ASIC Design for the Efficient Computation of Line Spectral Frequencies Using Chebyshev Polynomial Series

David L. Reynolds
L-3 Communications
david.reynolds@l-3com.com

Linda M. Head
Rowan University
head@rowan.edu

Ravi P. Ramachandran
Rowan University
ravi@rowan.edu

***Abstract*—Various methods for the efficient computation of line spectral frequencies (LSFs) have been proposed to address the computationally intensive task of isolating the roots of high-order polynomials in linear predictive (LP) systems. An ASIC implementation of one such algorithm to compute LSFs has been developed, simulated and synthesized. The design, expressed entirely in VHDL, is intended for implementation into larger speech processing systems. The design developed is suitable for speech coding applications requiring a 10th order LP analysis and for speaker recognition applications which need a 12th order analysis. The resulting efficient design is of low complexity, modular, optimized for speed and area and can be used in larger speech processing systems to offload main application processors and DSPs. When the designed ASIC is part of a speaker identification system, the same accuracy as a software implementation is obtained.**

I. Introduction

Speaker recognition and speech coding systems based on line spectral frequencies (LSFs) are used in a variety of applications, many of them on embedded computing platforms with limited power, memory and processing resources available. For these systems, the linear predictive (LP) coefficients are converted to the equivalent LSF parameter set. However, the computation of LSFs requires the isolation of high-order polynomial roots, typically of 10th order (speech coding) and 12th order (speaker recognition). Algorithms for the isolation of roots for these orders consume significant computational resources. The LSFs are calculated from polynomial roots on the unit circle thereby making the rootfinding algorithm efficient [1]. An application specific integrated circuit (ASIC) is designed and simulated based on the algorithm in [1].

The use of ASICs as a flexible, high-performance circuit solution is currently undergoing a transformation to integrated cell-based ASIC design with the addition of FPGAs, processor arrays and other platforms [2]. Although the current trend is indeed toward platform solutions, structured fabrics and heavily constrained layout for low volume markets, engineers in industry have shown that the inclusion of standard-cell design of ASICs yields performance advantages for high-end designs at even nanometer geometries.

The design of ASICs (or algorithm-on-a-chip) is important to the development of many products that incorporate signal processing. These subsystems are characterized by hardwired computational functionality and interconnects that can be used in component fashion in a larger VLSI system [3]. Use of the functional subunits has enhanced productivity for the design of the larger VLSI systems and is most often used as an intellectual property (IP) unit in the design rather than a fabricated stand-alone chip.

The advantage of using algorithm-on-chip methods and IP allows designers to develop efficient silicon systems under decreased time-to-market pressures. With the development of IP that implements specific algorithms, it is possible to develop libraries that can be used to assemble hierarchical chip designs. This results in an algorithm-to-architecture-to-chip design flow [4] that gives hardware designers comparable flexibility to software versions of DSP development. There is continuing work on the development of innovative, domain-specific cell based ASIC design flows that target digital signal processing applications [5]. These developments are designed to narrow the performance gap between the full custom ASIC design method and the conventional standard-cell based ASIC design method. Cell libraries developed for these applications take advantage of existing algorithm-on-a-chip IP and work continues to take advantage of the development of new hardware algorithm designs such as the one we present.

Advances in algorithm-on-chip methods are in many areas. ASICs that perform lossless data compression is the subject of [6]. A hardware version of the watershed transformation used in image segmentation has been shown to enhance efficient memory usage when compared with the software implementations [7]. Imaging technologies take advantage of the designer's ability to create an implementation of global algorithms in an ASIC [8]. The development of ASICs is expanding in medical devices for remote diagnosis where low power, portability, and high reliability are crucial. A typical example is the work being done on real-time cardiac monitoring. Researchers are developing intelligent electrocardiogram sensors that incorporate a newly developed cardiac arrhythmias detection algorithm in hardware implementation [9].

An ASIC architecture of the method presented in [1] was developed [10] and further expanded from initial architecture to design for synthesis. The intent of the system was to provide a VHDL design for the fast computation of LSFs in a form suitable for implementation into typical speech processing systems. Building on this prior work:

1) More detailed design aspects are presented,
2) Synthesis and layout results are discussed and
3) Performance aspects of the ASIC as part of a speaker identification system are presented.

II. Mathematical Background

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

1-4244-1176-9/07/$25.00 ©2007 IEEE.

