An Evaluation of the Goertzel Algorithm for Low-Power, Embedded Systems

Stephen Brown, Joe Timoney, and Tom Lysaght

Department of Computer Science
National University of Ireland, Maynooth
IRELAND

Abstract — **The Goertzel Algorithm provides an efficient mechanism for tone detection in signal processing. However, on Wireless Sensor Network nodes, there is often not sufficient processing power to implement this using floating-point arithmetic, so fixed-point implementations are used. In this paper we compare the quality of the results for a typical audio application with different resolutions of fixed-point support using MATLAB. We also provide performance figures based on implementing this on a representative sensor node platform. Our results show that, for this application-space, 16-bit fixed-point arithmetic provides the best tradeoff of accuracy against performance. No previous work has addressed this specific aspect of implementating the Goertzel algorithm.**

Keywords — **Goertzel Algorithm, Performance, Fixed-Point, WSN.**

I Introduction

The Goertzel algorithm[1] provides efficent tone detection in a signal. It can be regarded as the evaluation of a single *bin* of the FFT, or as a narrow-band filter. It is suitable for use in low power, embedded-systems where the FFT is not viable or not necessary. Multiple tones can be detected by running the algorithm multiple times, unlike the FFT which calculates for all bins simultaneously. Examples include structural health monitoring[2], instrument tuning [3], power metering[4] and active damping[5], voice [6] and underwater communications[7], security applications such as fingerprint identification [10], and mobile applications such as crowd counting [8] and gesture sensing[9].

In low-cost systems without a FPU, fixed-point arithmetic can be used as a high-speed alternative to floating-point software emulation, as long as the problems of overflow and underflow are avoided (due to the restricted range of values that can be represented)[11]. In general, fixed-point will suffer from reduced accuracy due to the inability to represent the same range of values as a floating-point representation of the same bit-width.

In this paper we address the issue of determining how best to implement the Goertzel Algorithm on a low power, 8-bit CPU, representing a typical Wireless Sensor Network (WSN) node. There are two questions: what accuracy can be achieved for different bit-widths, and how to evaluate and compare the results (in order to select the optimal solution).

We present accuracy results from executing fixed-point Goertzel in MATLAB, performance results from implementing the algorithm on an 8-bit Atmel CPU, and a generic procedure for evaluating the results. For the scenario presented, a 16-bit fixed-point representation provides the best balance between performance and accuracy.

II Related Work

The basic algorithm[1] is defined as:

$$
\begin{aligned}
U_{N+2} &= U_{N+1} = 0 \\
U_k &= a_k + 2cos(x)U_{k+1} - U_{k+2}, k = N..1 \\
C &= a_0 + U_1 cos(x) - U_2 \\
S &= U_1 sin(x)
\end{aligned}
$$

Where a_k is the kth sample, x is the *target* frequency, C is the signal amplitude at that frequency, and S is the phase.

The Goertzel algorithm was introduced in the 1950s as an efficient technique for determining the magnitude of a component at a known frequency. It is still being used in applications today, a testament to its value. For example, in [19] it is applied to dual tone multiple frequency DTMF[16][17] detection using an FPGA device. This particular application domain has been dominated by this algorithm since the 1970s[18]. Other applications included spectral analysis for violin tuning[3] and it has formed the analytical basis for a number of non-uniform spectral analysis techniques that have been proposed in recent times. Furthermore, its sample-by-sample analysis means that effectively it is a "sliding" frequency analysis technique without requiring buffering of the input data. This makes it very attractive for real-time implementations on low complexity hardware.

III Fixed-Point Implementation

The C-language fixed-point implementation uses in-line functions to implement the basic arithmetic operations requried for the Goertzel algorithm. The simple *add* and *subtract* operations can be performed using the normal C operators. The *multiply* and *divide* operations require more complex code to minimise underflow, and to multiple/divide the result to maintain the fixed-point position. A *sqrt* function is also required: this was implemented using an iterative 2-variable approximation[12].

On an 8-bit CPU there is no benefit to using precisions that are not a multiple of 8. Most of the *load* and *store* operations are 8-bit, as are the mathematical operations. For completeness, however, we have evaluated the algorithm with fixed points widths using fixed-point widths of 6-16 bits.

As the purpose of the experiments is to compare the different implementations, rather than produce an optimal system, no windowing function[13][14] has been applied.

IV Results

The results shown here were collected using MATLAB^{TM}to compare the accuracy of the different fixed-point sizes against floating-point, which was used as a accuracy reference point (shown in the figures as a width of 17).

a) Experimental Setup

Three test signals were used. A noise-free, single-tone to evaluate the accuracy of each implementation; multiple noise-free tones to evaluate the quality of separation; and a single tone with added noise to evaluate robustness (shown in Fig. 1).

