is used as a global resource for the root finding algorithm block.
7. *fpmult*: A floating-point multiplication block as a global resource for all algorithm blocks requiring it.
8. *fpadd*: A floating-point addition block as a global resource for all algorithm blocks requiring it.
9. *fpdiv*: A floating-point division block as a global resource for all algorithm blocks requiring it.

Other VLSI implementations of speech coding systems implement a full floating point unit (FPU) or DSP core [3][4]. However, this approach would unnecessarily increase the size and complexity of the implementation of LSF computation. Figure 2 shows the top level architecture of the design. An algorithm block for each major algorithmic task is defined. Each of these blocks makes use of global resources, which are defined as utility blocks.

3. GLOBAL UTILITY BLOCKS

The decision was made to make use of floating point numerics in the execution of the algorithm. Based on experimental results with software implementations on general purpose processors, the 32-bit IEEE 754 single precision bit floating point format was selected. While it may be acceptable in practical applications to use normalized integer computations or fixed point representations, because the dynamic range of the $A(z)$ coefficients cannot be guaranteed, floating point was deemed to be the best choice. Also, interfacing with a host processor or other external systems was thought to be simpler if numerical scaling need not be performed prior to presenting data to the system.

The floating point multiplier and adder are based on those described in [5] and the divider is derived from the same general architecture. All are modified for IEEE 754 floating point format compliance. As implemented in the system, the solution to either of these operations is resolved in one clock cycle (indicated by pre-synthesis results).

The Chebyshev polynomial series evaluation block is a direct implementation of the Clenshaw Recurrence Formula and simply executes the following:

$$\begin{aligned} Y(x) &= \sum_{k=0}^{N-1} c_k T_k(x) \qquad (10) \\ &= \sum_{k=0}^{N-1} [b_k(x) - 2x b_{k+1}(x) + b_{k+2}(x)] T_k(x) \\ &= \frac{b_0(x) - b_2(x) + c_0}{2} \end{aligned}$$

The backward recurrence $b_k(x) = 2x b_{k+1}(x) - b_{k+2}(x) + c_k$ with initial conditions $b_N(x) = b_{N+1}(x) = 0$ is used to calculate $b_0(x)$ and $b_2(x)$ to get $Y(x)$ as above. The Chebyshev evaluation block also makes use of the floating point addition and multiplication blocks.

4. MAIN ALGORITHM BLOCKS

The main algorithm blocks consist of the major algorithms described above. First, the *atof1f2* computes $F_1(z)$ and $F_2(z)$ from $A(z)$. Next, $F_1(z)$ and $F_2(z)$ are deflated by their trivial roots by the *polydiv* block. The deflation block is one entity and has an input bit indicating which root is to be removed. The resulting output is presented to a block which finds $G_1(x)$ and $G_2(x)$ *chebform*. Then, the zero crossings are isolated by the *rootfinder* block. This is the portion of the circuit that scans the linear region from +1 to -1 with a coarse resolution and detects zero crossings. Following the detection of a given zero crossing, the immediate region is re-scanned with a finer resolution. In order to avoid a complete 32-bit IEEE 754 format magnitude comparison, a change in the sign bit is detected to isolate a zero crossing. At each point in the scan, the Chebyshev polynomial series evaluation block, *clenshaw*, is invoked to determine the value of the series. Advantage of the interlacing property is taken to avoid evaluating both polynomial series over the entire region. The algorithm block will switch from one to the other as each zero crossing is found. The ultimate output of the zero crossing isolation is the set of values of x corresponding to the roots of $G_1(x)$ and $G_2(x)$ (*rootfinder*). The final block computes the LSFs as the inverse cosine of the roots

of $G_1(x)$ and $G_2(x)$. For this implementation, a truncated Taylor series approximation is performed by the *acos* block.