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]:

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

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

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 cosine series expansions [1]. These cosine series may be expressed in the form of Chebyshev polynomials in x by applying the frequency mapping $cos m\omega = T_m(x)$ where $T_m(x)$ is the mth order Chebyshev polynomial in x. The above steps as applied to $G_1(z)$ and $G_2(z)$ leads to

$$\begin{aligned} G_1(x) &= 2T_{M_1}(x) + 2g_1(1)T_{M_1-1}(x) + ... \\ &+ 2g_1(M_1 - 1)T_1(x) + g_1(M_1) \end{aligned} \quad (6)$$

$$\begin{aligned} G_2(x) &= 2T_{M_2}(x) + 2g_2(1)T_{(M_2-1)}x + ... \\ &+ 2g_2(M_2 - 1)T_1(x) + g_2(M_2) \end{aligned} \quad (7)$$

Any Chebyshev polynomial series of this form can be evaluated efficiently through the application of Clenshaw's Recurrence Formula [11]. 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 $x = -1$ to $x = 1$. The roots are isolated through the evaluation of the Chebyshev polynomial series over this region and observing the zero crossings. 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 $G_2(x)$.

III. State Machine Architecture

The architecture discussed in [10] is based on functional decomposition of the algorithm presented above. This algorithm was chosen because it lends itself to straightforward decomposition and implementation in hardware. The design developed consists of the following VHDL entities, each of which implements a specific step in the algorithm:

1) *atopq* computes $F_1(z)$ and $F_2(z)$ from $A(z)$.
2) *polydiv* deflates $F_1(z)$ and $F_2(z)$ by the roots at $z = \pm 1$ to get $G_1(z)$ and $G_2(z)$ respectively.
3) *chebform* arranges the deflated polynomials $G_1(z)$ and $G_2(z)$. into the Chebyshev series form $G_1(x)$ and $G_2(x)$.
4) *rootfinder* evaluates the Chebyshev polynomial series over the region [-1,+1] and identifies the specific root locations.
5) *accos* computes the arccosine of the Chebyshev polynomial series roots to obtain the LSFs.
6) *clenshaw* evaluates the Chebyshev polynomial series at a specific point using the Clenshaw recurrence [11].
7) *fpmult* is a simple 32-bit floating point multiplier.
8) *fpadd* is a simple 32-bit floating point adder.
9) *fpdiv* is a simple 32-bit floating point divider.

The floating point entities are used as a convenience in development of an ASIC for evaluation purposes. It is assumed that any larger system into which this VLSI design would be incorporated would have global floating-point resources available for use by all components of the overall system. The floating point units used are simple combinatorial constructs and would not be optimal for implementation in a real-world system, however they are more than adequate for the evaluation of this design.

With the exception of the combinatorial floating point entities, all other components are based on a similar state machine architecture. The components are arranged in a cascaded manner such that the output of one is presented on the input of another in the order dictated by the algorithm. In general, a *START* and *DONE* bit are used to arbitrate the data flow from entity to entity. The state machine begins running when the *START* bit for the entity in question is asserted. State transitions occur as data is processed. For performance of floating-point operations, data is presented to 32-bit wide ports which interface with the external floating-point entities. The results are offloaded upon the next state transition. Figure 1 illustrates the state machine flow for the *atopq* entity, which is typical of all components of the system. Figure 2 shows an excerpt of the VHDL implementation used for the *atopq* entity, which is also typical of all entities in the system.

Entity	Gates
acos	7410
atopq	7699
chebform	5204
polydiv	5517
clenshaw	3139
fpmult	6548
fpadd	2285
fpdiv	7419
rootfinder	6570

TABLE I
VHDL Synthesis Results

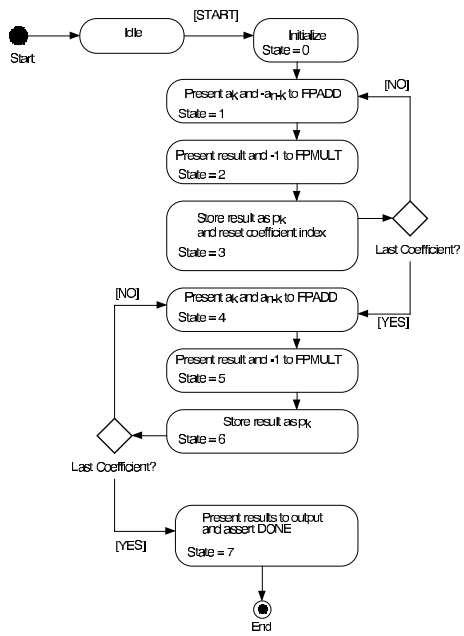

Fig. 1. State Machine Structure of the *atopq* Entity

Entity	Max. Clock	Total Clocks
acos	10ns	102
atopq	10ns	38
chebform	10ns	13
polydiv	10ns	13
clenshaw	10ns	18
fpmult	10ns	1
fpadd	10ns	1
fpdiv	10ns	1
rootfinder	10ns	4658 (Worst Case)