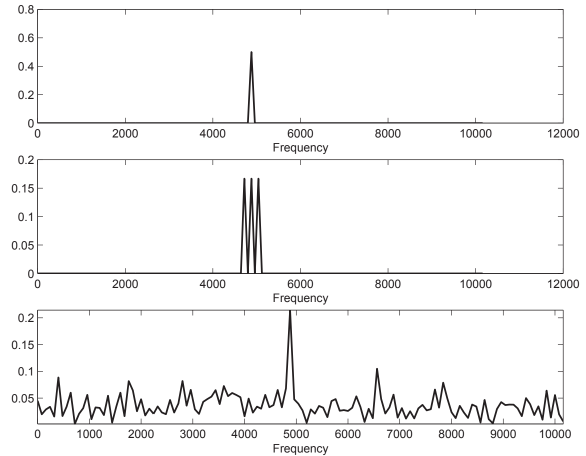

Fig. 1: FFT of the Test Signals.

b) Accuracy Results

The results are presented here for the three different test signals. Eqn. 1 was used to assess the relative accuracy (RA) of the result produced by the different algorithms (v_i) compared to the *double* implementation result (v_{ref}) at each bin value i.

$$RA = 1.0 - \prod_{i=1}^{n} \left[1.0 - abs\left(v_{ref} - v_i\right)\right] \qquad (1)$$

Single Tone Signal

The comparative results for the different width arithmetics are shown in Fig. 2. This shows the output of the Goertzel Algorithm (Frequency axis) for each of the different widths of fixed-point arithmetic (Number of Bits axis); the *double* results provide a baseline for comparison, and are shown as a width of 17. Below 10-bits, artefacts become obvious. Note that *float* provided almost identical results to *double*, and so are not included.

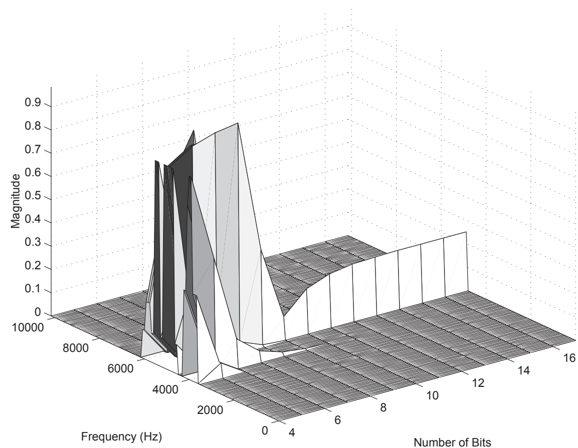

Fig. 2: Single Tone Comparison.

An accuracy comparison is presented in Table 1. The key factor here is that below 12 bits the accuracy of results drops below 99%, and below 8 bits significant errors are introduced by the lack of precision.

Table 1: Accuracy Results.

Arithmetic	Accuracy
fixed-point (4)	1.21E-12
fixed-point (5)	9.06E-06
fixed-point (6)	3.42E-04
fixed-point (7)	1.31E-01
fixed-point (8)	7.53E-01
fixed-point (9)	8.34E-01
fixed-point (10)	9.36E-01
fixed-point (11)	9.77E-01
fixed-point (12)	9.94E-01
fixed-point (13)	9.98E-01
fixed-point (14)	1.00E+00
fixed-point (15)	9.97E-01
fixed-point (16)	9.97E-01
floating point	1.00

Multi-Tone Signal

The comparative results for the different widths with three tones are shown in Fig. 3. The *double* results again provide the baseline for comparison. Note that below a width of 9 bits, significant aftefacts are introduced.

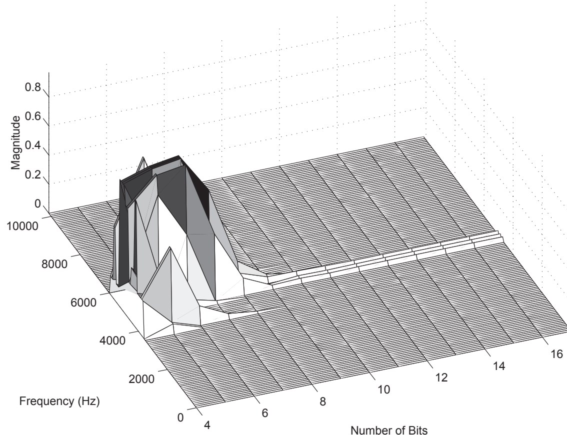

Fig. 3: Three-Tone Comparison.