Each of the major blocks share a similar internal structure. The input data is presented to the entity via signal ports. A state machine sensitive to both rising and falling clock edges is implemented. For external utility blocks, signal interfaces are provided, for example, the *fpmult* block is presented with the multiplier and multiplicand to obtain a product. The floating point entities are designed to resolve an output within one clock cycle. The *clenshaw* entity, however, has an internal state machine and requires several clock cycles to resolve a solution. This particular entity has logic to control start and signal completion of the Chebyshev evaluation. This logic is tied into the state machine of the *rootfinder* entity. The entities are cascaded with the output of one block feeding the input of the next.

All of the previously described entities are tied together at the top level in structural VHDL. The input and output ports of the top level entity are (1) Twelve 32 bit $A(z)$ coefficient inputs, (2) Twelve 32 bit LSF outputs, (3) Clock line in and (4) Done line out indicating completion. The twelve inputs are 32-bit wide vectors in IEEE 754 floating point format. Naturally, to interface the top level entity to a smaller width bus, appropriate logic would be used as a bus interface. However, for the purposes of verifying performance of the core algorithm, this is not a major issue. Prior to synthesis for actual implementation, a more convenient bus interface would be added.

5. VHDL SIMULATION AND SYNTHESIS RESULTS

The first step in the VHDL implementation process was the coding of the algorithm and pre-synthesis simulation to verify accurate performance. The intent at this level is to ensure that any assumptions made, particularly those that limit precision of floating point operations, do not impact the accuracy of the computed LSFs.

It can be seen that the core VHDL entity is presented with up to 12 32-bit vectors containing the $A(z)$ coefficients and outputs 12 computed LSFs. The entity is provided with a clock used for sequencing of operations. Effort was made to perform operations on both rising and falling clock edges to maximize throughput at the cost of circuit complexity. For synthesis into actual hardware, a bus interface would be added to the entity to interface to a typical 16- or 32-bit bus architecture for the writing and reading of data.

The entities that require the most clock cycles to obtain a result are *rootfinder* and *acos* at 377 and 277 respectively. The *atof1f2* entity requires 39 clock cycles. The fewest clock cycles are required by the *polydiv*, *clenshaw* and *chebform* entities at 14, 14 and 12 respectively. VHDL simulation was performed using Modelsim and it should be noted that no post-synthesis parasitics are incorporated in these results. Simulation was conducted using test benches developed specifically for each entity using the TIMIT database speech data sets. Assuming a clock speed of 100 MHz, the overall time to resolve a result given a 12th order $A(z)$ input coefficient vector would be approximately 7.33us. Note that the *rootfinder* entity will exhibit slightly nondeterministic performance depending upon the specific location of a zero crossing in relation to the location at which it was detected. However, this will not be significant with the values of coarse and fine increments used.

Because this simulation contains no parasitic or timing parameters, post synthesis simulation must be run to obtain performance information on an actual silicon implementation. The circuit was synthesized using Leonardo Spectrum and the MOSIS AMI05 synthesis library from Mentor Graphics. Table 1 shows the result of the synthesis for each entity. In Table 1, the primatives are the low level gates to implement the design. The operators are the logic that perform operations (like the ripple carry adders and magnitude comparators). Note that the individual entity metrics ignore certain overheads which would be present in the overall system. Should this architecture be implemented in a larger system as intended, the overall architectural concerns must be evaluated and the design suitably modified. For example, a larger system may be of a complexity level as to require a full floating point unit, in which case the *fpmult* and *fpadd* entities may be redundant. Also, the bus interfaces with any external systems must be designed for convenience as well as performance which may dictate the implementation of more conventional bus interfaces.

Entity	Primatives	Operators
acos	1304	144
atof1f2	2444	144
chebform	1888	75
polydiv	1504	73
clenshaw	758	134
fpmult	79	23
fpadd	585	18
fpdiv	643	36
rootfinder	2468	109

Table 1: VHDL Synthesis Results

The results from VHDL level simulation as well as synthesis results indicate that the approach for achieving an application specific VLSI design for LSF computation is feasible. For further study, the performance of the architecture described will be examined incorporating post-synthesis parasitics.