TABLE II
CLOCK RATES AND TOTAL CLOCKS

IV. SYNTHESIS OF VHDL ENTITIES

Synthesis of each entity was performed using Leonardo-Spectrum and AMI's C5 process with primitives and gates from the ASIC Design Kit provided by Mentor Graphics. A VHDL model of the resulting circuit was generated and used for simulation. Gate counts for each entity are shown in Table I. The VHDL model provided by the synthesis tool was used to determine the operational limits of the design. Each entity was simulated to the maximum clock period design goal of 10ns. The VHDL models used in this simulation account for the approximate gate level delays inherent in the synthesis library devices used by the synthesis tool to implement the actual design and are therefore more representative of the performance of an actual physical system. Note that additional delays due to parasitic capacitance will occur in the actual system. Table II summarizes the results of the synthesis as well as the total number of clock periods required for an output to be resolved. Because the *rootfinder* entity makes use of *clenshaw*, a modified test bench which eliminates *clenshaw* was developed for the purpose of isolating throughput performance of the *rootfinder* entity. When *rootfinder* processes, for each location on the x axis, a Chebyshev polynomial is evaluated using the *clenshaw* entity. This occurs 100 times for the coarse resolution

```
begin
if START = '0' then
        DONE <= '0';
        STATE := 0;
        XMULT <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        YMULT <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        XADD <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
        YADD <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";

else
        -- Here is the state machine that executes if the START bit is asserted.
        case STATE is

                when 0 =>
                -- In this state, we examine the START bit and transition if it is asserted.
                STATE := 1;

                when 1 =>
                ...
                STATE := 2;

                ...
                ...

                when 6 =>
                Q(index + 1) := ZMULT;
                if INDEX = 11 then
                        STATE := 7;
                else
                        INDEX := index + 1;
                        STATE := 4;
                end if;

                when 7 =>
                ...
                XMULT <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
                YMULT <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
                XADD <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
                YADD <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
                STATE := 8;

                when others =>
                DONE <= '1';

        end case;
end if;

end process;
```

Fig. 2. VHDL State Machine Implementation of *atopq* Entity

scan and a worst case of 156 times for the fine resolution scan. The actual number of evaluations for the fine resolution scan is dependent on the specific root locations and will vary from frame to frame. For this reason, *rootfinder* was isolated from *clenshaw* for the purposes of benchmarking the entity by itself. These results indicate that the *clenshaw* entity is the performance bottleneck in the overall system. The *clenshaw* entity is unique in that it exercises the combinatorial floating point elements more than the other components. It should be noted that the *rootfinder* performance in terms of total number of clock cycles required to compute an output will vary slightly depending upon the actual root locations due to the method used to determine precise root locations using a coarse and fine increment scan of the [-1,+1] interval.

Given the number of clock cycles required for resolution of each entity given in Table II, a solution to a 12th order input vector would be computed within a maximum of 4824 clock cycles and will typically be lower depending upon specific root locations. Overall performance of the system indicates that it may be operated at a clock frequency of at least 100MHz. At this rate, the LSFs of a 12th order system may be resolved in about 50us, which is significantly shorter than the length of a typical frame of speech in applications of this type.

EDIF files for each entity were generated and used in the subsequent layout. Each entity was individually placed and routed. The resulting modules are then combined schematically and the resulting overall chip layout obtained.

V. ARCHITECTURAL IMPACT ON PERFORMANCE

The design presented is based on the functional decomposition of the algorithm described in [1]. The general structure of the design is a cascade of sequential processing elements. Data is presented to the input, processed by the first element, and the result passed downstream to the remaining elements for further processing. The end result is presented to the system outputs by the last element.

Entity	Size (λ)	Area (mm^2)
acos	11408 x 7999	11.178
atopq	12303 + 8954	13.495
chebform	7268 x 6706	5.971
polydiv	8322 + 6815	6.948
clenshaw	5461 + 5252	3.513
fpmult	7106 + 7046	6.133
fpadd	5086 + 4303	2.681
fpdiv	8464 x 7985	8.279
rootfinder	8246 x 7985	7.716

TABLE III
AREA BY ENTITY

The sequential nature of the design does not take advantage of opportunities for performance improvements through parallelism due to the increase of the size of each entity on the chip. A parallel architecture where dedicated processing units for the symmetric and antisymmetric polynomial coefficients would increase throughput, however at the expense of increased chip area. The *polydiv* entity processes one polynomial at a time and has a selector bit to determine the root to be removed. The *chebform* entity is provided with both sets of polynomial coefficients, but exhibits little internal parallelism, although greater than that seen in *polydiv*. Two independent *chebform* entities would increase system throughput, again at the cost of increased area. Simulation demonstrates that the design presented exhibits adequate performance in real-world applications and is smaller in size than an alternate design with increased parallelism. Throughput performance of the design presented is more than adequate for all practical applications.