Single-Tone, Noisy Signal

The comparative results for the different arithmetics for a single tone with added noise are shown in Fig. 4. The *double* results again provide the baseline for comparison. Note that below a width of 8 bits, significant aftefacts are introduced.

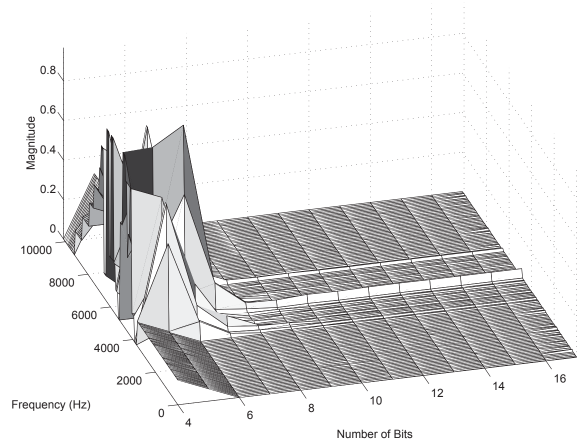

Fig. 4: Noisy, Single Tone Comparison.

c) Performance Results

The basic performance of the add, subtract, multiply, divide, and sqrt functions was measured on an Atmega128-RFA1[1] node using the 16-MHz clock (providing single-cycle precision and accuracy), for comparison against the full Goertzel function execution times. Some of the key results are shown in Table 2. Note that these figures include the fetch and store operations also, which have a significant impact in an 8-bit architecture.

Table 2: Basic Execution times.

Operation	Cycles	micro-seconds
double.add	125	7.813
double.multiply	141	8.813
long.add	20	1.250
long.multiply	66	4.125
short.add	10	0.625
short.multiply	18	1.125

The Goertzel function *add()* is called once per data sample (to process that sample) –and the function *getMag()* is called once per block (to get the magnitude of the response). The execution time of the Goeetzel algorithm *add_value()* and *get_magnitude()* functions was also measured on the same system, as shown in Tables 3 and 4.

Table 3: Goertzel Function Execution times (16-bit fixed-point)

Operation	Cycles	Execution Time
add()	68	4.25μs
getMag()	405	25.313μs

These show the strong motivation for using fixed-point arithmetic on such a platform. The figures are averaged over 4,000 calls to add() and

[1]See http://www.atmel.com/Images/doc8266.pdf

Table 4: Goertzel Function Execution times (floating-point)

Operation	Cycles	Execution Time
add()	450	28.125μs
getMag()	1300	81.25μs

200 calls to getMag(). Note that additional, common overheads reduce the difference between the basic execution times of the raw arithmetic and the execution times of the Goertzel function.

The total CPU cost (CPU in μs-per-second) for processing the Goertzel function can be described by Eqn. 2, where sr is the sample rate and br is the block rate.

$$CPU = sr * add() + br * getMag() \quad (2)$$

Substituting the measured values from Table 3 into Eqn. 2 gives the CPU load for executing the 16-bit fixed-point Goertzel implementation. This is shown in Eqn. 3 for a sample rate[2] (sr) of 20,480 and a block size of 256 samples–giving a block rate (br) of 20,480/256.

$$\begin{aligned} CPU &= 20,480 * 4.25 + (20,480/256) * 25.31 \\ &= 87,040 + 30428 \\ &= 117,468[\mu s/second] \\ &= 11.75\% \end{aligned} \quad (3)$$

The results clearly show the benefit of the 16-bit fixed-point arithmetic. The slight reduction in accuracy is not important in the context of the Goertzel Algorithm, and produces a significant performance improvement over the more accurate results. Using floating-point (emulated in software) is not a feasible option on this platform with a 20kHz sampling rate as the CPU is not fast enough to keep up with the data (greater than 100% utilisation). The improved accuracy-speed factor for 8-bit fixed point does not compensate for the very poor frequency resolution (as shown in Figs. 2-4).

V Conclusions and Future Work

In this paper we present new results comparing the accuracy the Goertzel Algorithm with different precisions of fixed-point arithmetic. This is important for 8-bit CPUs where the low bus bandwidth imposes significant extra costs for wider values (i.e. multi-byte arithmetic).

This work provides a methodology and results for selecting an appropriate tradeoff between the quality of the results against the speed of execution (compared to a *double* implementation). The conclusion of this paper is that for the scenario presented, a 16-bit fixed-point implementation of the Goertzel algorithm provides the best tradeoff between performance and accuracy. Floating-point implementations are not feasible at this sampling rate.