6. REFERENCES

1. 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.
2. W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.
3. 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.
4. J. McDonough, C. Chang, P. Kantak, C. Sakamaki, R. Singh and M.C. Tsai, "A Single Chip QCELP Vocoder for CDMA Digital Cellular", *IEEE Custom Integrated Circuits Conference*, pp. 211–214, 1994.
5. K. C. Chang, *Digital Systems Design with VHDL and Synthesis*, IEEE Computer Society Press, 1999.
6. J. Allan and W. Luk, "Parameterised Floating-Point Arithmetic on FPGAs", *IEEE International Conference on Acoustics, Speech and Signal Processing* pp. 897–900, May 2001.

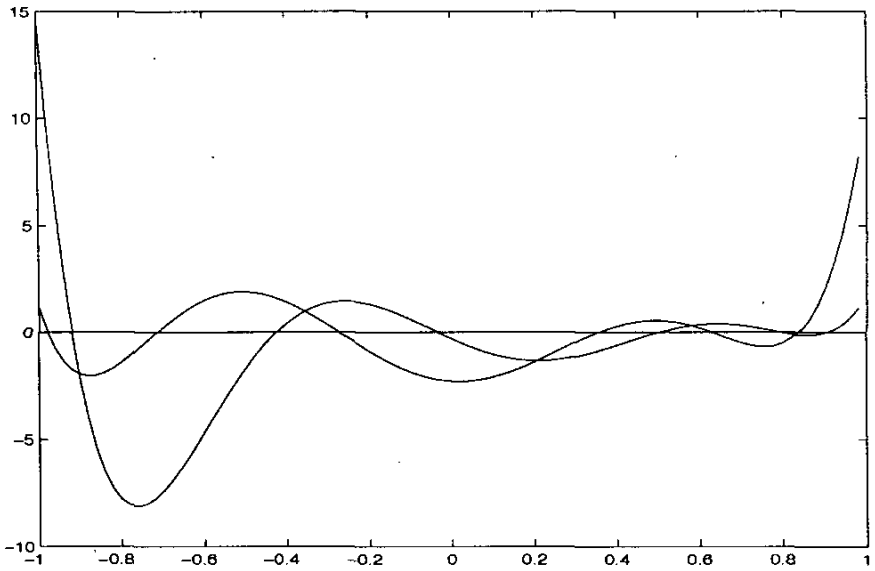

Figure 1: Plots of Chebyshev Polynomial Series $G_1(x)$ and $G_2(x)$ (12th Order)

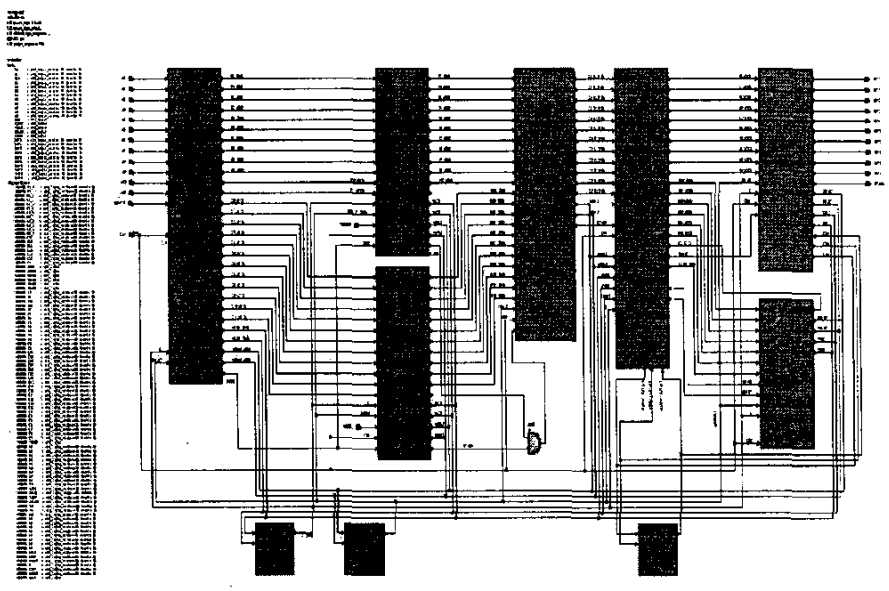

Figure 2: Top Level Block Architecture