Table III enumerates the size of each entity following layout. Excluding the floating point elements, a total chip area for the design is approximately 49 mm^2. Again, in an actual system global floating point resources would be available and would likely be optimized for the overall architecture. Although somewhat large, the design processes 12th order data in native 32-bit floating point format. The use of floating point representations is a direct driver of overall size.

VI. SPEAKER IDENTIFICATION APPLICATION

The developed ASIC is part of a closed set speaker identification system with LSFs as the feature vector and a vector quantizer (VQ) classifier. The New England dialect of the TIMIT database consisting of 38 speakers is used. There are 10 sentences for each speaker with the first 5 used for training and the remaining 5 used for testing. For both training and testing, the speech is downsampled to 8 kHz and preemphasized by the filter $1 - 0.95z^{-1}$.

The autocorrelation method is used to get a 12th order linear prediction (LP) polynomial $A(z)$ over frames of 30 ms duration. The overlap between frames is 20 ms. For each speech frame (silent frames are eliminated by energy thresholding), the $A(z)$ is converted into a 12 dimensional LSF vector. The training involving the above steps and the design of a VQ codebook for each speaker is done entirely by a software implementation (C code implementation). The testing is done in two ways to evaluate the performance of the ASIC in a practical scenario. First, a complete software C implementation is used to compute the LSF vector and evaluate the speaker identification performance. Second, the developed ASIC is used to convert $A(z)$ to an LSF vector for each speech frame with the other steps being done by a C implementation. Various combinations of training and testing involving clean speech, speech corrupted by additive white Gaussian noise and speech subjected to telephone channel effects were attempted. The C and ASIC implementations of the algorithm for computing the LSFs led to the same speaker identity for any test utterance and hence, led to the same speaker identification performance for any training/testing combination,

VII. SUMMARY AND CONCLUSIONS

A complete design and layout of an ASIC implementation of an algorithm to compute LSFs from LP coefficients has been completed and optimized for real-time applications. The synthesized circuit has been used to estimate limits of throughput based on gate level delays. The unique features of this design include (1) the circuit is adaptable for an LP order of 10 or 12, (2) the hardware is optimized for speed and area, (3) the modular sequential design allows for portability of each VHDL entity and (4) the design can be integrated in larger speech processing systems. The integration into a speaker identification system is shown.

VIII. 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) P. S. Zuchowski, C. B. Reynolds, R. J. Grupp, S. G. Davis, B. Cremen, and B. Troxel, "A Hybrid ASIC and FPGA Architecture", *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, pp 187–194, 2002.
3) A. Hemani, "Charting the EDA Roadmap", *IEEE Circuits and Devices Magazine*, Vol. 20, No. 6, pp 5–10, 2004.
4) J. McCanny, D. Ridge and J. Yi Hu Hunter, "Hierarchical VHDL libraries for DSP ASIC design", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp 675–678, April 1997.
5) B. Ren, A. Wang, J. Bakshi, K. Liu, W. Li and W. Dai, "A Domain-Specific Cell Based ASIC Design Methodology for Digital Signal Processing Applications", *Design, Automation and Test in Europe Conference and Exhibition Designers Forum*, Vol. 3, pp. 280–285, February 2004.
6) R Vitulli, "PRDC: An ASIC device for lossless data compression implementing the Rice algorithm", *IEEE International Geoscience and Remote Sensing Symposium*, p. 320, September 2004.
7) K. P. Gupta and S. Srinivasan, "Reduced memory implementation of modified serial watershed algorithm based on ordered queue", *International Conference on Information Technology: Coding and Computing*, pp 514–518, April 2003.
8) H. Yu, R. Miyaoka and T. Lewellen, "Analog ASICs for a depth of interaction (DOI) positron emission tomography (PET) detector module", *IEEE Nuclear Science Symposium*, Vol. 2, pp. 9/82–9/86, 2000.
9) H. Zhou, K. M. Hou, J. Ponsonnaille, L. Gineste and C. De Vaulx, " A continuous cardiac arrhythmias detection system: RECAD", *IEEE Engineering in Medicine and Biology Society Conference*, pp. 875–881, September 2005.
10) D. L. Reynolds, L. M. Head and R. P. Ramachandran, "VLSI architecture for the efficient computation of line spectral frequencies", *IEEE International Symposium on Circuits and Systems*, Bangkok, Thailand, pp. 718-721, May 2003.
11) W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 1992.