Future work will include measurement of the effectiveness of these results in a real-world tone-detection application on the Atmega128-RFA1 WSN platform, a comparison when a high-efficiency windowing function has been applied (e.g. Dolph-Chebyshev[15]), and futher consideration of the 8-bit fixed-point problems to see if these can be overcome in order to realise the significant performance benefits. The figures show some promise for at least a partial 8-bit implementation which would be a very significant result for 8-bit systems allowing very high-speed processing of multiple tones.

[2] Selected to provide 10Khz bandwidth, typical of low-cost microphones, and also making the block size divisible by a power of 2, providing faster arithmetic

References

[1] G. Goertzel "An Algorithm for the Evaluation of Finite Trigonometric Series", *The American Mathematical Monthly*, 65(1):34–35, 1978.

[2] M. Bocca *et al.* "Structural Health Monitoring in Wireless Sensor Networks by the Embedded Goertzel Algorithm". *IEEE/ACM Second International Conference on Cyber-Physical Systems*, 206–214, 2011.

[3] Z. J. Wang *et al.* "Electronic Assisting Violin Tuner". *TENCON 2012 - 2012 IEEE Region 10 Conference*, 1–6, 2012.

[4] R. Pena-Alzola *et al.* "Self-commissioning Notch Filter for Active Damping in Three Phase LCL-filter Based Grid Converters". *Power Electronics and Applications (EPE), 2013 15th European Conference on*, 1–9, 2013.

[5] K. Koziy *et al.* "A Low-Cost Power-Quality Meter With Series Arc-Fault Detection Capability for Smart Grid". *IEEE TRANSACTIONS ON POWER DELIVERY*, 28(3): 1584–1591, 2013.

[6] V. Gabale *et al* "Building a low cost low power wireless network to enable voice communication in developing regions" *SIGMOBILE Mob. Comput. Commun. Rev.*, 16(2):2–15, 2012.

[7] D. Kohlsdorf *et al* "An Underwater Wearable Computer for Two Way Human-dolphin Communication Experimentation" *Proceedings of the 2013 International Symposium on Wearable Computers (ISWC'13)*, 147–148, 2013.

[8] P. G. Kannan *et al* "Low Cost Crowd Counting Using Audio Tones" *Proceedings of the 10th*

ACM Conference on Embedded Network Sensor Systems (SenSys '12), 155–168, 2013.

[9] S. Gupta *et al* "SoundWave: Using the Doppler Effect to Sense Gestures" *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12)*, 1911–1914, 2012.

[10] T. Y. Tang *et al* "Efficient Implementation of Fingerprint Verification for Mobile Embedded Systems using Fixed-point Arithmetic" *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC'04)*, 821–825, 2004.

[11] M. Medina-Melendrez *et al.* "Overflow analysis in the fixed-point implementation of the first-order Goertzel algorithm for complex-valued input sequences". *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*, 620–623, 2009.

[12] M. V. Wilkes *et al.* "The Preparation of Programs for an Electronic Digital Computer", *Addison-Wesley* 1951.

[13] F. J. Harris. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform". *Proceedings of the IEEE*, 66(1):51–83, 1978.

[14] A. H. Nuttall. "Some Windows with Very Good Sidelobe Behavior". *IEEE TRANSACTIONS ON ACOUSTIC, SPEECH, AND SIGNAL PROCESSING*, ASSP-29(1):84–91, 1981.

[15] P. Lynch. "The Dolph-Chebyshev Window: A Simple Optimal Filter". *Monthly Weather Review*, 125:655–660, 1997.

[16] M. D. Felder *et al.* "Efficient Dual-Tone Multifrequency Detection Using the Nonuniform Discrete Fourier Transform". *IEEE Signal Processing Letters*, 5(7):160–163, 1998.

[17] R. Beck *et al.* "Finite-Precision Goertzel Filters Used for Signal Tone Detection". *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: ANALOG AND DIGITAL SIGNAL PROCESSING*, 48(6):691–700, 2001.

[18] S. Fedorenko. "The Goertzel-Blahut Algorithm is Closely Related to the Fast Fourier Transform". *Problems of Redundancy in Information and Control Systems (RED), 2012 XIII International Symposium on*, 20–21, 2012.

[19] A.Arif *et al.* "Design Options for DTMF detection using Goertzel Algorithm on Reconfigurable Fabric". *Computer,Control & Communication (IC4), 2013 3rd International Conference on*, 1–5, 2013.